**Título:** Exploratorio basado en datos reales de la APP Strava en la zona de Jeddah en Arabia Saudí.

**Autor:** Francisco García Núñez

**Resumen**

En este documento se realiza un análisis exploratorio con datos reales de la aplicación deportiva Strava.

El objetivo del proyecto es meramente educativo, donde intentaré reforzar conocimientos obtenidos a lo largo de lo que llevamos de bootcamp y poder llevarlos a la práctica.

Se busca realizar un análisis de los datos obtenidos a través de la plataforma Kaggle. Donde el principal problema era la dificultad de detectar posibles trampas en segmentos de un recorrido determinado.

En segundo, lugar dejaremos preparado el documento para aplicar posteriormente un método para predecir el tiempo o la posición que ocuparía en el ranking por segmento según las variables disponibles.

**Introducción**

En este documento trataremos un caso real que ha sido obtenido de la plataforma [www.kaggle.com](http://www.kaggle.com). En concreto trataremos un problema subido a este sitio web por uno de los usuarios, no siendo esponsorizado por la compañía Strava.

Strava es una red social basada en Internet y [GPS](https://es.wikipedia.org/wiki/GPS) enfocada a deportistas como pueden ser ciclistas y corredores y una aplicación de seguimiento GPS deportiva. La red la gestiona una compañía del mismo nombre con sede en [San Francisco](https://es.wikipedia.org/wiki/San_Francisco_(California)) (Wikipedia).

Para entender bien el problema, contextualizaremos que un segmento de Strava: “Un segmento de Strava es un camino con una distancia limitada que cualquiera puede tomar (dado que está grabando una actividad) para participar en una tabla de clasificación y ver quién es el más rápido. El primero en la tabla de clasificación de ambos géneros se conoce como el Rey / Reina de la Montaña (KOM, QOM)”.

El usuario en cuestión, propuso la problemática de que a pesar del esfuerzo de la empresa Strava para saber quién falsea los datos, muchos usuarios encuentran la manera para realizar de manera “ilegal” esos segmentos e ir avanzando en la clasificación de los diferentes grupos. ¿Cómo se falsean esos datos? En vez de realizar estos segmentos como debiera, los realizan en motocicletas, coches o cualquier otro vehículo motorizado. El propósito de este Kaggle, es identificar que usuarios podrían estar falseando esos datos.

Puesto que el propósito del kaggle no es muy atractivo, a mi parecer. Dejaremos el documento habilitado para aplicar, en un futuro no muy lejano, un método en el que el target sea el tiempo en el que realizará el segmento, la posición que ocupará en el ranking por segmento, que identifique los rangos de edad o el género al que pertenece el usuario. A nivel personal me parece más atractivo para los usuarios, ya que siempre intentarás mejorar lo que te están prediciendo.

**1. Requisitos**

Para el desarrollo de este exploratorio se hará a través de un IDE de Python a través de *Anaconda*. En mi caso se utilizará *Jupyter Notebook,* el utilizado durante el bootcamp y por lo tanto con el que estoy más familiarizado. El sistema operativo con el que cuenta mi equipo se trata de Windows 10, sistema comercial a nivel usuario puesto que no tengo idea de otros sistemas operativos como Linux (S.O de código abierto).

**2. Datos**

Según el usuario de *Kaggle* que decidió subir el Dataset, es crear un modelo en el que se analicen todas las entradas realizadas en *Strava* para determinados segmentos y marcarlos como sospechosos de falsear datos. Puesto que todavía no sabemos realizar modelos de Machine Learning, simplemente se realizará un exploratorio y análisis de los datos.

Pasos:

1. Para poder acceder a los datos se necesita una cuenta de usuario en *Kaggle*, se puede hacer de manera gratuita. Los datos son de un problema que se plateó hace 7 meses por un usuario particular de la plataforma. La licencia es de dominio público. El problema solo tiene un archivo para la descarga ‘jeddah\_strava\_segments.csv’ que tiene un peso de 236 K.

El fichero contiene:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Feature name** | **Description** |
| String | user*age*group | The age group of the participant |
| String | user*weight*category | The weight category of the participant |
| int | user\_id | The User ID of the participant |
| Date | attempt\_date | The date of attempt of the entry in the leaderboard |
| String | gender | The gender of the participant |
| int | smt\_rank | The participant's rank in the leaderboard |
| float | smt*avg*spd | The participant's average speed in km/h within the segment |
| int | smt*finish*seconds | The time taken for the participant to complete the segment in seconds |
| String | smt\_name | The name of the segment |
| String | act\_title | The title of the activity which included the segment attempt |
| float | act*avg*spd | The participant's average speed in km/h during the activity |
| float | act*max*spd | The participant's maximum speed in km/h during the activity |
| float | act*total*km | The total distance of the participant's activity in kilometers |
| int | act*moving*seconds | The total time which the participant spent moving during the activity in seconds |
| int | act*total*seconds | The total time of the activity (including stop times) in seconds |
| int | has*hr*data | Whether there was data of participant's heart rate in the segment attempt |

Tabla: fichero *jeddah\_strava\_segments*

**3. Análisis exploratorio**

Un análisis exploratorio consiste en el procesamiento y transformación de los datos de entrada que nos vienen dados de un archivo (en nuestro caso csv) o API de la manera que nos sean útiles para el modelo.

Los análisis exploratorios no son todos iguales, todo dependerá del objetivo final del proyecto y de la naturaleza de los datos; cantidad, nulos, duplicados, tipo…

Para comenzar, se hará la lectura de los datos.

Antes de abrir el archivo o el API habrá que importar las librerías que no serán necesarias y en las que nos apoyaremos. El lenguaje de programación utilizado es Python 3, que es el de referencia del proyecto.

