Trabajo Fin de Grado

Grado en Ingeniería de la Organización Industrial

Aplicación de la Ciencia de Datos para la predicción de la Demanda

Autor: Marta Pérez Ramis

Tutor: Antonio Plácido Moreno Beltrán

Departamento de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Aplicaciones de la Ciencia de Datos para la predicción de la Demanda

Autor:

Marta Pérez Ramis

Tutor:

Antonio Plácido Moreno Beltrán

Departamento de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Aplicación de la Ciencia de Datos para la predicción de la Demanda

Autor: Marta Pérez Ramis

Tutor: Antonio Plácido Moreno Beltrán

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

A mis padres y hermanos, por comprenderme y apoyarme en todo momento de mi vida. Y a Miguel, por haber sido un pilar fundamental en este proyecto y en mi vida. Sin ellos no hubiese llegado hasta aquí, ni sería quien soy hoy.

Y a Plácido, por guiarme y apoyarme en este trabajo.

Gracias

Sevilla, 2017

Resumen

En los últimos años ha cobrado gran importancia los conjuntos de datos a partir de la revolución de las tecnologías, que permiten el registro y control de la información, de forma fácil y sencilla, de los distintos parámetros en las disintas empresas e industrias, denominado históricos. En la actualidad existe un sinfín de herramientas y aplicaciones para establecer la predicción de la demanda

En este proyecto se analizará lod datos a partir de la aplicación de Ciencias de Datos mediante el empleo de la herramienta de programación Python en distintos algoritmos de regresión para identificar la tendencia del mercado mediante las distintas variables del modelo recogido en el histórico. La predicción de las distintas variables de conteo de la demanda se establecerá a través de la aplicación de los algoritmos de Machine Learning sobre los modelos de regresión.

Se evaluará, mediante distintas métricas de puntuación, la aproximación de los modelos de ajustes a los datos, cálculados a partir de los algoritmos de predicción, a los valores históricos.

Se compararán los resultados de los distintos algoritmos y variaciones para establecer cual es el modelo que mejor se ajusta a los datos.

Abstract

In recent years, data sets have been of great importance since the technological revolution, which allow records and control of information, so easy and simple, of the different parameters in the companies and the industries, called historical. Currently, there are a wealth of tools and applications for predicting demand.

In this project we analyzed the data from the application of Data Sciences by using the Python programming tool in different regression algorithms to identify the market trend in the different variables of the model collected in the historical. The prediction of the different counts variables of demand is established by the application of Machine Learning algorithms on the regression models.

It will be evaluated, through different scoring metrics, the approximation of the adjustments models to the data, calculations from the prediction algorithms, historical values.

We will compare the results of the algorithms and the variations to establish which the model that best fits the data is.

Índice

Índice de Tablas

Índice de Figuras

Notación

MSE	Error cuadrático medio – Mean squared error
RMSLE	Root mean squared logarithmic error
R^2	Coeficiente de determinación
CV	Cross Validation
KDD	Knowledge Discovery in Databases
SVM	Support Vectorial Machine
u	Perturbación de la recta de regresión
X	Variable explicativa o independiente
Y, y	Variable a predecir o dependiente
β	Coeficiente de las variables independientes
SCT	Suma de cuadrados total de los valores de y en proporción a los valores observados
SCR	Suma de cuadrados de la regresión del modelo especifico, valores predichos
SCE	Suma de cuadrados de los errores
S	Conjunto de entrenamiento para el kernel en SVM
y_e	Valor medio de la variable y observada y
y_{pred}	Valor estimado o predicho
k	Número de iteraciones y subconjuntos para CV
$\overline{y_D}$ $\overline{y_I}$	Valor medio del nodo predecesor I y D en árbol de decisión
$\bar{y}$	Valor promedio de la variable y
<	Menor o igual
>	Mayor o igual
y_m	Valor promedio de la variable y
x_m	Valor promedio de la variable x
ξ_i	Variables de holguras, reales y positivas

C	Constante grande definida por el usuario
ω, b	Coeficientes reales
ε	Diámetro del tubo formado por SVM

1 INTRODUCCIÓN

1.1 INTRODUCCIÓN A LA CIENCIA DE DATOS

En general, el mundo industrial y empresarial produce niveles altos de datos, incrementándose con el aumento de la tecnología en las industrias. Por este motivo, es importante ser capaz de trabajar con este conjunto de datos, relacionarlos y analizarlos para la toma de decisiones.

La Ciencia de datos es un campo de estudio amplio, difícil de definir cada uno de los conceptos, en el que se conectan métodos científicos, habilidades informáticas, procesos, sistemas y modelos matemáticos para extraer el conocimiento de grandes volúmenes de datos (*O´Neil, 2014*). A partir del conocimiento adquirido se analiza estadísticamente, en un campo experimentado, bajo el concepto de minería de datos (KDD) y aprendizaje automático, para conseguir un modelo predictivo.

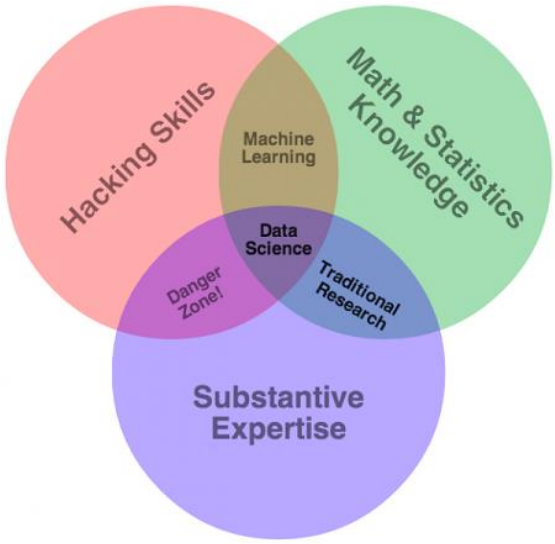

Figura 1. 1 Diagrama de ciencia de datos de Drew Conray (AllDataScience)

Las aplicaciones informáticas desarrollan el acceso, recopilación, limpieza y almacenamiento de datos, empleado herramientas de programación, base de datos y visualización. A su vez en las herramientas matemáticas se emplea técnicas de algebra lineal, cálculo, estadística descriptiva e inferencia estadística para la estimación de parámetros para el modelo de predicción.

Centrándose en la aplicación en la industria, la actualización de las nuevas tecnologías ha provocado un aumento de datos que necesitan ser procesados y modelados para obtener información precisa de la organización, como es la predicción en demanda y stocks.

El aprendizaje automático supervisado, para el tratamiento de análisis de datos en este trabajo, tratará el análisis predictivo. El análisis predictivo y automático es la metodología que permite facilitar el aprendizaje de programas informáticos, adaptándose a los cambios temporales según históricos para predecir el futuro, como es la demanda en las distintas industrias de producción y la selección de algoritmos de estimación y evaluación de los resultados obtenidos.

El término "ciencia de los datos" se estableció en 1997, en una conferencia realizada en la Universidad de Michigan por el estadístico Chien-Fu Wu. Se definió como datos estadísticos, ya que se formaban estudios y pruebas en base a ellos. Sin embargo, en 2001 el estadístico y científico de la computación Wiliam S. Cleveland estableció una disciplina independiente que trataba la estadística campo de aplicación de la ciencia de datos, y que dividía en seis áreas que abarca la ciencia de datos: investigaciones multidisciplinarias, modelos y métodos de datos, informática con datos, pedagogía, evaluación de herramientas y teoría (*AllDataScience, 2016*).

Entre 2002 y 2003, se comenzó a publicar información centrada en cuestiones como la descripción de los sistemas de datos, su publicación en Internet, aplicaciones, cuestiones jurídicas, y la Universidad de Columbia publico The Journal of Data Science, para proporcionar una plataforma para todos los trabajadores de datos para presentar sus puntos de vista e intercambiar ideas.

La evolución histórica de Ciencia de datos se muestra en la siguiente imagen de la representación del Impacto de los grandes datos en Analytics por Capgemini (*AllDataScience, 2016*):

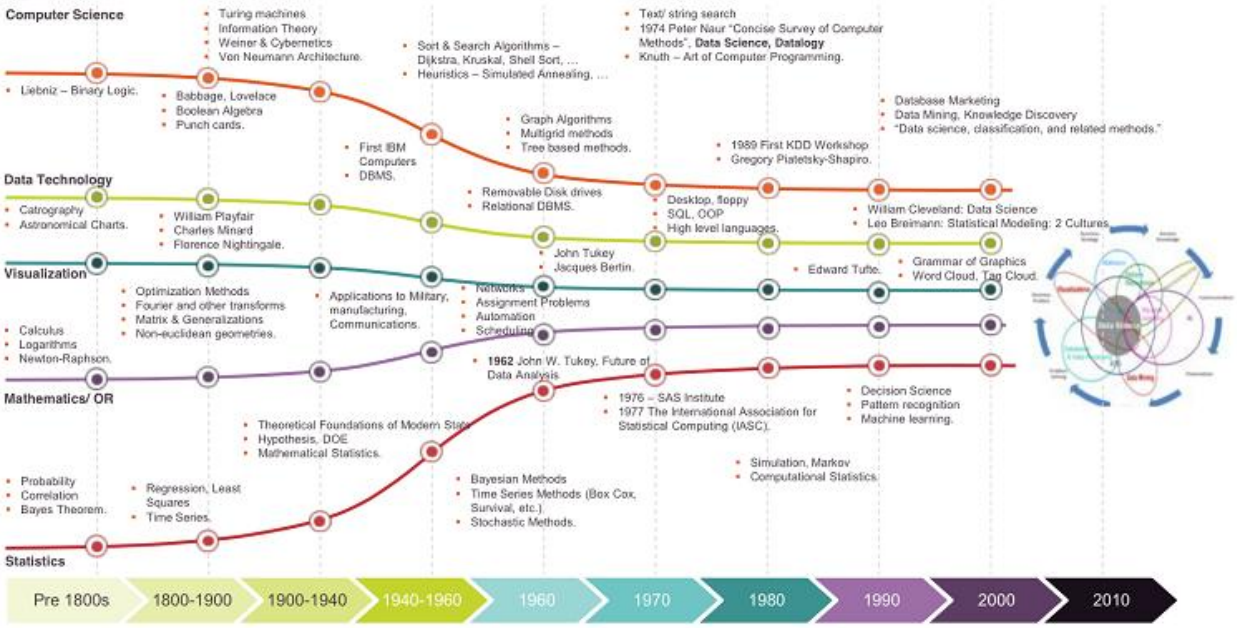

Figura 1. 2. Cronograma (AllDataScience)

Al igual que la definición de ciencia de datos, el perfil del científico de dato es una persona multidisciplinar capaz de analizar datos de tal forma que obtiene información, como es la tendencia del mercado, o predecir la demanda.

1.1.1 Alcance

El principal objetivo de este trabajo es la aplicación de la Ciencia de Datos para el análisis y predicción de la demanda a través del tratamiento de base de datos extraídos de la plataforma "***Kaggle***", que permite que las compañías publiquen sus datos en un concurso abierto para que expertos en Data Mining de todo el mundo puedan descargarse esos datos y propongan soluciones a los problemas de la compañía en cuestión.

El trabajo se centrará en la aplicación de algoritmos de aprendizaje automático centrados en distintos modelos de regresión para un buen ajuste del modelo predictivo basado en datos históricos de la demanda. Se realizará un análisis comparativo en base a los resultados obtenidos, así como su aplicación en la base de datos recogida por "***Kaggle***", para comparar la puntuación obtenida con otros modelos a escala mundial.

Para la realización del trabajo se empleará un equipo Macbook (13-inch, 2009 Late) de 8 GB de memoria RAM.

1.1.2 Objetivo General

La predicción de la demanda a corto y largo plazo es una operación muy importante a la hora de gestionar los recursos y stock de las distintas empresas e industrias mundiales. Es por ello que en este Trabajo de Fin de Grado "*Aplicación de Ciencia de Datos para la predicción de la demanda*" trata de aplicar la metodología de Ciencia de datos y Big Data en la predicción de la demanda a partir de grandes volúmenes de información mediante la metodología KDD de aprendizaje automático.

Se expone la aplicación del lenguaje de programación *Python*, como herramienta, para establecer los futuros valores de la demanda a partir de los distintos algoritmos basados y entrenados en el conjunto de datos recogidos.

No se busca crear una aplicación o algoritmo de predicción en sí, sino mostrar otra posible herramienta de predicción basada en el leguaje de programación *Python* mediante sencillos códigos aplicables a cualquier modelo e industria.

1.1.3 Objetivo Específico

El objetivo de este trabajo es emplear la herramienta de programación Python, sin el objetivo de alxanzar niveles software, con el fin predecir la demanda a partir de la aplicación de técnicas de aprendizaje automático mediante el análisis y preprocesamiento del conjunto de datos para la selección de variables significativas para la demanda del modelo.

A partir de la selección de variables significativas, búsqueda de patrones y la extracción de conocimientos del conjunto de datos se procederá a desarrollar distintos modelos de predicción basados en algoritmos de regresión como son Regresión lineal múltiple, Poisson, Árbol de decisión, Random forest y SVM.

La forma de evaluar los distintos modelos de regresión entrenados se aplicarán las distintas métricas de coeficiente de determinación, error cuadrático medio y RSMLE.

Para validar los resultados obtenidos por los métodos de regresión se aplicará validación cruzada con el objetivo de encontrar el algoritmo que mejor se ajuste a los datos, sin obtener modelos sobreajustados.

1.2 INTRODUCCIÓN A LA METODOLOGÍA

1.2.1 Metodología KDD (Knowledge Discovery in Databases)

Denominado “Descubrimiento de conocimientos en base de datos”, es la metodología de minería de datos empleada para descubrir conocimientos e información potencialmente útil dentro de los datos recogidos en una organización aplicando el campo de inteligencia artificial.

Combina las distintas técnicas de aprendizaje automático de forma iterativa, interactivo, reconocimiento de patrones (metodología APCD), estadística, base de datos y visualización de los conocimientos obtenidos de la base de datos con el fin de determinar las posibles relaciones o modelos entre los distintos datos.

Por este motivo es preciso evaluar la validez, utilidad y simplicidad de los patrones obtenidos en las técnicas de minería de datos, teniendo en cuenta el objetivo final de incorporar el conocimiento obtenido en el procesamiento de los datos históricos para tomar futuras decisiones que afectan a los resultados y partes interesadas.

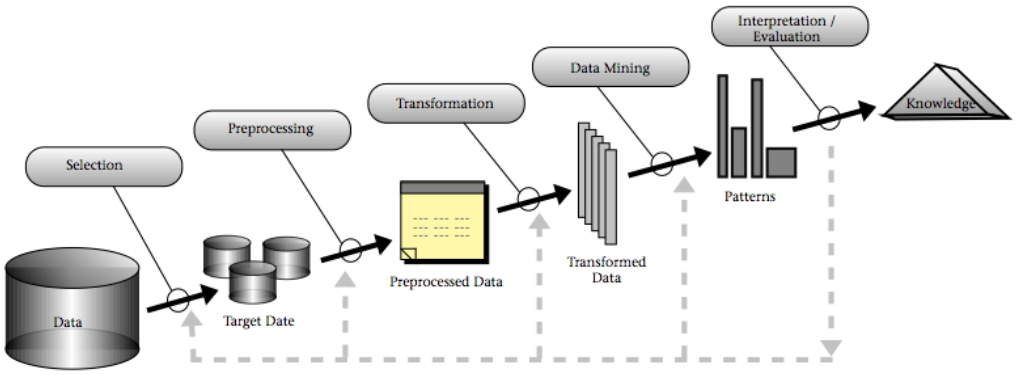

Figura 1. 3. Esquema de fases de la metodología KDD

Como se muestra en la figura anterior, la metodología KDD se divide en 5 fases:

1. Selección de datos. En esta etapa se determinan las fuentes de datos y el tipo de información a utilizar. Es la etapa donde los datos relevantes para el análisis son extraídos desde la o las fuentes de datos.
2. Preprocesamiento. Se basa en la preparación y limpieza de los datos extraídos desde las distintas fuentes de datos en una forma manejable, necesaria para las fases posteriores. En esta etapa se utilizan diversas estrategias para trabajar con datos faltantes o en blanco, datos inconsistentes o que están fuera de rango, obteniéndose al final una estructura de datos adecuada para su posterior transformación.

3. Transformación. Consiste en el tratamiento preliminar de los datos, transformación y generación de nuevas variables a partir de las ya existentes con una estructura de datos apropiada. Aquí se realizan operaciones de agregación o normalización, consolidando los datos de una forma necesaria para la fase siguiente.
4. Data Mining. Es la fase de modelar propiamente, en donde se aplican métodos inteligentes con el objetivo de extraer patrones previamente desconocidos, válidos, nuevos, potencialmente útiles y comprensibles y que están contenidos u "ocultos" en los datos.
5. Interpretación y Evaluación. Se identifican los patrones obtenidos y que son realmente interesantes, basándose en algunas medidas y se realiza una evaluación de los resultados obtenidos.

1.2.2 Python

Python es un lenguaje de programación de alto nivel, sencilla, expresiva y de precisión sintáctica, en el cual en pocas líneas es posible programar un algoritmo bastante complejo. Es un lenguaje de propósito general de un diseño claro y simple del código y sintaxis, permitiendo programas de fácil comprensión, tanto a pequeña como a gran escala.

Este lenguaje de programación fue creado a finales de los años 80 por Guido Van Rossum, un informático y matemático del centro de investigación en los Países Bajos como sucesor del lenguaje ABC.

Presenta varias formas de programación, como la programación orientada a objeto, la programación imperativa y la programación funcional.

La programación orientada a objeto (POO) es un lenguaje que permite realizar las tareas de distintas maneras, consiguiendo el mismo fin. Es decir, depende de la persona o programador para establecer el criterio a seguir para conseguir el *objeto* final.

Sin embargo, la programación imperativa es un lenguaje más cercano a la arquitectura de computadores, más estructurado, diseñado para ejecutar el código máquina sin variar el código ni depender de distintos programadores.

Por último la programación funcional es un lenguaje de programación donde las variables no tienen estado, no varían a lo largo del tiempo y son inmutables en el proceso de ejecución (*Summerfield, 2010*). No tienen funciones cíclicas, todo se procesa utilizando recursividad de las funciones.

Este lenguaje de código abierto que posee una biblioteca estándar amplia y comprensiva, e intérpretes disponibles para muchos sistemas operativos, permitiendo que el código Python

funcione en una amplia variedad de sistemas. Nos centramos en las bibliotecas especializadas en los datos con los que trabajemos a partir de la programación orientada a objeto.

Por otro lado, un programa en Python suele ocupar bastante menos espacio que lenguajes como Java o C++, realizando el trabajo de forma más rápida en menos código, permitiendo su ejecución sin compilación, si se desea. Además, Python permite ampliar o integrar su programación a otros lenguajes como herramienta auxiliar, ofreciendo mayor flexibilidad a la hora de trabajar con determinados datos.

Sus posibles desventajas es la velocidad de ejecución, que es más lenta en comparación con el resto de lenguajes de programación al no compilar a nivel de máquina para adquirir la característica de "portabilidad". Para minorar este hecho se trabaja la paralelización de las tareas.

Paquetes de Python que se emplearán en los siguientes apartados de este trabajo:

- **NumPy**: abreviatura de Numerical Python, es un paquete para la computación científica con códigos poderosos para los cálculos matemáticos, como son por ejemplo las combinaciones de números aleatorios, matrices, arrays multidimensionales etc. Permite que los datos numéricos sean mucho más eficiente de almacenar y manipular (*Peña, 2016*).

- **Pandas**: es un paquete muy importante y útil para trabajar con bases de datos completas o incompletas, con dos tipos de estructuras básicas rápidas, fáciles y expresivas (*Mckinney, 2012*):
  - Series: arrays unidimensionales formados por cualquier tipo de datos (números enteros, cadenas, flotantes, objetos, etc.) producidos en el paquete NumPy y pueden ser creados a partir de diccionarios
  - Dataframe: estructura de datos bidimensional que representan los datos en función de dos índices (columnas y filas), similar a las tablas de una hoja de cálculo con sus etiquetas de índice.

  Pandas permite combinar las funciones de alto rendimiento de NumPy con las hojas de cálculo y base de datos relacionadas, permitiendo agregar y seleccionar conjunto de datos. Otras de sus cualidades es que admite el empleo de datos faltantes (visualizados como NaN).

- **Matplotlib**: una de las bibliotecas de Python más utilizadas, con un código que permite cualquier tipo de representación o visualización de forma fácil de los datos a analizar, donde se puede manipular cualquier propiedad del gráfico, como por ejemplo, los ejes.

- **Scikit-learn**: paquete obligatorio para aplicar modelos predictivos en Python basados en los paquetes anteriormente explicados de NumPy y matplotlib y que permite implementar técnicas de regresión lineal, agrupaciones, árboles de decisión y los métodos precisos para establecer la métrica de los resultados

- **Seaborn**: librería que permite hacer más visuales las representaciones estadísticas en Python. Está basado en matplotlib, pandas y numpy, librerías anteriormente explicadas, permite añadir una paleta de colores a la representación, regresión lineal para diferentes tipos de variables independientes y dependientes.

- **Statsmodel**: módulo de Python que proporciona clases, funciones de modelos, pruebas y explotación de datos estadísticos. Esta librería emplea fórmulas de R junto con marcos de pandas, un ejemplo es la *Figura 1.4* aplicado sobre el futuro modelo que se desarrollará:

Dep. Variable:	casual	**No. Observations:**	10886
Model:	GLM	**Df Residuals:**	10867
Model Family:	Poisson	**Df Model:**	18
Link Function:	log	**Scale:**	1.0
Method:	IRLS	**Log-Likelihood:**	-1.2876e+05
Date:	Sat, 01 Jul 2017	**Deviance:**	2.1081e+05
Time:	12:24:00	**Pearson chi2:**	2.28e+05
No. Iterations:	6		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	1.2156	0.058	21.026	0.000	1.102	1.329
temp	0.0827	0.000	226.529	0.000	0.082	0.083
holiday	0.3209	0.014	23.509	0.000	0.294	0.348
humidity	-0.0181	0.000	-166.916	0.000	-0.018	-0.018
windspeed	0.0040	0.000	19.350	0.000	0.004	0.004
Tiempo_1	0.2209	0.088	2.513	0.012	0.049	0.393
Tiempo_2	0.3333	0.088	3.790	0.000	0.161	0.506
Tiempo_3	0.0439	0.088	0.498	0.619	-0.129	0.217
Tiempo_4	0.6175	0.320	1.927	0.054	-0.011	1.246
Estación_1	0.2565	0.016	15.587	0.000	0.224	0.289
Estación_2	0.4635	0.015	31.350	0.000	0.434	0.492
Estación_3	0.0812	0.016	5.167	0.000	0.050	0.112
Estación_4	0.4145	0.018	23.119	0.000	0.379	0.450
workingday	-0.3195	0.012	-25.716	0.000	-0.344	-0.295
month	0.0426	0.002	21.013	0.000	0.039	0.047
hour	0.0438	0.000	153.376	0.000	0.043	0.044
dayweek_Friday	0.1955	0.006	30.245	0.000	0.183	0.208
dayweek_Monday	0.0453	0.006	7.024	0.000	0.033	0.058
dayweek_Saturday	0.6274	0.017	36.626	0.000	0.594	0.661
dayweek_Sunday	0.5869	0.017	34.205	0.000	0.553	0.620
dayweek_Thursday	-0.0757	0.007	-11.090	0.000	-0.089	-0.062
dayweek_Tuesday	-0.0774	0.007	-11.208	0.000	-0.091	-0.064
dayweek_Wednesday	-0.0863	0.007	-12.598	0.000	-0.100	-0.073

Figura 1. 4 Ejemplo de aplicación de la librería Statsmodel

Para prepara el equipo en la elaboración de este trabajo en Python es necesario la instalación de Anaconda, un administrador de tareas gratuito disponible para los distintos sistemas operativos que incluye más de 100 de paquetes y de 720 códigos de fácil instalación.

Para descargarlo entrar en este enlace: https://www.continuum.io/downloads

file:///C:/Users/usuario/Desktop/Download%20Anaconda%20Now!%20_%20Continuum.html

Para el desarrollo del modelo se utiliza en este caso se uso Python 3.6 versión 64 bits:

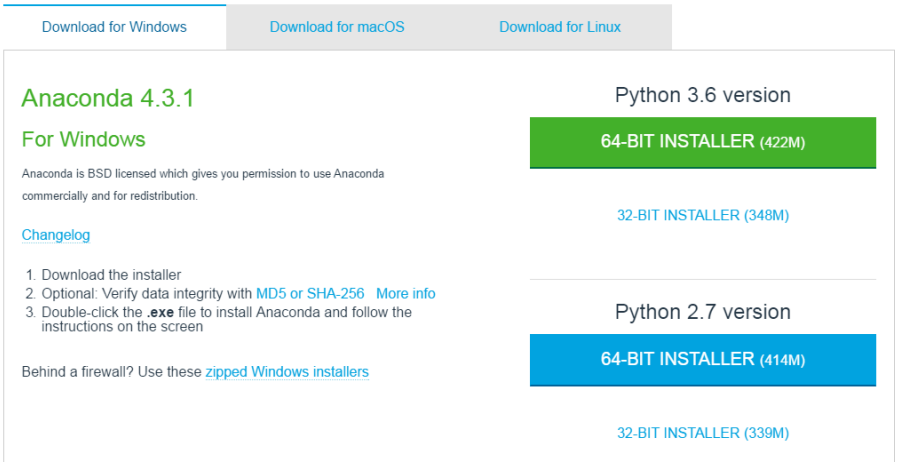

Figura 1.5 Opción de descarga para Anaconda

En Anaconda se empleará el entorno de desarrollo científico interactivo integrado en línea Jupyter Notebook para ejecutar el código a través de desarrollo web, donde se pueden almacenar, intercambiar y mostrar el conjunto de datos desde el navegador.

