



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

## 2018 Yellow Taxi Trip Data

*Temas selectos de computación*

Equipo 2:

Mejía Alba Lizbeth  
Melo López Alicia Liliana  
Mendoza Castro Luis Gerardo  
Nava Tepozan Osvaldo Daniel  
Roman Jaimes Chistopher  
Sánchez Chavez Alejandra Monserr  
Sánchez Esponosa Luis Fernando

18 de enero de 2021

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Análisis exploratorio de datos</b>	<b>3</b>
2.1. Descripción de los datos . . . . .	3
2.2. Datos ausentes . . . . .	5
2.3. Variables Categóricas . . . . .	6
2.4. Variables Continuas . . . . .	9
2.5. Detección de datos atípicos . . . . .	11
<b>3. Ingeniería de datos</b>	<b>15</b>
<b>4. Diseño de estrella y tablero para Inteligencia de negocios</b>	<b>17</b>
4.1. Extract, Transform and Load (ETL) . . . . .	17
4.2. Cubo OLAP (Modelo de Estrella) . . . . .	17
4.3. Tablero para Inteligencia de Negocios . . . . .	19
<b>5. Modelo Supervisado</b>	<b>22</b>
5.1. Árbol de decisión . . . . .	22
5.2. Red neuronal . . . . .	24
<b>6. Estrategia de Aplicacion de los modelos</b>	<b>26</b>
<b>7. Conclusiones</b>	<b>27</b>
<b>8. Anexo con el código utilizado</b>	<b>28</b>
8.1. Análisis exploratorio . . . . .	28
8.2. Cubo OLAP . . . . .	29
8.3. Funciones AE . . . . .	30
8.4. Muestras . . . . .	31
8.5. Ingeniería y Modelos . . . . .	32

## 1. Introducción

Nueva York es la ciudad más grande en Estados Unidos, y una de las más grandes del mundo, el ritmo de vida de sus ciudadanos es muy acelerado, muchas personas requieren llegar a sus empleos en algún punto de la ciudad y posteriormente regresar a sus hogares, otras personas necesitan acercarse a los grandes comercios para satisfacer sus necesidades. Por estas razones los medios de transporte son cruciales en una ciudad de esta magnitud, y el servicio de taxi no es la excepción, ya que en Nueva York se pueden ver taxis amarillos por todas partes, representan una de las mayores fuentes de empleo, el servicio genera cientos de millones de dólares al año, por ende se requiere tener una cultura de recolectar los datos que se generan día a día a causa de este gran fenómeno.

Con base a la información expuesta en la página de la Taxi and Limousine Commission (TLC), hemos recopilado una serie de archivos, el principal es una tabla con 112,234,626 registros de viajes llevados a cabo en la ciudad de Nueva York por los taxis amarillos (Yellow Taxi) en el año 2018, los demás archivos contienen complementos a esta tabla, tal como catálogos, geometrías, etc. Esta tabla contiene detalles de cada uno de los viajes, como el tiempo y el lugar de partida y de arribo, la distancia recorrida en millas, el medio con el que se pagó el viaje, el monto total pagado por cada viaje descompuesto en sus componentes principales, detalles que justifican las tarifas cobradas e incluso la compañía proveedora del registro del viaje.

El objetivo de este análisis es conocer y evaluar el estado en el que están los datos, qué cantidad de registros tienen datos ausentes, atípicos o incoherentes, para posteriormente darles una limpieza y un tratamiento adecuado. Una vez hecho lo anterior, se omiten algunas columnas que no tienen mucha relevancia para el análisis y con las columnas restantes se busca formar una estructura analítica que nos reporte el volumen de viajes y los hechos financieros aunados a los mismos.

Después de estructurar los datos, se necesita una herramienta de consulta, con los principales indicadores de negocio, que tenga la capacidad de darnos un resumen del comportamiento de los viajes en puntos del tiempo y puntos geográficos específicos. Esto se consigue a través de un tablero de inteligencia de negocios, que facilita la visualización de los datos con gráficas de distintos tipos.

Como parte final del estudio, se busca predecir el volumen de viajes en cualquier hora del año, los propósitos de esto se plantean más adelante en el texto, se construye una matriz predicción con ventanas de tiempo, usando los viajes en horas anteriores. Una vez que se tiene esta matriz se ocupa para entrenar un modelo de árbol de decisión y una red neuronal que son nuestras herramientas de predicción. Con dichas herramientas haremos predicciones para una solución a nuestro problema de negocio.

## 2. Análisis exploratorio de datos

En esta sección se caracterizan las variables con las que cuenta la tabla de viajes, la cual al tener un número tan grande de registros fue cargada al software mediante chunks. Lo primero que se busca es ver si hay registros nulos en alguna de las columnas de la tabla, las cuales para efectos de este texto llamaremos indistintamente variables o campos, de darse el caso, se listan las variables que contienen este tipo de registros y se estudia con qué porcentaje de estos registros cuenta cada una. Una vez hecho lo anterior, se procede a clasificar las variables, identificando cuáles de estas son variables categóricas y cuáles son variables continuas. Hecha esta distinción, de cada variable se obtienen su frecuencia o su densidad asociada según sea el caso. Se sigue con la búsqueda de datos atípicos o datos que son evidentemente erróneos contenidos en los campos. Finalmente, en el caso de las variables continuas, se busca si hay una relación lineal entre algunas de estas, esto mediante el uso del coeficiente de correlación, una herramienta estadística muy sencilla pero con grandes implicaciones a la hora de estudiar múltiples variables. En el caso de las variables discretas, solo se muestran algunas gráficas en las que se puede intuir claramente que tienen cierto impacto en el valor que toma alguna otra variable ya sea discreta o continua.

### 2.1. Descripción de los datos

Antes de comenzar el análisis exploratorio se da una descripción tanto del significado como de los tipos de dato que contiene cada uno de los campos.

- **Vendor:** Es la compañía que proveé el registro de la información sobre el viaje.
- **tpep\_pickup\_datetime:** Nos indica la fecha y la hora en que se activó el medidor.
- **tpep\_dropoff\_datetime:** Nos indica la fecha y la hora en que se desactivó el medidor.
- **passenger\_count:** Nos indica el número de pasajeros que se encuentran dentro del vehículo, dicho valor debe ser ingresado por el conductor. En un taxi convencional, el número máximo de pasajeros que pueden ocupar la misma cabina son cuatro, aun que hay taxis más grandes parecidos a minivans con una capacidad de hasta cinco pasajeros, cada pasajero adulto si así lo desea puede llevar a un niño en su regazo, además si el espacio en la cabina se agota o cierto pasajero tiene problemas para entrar en ella, este tiene derecho a ir en el asiento del copiloto, a un lado del conductor.
- **trip\_distance:** Nos indica las distancia del viaje (en millas) reportada por el taxímetro.
- **PULocation:** Nos indica la zona de la Comisión de Taxis y Limusinas de la ciudad de NY (TLC por sus siglas en inglés) en donde se activó el taxímetro.
- **DOLocation:** Nos indica la zona de la Comisión de Taxis y Limusinas de la ciudad de NY (TLC por sus siglas en inglés) en donde se desactivó el taxímetro.
- **Ratecode:** Es un código que afecta a la tarifa cobrada al término de un viaje.
  1. *Standard rate*: Suele asignarse a viajes que fueron tomados en la calle y por esa razón son estos los más económicos.
  2. *JFK, Newark y Nassau or Westchester*: Se refieren a viajes especiales pedidos a la salida de alguno de los aeropuertos disponibles en Nueva York
  3. *Negotiated fare*: Es un código para casos en los que se llega a cierto acuerdo con el pago o cuando hay disconformidad del cliente.
  4. *Group ride*: Es una tarifa especial a un viaje en grupo.
- **store\_and\_fwd\_flag:** Es un indicador binario que nos informa en el caso de tomar el valor “Y” que la información sobre el viaje fue subida al servidor tiempo después de que acabara el viaje, debido a que en el momento de este el conductor no tenía conexión a la red.

- **payment\_type:** Nos indica el método con el cual el usuario pagó el viaje.
  1. *Credit card*: Tarjeta de Crédito.
  2. *Cash*: Efectivo.
  3. *No charge*: Para casos en los que no se cobra tarifa, por ejemplo cuando un adulto mayor cuenta con una suscripción especial que lo abstiene de pagar por el viaje.
  4. *Dispute*: Cuando hay un altercado entre el pasajero y el conductor, o una simple disconformidad, se puede llegar a un acuerdo con el pago del viaje.
  5. *Unknown*: Método de pago desconocido.
  6. *Voided trip*: Viaje anulado.
- **fare\_amount:** Nos indica la tarifa que se paga por el viaje, esta es calculada de acuerdo al tiempo y distancia del viaje.
- **extra:** Nos indica los cargos extras que se hacen a la tarifa estos generalmente son de \$0.5 o \$1.0 por viaje en horas pico o viajes durante la noche respectivamente.
- **mta\_tax :** Nos indica el impuesto que se paga a Metropolitan Transport Authority (MTA) de \$0.50 por circular un vehículo privado durante las horas pico en NY.
- **tip\_amount :** Nos indica el monto de la propina que el usuario asigna, este monto se carga automáticamente a la tarjeta de crédito. Las propinas en efectivo no están incluidas.
- **tolls\_amount :** Nos indica el importe total de todos los peajes pagados en el viaje.
- **improvement\_surcharge :** Es un cargo automático por mejora del servicio de taxis, se cobra al dar el banderazo inicial del viaje. Comenzó a aplicarse en 2015 con \$0.30.
- **total\_amount :** Es el monto total cobrado por el viaje. No incluye propinas en efectivo.

Column	Dtype
0 tpep_pickup_datetime	datetime64[ns]
1 tpep_dropoff_datetime	datetime64[ns]
2 passenger_count	int64
3 trip_distance	float64
4 store_and_fwd_flag	object
5 fare_amount	float64
6 extra	float64
7 mta_tax	float64
8 tip_amount	float64
9 tolls_amount	float64
10 improvement_surcharge	float64
11 total_amount	float64
12 Vendor	object
13 Ratecode	object
14 payment_type	object
15 PULocation	object
16 DOLocation	object

Cuadro 1: Tipo de dato

## 2.2. Datos ausentes

Las variables que presentan datos ausentes son las de *Vendor*, *Ratecode*, *PULocation* y *DOLocation*, siendo esta última la que contiene el mayor porcentaje de este tipo de datos entre las cuatro, con un 1.6539 % de datos faltantes, seguida de *PULocation* con 1.6268 %, *Vendor* con 0.4181 % y finalmente *Ratecode* con 0.0047 %. En seguida se muestra una representación gráfica de la proporción de datos ausentes.

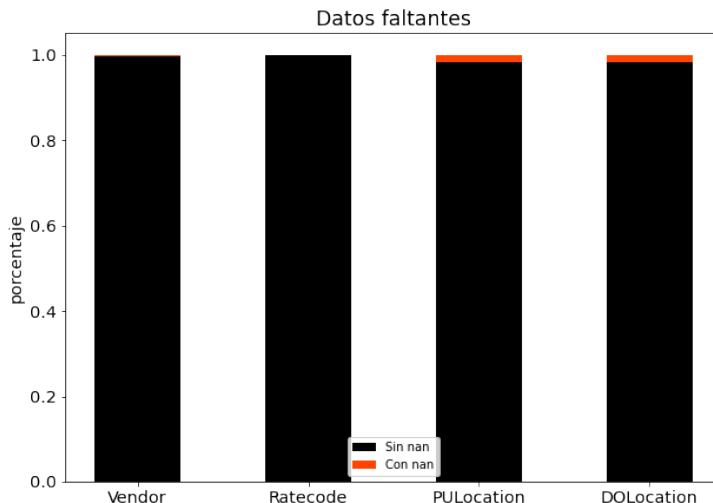


Figura 1: Datos faltantes

Podremos observar de la Figura 1 que los datos faltantes son "insignificantes", pues hay una cantidad muy reducida de estos en comparación con los datos no nulos. Podríamos eliminar los registros que contienen a este tipo de datos pero en este caso no lo haremos por fines de nuestros estudio, y por los cuales, de cada fila lo que nos interesa es la variable *tpep\_pickup\_datetime* (tiempo de activación del medidor una vez que un viaje ha iniciado) en la cual no tenemos ninguna observación faltante.

### 2.3. Variables Categóricas

Las variables categóricas son aquellas que toman una cantidad de valores finito, dichos valores pueden o no tener un orden, las variables categóricas dividen las observaciones de cierto evento en grupos bien definidos que cumplen algún criterio o que tienen una característica específica. Entre nuestras variables, identificamos como categóricas a *store\_and\_fwd\_flag*, *Vendor*, *passenger\_count*, *payment\_type*, *PULocation* y *DOLocation*. Una vez identificadas podemos hacer un conteo de la cantidad de valores distintos que pueden tomar cada una de nuestras variables.

Variable	no. valores que toma
0 Ratecode	6
1 store_and_fwd_flag	2
2 Vendor	2
3 passenger_count	12
4 payment_type	5
5 PULocation	260
6 DOLocation	260

Cuadro 2: Cantidad de valores que pueden tomar las variables categóricas

Como se muestra en el *Cuadro 2*, la variable *Ratecode* toma los seis valores que se muestran en la información inicial sobre la misma, *store\_and\_fwd\_flag* toma solo dos valores (*Y* y *N*), *Vendor* toma solo dos valores, que son las dos compañías que participan en el registro de la información sobre cada viaje, la variable *passenger\_count* toma 12 valores, de los cuales algunos seguramente son erróneos dado el número máximo de pasajeros que pueden ocupar la misma cabina, en el caso de *payment\_type*, esta solo toma 5 valores de los 6 que se esperaría según la información inicial, esto ocurre debido a que ningún registro en esta columna es *Voided trip*, finalmente las variables *PULocation* y *DOLocation* toman ambas 260 valores distintos, correspondientes a la zona de taxis en la que inicio y en la que concluyó el viaje respectivamente.