De manera inicial, empezaremos leyendo todos los datos.

Importamos las librerías en las que nos apoyamos. Vamos a programar en Python 3 (Python3, 2020), que es el lenguaje de referencia en el proyecto.

Vamos a explicar las librerías en las que nos apoyaremos:

* Numéricas: Numpy
* Dataframes: Pandas
* Científicas: SciPy, Stat
* Preprocesado: Preprocessing, MinMaxScaler
* Visualización: Seaborn, Pyplot
* Entrenamiento y test: Train\_test\_split
* Métricas: metrics(sklearn)

A continuación explicaremos paso a paso:

En primer lugar, leemos el archivo con una decodificación “utf-8” para leer los caracteres en alfabeto latino y parte del alfabeto árabe, puesto que es un dataset de Arabia Saudí. Intenté leerlo con la codificación “latin-1” y me salían columnas ilegibles.

df.columns

Index(['user\_age\_group', 'user\_id', 'attempt\_date', 'gender', 'smt\_rank',

'smt\_avg\_spd', 'smt\_finish\_seconds', 'smt\_name', 'user\_weight\_category',

'act\_title', 'act\_avg\_spd', 'act\_max\_spd', 'act\_total\_km',

'act\_moving\_seconds', 'act\_total\_seconds', 'has\_hr\_data'],

dtype='object')

Podemos verificar el nombre de las columnas con la tabla del punto 2, donde se indican todos las columnas del dataset.

Comprobamos el tamaño del data frame:

df.shape

(7865, 16)

Podemos comprobar que tenemos una muestra de 7865 registros (filas) con 16 atributos (columnas) diferentes.

Para poder hacernos una idea del dataset, hacemos una visión de los diferentes atributos para conocer cómo se denominan, cómo se distribuyen los datos, etc.

A pesar de mirar por encima la tabla completa, se desconoce si existen datos nulos dentro de nuestro dataset. Para comprobarlos hacemos:

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 7865 entries, 0 to 7864

Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 user\_age\_group 7865 non-null object

1 user\_id 7865 non-null int64

2 attempt\_date 7865 non-null object

3 gender 7865 non-null object

4 smt\_rank 7865 non-null int64

5 smt\_avg\_spd 7865 non-null float64

6 smt\_finish\_seconds 7865 non-null int64

7 smt\_name 7865 non-null object

8 user\_weight\_category 7430 non-null object

9 act\_title 7865 non-null object

10 act\_avg\_spd 7865 non-null float64

11 act\_max\_spd 7865 non-null float64

12 act\_total\_km 7865 non-null float64

13 act\_moving\_seconds 7865 non-null int64

14 act\_total\_seconds 7865 non-null int64

15 has\_hr\_data 7865 non-null int64

dtypes: float64(4), int64(6), object(6)

memory usage: 983.2+ KB

Como podemos observar no existe ningún nulo en ninguna de las columnas. Pero para asegurarnos realizamos otra sentencia para detectar algún nulo:

df.isnull().sum()

user\_age\_group 0

user\_id 0

attempt\_date 0

gender 0

smt\_rank 0

smt\_avg\_spd 0

smt\_finish\_seconds 0

smt\_name 0

user\_weight\_category 435

act\_title 0

act\_avg\_spd 0

act\_max\_spd 0

act\_total\_km 0

act\_moving\_seconds 0

act\_total\_seconds 0

has\_hr\_data 0

dtype: int64

En este caso podemos ver, que en la columna “user\_weight\_category” hay 435 datos nulos. Ese atributo es poco relevante en nuestro dataset y en nuestro exploratorio. El porcentaje de datos nulos en este caso sería un 5.53% del total de los mismos.

Con las siguientes sentencias podemos ver los rangos de edades que vienen dados en el dataset y el número total de etiquetas.

print(X\_train['user\_age\_group'].unique())

print('Número total de etiquetas de edad ' + str(len(X\_train['user\_age\_group'].unique())))

['25 to 34' '19 and under' '35 to 44' '55 to 64' '45 to 54' '20 to 24'

'65 to 69' '75+' '70 to 74']

Número total de etiquetas de edad 9

X\_train.user\_age\_group.value\_counts() -> Vemos la distribución de cantidad de registros en “*X\_train*”:

25 to 34 2708

35 to 44 1866

45 to 54 667

20 to 24 568

19 and under 297

55 to 64 152

65 to 69 16

75+ 15

70 to 74 3

Name: user\_age\_group, dtype: int64

Para seguir con la creación de variables categóricas, creamos una variable “edad” que coge el DataFrame origianl para no tener que crear dos variables similares cuando lo hagamos en “X\_test”. A continuación vemos el código necesario para crear las categorías:

le = preprocessing.LabelEncoder() #creamos un objeto LabelEncoder, con las diferentes etiquetas de género nos atribuye un número de clase LabelEncoder

le.fit(edad) #Realizamos el fit a la variable 'edad' creada en la celda anterior

edad\_OneHot\_Encoder = le.transform(X\_train.user\_age\_group)

#creamos una variable nueva a la que se aplicará el 'transform' con el objeto 'le (label encoder)' con la columna 'user\_age\_group'

X\_train['edad\_OneHot\_Encoder'] = edad\_OneHot\_Encoder #Creamos una columna nueva en al DF

X\_train.loc[:,['user\_age\_group', 'edad\_OneHot\_Encoder']].head(10)

#vemos una muestra de cómo quedaría la columna de partida 'user\_age\_group' con las etiquetas atribuidas en la columna 'edad\_OneHot\_Encoder'