Para abrir Jupyter existen dos opciones:

- Desde la barra de herramientas de Windows, permite la búsqueda y apertura del entorno web inmediata.
- Desde la línea de comando del ordenador escribiendo Jupyter Notebook.

En ambos caso, se abrirá el navegador web que conectará con la instancia del programa. La *Figura 1.8* muestra la apertura del navegador:

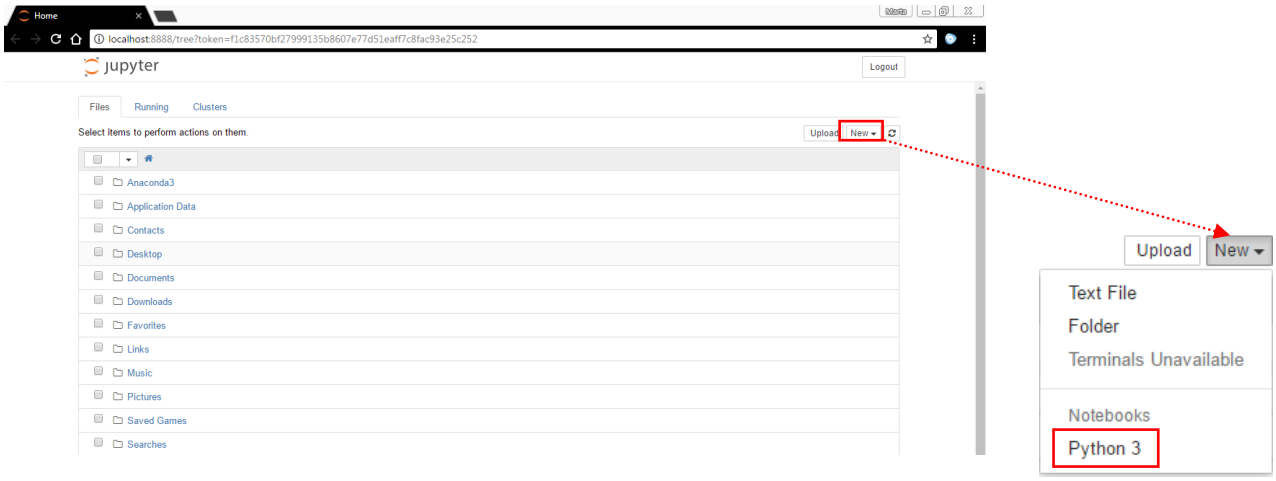

Figura 1. 6 Menú de trabajo Jupyter Notebook

Para crear el cuaderno de trabajo, se debe abrir a partir del recuadro rojo de la imagen anterior, un archivo nuevo, y marcar que queremos crear "*Notebook para Python*". A continuación se abrirá una página aparte con la siguiente imagen:

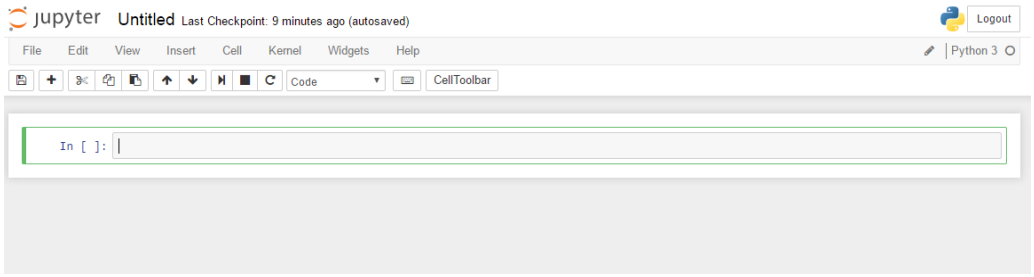

Figura 1. 7 Hoja de trabajo de Jupyter Notebook

A continuación se podrá comenzar con la programación de la base de datos mediante las líneas de código que permiten visualizar, a su vez, la programación y los resultados de forma dinámica. Permite añadir comentarios y título a cada hoja de trabajo, pero no el intercambio de información entre dos o más hojas de trabajo (*Babcock, 2016*).

2 MODELIZACIÓN ESTADÍSTICA Y APRENDIZAJE AUTOMÁTICO

2.1 DATOS DEL MODELO

La base de datos se obtiene de la plataforma online "***Kaggle***" que proporciona un repositorio para que las compañías publiquen sus datos en un concurso abierto donde los expertos en Data Mining de todo el mundo puedan descargar esos datos y propongan soluciones a los problemas de la compañía en cuestión mediante un sistema de bonificación para los tres mejores resultados.

Este trabajo analizará la base de datos de un sistema de intercambio o alquiler público de bicicletas automatizado a partir de una red pública de kioscos del programa Capital Bikeshare en Washington, DC.

La base de datos está compuesta por dos conjuntos:

- **train**: conjunto de entrenamiento compuesto por 19 primeros días de cada mes.
- **test:** conjunto de prueba/predicción formado a partir del día 20 de cada mes.
- **sampleSubmission**: conjunto de resultados de predicción, creado para cada algoritmo, y que facilita la comparación de los resultados obtenidos con el resto de predicciones realizada a través de la plataforma "***Kaggle***".

La base de datos está formada por la recopilación de este servicio durante dos años, de cada hora y día, y adicionalmente se recogen:

- **Fecha y hora:** (datetime) Año-Mes-Día
- **Estación del año** (season):
  - 1: Primavera.
  - 2: Verano.
  - 3: Otoño.

    - 4: Invierno.
- **Tiempo**, (weather) donde se establecen cuatro valores:
    - 1: Claro, algunas nubes, parcialmente nublado o nuboso.
    - 2: Niebla, nubes dispersas y nubes fragmentada con niebla.
    - 3: Lluvia ligera, tormenta con nubes dispersas.
    - 4: Lluvia fuerte, granizo y tormenta con niebla espesa.
- **Holiday**:(holiday) si el día es festivo.
- **Workingday**: (workingday) si es día laborable y no un fin de semana.
- **Temperatura** (temp) en grados Celsius.
- **Sensación térmica** (atemp) en grados Celsius.
- **Humedad**: (humidity) humedad relativa.
- **Velocidad del viento**: (windspeed).
- **Usuarios registrados en el sistema**: (registered).
- **Usuarios no registrados en el sistema**: (casual).
- **Número total de bicicletas alquiladas** (count).

2.2 APLICACIÓN DE LA PROGRAMACIÓN AL MODELO DE PREDICCIÓN

A partir del entorno interactivo de Jupyter Notebook trabajaremos con el fin de visualizar los resultados simultáneamente. En este trabajo se realizo una hoja de trabajo para cada tipo de algoritmo a aplicar al modelo, así como otra para la visualización de datos.

El primer paso es importar las librerías que son necesarias para el análisis de datos y que hemos descrito anteriormente. Se simplifica el nombre de las librerías, por lo que cada vez que empleemos alguna de las funciones o comandos, por ejemplo pandas, se especificará ***pd."Nombre de la función"***:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Figura 2. 1 Instalación de las librerías

Para conocer que datos posee el conjunto necesitamos abrir el archivo que contiene el conjunto de datos denominado "train" mediante el comando de Pandas 'pd.read_csv' y el directorio donde se encuentra el archivo:

```
dt=pd.read_csv('/Users/usuario/Downloads/train.csv')
dt.head()
```

Figura 2. 2 Apertura del conjunto de datos

Se define en un dataframe, mostrándose un pequeño conjunto de datos a partir de dt.head(), que permite mostrar las cinco primeras filas del data set, ya que la tabla completa es demasiado extensa:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Figura 2. 3 Muestra de la matriz de datos

Con la aplicación de códido dt_tail() se representarían las cinco ultimas filas del data set. Para averiguar de qué tamaño es el conjunto de datos se emplea el siguiente código, significa que tenemos 12 variables (columnas) y 10.886 instancias o registros (filas):

```
In [3]:
dt.shape
Out[3]:
(10886, 12)
```

Figura 2. 4 Tamaño de la muestra

Para el tratamiento de datos se debe de tener en cuenta el tipo de variables que presenta el dataframe, en este caso existen variables numéricas o cuantitativas y variables categóricas. El conjunto de variables categóricas lo forman las variables de datetime, season y weather, mientras que el resto son variables cuantitativas. Cabe destacar que las variables "*holiday*" y "*workingday*" son variables de clasificación, con valores 0 y 1, pero que no pueden ser tratadas como variables categóricas ni complementarias a falta del registro de que días son fines de semana, por lo que se le tratará como variable cuantitativa.

Para las variables cuantitativas realizamos un análisis estadístico a partir del comando 'dt.describe()', en el cual se muestra el número de muestras, el valore medio, la desviación típica, el valor mínimo y máximo, y los distintos percentiles para cada variable cuantitativa:

	holiday	workingday	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	0.028569	0.680875	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	0.166599	0.466159	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	0.000000	0.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	0.000000	0.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	0.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	0.000000	1.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	1.000000	1.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

Figura 2. 5 Valores estadísticos de las variables cuantitativas

Para conocer los posibles valores de las distintas variables en la muestra se emplea el siguiente comando aplicado de ejemplo en la variable de temperatura, *'dt.temp.value_counts()'*:

```
14.76    467
26.24    453
28.70    427
13.94    413
18.86    406
22.14    403
```

Figura 2. 6 Ejemplo de la función de conteo de valores

La columna de la izquierda muestra algunos de los distintos valores que puede tomar la variable *"temp"*, y la columna de la derecha muestra el número de veces que aparece dicha temperatura en el dataset. Para conocer el número de valores distintos que puede tener dicha variable se emplea el siguiente código:

```
len(dt.temp.value_counts())
```

```
Out[14]:
```

```
49
```

Figura 2. 7 Ejemplo de la función tamaño

Centrándonos en las variables cuantitativas a predecir en el algoritmo de aprendizaje automático, son las variables registered, casual y count. Esta última está formada por la suma de los usuarios registrados (registered) y casuales (casual) por fecha y hora del registro. Para conocer el número exacto de usuarios de cada variable, se desarrolla los siguientes códigos:

```
dt.registered.sum()
```

```
Out[5]:
```

```
1693341
```

```
dt.casual.sum()
```

```
Out[6]:
```

```
392135
```

```
#Opción 1:
dt['count'].sum()
```

```
Out[7]:
```

```
2085476
```

```
#Opción 2:
total_us= dt.registered.sum()+dt.casual.sum()
total_us
```

```
Out[8]:
```

```
2085476
```

Figura 2. 8 Número de los distintos tipos de usuarios

Como se observa, existen más usuarios registrados (1.693.341 usuarios) que casuales (392.135 usuarios), este dato deberá tenerse en cuenta a la hora de valorar los distintos modelos de predicción por separado.

Las variables categóricas necesitan un tratamiento previo al análisis, en cual se convierten en variables dummy o cualitativas tomando valores binarios por cada columna. Es decir, por cada valor que toma la variable categórica se creará una columna del valor y tomara valor 1 si la instancia presenta ese valor de la variable categórica y 0 en caso contrario. Se muestra un ejemplo con la variable weather en la siguiente imagen:

```
dum_tiempo=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_tiempo.head()
```

Out[5]:

	Tiempo_1	Tiempo_2	Tiempo_3	Tiempo_4
0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0

Figura 2. 9 Ejemplo de variable categórica

Esto se aplica de la misma forma a la variable datetime y season. Todas estas son agregadas (*Babcock, 2016*) al conjunto de datos para un mejor análisis definiéndose de la siguiente forma:

```
column_name=dt.columns.values.tolist()
dt2=dt[column_name].join(dum_tiempo)
column_name1=dt2.columns.values.tolist()
dto=dt2[column_name1].join(dum_estacion)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()
```

Figura 2. 10 Ejemplo de cómo agregar las variables dummy al conjunto de datos

Implica un aumento del tamaño de data set, con respecto al número de variables en:

```
dtt.shape
```

```
Out[17]:
```

```
(10886, 31)
```

Figura 2. 11 Tamaño del conjunto de datos

Para simplificar el conjunto de datos se elimina las variables convertidas en dummy, puesto que no aportan nada al modelo, en conclusión, las variables del conjunto de datos se definen en 28.

```
del dtt['season']
del dtt['weather']
del dtt['day']
```

Figura 2. 12 Eliminación de variables del conjunto de datos

Se procede a la visualización de los distintos datos, con el fin de obtener mayor información del dataset. Representamos la variable "*humidity*":

```
sns.distplot(dtt.humidity.dropna())
sns.plt.show()
```

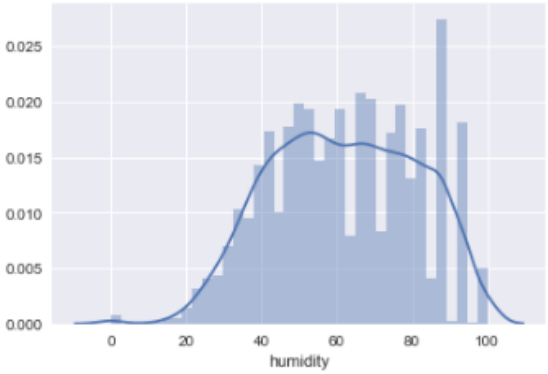

Figura 2. 13 Valor de la variable humedad

El siguiente gráfico hace referencia al número de usuarios totales, "*count*", por cada tipo de weather, donde cabe destacar el alto nivel de usuarios en primavera y verano:

```
sns.countplot(x="weather", data=dtt, palette="Greens_d")
sns.plt.show()
```

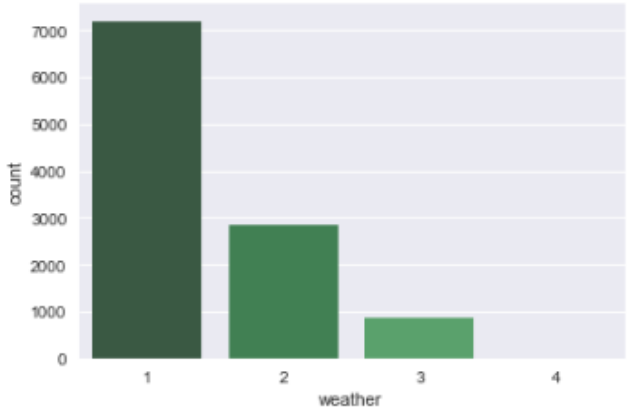

Figura 2. 14 Nivel de usuarios para cada estado del tiempo

A continuación se representa un gráfico que relaciona el número de usuarios totales, "*count*", a cada hora del día:

```
dtt.plot(kind='scatter',x='hour',y='count')
plt.show()
```

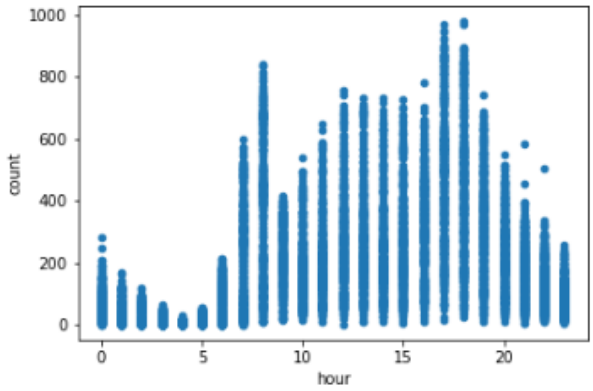

Figura 2. 15 Número de usuarios totales para cada hora

Para comparar estos valores, se estimara el número de usuarios registrados y casuales con el fin de diferenciar los periodos de tiempo en el que cada tipo de usuario influye:

```
dtt.plot(kind='scatter',x='hour',y='registered')
plt.show()
```

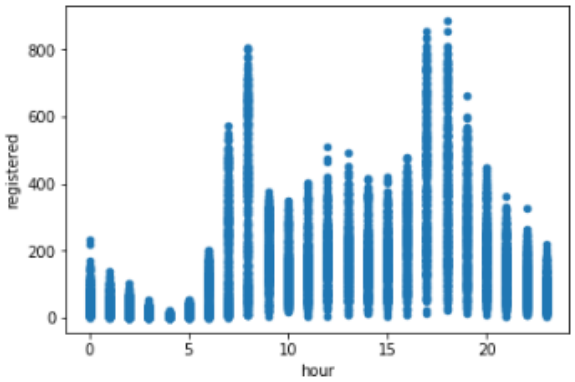

Figura 2. 16 Número de usuarios registrados para cada hora

```
dtt.plot(kind='scatter',x='hour',y='casual')
plt.show()
```

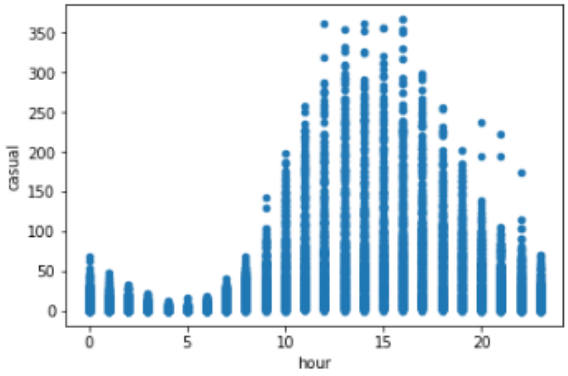

Figura 2. 17 Número de usuarios casuales para cada hora

Se observa para los usuarios casuales, *Figura 2.17*, las horas más demandadas son entre las 14 y 19 horas, donde ronda aproximadamente los 300 usuarios, mientras que en los usuarios registrados se mantiene más estable a lo largo del día, siendo aproximadamente los 500 usuarios y presentando picos de hasta 800 usuarios entre las 7 y 8 de la mañana y de las 16 y las 18.

Para seguir comparando las diferencias entre los dos tipos de usuarios, observamos el número de cada tipo de usuario en cada mes del año:

```
dtt.plot(kind='scatter',x='month',y='registered')
plt.show()
```

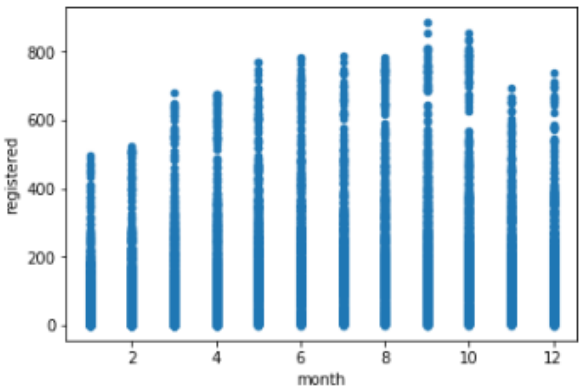

Figura 2. 19 Número de usuarios registrados para cada mes

```
dtt.plot(kind='scatter',x='month',y='casual')
plt.show()
```

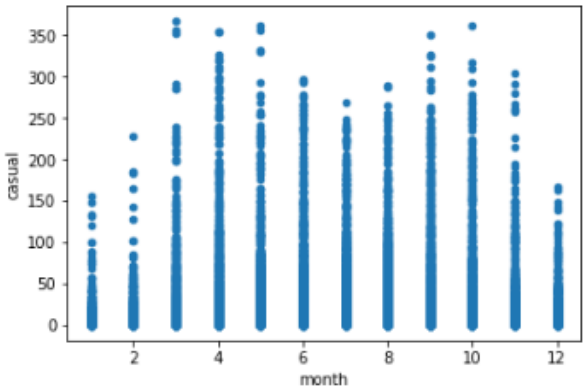

Figura 2. 18 Número de usuarios casuales para cada hora

Observamos que los niveles de usuarios registrados son siempre mayores ya que es mayor el nivel de usuarios registrados en el dataset, y aumentan en los meses de septiembre y octubre, mientras que los usuarios casuales los valores más altos se dan en los meses de marzo, abril, septiembre y octubre, pero en ninguno alcanza el nivel de los usuarios registrados.

Buscamos el rango de las variables independientes de valores cuantitativos con el fin de observar algún patrón (*Phuong, 2017*):

```
dtt.boxplot(column="temp")
plt.show()
```

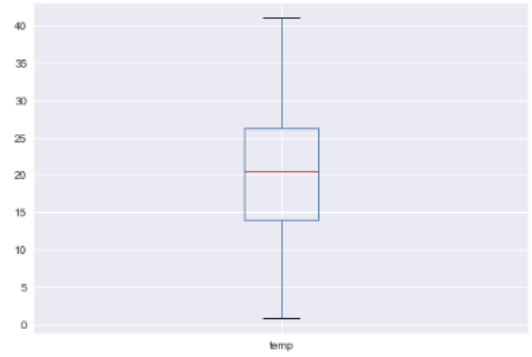

Figura 2. 20 Diagrama de cajas y bigotes de la variable temperatura

```
dtt.boxplot(column="atemp")
plt.show()
```

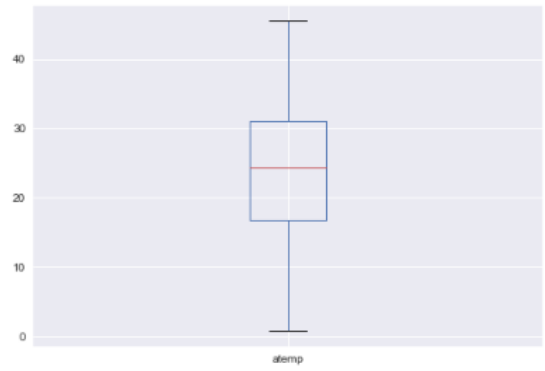

Figura 2. 21 Diagrama de cajas y bigotes de la variable sensación térmica

La temperatura y la sensación térmica no muestran los mismos valores, siendo 20ºC la temperatura media y 24ºC el valor medio de sensación térmica.

```
dtt.boxplot(column="hour")
plt.show()
```

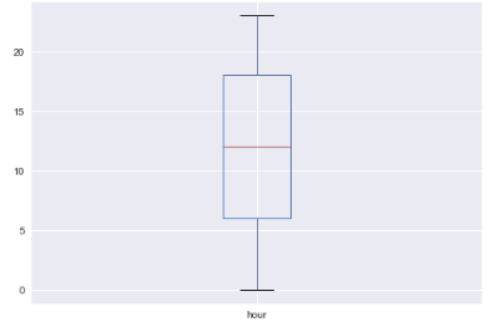

Figura 2. 23 Diagrama de cajas y bigotes de la variable hora

```
dtt.boxplot(column="humidity")
plt.show()
```

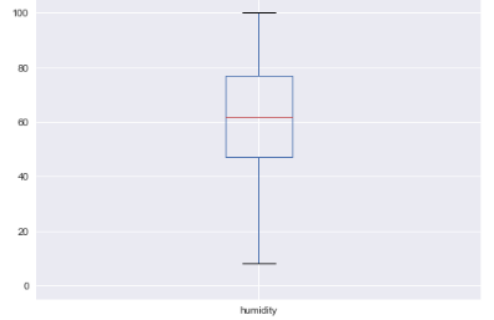

Figura 2. 22 Diagrama de cajas y bigotes de la variable humedad

La hora que más aparece en el registro es a las 12 horas, la humedad se mantiene entre valores estándares y la velocidad del aire no supera los 30 km/h.

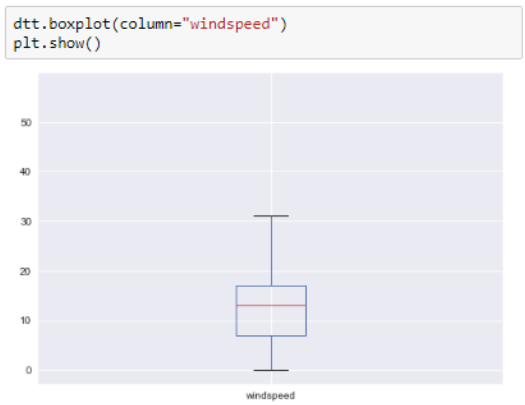

Figura 2. 24 Diagrama de cajas y bigotes de la variable velocidad del viento

Por último, se establece la relación entre las distintas variables del modelo por la función de correlación entre las variables a partir del código:

```
corrvariables = dtt[["temp","atemp","holiday",
                     "day","month","workingday",
                     "casual","registered","humidity"
                     "windspeed","count"]].corr()
mask = np.array(corrvariables)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sns.heatmap(corrvariables, mask=mask,vmax=.8,
            square=True,annot=True)
plt.show()
```

Figura 2. 25 Código para correlacionar variables

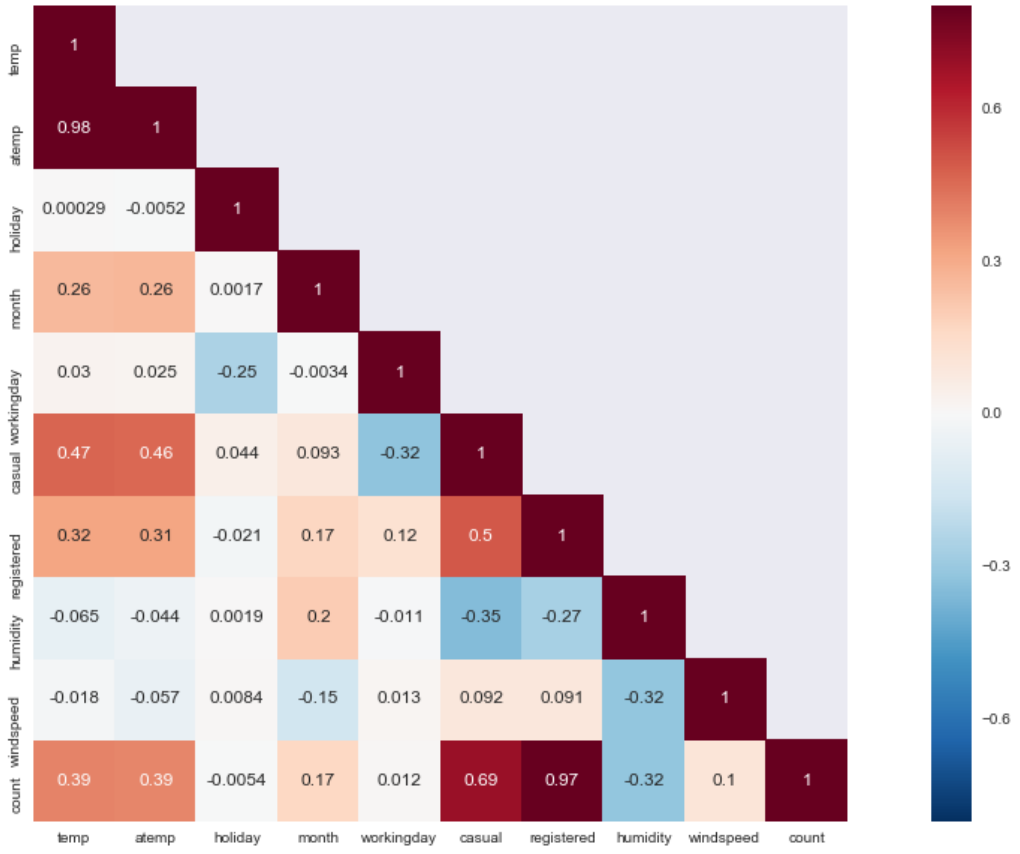

Figura 2. 26 Correlación entre las distintas variables

Analizando la ***Figura 2.26*** se aprecia relación entre las variables de temperatura y sensación de calor con los usuarios registrados y casuales, siendo mayor en este ultimo tipo de usuarios, ya que a mejor temperatura y tiempo los usuarios establecen una mayor demanda de bicicletas.