## Frecuencia de las Variables categóricas

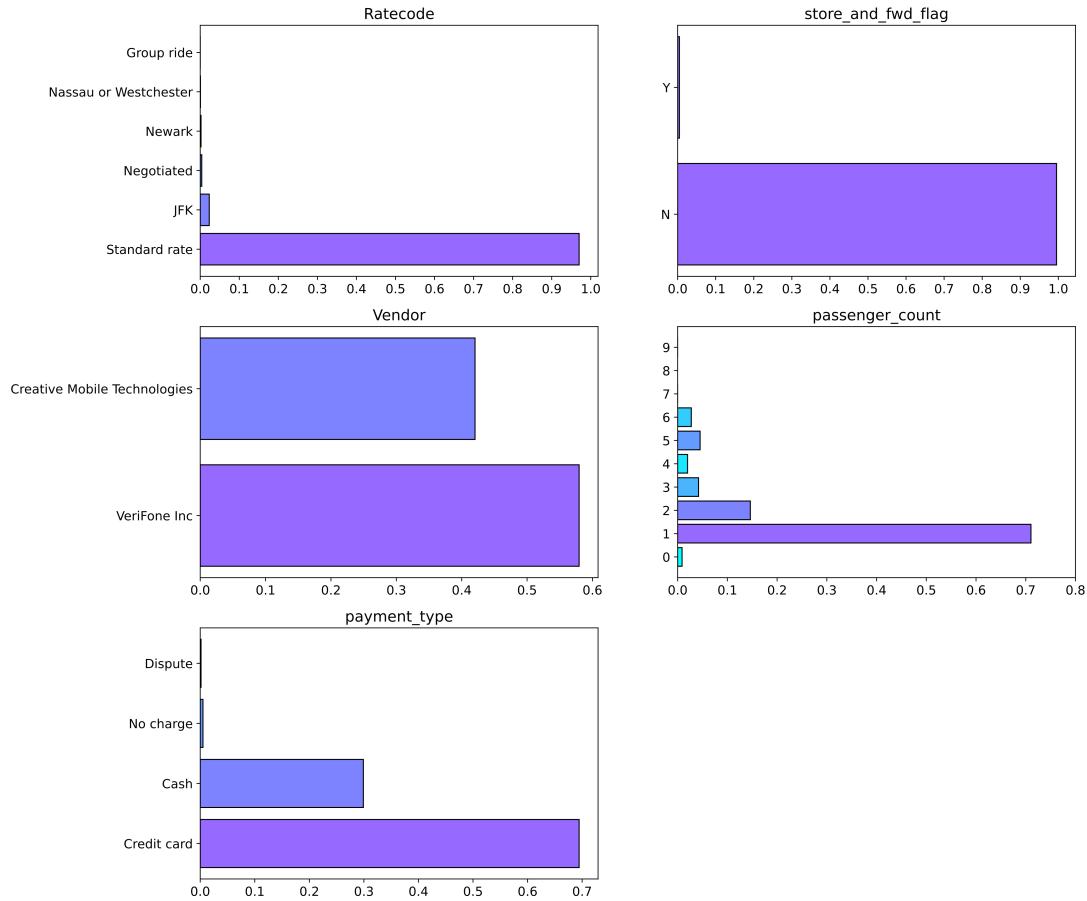


Figura 2: Frecuencia de variables categóricas

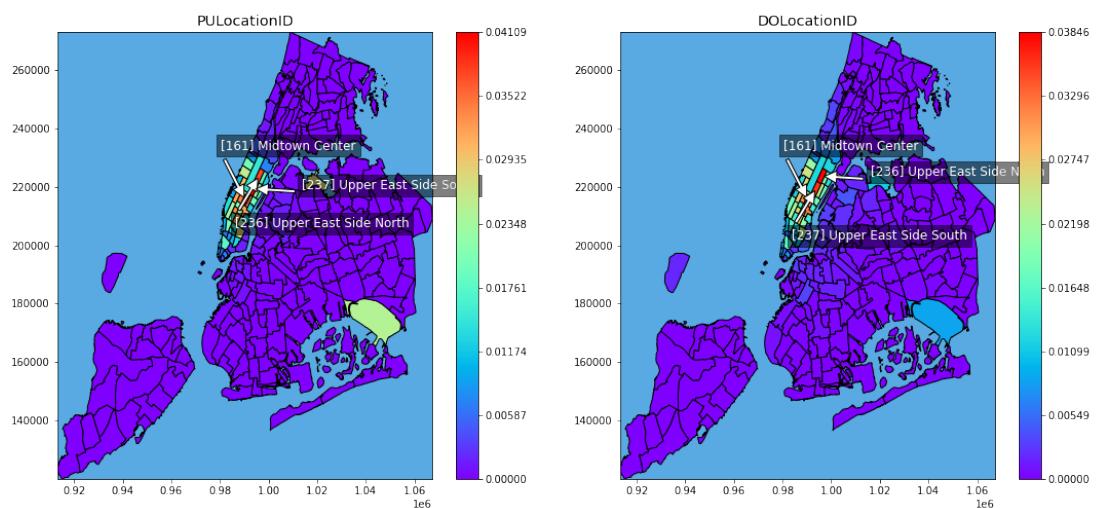


Figura 3: Frecuencia de variables categóricas de tipo lugar

En las figuras anteriores (2 y 3) tenemos la frecuencia de cada una de las variables categóricas, podemos notar que en el caso del campo *Ratecode* la mayoría de los valores que se tomaron son *Standard\_rate*. En la variable *store\_and\_fwd\_flag* los valores se concentran en la bandera *N*, indicando que casi todos los viajes fueron registrados en el servidor al momento, dicho de otra forma, los conductores de taxis raramente se quedaron sin conexión a la red. Hablando de compañías que registran los viajes la que predomina es *VeriFone Inc.* En el campo de tipo de pago, la mayor parte de la frecuencia la acumulan (*Credit Card* y *Cash*). Las variables *PULocation* y *DOLocation* tienen una frecuencia muy similar, la mayoría de viajes circularon en los alrededores del distrito de Manhattan, y poco a poco la frecuencia en las observaciones comienza a bajar conforme nos alejamos de dicho distrito.

	<b>LocationID</b>	<b>PUcount</b>	<b>DOcount</b>	<b>TOTALcount</b>	<b>zone</b>	<b>borough</b>
<b>0</b>	237	4629205	4047374	8676579	Upper East Side South	Manhattan
<b>1</b>	161	4317981	4250577	8568558	Midtown Center	Manhattan
<b>2</b>	236	4203814	4328633	8532447	Upper East Side North	Manhattan
<b>3</b>	162	3944764	3448376	7393140	Midtown East	Manhattan
<b>4</b>	230	3821688	3471901	7293589	Times Sq/Theatre District	Manhattan

Figura 4: Top 5 de zonas con mayor cantidad de inicios de viajes

	<b>LocationID</b>	<b>PUcount</b>	<b>DOcount</b>	<b>TOTALcount</b>	<b>zone</b>	<b>borough</b>
<b>0</b>	236	4203814	4328633	8532447	Upper East Side North	Manhattan
<b>1</b>	161	4317981	4250577	8568558	Midtown Center	Manhattan
<b>2</b>	237	4629205	4047374	8676579	Upper East Side South	Manhattan
<b>3</b>	170	3620862	3588105	7208967	Murray Hill	Manhattan
<b>4</b>	230	3821688	3471901	7293589	Times Sq/Theatre District	Manhattan

Figura 5: Top 5 de zonas con mayor cantidad de términos de viajes.

En el campo *passenger\_count* la mayoría de veces se toma el valor 1, el cual indica que un viaje llevaba un pasajero en total. Esto se puede observar más claramente si graficamos la frecuencia acumulada de esta variable.

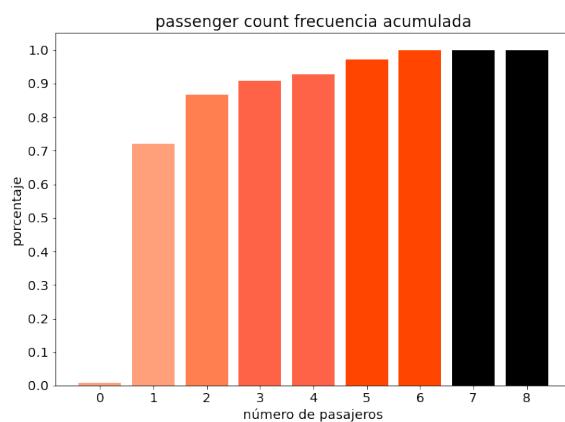


Figura 6: Frecuencia acomulada *passenger\_count*

## 2.4. Variables Continuas

Las variables continuas son aquellas que pueden tomar un número infinito no numerable de valores, por lo que de lógicamente primero clasificamos como de este tipo a aquellas columnas que contienen a los hechos. Sindo así, como variables continuas seleccionamos a *tpep\_pickup\_datetime*, *tpep\_dropoff\_datetime*, *trip\_distance*, *fare\_amount*, *extra*, *mta\_tax*, *tip\_amount*, *tolls\_amount*, *total\_amount*, e *improvement\_surcharge*.

### Distribución de las variables continuas

De inicio a las variables *tpep\_pickup\_datetime*, *tpep\_dropoff\_datetime* les damos un trato un tanto distinto, debido a que sus registros son del tipo *datetime*, decidimos extraer de cada registro el día del año en el que se dió el viaje, de esta forma pudimos captar una aproximación a la distribución y a la distribución acumulativa de estas variables Figuras (7 y 8). Podemos notar que extrañamente hay un volumen muy elevado de viajes entre los días 50 y 100 correspondiente al mes de Marzo.

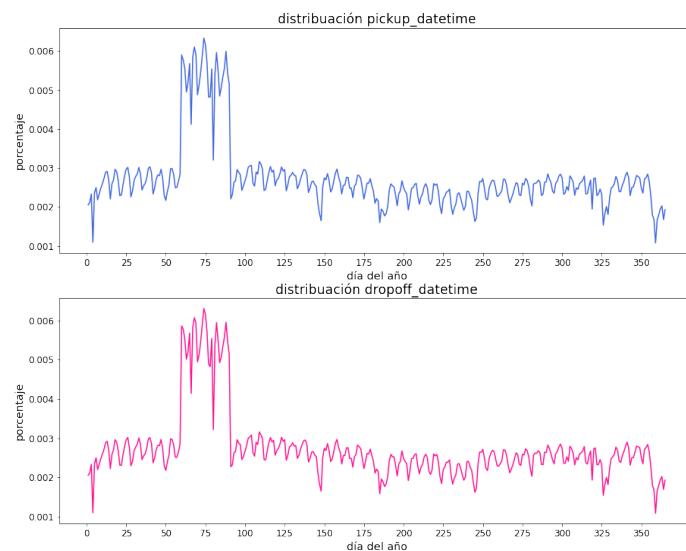


Figura 7: Aproximación a la distribución de las variables de tiempo

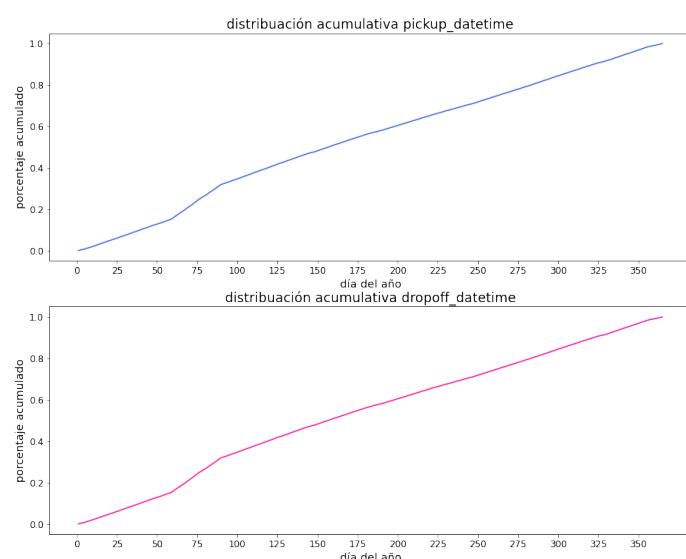
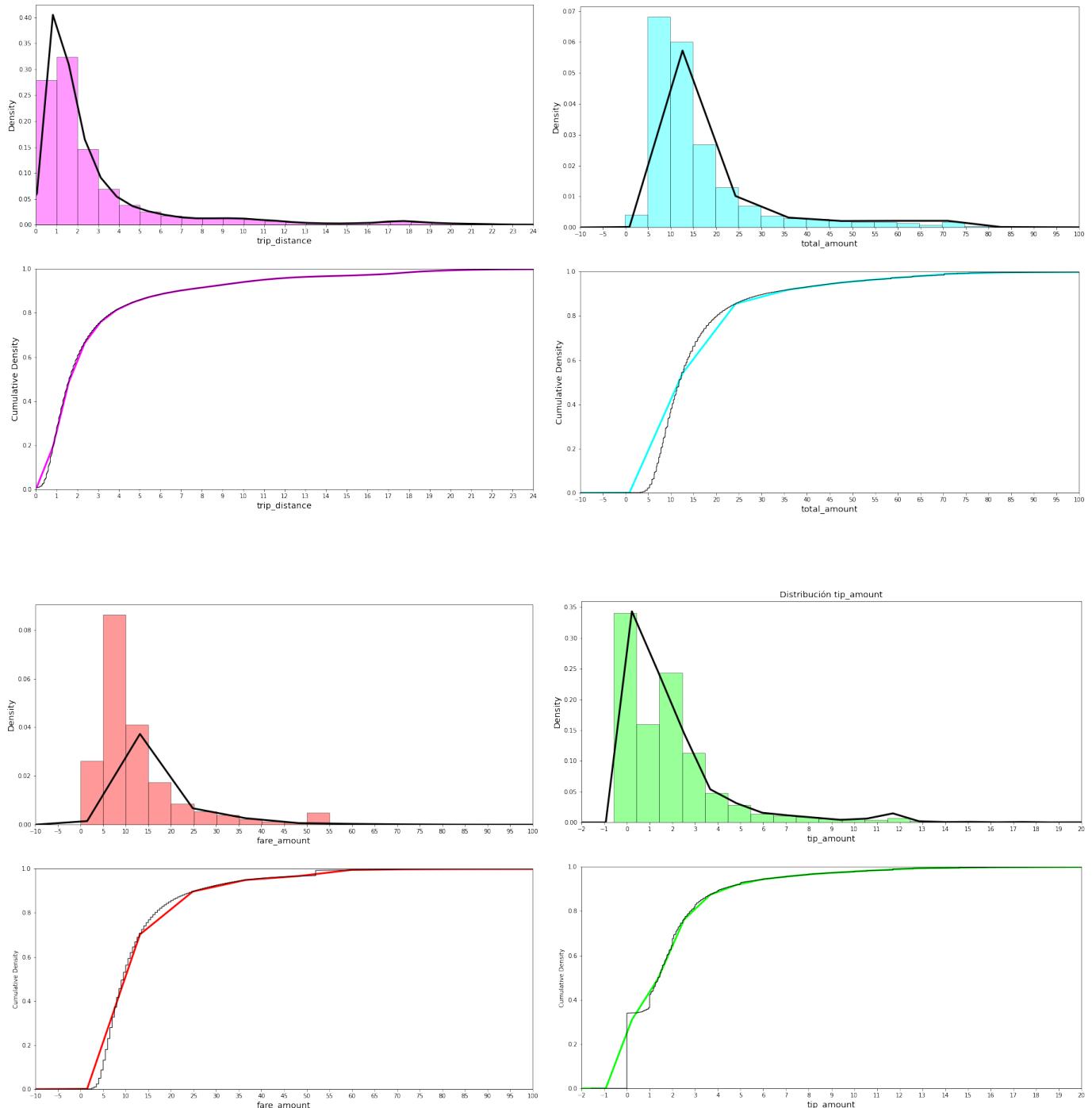


Figura 8: Aproximación a la distribución acomulativa de las variables de tiempo

El resto de variables continuas contienen hechos financieros a excepción de *trip\_distance* que contiene los hechos de distancia del viaje. De estas variables se obtuvo un histograma al cual se le ajustó una curva de densidad probable mediante *kernels Gaussianos*. Con base a esto se obtuvo una función de distribución probable. Abajo se ilustran la variables *trip\_distance* y *total\_amount* además de las variables *fare\_amount* y *tip\_amount* que son las de mayor peso en el monto final pagado por un viaje.



## 2.5. Detección de datos atípicos

Un dato atípico (*outlier*) es un dato que está muy alejado numéricamente del resto de observaciones de una variable aleatoria, estos generan conclusiones estadísticas engañosas debido a que repercuten directamente en la media y la desviación estándar de un conjunto de datos, y cabe mencionar que este par de estadísticos son muy sensibles a datos de este tipo. Estos datos casi siempre representan un porcentaje pequeño del total y bien pueden ser datos que si ocurren pero con una probabilidad muy baja o datos incorrectos a causa de un error de medición o de registro. Es importante darle un tratamiento a los datos atípicos porque aumentan mucho la percepción que se tiene sobre la variabilidad de un fenómeno aleatorio.

Como primer paso para la detección de datos atípicos se toma una muestra de un millón de registros de nuestra tabla de viajes y se procede a dar una descripción general, primero de las variables continuas excluyendo aquellas de tiempo. La descripción muestra la media, desviación estándar, el mínimo y el máximo de las observaciones de cada variable y los cuartiles 25 %, 50 % y 75 %.

	<b>trip_distance</b>	<b>fare_amount</b>	<b>extra</b>	<b>mta_tax</b>	<b>tip_amount</b>	<b>tolls_amount</b>	<b>improvement_surcharge</b>	<b>total_amount</b>
<b>count</b>	1000000.000	1000000.000	1000000.000	1000000.000	1000000.000	1000000.000	1000000.000	1000000.000
<b>mean</b>	2.930	12.981	0.331	0.497	1.874	0.346	0.300	16.333
<b>std</b>	3.795	11.938	0.463	0.044	2.625	1.810	0.016	14.627
<b>min</b>	0.000	-300.000	-4.500	-0.500	-1.580	-6.000	-0.300	-300.300
<b>25%</b>	0.960	6.500	0.000	0.500	0.000	0.000	0.300	8.500
<b>50%</b>	1.600	9.500	0.000	0.500	1.400	0.000	0.300	11.800
<b>75%</b>	3.000	14.500	0.500	0.500	2.450	0.000	0.300	17.800
<b>max</b>	151.700	2020.200	18.000	15.000	226.000	765.760	1.000	2020.500

Sólo visualizando la información en la tabla anterior, para el caso de *trip\_distance* no podemos dar un argumento claro para poner un límite en el número máximo de millas de distancia de un viaje, aunque un viaje de alrededor de 150 millas no suena usual, el mínimo de las distancias es cero, por lo que ahí también puede haber un problema, ya que para que un viaje pueda ser considerado como tal, intuitivamente tiene que tener una distancia mínima mayor a cero. En el campo *fare\_amount* es claro que hay datos atípicos, aunque aún no damos un criterio claro para conocerlos, no podemos asegurar que no haya viajes con tarifa negativa, ya que al haber posibilidad de negociar el cobro de un viaje por una inconformidad podría darse el caso, pero tener un mínimo de -\$300 dólares como cota mínima no parece correcto, por otro lado, un máximo de \$ 2020.2 como tarifa también se ve exagerado cuando la media se acerca a 13 dólares. En los campos de *tip\_amount* y *tolls\_amount* los máximos también se ven anormales, ya que la media y desviación estándar son muy bajas como para que haya valores de 226 y 765.76 en uno y otro.

En la siguiente gráfica de dispersión se puede visualizar de manera rápida que existen valores lejanos a la mayoría de observaciones.