El resultado final sería el siguiente:

|  |  |
| --- | --- |
| Nomenclatura Dataset | Categoría |
| 19 and under | 0 |
| 20 to 24 | 1 |
| 25 to 34 | 2 |
| 35 to 44 | 3 |
| 45 to 54 | 4 |
| 55 to 64 | 5 |
| 65 to 69 | 6 |
| 70 to 74 | 7 |
| 75+ | 8 |

También crearemos una nueva variable categórica para distinguir el género de los usuarios de la aplicación. Debido a que el Dataset original es un STR, la creamos para que sea más fácil pintar las diferentes gráficas y que para el futuro modelo sea más sencillo:

print(X\_train['gender'].unique())

print('Número total de etiquetas de género ' + str(len(X\_train['gender'].unique())))

['male' 'female']

Número total de etiquetas de género 2

Al igual que con la variable 'edad' lo hacemos con esta, cogiendo del dataframe original.

genero = df['gender'].unique()

le.fit(genero) #Realizamos el fit a la variable 'genero' creada en la celda anterior

gender\_OneHot\_Encoder = le.transform(X\_train.gender)

#creamos una variable nueva a la que se aplicará el 'transform' con el objeto 'le (label encoder)' con la columna 'gender\_age\_group'

X\_train['gender\_OneHot\_Encoder'] = gender\_OneHot\_Encoder #Creamos una columna nueva en al DF

X\_train.loc[:,['gender', 'gender\_OneHot\_Encoder']].head(10)

#vemos una muestra de cómo quedaría la columna de partida 'user\_age\_group' con las etiquetas atribuidas en la columna 'edad\_OneHot\_Encoder'

Obteniendo como resultado final:

|  |  |
| --- | --- |
| Nomenclatura Dataset | Categoría |
| Female | 0 |
| Male | 1 |

Para que el modelo identifique correctamente los diferentes segmentos que el usuario registra, hemos creado una categoría para cada uno de ellos. Será la manera más fácil de identificarlos:

|  |  |
| --- | --- |
| Nomenclatura Dataset | Categoría |
| Al Fardoos to shellfish round about | 0 |
| Headquarters Business Park | 1 |
| King Street side | 2 |
| North Corniche | 3 |
| Obhur North Short | 4 |
| Oghor 2 SailsIsland | 5 |
| PRINCE SULTAN - ASSALAM TO FATIMA | 6 |
| S.Ubhur Alkurnaysh South Bound with detour | 7 |
| Starbucks to Majid | 8 |

El procedimiento para la creación de esta nueva etiqueta ha sido muy similar al de género y al del rango de edad:

le.fit(segmento) #Realizamos el fit a la variable 'smt\_name' creada en la celda anterior

gender\_OneHot\_Encoder = le.transform(df.smt\_name)

#creamos una variable nueva a la que se aplicará el 'transform' con el objeto 'le (label encoder)' con la columna 'smt\_name'

df['smt\_name\_OneHot\_Encoder'] = gender\_OneHot\_Encoder #Creamos una columna nueva en al DF

df.loc[:,['smt\_name', 'smt\_name\_OneHot\_Encoder']]

Para poder visualizar de una mejor manera algunos datos más relevantes, hemos pasado los datos de la columna “act\_moving\_seconds”, que estaba en segundos a minutos y modificando posteriormente el nombre de la columna a “act\_moving\_min”.

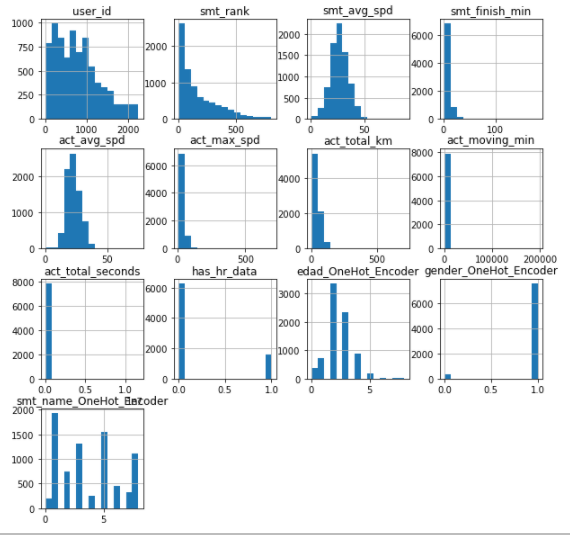
X\_train['act\_moving\_seconds'] = X\_train['act\_moving\_seconds']/60

X\_train.rename(columns={'act\_moving\_seconds': 'act\_moving\_min'}, inplace=True)

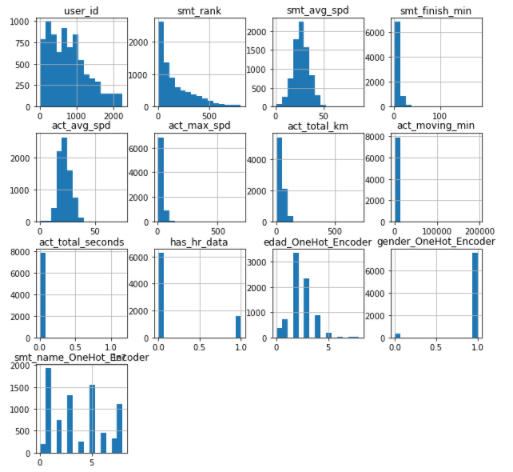
En primer lugar, vamos a estudiar la asimetría estadística para conocer como de simétricos son los datos de partida. En este caso vamos a representar la simetría del DF antes y después de la eliminación y tratamiento de outliers.

Para continuar, mostraremos los histogramas para entrar en la representación gráfica de las variables, intuyendo la forma que tienen las funciones de densidad de probabilidad.