La relación que se muestra entre los distintos tipos de usuarios con el número de usuarios totales es lógica puesto que está formado por la suma de dichos valores.

2.3 DESARROLLO DE LOS ALGORITMOS A APLICAR

En este apartado se desarrolla los métodos estadísticos en lo que se trata de explicar las variables objetivos a partir de las variables independientes y la combinación lineal de estas:

1. **Regresión lineal múltiple**: metodología basada en la minimización de la suma de mínimos cuadrados del conjunto de datos X (variables explicativas o independientes) para devolver una aproximación lineal del valor Y (variable predicha o dependiente), creando una función de hiperplano destinada a comprobar hipótesis y relaciones causales. Es decir, es una función lineal con j regresores, tantos como X existan en el modelo, para calcular el valor ajustado viene dado por la siguiente función:

$$y_i = \beta_1 + \beta_2 x_{2i} + \beta_3 x_{3i} + \cdots + \beta_j x_{ji} + u \quad (1)$$

Por lo que en el modelo existen j ecuaciones y j incognitas, en concreto, para el modelo de regresión lineal múltiple es necesario exponer un conjunto de hipótesis estadísticos:

- La relación entre las distintas variables es lineal.
- Los valores de las variables independientes son fija en las repetidas muestras.
- Los parámetros $\beta_1, \beta_2, \beta_3, \dots, \beta_j$ son constantes
- La media de la perturbación es cero ($u = 0$)

- Consiste en expresar la esperanza condicionada de la variable dependiente como combinación lineal de las variables independientes:

$$E(Y|X = x_i) = u_i = z_i^t \beta \quad (2)$$

Donde z_i es un vector de diseño que representa la función apropiada de las variables dependientes.

- Se debe eliminar las variables independientes relacionadas, que producen un fenómeno denominado colinealidad, ya que representa el error estándar de la predicción a la hora de ser usadas para predecir una variable dependiente se produce un fenómeno de redundancia. Las variables deben ser ordinales o dummy.

Según se muestra en [3], para estimar el valor de la regresión lineal múltiple a partir de datos simulados, y suponiendo que son lineales, sin valor residual, se define valor estimado:

$$y_{pred} = \alpha + \beta_i x_i \quad (3)$$

Definidos a partir de unos datos el valor real del modelo como:

$$y = \alpha + \beta_i x_i \tag{4}$$

Por último, se establece el valor medio de los resultados y_{pred} . Establecen tres tipos de rectas con lo que se calculan los coeficientes α y β a partir de la suma de cuadrados total:

$$SCT = \sum(y - y_e)^2 \tag{5}$$

El error se expresa como la diferencia entre el valor medio de la variable dependiente y el valor estimado:

$$SCE = \sum(y_e - y_{pred})^2 \tag{6}$$

El término aleatorio es el que crea la diferencia de la suma de cuadrados:

$$SCR = \sum(y - y_{pred})^2 \tag{7}$$

Lo que significa la igualdad de: $SCT = SCR + SCE$ (8)

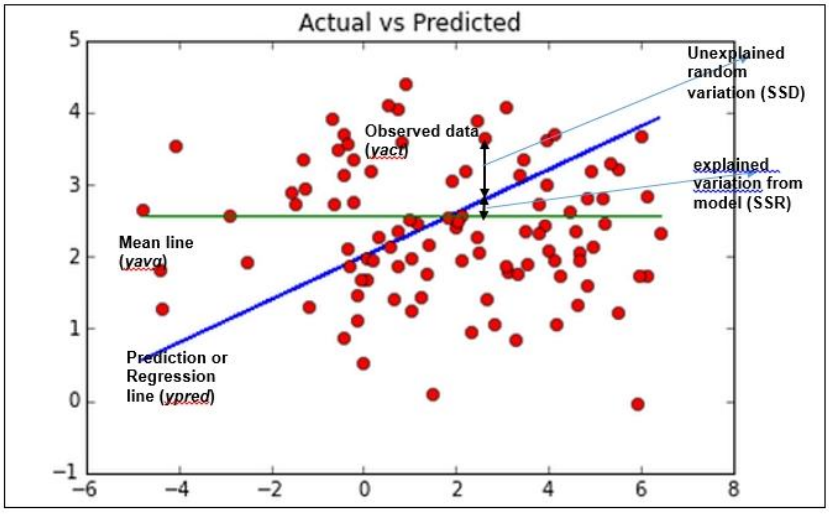

Figura 2. 27 Representación de las distintas rectas, (Kumar, 2016)

Cuanto mayor sea el error total (SCE) mejor será el modelo de predicción, puesto que se aproxima al valor medio. Esta relación se muestra en el coeficiente de determinación como indicador de bondad del modelo analizado, siendo un criterio de calidad:

$$0 \leq R^2 = \frac{SSR}{SST} \leq 1 \quad (9)$$

Cuanto más cerca este del 1 el coeficiente de correlación, mejor será el modelo aunque no signifique que el modelo sea eficiente. Para comprobar que es eficiente tendríamos que analizar otros factores del modelo. Con esto se obtiene las estimaciones de α y β como:

$$\beta = \sum \frac{(x_i - x_m)(y_i - y_m)}{(x_i - x_m)^2} \quad (10)$$

$$\alpha = y_m - \beta * x_m \quad (11)$$

Si $\beta = 0$ no existe relación entre la variable dependiente y la variable independiente, sino existe correlacion suficiente entre ambas variables.

2. Poisson: se emplea en fenómenos en los que la variable dependiente es de conteo, tomando valores desde 0 hasta un límite superior que no tiene por qué estar definido, y se ajustan a una distribución de Poisson. La variables independientes pueden ser continuas o categóricas, definiéndose los parámetros de la distribución de Poisson como:

$$Y \sim P(\mu) \quad (12)$$

$$P(Y = y) = \frac{e^{\mu} \mu^{y}}{y!} \quad (12)$$

$$E(Y) = Var(Y) = \mu \quad (13)$$

Al poseer la propiedad de igualdad entre la media y la varianza, permite solucionar los problemas de heterocedasticidad, perturbaciones de la varianza al no ser constante a lo largo de las observaciones para un conjunto de datos independientes, cuya suma es la suma de modelos que se rigen bajo la distribución de Poisson $Y \sim P(\mu_i)$, creando un modelo lineal con una matriz z de diseño:

$$\mu_i = z_i^t \beta \quad (14)$$

Esta matriz puede tomar cualquier valor real mientras que el valor de cada media debe ser no negativo, por ello se utiliza la función de enlace el logaritmo en función canónica:

$$\log(\mu_i) = z_i^t \beta \tag{15}$$

Al aplicar el Modelo Lineal General se tiene que:

$$E(Y_i|x_i) = e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p} \tag{16}$$

Permaneciendo la función de velosimitud para n observaciones independientes de Poisson como el conjunto de probabilidades individuales:

$$logL(\beta) = \sum_{i=1}^{n}\{y_i \log(\mu_i) - \mu_i\} \tag{17}$$

Para su evaluación también se aplicara la diferencia que define el coeficiente de determinación explicado en el apartado anterior.

3. SVM: Máquinas de soportes vectoriales, es una estructura de aprendizaje automático formada por un conjunto de métodos estadísticos de aprendizaje supervisado que se utiliza para la clasificación y regresión creado en los años setenta por Vapnik, el método general en los noventa, y basado en el principio de minimización estructural del riesgo (*Carmona, 2014*). La librería SVM implica que el conjunto de datos de aprendizaje sean linealmente separables aplicado en el kernel en dos pasos, primero la formación del conjunto de datos para obtener un modelo, y segundo la utilización del modelo para predecir la información del conjunto de prueba.

Parte de un conjunto de n observaciones, cada una descrita por un vector de características d-dimensionales, y que vienen acompañadas por el valor de una variable de respuesta y que se desea predecir a partir de un conjunto de entrenamiento (*Smola, 2004*):

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \tag{18}$$

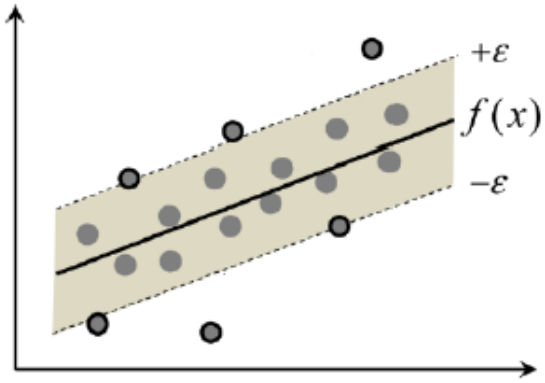

Figura 2. 28 Representación de la función SVM

Para medir los errores (ε) empleamos una función de pérdida, que debe ser nula en las observaciones que se encuentren en el interior del tubo de la imagen, y estrictamente positivos para los que se encuentren en el exterior del tubo. Se busca transformar el espacio de entrada en otra dimensión superior en el que el problema puede ser resuelto mediante un hiperplano óptimo, separador del espacio, cuya ecuación es (*Gammerman, 2015*):

$$\begin{aligned} \text{mín} \quad & \tfrac{1}{2}\omega^t\omega + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.a.} \quad & y_i - (\omega^t x_i + b) \leq \varepsilon + \xi_i \qquad i = 1, \ldots, n \\ & \omega^t x_i + b - y_i \leq \varepsilon + \xi_i^* \qquad i = 1, \ldots, n \\ & \omega \in \mathbb{R}^d, b \in \mathbb{R}, \xi_i, \xi_i^* \geq 0 \qquad i = 1, \ldots, n \end{aligned} \tag{19}$$

Al maximizar el margen se minimiza el error de generalizado del regresor, definiendo el hiperplano separador para el caso linealmente independiente como:

$$|g(x_n)| = t_n g(x_n) \tag{20}$$

Una vez entrenado la máquina de soporte vectorial, se aplica una fase de validación cruzada con el fin de analizar el sobreajuste del conjunto de entrenado.

Para la evaluación de este algoritmo se suele emplear el error cuadrático medio, el coeficiente de determinación y RSMLE.

Se caracteriza por:

- Eficaz para datos de grandes dimensiones.
- Eficaz cuando el número de instancias es menos que el de variables, con la desventaja de que el rendimiento en bajo.

4. Árbol de decisión: modelo predictivo que emplea técnicas de análisis discriminantes, la desviación, para poder predecir los valores a partir de una función establecida por el conjunto de variables predictoras.

La metodología de resolución es mediante un grafo etiquetado e implantado en todos sus nodos, basándose en la división de nodos divididos a su vez en subnodos a partir de ciertos criterios definidos por los conceptos adquiridos basados en los históricos, que reduzcan al máximo la varianza. Están constituidos de las siguientes partes:

- Nodos interiores (atributos)
- Arcos (posibles valores del nodo origen)
- Hojas (valor de la regresión)

El procedimiento que lleva a cabo la preparación del árbol de decisión es:

- Sobreajuste: a partir de los ejemplos de entrenamiento del modelo, si existe una hipótesis que se ajusta peor pero actúa mejor sobre la distribución completa de las instancias o atributos que presentan una aparente regularidad pero no son relevantes en realidad.
- Ruido: errores presentes en la base de datos a tratar o causado por sobreajuste del modelo.
- Poda: evita ruidos y errores de sobreajustes a partir de evitar el desarrollo del árbol antes de que se ajuste perfectamente, o transformar a condiciones de regla directamente en la aplicación del árbol, ambas son poda a posteriori.

Para cada atributo se calcula la desviación esperada como criterio de selección en la división de cada etapa, que es la suma de cuadrados del error (SCE), siendo árbol de regresión:

$$SCE = \sum_{\boldsymbol{i=1}} (y_i - \bar{y})^2 \tag{21}$$

Para cada nodo se realiza este paso, de forma que se tendrá dos nuevos nodos al continuar por cada "rama" de forma:

$$SCE = \sum_{i \in t_I} (y_i - \overline{y_I})^2 + \sum_{i \in t_D} (y_i - \overline{y_D})^2 + n_I (\overline{y_I} - \bar{y})^2 + n_D (\overline{y_D} - \bar{y})^2 = SCE_W + SCE_B \tag{22}$$

Al realizar la división, la nueva suma de cuadrados será:

$$SCE_W = \sum_{i \in t_I} (y_i - \overline{y_I})^2 + \sum_{i \in t_D} (y_i - \overline{y_D})^2 \tag{23}$$

El criterio de selección para cada nodo consiste en seleccionar aquel par que produce el menor valor de $\boldsymbol{SCE_W}$ para ese nodo, y equivalente se busca el mayor valor de $\boldsymbol{SCE_D}$.

Para la búsqueda del mejor árbol de decisión se examina el árbol más simple y corto, con el menor número de atributos es capaz de predecir la decisión. El nodo terminal devuelve un valor promedio de la salida, por lo que se validará el modelo mediante la validación cruzada evaluando el error cuadrático medio.

5. **Random Forest Regression**: es un conjunto de árboles de regresión aleatorios, de amplio uso, ya que presenta un rendimiento especialmente bueno para datos de alta dimensionabilidad. Se ajusta a un número de árboles de decisión en varias submuestras del conjunto de datos, para utilizar un valor promedio que mejora la precisión de la predicción y controla el ajuste excesivo, consiguiendo así mejores resultados que con el árbol de decisión. Cabe destacar, que el tamaño de la muestra es el mismo que el tamaño del dataset de entrada.

La metodología para el modelo de predicción es la siguiente:

1. Decidir qué variables van a ser los datos de entrada al bosque para obtener el valor de la variable a predecir, que depende de la definición de los datos de entrada.
2. Aleatoriamente se crea el conjunto de entrenamiento a partir del conjunto de datos de entrada.
3. Aplicación de "*cross validation*" (Validación cruzada) al conjunto de entrenamiento con el fin de eliminar el sobreajuste del modelo.
4. En cada división del árbol en cada nodo, la búsqueda de la mejora variable para dividir los datos no se realiza sobre todas las variables sino sobre el subconjunto de las mismas, que se realiza de forma aleatoria.
5. Los anteriores procesos se repiten iterativamente hasta llegar tener un conjunto de árboles de decisión entrenados sobre diferentes datos y atributos del conjunto.
6. Una vez entrenado el algoritmo, la evaluación de la predicción se realiza con el conjunto de árboles, teniendo en cuenta el valor promedio de los resultados.

Por otro lado, la técnica de random forest puede ser paralelizado eficazmente puesto que cada árbol puede construirse de manera independiente.

Además, el comportamiento del algoritmo es en función del número de árboles que incorpora en el algoritmo entrenado. Con lo cual un incremento del número de árboles permite una mayor diversidad de los mismos consiguiéndose una reducción del error, aunque cabe destacar que a partir de un determinado número de árboles la mejora se estanca.

Cuanto mayor es la correlación entre dos árboles cualesquiera, menor será el error producido por el algoritmo.

2.3.1 Validación cruzada

Herramienta de análisis que permite evaluar análisis estadísticos en base a una partición del conjunto de datos de entrenamiento en ***k*** partes para crear una estructura y modelos de minería de datos que permitan determinar la validez del modelo a partir de ***k*** aprendizajes, sin producirse sobreajustes.

Este algoritmo está compuesto de dos fases, entrenamiento y generación de resultados (*Pérez, 2015*).

En la fase de entrenamiento se selecciona el conjunto de datos a entrenar y se indica cual es las variables a predecir o probar. Además se especifica el número de plegamientos en los que se desea crear particiones de los datos del modelo.

El método utilizado es *k-fold* se basa en un conjunto divididos en k subconjuntos, de manera que para cada subconjunto se aplica el método *hold-out* ***k*** veces, utilizando cada vez un subconjunto distinto para validar el modelo empleado con ***k-1*** subconjuntos.

La ventaja de aplicar el método *k-fold* es que todos los datos son utilizados para entrenar y validar, por lo que se obtiene resultados más representativos, realizado n veces de forma aleatoria por el método de *hold-out*, no garantizando que los casos de entrenamiento y validación no se repitan.

Esta herramienta se utilizará en este proyecto para evaluar la puntuación obtenida en los distintos modelos de ajustes de cada algoritmo, validando si se ha producido sobreajuste alguno en los modelos de entrenamiento.

2.3.2 Scoring

1. **Coeficiente de terminación (R^2):** medida de bondad de ajuste que muestra la varianza de la variable dependiente del valor de regresión. Es cuánto las variables independientes explican la variable dependiente, indica el porcentaje de la varianza de la variable dependiente explicado por el conjunto de variables independientes. Cuanto mayor sea la R-cuadrado más explicativo y mejor es el modelo causal.

   El coeficiente de determinación se basa en la siguiente descomposición (*Kumar, 2016*):

   $$SCT = SCE + SCR \tag{26}$$

   $$R^2 = \frac{SCE}{SCT} \tag{27}$$

   Donde SCT es la suma de cuadrados totales, SCE es la suma de cuadrados y SCR es la suma de cuadrados residual, con lo que se define la función del coeficiente de determinación entre los valores:

   $$0 \leq R^2 \leq 1 \tag{28}$$

   Si el coeficiente de determinación es 0, significa que la varianza es cero es cuando el ajuste del modelo predicho a partir de los valores históricos es perfecto. Lo contrario es el valor 1.

   Para valores pequeños del coeficiente de determinación implica una amplia varianza de la perturbación, por lo que los coeficientes incógnita del modelo no se pueden predecir a partir de dicho modelo.

2. **ECM error cuadrático medio:** herramienta empleada como estimador para medir el valor promedio de los errores al cuadrado, entre la diferencia del estimador y lo que se estima. Midiendo la calidad del estimador insesgado o predictor mediante la siguiente expresión (Gonzalez, 1992):

   $$MSE = \frac{1}{n}\sum_{i=1}^{n}(\overline{Y_i} - Y_i)^2 = E[(T-\theta)^2] = E[(T(X_1, \dots, X_n) - \theta)^2] \tag{29}$$

- $\overline{Y_i}$ es el vector predictor de n predicciones.
- Y_i es el vector real.
- La diferencia mide el error que se comete al estimar θ mediante T(X).
- Se eleva al cuadrado para evitar valores negativos.

3. **ERL** o **RMSLE:** este modelo de regresión es una alternativa cuando el modelo lineal no logra un coeficiente de determinación apropiado. El caso de RMSLE se toma el registro de las predicciones y los valores reales, cambiando la varianza que está midiendo.

   Se expresa de la siguiente manera:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2} \quad (30)$$

- a_i: valores reales, históricos.
- p_i: valores predictivos.
- n: número total de observaciones.
- Log(x): número natural

3 IMPLEMENTACIÓN DE LOS MODELOS

A continuación se procede a la resolución del modelo aplicando los distintos algoritmos del apartado anterior. Previamente se realiza un pretratamiento del modelo para que sea más fácil de trabajar, desarrollado en el apartado 2 en ***APLICACIÓN DE LA PROGRAMACIÓN AL MODELO DE PREDICCIÓN.***

Teniendo el modelo preparado se aplica una partición del conjunto de entrenamiento que sigue la metodología de la siguiente forma:

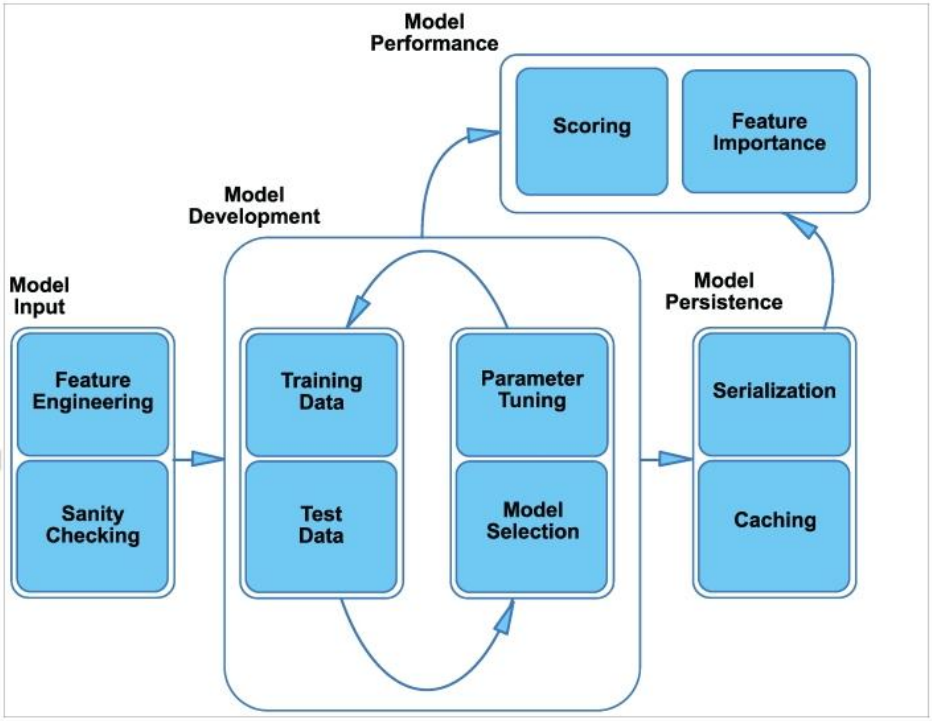

Figura 3. 1 Esquema de validación cruzada de un conjunto de datos de entrenamiento (Kumar, 2016)

Esta metodología se desarrolla en el apartado anterior en Validación cruzada, por lo que se aplica una división del modelo de entrenamiento del 80-20 entre el submodelo que se usará de entrenamiento y el que evaluará la predicción (*Kumar, 2016*).

Mediante el subconjunto del modelo de entrenamiento se entrenará el modelo de predicción para aplicárselo a la partición del subconjunto de prueba. El modelo de entrenamiento que se obtiene será aplicado al conjunto inicial de prueba para una verdadera evaluación del ajuste del modelo de predicción:

```
from sklearn.cross_validation import train_test_split
train, test = train_test_split(dtt, test_size = 0.2)
```

Figura 3. 2 Código de aplicación CV

A la hora de aplicar el algoritmo se empleará distintas variables a predecir, que son:

- Usuarios Casuales
- Usuarios Registrados
- Usuarios Totales
- Usuarios Casuales + Usuarios Registrados

Por lo que se define distintos conjuntos de entrenamiento y testing del conjunto de entrenamiento inicial, dependiendo de la variable a predecir serán:

```
train_X=train[cols]
train_Ycasual=train['casual']
train_Yreg=train['registered']
train_Ycount=train['count']
test_X=test[cols]
test_Ycasual=test['casual']
test_Yreg=test['registered']
test_Ycount=test['count']
```

Figura 3. 3 Distintos conjuntos de entrenamiento

Indicar que el código empleado en todo este trabajo está desarrollado en el Anexo I de este documento.