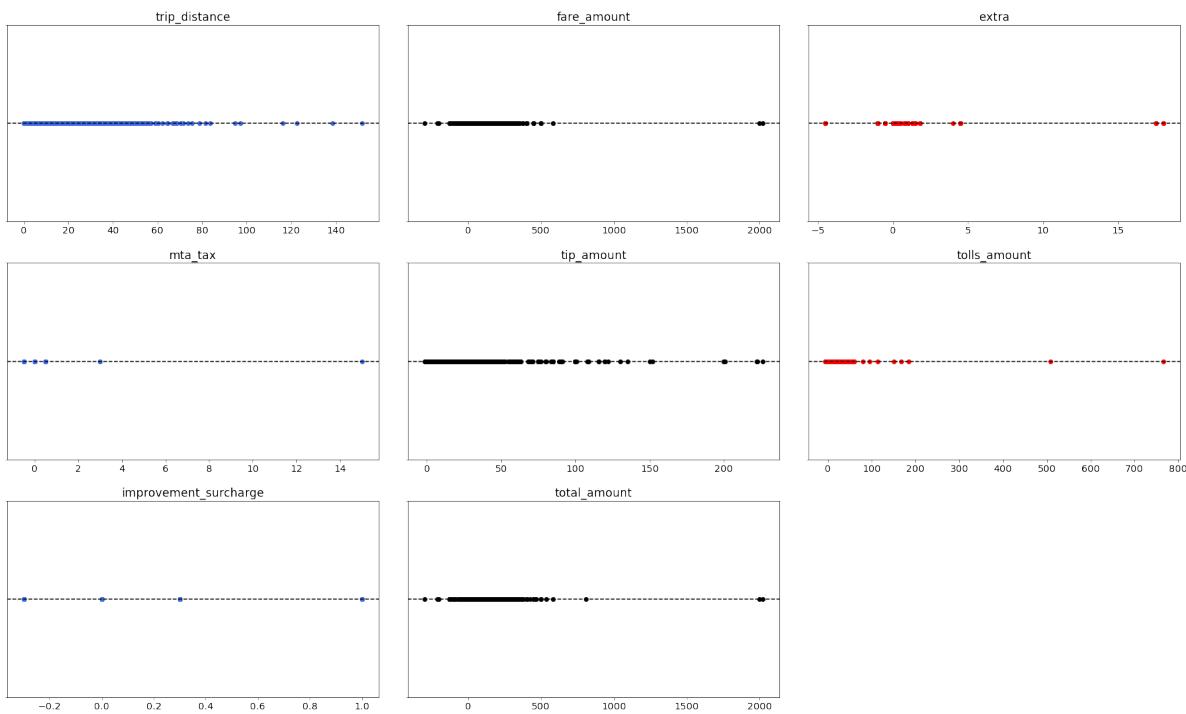


Figura 9: Gráficas de dispersión de los datos

### Criterio Boxplot ajustado

El criterio del boxplot ajustado es un método para discriminación de datos atípicos que se basa en el método boxplot tradicional, el cual es un método muy sencillo de aplicar y a la vez muy robusto, por lo que es muy usado a la hora de buscar atípicos. Se basa en el primer y tercer cuartil de un conjunto de datos, y en la mediana de los mismos, de manera gráfica propone unas barreras inferior y superior a partir de los cuales, los datos pueden considerarse atípicos, este método no tiene una base estadística muy sólida, sin embargo es de los pocos que pueden aplicarse a datos que no se suponen provenir de una distribución normal o una distribución simétrica. Cuanado no existe una simetría en los datos, muchos más de estos serán catalogados erróneamente como atípicos, en este punto entra el método boxplot ajustado, el cual se basa en un estadístico llamado *medcouple*, que a su vez se basa en la mediana, esto debido a que la mediana es un estadístico robusto, es decir, su valor no se ve muy influenciado ante la presencia de atípicos, como si lo hacen la media y la desviación estándar; Una vez calculado el *medcouple*, este tendrá dos propósitos, el primero se basa en su valor, el cual ese encuentra entre -1 y 1, y según sea el caso, nos indica si la asimetría de los datos es positiva, negativa o nula; El segundo propósito de este estadístico es participar en la construcción de las barreras adecuadas para discriminación de datos cuando existe una asimetría ya sea positiva o negativa.

Haciendo uso del boxplot ajustado, en el *Cuadro 3* de abajo, para algunas de las variables se muestran los intervalos fuera de los cuales los datos se consideran atípicos y el porcentaje de este tipo de datos que fue detectado, de momento sólo listamos la variable *trip\_distance* y las variables que más peso tienen en el monto total del viaje.

Variable	Intervalo	Porcentaje de atípicos
trip_distance	[0,21.5]	0.341 %
fare_amount	[2.5,80]	0.244 %
tip_amount	[0,13]	0.631 %
tolls_amount	[0,5.77]	0.330 %
total_amount	[3.3,84]	0.432 %

Cuadro 3

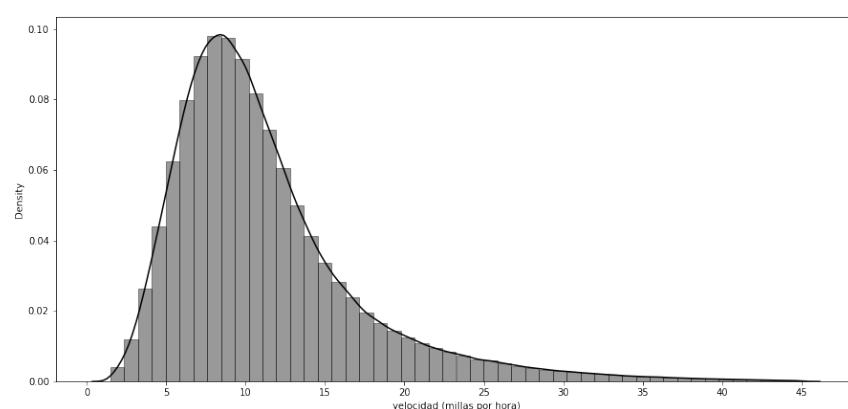
Para construir los intervalos en el *Cuadro 3*, además de usar el criterio del boxplot ajustado se consideraron algunos aspectos que ofrece la descripción inicial de las variables, un viaje puede registrar una distancia cero o apenas mayor en el caso de que el cliente se arrepienta de tomar el viaje, eso sí, este tendrá que pagar la tarifa mínima que es de \$ 2.5, y se cobra tan solo el medidor comience a trabajar, las propinas y los peajes no pueden tomar valores menores que cero, y dado que el *mta\_tax* y el *improvement\_surcharge* son montos obligatorios, el monto total mínimo por un viaje es de \$3.3.

Después de analizar más a fondo las variables *mta.tax*, *extra* y *improvement\_surcharge* se puede notar que en realidad no son variables continuas porque solo pueden tomar valores en un conjunto finito; *mta\_tax* tiene un valor constante de \$0.5 e *improvement\_surcharge* de igual manera es constante \$0.3. Finalmente, *extra* solo puede tomar valores \$ 0.3, \$ 0.5 y \$ 1.0. Una vez entendido esto, a los valores en estos campos que sean distintos a los mencionados se les considera erróneos y por tanto atípicos, en el siguiente cuadro (*Cuadro 4*) se muestra el porcentaje de este tipo de valores.

Variable	Valores que toma	Porcentaje de atípicos
extra	0.0, 0.5 y 1.0	0.5103 %
mta_tax	0.5	0.4615 %
improvemente_sorcharget	0.3	0.0942 %

Cuadro 4

Finalmente, con las variables *tpep\_pickup\_datetime* y *tpep\_dropoff\_datetime* no hay un criterio directo para conocer el porcentaje de atípicos, por lo que se puede optar por usar un campo auxiliar con la velocidad promedio del viaje, esta se calcula dividiendo el campo *trip\_distance* entre la resta de las dos variables mencionadas. Una vez que se tiene esta columna auxiliar se buscan sus barreras inferior y superior, ya que por ejemplo, si hubiera un viaje con una velocidad promedio exageradamente alta, esto reportaría que la distancia de viaje fue recorrida en un tiempo muy pequeño, y una velocidad cercana a cero reportaría que la distancia del viaje fue recorrida en un periodo de tiempo muy grande. Al campo de velocidad promedio le aplicamos el criterio boxplot ajustado y este discrimina como atípicos a viajes con una velocidad promedio menor a 1.5 mph o mayor a 45 mph, con lo que los campos *tpep\_pickup\_datetime* y *tpep\_dropoff\_datetime* contienen entre los dos un 1.157 % de valores atípicos.



## Valores atípicos en las variables categóricas

En el campo *passenger\_count* tenemos viajes desde cero pasajeros, aunque estos son casos válidos según el artículo “*What Happens When NYC’s Taxis Have No Passengers*”, *John Saul, 2020*, representan un porcentaje apenas del 0.893 % de la totalidad de viajes por lo que los consideramos atípicos, a estos les sumamos los viajes con un número de pasajeros mayor o igual que 7, con lo que nos resultan un total de 0.8949 % de datos atípicos en este campo.

## Correlación entre variables continuas

En la *Figura 10* se muestra en una escala de colores entre azul y verde la correlación entre las variables continuas, excluyendo las de tiempo. Como se puede intuir, el campo *fare\_amount* tiene una fuerte correlación positiva con *total\_amount* ya que el primero determina la mayor parte del valor que tomará el segundo, después le siguen *tip\_amount* y *tolls\_amount* que como habíamos dicho en ocasiones anteriores también tienen un gran peso en el monto final pagado. El campo *trip\_distance* tiene una relación lineal directa con todos los campos de hechos financieros recien mencionados, a mayor distancia recorrida mayor tarifa, más peajes y mejores propinas.

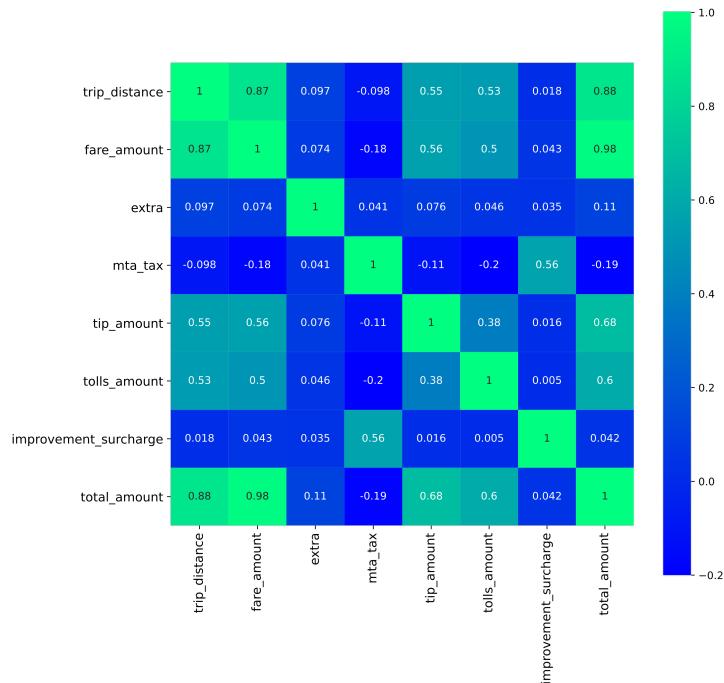


Figura 10: Correlación de variables continuas sin variables de tiempo

### 3. Ingeniería de datos

En esta sección nuestro objetivo es poder predecir el volumen de viajes de la siguiente hora donde tomamos como *unidad muestral* al volumen de viajes por hora. Como nota, decidimos cargar todos los datos (sin muestra), esto nos llevó al menos 5 minutos en cargar y agregar dichos datos con un código realizado por nosotros. Nuestra información es de 8760 registros o *unidades muestrales* ya que dicha información es de un año.

Para lograr nuestro objetivo ocuparemos los datos del pasado para hacer nuestra predicción, esto nos lleva a proponer la matriz de variables explicativas ( $\mathcal{X}$ ) como los volúmenes de viajes de las  $n$  horas anteriores y la variable objetivo ( $y$ ) como el volumen de viajes de la siguiente hora donde se decide que  $n = 24$  puesto que esto nos proporciona la información completa del día anterior y arroja un "buen" score.

La matriz  $\mathcal{X}$  tiene que tener 25 columnas, 24 para las horas anteriores y una adicional con el id (*idh*) de la hora en que se quiere predecir. Entonces para obtener lo anterior se cargaron todas las observaciones (sin muestra) de la columna *tpep\_pickup\_datetime* agregando por fecha y hora el número de viajes y obteniendo los resultados de la *Figura 11*.

	viajes	idh
0	16783	0
1	19061	1
2	16601	2
3	12631	3
4	8740	4
...	...	...
8755	14876	8755
8756	14435	8756
8757	14117	8757
8758	10741	8758
8759	8642	8759

Figura 11: Datos agregados

La cual tiene 8760 filas ( $365 * 24$ ) y además como nuestro modelo ocupará las 24 horas anteriores para poder predecir, no podremos incluir las primeras 24 horas en la matriz predictora, esto significa que la matriz  $\mathcal{X}$  tendrá dimensiones (8736, 25). Obteniendo la matriz que se muestra en la *Figura 12*.

idh	Ant_24	Ant_23	Ant_22	Ant_21	Ant_20	Ant_19	Ant_18	Ant_17	Ant_16	...	Ant_10	Ant_9	Ant_8	Ant_7	Ant_6	Ant_5	Ant_4	Ant_3	Ant_2	Ant_1
24	16783	19061	16601	12631	8740	4229	3414	3433	3643	...	12415	13277	12552	12759	11821	10458	8357	8852	7906	5411
25	19061	16601	12631	8740	4229	3414	3433	3643	4940	...	13277	12552	12759	11821	10458	8357	8852	7906	5411	3584
26	16601	12631	8740	4229	3414	3433	3643	4940	6974	...	12552	12759	11821	10458	8357	8852	7906	5411	3584	1724
27	12631	8740	4229	3414	3433	3643	4940	6974	9984	...	12759	11821	10458	8357	8852	7906	5411	3584	1724	1205
28	8740	4229	3414	3433	3643	4940	6974	9984	11627	...	11821	10458	8357	8852	7906	5411	3584	1724	1205	918

Figura 12: Matriz  $\mathcal{X}$

Por otro lado para la matriz  $\vec{y}$  solo agregamos por día y hora la suma de los viajes, de esta manera obtenemos la matriz que se observa en la *Figura 13*.

	<b>idh</b>	<b>Y</b>
<b>0</b>	24	3584
<b>1</b>	25	1724
<b>2</b>	26	1205
<b>3</b>	27	918
<b>4</b>	52	1368
...	...	...
<b>8441</b>	8755	14876
<b>8442</b>	8756	14435
<b>8443</b>	8757	14117
<b>8444</b>	8758	10741
<b>8445</b>	8759	8642

Figura 13: Matriz  $\vec{y}$ 

En nuestro análisis exploratorio la matriz  $\mathcal{X}$  no tuvo datos ausente ni variables con varianza nula y se eliminaron el 3.3 % de datos extremos los cuales los estimamos con el método de percentil  $P_{0.1}$ ,  $P_{99.9}$ .

## 4. Diseño de estrella y tablero para Inteligencia de negocios

### 4.1. Extract, Transform and Load (ETL)

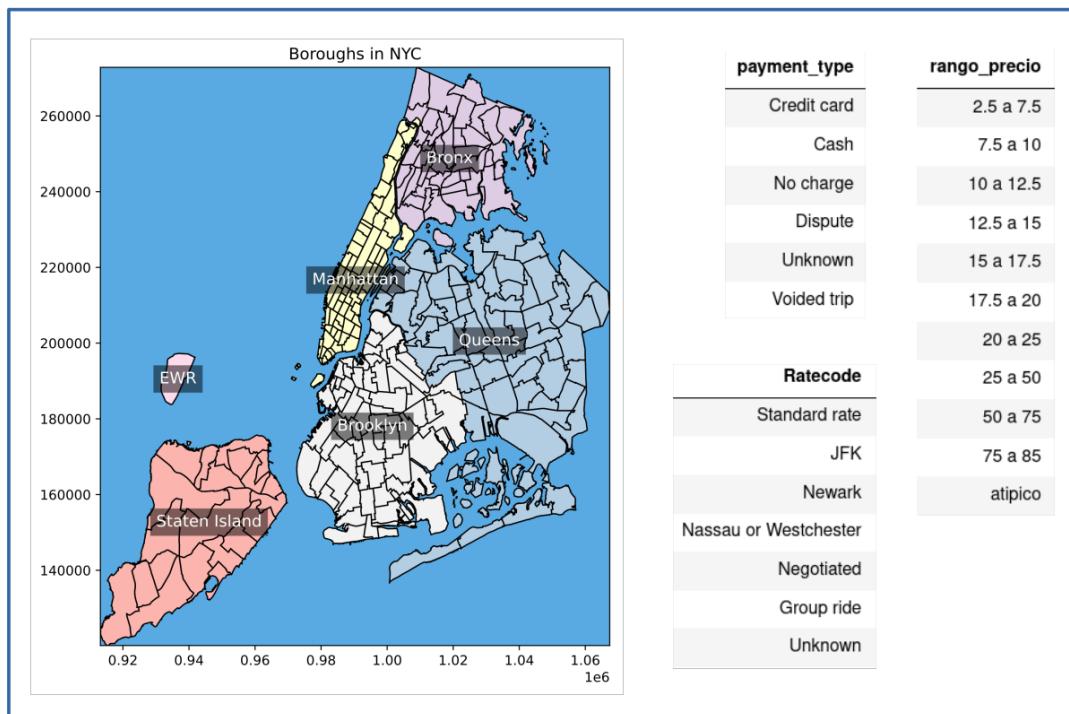
Como primera tarea a través de un módulo que contiene una clase de generación de muestras, nos traemos una representativa de un millón de registros. Una vez cargados los datos, y conociendo detalles de estos (detectados en el análisis exploratorio), procedemos a darles un tratamiento, las variables *tpep\_pickup\_datetime* y *tpep\_dropoff\_datetime* se colocan en formato datetime, también se les rectifica el año 2018, debido a que hay algunos errores de registro con un año distinto y finalmente se redondean sus registros a cuartos de hora. Las variables *mta\_tax*, *extra* y *improvement\_surcharge*, como habíamos mencionado anteriormente en el texto, en realidad solo pueden tomar un número limitado de valores, por lo que estas se rectifican reemplazando los valores erróneos por valores generados con la misma distribución de ese campo. Esta misma idea de reemplazar los valores atípicos en vez de eliminarlos se aplica al resto de variables continuas, por ejemplo, tratando de regresar los a la media, esto lo hacemos por propósitos de nuestro estudio, el cual tiene como enfoque principal conocer el volumen de viajes.