La primera imagen muestra los histogramas del DF antes y la segunda después del tratamiento de los outliers :



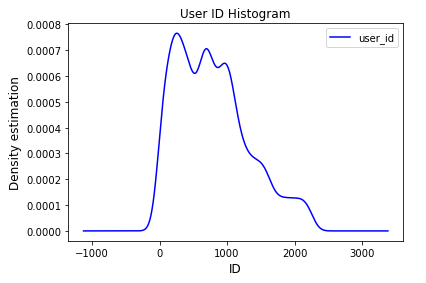
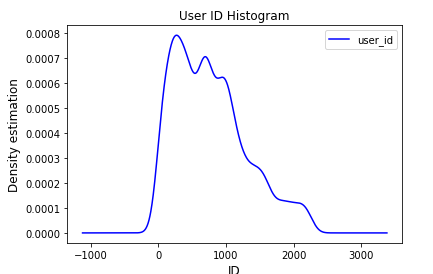
*Ilustración: Histograma DF antes del tratamiento de outliers*



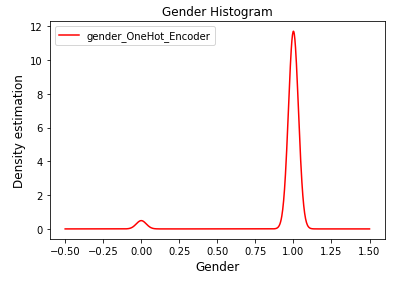
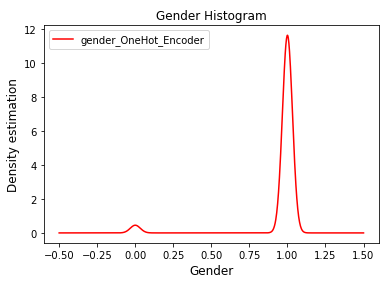
*Ilustración: Histograma después del tratamiento de outliers.*

Podemos comprobar que la simetría de los datos se mantiene antes y después del tratamiento de los datos con los outliers.

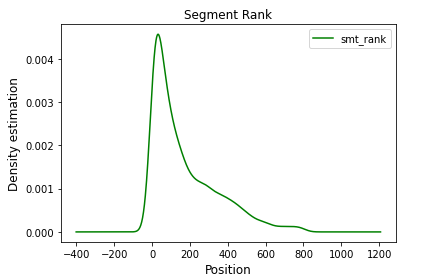
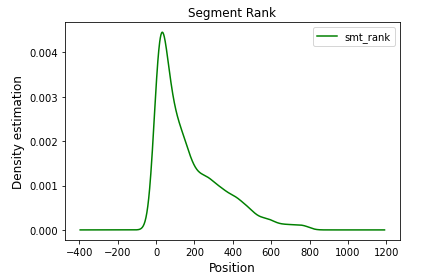
Vamos a comparar los datos con las variables antes y después de la eliminación de los outliers, para ver si la distribución de los mismos varía

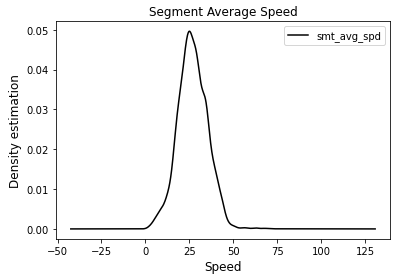
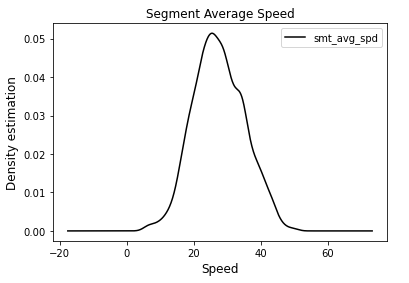
*Ilustración: Izq. Antes – Der. Después. User\_id*

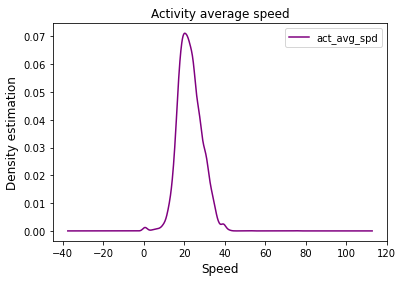
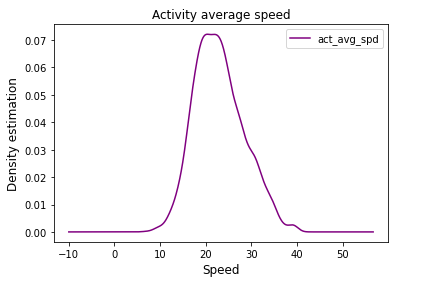
*Ilustración: Izq. Antes – Der. Después. Gender*

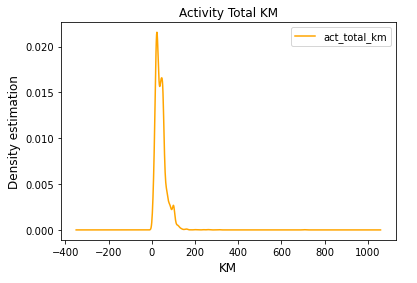
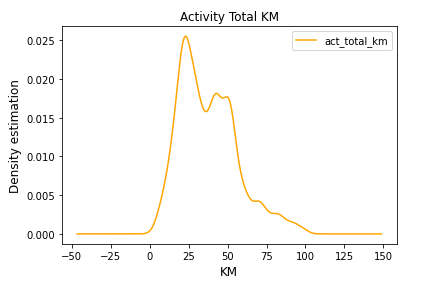
*Ilustración: Izq. Antes – Der. Después. Segment Rank*