3.1 REGRESIÓN LINEAL MÚLTIPLE

Se aplica el modelo de regresión lineal de la librería Sklearn (*Hauck, 2014*) y se importa los distintos procedimientos de métricas para el modelo de regresión lineal múltiple a aplicar, teniendo en cuenta que excepto RMSLE el resto proceden de las librerías disponibles en Anaconda:

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
```

Figura 3. 4 Instalación de las librerías necesarias para la métrica

Por este motivo se ejecuta una función definiendo a RMSLE:

```
from math import log
def rmsle(y, y_pred):
    y=y.values
    assert len(y) == len(y_pred)
    terms_to_sum = [(math.log(y_pred[i] + 1) - math.log(y[i] + 1)) ** 2.0 for i,pred in enumerate(y_pred)]
    return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5
```

Figura 3. 5 Función RMSLE

Una vez el modelo preparado, se realiza el entrenamiento en cada una de las distintas variables a predecir, a partir de la función *fit* e introduciéndole el conjunto de datos referente a la variable dependiente:

```
reg = linear_model.LinearRegression()
reg.fit(train_X,train_Ycasual)
reg1 = linear_model.LinearRegression()
reg1.fit(train_X,train_Yreg)
reg2 = linear_model.LinearRegression()
reg2.fit(train_X,train_Ycount)
reg3= linear_model.LinearRegression()
reg3.fit(train_X,train_Ycasual+train_Yreg)
```

Figura 3. 6 Entrenamiento del Regresión Lineal Múltiple

Una vez entrenado los distintos algoritmos se le realiza la predicción sobre el conjunto particionado con validación cruzada al inicio de la programación:

```
predict_ycasual = reg.predict(test_X)
predict_ycasual
```

```
array([  10.82300405,    8.44015771,  112.36312056, ...,
  39.750472  ,
         24.20147368,   77.95688125])
```

Figura 3. 7 Predicción del modelo entrenado casual

Devolviendo un array con cada una de las predicciones realizadas a partir del algoritmo entrenado. Esto se aplica en cada conjunto de entrenamiento definido:

```
predict_yreg = reg1.predict(test_X)
predict_ycount = reg2.predict(test_X)
predict_ytotal = reg3.predict(test_X)
```

Figura 3. 8 Predicción de los distintos modelos entrenados

Para buscar mejores predicciones, en el caso del algoritmo de Regresión Lineal Múltiple, a los datos del modelo se le aplica normalización de los modelos de ajuste para la predicción. Por lo que se importa la variable ***"scale"*** que permite normalizar los parámetros:

```
from sklearn.preprocessing import scale
```

Figura 3. 9 Importar función Escalar

Se entrenan nuevos modelos de predicción en base a la normalización de parámetros:

```
norm=linear_model.LinearRegression(normalize=scale)
normcas=norm.fit(train_X,train_Ycasual)
normreg= norm.fit(train_X,train_Yreg)
normcount=norm.fit(train_X,train_Ycount)
```

Figura 3. 10 Normalización y entrenamiento del modelo

Se establecen las predicciones para cada conjunto de entrenamiento normalizado:

```
prednorcas=normcas.predict(test_X)
prednorreg=normreg.predict(test_X)
prednorcount=normcount.predict(test_X)
```

Figura 3. 11 Predicción del modelo normalizado

En resultados se comparará el valor obtenido por estos dos procesos.

3.2 POISSON

Para la aplicación del algoritmo de Poisson es necesario emplear la librería *Statsmodel,* que permite explorar datos a partir de modelos estadísticos, como lo es la regresión según la distribución discreta de Poisson. Instalamos la librería:

```
from statsmodels.genmod.families import Poisson
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

Figura 3. 12 Instalación de las librerías para Poisson

Definimos las distintas rectas de regresión, definidas en la ecuación 3 de regresión lineal múltiple despejando la variable y o variable dependiente. En este caso hay que despejar las distintas rectas de regresión para las distintas variables de predicción según la variable dependiente. A continuación se agrupa todas las variables independientes del modelo para simplificar la fórmula de regresión:

```
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1', 'Tiempo_2','Tiempo_3', 'Tiempo_4',
      'Estación_1', 'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a
```

Figura 3. 13 Definición de las variables independientes de la función

Donde a es la suma de todas las variables independientes del modelo, las cuales restan a las distintas variables dependientes o de predicción:

```
a = " + ".join(cols)
form='casual ~ '+a
form1='registered ~ '+a
form2='count ~ '+a
```

Figura 3. 14 Definición de las distintas funciones

Entrenamos el primer modelo, para la variable dependiente de usuarios casuales, donde se define la fórmula que se aplica, el dataframe que contiene los valores de dichas variables presentes en la fórmula, y la distribución que se establece, en este caso Poisson:

```
mod = smf.glm(formula=form, data=dtt, family=sm.families.Poisson()).fit()
mod.pvalues
```

```
Intercept             3.798770e-98
temp                  0.000000e+00
holiday              3.335414e-122
humidity              0.000000e+00
windspeed             2.020903e-83
Tiempo_1              1.197035e-02
Tiempo_2              1.507832e-04
Tiempo_3              6.186095e-01
Tiempo_4              5.401111e-02
Estación_1            8.959313e-55
Estación_2           9.662426e-216
Estación_3            2.380256e-07
Estación_4           2.980980e-118
workingday           7.717965e-146
month                 5.022298e-98
hour                  0.000000e+00
dayweek_Friday       6.093568e-201
dayweek_Monday        2.156657e-12
dayweek_Saturday     1.109795e-293
dayweek_Sunday       2.064023e-256
dayweek_Thursday      1.400597e-28
dayweek_Tuesday       3.718456e-29
dayweek_Wednesday     2.173020e-36
dtype: float64
```

Figura 3. 15 Entrenamiento y valores del modelo casual

El código devuelve los *pvalues* de cada una de las variables independientes del modelo. Los *pvalues* representan el nivel de correlación de esa variable con la variable dependiente o de predicción. Más adelante se analizará estos valores de manera significativa. A continuación se procede de la misma forma para el resto de variables de predicción:

```
mod1 = smf.glm(formula=form1, data=dtt, family=sm.families.Poisson()).fit()
mod1.pvalues
```

```
Intercept             0.000000e+00
temp                  0.000000e+00
holiday               0.000000e+00
humidity              0.000000e+00
windspeed            7.924613e-231
Tiempo_1             3.622980e-100
Tiempo_2             1.426745e-135
Tiempo_3              1.468037e-10
Tiempo_4              3.947106e-75
Estación_1            0.000000e+00
Estación_2            0.000000e+00
Estación_3            0.000000e+00
Estación_4            0.000000e+00
workingday            0.000000e+00
month                 0.000000e+00
hour                  0.000000e+00
dayweek_Friday        0.000000e+00
dayweek_Monday        0.000000e+00
dayweek_Saturday      0.000000e+00
dayweek_Sunday        0.000000e+00
dayweek_Thursday      0.000000e+00
dayweek_Tuesday       0.000000e+00
dayweek_Wednesday     0.000000e+00
dtype: float64
```

Figura 3. 16 Entrenamiento y valores del modelo registrado

```
mod2= smf.glm(formula=form2, data=dtt, family=sm.families.Poisson()).fit()
mod2.pvalues
```

```
Intercept             0.000000e+00
temp                  0.000000e+00
holiday               0.000000e+00
humidity              0.000000e+00
windspeed            9.790485e-272
Tiempo_1             1.454704e-112
Tiempo_2             7.551582e-158
Tiempo_3              4.904124e-18
Tiempo_4              3.642204e-90
Estación_1            0.000000e+00
Estación_2            0.000000e+00
Estación_3            0.000000e+00
Estación_4            0.000000e+00
workingday            0.000000e+00
month                 0.000000e+00
hour                  0.000000e+00
dayweek_Friday        0.000000e+00
dayweek_Monday        0.000000e+00
dayweek_Saturday      0.000000e+00
dayweek_Sunday        0.000000e+00
dayweek_Thursday      0.000000e+00
dayweek_Tuesday       0.000000e+00
dayweek_Wednesday     0.000000e+00
dtype: float64
```

Figura 3. 17 Entrenamiento y valores del modelo total

Se calcula las predicciones para cada modelo entrenado y en base a los datos que forman el conjunto de prueba:

```
po=mod.predict(test_X)
po
```

```
array([   9.85387444,   71.58355628,  116.616
11717, ...,   31.42759958,
         11.42009477,   34.26601203])
```

```
po1=mod1.predict(test_X)
po2=mod2.predict(test_X)
```

Figura 3. 18 Predicción de los distintos modelos de entrenamiento

3.3 ÁRBOL DE DECISIÓN

Como en los algoritmos anteriores, se comienza a partir del pretratamiento de datos y validación cruzada del conjunto de entrenamiento, así como su subdivisión en las distintas variables dependientes que se definen en el modelo.

Para comenzar con la aplicación del Árbol de Decisión se debe importar de la librería Sklearn el algoritmo básico. A continuación se representa la acción de importar la librería, así como los distintos modelos de entrenamiento a realizar:

```
from sklearn.tree import DecisionTreeRegressor
```

```
casual_tree = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
casual_tree.fit(train_X,train_Ycasual)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_samples_leaf=10, min_samples_split=30,
           min_weight_fraction_leaf=0.0, presort=False, random_state=0,
           splitter='best')
```

```
reg_tree = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
reg_tree.fit(train_X,train_Yreg)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_samples_leaf=10, min_samples_split=30,
           min_weight_fraction_leaf=0.0, presort=False, random_state=0,
           splitter='best')
```

```
count_tree = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
count_tree.fit(train_X,train_Ycount)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_samples_leaf=10, min_samples_split=30,
           min_weight_fraction_leaf=0.0, presort=False, random_state=0,
           splitter='best')
```

Figura 3. 19 Instalación de la función de Árbol de Decisión y entrenamiento de modelos

Donde:

- min_samples_split, es el número de muestras necesarias para dividir un nodo interno.
- min_samples_leaf, es el número mínimo de muestras necesarias para estar en un nodo de la hoja del árbol.
- random_state, número de instancias en estado aleatorio, es la semilla aleatoria generadora de números aleatorios. En este modelo no es necesario y se le asigna valor cero.

Son los parámetros que definen el tipo de árbol a aplicar, se ha aplicado el mismo valor para los distintos modelos de regresión con el fin de valorar y comparar los distintos resultados.

El siguiente paso es la predicción de los modelos entrenados a partir de los datos de entrada definidos por validación cruzada de prueba (test_X):

```
casual_tree_pred=casual_tree.predict(test_X)
reg_tree_pred=reg_tree.predict(test_X)
count_tree_pred=count_tree.predict(test_X)
```

Figura 3. 20 Predicción de modelos entrenados

Otra posibilidad para obtener mejores resultado es aplicar ***Feature Importance*** a cada conjunto de modelos. Esta metodología se calcula para cada variable del conjunto y para un solo árbol de decisión, y se consiguen mediante la aplicación del siguiente código para cada conjunto de datos casuales, registrados y usuarios totales:

```
peso=casual_tree.feature_importances_
```

```
dic={'importancia':peso,'variable':train_X.columns}
wr=pd.DataFrame(dic,columns=['variable','importancia'])
```

```
wir=wr.sort_values(by='importancia',ascending=False)
wir
```

```
peso1=reg_tree.feature_importances_
dic1={'importancia':peso1,'variable':train_X.columns}
wr1=pd.DataFrame(dic1,columns=['variable','importancia'])
wir1=wr1.sort_values(by='importancia',ascending=False)
```

```
peso2=count_tree.feature_importances_
dic2={'importancia':peso2,'variable':train_X.columns}
wr2=pd.DataFrame(dic2,columns=['variable','importancia'])
wir2=wr2.sort_values(by='importancia',ascending=False)
```

Figura 3. 21 Aplicación de Feature Importance en usuarios causales

Esta metodología calcula los pesos de importancia de cada variable en cada conjunto de predicción, función de la correlación que tiene cada variable independiente con la variable a predecir en los distintos modelos. Para visualizarlo mejor los pesos de importancia se transforma en la siguiente tabla para facilitar la muestra reflejando el peso de importancia para cada variable del conjunto según la variable dependiente a predecir. A continuación se muestra un ejemplo de la clasificación de ***Feature Importance*** para el conjunto de datos de usuarios casuales:

	variable	importancia
14	hour	0.407485
0	temp	0.291499
12	workingday	0.181654
2	humidity	0.063663
13	month	0.016995
3	windspeed	0.012982
17	dayweek_Saturday	0.008803
15	dayweek_Friday	0.007353
18	dayweek_Sunday	0.005722
6	Tiempo_3	0.001110
9	Estación_2	0.000903
5	Tiempo_2	0.000426
10	Estación_3	0.000307
8	Estación_1	0.000285
4	Tiempo_1	0.000283
16	dayweek_Monday	0.000268
21	dayweek_Wednesday	0.000098
19	dayweek_Thursday	0.000088
20	dayweek_Tuesday	0.000060
11	Estación_4	0.000014
1	holiday	0.000000
7	Tiempo_4	0.000000

Figura 3. 22 Matriz de valores Feature Importance

Para simplificar los modelos se selecciona las 10 variables con mayores pesos de importancia en cada uno de los modelo y con ellas se crea los nuevos conjuntos de entrenamiento para cada conjunto de datos. Se repite el algoritmo de árbol de regresión bajo los mismos valores para los parámetros característicos de dicho algoritmo, anteriormente descrito:

```
col=wir.iloc[0:10].variable.values
```

```
train_Xfttcasual=train[cols]
train_Yfttcasual=train['casual']
test_Xfttcasual=test[cols]
test_Yfttcasual=test['casual']
```

```
freecasual = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
freecasual.fit(train_Xfttcasual,train_Yfttcasual)
fcapred=freecasual.predict(test_Xfttcasual)
```

```
colm=wir.iloc[0:10].variable.values
train_Xfttreg=train[colm]
train_Yfttreg=train['registered']
test_Xfttreg=test[colm]
test_Yfttreg=test['registered']
```

```
freereg = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
freereg.fit(train_Xfttreg,train_Yfttreg)
frepred=freereg.predict(test_Xfttreg)
```

```
colc=wir.iloc[0:10].variable.values
train_Xfttcount=train[colc]
train_Yfttcount=train['count']
test_Xfttcount=test[colc]
test_Yfttcount=test['count']
```

```
freecou = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
freecou.fit(train_Xfttcount,train_Yfttcount)
fcopred=freecou.predict(test_Xfttcount)
```

Figura 3. 23 Aplicación de Feature Importance en usuarios registrados y totales

En función de cada conjunto nuevo creado de cada una de las variables con mayor peso en cada modelo, las variables no tienen porque ser las mismas en los distintos modelos, ya que no tienen las mismas correlaciones las variables independientes con las distintas variables de predicción.

Se realiza el entrenamiento de cada conjunto creado a partir de los 10 pesos más relevantes y se procederá al análisis de puntuación de resultados.

Por último, se presenta la aplicación de la herramienta **GridSearch** que genera distintos valores de un mismo parámetro, permitiendo así valorar el intervalo definido de los distintos parámetros con el que evaluar el ajuste del modelo.

Permite encontrar el valor de cada parámetro que mejor se ajuste al modelo de regresión. Para aplicarlo se necesita importar la librería desde **Sklearn** y definir los valores entre los que se estudiaran cada una de las variables que defina en "***param_grid":***

```
from sklearn.grid_search import GridSearchCV
```

```
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "min_samples_split": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10]}
```

Figura 3. 24 Instalación de GridSearch

En la definición de parámetros se ha establecido:

- **Max_depth**: define la profundidad máxima del árbol. No se establece límite superior para este parámetro.
- **Max_features**: el número de características a buscar una mejor división.
- **Min_samples_split:** define el número mínimo de muestras que se requieren en un nodo a considerar para la división, controlando así el ajuste excesivo.
- **Min_samples_leaf**: define el número mínimo de muestras en un nodo final.

Se realiza el modelo de entrenamiento para los valores de la variable de usuarios totales que se definen a continuación:

```
gridfcou = DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0)
```

Figura 3. 25 Entrenamiento del modelo en las aplicación de GridSearch

Esto formará parte del modelo de entrenamiento y predicción completa se analizara en el siguiente punto de **Resultados** ya que esta metodología es necesario especificar el conjunto de entrenamiento de datos y el procedimiento de puntuación en la misma línea de código.

3.4. RANDOM FOREST

Método denominado ***"bosque aleatorio***", es la aplicación del algoritmo anterior de en varias ocasiones, según se defina (*Patil, 2015*). Para aplicarlo es necesario importar de la librería *Sklearn* la metodología ***RandomForestRegressor***, como se muestra en la siguiente imagen, además del resto de procedimientos anteriormente descritos en los algoritmos como es la partición del conjunto de entrenamiento:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.cross_validation import train_test_split
train, test = train_test_split(dtt, test_size = 0.2)
```

Figura 3. 26 Instalación de RandomForest

Se aplica a cada conjunto de entrenamiento creado para cada variable de dependiente o de predicción establecida. Se comienza con la definición de los modelos de entrenamiento basados en el algoritmo Random Forest de parámetros:

- **N_estimators**: es el número de árboles que componen al bosque, en nuestro modelo se establece el valor de 500 árboles, siguiendo el criterio de la amplitud de datos del modelo.
- **N_jobs**: es el número de maquinas en paralelo establecido, para acelerar el proceso se define 2.
- **Oob_score**: es True para emplear muestras fuera de bolsa con el fin de estimar datos no previstos.

A continuación se entrenan los modelos de ajustes para cada conjunto de datos según las variables dependientes definidas en este modelo:

```
rfcasual= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
rfcasual.fit(train_X,train_Ycasual)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=2, oob_score=True, random_state=None,
           verbose=0, warm_start=False)
```

```
rfreg= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
rfreg.fit(train_X,train_Yreg)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=2, oob_score=True, random_state=None,
           verbose=0, warm_start=False)
```

```
rfcount= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
rfcount.fit(train_X,train_Ycount)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=2, oob_score=True, random_state=None,
           verbose=0, warm_start=False)
```

Figura 3. 27 Entrenamiento de los distintos modelos en RandomForest

El siguiente paso es establecer los modelos de predicción en base al entrenamiento realizado para cada conjunto de datos de prueba generados en la validación cruzada del conjunto de entrenamiento inicial:

```
casualpr=rfcasual.predict(test_X)
regpr=rfreg.predict(test_X)
countpr=rfcount.predict(test_X)
```

Figura 3. 28 Predicción de los modelos entrenados

Otra herramienta a aplicar, como en el algoritmo anterior y en el que está basado este es para obtener mejores resultado es ***Feature Importance*** a cada conjunto de modelos. Esta metodología se calcula para cada variable del conjunto y para un solo árbol de decisión, y se consiguen mediante la aplicación del siguiente código para cada conjunto de datos casuales, registrados y usuarios totales:

```
importcasual=rfcasual.feature_importances_
importcasual
```

```
array([  2.63376183e-01,   2.58196357e-03,   8.165120
44e-02,
         2.77968470e-02,   3.69813841e-03,   2.361497
81e-03,
         2.44120322e-03,   9.84631336e-08,   3.238925
52e-03,
         4.58486084e-03,   1.85022291e-03,   1.669883
02e-03,
         1.77971751e-01,   2.55923945e-02,   3.764713
66e-01,
         6.22478307e-03,   2.63982628e-03,   7.562836
80e-03,
         3.04689952e-03,   1.14147198e-03,   9.164305
04e-04,
         3.18121163e-03])
```

```
importregist=rfreg.feature_importances_
importregist
```

```
importcount=rfcount.feature_importances_
importcount
```

Figura 3. 29 Aplicación de Feature Importance en los distintos modelos

Se vuelve a definir un dataframe con los nombres de las variables y el peso de importancia de esa variable independiente en la predicción de las variables dependientes de los distintos usuarios:

```
d={'importancia':importcasual,'variable':train_X.columns}
fti=pd.DataFrame(d,columns=['variable','importancia'])
```

```
mfti=fti.sort_values(by='importancia',ascending=False)
mfti
```

```
d1={'importancia':importregist,'variable':train_X.columns}
ftir=pd.DataFrame(d1,columns=['variable','importancia'])
```

```
mftir=ftir.sort_values(by='importancia',ascending=False)
mftir
```

```
d2={'importancia':importcount,'variable':train_X.columns}
ftic=pd.DataFrame(d2,columns=['variable','importancia'])
mftic=ftic.sort_values(by='importancia',ascending=False)
mftic
```

Figura 3. 30 Obtención de los valores de Feature Importance

Las variables más importantes para los distintos modelos de entrenamiento se muestran en tablas creadas por el dataframe anterior, en este caso se vuelve a mostrar un ejemplo de la tabla de usuarios casuales en función de más a menos importante:

	variable	importancia
14	hour	3.764714e-01
0	temp	2.633762e-01
12	workingday	1.779718e-01
2	humidity	8.165120e-02
3	windspeed	2.779685e-02
13	month	2.559239e-02
17	dayweek_Saturday	7.562837e-03
15	dayweek_Friday	6.224783e-03
9	Estación_2	4.584861e-03
4	Tiempo_1	3.698138e-03
8	Estación_1	3.238926e-03
21	dayweek_Wednesday	3.181212e-03
18	dayweek_Sunday	3.046900e-03
16	dayweek_Monday	2.639826e-03
1	holiday	2.581964e-03
6	Tiempo_3	2.441203e-03
5	Tiempo_2	2.361498e-03
10	Estación_3	1.850223e-03
11	Estación_4	1.669883e-03
19	dayweek_Thursday	1.141472e-03
20	dayweek_Tuesday	9.164305e-04
7	Tiempo_4	9.846313e-08

Figura 3. 31 Valores obtenidos de Feature Importance

Establezco la selección de variables según las diez primeras, con mayores pesos de importancia según el criterio de correlación:

```
mfti.iloc[0:10].variable.values
```

```
array(['hour', 'temp', 'workingday', 'humidity', 'windspeed', 'mont
h',
       'dayweek_Saturday', 'dayweek_Friday', 'Estación_2', 'Tiempo_
1'], dtype=object)
```

Figura 3. 32 Selección de las 10 primeras variables

Creo un nuevo conjunto de datos con las variables independientes más relevantes para el modelo de usuarios casuales según el siguiente código:

```
colu=mfti.iloc[0:10].variable.values
```

```
train_Xftcasual=train[colu]
train_Yftcasual=train['casual']
test_Xftcasual=test[colu]
test_Yftcasual=test['casual']
```

```
colu2=mftir.iloc[0:10].variable.values
```

```
train_Xftreg=train[colu2]
train_Yftreg=train['registered']
test_Xftreg=test[colu2]
test_Yftreg=test['registered']
```

```
colu3=mftic.iloc[0:10].variable.values
```

```
train_Xftcount=train[colu3]
train_Yftcount=train['count']
test_Xftcount=test[colu3]
test_Yftcount=test['count']
```

Figura 3. 33 Creación de nuevos conjuntos de datos

A partir de los nuevos conjuntos de entrenamiento y prueba, los modelos de los nuevos conjuntos de predicción para todos los usuarios en función de los nuevos conjuntos de entrenamiento, train_Xftcasual, train_Yftcasual, train_Xftreg, train_Yftreg, train_Xftcount y train_Yftcount :

```
ftcasual= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
ftcasual.fit(train_Xftcasual,train_Yftcasual)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=2, oob_score=True, random_state=None,
           verbose=0, warm_start=False)
```

```
ftpc=ftcasual.predict(test_Xftcasual)
ftpc
```

```
array([ 35.77 ,  16.402,   1.118, ...,   0.182,   1.614,  13.79 ])
```

```
ftreg= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
ftreg.fit(train_Xftreg,train_Yftreg)
ftpr=ftreg.predict(test_Xftreg)
ftpr
```

```
array([ 123.98333333,  150.436     ,   18.098     , ...,   17.088
  ,
        24.222     ,  330.98      ])
```

```
ftcount= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
ftcount.fit(train_Xftcount,train_Yftcount)
ftpco=ftcount.predict(test_Xftcount)
ftpco
```

```
array([ 119.962,  166.374,   16.558, ...,   17.75 ,   24.378,  364.154])
```

Figura 3. 34 Entrenamiento de nuevos modelos

Se crea nuevos conjuntos de prueba en base a las 10 variables con mayor peso para cada modelo de entrenamiento según el tipo de usuario.

Por último, al igual que el algoritmo anterior **Árbol de decisión**, para la búsqueda de otros valores se aplica la herramienta ***GridSearch*** para generar un conjunto de valores de cada parámetro que se ajuste de manera más eficiente al modelo:

```
from sklearn.grid_search import GridSearchCV
```

```
param_grid = {"max_depth": [3, 5, None],
              "max_features": [None, 'sqrt'],
              "min_samples_split": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10]}
```

```
rfrcount= RandomForestRegressor(n_estimators=100,n_jobs=2)
```

Figura 3. 35 Instalación y definición de las variables de GridSearch

3.5. SVM

Para aplicar el algoritmo de Máquina de Soporte Vectorial en regresión, es necesario definir qué tipo de kernel deseamos para la agrupación de puntos, un ejemplo en modelos de clasificación son las siguientes imágenes:

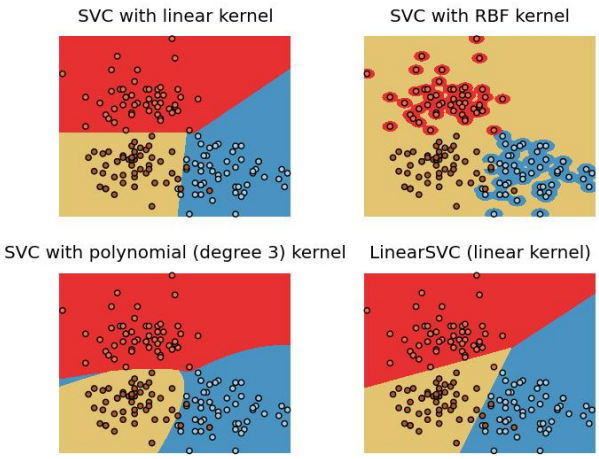

Figura 3. 36 Representación de kernels en SVM

Trabajaremos con tres tipos de kernel, que serán RBF, lineal y polinomio. Es necesario, para comenzar, importar la librería de soporte vectorial de regresión desde (SVR) desde la librería de Sklearn con la función máquina de soporte vectorial (svm), código que se muestra a continuación:

```
from sklearn.svm import SVR
```

Figura 3. 37 Instalación de la función SVM en regresión

3.5.1. RBF

RBF en aprendizaje automático es la función base radial (Radial base function), o kernel gaussiano de parámetros gamma y C, ambos positivos (Chang Chung, 2013). El parámetro gamma hace referencia al radio del tubo que crea el algoritmo alrededor del conjunto de datos a entrenar, mientras que C establece el nivel de ajuste a los datos de entrenamiento (*Hsu, 2016*). Un C bajo crea una superficie una superficie de decisión

pequeña siendo el modelo de entrenamiento más restrictivo, mientras cuanto mayor es, la superficie de decisión es mayor, incluyendo un mayor conjunto de entrenamiento, incluyendo más muestras del soporte vectorial.

El valor de gamma establece la sensibilidad del modelo, puede variar entre los valores $(10^{-3}, 10^{3})$. Si el valor de gamma es grande el radio del tubo solo incluye el propio vector de soporte, aunque el valor de C este sobreajustado. Si por el contrario, es pequeño su valor, el modelo es demasiado limitado y no captaría la complejidad de los datos de entrenamiento.

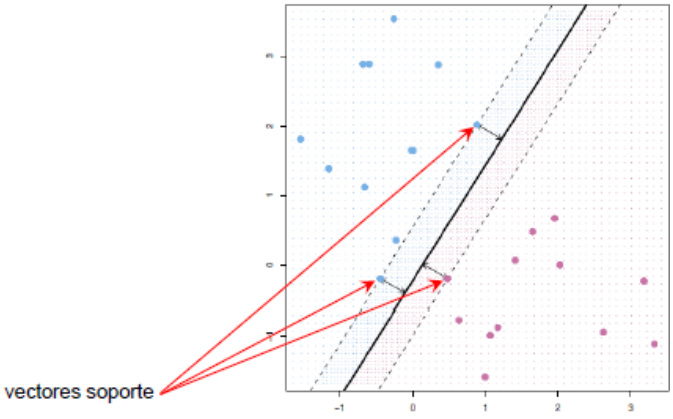

Figura 3. 38 Representación de los parámetros de RBF

Definimos los distintos conjuntos de entrenamiento a partir de los valores establecidos de gamma y C representados en la siguiente imagen:

```
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.001)
svr = svr_rbf.fit(train_X, train_Ycount)
```

```
svr1 = svr_rbf.fit(train_X, train_Yreg)
```

```
svr2 = svr_rbf.fit(train_X, train_Ycasual)
```

Figura 3. 39 Entrenamiento de los modelos RBF

Se calcula el modelo de predicción de la misma forma que en algoritmos anteriores, mediante el conjunto de datos de prueba:

```
svr_pred=svr.predict(test_X)
svr_pred
```

```
array([   8.42349789,    3.49512935,  123.24599149, ...,
34.44081098,
         57.21697477,    1.68966455])
```

```
svr1_pred=svr1.predict(test_X)
svr1_pred
```

```
array([   8.42349789,    3.49512935,  123.24599149, ...,
34.44081098,
         57.21697477,    1.68966455])
```

```
svr2_pred=svr2.predict(test_X)
svr2_pred
```

```
array([   8.42349789,    3.49512935,  123.24599149, ...,
34.44081098,
         57.21697477,    1.68966455])
```

Figura 3. 40 Predicción de los modelos de entrenamiento RBF

A partir de estos valores se aplicaran los distintos procedimientos de puntuación para el ajuste del modelo predicho.

Para conseguir que el modelo se ajuste de forma efectiva se aplica una escala entre 0 y 1 a los valores de los parámetros para su estandarización, eliminando así los posibles valores negativos en los modelos de predicción y en los procedimientos de puntuación. Los valores categóricos respetaran sus valores 0 y 1 mientras que el resto de variables se encontrarán dentro de estos límites.

Para aplicar la estandarización o escalabilidad es necesario importar la siguiente librería de ***Sklearn*** en el cual se define los límites de estandarización, en este caso 0 y 1 con la variable scaler y su parámetro Feature_range:

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler =MinMaxScaler(feature_range=(0, 1), copy=True)
```

Figura 3. 41 Instalación de la función escalar

Se necesita eliminar la variable datetime, ya que no es una variable numérica y no es posible estandarizarlo. A continuación se crea un nuevo dataframe a partir de la estandarización de las variables:

```
dtt_sc=scaler.fit_transform(dtt)
```

```
dttsca=pd.DataFrame(dtt_sc,columns=dtt.columns)
dttsca.head()
```

Figura 3. 42 Creación de un nuevo conjunto de datos

Se muestra el valor que obtienen algunas de las variables del modelo:

	holiday	workingday	temp	atemp	humidity	windspeed	casual	registered	count
0	0.0	0.0	0.224490	0.305068	0.81	0.0	0.008174	0.014673	0.015369
1	0.0	0.0	0.204082	0.288064	0.80	0.0	0.021798	0.036117	0.039959
2	0.0	0.0	0.204082	0.288064	0.80	0.0	0.013624	0.030474	0.031762
3	0.0	0.0	0.224490	0.305068	0.75	0.0	0.008174	0.011287	0.012295
4	0.0	0.0	0.224490	0.305068	0.75	0.0	0.000000	0.001129	0.000000

5 rows × 27 columns

Figura 3. 43 Valores del nuevo conjunto de datos

El siguiente paso es entrenar los distintos modelos de predicción a partir de los valores resultantes de la escala tomada en todas las variables independientes:

```
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.001)
svr = svr_rbf.fit(train_X, train_Ycount)
```

```
svr1 = svr_rbf.fit(train_X, train_Yreg)
```

```
svr2 = svr_rbf.fit(train_X, train_Ycasual)
```

```
svr_pred=svr.predict(test_X)
svr_pred
```

```
array([ 0.12405114,  0.083003  ,  0.06590741, ...,  0.016226
82,
        0.16500206,  0.15898309])
```

```
svr1_pred=svr1.predict(test_X)
svr1_pred
```

```
array([ 0.12405114,  0.083003  ,  0.06590741, ...,  0.016226
82,
        0.16500206,  0.15898309])
```

```
svr2_pred=svr2.predict(test_X)
svr2_pred
```

```
array([ 0.12405114,  0.083003  ,  0.06590741, ...,  0.016226
82,
        0.16500206,  0.15898309])
```

Figura 3. 44 Entrenamiento y predicción de los nuevos modelos para RBF

En el apartado de Resultados se compararan los resultados obtenido sin la aplicación de la estandarización y sin él.

3.5.2. Lineal

Se aplica a partir de la función de soporte de maquina vectorial para regresión SVR, pero a diferencia del modelo de RBF, solo es necesario el valor de la variable C, manteniéndose en el mismo valor:

```
svr_lineal = SVR(kernel='linear',C=1e3)
```

```
svrl= svr_lineal.fit(train_X, train_Ycount)
svrl1= svr_lineal.fit(train_X, train_Yreg)
svrl2= svr_lineal.fit(train_X, train_Ycasual)
```

Figura 3. 45 Definición de parámetros de Linear y entrenamiento de modelos

Se procede de la misma manera, se entrenan los modelos a partir de los distintos conjuntos de datos definidos al inicio. A partir de estos se realiza el modelo de predicción del conjunto de prueba a partir del kernel lineal:

```
svrl_pred=svrl.predict(test_X)
svrl_pred
```

```
array([-29.40958271,  45.03137493,  33.60031607, ...,  -8.009210
66,
        31.21601841,  55.18568401])
```

```
svrl1_pred=svrl1.predict(test_X)
svrl1_pred
```

```
array([-29.40958271,  45.03137493,  33.60031607, ...,  -8.009210
66,
        31.21601841,  55.18568401])
```

```
svrl2_pred=svrl2.predict(test_X)
svrl2_pred
```

```
array([-29.40958271,  45.03137493,  33.60031607, ...,  -8.009210
66,
        31.21601841,  55.18568401])
```

Figura 3. 46 Predicción de los modelos Linear

Tras definir el modelo de predicción se evaluara el ajuste de predicción de cada conjunto de datos.

Al igual que en la función RBF, se ha aplicado las estandarización de los valores de cada variable del modelo. Se comparara los resultados obtenidos en el siguiente punto del proyecto.

3.5.3. Polinomial

A partir de la función SVR de la librería **Sklearn** definimos el kernel o función polinomio a partir de los parámetros C, anteriormente definido, y el grado del polinomio a establecer en los modelos de regresión:

```
svr_poly = SVR(kernel='poly',C=1, degree=2)

svrp= svr_poly.fit(train_X, train_Ycount)

svrp1= svr_poly.fit(train_X, train_Yreg)

svrp2= svr_poly.fit(train_X, train_Ycasual)
```

Figura 3. 47 Definición de parámetros de Poly y entrenamiento de modelos

Como en el resto de modelos de SVM, se entrenan los modelos de los distintos conjuntos de datos definidos. Se calcula el modelo de predicción del conjunto de prueba a partir del kernel polinomial:

```
svrp_pred=svrp.predict(test_X)
svrp_pred
```

```
array([  2.61191347,  55.25434117,  27.46511984, ...,   6.12918497,
        29.93718424,  22.83145346])
```

```
svrp1_pred=svrp1.predict(test_X)
svrp1_pred
```

```
array([  2.61191347,  55.25434117,  27.46511984, ...,   6.12918497,
        29.93718424,  22.83145346])
```

```
svrp2_pred=svrp2.predict(test_X)
svrp2_pred
```

```
array([  2.61191347,  55.25434117,  27.46511984, ...,   6.12918497,
        29.93718424,  22.83145346])
```

Figura 3. 48 Predicción de los modelos Poly

Tras definir el modelo de predicción se evaluara el ajuste de predicción de cada conjunto de datos.

Al igual que en la función RBF y el modelo lineal, se ha aplicado la estandarización de los valores de cada variable del modelo. Se comparara los resultados obtenidos en el siguiente punto del proyecto.

4 RESULTADOS

En este apartado se procede a realizar los distintos procedimientos de puntuación para cada algoritmo y en cada una de sus variantes de modelos de ajustes para la predicción.

En cada uno de los algoritmos aplicados en la sección 3 ha sido necesario importar las librerías de Sklearn dos de los tres procedimientos de puntuación desarrollados en el apartado 2.3 de este trabajo:

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

Figura 4. 1 Importamos las librerías para las métricas de puntuación

Para el tercer procedimiento ha sido necesario definir una función a partir de la librería ***math*** de la siguiente forma:

```
import math
from math import log
def rmsle(y, y_pred):
    y_pred[y_pred<0]=0
    y=y.values
    assert len(y) == len(y_pred)
    terms_to_sum = [(log(y_pred[i] + 1) - log(y[i] + 1)) ** 2.0 for i,pred in enumerate(y_pred)]
    return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5
```

Figura 4. 2 Función RMSLE

Para asegurar de que no se producen error por valores negativos se establece que los valores de las predicciones que se vayan a realizar que sean menores a 0, se igualarán a 0.

Para implementar el coeficiente de determinación se ha tomado dos opciones de estimación, una es por el algoritmo por defecto en las distintas rectas de regresión (score), y la otra forma sería a partir de la librería importa en la ***figura 4.1*** (r2_score).

Para definir la métrica a aplicar por defecto de regresión es necesario establecer los conjuntos de entrenamiento de las variables independientes y dependientes. Un ejemplo lo estudiamos en la siguiente imagen sobre el algoritmo de regresión lineal múltiple:

```
reg.score(train_X,train_Ycasual)
0.47542329579766074

reg1.score(train_X,train_Yreg)
0.28121960321739237

reg2.score(train_X,train_Ycount)
0.34610836567776904

reg3.score(train_X,train_Ycasual+train_Yreg)
0.34610836567776904

r2_score(test_Ycasual,predict_ycasual)
0.46569562337218862

r2_score(test_Yreg,predict_yreg)
0.29723135198800232

r2_score(test_Ycount,predict_ycount)
0.35531078139932493

r2_score(test_Ycasual+test_Yreg,predict_ycasual+predict_yreg)
0.35531078139932493
```

Figura 4. 3 Ejemplo del cálculo de coeficiente de determinación

La diferencia entre estas dos opciones no es significativa en cuanto a los resultados obtenidos, pero se seguirá aplicando para el análisis de resultado entre los distintos algoritmos.

El siguiente procedimiento de puntuación es el error cuadrático medio (mean squared error), también aplicado a los distintos conjuntos de datos en el algoritmo de regresión lineal múltiple:

```
mean_squared_error(test_Ycasual,predict_ycasual)
1369.0632079879333

mean_squared_error(test_Yreg,predict_yreg)
16627.588558113868

mean_squared_error(test_Ycount,predict_ycount)
22098.511886739594

mean_squared_error(test_Ycasual+test_Yreg,predict_ytotal)
22098.511886739594
```

Figura 4. 4 Ejemplo del cálculo de error cuadrático medio

Tampoco este modelo de predicción se ajusta bien al modelo de datos a predecir, puesto que cuanto menor sea, mejor será su ajuste.

Por último, se aplica el algoritmo RMSLE por el que, al emplear logaritmo será necesario anular cualquier predicción negativa que haya podido crear el modelo de ajuste:

```
predict_ycasual[predict_ycasual<0]=0
predict_yreg[predict_yreg<0]=0
predict_ycount[predict_ycount<0]=0
```

Figura 4. 5 Establecer valores negativos a cero

El siguiente paso será introducirle los valores predichos y reales del modelo, siguiendo con el ejemplo del algoritmo de regresión lineal múltiple:

```
rmsle(test_Ycasual.values,predict_ycasual)
```

```
1.2241160965937148
```

```
rmsle(test_Yreg.values,predict_yreg)
```

```
1.2427297451238517
```

```
rmsle(test_Ycount.values,predict_ycount)
```

```
1.2815171957917946
```

Figura 4. 6 Ejemplo del cálculo de RMSLE

Este algoritmo tampoco genera en este conjunto de datos un modelo de predicción ajustado al conjunto de datos de los que se dispone, ya que no se aproxima a 0.

4.1. RESOLUCIÓN GENERAL DE LOS DISTINTOS ALGORITMOS DE REGRESIÓN

Para todos los algoritmos de regresión explicados anteriormente se ha aplicado las distintas metricas de valoración, que son R^2, MSE y RMSLE.

El algoritmo de Poisson, no se ha incluido en la tabla puesto que no se aplica bajo las mismas condiciones ni desde la misma librería, puesto que la librería Statsmodels, pero si se tendrá en cuenta los resultados de las distintas metricas aplicadas en ella

Tabla 1 Valores de las métricas aplicadas en el algoritmo Poisson

		R^2	MSE	RSMLE
POISSON	Casual	0,51	1264,6	1,08
	Registered	0,23	17197,74	1,25
	Count	0,3	228077,25	1,22

En la siguiente tabla se muestra el resto de resultados con la indicación de la metodología seguida, explicada en apartados anteriores y en cada métrica establecida:

Tabla 2 Valores de las distintas métricas en los distintos algoritmos

				Coef. Determinación	R^2	MSE	RSMLE
Regresión lineal múltiple	Inicial	Casual		0,477	0,459	1382,55	1,248
		Registered		0,289	0,269	16740,94	1,275
		Count		0,353	0,331	22064,77	1,275
		Suma		0,353	0,331	22064,77	1,227
	Normalización	Casual		0,472	0,483	1206,67	1,233
		Registered		0,286	0,288	16126,92	1,277
		Count		0,349	0,35	21055,64	1,288
		Suma		0,35	0,34	20913,22	1,227
Árbol de Decisión	Inicial	Casual		0,824	0,83	403,88	0,6
		Registered		0,819	0,819	4237,34	0,45
		Count		0,823	0,823	6794,86	0,46
		Suma		0,823	0,838	5466,64	0,43
	Feature Importance	Casual		0,878	0,824	434,76	0,6
		Registered		0,867	0,811	4400,9	0,46
		Count		0,869	0,801	6637,18	0,46
		Suma		0,869	0,831	5665,6	0,45
	GridSearch	Casual		-	0,81	-471,57	1,646
		Registered		-	0,77	5142,66	1,5217
		Count		-	0,77	7496,3	1,5364
		Suma		-	0,77	7496,3	1,536
Random Forest	Inicial	Casual		0,89	0,89	286,66	0,533
		Registered		0,86	0,874	3060,5	0,412
		Count		0,872	0,879	4203,94	0,409
		Suma		0,872	0,885	3985,69	0,403
	Feature Importance	Casual		0,984	0,883	306,53	0,542
		Registered		0,979	0,86	3163,36	0,422
		Count		0,98	0,868	4354,69	0,421
		Suma		0,98	0,873	4165,17	0,411
	GridSearch	Casual		-	0,883	288	1,351
		Registered		-	0,85	3416,91	1,288
		Count		-	0,86	4559	1,282
		Suma		-	0,86	4555,82	1,28553
SVM	RBF	Inicial	Casual	0,753	0,734	610,29	0,74
			Registered	-0,48	-0,474	33150,16	1,92
			Count	0,51	-0,536	49280,04	2,04
			Suma	-0,54	-0,127	36190,23	1,59
		Estand.	Casual	-	0,612	0,0073	0,0665
			Registered	-	-0,039	0,031	0,137
			Count	-	0,052	0,034	0,141
			Suma	-	0,1	0,032	0,134

	Lineal	Inicial	Casual	0,39	0,39	1463,39	1,31
			Registered	-0,55	-0,58	35134,99	2,21
			Count	-0,62	-0,65	52982,24	2,31
			Suma		-0,3	41615,2	2,08
		Estand.	Casual	0,47	0,61	0,01	0,077
			Registered	-0,11	-0,067	0,032	0,136
			Count	-0,04	-0,012	0,036	0,144
			Suma	-	0,074	0,033	0,135
	Polinomial	Inicial	Casual	0,589	0,6	982,75	0,99
			Registered	0,314	-0,57	34799,61	1,99
			Count	0,418	-0,64	52453,16	2,13
			Suma	0,255	-0,25	39983,06	1,74
		Estand.	Casual	-	0,59	1002,46	1,01
			Registered	-	-0,53	35718,56	1,97
			Count	-	-0,59	53563,9	2,1
			Suma	-	-0,21	40678,95	1,7

5 DISCUSIÓN DE LOS RESULTADOS

En este apartado se comparan los resultados obtenidos en la aplicación de los distintos algoritmos a los procedimientos de métricas.

Observando la *Tabla 2*, concretamente la columna "*Coef. De Determinación*" no ha sido posible aplicarlo en todos los algoritmos ya que no existe la función para los códigos de dicho algoritmo o existen problemas de recursos computacionales del que se dispone.

Para está métrica, el mejor resultado se consigue con la aplicación de *RandomForest,* en la versión de *Feature Importance*, donde se selecciona de las variables con mayor peso de importancia para cada conjunto de datos.

Para los valores de esta métrica obtenidos en el algoritmo *SVM*, se produce un error de ajuste al obtener valores negativos del coeficiente, es decir, se ajusta de forma contraria a los datos con los que se valida, cuando crece el valor predicho disminuye el valor real.

Comparando las dos formas de estimar el coeficiente de determinación, no existe variación para los distintos modelos de regresión, exceptuando el modelo "Suma" que experimenta una leve mejoría con la aplicación existente en algunos algoritmos de la aplicación de "*Coef. Determinación*".

La siguiente métrica a analizar el error cuadrático medio "*MSE*" (Mean Squared Error). Los peores resultados, los valores más elevados, son en los algoritmos de SVM para todos tipos de kernel en su versión original, y en el algoritmo Regresión Lineal Múltiple. En el algoritmo SVM se debe a la definición de los distintos parametros del kernel, ya que no se ha notado mejoría en las variaciones probadas.

Sin embargo, los mejores resultados surgen en los algoritmos de Árbol de Decisión, Random Forest y SVM en su versión estándar, los dos primeros algoritmos en su versión original y sin aplicar ninguna modificación, ya que, en la opción SVM, se anulan los valores picos de las variables. En el modelo Random Forest se establece el árbol de decisión que mejor se ajuste al modelo, por lo que se reduce el error cuadrático medio.

Por último, se analiza el resultado de aplicar la métrica RMSLE. Al igual que en la métrica anterior, los peores resultados se obtienen en la aplicación de SVM en su modelo inicial, y los mejores, exactamente, en los mismos algoritmos que resultan en la aplicación de MSE.

En el modelo SVM sería necesario evaluar otros parámetros en cada uno de los kernels aplicados, con el fin de deducir si este algoritmo de regresión es aplicable para modelos de predicción.

Para evaluar si existe sobreajuste en el resultado de los distintos algoritmos se aplica Validación Cruzada para la valoración de las distintas métricas a aplicar. Es necesario importar la función "cross_val_score" a partir de la librería ***Sklearn***:

```
from sklearn.cross_validation import cross_val_score
```

```
scorel =np.mean( cross_val_score(svrl, train_X, train_Ycount,scoring='r2', cv=10,n_jobs=2))
scorel
```

Figura 5. 1 Instalación y ejemplo de aplicación de la librería de CV de R^2

En la *Figura 5.1* se muestra la forma de establecer la validación cruzada para el coeficiente de determinación. El código devuelve el valor medio a partir de la librería ***Numpy*** del modelo entrenado (svrl), el conjunto de entrenamiento (train_X), el conjunto a predecir (train_Ycount), la métrica a emplear (r2), el valor de k (número de particiones para la comparación=10) y el número de máquinas en paralelo (2).

A continuación se muestra la aplicación de validación cruzada aplicada para la métrica de error cuadrático medio:

```
scorel1 = np.mean(cross_val_score(svrl, train_X, train_Ycount,scoring='mean_squared_error', cv=1
scorel1
```

Figura 5. 2 CV aplicado al error cuadrático medio

A partir de make_scorer, de la librería de ***Sklearn***, se estandarización de la función que realiza la métrica RSMLE para implantarlo como parámetro en la función de validación cruzada, mediante el siguiente código:

```
scorel2= np.mean(cross_val_score(svrl, train_X, train_Ycount,scoring=ms, cv=10))
scorel2
```

```
from sklearn.metrics import make_scorer
```

```
ms = make_scorer(rmsle)
```

Figura 5. 3 CV aplicado a RMSLE

Aplicándolo a cada uno de los algoritmos se obtiene los siguientes valores:

Tabla 3 Valores de las distintas métricas aplicando CV en los distintos algoritmos

Con CV				R^2	MSE	RMSLE
Regresión lineal múltiple	Inicial	Casual		-0,26	1444,67	1,253
		Registered		-0,084	18111,37	1,326
		Count		-0,07	23905,9	1,316
	Normalización	Casual		-1,98	9,82	1,254
		Registered		-4,36	6,42	1,35
		Count		-1,376	2,377	1,335
Árbol de Decisión	Inicial	Casual		0,809	474,74	0,6
		Registered		0,8129	4214,78	0,446
		Count		0,811	6157,81	0,459
		Suma		0,811	6157,81	0,459
	Feature Importance	Casual		0,8	474,74	0,6
		Registered		0,804	4408,76	0,456
		Count		0,807	6259,44	0,462
Random Forest	Inicial	Casual		0,884	286,36	0,55
		Registered		0,851	3392,24	0,411
		Count		0,862	4510,7	0,411
		Suma		0,862	4510,7	0,411
	Feature Importance	Casual		0,878	300,44	0,561
		Registered		0,843	35677,24	0,421
		Count		0,854	4762,84	0,424
		Suma		0,854	4762,84	0,424
SVM	RBF	Inicial	Casual	0,74	662,22	0,74
			Registered	0,38	14137,88	0,81
			Count	0,51	16303,34	0,807
		Estand.	Casual	0,58	0,007	0,066
			Registered	0,347	0,019	0,104
			Count	0,43	0,02	0,105
	Lineal	Inicial	Casual	-	-	-
			Registered	-	-	-
			Count	0,31	-	1,212
			Suma	-	-	-
		Estand.	Casual	0,466	0,01	0,076
			Registered	0,27	0,021	0,111
			Count	0,34	0,023	0,115
	Polinomial	Inicial	Casual	-	-	-
			Registered	-	-	-
			Count	-	-	-
		Estand.	Casual	0,548	0,009	0,073
			Registered	0,318	0,019	0,106
			Count	0,4	0,021	0,11

En La *Tabla 3* la aplicación de la métrica de coeficiente de determinación, como "*Coef. Determinación*", no ha sido evaluada a no ser posible la compatibilidad con la función de validación cruzada. Sin embargo, si se aplica esta métrica a partir de la función que define la librería ***Sklearn**, Figura 4.1.*

De igual forma, el algoritmo Poisson no es evaluado mediante validación cruzada por incompatibilidad de la librería ***Statsmodel*** con la función de ***Sklearn*** de validación cruzada.

Las filas sin valores, representado de esta forma "-", especifican los algoritmos que no se han podido desarrollar por problemas de recursos computacionales del que se dispone, no permitiendo aplicar validación cruzada a los algoritmos SVM, puesto que no permite trabajar con tal cantidad de datos.

Los mejores resultados se obtienen en las versiones estándar del algoritmo SVM, pero esto es debido a los valores entre 0 y 1 que tienen las variables numéricas, por lo que se produciría un sobreajuste.

La plataforma ***Kaggle*** permite evaluar los modelos de ajustes establecidos por los usuarios, de forma que crea ranking con los resultados establecidos en la plataforma por la métrica RSMLE.

Para proceder al ranking es necesario emplear el dataset "***Test***" presente en la plataforma como conjunto de datos a predecir por los modelos definidos por cada usuario de la plataforma.

```
dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()
```

Figura 5. 4 Importar conjunto de datos de prueba

Conjunto de datos formados por los días a predecir, solo presenta las variables independientes. Se muestra las primeros 5 filas del dataset:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	hour	day
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027	2011	1	0	Thursday
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	1	Thursday
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	2	Thursday
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	3	Thursday
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	4	Thursday

Figura 5. 5 Valores del conjunto de datos de prueba

Para trabajar con este conjunto de datos es necesario aplicar exactamente el mismo pretratamiento a las variables independientes que se empleo en el dataset de entrenamiento “Train”. Se procede a tratar las variables dummy y la eliminación de las variables que no son necesarias:

```
dtl.datetime = dtl.datetime.apply(pd.to_datetime)
dtl['year']= dtl.datetime.apply(lambda x : x.year)
dtl['month'] = dtl.datetime.apply(lambda x : x.month)
dtl['hour'] = dtl.datetime.apply(lambda x : x.hour)
dtl['day'] = dtl.datetime.apply(lambda x: x.weekday_name)
dtl.head()
```

```
dum_we=pd.get_dummies(dtl['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dtl['season'],prefix='Estación')
dum_dia=pd.get_dummies(dtl['day'],prefix='dayweek')
```

```
column=dtl.columns.values.tolist()
dt2=dtl[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtest=dto[column2].join(dum_dia)
dtest.head()
```

```
dtest.shape
```

```
(6493, 25)
```

Figura 5. 6 Tratamiento de datos y tamaño de la muestra

Se muestra el número de variables y registros existentes para el modelo, quedando un conjunto de datos de la siguiente forma:

	datetime	holiday	workingday	temp	atemp	humidity	windspeed	year	month	hour	...	Estación_2	Estación_3	Estación_4	dayweek_Friday
0	2011-01-20 00:00:00	0	1	10.66	11.365	56	26.0027	2011	1	0	...	0.0	0.0	0.0	0.0
1	2011-01-20 01:00:00	0	1	10.66	13.635	56	0.0000	2011	1	1	...	0.0	0.0	0.0	0.0
2	2011-01-20 02:00:00	0	1	10.66	13.635	56	0.0000	2011	1	2	...	0.0	0.0	0.0	0.0
3	2011-01-20 03:00:00	0	1	10.66	12.880	56	11.0014	2011	1	3	...	0.0	0.0	0.0	0.0
4	2011-01-20 04:00:00	0	1	10.66	12.880	56	11.0014	2011	1	4	...	0.0	0.0	0.0	0.0

Figura 5. 7 Datos procesados

Para la evaluación es necesario crear el conjunto de entrenamiento y el conjunto de prueba. El conjunto de entrenamiento será el mismo para todos los algoritmos aplicados en este trabajo, formado por las variables del conjunto de datos “***Test***”:

```
X_test = dtest[cols]
```

Figura 5. 8 Definición de variables de entrada

Para el conjunto de prueba se definirá uno por cada algoritmo y variante, el cual recibirá de entrada el modelo entrenado. A continuación se muestra un ejemplo con el modelo de regresión lineal múltiple en RSMLE estimando los valores para las variables de entrenamiento:

```
y_test = reg2.predict(X_test)
```

Figura 5. 9 Ejemplo de definición del modelo de entrenamiento establecido

Para proceder a la evaluación de los distintos algoritmos en la plataforma ***Kaggle,*** es necesario crear un fichero *csv*, en el cual se establece en el enunciado que debe estar formado por la variable "*datetime*" y los valores de "*y_test*". Un ejemplo es la *Figura 5.10.*

```
submission_0 = pd.DataFrame({'datetime':dtest.datetime.values,'count':y_test},columns=['datetime','count'])
submission_0.loc[submission_0['count']<0.0,'count']=0.0
```

Figura 5. 10 Creación de un conjunto de datos

Para crear el conjunto de datos se emplea la librería **Pandas**, mediante la función **DataFrame**. Para evitar problemas derivados de los modelos de ajustes, se establece que los valores estimados menores que 0 valgan 0, puesto que la variable dependiente es de conteo, será imposible que se dé esta circunstancia.