Los registros ausentes en las variables categóricas se reemplazan por “*Unknown*”, categoría que posteriormente se añadirá a los catálogos de cada una de las variables.

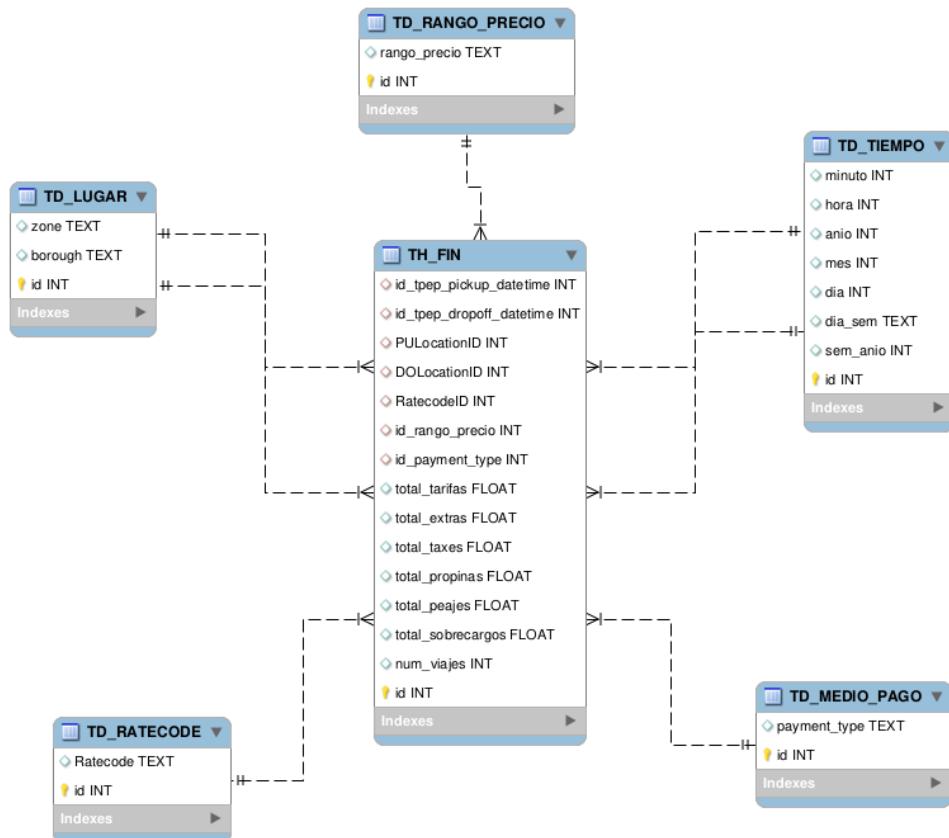
### 4.2. Cubo OLAP (Modelo de Estrella)

Una vez tratados los datos procedemos con la generación de nuestro modelo de estrella. Este modelo contiene los hechos de principal interés en nuestro estudio, estos son todos los viajes en la muestra, y los hechos financieros de estos mismos. Como dimensiones más importantes tenemos *tiempo* y *lugar*, ya que nos interesa conocer el número de viajes y el monto total de estos en cierta zona de taxis en cierto momento en el tiempo, las dimensiones *Medio de Pago* y *Ratecode* se agregan debido a que estos campos tienen una gran influencia en el monto total promedio por un viaje. Y la dimensión *Rango de Precio* ayuda a conocer el comportamiento del fenómeno estudiado en distintos niveles de precios, los niveles de precios existen, y son muy claros, ya que dependiendo del *Ratecode* de un viaje, la tarifa mínima cambia totalmente. Como grado de disagregación máxima en la dimensión *tiempo* tenemos 15 minutos a consecuencia del tratamiento que le dimos a los datos, la dimensión *lugar* puede ser partida en los distritos que conforman a la ciudad de Nueva York, mismos que pueden ser partidos en las zonas de taxis que les pertenecen.

Dimensiones	Hechos
Tiempo	Hechos financieros
Lugar	Número de viajes
Medio de pago	
Ratecode	
Rango de precio	



## Cubo OLAP Modelo Relacional



### 4.3. Tablero para Inteligencia de Negocios

Para el diseño del tablero de inteligencia de negocios utilizamos la herramienta Tableau, con la cual una vez conectada a nuestro modelo de estrella situado en SQL ejecutamos una consulta estratégica al mismo y con base a tablas dinámicas generamos gráficos que ordenan la información de forma elegante y con capacidad de filtrar la y ver los efectos de estos filtros en tiempo real (*Figura 15* ).

Los gráficos que se incluyen son, como parte principal, y en la cual actúan todos los filtros, una serie de tiempo que muestra el comportamiento del número de viajes y la suma del monto total por estos. Se tiene un top de las 5 zonas con mayor número de inicios de viajes y un top de las 5 con mayor número de términos de viajes. Más abajo un par de gráficas de barras con el monto total de los viajes por día de la semana y por hora del día, cada una con su respectiva visualización de los hechos financieros que componen a dicho monto. Un par de gráficas de barras en una escala roji-verde, que indican el promedio del monto total de los viajes según *Ratecode* o *Medio de Pago* respectivamente. Un par de gráficos de pastel con la suma del número de viajes según *Ratecode* o *Medio de Pago* y la suma del monto total de los mismos. Una vez que se consigue la estructura deseada, todos nuestros valores se multiplican por un dactor para que los datos de la muestra escalen a la realidad de nuestros datos. En las dos páginas siguientes se tiene una visualización del tablero, la primera es la vista original, la segunda es una muestra del funcionamiento de los filtros, con el comportamiento se el distrito de Queens, los Lunes en un horario de 13:00 a 15:00 horas, y con un *Ratecode* etándar.

Si cuenta con una cuenta en Tableau puede visualizar el tablero aquí: [https://public.tableau.com/views/TableroNYCTaxi2018/Dashboard1?:language=es&:display\\_count=y&publish=yes&:origin=viz\\_share\\_link](https://public.tableau.com/views/TableroNYCTaxi2018/Dashboard1?:language=es&:display_count=y&publish=yes&:origin=viz_share_link)

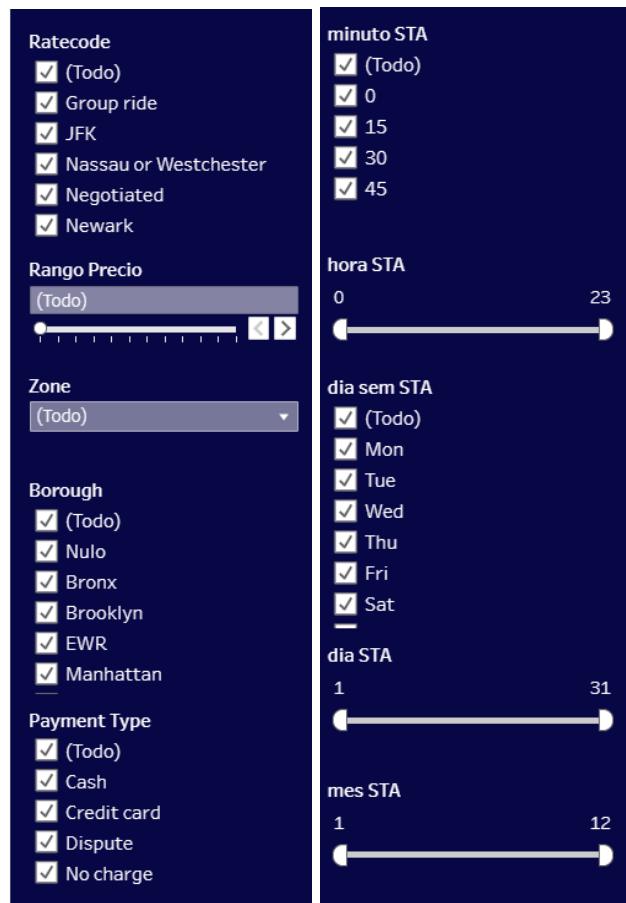
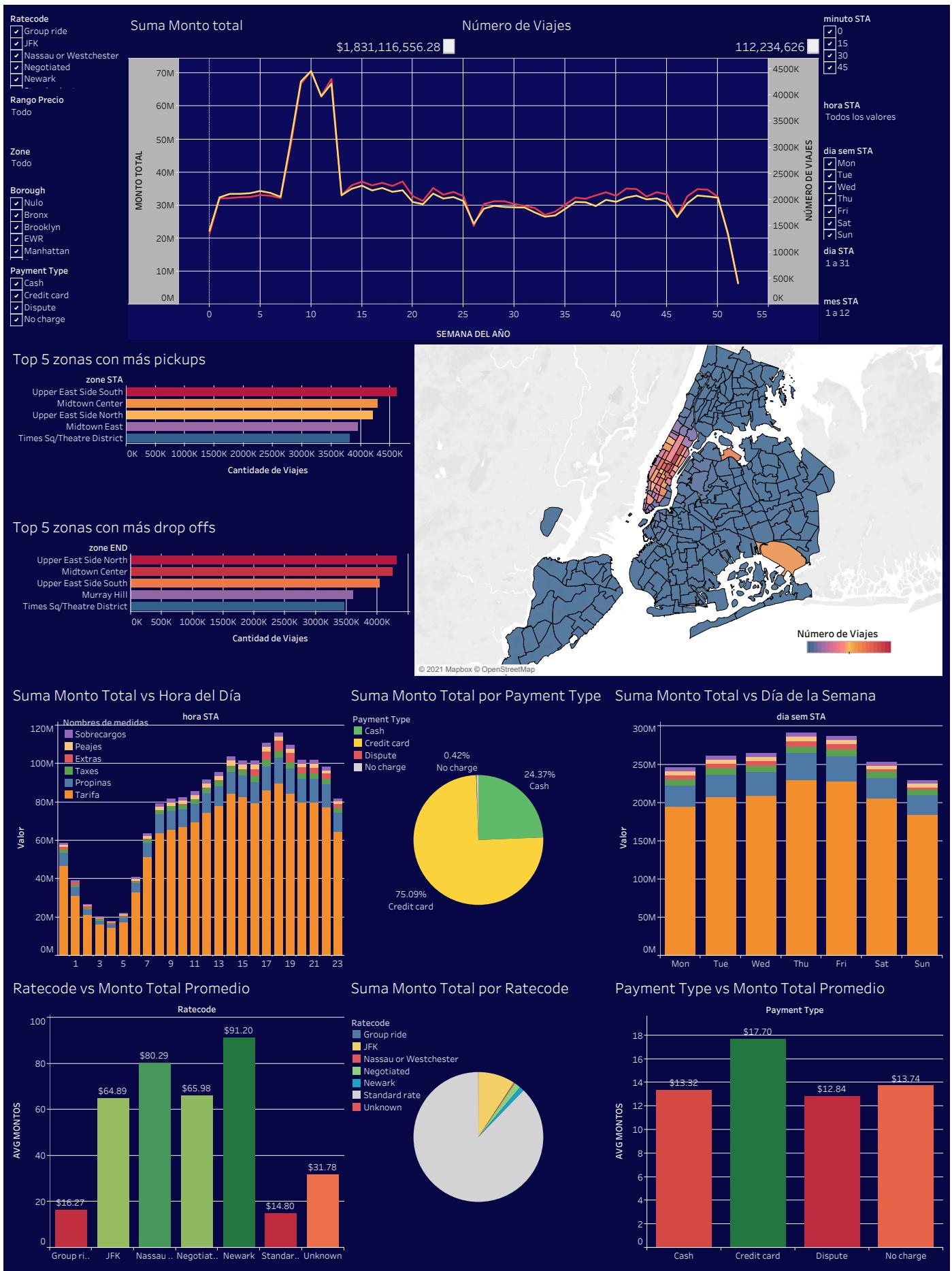
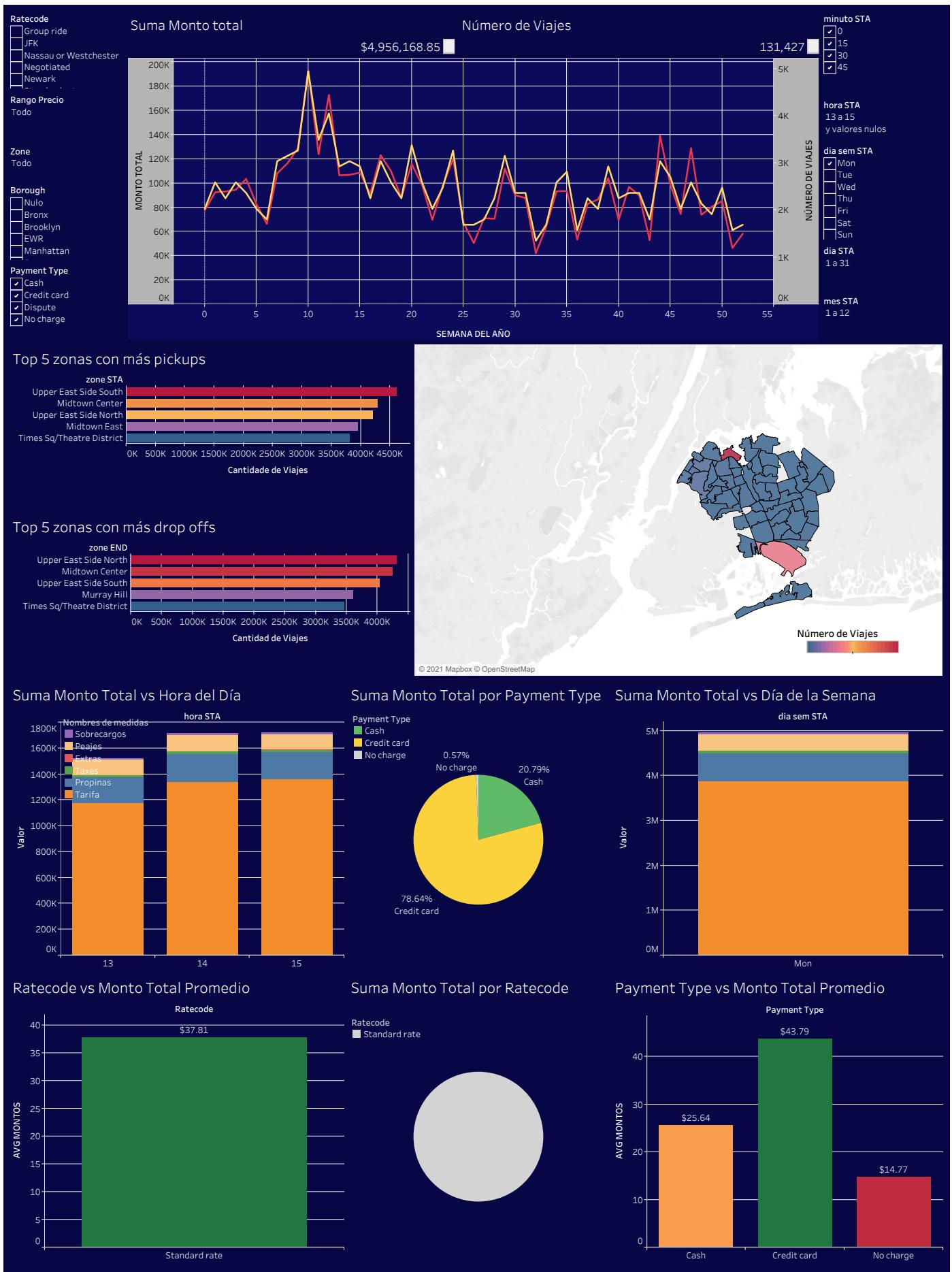


Figura 14: Filtros del tablero de inteligencia de negocios





## 5. Modelo Supervisado

### 5.1. Árbol de decisión

Según *GridSearchCV* el mejor modelo de árbol de decisión viene dado por lo siguiente (*Figura 15*).

```
Mejores parametros: {'criterion': 'mae', 'max_depth': 17, 'min_samples_leaf': 50, 'splitter': 'best'}
Mejor std: 20.48152463657
Mejor score: 1057.6381873030343
```

Figura 15: CV y paraméetros en árbol de decisión

En la validación cruzada si repetimos el mismo modelo varias veces obtendremos en promedio un score (*MAE*) de 1057.6381873030343 que puede considerarse pequeña si tomamos en cuenta que el promedio de viajes por hora es 12812 aproximadamente y como la desviación estandar es "pequeña" cada vez que corramos el modelo obtendremos resultados muy parecidos, en general, obtenemos un "buen modelo" con los parametros anteriores.

- Métricas del modelo.