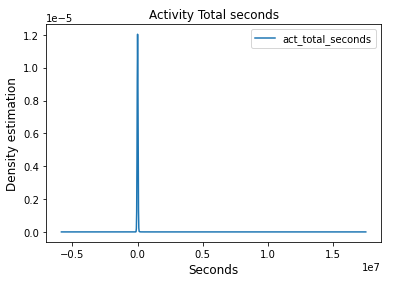
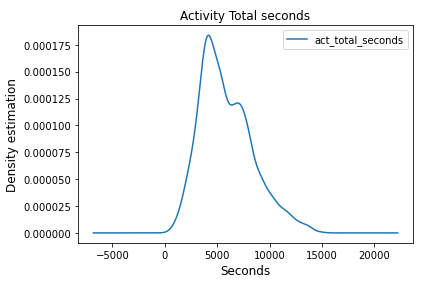
*Ilustración: Izq. Antes – Der. Después. Segment Average Speed*

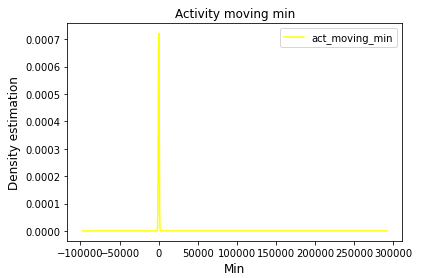
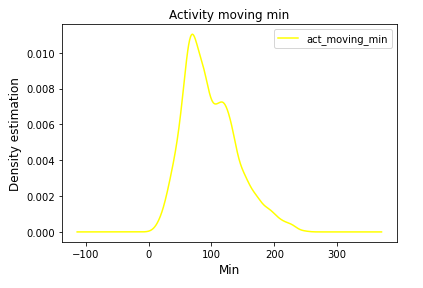
*Ilustración: Izq. Antes – Der. Después. Activity Average Speed*

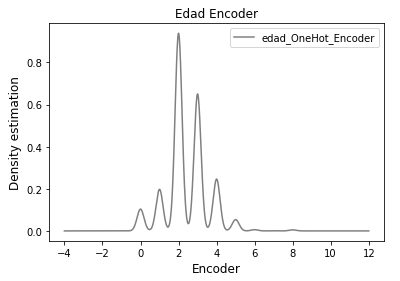
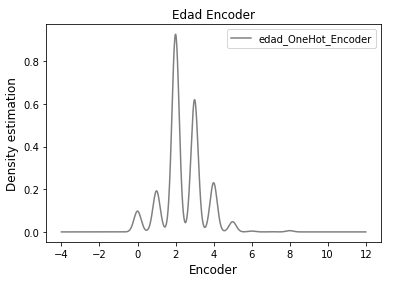
*Ilustración: Izq. Antes – Der. Después. Activity Total KM*

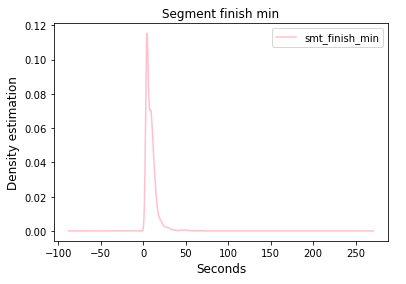
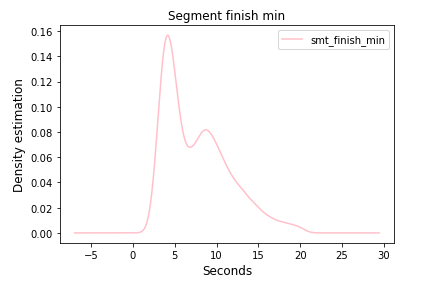
*Ilustración: Izq. Antes – Der. Después. Activity Total Seconds*

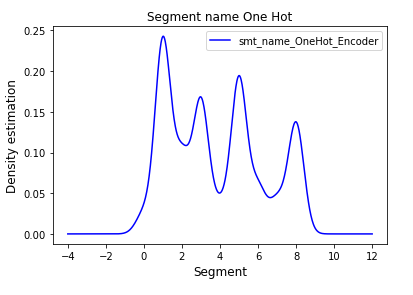
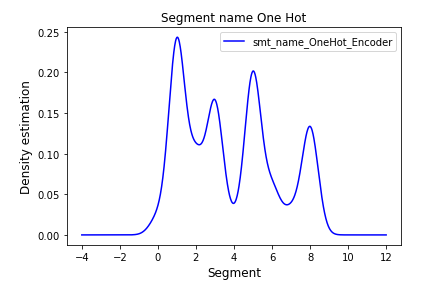
*Ilustración: Izq. Antes – Der. Después. Activity moving min*

*Ilustración: Izq. Antes – Der. Después. Edad Encoder*

*Ilustración: Izq. Antes – Der. Después. Segment finish min*

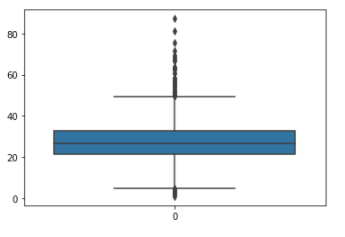
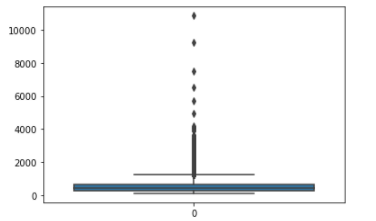
 

*Ilustración: Izq. Antes – Der. Después. Segment Name One Hot*

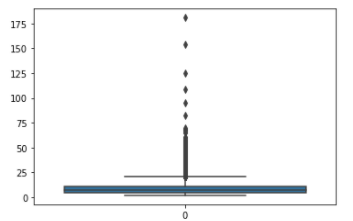
Vemos que al eliminar outliers los datos en algunas de las gráficas no se distribuyen de igual manera, sobre todo porque algunos datos estaban muy fuera del conjunto de los datos. El ejemplo claro es en el tiempo de duración de la activad y en la velocidad.