Lo último será convertir la matriz de datos en un fichero:

```
submission_0.to_csv('/Users/Miguel/Desktop/Submissions/sub_0.csv',index=False)
```

Figura 5. 11 Guardar en un archivo csv el nuevo conjunto de datos

Esta metodología se aplicara a cada uno de los algoritmos desarrollados anteriormente con el fin de obtener una valoración externa de los modelos de ajustes.

A continuación se recoge los resultados obtenidos, en la plataforma de la los modelos de predicción definidos:

Tabla 4 Valores de Predicción obtenidos de la plataforma Kaggle

Algoritmos	Resultado Kaggle
Regresión lineal múltiple	1,31760
Regresión lineal múltiple Normalizado	1,32335
Poisson	1,24237
Árbol de Decisión	0,53392
Árbol de Decisión GS	1,55697
Árbol de Decisión GS sumatorio	1,55697
Random Forest	0,50098
Random Forest GS	1,27249
Random Forest GS sumatorio	1,27158
SVM- RBF	0,88789
SVM-Lineal	1,25900
SVM-Polinomio	1,30706
SVM- RBF Estandarizado	4,76189
SVM- Lineal Estandarizado	4,40758
SVM- Polinomio Estandarizado	4,54416

El mejor resultado se obtiene con el algoritmo Random Forest seguido de Árbol de Decisión, esto es debido a la sencillez y rapidez de ambos algoritmos, siendo Random Forest un algorimto basado en Árbol de Decisión.

Random Forest obtiene un mejor ajuste del modelo al formar un conjuto de árboles aleatorios e independientes analizados de manera particular cada uno. La mejora se entiende por la aleatoriedad a la hora de construir los distintos arboles de decisión.

Si observamos el ranking los dos mejores resultados de la métrica RSMLE es 0 y 0,3375, Con el resultado obtenido se sitúa en la posición 5.595 de 12.2961 que forma el ranking.

6 CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

La predicción de la demanda, con la mayor exactitud para estimar la tendencia del mercado, es primordial a la hora de establecer estrategias a medio o largo plazo en cada una de las industrias, así como a la hora de desarrollar presupuestos, contabilidad y la estrategia operativa a desarrollar.

La manera en la que en este trabajo ha abordado la estimación de la predicción ha sido mediante la programación, en Python, de algoritmos de regresión a través de la metodología KDD.

Se ha seleccionado los datos, variables dependientes e independientes, del conjunto de entrenamiento "***Train***", analizando la relación entre dichas variables a fin de establecer que variables son las que entrenarán al modelo de predicción.

El siguiente paso es preprocesar y transforman las variables del conjunto con el objetivo de facilitar el trabajo y entrenamiento del modelo mediante los distintos tipos de variables existentes, como son las variables cuantitativas y las categóricas.

Se continua se modela los distintos algoritmos a aplicar mediante las librerías y funciones existentes en Python y Anaconda para producir los modelos de ajuste según las tendencia de los datos históricos analizados.

Por último, con la obtención de los modelos de ajustes, se aplicará en el modelo de prueba y se analizará la exactitud de la predicción en los distintos resultados de cada algoritmo. Se examina posibles mejoras aplicando distintos tipos de estandarización de las variables. La evaluación del sobreajustes de los modelos se realiza por el método de validación cruzada.

Random Forest es el algoritmo que mejor de ha ajustado a los datos de predicción en cada una de las métricas aplicadas debido a la aleatoriedad que presenta a la hora de formar los distintos arboles del modelo, obteniéndose los siguientes resultados en la métrica evaluada:

Tabla 5 Resumen de resultados

		Sin CV	Con CV	Kaggle
Random Forest	**RMSLE**	0,403	0,411	0,50098

Con la elaboración de este trabajo se ha adquirido nuevas habilidades referentes a la programación, estadística y minería de datos, habilidades muy usadas en los distintos campos por el incremento de las tecnologías en la industria.

Personalmente, este proyecto me ha aportado una nueva visión a la hora de tratar los datos, adquiriendo nuevas capacidades y enfoques de la industria. Los resultados, aunque mejorables, se aproximan a tendencia de la demanda de alquiler de bicicletas mediante la aplicación de la herramienta Python en algoritmos de predicción.

Por lo que se valora haber alcanzado el objetivo fijado de valorar la programación en Python como herramienta muy útil y sencilla para la predicción y tratamiento de grandes volúmenes de datos.

En futuros trabajos se podría aplicar otros algoritmos de penalización de mínimos cuadrados que no han sido aplicados en este proyecto como son Lasso, Ridge y Elastic Net. Ridge minimiza la diferencia entre el valor real y estimado.

Además, se debería desarrollar en mayor profundidad la aplicación del algoritmo SVM, así como el análisis de parámetros para cada uno de los kernels del modelo.

Por otro lado existe el método de penalización Lasso, es una técnica de regresión regularizada con penalización, mediante la contracción de los coeficientes que podría estabilizar la predicción. A partir de cierto valor del parámetro produce estimaciones nulas para algunos coeficientes. Por ultimo, el algoritmo Elastic Net que selecciona las variables para superar añgunas de sus limitaciones, puediendo seleccionar un grupo de variables correlacionadas.

Por último, podría aplicarse el algoritmo Red Neuronal Artificial basado en aprendizaje de sistemas biológicos mediante redes complejas de neuronas interconectadas. Está metodología estructurada en capas supondría un nuevo enfoque en cuanto a los algoritmos aplicados en este proyecto. Este algoritmo es interesante, pero debido a la falta de tiempo en el periodo que se ha desarrollado el proyecto, no ha sido posible realizar dicho análisis.

7 BIBLIOGRAFÍA

Babcock, J., (August 2016), *Mastering Predictive Analytics with Python*, Packt Publishing.

Carmona E., (Julio 2014), *Tutorial sobre Máquinas de Vectores de Soporte (SVM),* Dpto. de Inteligencia Artificial, ETSII, UNED.

Chang, Chung; Lin, Jen; (March 2013), *A Library for Support Vector Machines*, National Taiwan University.

Gammerman, A.; Vavk, V.; Papadopoulo, H.; Aprill (2015), *Statistical Learning and Data Sciences*, UK Egham, Third Instarnational Symposium.

Gonzalez, M. Pilar, (1992), *Error Cuadrático Medio de Predicción para Modelos Estructurales de Series Temporales,* Estadística Española Vol. 34, Núm. 129.

Hauck, T., (November 2014), *scikit-learn Cookbook*, Packt Publishing.

Hsu, Wei; Chang, Chung; Lin, Jen; (May 2016), *A Practical Guide to Support Vector Classification*, National Taiwan University.

Kumar A., (February, 2016), *Learning Predictive Analytics with Python*, Packt Publishing.

Mckinney, W., (October 2012), *Python for Data Analysis*; O´REILLY

O'Neil, R. S. (2014). Doing Data Science. O'Reilly Media, Inc.: 1005 Gravenstein Highway North, Sebastopol, CA.

Patil A., Musale K., (April 2015), *Bike Share Demand Prediction using RandomForests*, IJISET- International journel of Innovative Science, Engineering & Technology

Peña Ros, Rosalia; (2016*), Resolución de problemas para ingenieros con Python estructurado*, Madrid, España, Garceta.

Pérez,LI., Delegido, J., Rivera,J.P., Verrelst, J., (2015), *Análisis de métodos de validación cruzada para la obtención robusta de parámetros biofísicos*, Revista de Teledetección, Universidad de Valencia.

Phuong, Czygan, M., Kumar, A., Raman, K., (March 2017), *Python: Data Analytics and Visualization*, Packt Publishing.

Smola, A.J., Schölkopf, B., (2004), *A tutorial on support vector regression*, Statistics and Computing.

Summerfield, M., (2010), *Python 3*, Madrid, España, Anaya Multimedia.

What is Data Science? (Enero, 2016), AllDataScience, dirección: http://alldatascience.com

8 ANEXO I

ANEXO I. IMPLEMENTACIÓN EN JUPYTER

I.1 - Visualización de datos

```
# coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()

# In[3]:

len(dt.datetime)

# In[4]:

dt.shape

# In[84]:

dt.groupby('weather').mean()

# In[5]:

dt.groupby('weather').sum()

# Numero de usuarios registrados:

# In[7]:

dt['registered']

# In[19]:

dt.registered.sum()
```

```
# Numero de usuarios casuales:

# In[8]:

dt['casual']

# In[20]:

dt.casual.sum()

# In[22]:

total=dt.casual.sum()+dt.registered.sum()
total

# In[5]:

dt['count'].sum()

# In[6]:

dt.groupby('season').mean()

# In[4]:

dt.groupby('season').sum()

# In[85]:

sns.distplot(dt.humidity.dropna())
sns.plt.show()

# In[73]:

sns.distplot(dt.temp.dropna())
sns.plt.show()

# In[86]:

g = sns.FacetGrid(dt, row='count', col='season')
g.map(sns.distplot, "hour")
sns.plt.show()

# In[59]:

sns.countplot(x="weather", data=dt, palette="Greens_d")
sns.plt.show()

# In[75]:

sns.jointplot(data=dt, x='count', y='registered', kind='reg', color='g')
sns.plt.show()

# In[66]:
```

```
sns.countplot(x="season", data=dt, palette="Blues_d");
sns.plt.show()

# In[65]:

sns.countplot(y="season", hue="weather", data=dt, palette="Reds_d")
sns.plt.show()

# In[67]:

sns.pointplot(x="season", y="count", hue="weather", data=dt)
sns.plt.show()

# In[4]:

dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()

# In[5]:

dt1.shape

# In[6]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
dt.head()

# Tratar variable categóricas:

# In[7]:

dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_we.head()

# In[8]:

dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_sea.head()

# In[9]:

column_name=dt.columns.values.tolist()
dt2=dt[column_name].join(dum_we)
column_name1=dt2.columns.values.tolist()
dto=dt2[column_name1].join(dum_sea)
dto.head()

# In[10]:
```

```
del dto['season']
del dto['weather']
del dto['datetime']

# In[11]:

dto.groupby('day').sum()

# In[12]:

dto.groupby('hour').sum()

# In[13]:

dto.groupby('month').sum()

# In[11]:

dto.describe()

# In[12]:

dto.boxplot(column="count")
plt.show()

# In[13]:

dto.boxplot(column="temp")
plt.show()

# In[16]:

dto.boxplot(column="hour")
plt.show()

# In[24]:

dto['count'].plot()
plt.show()

# In[25]:

dto['registered'].plot()
plt.show()

# In[30]:

dto.plot(kind='scatter',x='hour',y='count')
plt.show()

# In[34]:

dto.plot(kind='scatter',x='month',y='count')
plt.show()
```

```
# In[39]:

pd.crosstab(dto.workingday,dto.holiday).plot(kind='bar')
plt.xlabel('Días')
plt.ylabel('Frecuencia de Alquiler')
plt.show()

# In[52]:

import seaborn as sns
sns.stripplot(x="day", y="count", data=dto)
sns.plt.show()

# In[58]:

sns.violinplot(x="count", y="month", hue="day", data=dto,)
sns.plt.show()

# In[ ]:

plt.figure()
weekday_name= dt['day'].unique()
for day in weekday_name:
   counts_mean, counts_std = [], []
   dt= dt.loc[dt['day']==day]
   for i in range(24):
       dt.hour = dt.loc[dt['hour']==i]
       counts_mean.append(dt['count'].mean())
       counts_std.append(dt['count'].std())
   #plt.errorbar(range(24), counts_mean, yerr=counts_std, label=day)
   plt.plot(counts_mean, label=day)
plt.legend()
plt.xlabel('hours')
plt.ylabel('counts')
plt.show()

# In[59]:

dto.workingday.value_counts()

# In[4]:

dto.casual.values

# In[60]:

dto.holiday.value_counts()

# In[50]:

len(dto.temp.value_counts())

# In[51]:
```

```
dto.temp.value_counts()

# In[53]:

len(dto.atemp.value_counts())

# In[55]:

len(dto.humidity.value_counts())

# In[58]:

dto.windspeed.value_counts()

# Categoricas:

# In[39]:

dto.Tiempo_1.value_counts()

# In[40]:

dto.Tiempo_2.value_counts()

# In[41]:

dto.Tiempo_3.value_counts()

# In[42]:

dto.Tiempo_4.value_counts()

# In[43]:

dto.Estación_1.value_counts()

# In[44]:

dto.Estación_2.value_counts()

# In[45]:

dto.Estación_3.value_counts()

# In[46]:

dto.Estación_4.value_counts()

# In[15]:

import seaborn as sn

# In[16]:
```

```
dto.corr()

# In[17]:

corrvariables =
dto[["temp","atemp","casual","registered","humidity","windspeed","count"]].co
rr()
mask = np.array(corrvariables)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corrvariables, mask=mask,vmax=.8, square=True,annot=True)
plt.show()

# In[28]:

plt.plot(dto.registered, label='registrado', color='red', linestyle='-',
linewidth=2)
plt.legend()
plt.show()

# In[27]:

plt.plot(dto.casual, label='casual', color='blue', linestyle='-',
linewidth=2)
plt.legend()
plt.show()

# In[35]:

colors = colors_edu =('Registered':'r','Casual:'b')
    plt.scatter(dto['registered'], dto['casual'], c=dto['count'].apply(lambda
x:‹→colors[x]), s=100)

# In[77]:

get_ipython().magic('matplotlib inline')
dt['count']=dt['count'].dropna()
dt['atemp']=dt['atemp'].dropna()
plt.plot(dt['count'],dt['atemp'],'ro')
plt.xlabel('Nº de alquieler de bicis')
plt.ylabel('Sensación de calor')

# In[78]:

dt['count']=dt['count'].dropna()
dt['registered']=dt['registered'].dropna()
plt.plot(dt['registered'],dt['count'],'ro')
plt.xlabel('Usuarios registrados')
plt.ylabel('Nº de alquieler de bicis')

# In[79]:

dt['count']=dt['count'].dropna()
```

```
dt['casual']=dt['casual'].dropna()
plt.plot(dt['casual'],dt['count'],'ro')
plt.xlabel('Usuarios no registrados')
plt.ylabel('Nº de alquieler de bicis')

# In[80]:

dt['atemp']=dt['atemp'].dropna()
dt['casual']=dt['casual'].dropna()
plt.plot(dt['casual'],dt['atemp'],'ro')
plt.xlabel('Usuarios no registrados')
plt.ylabel('Sensación térmica')

# In[42]:

df=dto[dto.Estación_1==1][dto.Tiempo_1==1]
df
```

I.2 - Regresión lineal múltiple.

```
# coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Tratamiento de datos:

# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()

# In[3]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
#dt['numberweek']=dt.datetime.apply(lambda x: (x.day-1)//7+1)
dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt['day'],prefix='dayweek')
column=dt.columns.values.tolist()
dt2=dt[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()

# In[4]:

del dtt['season']
del dtt['weather']
del dtt['day']
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1',
'Tiempo_2','Tiempo_3', 'Tiempo_4', 'Estación_1',
   'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a

# In[5]:

import math
from math import log
#A function to calculate Root Mean Squared Logarithmic Error (RMSLE)
def rmsle(y, y_pred):
   assert len(y) == len(y_pred)
```

```
   y=y.values
   y_pred[y_pred<0]=0
   terms_to_sum = [(log(y_pred[i] + 1) - log(y[i] + 1)) ** 2.0 for i,pred in
enumerate(y_pred)]
   return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5

# In[6]:

from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
train, test = train_test_split(dtt, test_size = 0.2)

# In[7]:

train_X=train[cols]
train_Ycasual=train['casual']
train_Yreg=train['registered']
train_Ycount=train['count']
test_X=test[cols]
test_Ycasual=test['casual']
test_Yreg=test['registered']
test_Ycount=test['count']

# ## Entrenamos los distintos conjuntos de entrenamiento:

# In[8]:

reg = linear_model.LinearRegression()
reg.fit(train_X,train_Ycasual)
reg1 = linear_model.LinearRegression()
reg1.fit(train_X,train_Yreg)
reg2 = linear_model.LinearRegression()
reg2.fit(train_X,train_Ycount)
reg3= linear_model.LinearRegression()
reg3.fit(train_X,train_Ycasual+train_Yreg)

# ## Predicciones:

# In[9]:

predict_ycasual = reg.predict(test_X)
predict_ycasual

# In[10]:

predict_yreg = reg1.predict(test_X)
predict_yreg

# In[11]:

predict_ycount = reg2.predict(test_X)
predict_ycount
```

```
# In[12]:

predict_ytotal = reg3.predict(test_X)
predict_ytotal

# Cálculo del coeficiente de determinación:

# In[13]:

reg.score(train_X,train_Ycasual)

# In[14]:

reg1.score(train_X,train_Yreg)

# In[15]:

reg2.score(train_X,train_Ycount)

# In[16]:

reg3.score(train_X,train_Ycasual+train_Yreg)

# Cálculo:

# In[17]:

r2_score(test_Ycasual,predict_ycasual)

# In[18]:

r2_score(test_Yreg,predict_yreg)

# In[19]:

r2_score(test_Ycount,predict_ycount)

# In[20]:

r2_score(test_Ycasual+test_Yreg,predict_ycasual+predict_yreg)

# Cálculo del error cuadratico medio:

# In[21]:

mean_squared_error(test_Ycasual,predict_ycasual)

# In[22]:

mean_squared_error(test_Yreg,predict_yreg)

# In[23]:

mean_squared_error(test_Ycount,predict_ycount)
```

```
# In[24]:

mean_squared_error(test_Ycasual+test_Yreg,predict_ytotal)

# RSMLE:

# Anulamos los valores negativos de los resultados:

# In[25]:

predict_ycasual[predict_ycasual<0]=0
predict_yreg[predict_yreg<0]=0
predict_ycount[predict_ycount<0]=0

# In[26]:

rmsle(test_Ycasual,predict_ycasual)


# In[27]:

rmsle(test_Yreg,predict_yreg)

# In[28]:

rmsle(test_Ycount,predict_ycount)

# In[80]:

rmsle(test_Ycount,predict_ycasual+predict_yreg)

# ## Intentamos mejorar puntuacion:

# In[39]:

from sklearn.preprocessing import scale

# In[40]:

norm=linear_model.LinearRegression(normalize=scale)
norm.fit(train_X,train_Ycasual)

# In[41]:

norm1=linear_model.LinearRegression(normalize=scale)
norm1.fit(train_X,train_Yreg)

# In[42]:

norm2=linear_model.LinearRegression(normalize=scale)
norm2.fit(train_X,train_Ycount)

# In[43]:
```

```
norm_pred=norm.predict(test_X)
# In[44]:
norm1_pred=norm1.predict(test_X)
# In[45]:
norm2_pred=norm2.predict(test_X)
# In[46]:
norm.score(train_X,train_Ycasual)
# In[47]:
norm1.score(train_X,train_Yreg)
# In[48]:
norm2.score(train_X,train_Ycount)
# In[81]:
norm2.score(train_X,train_Yreg+train_Ycasual)
# In[49]:
r2_score(test_Ycasual, norm_pred)
# In[50]:
r2_score(test_Yreg, norm1_pred)

# In[51]:
r2_score(test_Ycount, norm2_pred)
# In[52]:
r2_score(test_Ycount, norm_pred+ norm1_pred)
# In[82]:
mean_squared_error(test_Ycasual, norm_pred)
# In[83]:
mean_squared_error(test_Yreg, norm1_pred)
# In[84]:
```

```
mean_squared_error(test_Ycount, norm2_pred)

# In[85]:

mean_squared_error(test_Ycount, norm_pred+norm1_pred)

# In[53]:

rmsle(test_Ycasual, norm_pred)

# In[54]:

rmsle(test_Yreg, norm1_pred)

# In[55]:

rmsle(test_Ycount, norm2_pred)

# In[56]:

rmsle(test_Ycount, norm_pred+ norm1_pred)

# ## Validación cruzada:

# #### Para los usuarios casuales:

# In[64]:

from sklearn.metrics import r2_score,make_scorer
from sklearn.cross_validation import cross_val_score
a = make_scorer(rmsle)

# In[65]:

reg_cv = linear_model.LinearRegression()

# In[66]:

scores_cv = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['casual'],
scoring='mean_squared_error', cv=10,n_jobs=2))
scores_cv

# In[67]:

score1_cv = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['casual'],
scoring='r2',cv=10,n_jobs=2))
score1_cv

# Para RMSLE:

# In[68]:

from sklearn.metrics import make_scorer
```

```
# In[69]:

ms = make_scorer(rmsle)

# In[70]:

score1_cv = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['casual'],
scoring=ms, cv=10,n_jobs=2))
score1_cv

# Para los usuarios registrados:

# In[71]:

scoresr = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['registered'],
scoring='mean_squared_error',cv=10,n_jobs=2))
scoresr

# In[72]:

scoresr1=np.mean(cross_val_score(reg_cv, dtt[cols], dtt['registered'],
scoring='r2', cv=10,n_jobs=2))
scoresr1

# In[73]:

scoresr2 = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['registered'],
scoring=ms,cv=10,n_jobs=2))
scoresr2

# Para el total de usuarios:

# In[74]:

from sklearn.cross_validation import cross_val_score
scorec = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['count'],
scoring='mean_squared_error', cv=10,n_jobs=2))
scorec

# In[75]:

scorec1 = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['count'],
scoring='r2', cv=10,n_jobs=2))
scorec1

# In[76]:

scorec2 = np.mean(cross_val_score(reg_cv, dtt[cols], dtt['count'],
scoring=ms,cv=10,n_jobs=2))
scorec2

# ### Cross validation para los conjuntos de datos normalizados:

# Usuarios casuales:
```

```
# In[87]:

scoren_cv = np.mean(cross_val_score(norm, dtt[cols], dtt['casual'],
scoring='r2', cv=10,n_jobs=2))
scoren_cv

# In[88]:

scoren1_cv = np.mean(cross_val_score(norm, dtt[cols], dtt['casual'],
scoring='mean_squared_error', cv=10,n_jobs=2))
scoren1_cv

# In[89]:

scoren2_cv = np.mean(cross_val_score(norm, dtt[cols], dtt['casual'],
scoring=ms, cv=10,n_jobs=2))
scoren2_cv

# Usuarios registrados:

# In[90]:

scorenr_cv = np.mean(cross_val_score(norm1, dtt[cols], dtt['registered'],
scoring='r2', cv=10,n_jobs=2))
scorenr_cv

# In[91]:

scorenr1_cv = np.mean(cross_val_score(norm1, dtt[cols], dtt['registered'],
scoring='mean_squared_error', cv=10,n_jobs=2))
scorenr1_cv

# In[92]:

scorenr2_cv = np.mean(cross_val_score(norm1, dtt[cols], dtt['registered'],
scoring=ms, cv=10,n_jobs=2))
scorenr2_cv

# Usuarios totales:

# In[93]:

scorec1_cv = np.mean(cross_val_score(norm2, dtt[cols], dtt['count'],
scoring='r2', cv=10,n_jobs=2))
scorec1_cv

# In[94]:

scorec_cv = np.mean(cross_val_score(norm2, dtt[cols], dtt['count'],
scoring='mean_squared_error', cv=10,n_jobs=2))
scorec_cv

# In[95]:
```

```
scorec2_cv = np.mean(cross_val_score(norm2, dtt[cols], dtt['count'],
scoring=ms, cv=10,n_jobs=2))
scorec2_cv

# Suma:

# In[96]:

scorec1_cv = np.mean(cross_val_score(norm2, dtt[cols],
dtt['casual']+dtt['registered'], scoring='r2', cv=10,n_jobs=2))
scorec1_cv

# In[97]:

scorec124_cv = np.mean(cross_val_score(norm2, dtt[cols],
dtt['casual']+dtt['registered'], scoring='mean_squared_error',
cv=10,n_jobs=2))
scorec124_cv

# ## Predicción-test:

# In[29]:

dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()

# In[30]:

dt1.datetime = dt1.datetime.apply(pd.to_datetime)
dt1['year']= dt1.datetime.apply(lambda x : x.year)
dt1['month'] = dt1.datetime.apply(lambda x : x.month)
dt1['hour'] = dt1.datetime.apply(lambda x : x.hour)
dt1['day'] = dt1.datetime.apply(lambda x: x.weekday_name)
dt1.head()

# In[31]:

dum_we=pd.get_dummies(dt1['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt1['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt1['day'],prefix='dayweek')

# In[32]:

column=dt1.columns.values.tolist()
dt2=dt1[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtest=dto[column2].join(dum_dia)
dtest.head()

# In[33]:
```

```
del dtest['season']
del dtest['weather']
del dtest['day']

# In[98]:

dtest.head()

# In[99]:

dtest.shape

# In[34]:

X_test = dtest[cols]

# In[57]:

y_test = reg2.predict(X_test)

# In[59]:

submission_0 =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_test},columns=['date
time','count'])
submission_0.loc[submission_0['count']<0.0,'count']=0.0

# Lo convierto en fichero:

# In[60]:

submission_0.to_csv('/Users/Miguel/Desktop/Submissions/sub_0.csv',index=False
)

# ### Normalizado:

# In[61]:

y_testN = norm2.predict(X_test)

# In[62]:

submission_0N =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testN},columns=['dat
etime','count'])
submission_0N.loc[submission_0N['count']<0.0,'count']=0.0

# In[63]:

submission_0N.to_csv('/Users/Miguel/Desktop/Submissions/sub_0N.csv',index=Fal
se)

# ### CV:
```

```
# In[79]:

y_testCV = scorec2.predict(X_test)
submission_0CV =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testCV},columns=['da
tetime','count'])
submission_0CV.loc[submission_0CV['count']<0.0,'count']=0.0
submission_0CV.to_csv('/Users/Miguel/Desktop/Submissions/sub_0CV.csv',index=F
alse)
```