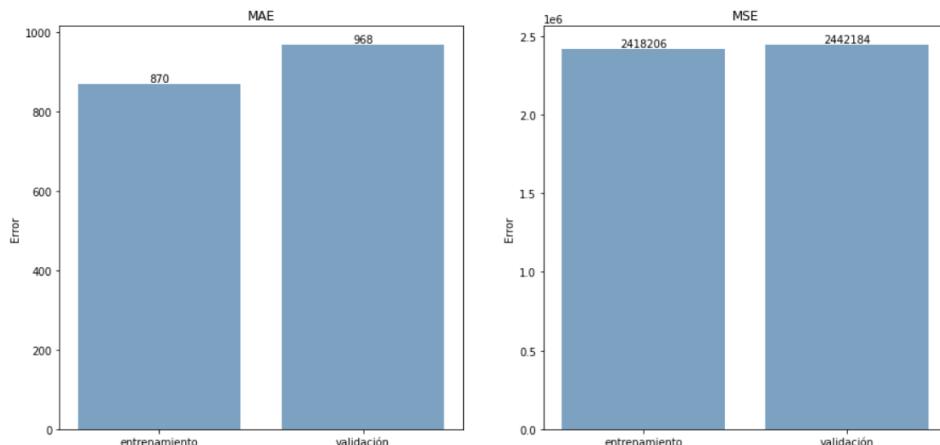


Figura 16: Métricas en árbol de decisión

Observando el *MSE* se notan valores muy grandes a comparación del *MAE*, esto se debe a que la primera métrica penaliza más a valores grandes ya que se eleva al cuadrado el error, aunado al hecho de que el promedio de viajes por hora también es grande (12812), esto explica los valores grandes en el *MSE*.

Según el *MAE* obtendremos un error de estimación de 968 en promedio, que difiere ligeramente del conjunto de entrenamiento (870) y lo cual nos indica que hay muy poco sobreajuste. En general, cuando hagamos una predicción esperaríamos un error promedio de 982, además obtuvimos un error relativo en el conjunto de validación del 10% aproximadamente. Todo esto nos indica que nuestro modelo es "bueno". Se muestran en las *Figuras 17 y 18* las gráficas de las densidades y distribución en el conjunto de validación.

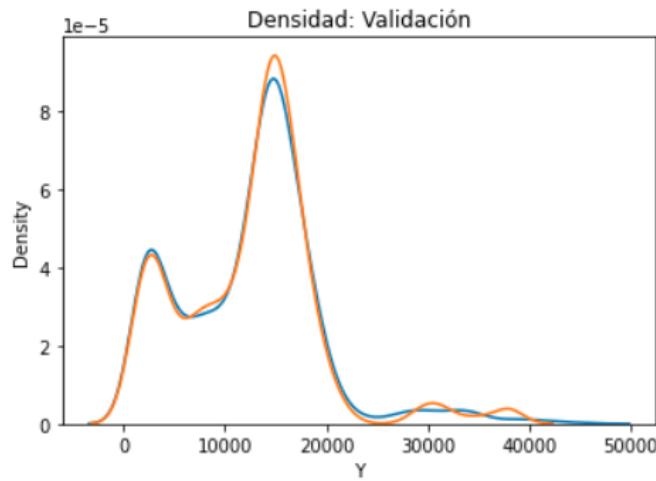


Figura 17: Densidad árbol de decisión en validación

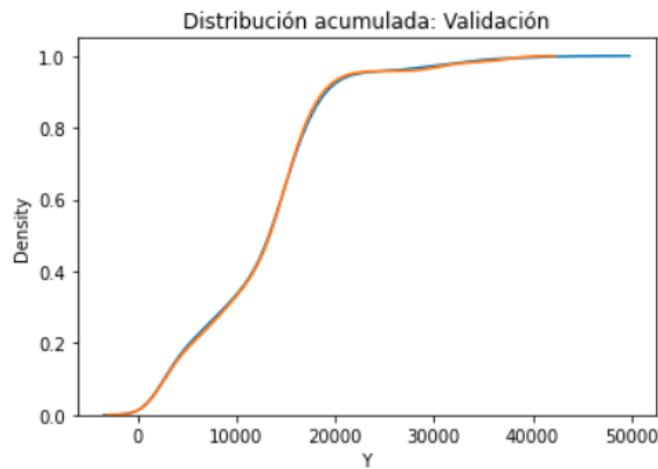


Figura 18: Distribución árbol de decisión en validación

Por último en la *Figura 19* se muestra el ajuste, lo que nos quiere decir es que obtenemos mejores predicciones cuando el volumen de vijes es menor que cuando es mayor.

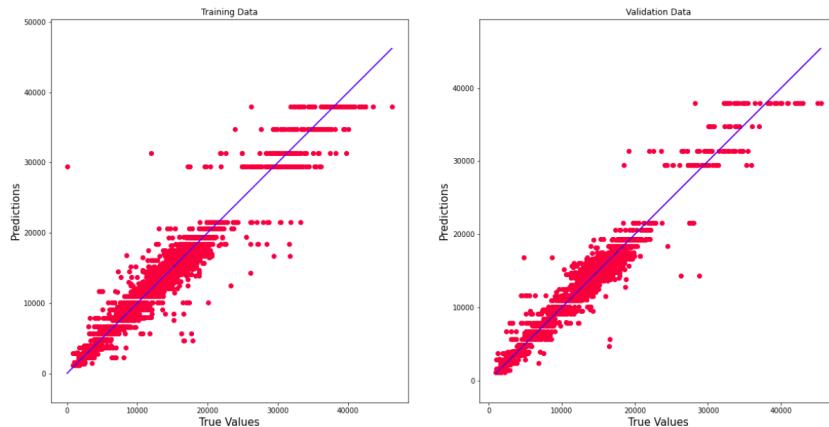


Figura 19: Ajuste en árbol de decisión

## 5.2. Red neuronal

El mejor modelo que obtuvimos esta dado por lo siguiente (*Figura 20*).

```
Mejores parametros: {'learning_rate': 'adaptive', 'hidden_layer_sizes': (23, 38, 22), 'alpha': 0.3809000000000008
9, 'activation': 'relu'}
Desviación estándar: 24.858579602225234
Mejor score: 634.3589660281356
```

Figura 20: CV en red neuronal

En la validación cruzada si repetimos el mismo modelo varias veces obtendremos en promedio un score(*MAE*) de 634.3589660201356 que es un error promedio más pequeño que el árbol de decisión y como la desviación estándar es "muy pequeña" vez que corramos el modelo obtendremos resultados muy parecidos, en general obtenemos un "buen modelo" y mejor que el del árbol de decisión.

- Métricas del modelo.

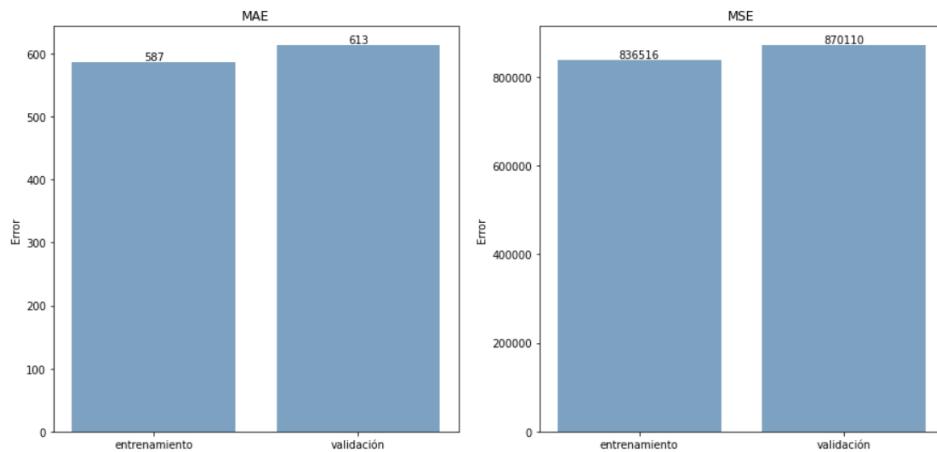


Figura 21: Métricas en red neuronal

De manera análoga que en el árbol de decisión tenemos unos *MSE* muy grandes comparados con el *MAE* esto se debe a lo ya comentado antes. Lo importante a destacar es que en el *MAE* obtenemos 613 que no difiere prácticamente nada al obtenido en los datos de entrenamiento(587) lo cual nos dice que no hay sobre ajuste prácticamente. En general cuando hagamos una predicción esperamos un error promedio de 613. Por consiguiente obtenemos una buena estimación, esto viene confirmado por el error relativo que obtuvimos de aproximadamente 6.5 % en el conjunto de validación.

Se muestran también las gráficas *Figura 22 y 23* de las densidades y distribución en el conjunto de validación en la cual se observa un mejor ajuste de la densidad y distribución acumulada.

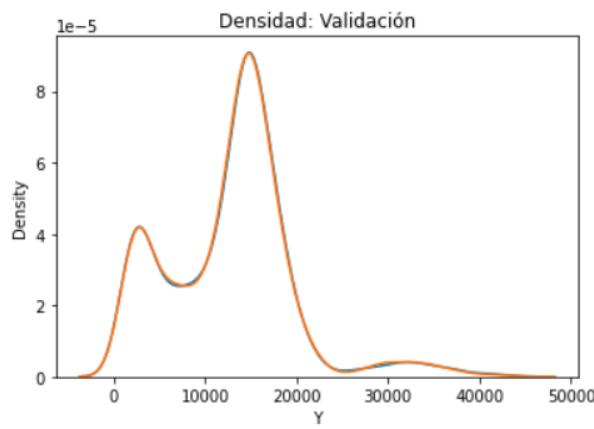


Figura 22: Densidad red neuronal en validación

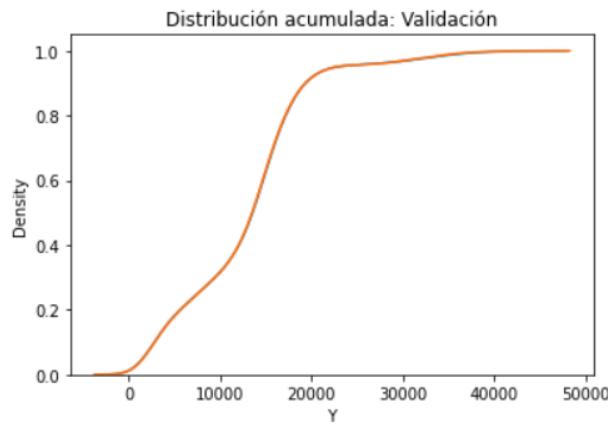


Figura 23: Distribución red neuronal en validación

Por último en la gráfica mostramos el ajuste *Figura 24* y concluimos que la red neuronal nos proporciona un modelo más robusto (con mejor ajuste) y con mejor score.

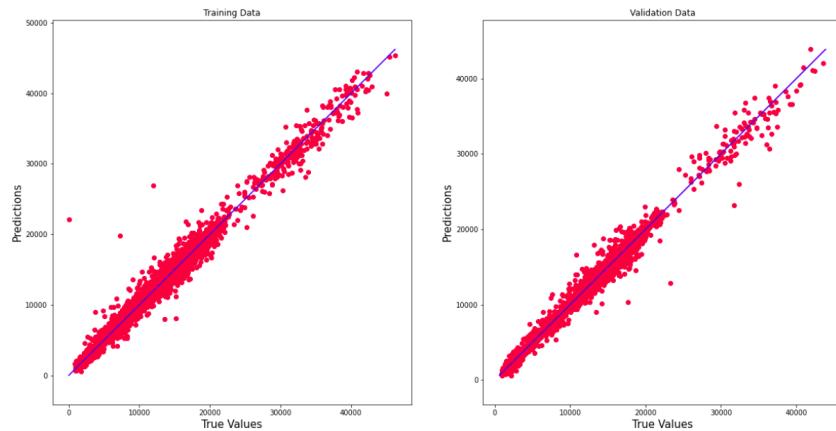


Figura 24: Ajuste en red neuronal

## 6. Estrategia de Aplicacion de los modelos

Actualmente los detalles del precio por un viaje en taxi son los siguientes:

- 2.50 el cargo inicial
- Más 50 *centavos* por 1/5 *milla* cuando se viaja por encima de las 12mph o por 60 segundos cuando el tráfico está parado o el carro está en alto total
- Más 50 *centavos* MTA recargo estatal para todos los viajes que terminan en New York City o Nassau, Suffolk, Westchester, Rockland, Dutchess, Orange o Putnam Counties.
- Más 30 *centavos* de recargo por mejora.

Adicionalmente existen 2 cargos más que serán los que trataremos a fondo que son

- Más 50 *centavos* de recargo de recargo nocturno en horarios de 8 pm a 6 am
- Más 1.00 de recargo en horas pico establecidas de 4 pm a 8 pm en días entre semana.

Como podemos ver se hace un recargo constante dentro de intervalos de horarios ya establecidos, pero gracias a nuestros modelos predictivos sabemos que el volumen de los viajes no será el mismo dentro de todo el intervalo mencionado arriba.

Así que nosotros proponemos una solución que nos permita optimizar nuestra tarifa de recarga para poder tener una tarifa inteligente y que nos pueda crear un escenario del tipo “ganar/ganar” con el cliente. Nosotros tendremos una buena estimación del volumen de los viajes a la siguiente hora con la ayuda de la obtención de los datos de las variables que se utilizaron en la *Matriz Predictiva* y de esta manera podremos saber a tiempo real el volumen de gente que esperamos a la siguiente hora y en lugar de tener un recargo constante que sería de 1.00 en horas pico, podemos tener una tarifa que pueda disminuir o incrementar este precio de acuerdo al volumen estimado.

De esta manera en horarios no tan demandados podremos disminuir el precio de recarga siendo más atractivos para el cliente y que no se sientan obligados a que a partir de cierto horario el precio subirá estrictamente un dólar y por otro lado, si hay mucho un volumen demasiado alto de igual manera no quedarnos en un precio ya establecido sino que se adecúe a nuestra tarifa inteligente, justamente para poder aprovechar esas oportunidades y que en algunas ocasiones que se den esas demandas atípicas podamos sacarle más ganancia de lo que le sacaríamos si tuviéramos esa tarifa ya preestablecida.

Claramente si hay más demanda más incrementará el total de dinero que se va a ganar, entonces, si ajustamos la tarifa de una manera que esté en función de la demanda, si llegamos a tener un exceso estimado de esta podremos subir la tarifa de tal forma de encontrar algún punto óptimo en el cual el precio siga sin subir de una manera exorbitante para el usuario sino por el contrario que suba un precio al cual al usuario no le afecte pero que en nosotros por ese pequeño aumento inteligente que le dimos al precio en función de la alta demanda nos devuelva ganancias espectaculares justamente creando este escenario del tipo “ganar/ganar” en el que el usuario no se siente “estafado” debido a alguna tarifa alta y que a nosotros nos pueda generar grandes utilidades.

## 7. Conclusiones

El análisis exploratorio fue una parte crucial para entender los datos y saber que variables se ocuparían para crear las matrices  $\mathcal{X}$  u  $\vec{y}$  que se ocuparían a su vez para la modelación, nos dimos cuenta que para llegar al modelo tuvimos que comparar dos métodos (árbol de decisión y red neuronal) ver cuál es el óptimo, a pesar de ser modelos buenos nos dimos cuenta que la red neuronal fue más robusta (con mejores métricas) pero al ser una caja negra debe ser evitada en aplicaciones que requiera una explicación de funcionamiento y se obtariamos por el árbol de decisión o algún otro modelo. Obteniendo el modelo desarrollamos una seguridad de uso dónde se puede analizar y sintetizar una estrategia que incrementaría las ganancias a la empresa, con una mejor distribución. Durante el desarrollo del trabajo se aprendieron técnicas para extraer mejores variables y para programar de forma más eficiente.

Para obtener una estrategia con ganancias tuvimos que analizar y estudiar la tarifa que se tiene actualmente y ver como esa misma tarifa se puede mejorar tanto para el cliente como para el proveedor del servicio llegando a un punto óptimo en el cual gracias a lo que es actualmente el activo más importante en el mundo, los datos, se puede llegar a una solución inteligente únicamente haciendo uso de la información que hemos recolectado llegando a un punto donde la explotamos convirtiéndola en conocimiento para poder optimizar nuestros costos de la mejor manera.