A continuación, para conocer los outliers (valores de baja aparición) recurriremos a los diagramas de cajas, los cuales los marcan claramente. Lo haremos desde el Dataframe principal.

En primer lugar analizaremos variables por segmentos:

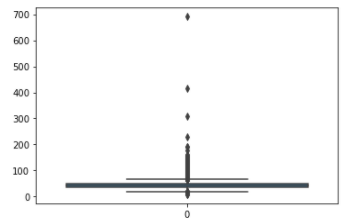
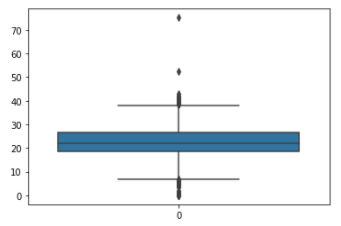
 

*Ilustración: Diagrama de cajas - smt\_avg\_spd y smt\_finish\_seconds*

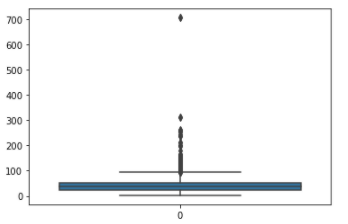


*Ilustración:Diagrama de caja – smt\_finish\_minuts*

En segundo lugar, variables por actividad:

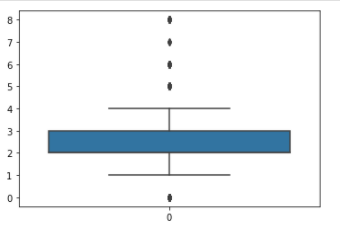


*Ilustración:Diagrama de caja – act\_avg\_spd - act\_max\_spd*



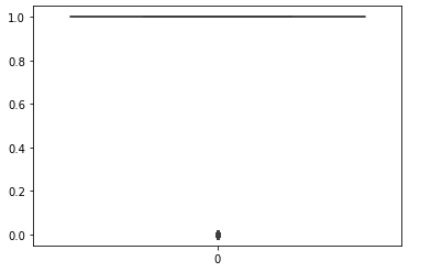
*Ilustración:Diagrama de caja – act\_total\_km*

Por último he hecho un boxplot con la columna de categoría creadas para ver qué es lo más frecuente:

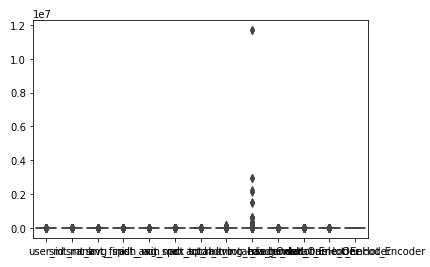
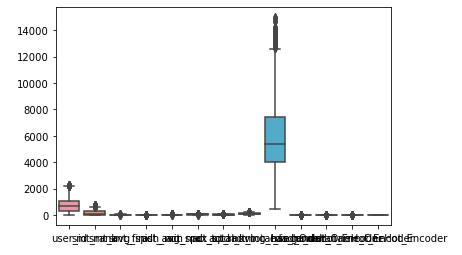


*Ilustración:Diagrama de caja – edad\_OneHot\_Encoder*

Como podemos ver a continuación, en Arabia Saudí parece ser que el ser mujer y hacer este tipo de deporte se considera outlier:

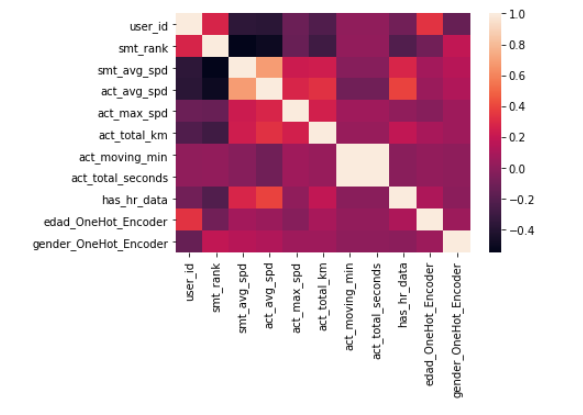


*Ilustración:Diagrama de caja – gender\_OneHot\_Encoder*

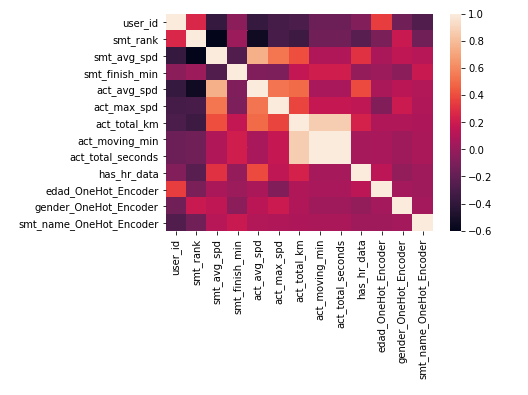
 

*Ilustración:Diagrama de cajas. Izq. Antes de quitar Outliers. Der. Después de quitar los más significativos.*

Podemos ver las correlaciones antes de quitar los outliers y si varía posteriormente. Vemos un mapa de calor de antes y después.



*Ilustración: Mapa de calor de correlaciones. Antes de quitar Outliers*



*Ilustración: Mapa de calor de correlaciones. Después de quitar Outliers*

En un primer golpe de vista vemos, que las variables con mayor relación en la matriz tienen que ver con la velocidad (tanto en segmento como en actividad) con el tiempo a la hora de finalizar el segmento y el tiempo de actividad total.