I.3 - Regresión de Poisson.

```
# coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# ## Train:

# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()

# In[3]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt['day'],prefix='dayweek')
column=dt.columns.values.tolist()
dt2=dt[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()

# In[4]:

del dtt['season']
del dtt['weather']
del dtt['day']
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1',
'Tiempo_2','Tiempo_3', 'Tiempo_4', 'Estación_1',
   'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a

# RMSLE:

# In[5]:

import math
from math import log
#A function to calculate Root Mean Squared Logarithmic Error (RMSLE)
def rmsle(y, y_pred):
```

```
   assert len(y) == len(y_pred)
   terms_to_sum = [(math.log(y_pred[i] + 1) - math.log(y[i] + 1)) ** 2.0 for
i,pred in enumerate(y_pred)]
   return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5

# # Dataset casual:

# Partición de train:

# In[6]:

from sklearn.cross_validation import train_test_split
train, test = train_test_split(dtt, test_size = 0.2)

# In[7]:

train_X=train[cols]
train_Ycasual=train['casual']
train_Yreg=train['registered']
train_Ycount=train['count']
test_X=test[cols]
test_Ycasual=test['casual']
test_Yreg=test['registered']
test_Ycount=test['count']

# In[8]:

train_X.columns

# In[9]:

a = " + ".join(cols)
form='casual ~ '+a
form1='registered ~ '+a
form2='count ~ '+a

# In[10]:

from statsmodels.genmod.families import Poisson
import statsmodels.api as sm
import statsmodels.formula.api as smf

# In[11]:

mod = smf.glm(formula=form, data=dtt, family=sm.families.Poisson()).fit()
mod.pvalues

# In[12]:

mod1 = smf.glm(formula=form1, data=dtt, family=sm.families.Poisson()).fit()
mod1.pvalues

# In[13]:
```

```
mod2= smf.glm(formula=form2, data=dtt, family=sm.families.Poisson()).fit()
mod2.pvalues

# In[14]:

po=mod.predict(test_X)

# In[15]:

po1=mod1.predict(test_X)

# In[16]:

po2=mod2.predict(test_X)

# Cálculo del coeficiente de determinación:

# In[17]:

from sklearn.metrics import r2_score

# In[18]:

r2_score(test_Ycasual.values,po.values)

# In[19]:

r2_score(test_Yreg.values,po1.values)

# In[20]:

r2_score(test_Ycount.values,po2.values)


# Cálculo del error cuadratico medio:

# In[21]:

from sklearn.metrics import mean_squared_error

# In[22]:

mean_squared_error(test_Ycasual.values,po.values)

# In[23]:

mean_squared_error(test_Yreg.values,po1.values)

# In[24]:

mean_squared_error(test_Ycount.values,po2.values)

# Cálculo del RMSLE:
```

```
# In[25]:

rmsle(test_Ycasual.values,po.values)

# In[26]:

rmsle(test_Yreg.values,po1.values)

# In[27]:

rmsle(test_Ycount.values,po2.values)

# ## Predicción.test:

# In[28]:

dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()

# In[29]:

dt1.datetime = dt1.datetime.apply(pd.to_datetime)
dt1['year']= dt1.datetime.apply(lambda x : x.year)
dt1['month'] = dt1.datetime.apply(lambda x : x.month)
dt1['hour'] = dt1.datetime.apply(lambda x : x.hour)
dt1['day'] = dt1.datetime.apply(lambda x: x.weekday_name)
dt1.head()

# In[30]:

dum_we=pd.get_dummies(dt1['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt1['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt1['day'],prefix='dayweek')

# In[31]:

column=dt1.columns.values.tolist()
dt2=dt1[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtest=dto[column2].join(dum_dia)
dtest.head()

# In[32]:

del dtest['season']
del dtest['weather']
del dtest['day']

# In[33]:

X_test = dtest[cols]
```

```
# In[34]:

y_poi_test = mod2.predict(X_test)

# In[35]:

submission_poi_1 =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_poi_test},columns=['
datetime','count'])

# Lo convierto en fichero:

# In[36]:

submission_poi_1.to_csv('/Users/Miguel/Desktop/Submissions/sub_poi_1.csv',ind
ex=False)
```

I.4 Árbol de decisión.

```
# coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()

# In[3]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt['day'],prefix='dayweek')
column=dt.columns.values.tolist()
dt2=dt[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()

# In[4]:

del dtt['season']
del dtt['weather']
del dtt['day']
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1',
'Tiempo_2','Tiempo_3', 'Tiempo_4', 'Estación_1',
   'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a

# In[5]:

import math
from math import log
#A function to calculate Root Mean Squared Logarithmic Error (RMSLE)
def rmsle(y, y_pred):
   y=y.values
   y_pred[y_pred<0]=0
```

```
   assert len(y) == len(y_pred)
   terms_to_sum = [(math.log(y_pred[i] + 1) - math.log(y[i] + 1)) ** 2.0 for
i,pred in enumerate(y_pred)]
   return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5

# In[6]:

from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

# In[7]:

from sklearn.cross_validation import train_test_split
train, test = train_test_split(dtt, test_size = 0.2)

# In[8]:

train_X=train[cols]
train_Ycasual=train['casual']
train_Yreg=train['registered']
train_Ycount=train['count']
test_X=test[cols]
test_Ycasual=test['casual']
test_Yreg=test['registered']
test_Ycount=test['count']

# In[9]:

from sklearn.tree import DecisionTreeRegressor

# In[10]:

casual_tree =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)
casual_tree.fit(train_X,train_Ycasual)

# In[11]:

reg_tree =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)
reg_tree.fit(train_X,train_Yreg)

# In[12]:

count_tree =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)
count_tree.fit(train_X,train_Ycount)

# In[13]:

casual_tree_pred=casual_tree.predict(test_X)
```

```
reg_tree_pred=reg_tree.predict(test_X)
count_tree_pred=count_tree.predict(test_X)

# Score:

# In[14]:

casual_tree.score(test_X,test_Ycasual)

# In[15]:

reg_tree.score(test_X,test_Yreg)

# In[16]:

count_tree.score(test_X,test_Ycount)

# In[17]:

count_tree.score(test_X,test_Ycasual+test_Yreg)

# In[18]:

r2_score(test_Ycasual,casual_tree_pred)

# In[19]:

r2_score(test_Yreg,reg_tree_pred)

# In[20]:

r2_score(test_Ycount,count_tree_pred)

# In[21]:

r2_score(test_Ycount,casual_tree_pred+reg_tree_pred)

# MSE:

# In[22]:

mean_squared_error(test_Ycasual,casual_tree_pred)

# In[23]:

mean_squared_error(test_Yreg,reg_tree_pred)

# In[24]:

mean_squared_error(test_Ycount,count_tree_pred)

# In[25]:

mean_squared_error(test_Ycount,casual_tree_pred+reg_tree_pred)
```

```
# RMSLE:

# In[26]:

rmsle(test_Ycasual,casual_tree_pred)

# In[27]:

rmsle(test_Yreg,reg_tree_pred)

# In[28]:

rmsle(test_Ycount,count_tree_pred)

# In[29]:

rmsle(test_Ycount,casual_tree_pred+reg_tree_pred)

# ## Featura importance:

# Cassual:

# In[33]:

peso=casual_tree.feature_importances_

# In[34]:

dic={'importancia':peso,'variable':train_X.columns}
wr=pd.DataFrame(dic,columns=['variable','importancia'])

# In[35]:

wir=wr.sort_values(by='importancia',ascending=False)
wir

# In[36]:

col=wir.iloc[0:10].variable.values

# In[45]:

train_Xfttcasual=train[cols]
train_Yfttcasual=train['casual']
test_Xfttcasual=test[cols]
test_Yfttcasual=test['casual']

# In[38]:

freecasual =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)
freecasual.fit(train_Xfttcasual,train_Yfttcasual)
```

```
fcapred=freecasual.predict(test_Xfttcasual)

# In[39]:

freecasual.score(train_Xfttcasual,train_Yfttcasual)

# In[40]:

r2_score(test_Yfttcasual,fcapred)

# In[41]:

mean_squared_error(test_Yfttcasual,fcapred)

# In[43]:

rmsle(test_Yfttcasual,fcapred)

# Registrados:

# In[44]:

peso1=reg_tree.feature_importances_
dic1={'importancia':peso1,'variable':train_X.columns}
wr1=pd.DataFrame(dic1,columns=['variable','importancia'])
wir1=wr1.sort_values(by='importancia',ascending=False)

# In[45]:

colm=wir.iloc[0:10].variable.values
train_Xfttreg=train[colm]
train_Yfttreg=train['registered']
test_Xfttreg=test[colm]
test_Yfttreg=test['registered']

# In[46]:

freereg =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)
freereg.fit(train_Xfttreg,train_Yfttreg)
frepred=freereg.predict(test_Xfttreg)

# In[47]:

freereg.score(train_Xfttreg,train_Yfttreg)

# In[48]:

r2_score(test_Yfttreg,frepred)

# In[49]:

mean_squared_error(test_Yfttreg,frepred)
```

```
# In[51]:

rmsle(test_Yfttreg,frepred)

# Count:

# In[46]:

peso2=count_tree.feature_importances_
dic2={'importancia':peso2,'variable':train_X.columns}
wr2=pd.DataFrame(dic2,columns=['variable','importancia'])
wir2=wr2.sort_values(by='importancia',ascending=False)

# In[47]:

colc=wir2.iloc[0:10].variable.values
train_Xfttcount=train[colc]
train_Yfttcount=train['count']
test_Xfttcount=test[colc]
test_Yfttcount=test['count']

# In[48]:

freecou =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)
freecou.fit(train_Xfttcount,train_Yfttcount)
fcopred=freecou.predict(test_Xfttcount)

# In[49]:

freecou.score(train_Xfttcount,train_Yfttcount)

# In[50]:

r2_score(test_Yfttcount,fcopred)

# In[51]:

mean_squared_error(test_Yfttcount,fcopred)

# In[52]:

rmsle(test_Yfttcount,fcopred)

# Uniendo modelo casual y registrado:

# In[60]:

freecou.score(train_Xfttcount,train_Yfttcasual+train_Yfttreg)

# In[61]:
```

```
r2_score(test_Yfttcount,fcapred+ frepred)

# In[62]:

mean_squared_error(test_Yfttcount,fcapred+ frepred)

# In[64]:

rmsle(test_Yfttcount,fcapred+ frepred)

# In[59]:

dt_aux=pd.DataFrame({"Y_obs":test_Ycount,"Y_pred":reg_tree_pred})

dt_aux.plot(kind='scatter',y='Y_obs',x='Y_pred')
plt.show()

# ## Cross Validation:

# In[95]:

from sklearn.cross_validation import cross_val_score

# Usuarios totales:

# In[96]:

score = np.mean(cross_val_score(count_tree, train_X,
train_Ycount,scoring='mean_squared_error', cv=10,n_jobs=2))
score

# In[97]:

score1 = np.mean(cross_val_score(count_tree, train_X,
train_Ycount,scoring='r2', cv=10,n_jobs=2))
score1

# In[98]:

from sklearn.metrics import make_scorer
ms = make_scorer(rmsle)

# In[99]:

score2 = np.mean(cross_val_score(count_tree, train_X,
train_Ycount,scoring=ms, cv=10,n_jobs=2))
score2

# Usuarios registrados:

# In[100]:

scoreg = np.mean(cross_val_score(reg_tree, train_X,
train_Yreg,scoring='mean_squared_error', cv=10,n_jobs=2))
```

```
scoreg

# In[101]:

scoreg1 = np.mean(cross_val_score(reg_tree, train_X, train_Yreg,scoring='r2',
cv=10,n_jobs=2))
scoreg1

# In[102]:

scoreg2 = np.mean(cross_val_score(reg_tree, train_X, train_Yreg,scoring=ms,
cv=10,n_jobs=2))
scoreg2

# Usuarios casuales:

# In[103]:

scorecas = np.mean(cross_val_score(casual_tree, train_X,
train_Ycasual,scoring='mean_squared_error', cv=10,n_jobs=2))
scorecas

# In[104]:

scorecas1 = np.mean(cross_val_score(casual_tree, train_X,
train_Ycasual,scoring='r2', cv=10,n_jobs=2))
scorecas1

# In[105]:

scorecas2 = np.mean(cross_val_score(casual_tree, train_X,
train_Ycasual,scoring=ms, cv=10,n_jobs=2))
scorecas2

# Suma de usuarios parciales:

# In[106]:

scoretot = np.mean(cross_val_score(count_tree, train_X,
train_Ycasual+train_Yreg,scoring='mean_squared_error', cv=10,n_jobs=2))
scoretot

# In[107]:

scoretot1 = np.mean(cross_val_score(count_tree, train_X,
train_Ycasual+train_Yreg,scoring='r2', cv=10,n_jobs=2))
scoretot1

# In[108]:

scoretot2 = np.mean(cross_val_score(count_tree, train_X,
train_Ycasual+train_Yreg,scoring=ms, cv=10,n_jobs=2))
scoretot2
```

```
# ### Validación cruzada con feature importance:

# In[ ]:

train_Xfttcount=train[colc]
train_Yfttcount=train['count']
test_Xfttcount=test[colc]
test_Yfttcount=test['count']

# In[114]:

scoreftcount= np.mean(cross_val_score(freecou, train_Xfttcount,
train_Yfttcount,scoring='mean_squared_error', cv=10,n_jobs=2))
scoreftcount

# In[115]:

scoreftcount1= np.mean(cross_val_score(freecou, train_Xfttcount,
train_Yfttcount,scoring='r2', cv=10,n_jobs=2))
scoreftcount1

# In[116]:

scoreftcount2= np.mean(cross_val_score(freecou, train_Xfttcount,
train_Yfttcount,scoring=ms, cv=10,n_jobs=2))
scoreftcount2

# Usuarios registrados:

# In[ ]:

train_Xfttreg=train[colm]
train_Yfttreg=train['registered']
test_Xfttreg=test[colm]
test_Yfttreg=test['registered']

# In[117]:

scoreftreg= np.mean(cross_val_score(freereg, train_Xfttreg,
train_Yfttreg,scoring='mean_squared_error', cv=10,n_jobs=2))
scoreftreg

# In[118]:

scoreftreg1= np.mean(cross_val_score(freereg, train_Xfttreg,
train_Yfttreg,scoring='r2', cv=10,n_jobs=2))
scoreftreg1

# In[119]:

scoreftreg2= np.mean(cross_val_score(freereg, train_Xfttreg,
train_Yfttreg,scoring=ms, cv=10,n_jobs=2))
scoreftreg2
```

```
# Usuarios casuales:

# In[ ]:

train_Xfttcasual=train[cols]
train_Yfttcasual=train['casual']
test_Xfttcasual=test[cols]
test_Yfttcasual=test['casual']

# In[110]:

scoreftcasual = np.mean(cross_val_score(freecasual, train_Xfttcasual,
train_Yfttcasual,scoring='mean_squared_error', cv=10,n_jobs=2))
scoreftcasual

# In[111]:

scoreftcasual1 = np.mean(cross_val_score(freecasual, train_Xfttcasual,
train_Yfttcasual,scoring='r2', cv=10,n_jobs=2))
scoreftcasual1

# In[112]:

scoreftcasual2 = np.mean(cross_val_score(freecasual, train_Xfttcasual,
train_Yfttcasual,scoring=ms, cv=10,n_jobs=2))
scoreftcasual2

# In[ ]:

# ## GRIDSEARCH:

# In[62]:

from sklearn.grid_search import GridSearchCV

# In[63]:

param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "min_samples_split": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10]}

# In[57]:

gridfcou =
DecisionTreeRegressor(min_samples_split=30,min_samples_leaf=10,random_state=0
)

# In[ ]:

gridfcou.fit()

# Usuarios totales:
```

```
# In[68]:

grid_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Ycount)

# In[69]:

grid_search.best_score_

# In[60]:

from sklearn.metrics import make_scorer
ms = make_scorer(rmsle)

# In[61]:

grid_search2 = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Ycount)

# In[71]:

grid_search2.best_score_

# In[64]:

grid_search3 = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Ycount)

# In[65]:

grid_search3.best_score_

# Usuarios registrados:

# In[77]:

gridreg_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Yreg)

# In[79]:

gridreg_search.best_score_

# In[78]:

gridreg1_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Yreg)

# In[80]:

gridreg1_search.best_score_
```

```
# In[81]:

gridreg2_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Yreg)

# In[82]:

gridreg2_search.best_score_

# Usuarios casuales:

# In[83]:

gridcas_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Ycasual)

# In[84]:

gridcas_search.best_score_

# In[85]:

gridcas1_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Ycasual)

# In[86]:

gridcas1_search.best_score_

# In[87]:

gridcas2_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Ycasual)

# In[88]:

gridcas2_search.best_score_

# Usuarios registrados+Usuarios casuales:

# In[89]:

gridtot_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Ycasual+train_Yreg)

# In[90]:

gridtot_search.best_score_

# In[91]:
```

```
gridtot1_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Ycasual+train_Yreg)

# In[92]:

gridtot1_search.best_score_

# In[68]:

gritot2_search = GridSearchCV(gridfcou, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Ycasual+train_Yreg)

# In[94]:

gritot2_search.best_score_

# ## Predicción-Test:

# In[30]:

dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()

# In[31]:

dt1.datetime = dt1.datetime.apply(pd.to_datetime)
dt1['year']= dt1.datetime.apply(lambda x : x.year)
dt1['month'] = dt1.datetime.apply(lambda x : x.month)
dt1['hour'] = dt1.datetime.apply(lambda x : x.hour)
dt1['day'] = dt1.datetime.apply(lambda x: x.weekday_name)
dt1.head()

# In[32]:

dum_we=pd.get_dummies(dt1['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt1['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt1['day'],prefix='dayweek')

# In[33]:

column=dt1.columns.values.tolist()
dt2=dt1[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtest=dto[column2].join(dum_dia)
dtest.head()

# In[34]:

del dtest['season']
del dtest['weather']
del dtest['day']
```

```
# In[38]:

X_test = dtest[cols]
y_testad = count_tree.predict(X_test)

# In[39]:

submission_ad =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testad},columns=['da
tetime','count'])

# In[40]:

submission_ad.loc[submission_ad['count']<0.0,'count']=0.0

# In[41]:

submission_ad.to_csv('/Users/Miguel/Desktop/Submissions/sub_ad.csv',index=Fal
se)

# ### FI:

# In[53]:

y_testadfi = freecou.predict(X_test)

# In[ ]:

submission_adfi =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testadfi},columns=['
datetime','count'])
submission_adfi.loc[submission_ad['count']<0.0,'count']=0.0
submission_adfi.to_csv('/Users/Miguel/Desktop/Submissions/sub_adfi.csv',index
=False)

# ### GS:

# In[66]:

y_testadgi =grid_search3.predict(X_test)

# In[67]:

submission_adgi =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testadgi},columns=['
datetime','count'])
submission_adgi.loc[submission_adgi['count']<0.0,'count']=0.0
submission_adgi.to_csv('/Users/Miguel/Desktop/Submissions/sub_ad.csv',index=F
alse)

# Suma:

# In[69]:
```

```
y_testsumatorio=gritot2_search.predict(X_test)
submission_adsumatorio =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testsumatorio},colum
ns=['datetime','count'])
submission_adsumatorio.loc[submission_adgi['count']<0.0,'count']=0.0
submission_adsumatorio.to_csv('/Users/Miguel/Desktop/Submissions/sub_adsumato
rio.csv',index=False)
```