## 8. Anexo con el código utilizado

### 8.1. Análisis exploratorio

#### Análisis Exploratorio NYC Yellow Taxi Database 2018

##### Librerías

```
In [1]: import pandas as pd
import numpy as np
import os
import math
import datetime
import seaborn as sns
import matplotlib.pyplot as plt
import shapefile
from funciones import get_boundaries, draw_zone_map, draw_region_map
```

##### Lectura de Datos

Por si se quisiera correr todo con la muestra, NO MOVER

```
df=pd.read_csv('muestra.csv') df.head()
```

```
In [2]: ruta = "/home/osvaldo/Downloads/nyc_taxi/"
archivo = "2018_Yellow_Taxi_Trip_Data.csv"

In [3]: chunk_list = [] #Carga de los datos por chunks
contador = 0
for chunk in pd.read_csv(os.path.join(ruta,archivo),chunksize = 500000, iterator = True):
    contador += 1
    print(contador)
    chunk_list.append(chunk)
```

```
In [4]: df=pd.concat(chunk_list, axis = 0) #Concatenar los chunks
```

```
In [5]: df.head()
```

```
Out[5]: VendorID tpep_pickup_datetime tpep_dropoff_datetime passenger_count trip_distance RatecodeID store_and_fwd_flag PULocationID DOLocation
0 2 09/22/2018 11:46:37 PM 09/23/2018 12:07:10 AM 1 8.00 1 N 230
1 2 09/22/2018 11:00:43 PM 09/22/2018 11:09:36 PM 1 1.70 1 N 141
2 2 09/22/2018 11:10:32 PM 09/22/2018 11:25:50 PM 2 3.84 1 N 163
3 2 09/22/2018 11:27:25 PM 09/22/2018 11:36:29 PM 2 1.80 1 N 166
4 2 09/22/2018 11:50:39 PM 09/23/2018 12:01:52 AM 2 2.43 1 N 229
```

```
In [6]: df.shape
```

```
Out[6]: (112234626, 17)
```

##### Tratamiento de los Datos

```
In [7]: #Cambio de campos de fecha a datetime
campos_fecha=['tpep_pickup_datetime','tpep_dropoff_datetime']
formato_fecha = '%m/%d/%Y %I:%M:%S %p'
for x in campos_fecha:
    df[x]=pd.to_datetime(df[x],
                           format=formato_fecha)
```

```
In [8]: df['tpep_pickup_datetime'].map(lambda x:x.strftime('%Y')).value_counts().to_frame() #Busca años erroneos
```

```
Out[8]: tpep_pickup_datetime
2018 112233121
2009 585
2008 406
2017 227
2019 181
2002 28
2003 18
2084 16
2020 16
2001 14
2026 2
```

tpep_pickup_datetime	
2029	2
2042	2
2031	2
2037	2
2053	1
2021	1
2041	1
2032	1

```
In [10]: for x in campos_fecha: #Reemplazar años erroneos por 2018
    df[x]=df[x].map(lambda z:z.replace(year=2018))
df.head()
```

```
Out[10]:   VendorID tpep_pickup_datetime tpep_dropoff_datetime passenger_count trip_distance RatecodeID store_and_fwd_flag PULocationID DOLocationID
0         2 2018-09-22 23:46:37 2018-09-23 00:07:10           1            8.00             1                 N              230
1         2 2018-09-22 23:00:43 2018-09-22 23:09:36           1            1.70             1                 N              141
2         2 2018-09-22 23:10:32 2018-09-22 23:25:50           2            3.84             1                 N              163
3         2 2018-09-22 23:27:25 2018-09-22 23:36:29           2            1.80             1                 N              166
4         2 2018-09-22 23:50:39 2018-09-23 00:01:52           2            2.43             1                 N              229
```

## Catalogos

```
In [11]: catalogos=[('VendorID',['Creative Mobile Technologies','VeriFone Inc']),
('RatecodeID',['Standard rate','JFK','Newark',
'Nassau or Westchester','Negotiated','Group ride']),
('payment_type',['Credit card','Cash','No charge',
'Dispute','Unknown','Voided trip'])]

l_cat=[('%s %x',pd.DataFrame({'col':y},index=[j for j in range(1,len(y)+1)])\
.reset_index().rename(columns={'index':x,'col':'x_%s%x'})) for x,y in catalogos]
```

```
In [12]: sf=shapefile.Reader('shape/taxi_zones.shp')
fields=[field[0] for field in sf.fields[1:]]
shp_dic=dict(zip(fields,list(range(len(fields)))))

df_loc=pd.DataFrame([dict(zip(fields,i)) for i in sf.records()])
df_loc=df_loc.drop('LocationID',axis=1).rename(columns={'OBJECTID':'LocationID'})
df_loc.to_csv('cat_location.csv',index=False)
df_loc.head()
```

```
Out[12]:   LocationID Shape_Leng Shape_Area          zone      borough
0            1     0.116357  0.000782 Newark Airport       EWR
1            2     0.433470  0.004866  Jamaica Bay      Queens
2            3     0.084341  0.000314 Allerton/Pelham Gardens      Bronx
3            4     0.043567  0.000112   Alphabet City    Manhattan
4            5     0.092146  0.000498    Arden Heights Staten Island
```

```
In [13]: campos_lugar=['PULocationID','DOLocationID']
PU_D0_counts=df[x].value_counts().to_frame().reset_index()\
.rename(columns={'index':'LocationID',
               x:x[:2]+'count'}) for x in campos_lugar]
```

```
In [14]: locations=df_loc[['LocationID','zone']].copy()
for x in campos_lugar:
    aux=( '%s %x',locations.rename(columns={'LocationID':x,'zone':x_%s%x})) 
    l_cat.append(aux)
```

```
In [15]: l_cat[0][1]
```

```
Out[15]:   VendorID      x_VendorID
0         1 Creative Mobile Technologies
1         2                  VeriFone Inc
```

```
In [16]: l_cat[1][1].to_csv('cat_ratecode.csv',index=False)
l_cat[1][1]
```

```
Out[16]:   RatecodeID      x_RatecodeID
0         1 Standard rate
1         2                JFK
2         3                Newark
3         4 Nassau or Westchester
```

RatecodeID	x_RatecodeID
4	5 Negotiated
5	6 Group ride

```
In [17]: l_cat[2][1].to_csv('cat_payment_type.csv',index=False)
l_cat[2][1]
```

```
Out[17]:   payment_type x_payment_type
0           1      Credit card
1           2          Cash
2           3    No charge
3           4     Dispute
4           5     Unknown
5           6  Voided trip
```

## Cruce de catalogos

```
In [18]: for key,cat in l_cat:
    print(key)
    if key[-2:]=='ID':
        df = df.merge(cat,on=key,
                      how='left').drop(key, axis=1).rename(columns={'x_%s'%key:key[:-2]}) 
    else:
        df = df.merge(cat,on=key,
                      how='left').drop(key, axis=1).rename(columns={'x_%s'%key:key})
```

VendorID  
RatecodeID  
payment\_type  
PULocationID  
DOLocationID

```
In [19]: df.tail()
```

```
Out[19]:   tpep_pickup_datetime tpep_dropoff_datetime passenger_count trip_distance store_and_fwd_flag fare_amount extra mta_tax tip_amo
112234621 2018-09-22 23:45:24 2018-09-23 00:09:13 1 3.50 N 17.0 0.5 0.5 1
112234622 2018-09-22 23:05:37 2018-09-22 23:16:16 1 1.98 N 9.5 0.5 0.5 2
112234623 2018-09-22 23:18:34 2018-09-22 23:47:05 1 4.80 N 21.0 0.5 0.5 4
112234624 2018-09-22 23:26:11 2018-09-22 23:48:54 2 2.19 N 15.0 0.5 0.5 3
112234625 2018-09-22 23:52:30 2018-09-23 00:01:50 2 1.24 N 8.0 0.5 0.5 0
```

## Valores atípicos

```
In [15]: df.isna().values.any() #Arreglo booleano, True si existe algún
#valor faltante en alguno de los campos
```

```
Out[15]: True
```

```
In [ ]: df.info()
```

```
In [21]: cols_nas=[x for x in df.columns if df[x].isna().values.any()]
for x in cols_nas:
    print(x)
    print(df[x].isna().value_counts()[1])
    print(round(df[x].isna().value_counts(True)[1]*100,4), '%', '\n')
```

Vendor  
491751  
0.4381 %

Ratecode  
5222  
0.0047 %

PULocation  
1825820  
1.6268 %

DOLocation  
1856259  
1.6539 %

```
In [22]: cols_nas
```

```
Out[22]: ['Vendor', 'Ratecode', 'PULocation', 'DOLocation']
```

```
In [ ]: with_nas = [df[col].isna().value_counts(normalize = True)[1] for col in cols_nas]
without_nas = [df[col].isna().value_counts(normalize = True)[0] for col in cols_nas]

ind=np.arange(len(cols_nas))

plt.subplots(figsize=(10,7))
plt.bar(ind, without_nas, width=0.5,label='Sin nan',color='black')
plt.bar(ind, with_nas, width=0.5,bottom=without_nas,label='Con nan',color='orangered')
plt.ylabel('porcentaje',size='x-large')
plt.title('Datos faltantes',size='xx-large')
plt.xticks(ind,cols_nas,size='x-large')
plt.yticks(size='x-large')
plt.legend(loc = "lower center")
plt.show()
```

## Variables Categóricas y Frecuencia

```
In [24]: variables_categoricas=['Ratecode','store_and_fwd_flag','Vendor', #Cuenta los valores unicos en los campos categóricos
                           'passenger_count','payment_type',
                           'PULocation', 'DOLocation']
cant_val = [len(df[i].unique()) for i in variables_categoricas]
pd.DataFrame( {"variable":variables_categoricas, "# valores que toma":cant_val})
```

```
Out[24]:
```

	variable	# valores que toma
0	Ratecode	7
1	store_and_fwd_flag	2
2	Vendor	3
3	passenger_count	12
4	payment_type	5
5	PULocation	261
6	DOLocation	261

```
In [25]: frecuencias=[]
for i in variables_categoricas:
    aux=df[i].reset_index().groupby(i)\n        .agg('count').rename(columns={'index':'frec'})\n\n    if i not in [variables_categoricas[x] for x in (-1,-2,-4)]: #Nos interesa que algunos campos queden\n        aux=aux.sort_values('frec',ascending=False) #ordenados por orden alfabetico por lo que\n                                                #los excluimos del ordenamiento por frecuencia\n\n    aux['frec %']=aux['frec']/len(df) #frecuencia relativa\n    aux['acum']=aux['frec'].cumsum() #frecuencia acumulada\n    aux['acum %']=aux['acum']/len(df) #frecuencia acumulada relativa\n    frecuencias.append(aux)\nprint(aux, '\n\n')
```

	frec	frec %	acum	acum %
Ratecode				
Standard rate	108895988	0.970253	108895988	0.970253
JFK	2589646	0.023074	111485634	0.993327
Negotiated	443848	0.003955	111929482	0.997281
Newark	228859	0.002039	112158341	0.999320
Nassau or Westchester	70115	0.000625	112228456	0.999945
Group ride	948	0.000008	112229404	0.999953

	frec	frec %	acum	acum %
store_and_fwd_flag				
N	111699074	0.995228	111699074	0.995228
Y	535552	0.004772	112234626	1.000000

	frec	frec %	acum	acum %
Vendor				
VeriFone Inc	64716535	0.576618	64716535	0.576618
Creative Mobile Technologies	47026340	0.419000	111742875	0.995619

	frec	frec %	acum	acum %
passenger_count				
0	1003298	8.939291e-03	1003298	0.008939
1	79786664	7.108917e-01	80789962	0.719831
2	16468127	1.467295e-01	97258089	0.866560
3	4684094	4.173484e-02	101942183	0.908295
4	2209579	1.968714e-02	104151762	0.927982
5	5040905	4.491399e-02	109192667	0.972896
6	3040893	2.709407e-02	112233560	0.999991
7	425	3.786710e-06	112233985	0.999994
8	349	3.109557e-06	112234334	0.999997
9	290	2.583873e-06	112234624	1.000000
96	1	8.909906e-09	112234625	1.000000
192	1	8.909906e-09	112234626	1.000000

	frec	frec %	acum	acum %
payment_type				
Credit card	77928307	6.943339e-01	77928307	0.694334
Cash	33556849	2.989884e-01	111485156	0.993322
No charge	582599	5.190902e-03	112067755	0.998513

```
Dispute      166868  1.486778e-03  112234623  1.000000
Unknown       3     2.672972e-08  112234626  1.000000
```

PULocation	frec	frec %	acum	acum %
Allerton/Pelham Gardens	1323	0.000012	1323	0.000012
Alphabet City	234610	0.002090	235933	0.002102
Arden Heights	135	0.000001	236068	0.002103
Arrochar/Fort Wadsworth	441	0.000004	236509	0.002107
Astoria	181614	0.001618	418123	0.003725
...	...	...	...	...
Woodlawn/Wakefield	2094	0.000019	106302807	0.947148
Woodside	61337	0.000547	106364144	0.947695
World Trade Center	651068	0.005801	107015212	0.953496
Yorkville East	1311101	0.011682	108326313	0.965177
Yorkville West	2082493	0.018555	110408806	0.983732

[260 rows x 4 columns]

DOLocation	frec	frec %	acum	acum %
Allerton/Pelham Gardens	7715	0.000069	7715	0.000069
Alphabet City	528064	0.004705	535779	0.004774
Arden Heights	773	0.000007	536552	0.004781
Arrochar/Fort Wadsworth	2285	0.000020	538837	0.004801
Astoria	482672	0.004301	1021509	0.009102
...	...	...	...	...
Woodlawn/Wakefield	12154	0.000108	106156678	0.945846
Woodside	124199	0.001107	106280877	0.946953
World Trade Center	541182	0.004822	106822059	0.951775
Yorkville East	1424917	0.012696	108246976	0.964470
Yorkville West	2131391	0.018990	110378367	0.983461

[260 rows x 4 columns]

In [56]: `frecuencias[3].round(7)`

Out[56]:

passenger_count	frec	frec %	acum	acum %
0	1003298	0.008939	1003298	0.008939
1	79786664	0.710892	80789962	0.719831
2	16468127	0.146730	97258089	0.866560
3	4684094	0.041735	101942183	0.908295
4	2209579	0.019687	104151762	0.927982
5	5040905	0.044914	109192667	0.972896
6	3040893	0.027094	112233560	0.999991
7	425	0.000004	112233985	0.999994
8	349	0.000003	112234334	0.999997
9	290	0.000003	112234624	1.000000
96	1	0.000000	112234625	1.000000
192	1	0.000000	112234626	1.000000

```
lista_acumulado=list(frecuencias[3]['acum %'])
l,c,t,o,b='lightsalmon','coral','tomato','orangered','black' #Colores
plt.subplots(figsize=(10,7))
plt.bar(np.arange(0,len(lista_acumulado)),list(lista_acumulado),
color=[l,l,c,t,o,o,b])
plt.title('passenger count frecuencia acumulada',size='xx-large')
plt.xlabel('número de pasajeros',size='x-large')
plt.ylabel('porcentaje',size='x-large')
plt.xticks(np.arange(0,len(lista_acumulado)),size='x-large')
plt.yticks(np.arange(0,1.1,0.1),size='x-large')
plt.show()
```

```
campos=variables_categoricas[:-2]
lista=[df[i].copy().value_counts(True).head(10).to_frame().reset_index().rename(columns={i:'total'}) for i in campos]
colores=plt.get_cmap('cool')
plt.subplots(figsize=(15,15))
for i,j in enumerate(lista):
    plt.subplot(3,2,i+1)
    plt.barh(j['index'],j.total,
            color=[colores(k) for k in np.arange(0,169,25)[::-1]],
            edgecolor='black')
    if campos[i]=='passenger_count':
        plt.yticks(list(range(lista[i]['index'].max()+1)),size='large')
    else:
        plt.yticks(size='large')
    plt.xticks(size='large')
    maximo=math.ceil(j.total.max()*10)/10
    plt.xticks(np.arange(0,maximo+0.1,0.1),size='large')
    plt.title(campos[i],size='x-large')
    #for k in range(len(lista[i])):
        #plt.text(maximo+maximo*0.05,k-0.1,
```

```

        #'{:,.2f}'.format(round(j.total[k]*100,4)),size='x-large',color='navy')
plt.show()

In [28]: aux=pd.concat(PU_D0_counts).merge(df_loc[['LocationID']],on='LocationID',how = 'outer').fillna(0) \
           .groupby(['LocationID'], as_index=False)\ \
           .agg({'PUcount': 'sum', 'D0count': 'sum'})\ \
           .sort_values('LocationID')
aux['TOTALcount'] = aux['PUcount'] + aux['D0count']
aux = aux.merge(df_loc[['LocationID','zone','borough']], left_on='LocationID', right_on='LocationID')
for i in ['PUcount','D0count','TOTALcount']:
    aux[i]=aux[i].astype(int)

In [29]: datos_grafica=[(dict(zip(aux['LocationID'].tolist(),
                               (aux[x[2:]+ 'count']/len(df)).tolist()))),aux.sort_values(x[2:]+ 'count',
                               ascending=False).set_index('LocationID')) for x in campos_lugar]

In [ ]: datos_grafica[0][1].head() #Top 5 zonas con más inicios de viajes

In [ ]: datos_grafica[1][1].head() #Top 5 zonas con más terminos de viajes

In [ ]: fig, ax = plt.subplots(nrows=1,ncols=2,figsize=(18,8))
for x,y in enumerate(campos_lugar):
    ax = plt.subplot(1, 2, x+1)
    ax.set_title(y[2:]+ 'LocationID')
    plt.title(y,size='x-large')
    draw_zone_map(ax,sf,shp_dic,heat=datos_grafica[x][0],text=datos_grafica[x][1].head(3).index.tolist())
plt.show()