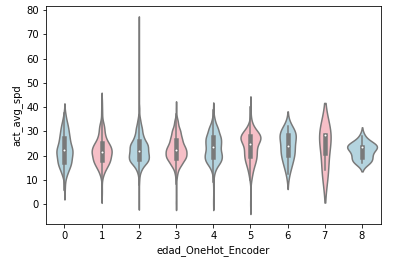
Vemos que los mapas de calor varían tras la eliminación de los outliers, las que mayor relación tienen son aquellas que tienen que ver con la velocidad y la distancia y el tiempo.

También hemos visto la relación que tienen todas la variables en gráficas entre sí, con “*pairplot*”. Mejor verlo en el notebook. Adjunto imágenes. La tabla de colores la he hecho de dos modos, una diferenciando por los rangos de edad y otra por género.

**4. Algunas gráficas de visualización**

A continuación vamos a ver unas gráficas para comprender un poco más el dataset.

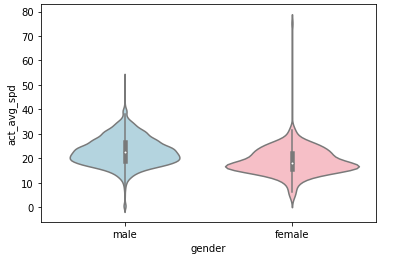
En primer lugar vemos un diagrama de violín donde comparamos en X la edad (con las diferentes categorías creadas con anterioridad) del usuario con la velocidad en Y.



*Ilustración: Diagrama de violín Edad-Velocidad media*

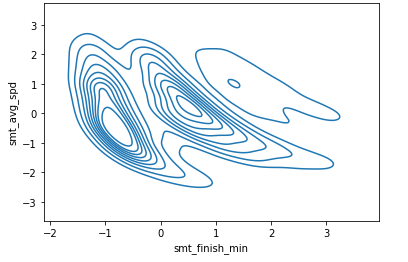
En el siguiente diagrama podemos ver la velocidad que alcanzan de media por actividad con el sexo (utilizando las categorías creadas anteriormente).

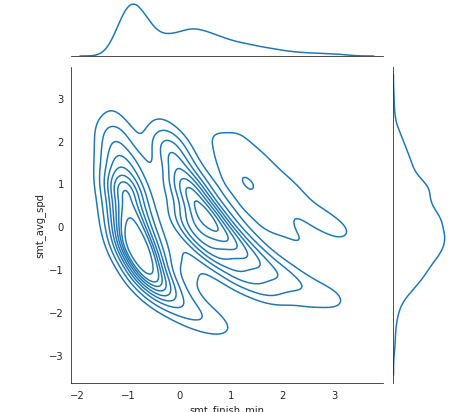
Vemos que las mujeres el número de mujeres (en este dataset, mucho más reducido que el número de hombres) tienen de media una velocidad menor que la de hombres, pero alcanzado la máxima por su parte.

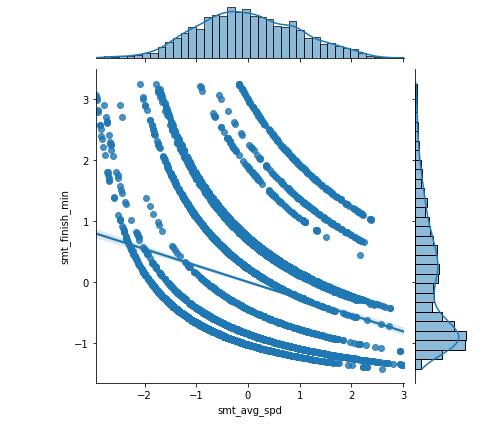


*Ilustración: Diagrama de violín Sexo-Velocidad media*

A continuación, mostraremos los mismos datos con diferentes gráficas. Todas comparando en X el tiempo por segmento y en Y la velocidad por el mismo.







En todas podemos ver de una manera u otra, que como es normal, la velocidad está relacionada con el tiempo. A mayor velocidad menos tiempo. Vemos que la recta lleva una tendencia decreciente.

**5. Estandarización**

Para preparar el data set para un futuro modelo mejor del que hemos intentado hacer, hago la estandarización del mismo.

#### Estandarizado

df\_scale = preprocessing.StandardScaler().fit(df[['user\_id', 'smt\_rank','smt\_avg\_spd','act\_avg\_spd', 'act\_max\_spd', 'act\_total\_km',

'act\_moving\_min', 'act\_total\_seconds', 'edad\_OneHot\_Encoder', 'gender\_OneHot\_Encoder','smt\_name\_OneHot\_Encoder','smt\_finish\_min']])

df\_std = df\_scale.transform(df[['user\_id', 'smt\_rank','smt\_avg\_spd','act\_avg\_spd', 'act\_max\_spd', 'act\_total\_km',

'act\_moving\_min', 'act\_total\_seconds', 'edad\_OneHot\_Encoder', 'gender\_OneHot\_Encoder','smt\_name\_OneHot\_Encoder','smt\_finish\_min']])

type(df\_scale)

sklearn.preprocessing.\_data.StandardScaler

type(df\_std)

numpy.ndarray

print('La media tras la estandarzación:' + '\n'+ str(round(df\_std.mean(),1)))

La media tras la estandarzación:

-0.0

print('La desviación típica tras la estandarzación:' + '\n'+ str(round(df\_std.std(),1)))

La desviación típica tras la estandarzación:

1.0

Como podemos comprobar, ajustamos los datos para que la media sea 0 y la desviación típica esté en 1. En este momento lo dejamos preparado para su utilización posterior

**6. Avance para un modelo de regresión lineal.**

Lo primero de todo, separaremos los datos en el X\_train, X\_test, y\_train e y\_test, para posteriormente aplicar el modelo que creamos conveniente y llegar a alcanzar el objetivo propuesto.