I.5. Random Forest.

```
# coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()

# In[3]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt['day'],prefix='dayweek')
column=dt.columns.values.tolist()
dt2=dt[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()

# In[4]:

del dtt['season']
del dtt['weather']
del dtt['day']
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1',
'Tiempo_2','Tiempo_3', 'Tiempo_4', 'Estación_1',
   'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a

# In[5]:

import math
from math import log
#A function to calculate Root Mean Squared Logarithmic Error (RMSLE)
def rmsle(y, y_pred):
   y=y.values
   y_pred[y_pred<0]=0
   assert len(y) == len(y_pred)
```

```
   terms_to_sum = [(math.log(y_pred[i] + 1) - math.log(y[i] + 1)) ** 2.0 for
i,pred in enumerate(y_pred)]
   return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5

# In[6]:

from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.cross_validation import train_test_split
train, test = train_test_split(dtt, test_size = 0.2)

# In[7]:

train_X=train[cols]
train_Ycasual=train['casual']
train_Yreg=train['registered']
train_Ycount=train['count']
test_X=test[cols]
test_Ycasual=test['casual']
test_Yreg=test['registered']
test_Ycount=test['count']

# In[8]:

rfcasual= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
rfcasual.fit(train_X,train_Ycasual)

# In[9]:

rfreg= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
rfreg.fit(train_X,train_Yreg)

# In[10]:

rfcount= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
rfcount.fit(train_X,train_Ycount)

# In[11]:

casualpr=rfcasual.predict(test_X)
regpr=rfreg.predict(test_X)
countpr=rfcount.predict(test_X)

# Coeficiente de determinación:

# In[12]:

rfcasual.score(test_X,test_Ycasual)

# In[13]:

rfreg.score(test_X,test_Yreg)
```

```
# In[14]:

rfcount.score(test_X,test_Ycount)


# In[15]:

rfcount.score(test_X,test_Ycasual+test_Yreg)

# In[16]:

r2_score(test_Ycasual,casualpr)

# In[17]:

r2_score(test_Yreg,regpr)

# In[18]:

r2_score(test_Ycount,countpr)

# In[19]:

r2_score(test_Ycount,casualpr+regpr)

# MSE:

# In[20]:

mean_squared_error(test_Ycasual,casualpr)

# In[21]:

mean_squared_error(test_Yreg,regpr)

# In[22]:

mean_squared_error(test_Ycount,countpr)

# In[23]:

mean_squared_error(test_Ycount,casualpr+regpr)

# RMSLE:

# In[26]:

rmsle(test_Ycasual,casualpr)

# In[27]:

rmsle(test_Yreg,regpr)

# In[28]:
```

```
rmsle(test_Ycount,countpr)

# In[29]:

rmsle(test_Ycount,casualpr+regpr)
# ## ------------------------------------------------------------------------
---------------------------------

# ## Feature importance:

# Modelo casual:

# In[30]:

importcasual=rfcasual.feature_importances_
importcasual

# In[31]:

importcasual
train_X.columns

# In[32]:

d={'importancia':importcasual,'variable':train_X.columns}
fti=pd.DataFrame(d,columns=['variable','importancia'])

# In[33]:

mfti=fti.sort_values(by='importancia',ascending=False)
mfti

# In[34]:

mfti.iloc[0:10].variable.values

# In[35]:

colu=['hour','temp','workingday','humidity','windspeed','month','dayweek_Frid
ay','dayweek_Saturday','Tiempo_1','Estación_2']

# In[36]:

train_Xftcasual=train[colu]
train_Yftcasual=train['casual']
test_Xftcasual=test[colu]
test_Yftcasual=test['casual']

# In[37]:

ftcasual= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
ftcasual.fit(train_Xftcasual,train_Yftcasual)
```

```
# In[38]:

ftpc=ftcasual.predict(test_Xftcasual)
ftpc

# In[39]:

ftcasual.score(train_Xftcasual,train_Yftcasual)

# In[40]:

r2_score(test_Yftcasual,ftpc)

# In[41]:

mean_squared_error(test_Yftcasual,ftpc)

# In[43]:

rmsle(test_Yftcasual,ftpc)

# Modelo registrado:

# In[44]:

importregist=rfreg.feature_importances_
importregist

# In[45]:

d1={'importancia':importregist,'variable':train_X.columns}
ftir=pd.DataFrame(d1,columns=['variable','importancia'])

# In[46]:

mftir=ftir.sort_values(by='importancia',ascending=False)
mftir

# In[47]:

colu2=mftir.iloc[0:10].variable.values

# In[48]:

train_Xftreg=train[colu2]
train_Yftreg=train['registered']
test_Xftreg=test[colu2]
test_Yftreg=test['registered']

# In[49]:

ftreg= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
ftreg.fit(train_Xftreg,train_Yftreg)
ftpr=ftreg.predict(test_Xftreg)
```

```
ftpr

# In[50]:

ftreg.score(train_Xftreg,train_Yftreg)

# In[51]:

r2_score(test_Yreg,ftpr)

# In[52]:

mean_squared_error(test_Yreg,ftpr)

# In[53]:

rmsle(test_Yreg,ftpr)

# Modelo completo con count:

# In[54]:

importcount=rfcount.feature_importances_
importcount

# In[55]:

d2={'importancia':importcount,'variable':train_X.columns}
ftic=pd.DataFrame(d2,columns=['variable','importancia'])
mftic=ftic.sort_values(by='importancia',ascending=False)
mftic

# In[56]:

colu3=mftic.iloc[0:10].variable.values

# In[57]:

train_Xftcount=train[colu3]
train_Yftcount=train['count']
test_Xftcount=test[colu3]
test_Yftcount=test['count']

# In[58]:

ftcount= RandomForestRegressor(n_jobs=2,oob_score=True,n_estimators=500)
ftcount.fit(train_Xftcount,train_Yftcount)
ftpco=ftcount.predict(test_Xftcount)
ftpco

# In[59]:

ftcount.score(train_Xftcount,train_Yftcount)
```

```
# In[60]:

r2_score(test_Yftcount,ftpco)

# In[61]:

mean_squared_error(test_Yftcount,ftpco)

# In[62]:

rmsle(test_Yftcount,ftpco)

# Unión registrado y casual:

# In[63]:

ftcount.score(train_Xftcount,train_Yftcasual+train_Yftreg)

# In[64]:

r2_score(test_Yftcount.values,ftpc+ftpr)

# In[65]:

mean_squared_error(test_Yftcount.values,ftpc+ftpr)

# In[66]:

rmsle(test_Yftcount,ftpc+ftpr)

# ## Cross Validation:

# In[99]:

from sklearn.cross_validation import cross_val_score

# Usuarios totales:

# In[100]:

score = np.mean(cross_val_score(rfcount, train_X,
train_Ycount,scoring='mean_squared_error', cv=10,n_jobs=2))
score

# In[101]:

score1 = np.mean(cross_val_score(rfcount, train_X, train_Ycount,scoring='r2',
cv=10,n_jobs=2))
score1

# In[102]:

from sklearn.metrics import make_scorer
ms = make_scorer(rmsle)
```

```
# In[103]:

score2 = np.mean(cross_val_score(rfcount, train_X, train_Ycount,scoring=ms,
cv=10,n_jobs=2))
score2

# Usuarios registrados:

# In[104]:

scoreg = np.mean(cross_val_score(rfreg, train_X,
train_Yreg,scoring='mean_squared_error', cv=10,n_jobs=2))
scoreg

# In[105]:

scoreg1 = np.mean(cross_val_score(rfreg, train_X, train_Yreg,scoring='r2',
cv=10,n_jobs=2))
scoreg1

# In[106]:

scoreg2 = np.mean(cross_val_score(rfreg, train_X, train_Yreg,scoring=ms,
cv=10,n_jobs=2))
scoreg2

# Usuarios casuales:

# In[107]:

scorecas = np.mean(cross_val_score(rfcasual, train_X,
train_Ycasual,scoring='mean_squared_error', cv=10,n_jobs=2))
scorecas

# In[108]:

scorecas1 = np.mean(cross_val_score(rfcasual, train_X,
train_Ycasual,scoring='r2', cv=10,n_jobs=2))
scorecas1

# In[109]:

scorecas2 = np.mean(cross_val_score(rfcasual, train_X,
train_Ycasual,scoring=ms, cv=10,n_jobs=2))
scorecas2

# Suma de usuarios:

# In[110]:

scoretot = np.mean(cross_val_score(rfcount, train_X, train_Ycasual
+train_Yreg,scoring='mean_squared_error', cv=10,n_jobs=2))
scoretot
```

```
# In[111]:

scoretot1 = np.mean(cross_val_score(rfcount, train_X, train_Ycasual
+train_Yreg,scoring='r2', cv=10,n_jobs=2))
scoretot1

# In[112]:

scoretot2 = np.mean(cross_val_score(rfcount, train_X, train_Ycasual
+train_Yreg,scoring=ms, cv=10,n_jobs=2))
scoretot2

# ### Validación cruzada con feature importance:

# Usuarios casuales:

# In[ ]:

train_Xftcasual=train[colu]
train_Yftcasual=train['casual']
test_Xftcasual=test[colu]
test_Yftcasual=test['casual']

# In[113]:

scoreftcasual = np.mean(cross_val_score(ftcasual, train_Xftcasual,
train_Yftcasual,scoring='mean_squared_error', cv=10,n_jobs=2))
scoreftcasual

# In[114]:

scoreftcasual1 = np.mean(cross_val_score(ftcasual, train_Xftcasual,
train_Yftcasual,scoring='r2', cv=10,n_jobs=2))
scoreftcasual1

# In[115]:

scoreftcasual2 = np.mean(cross_val_score(ftcasual, train_Xftcasual,
train_Yftcasual,scoring=ms, cv=10,n_jobs=2))
scoreftcasual2

# Usuarios registrados:

# In[116]:

scoreftreg1 = np.mean(cross_val_score(ftreg, train_Xftreg,
train_Yftreg,scoring='r2', cv=10,n_jobs=2))
scoreftreg1

# In[117]:

scoreftreg = np.mean(cross_val_score(ftreg, train_Xftreg,
train_Yftreg,scoring='mean_squared_error', cv=10,n_jobs=2))
```

```
scoreftreg

# In[118]:

scoreftreg2 = np.mean(cross_val_score(ftreg, train_Xftreg,
train_Yftreg,scoring=ms, cv=10,n_jobs=2))
scoreftreg2

# Usuarios totales:

# In[119]:

scoreftcount = np.mean(cross_val_score(ftcount, train_Xftcount,
train_Yftcount,scoring='r2', cv=10,n_jobs=2))
scoreftcount

# In[120]:

scoreftcount1 = np.mean(cross_val_score(ftcount, train_Xftcount,
train_Yftcount,scoring='mean_squared_error', cv=10,n_jobs=2))
scoreftcount1

# In[121]:

scoreftcount2 = np.mean(cross_val_score(ftcount, train_Xftcount,
train_Yftcount,scoring=ms, cv=10,n_jobs=2))
scoreftcount2

# Suma de usuarios:

# In[122]:

scorefttot = np.mean(cross_val_score(ftcount, train_Xftcount,
train_Yftcasual+train_Yftreg,scoring='r2', cv=10,n_jobs=2))
scorefttot

# In[123]:

scorefttot1 = np.mean(cross_val_score(ftcount, train_Xftcount,
train_Yftcasual+train_Yftreg,scoring='mean_squared_error', cv=10,n_jobs=2))
scorefttot1

# In[124]:

scorefttot2 = np.mean(cross_val_score(ftcount, train_Xftcount,
train_Yftcasual+train_Yftreg,scoring=ms, cv=10,n_jobs=2))
scorefttot2

# ## Gridsearch:

# In[20]:

from sklearn.grid_search import GridSearchCV
```

```
# In[21]:

param_grid = {"max_depth": [3, 5, None],
              "max_features": [None, 'sqrt'],
              "min_samples_split": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10]}

# In[22]:

rfrcount= RandomForestRegressor(n_estimators=100,n_jobs=2)

# Usuarios totales:

# In[71]:

grid_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Ycount)

# In[72]:

grid_search.best_score_

# In[73]:

grid_search2 = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Ycount)

# In[74]:

grid_search2.best_score_

# In[23]:

from sklearn.metrics import make_scorer
ms = make_scorer(rmsle)

# In[24]:

grid_search1 = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Ycount)

# In[77]:

grid_search1.best_score_

# Suma de usuarios:

# In[78]:

gridtot_search= GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Ycasual+train_Yreg)

# In[79]:
```

```
gridtot_search.best_score_

# In[80]:

gridtot1_search= GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Ycasual+train_Yreg)

# In[83]:

gridtot1_search.best_score_

# In[28]:

gridtot2_search= GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Ycasual+train_Yreg)

# In[84]:

gridtot2_search.best_score_

# Usuarios registrados:

# In[85]:

gridreg_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Yreg)

# In[87]:

gridreg_search.best_score_

# In[88]:

gridreg1_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Yreg)

# In[89]:

gridreg1_search.best_score_

# In[90]:

gridreg2_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Yreg)

# In[91]:

gridreg2_search.best_score_

# Usuarios casuales:
```

```
# In[92]:

gridcas_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='r2',cv=10, n_jobs=2).fit(train_X,train_Ycasual)

# In[93]:

gridcas_search.best_score_

# In[94]:

gridcas1_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring='mean_squared_error',cv=10,
n_jobs=2).fit(train_X,train_Ycasual)

# In[95]:

gridcas1_search.best_score_


# In[96]:

gridcas2_search = GridSearchCV(rfrcount, param_grid=param_grid,
verbose=3,scoring=ms,cv=10, n_jobs=2).fit(train_X,train_Ycasual)

# In[98]:

gridcas2_search.best_score_

 # ## Predicción-Test:

# In[11]:

dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()

# In[12]:

dt1.datetime = dt1.datetime.apply(pd.to_datetime)
dt1['year']= dt1.datetime.apply(lambda x : x.year)
dt1['month'] = dt1.datetime.apply(lambda x : x.month)
dt1['hour'] = dt1.datetime.apply(lambda x : x.hour)
dt1['day'] = dt1.datetime.apply(lambda x: x.weekday_name)
dt1.head()

# In[13]:

dum_we=pd.get_dummies(dt1['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt1['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt1['day'],prefix='dayweek')

 # In[14]:

column=dt1.columns.values.tolist()
```

```
dt2=dt1[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtest=dto[column2].join(dum_dia)
dtest.head()

 # In[15]:

del dtest['season']
del dtest['weather']
del dtest['day']

 # In[16]:

X_test = dtest[cols]
y_test = rfcount.predict(X_test)

 # In[17]:

submission_rf =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_test},columns=['date
time','count'])

 # In[18]:

submission_rf.loc[submission_rf['count']<0.0,'count']=0.0


# In[19]:

submission_rf.to_csv('/Users/Miguel/Desktop/Submissions/sub_rf.csv',index=Fal
se)

 # Grid:

# In[27]:

y_testsumatorio=grid_search1.predict(X_test)
submission_adsumatorio =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testsumatorio},colum
ns=['datetime','count'])
submission_adsumatorio.loc[submission_adsumatorio['count']<0.0,'count']=0.0
submission_adsumatorio.to_csv('/Users/Miguel/Desktop/Submissions/sub_rfgrid.c
sv',index=False)

 # In[31]:

y_testsumatorio=gridtot2_search.predict(X_test)
submission_adsumatoriot =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testsumatorio},colum
ns=['datetime','count'])
submission_adsumatoriot.loc[submission_adsumatoriot['count']<0.0,'count']=0.0
```

```
submission_adsumatoriot.to_csv('/Users/Miguel/Desktop/Submissions/sub_rfgrids
umatorio.csv',index=False)
```

I.6 Máquina de soporte vectorial.

```
 # coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()

 # In[3]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt['day'],prefix='dayweek')
column=dt.columns.values.tolist()
dt2=dt[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()

# In[4]:

del dtt['season']
del dtt['weather']
del dtt['day']
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1',
'Tiempo_2','Tiempo_3', 'Tiempo_4', 'Estación_1',
   'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a

 # In[5]:

from sklearn.cross_validation import train_test_split
```

```
train, test = train_test_split(dtt, test_size = 0.2)

# In[6]:

train_X=train[cols]
train_Ycasual=train['casual']
train_Yreg=train['registered']
train_Ycount=train['count']
test_X=test[cols]
test_Ycasual=test['casual']
test_Yreg=test['registered']
test_Ycount=test['count']

# In[7]:

from sklearn.svm import SVR
from sklearn.metrics import r2_score
import math
from math import log
def rmsle(y, y_pred):
   y=y.values
   y_pred[y_pred<0]=0
   assert len(y) == len(y_pred)
   terms_to_sum = [(log(y_pred[i] + 1) - log(y[i] + 1)) ** 2.0 for i,pred in
enumerate(y_pred)]
   return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5
from sklearn.metrics import mean_squared_error

# ## RBF

# In[8]:

svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.001)
svr = svr_rbf.fit(train_X, train_Ycount)

# In[9]:

svr1 = svr_rbf.fit(train_X, train_Yreg)
svr2 = svr_rbf.fit(train_X, train_Ycasual)

# In[10]:

svr_pred=svr.predict(test_X)
svr_pred

# In[11]:

svr1_pred=svr1.predict(test_X)
svr1_pred

# In[12]:

svr2_pred=svr2.predict(test_X)
svr2_pred
```

```
# In[13]:

svr_pred[svr_pred<0]=0
svr1_pred[svr1_pred<0]=0
svr2_pred[svr2_pred<0]=0

# In[14]:

r2_score(test_Ycount,svr_pred)


# In[15]:

r2_score(test_Yreg,svr1_pred)

# In[17]:

r2_score(test_Ycasual,svr2_pred)

# In[18]:

r2_score(test_Ycount,svr1_pred+svr2_pred)

# In[19]:

mean_squared_error(test_Ycount,svr_pred)

# In[20]:

mean_squared_error(test_Yreg,svr1_pred)


# In[21]:

mean_squared_error(test_Ycasual,svr2_pred)

# In[22]:

mean_squared_error(test_Ycount,svr1_pred+svr2_pred)

# In[24]:

rmsle(test_Ycount,svr_pred)

# In[26]:

rmsle(test_Yreg,svr1_pred)

# In[27]:

rmsle(test_Ycasual,svr2_pred)
```

```
# In[28]:

rmsle(test_Ycount,svr1_pred+svr2_pred)

# ### Cross validation:

# In[29]:

from sklearn.cross_validation import cross_val_score


# Para usuarios totales:

# In[30]:

score = np.mean(cross_val_score(svr, train_X,
train_Ycount,scoring='mean_squared_error', cv=10,n_jobs=2))
score

# In[31]:

score1 = np.mean(cross_val_score(svr, train_X, train_Ycount,scoring='r2',
cv=crossvalidation,n_jobs=1))
score1

# In[13]:

from sklearn.metrics import make_scorer

# In[14]:

ms = make_scorer(rmsle)

# In[41]:

score2 = np.mean(cross_val_score(svr, train_X, train_Ycount, scoring=ms,
cv=10))
score2

# Para usuarios registrados:

# In[42]:

score_reg = np.mean(cross_val_score(svr1, train_X,
train_Yreg,scoring='mean_squared_error', cv=10,n_jobs=2))
score_reg

# In[44]:

score_reg1 = np.mean(cross_val_score(svr1, train_X, train_Yreg,scoring='r2',
cv=10 ,n_jobs=2))
score_reg1

# In[45]:
```

```
score_reg2 =np.mean(cross_val_score(svr1, train_X, train_Yreg, scoring=ms,
cv=10,n_jobs=2))
score_reg2

# Para usuarios casuales:

# In[46]:

score_cas = np.mean(cross_val_score(svr2, train_X,
train_Ycasual,scoring='mean_squared_error', cv=10,n_jobs=2))
score_cas

 # In[48]:

score_cas1 = np.mean(cross_val_score(svr2, train_X,
train_Ycasual,scoring='r2', cv=10,n_jobs=2))
score_cas1

# In[49]:

score_cas2 = np.mean(cross_val_score(svr1, train_X, train_Ycasual,
scoring=ms, cv=10,n_jobs=2))
score_cas2


# In[ ]:

 # ## Lineal

# In[8]:

svr_lineal = SVR(kernel='linear',C=1e3)

 # In[9]:

svrl= svr_lineal.fit(train_X, train_Ycount)

 # In[10]:

svrl1= svr_lineal.fit(train_X, train_Yreg)

 # In[11]:

svrl2= svr_lineal.fit(train_X, train_Ycasual)

 # In[12]:

svr1_pred=svrl.predict(test_X)
svr1_pred

 # In[14]:
```

```
svrl1_pred=svrl1.predict(test_X)
svrl1_pred

 # In[15]:

svr12_pred=svrl2.predict(test_X)
svr12_pred

# In[16]:

svrl.score(train_X,train_Ycount)

 # In[17]:

svrl1.score(train_X,train_Yreg)

# In[18]:

svrl2.score(train_X,train_Ycasual)

 # In[19]:

r2_score(test_Ycount,svr1_pred)

# In[21]:

r2_score(test_Yreg,svrl1_pred)

# In[22]:

r2_score(test_Ycasual,svr12_pred)

# In[26]:

r2_score(test_Ycount,svrl1_pred+svr12_pred)

 # In[27]:

mean_squared_error(test_Ycount,svr1_pred)

 # In[30]:

mean_squared_error(test_Yreg,svrl1_pred)

 # In[31]:

mean_squared_error(test_Ycasual,svr12_pred)

 # In[34]:

mean_squared_error(test_Ycount,svrl1_pred+svr12_pred)

# In[37]:
```

```
#Hacemos nulos los valores negativos.
svr1_pred[svr1_pred<0]=0
svrl1_pred[svrl1_pred<0]=0
svr12_pred[svr12_pred<0]=0

 # In[38]:

rmsle(test_Ycount,svr1_pred)

 # In[40]:

rmsle(test_Yreg,svrl1_pred)

 # In[41]:

rmsle(test_Ycasual,svr12_pred)

 # In[46]:

rmsle(test_Ycount,svrl1_pred+svr12_pred)

 # ### Lineal Cross validation:

# Usuarios totales:

# In[12]:

from sklearn.cross_validation import cross_val_score

# In[ ]:

scorel =np.mean( cross_val_score(svrl, train_X, train_Ycount,scoring='r2',
cv=10,n_jobs=2))
scorel

 # In[ ]:

scorel1 = np.mean(cross_val_score(svrl, train_X,
train_Ycount,scoring='mean_squared_error', cv=10,n_jobs=2))
scorel1

 # In[15]:

scorel2= np.mean(cross_val_score(svrl, train_X, train_Ycount,scoring=ms,
cv=10))
scorel2


# Usuarios registrados:

# In[ ]:
```

```
scorel_reg = np.mean(cross_val_score(svrl1, train_X, train_Yreg,scoring='r2',
cv=10,n_jobs=2))
scorel_reg

# In[ ]:

scorel_reg1 =np.mean(cross_val_score(svrl1, train_X,
train_Yreg,scoring='mean_squared_error', cv=10,n_jobs=2))
scorel_reg1

# In[ ]:

scorel_reg2 = np.mean(cross_val_score(svrl1, train_X, train_Yreg,scoring=ms,
cv=10,n_jobs=2))
scorel_reg2

 # Usuarios casuales:

# In[ ]:

scorel_cas = np.mean(cross_val_score(svrl, train_X,
train_Ycasual,scoring='r2', cv=10,n_jobs=2))
scorel_cas

 # In[ ]:

scorel_cas1 =np.mean(cross_val_score(svrl, train_X,
train_Ycasual,scoring='mean_squared_error',cv=10,n_jobs=2))
scorel_cas1

 # In[ ]:

scorel_cas2 = np.mean(cross_val_score(svrl, train_X, train_casual,scoring=ms,
cv=10,n_jobs=2))
scorel_cas2

 # ## Polinomial

# In[20]:

svr_poly = SVR(kernel='poly',C=1, degree=2)

# In[21]:

svrp= svr_poly.fit(train_X, train_Ycount)

# In[52]:

svrp1= svr_poly.fit(train_X, train_Yreg)

 # In[53]:

svrp2= svr_poly.fit(train_X, train_Ycasual)
```

```
# In[54]:

svrp_pred=svrp.predict(test_X)
svrp_pred

 # In[55]:

svrp1_pred=svrp1.predict(test_X)
svrp1_pred

# In[56]:

svrp2_pred=svrp2.predict(test_X)
svrp2_pred

 # In[57]:

r2_score(test_Ycount,svrp_pred)

 # In[58]:

r2_score(test_Yreg,svrp1_pred)

# In[59]:

r2_score(test_Ycasual,svrp2_pred)

 # In[60]:

r2_score(test_Ycount,svrp1_pred+svrp2_pred)

# In[61]:

mean_squared_error(test_Ycount,svrp_pred)

 # In[62]:

mean_squared_error(test_Yreg,svrp1_pred)

 # In[63]:

mean_squared_error(test_Ycasual,svrp2_pred)

 # In[64]:

mean_squared_error(test_Ycount,svrp1_pred+svrp2_pred)

 # In[65]:

svrp_pred[svrp_pred<0]=0
svrp1_pred[svrp1_pred<0]=0
svrp2_pred[svrp2_pred<0]=0
```

```
# In[66]:

rmsle(test_Ycount,svrp_pred)

# In[67]:

rmsle(test_Yreg,svrp1_pred)


# In[68]:

rmsle(test_Ycasual,svrp2_pred)

# In[69]:

rmsle(test_Ycount,svrp1_pred+svrp2_pred)

# ### Polinomial Cross validation:

# In[ ]:

from sklearn.cross_validation import cross_val_score


# Usuarios totales:

# In[ ]:

scorep= np.mean(cross_val_score(svrp, train_X, train_Ycount,scoring='r2',
cv=10,n_jobs=2))
scorep

# In[ ]:

scorep1= np.mean(cross_val_score(svrp, train_X,
train_Ycount,scoring='mean_squared_error', cv=10,n_jobs=2))
scorep1

# In[ ]:

scorep2= np.mean(cross_val_score(svrp, train_X, train_Ycount,scoring=ms,
cv=10,n_jobs=2))
scorep2

# Usuarios registrados:

# In[ ]:

scorep_reg= np.mean(cross_val_score(svrp1, train_X,
train_Yreg,scoring='r2',cv=10,n_jobs=2))
scorep_reg

# In[ ]:
```

```
scorep_reg1=np.mean(cross_val_score(svrp1, train_X,
train_Yreg,scoring='man_squared_error',cv=10,n_jobs=2))
scorep_reg1

# In[ ]:

scorep_reg2= np.mean(cross_val_score(svrp1, train_X, train_Yreg,scoring=ms,
cv=10,n_jobs=2))
scorep_reg2

# Usuarios casuales:

# In[ ]:

scorep_cas= np.mean(cross_val_score(svrp2, train_X,
train_Ycasual,scoring='r2', cv=10,n_jobs=2))
scorep_cas

# In[ ]:

scorep_cas1= np.mean(cross_val_score(svrp2, train_X,
train_Ycasual,scoring='mean_squared_error', cv=10,n_jobs=2))
scorep_cas1

# In[ ]:

scorep_cas2= np.mean(cross_val_score(svrp2, train_X,
train_Ycasual,scoring=ms, cv=10,n_jobs=2))
scorep_cas2

# ## Predicción-test:

# In[12]:

dt1=pd.read_csv('/Users/Miguel/Downloads/test.csv')
dt1.head()

# In[13]:

dt1.datetime = dt1.datetime.apply(pd.to_datetime)
dt1['year']= dt1.datetime.apply(lambda x : x.year)
dt1['month'] = dt1.datetime.apply(lambda x : x.month)
dt1['hour'] = dt1.datetime.apply(lambda x : x.hour)
dt1['day'] = dt1.datetime.apply(lambda x: x.weekday_name)
dt1.head()

# In[14]:

dum_we=pd.get_dummies(dt1['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt1['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt1['day'],prefix='dayweek')

# In[15]:
```

```
column=dt1.columns.values.tolist()
dt2=dt1[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtest=dto[column2].join(dum_dia)
dtest.head()

# In[16]:

del dtest['season']
del dtest['weather']
del dtest['day']

# RBF:

# In[18]:

X_test = dtest[cols]
##y_test = svr.predict(X_test)

 # In[15]:

submission_svm =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_test},columns=['date
time','count'])

# In[16]:

submission_svm.loc[submission_svm['count']<0.0,'count']=0.0

# In[17]:

submission_svm.to_csv('/Users/Miguel/Desktop/Submissions/sub_svm.csv',index=F
alse)

# Lineal:

# In[19]:

y_testl = svrl.predict(X_test)
submission_svmlineal =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testl},columns=['dat
etime','count'])
submission_svmlineal.loc[submission_svmlineal['count']<0.0,'count']=0.0
submission_svmlineal.to_csv('/Users/Miguel/Desktop/Submissions/sub_svmlineal.
csv',index=False)

# Polinomial:

# In[22]:

y_testp = svrp.predict(X_test)
```

```
submission_svmpoli =
pd.DataFrame({'datetime':dtest.datetime.values,'count':y_testp},columns=['dat
etime','count'])
submission_svmpoli.loc[submission_svmpoli['count']<0.0,'count']=0.0
submission_svmpoli.to_csv('/Users/Miguel/Desktop/Submissions/sub_svmpoli.csv'
,index=False)
```

I.7. Máquina soporte vectorial con datos estandarizados.

El código de máquinas de soporte vectorial con datos normalizados tiene una implementación similar al apartado anterior, añadiendo dicha estandarización de datos:

```
# coding: utf-8

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# In[2]:

dt=pd.read_csv('/Users/Miguel/Downloads/train.csv')
dt.head()


# In[3]:

dt.datetime = dt.datetime.apply(pd.to_datetime)
dt['year']= dt.datetime.apply(lambda x : x.year)
dt['month'] = dt.datetime.apply(lambda x : x.month)
dt['hour'] = dt.datetime.apply(lambda x : x.hour)
dt['day'] = dt.datetime.apply(lambda x: x.weekday_name)
dum_we=pd.get_dummies(dt['weather'],prefix='Tiempo')
dum_sea=pd.get_dummies(dt['season'],prefix='Estación')
dum_dia=pd.get_dummies(dt['day'],prefix='dayweek')
column=dt.columns.values.tolist()
dt2=dt[column].join(dum_we)
column1=dt2.columns.values.tolist()
dto=dt2[column1].join(dum_sea)
column2=dto.columns.values.tolist()
dtt=dto[column2].join(dum_dia)
dtt.head()


# In[4]:

del dtt['season']
del dtt['weather']
del dtt['day']
```

```
a=[x for x in dtt.columns.values if 'dayweek_' in x]
cols=['temp', 'holiday', 'humidity', 'windspeed', 'Tiempo_1',
'Tiempo_2','Tiempo_3', 'Tiempo_4', 'Estación_1',
   'Estación_2', 'Estación_3', 'Estación_4', 'workingday','month','hour'] +a


# ## Estandarización:

# In[5]:

from sklearn.preprocessing import MinMaxScaler


# In[6]:

dtt.columns


# In[7]:

scaler =MinMaxScaler(feature_range=(0, 1), copy=True)


# In[8]:

del dtt['datetime']

# In[9]:

dtt.columns

# In[10]:

dtt_sc=scaler.fit_transform(dtt)

# In[11]:
dttsca=pd.DataFrame(dtt_sc,columns=dtt.columns)

# In[12]:

dttsca

# In[13]:

scaler.scale_
##El resto del código es igual que el apartado anterior del anexo. En este
documento, no se expone.
```