```

## Variables Continuas y Distribución

```

In [3]: variables_continuas=['tpep_pickup_datetime','tpep_dropoff_datetime',
                           'trip_distance','fare_amount', 'extra','mta_tax','tip_amount',
                           'tolls_amount','improvement_surcharge','total_amount']

var_descr=variables_continuas #Quitamos los campos de fechas porque
for i in range(2):           #los trataremos de forma distinta
    var_descr.pop(0)

In [ ]: dist_campus_fecha=df[x].map(lambda z:z.strftime('%j')).astype(int) \
         .value_counts(True).to_frame().reset_index() \
         .sort_values('index').reset_index().drop('level_0',axis=1) \
         .rename(columns={'index':'día_del_anio',x:'total_viajes'}) for x in campos_fecha

In [32]: for x in dist_campus_fecha:
    x['x_total_viajes']=x.total_viajes.cumsum()

In [ ]: plt.subplots(figsize=(15,12))
titulo=('pickup','dropoff')
colores=('royalblue','deeppink')
for x,y in enumerate(dist_campus_fecha):
    plt.subplot(2,1,x+1)
    plt.plot(y.día_del_anio,y.total_viajes,color=colores[x])
    plt.title('distribución %s_datetime' %(titulo[x]),size='xx-large')
    plt.xticks(np.arange(0,365,25),size='large')
    plt.yticks(size='large')
    plt.xlabel('día del año',size='x-large')
    plt.ylabel('porcentaje',size='x-large')

In [ ]: plt.subplots(figsize=(15,12))
titulo=('pickup','dropoff')
colores=('royalblue','deeppink')
for x,y in enumerate(dist_campus_fecha):
    plt.subplot(2,1,x+1)
    plt.plot(y.día_del_anio,y.x_total_viajes,color=colores[x])
    plt.title('distribución acumulativa %s_datetime' %(titulo[x]),size='xx-large')
    plt.xticks(np.arange(0,365,25),size='large')
    plt.yticks(size='large')
    plt.xlabel('día del año',size='x-large')
    plt.ylabel('porcentaje acumulado',size='x-large')

In [ ]: plt.subplots(figsize=(15,15))

plt.subplot(2,1,1)
plt.title('Distribución tip_amount',size='x-large')
ax=sns.distplot(df.tip_amount, hist = True,
kde = True, bimns=int(abs(max(df.tip_amount.tolist())-min(df.tip_amount.tolist()))),color ='lime',
hist_kws={'edgecolor':'black'},
kde_kws = {'linewidth': 3,'shade':False,'color':'black'})
ax.set_xlim(-2,20)
ax.set_xticks(list(np.arange(-2,21,1)))
ax.set_xlabel('tip_amount',size='x-large')
ax.set_ylabel('Density',size='x-large')

plt.subplot(2,1,2)
ax1=sns.kdeplot(df.tip_amount,cumulative=True,color='lime',linewidth=3)
ax=sns.ecdfplot(df.tip_amount,color='black',linewidth=1)
ax.set_xlim(-2,20)
ax.set_xticks(list(np.arange(-2,21,1)))
ax.set_xlabel('tip_amount',size='x-large')

```

```
    ax.set_ylabel('Cumulative Density')
    plt.show()

/home/osvaldo/entornos/tsc/lib/python3.8/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

In [ ]:

```
plt.subplots(figsize=(15,15))

plt.subplot(2,1,1)
plt.title('Distribución tip_amount',size='x-large')
ax=sns.distplot(df.tip_amount, hist = True,
kde = True, bins=int(abs(max(df.tip_amount.tolist())-min(df.tip_amount.tolist())))),color ='lime',
hist_kws={'edgecolor':'black'},
kde_kws = {'linewidth': 3,'shade':False,'color':'black'})
ax.set_xlim(-2,20)
ax.set_xticks(list(np.arange(-2,21,1)))
ax.set_xlabel('tip_amount',size='x-large')
ax.set_ylabel('Density',size='x-large')

plt.subplot(2,1,2)
ax1=sns.kdeplot(df.tip_amount,cumulative=True,color='lime',linewidth=3)
ax=sns.ecdfplot(df.tip_amount,color='black',linewidth=1)
ax.set_xlim(-2,20)
ax.set_xticks(list(np.arange(-2,21,1)))
ax.set_xlabel('tip_amount',size='x-large')
ax.set_ylabel('Cumulative Density')
plt.show()
```

In [ ]:

```
plt.subplots(figsize=(15,15))

plt.subplot(2,1,1)
ax=sns.distplot(df.fare_amount, hist = True,
kde = True, bins=int(abs(max(df.fare_amount.tolist())-min(df.fare_amount.tolist()))/5), color = 'red',
hist_kws={'edgecolor':'black'},
kde_kws = {'linewidth': 3,'shade':False,'color':'black'})
ax.set_xlim(-10,100)
ax.set_xticks(list(np.arange(-10,105,5)))
ax.set_xlabel('fare_amount',size='x-large')
ax.set_ylabel('Density',size='x-large')

plt.subplot(2,1,2)
ax1=sns.kdeplot(df.fare_amount,cumulative=True,color='red',linewidth=3)
ax=sns.ecdfplot(df.fare_amount,color='black',linewidth=1)
ax.set_xlim(-10,100)
ax.set_xticks(list(np.arange(-10,105,5)))
ax.set_xlabel('fare_amount',size='x-large')
ax.set_ylabel('Cumulative Density')
plt.show()
```

In [ ]:

```
plt.subplots(2,1,figsize=(15,15))
#sns.set(rc={'figure.figsize':(15,6)})
plt.subplot(2,1,1)
ax=sns.distplot(df.total_amount, hist = True,
kde = True, bins=int(abs(max(df.total_amount.tolist())-min(df.total_amount.tolist()))/5), color = 'cyan',
hist_kws={'edgecolor':'black'},
kde_kws = {'linewidth': 3,'shade':False,'color':'black'})
ax.set_xlim(-10,100)
ax.set_xticks(list(np.arange(-10,105,5)))
ax.set_xlabel('total_amount',size='x-large')
ax.set_ylabel('Density',size='x-large')

plt.subplot(2,1,2)
ax1=sns.kdeplot(df.total_amount,cumulative=True,color='cyan',linewidth=3)
ax=sns.ecdfplot(df.total_amount,color='black',linewidth=1)
ax.set_xlim(-10,100)
ax.set_xticks(list(np.arange(-10,105,5)))
ax.set_xlabel('total_amount',size='x-large')
ax.set_ylabel('Cumulative Density',size='x-large')
plt.show()
```

In [ ]:

```
plt.subplots(2,1,figsize=(15,15))

plt.subplot(2,1,1)
ax=sns.distplot(df.trip_distance, hist = True,
kde = True, bins=int(abs(max(df.trip_distance.tolist())-min(df.trip_distance.tolist()))), color = 'magenta',
hist_kws={'edgecolor':'black'},
kde_kws = {'linewidth': 3,'shade':False,'color':'black'})
ax.set_xlim(0,20)
ax.set_xticks(list(np.arange(0,25,1)))
ax.set_xlabel('trip_distance',size='x-large')
ax.set_ylabel('Density',size='x-large')

plt.subplot(2,1,2)
ax1=sns.kdeplot(df.trip_distance,cumulative=True,color='magenta',linewidth=3)
ax=sns.ecdfplot(df.trip_distance,color='black',linewidth=1)
ax.set_xlim(0,20)
ax.set_xticks(list(np.arange(0,25,1)))
ax.set_xlabel('trip_distance',size='x-large')
ax.set_ylabel('Cumulative Density',size='x-large')
plt.show()
```

## Descripción General

```
In [ ]: df[var_descr].describe().round(3)
```

## Busqueda de Valores Atípicos

```
In [ ]: plt.subplots(figsize=(25,15))
r,a,n='red','royalblue','black'
colores=[a,n,r,a,n,r,a,n]
for x,y in enumerate(var_descr):
    plt.subplot(3,3,x+1)
    plt.tight_layout(pad = 2)
    plt.axhline(0,color='black',linestyle='--')
    plt.scatter(df[y],np.zeros(len(df[y])),color=colores[x])
    plt.title(y,size='xx-large')
    plt.ylim(-1,1)
    plt.yticks([-1,1],[])
    plt.xticks(size='x-large')
plt.show()
```

### Porcentaje de valores atípicos variables continuas

trip\_distance

```
In [3]: x=df.loc[df.trip_distance>=21.5].shape[0]
print(round(x/len(df),5))
```

0.00341

fare\_amount

```
In [9]: x=df.loc[(df.fare_amount>80) | (df.fare_amount<2.5)].shape[0]
print(round(x/len(df),5))
```

0.00244

tip\_amount

```
In [8]: x=df.loc[(df.tip_amount>13) | (df.tip_amount<0)].shape[0]
print(round(x/len(df),5))
```

0.00631

tolls\_amount

```
In [7]: x=df.loc[(df.tolls_amount>5.77) | (df.tolls_amount<0)].shape[0]
print(round(x/len(df),5))
```

0.0033

total\_amount

```
In [11]: x=df.loc[(df.total_amount>84) | (df.total_amount<3.3)].shape[0]
print(round(x/len(df),5))
```

0.00432

## Variables Falsas Continuas

```
In [47]: discretizar=['mta_tax','extra','improvement_surcharge'] #Variables que deberian ser discretas
          #porque solo pueden tomar valores
          #en un conjunto finito
for i in discretizar:
    print(df[i].value_counts(True).head().round(6),'\n')
0.50    0.994897
0.00    0.004467
-0.50   0.000633
3.00    0.000001
0.87    0.000001
Name: mta_tax, dtype: float64

0.0    0.533283
0.5    0.300174
1.0    0.161928
4.5    0.004233
-0.5   0.000221
Name: extra, dtype: float64

0.3    0.999058
-0.3   0.000650
0.0    0.000289
1.0    0.000002
0.6    0.000000
Name: improvement_surcharge, dtype: float64
```

## Correlación entre variables continuas

```
In [4]: correlacion=df[var_descr].corr()
correlacion
```

```
Out[4]:      trip_distance  fare_amount    extra    mta_tax  tip_amount  tolls_amount  improvement_surcharge  total_amount
```

	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount
trip_distance	1.000000	0.872479	0.097077	-0.097762	0.547560	0.532566	0.018196	0.879143
fare_amount	0.872479	1.000000	0.073875	-0.177097	0.556283	0.500363	0.042555	0.979854
extra	0.097077	0.073875	1.000000	0.041439	0.076176	0.046233	0.035499	0.111528
mta_tax	-0.097762	-0.177097	0.041439	1.000000	-0.114313	-0.201804	0.564336	-0.185106
tip_amount	0.547560	0.556283	0.076176	-0.114313	1.000000	0.384411	0.016429	0.683281
tolls_amount	0.532566	0.500363	0.046233	-0.201804	0.384411	1.000000	0.004980	0.602006
improvement_surcharge	0.018196	0.042555	0.035499	0.564336	0.016429	0.004980	1.000000	0.042224
total_amount	0.879143	0.979854	0.111528	-0.185106	0.683281	0.602006	0.042224	1.000000

```
In [ ]: plt.subplots(figsize=(10,10))
ax=sns.heatmap(correlacion,square=True,cmap='winter',annot=True)
plt.xticks(size='large')
plt.yticks(size='large')
plt.show()
```

## Relación entre variables Categóricas con variables Continuas

### Impacto de variables categoricas en fare\_amount

```
In [ ]: plt.subplots(figsize=(25,25))
colores=[b,o,o,b,b]
colores1=[o,b,b,o,o]
for x,y in enumerate(campos):
    plt.subplot(3,2,x+1)
    plt.tight_layout(pad = 2)
    z=df[[y,'fare_amount']].groupby(y).mean().head(10).reset_index()
    plt.plot(z[y],z.fare_amount,color=colores1[x],linewidth=5)
    plt.bar(z[y],z.fare_amount,linewidth=2,color=colores[x])

    if y=='passenger_count':
        plt.xticks(list(range(z[y].max()+1)),size='xx-large',rotation=90)
    else:
        plt.xticks(size='xx-large',rotation=90)

    plt.yticks(size='xx-large')
    plt.ylabel('fare amount promedio',size='xx-large')
plt.show()
```

### Impacto de variables categoricas en tip\_amount

```
In [ ]: plt.subplots(figsize=(25,25))
colores=[b,o,o,b,b]
colores1=[o,b,b,o,o]
for x,y in enumerate(campos):
    plt.subplot(3,2,x+1)
    plt.tight_layout(pad = 2)
    z=df[[y,'tip_amount']].groupby(y).mean().head(10).reset_index()
    plt.plot(z[y],z.tip_amount,color=colores1[x],linewidth=5)
    plt.bar(z[y],z.tip_amount,linewidth=2,color=colores[x])

    if y=='passenger_count':
        plt.xticks(list(range(z[y].max()+1)),size='xx-large',rotation=90)
    else:
        plt.xticks(size='xx-large',rotation=90)

    plt.xticks(size='xx-large',rotation=90)
    plt.yticks(size='xx-large')
    plt.ylabel('tip amount promedio',size='xx-large')
plt.show()
```

### Impacto de variables categoricas en trip\_distance

```
In [ ]: plt.subplots(figsize=(25,25))
colores=[b,o,o,b,b]
colores1=[o,b,b,o,o]
for x,y in enumerate(campos):
    plt.subplot(3,2,x+1)
    plt.tight_layout(pad = 2)
    z=df[[y,'trip_distance']].groupby(y).mean().head(10).reset_index()
    plt.plot(z[y],z.trip_distance,color=colores1[x],linewidth=5)
    plt.bar(z[y],z.trip_distance,linewidth=2,color=colores[x])

    if y=='passenger_count':
        plt.xticks(list(range(z[y].max()+1)),size='xx-large',rotation=90)
    else:
        plt.xticks(size='xx-large',rotation=90)

    plt.xticks(size='xx-large',rotation=90)
    plt.yticks(size='xx-large')
    plt.ylabel('trip_distance',size='xx-large')
plt.show()
```

Aquí se detecta que valores en otros campos dan origen a valores atípicos

```
In [ ]: sns.set_style("whitegrid")
ax=sns.catplot('payment_type','fare_amount',data=df)
plt.xticks(rotation=90)
plt.ylim(-50,50)
plt.show()
```

```
In [ ]: ax=sns.catplot('Vendor','tolls_amount',data=df)
plt.show()
```

```
In [ ]: ax=sns.catplot('Vendor','tip_amount',data=df)
plt.ylim(-5,5)
plt.show()
```

```
In [ ]: ax=sns.catplot('Vendor','fare_amount',data=df)
plt.xticks(rotation=90)
# plt.ylim(-50,50)
plt.show()
```

```
In [ ]: ax=sns.catplot('Ratecode','fare_amount',data=df)
plt.xticks(rotation=90)
plt.ylim(-50,50)
plt.show()
```

```
In [ ]: ax=sns.catplot('Ratecode','tip_amount',data=df)
plt.xticks(rotation=90)
plt.ylim(-5,5)
plt.show()
```

## 8.2. Cubo OLAP

### Librerías

```
In [1]: import pandas as pd
import numpy as np
import json as js
import matplotlib.pyplot as plt
import shapefile
from funciones import get_boundaries, draw_zone_map, draw_region_map
from sqlalchemy import create_engine
```

### Conexión a mysql

```
In [2]: credenciales = js.load(open('cred.json','rb'))

usuario = credenciales['user']
pwd = credenciales['pwd']
host = credenciales['host']
port = credenciales['port']
db = credenciales['db']

engine = create_engine(f'mysql+pymysql://{usuario}:{pwd}@{host}:{port}/{db}').connect()
```

### Lectura de viajes

```
In [3]: df=pd.read_csv('muestra.csv')
df.head()

Out[3]:   VendorID tpep_pickup_datetime tpep_dropoff_datetime passenger_count trip_distance RatecodeID store_and_fwd_flag PULocationID DOLocationID
0         2 02/03/2018 12:12:10 PM 02/03/2018 12:20:01 PM           2       1.70          1             N          141          137
1         2 08/02/2018 01:42:31 PM 08/02/2018 02:23:00 PM           2       4.93          1             N          231          170
2         1 03/26/2018 06:39:06 PM 03/26/2018 06:50:35 PM           4       1.60          1             N          163            68
3         2 04/29/2018 03:57:20 AM 04/29/2018 04:09:51 AM           5       3.43          1             N          163          146
4         2 10/11/2018 01:26:11 PM 10/11/2018 01:39:04 PM           2       1.10          1             N          161          164
```

### Tratamiento de los datos

```
In [4]: campos_fecha=['tpep_pickup_datetime','tpep_dropoff_datetime']

formato_fecha = '%m/%d/%Y %I:%M:%S %p'
for x in campos_fecha:
    df[x]=pd.to_datetime(df[x],
                           format=formato_fecha)

In [5]: values=df.extra.value_counts(True)
new_values_extra=values.to_frame().reset_index()['index'][3:].tolist()
x=list(values)[3:]
new_proba=[x[y]/sum(x) for y in range(3)]

def new_value_extra(x): #Funcion que cambia los valores
    if x not in new_values_extra:
        x=np.random.choice(new_values_extra,p=new_proba)
    return x

for x in campos_fecha: df[x]=df[x].map(lambda z:z.replace(year=2018, minute=0, second=0))

df.RatecodeID.replace(99,7,inplace=True) df.mta_tax=np.full(len(df),0.5) df.improvement_surcharge=np.full(len(df),0.3)
df.extra=df.extra.map(new_value_extra)
```

```
In [6]: for x in campos_fecha:
    df[x]=df[x].map(lambda z:z.replace(year=2018, minute=(z.minute//15)*15, second=0))

df.RatecodeID.replace(99,7,inplace=True)
df.mta_tax=np.full(len(df),0.5)
df.improvement_surcharge=np.full(len(df),0.3)
df.extra=df.extra.map(new_value_extra)
```