La división que hemos realizado en los datos se ha dividido de la siguiente manera:

X = df[['user\_age\_group', 'user\_id', 'attempt\_date', 'gender', 'smt\_rank','smt\_avg\_spd', 'smt\_name', 'user\_weight\_category',

'act\_title', 'act\_avg\_spd', 'act\_max\_spd', 'act\_total\_km',

'act\_moving\_seconds', 'act\_total\_seconds', 'has\_hr\_data']]

y = df[['smt\_finish\_seconds']]

Barajamos los datos que tenemos y cogemos para el “*X\_train*” un 80% de los datos y para el test el 20%:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 42)

Obteniendo una distribución de los datos que quedará:

Total train features shape: (6292, 15)

Total train target shape: (6292, 1)

Total test features shape: (1573, 15)

Total test target shape: (1573, 1)

Todos los tratamientos que hagamos en “*X\_train*”, se harán posteriormente en “*X\_test*”. En este caso se establece una categoría en función de la columna *'user\_age\_group'*, debido a que sería interesante un futuro para la creación de un modelo de *machine learning*. La propuesta del modelo sería la de predecir qué tiempo realizará en los segmentos registrados dependiendo de la edad del usuario de la app.

Para ir practicando e ir avanzando he intentado crear un modelo de regresión lineal. Pero los resultados no han sido los esperados. Posiblemente porque el tratamiento de datos que he realizado no ha sido del todo acertado. Ante ello me lo quedo pendiente para seguir avanzando a título personal para realizarlo de la mejor manera posible.

**Un modelo de regresión es un modelo matemático que busca determinar la relación entre una variable dependiente (Y), con respecto a otras variables, llamadas explicativas o independientes (X).**

El modelo de regresión se suele utilizar en las Ciencias Sociales con el fin de determinar si existe, o no, relación causal entre una variable dependiente (Y) y un conjunto de otras variables explicativas (X). Asimismo, el modelo busca determinar cuál será el impacto sobre la variable Y ante un cambio en las variables explicativas (X). (<https://economipedia.com/>)

A continuación iré detallando lo aprendido hasta el momento:

En primer lugar creamos un objeto ‘LinearRegression’ que nos ayudará a entrenar y testear el modelo.

lr = LinearRegression()

Seguidamente entrenaremos el modelo con X\_train e y\_train pasándole el objeto creado con anterioridad:

lr.fit(X\_train, y\_train)

LinearRegression()

Puesto que nos indica el intercept\_ y el coef\_ lo sacamos en celdas aparte:

lr.intercept\_

1224.8248971366297

lr.coef\_

array([-1.11722553e-01, -6.45626591e-01, -3.87957277e+01, 7.40027587e+00,

9.94235679e-01, 3.21687988e+00, -8.14383529e-08, -4.88546999e-06,

2.32038072e+01, 1.47152969e+02])

Para seguir avanzando con el modelo, creamos la variable ‘predictions’ que ayudará a obtener las predicciones:

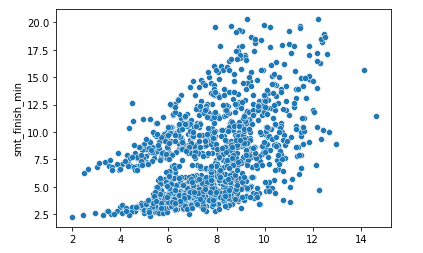
predictions = lr.predict(X\_test)

predictions

array([1001.64448398, 448.11746292, 343.04834549, ..., 353.65614362,

720.74004933, 690.15053766])

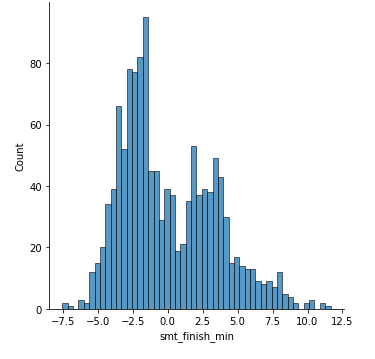
Vamos a pintar los datos de y\_test, con las predicciones que nuestro método ha realizado:



*Ilustración: Relación predicción con y\_test*

Vemos que la distribución de los datos no es la mejor que podríamos tener. Para tener una distribución de los datos correcta debería ser un línea paralela del eje X con el Y.

En la siguiente gráfico de barras mostramos los errores o residuos de nuestro modelo:



*Ilustración: Residuos del modelo*

Vemos que los residuos son bastante elevados, por lo que el modelo no ha ha funcionado correctamente.

Por otro lado, vamos a mostrar:

1. MSE: En estadística, el error cuadrático medio de un estimador mide el promedio de los errores al cuadrado, es decir, la diferencia entre el estimador y lo que se estima (Wikipedia).

2. RMSE: La raíz del error cuadrático medio o raíz de la desviación cuadrática media es una medida de uso frecuente de las diferencias entre los valores predichos por un modelo o un estimador y los valores observados (Wikipedia).

MSE: 145841.56311712408

RMSE: 381.8920830773061

Como podemos ver los errores son bastante altos. Mi modelo no está bien diseñado, como he dicho anteriormente posiblemente sea por el tratamiento de datos inicial ya que no ha sido el mejor que podría haber hecho. También posiblemente el poner variables, a mi parecer innecesarias en X.

A continuación podemos ver que el ‘score’ está muy pero que muy alejado de 1, tanto en Train como en Test, por lo que no ha funcionado ☹.

lr.score(X\_train, y\_train

0.2318844496495227

lr.score(X\_test, y\_test)

0.23148759632158866

Este es mi ejemplo de EDA, no ha salido todo lo bien que yo quería. Pero lo mejoraré por orgullo propio y espero que el resultado sea mucho mejor de lo que me ha salido hasta ahora.