```
In [7]: #Rectificamos el monto total
df.total_amount=df.fare_amount+df.extra+df.mta_tax+df.tip_amount+df.tolls_amount+df.improvement_surcharge
```

```
In [8]: df.tail()
```

```
Out[8]:   VendorID tpep_pickup_datetime tpep_dropoff_datetime passenger_count trip_distance RatecodeID store_and_fwd_flag PULocationID DOLocationID
999995         2 2018-09-10 01:00:00 2018-09-10 01:15:00           1       1.05          1             N            79
999996         2 2018-04-28 01:00:00 2018-04-28 01:00:00           2       1.05          1             N            264
```

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID
999997	1 2018-03-10 21:15:00	2018-03-10 21:30:00	1	2.70	1	N	163	
999998	2 2018-05-12 15:30:00	2018-05-12 15:30:00	1	0.42	1	N	151	
999999	1 2018-01-01 16:45:00	2018-01-01 16:45:00	1	0.70	1	N	237	

In [9]: df.dtypes

```
Out[9]: VendorID           int64
tpep_pickup_datetime    datetime64[ns]
tpep_dropoff_datetime   datetime64[ns]
passenger_count          int64
trip_distance            float64
RatecodeID              int64
store_and_fwd_flag       object
PULocationID            int64
DOLocationID            int64
payment_type             int64
fare_amount              float64
extra                   float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge   float64
total_amount              float64
dtype: object
```

## Dimensiones

### Dimensión tiempo

In [26]: aux = df[campos\_fecha].copy()
l=[aux[[x]].rename(columns={x:'tiempo'}) for x in campos\_fecha]

In [27]: td\_tiempo = pd.concat(l,ignore\_index=True)
td\_tiempo = td\_tiempo.drop\_duplicates().dropna()

```
td_tiempo = pd.concat(l,ignore_index=True) td_tiempo = td_tiempo.drop_duplicates().dropna() td_tiempo['tiempo'] = pd.to_datetime(td_tiempo['tiempo'])
td_tiempo['hora'] = td_tiempo['tiempo'].map(lambda x:x.strftime('%H')).astype(int) td_tiempo['anio'] = td_tiempo['tiempo'].map(lambda x:x.strftime('%Y')).astype(int) td_tiempo['mes'] = td_tiempo['tiempo'].map(lambda x:x.strftime('%m')).astype(int) td_tiempo['dia'] = td_tiempo['tiempo'].map(lambda x:x.strftime('%d')).astype(int) td(tiempo['dia_sem'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%U')).astype(int)) td(tiempo['sem_anio'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%U')).astype(int)) td(tiempo['id'] = td(tiempo.sort_values(['mes','dia','hora']).reset_index(drop=True) td(tiempo['id'] = td(tiempo.index+1 td(tiempo['id'] = td(tiempo['id'].astype(int) #Especificar tipo de dato par que entre así a mysql.
```

In [28]: td\_tiempo = pd.concat(l,ignore\_index=True)
td\_tiempo = td\_tiempo.drop\_duplicates().dropna()
td(tiempo['tiempo'] = pd.to\_datetime(td(tiempo['tiempo']))
td(tiempo['minuto'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%M')).astype(int))
td(tiempo['hora'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%H')).astype(int))
td(tiempo['anio'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%Y')).astype(int))
td(tiempo['mes'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%m')).astype(int))
td(tiempo['dia'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%d')).astype(int))
td(tiempo['dia\_sem'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%a')))
td(tiempo['sem\_anio'] = td(tiempo['tiempo'].map(lambda x:x.strftime('%U')).astype(int))
td(tiempo['id'] = td(tiempo.sort\_values(['mes','dia','hora','minuto']).reset\_index(drop=True)
td(tiempo['id'] = td(tiempo.index+1
td(tiempo['id'] = td(tiempo['id'].astype(int) #Especificar tipo de dato par que entre así a mysql.

In [29]: td\_tiempo

	tiempo	minuto	hora	anio	mes	dia	dia_sem	sem_anio	id
0	2018-01-01 00:00:00	0	0	2018	1	1	Mon	0	1
1	2018-01-01 00:15:00	15	0	2018	1	1	Mon	0	2
2	2018-01-01 00:30:00	30	0	2018	1	1	Mon	0	3
3	2018-01-01 00:45:00	45	0	2018	1	1	Mon	0	4
4	2018-01-01 01:00:00	0	1	2018	1	1	Mon	0	5
...	...	...	...	...	...	...	...	...	...
35011	2018-12-31 22:45:00	45	22	2018	12	31	Mon	52	35012
35012	2018-12-31 23:00:00	0	23	2018	12	31	Mon	52	35013
35013	2018-12-31 23:15:00	15	23	2018	12	31	Mon	52	35014
35014	2018-12-31 23:30:00	30	23	2018	12	31	Mon	52	35015
35015	2018-12-31 23:45:00	45	23	2018	12	31	Mon	52	35016

35016 rows × 9 columns

In [30]: td\_tiempo.shape

Out[30]: (35016, 9)

### Dimensión Lugar

```
In [31]: sf=shapefile.Reader('shape/taxi_zones.shp')
fields=[field[0] for field in sf.fields[1:]]
shp_dic=dict(zip(fields,list(range(len(fields)))))

df_loc=pd.DataFrame([dict(zip(fields,i)) for i in sf.records()])
df_loc=df_loc.drop(['LocationID'],axis=1).rename(columns={'OBJECTID':'LocationID'})
df_loc.to_csv('cat_location.csv',index=False)
```

```
In [32]: td_lugar=df_loc[['zone','borough','LocationID']].rename(columns={'LocationID':'id'})
td_lugar.loc[263]=['Unknown','Unknown',264]
td_lugar.loc[264]=['Unknown','Unknown',265]
td_lugar
```

Out[32]:

	zone	borough	id
0	Newark Airport	EWR	1
1	Jamaica Bay	Queens	2
2	Allerton/Pelham Gardens	Bronx	3
3	Alphabet City	Manhattan	4
4	Arden Heights	Staten Island	5
...	...	...	...
260	World Trade Center	Manhattan	261
261	Yorkville East	Manhattan	262
262	Yorkville West	Manhattan	263
263	Unknown	Unknown	264
264	Unknown	Unknown	265

265 rows × 3 columns

```
In [ ]: fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15,8))
ax = plt.subplot(1, 2, 1)
ax.set_title("Boroughs in NYC")
draw_region_map(ax, sf, shp_dic)
ax = plt.subplot(1, 2, 2)
ax.set_title("Zones in NYC")
draw_zone_map(ax, sf, shp_dic)
```

## Dimensión medio de pago

```
In [34]: td_medio_pago=pd.read_csv('cat_payment_type.csv')[['x_payment_type','payment_type']]
td_medio_pago=td_medio_pago.rename(columns={'payment_type':'id','x_payment_type':'payment_type'})
td_medio_pago
```

Out[34]:

	payment_type	id
0	Credit card	1
1	Cash	2
2	No charge	3
3	Dispute	4
4	Unknown	5
5	Voided trip	6

## Dimensión Ratecode

```
In [35]: td_ratecode=pd.read_csv('cat_ratecode.csv')[['x_RatecodeID','RatecodeID']]
td_ratecode=td_ratecode.rename(columns={'RatecodeID':'id','x_RatecodeID':'Ratecode'})
td_ratecode.loc[6]=['Unknown',7]
td_ratecode
```

Out[35]:

	Ratecode	id
0	Standard rate	1
1	JFK	2
2	Newark	3
3	Nassau or Westchester	4
4	Negotiated	5
5	Group ride	6
6	Unknown	7

## Dimensión rango de precios

```
In [36]: corte=[2.5,7.5,10,12.5,15,17.5,20,25,50,75,85]
id_rango=list(np.arange(1,11,1))
etiquetas=['2.5 a 7.5','7.5 a 10','10 a 12.5','12.5 a 15','15 a 17.5',
          '17.5 a 20','20 a 25','25 a 50','50 a 75','75 a 85','atípico']
id_rango.append(11)
td_rango_precio=pd.DataFrame(list(zip(etiquetas,id_rango)),columns=['rango_precio','id'])
```

```
In [37]: td_rango_precio
```

	rango_precio	id
0	2.5 a 7.5	1
1	7.5 a 10	2
2	10 a 12.5	3
3	12.5 a 15	4
4	15 a 17.5	5
5	17.5 a 20	6
6	20 a 25	7
7	25 a 50	8
8	50 a 75	9
9	75 a 85	10
10	atípico	11

## Tabla de Hechos

### Hechos financieros

```
In [38]: th_fin = df[['RatecodeID', 'PULocationID', 'DOLocationID', 'payment_type',
                 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
                 'improvement_surcharge', 'total_amount']+campos_fecha].copy()
th_fin=th_fin.rename(columns={'payment_type':'id_payment_type',
                             'fare_amount':'fare','tip_amount':'tips',
                             'tolls_amount':'tolls'})
th_fin.head()
```

	RatecodeID	PULocationID	DOLocationID	id_payment_type	fare	extra	mta_tax	tips	tolls	improvement_surcharge	total_amount	tpep_pickup_datetime
0	1	141	137		1	8.0	0.0	0.5	1.76	0.0	0.3	10.56
1	1	231	170		1	26.5	0.0	0.5	5.46	0.0	0.3	32.76
2	1	163	68		2	9.5	1.0	0.5	0.00	0.0	0.3	11.30
3	1	163	146		2	13.0	0.5	0.5	0.00	0.0	0.3	14.30
4	1	161	164		1	9.0	0.0	0.5	1.00	0.0	0.3	10.80

```
In [39]: id_rango.pop() #Aregar corte por rangos de monto total.
th_fin['id_rango_precio']=pd.cut(th_fin['total_amount'],bins=corte,
                                 labels=id_rango,include_lowest=True).astype(str)
th_fin.id_rango_precio=th_fin.id_rango_precio.replace('nan','11.0')
th_fin.id_rango_precio=th_fin.id_rango_precio.astype(float).astype(int)
th_fin.drop('total_amount',axis=1,inplace=True) #Eliminamos monto total porque
                                              #tiene dependencia funcional
```

```
In [40]: th_fin.head()
```

	RatecodeID	PULocationID	DOLocationID	id_payment_type	fare	extra	mta_tax	tips	tolls	improvement_surcharge	tpep_pickup_datetime	tpep_dropoff_datetime
0	1	141	137		1	8.0	0.0	0.5	1.76	0.0	0.3	2018-02-03 12:00:00
1	1	231	170		1	26.5	0.0	0.5	5.46	0.0	0.3	2018-08-02 13:30:00
2	1	163	68		2	9.5	1.0	0.5	0.00	0.0	0.3	2018-03-26 18:30:00
3	1	163	146		2	13.0	0.5	0.5	0.00	0.0	0.3	2018-04-29 03:45:00
4	1	161	164		1	9.0	0.0	0.5	1.00	0.0	0.3	2018-10-11 13:15:00

```
In [41]: for f in campos_fecha:
    th_fin = th_fin.merge(td_tiempo[['tiempo','id']],
                          left_on=f,
                          right_on='tiempo',
                          how='inner').drop('tiempo',axis=1).rename(columns={'id':f'{id}_{f}'})
```

```
In [42]: th_fin.head()
```

	RatecodeID	PULocationID	DOLocationID	id_payment_type	fare	extra	mta_tax	tips	tolls	improvement_surcharge	id_rango_precio	id_tpep_pickup_datetime	id_tpep_dropoff_datetime
0	1	141	137		1	8.0	0.0	0.5	1.76	0.0	3		
1	1	144	261		2	16.0	0.0	0.5	0.00	0.0	5		
2	1	264	264		1	15.5	0.0	0.5	1.00	0.0	5		
3	1	48	41		1	19.5	0.0	0.5	3.05	0.0	7		
4	1	230	142		1	6.5	0.0	0.5	1.46	0.0	2		

```
segm = ['PULocationID','RatecodeID','id_rango_precio','id_payment_type'] th_fin = th_fin.groupby(segm).agg(['sum','count']) thfin.columns = ["{}{}".format(x, y) for x in th_fin.columns] #Para eliminar la multicolumna. th_fin.reset_index(inplace=True) th_fin.drop(['fare_count','extra_count','mta_tax_count','tips_count','tolls_count'],axis=1,inplace=True)
```

```
In [43]: segm = ['id_tpep_pickup_datetime','id_tpep_dropoff_datetime',
            'PULocationID','DOLocationID','RatecodeID','id_rango_precio','id_payment_type']
th_fin = th_fin.groupby(segm).agg(['sum','count'])
```

```
th_fin.columns = [ "_" .join(x) for x in th_fin.columns] #Para eliminar la multicolumna.  
th_fin.reset_index(inplace=True)  
th_fin.drop(['fare_count','extra_count','mta_tax_count','tips_count','tolls_count'],axis=1,inplace=True)
```

```
In [44]: th_fin.head()
```

```
Out[44]:   id_tpep_pickup_datetime  id_tpep_dropoff_datetime  PULocationID  DOLocationID  RatecodeID  id_rango_precio  id_payment_type  fare_sum  extra_sum  
0               1                   1          107         186           1            1             2       6.0      0.5  
1               1                   1          141         140           1            1             1       4.5      0.5  
2               1                   1          265         265           5            9             1      55.0      0.0  
3               1                   2          48          68           1            3             1       7.5      0.5  
4               1                   2          79          211           1            3             1       8.5      0.5
```

```
In [27]: td_tiempo.drop('tiempo',axis=1,inplace=True)  
td_tiempo.to_sql(con=engine,if_exists='append',index=False,name='TD_TIEMPO')
```

```
In [46]: td_tiempo
```

```
Out[46]:    minuto  hora  anio  mes  dia  dia_sem  sem_anio  id  
0        0     0  2018    1    1     Mon      0      1  
1      15     0  2018    1    1     Mon      0      2  
2      30     0  2018    1    1     Mon      0      3  
3      45     0  2018    1    1     Mon      0      4  
4        0     1  2018    1    1     Mon      0      5  
...     ...    ...  ...  ...  ...    ...  ...  ...  
35011    45    22  2018   12   31     Mon      52  35012  
35012    0    23  2018   12   31     Mon      52  35013  
35013    15   23  2018   12   31     Mon      52  35014  
35014    30   23  2018   12   31     Mon      52  35015  
35015    45   23  2018   12   31     Mon      52  35016
```

35016 rows × 8 columns

```
In [29]: td_lugar.to_sql(con=engine,name='TD_LUGAR',if_exists='append',index=False)
```

```
In [30]: td_medio_pago.to_sql(con=engine,name='TD_MEDIO_PAGO',if_exists='append',index=False)
```

```
In [31]: td_ratecode.to_sql(con=engine,name='TD_RATECODE',if_exists='append',index=False)
```

```
In [32]: td_rango_precio.to_sql(con=engine,name='TD_RANGO_PRECIO',if_exists='append',index=False)
```

```
In [47]: th_fin.rename(columns=dict(zip(['fare_sum', 'extra_sum', 'mta_tax_sum', 'tips_sum', 'tolls_sum',  
        'improvement_surcharge_sum', 'improvement_surcharge_count'],  
        ['total_tarifas','total_extras','total_taxes','total_propinas','total_peajes','total_sobrecargos','num_viajes'])),inplace=True  
for s in segm:  
    th_fin[s] = th_fin[s].astype(int) #Especificar tipo de dato a cada columna para que entre así a mysql.  
    th_fin['id'] = th_fin.index+1 #Agregar un id.  
th_fin.to_sql(con=engine,if_exists='append',index=False,name='TH_FIN',chunksize=50000)
```

```
In [48]: th_fin.head()
```

```
Out[48]:   id_tpep_pickup_datetime  id_tpep_dropoff_datetime  PULocationID  DOLocationID  RatecodeID  id_rango_precio  id_payment_type  total_tarifas  total_extr  
0               1                   1          107         186           1            1             2       6.0  
1               1                   1          141         140           1            1             1       4.5  
2               1                   1          265         265           5            9             1      55.0  
3               1                   2          48          68           1            3             1       7.5  
4               1                   2          79          211           1            3             1       8.5
```

```
In [49]: th_fin.shape
```

```
Out[49]: (997167, 15)
```

### 8.3. Funciones AE

#### Funciones(Para el análisis exploratorio)

Modulo para crear clases de mapas por zonas.

```
In [ ]:
import pandas as pd
import numpy as np
import matplotlib as mpl
from shapely.geometry import Polygon
from descartes.patch import PolygonPatch
import math
import random
import itertools
import matplotlib.pyplot as plt

def get_boundaries(sf):
    lat, lon = [], []
    for shape in list(sf.iterShapes()):
        lat.extend([shape.bbox[0], shape.bbox[2]])
        lon.extend([shape.bbox[1], shape.bbox[3]])

    margin = 0.01 # buffer to add to the range
    lat_min = min(lat) - margin
    lat_max = max(lat) + margin
    lon_min = min(lon) - margin
    lon_max = max(lon) + margin

    return lat_min, lat_max, lon_min, lon_max

def draw_region_map(ax,sf,shp_dic,heat={}):
    continent = [235/256, 151/256, 78/256]
    ocean = (89/256, 171/256, 227/256)

    reg_list=[{'Staten Island':1, 'Queens':2, 'Bronx':3, 'Manhattan':4, 'EMR':5, 'Brooklyn':6}
    reg_x = {'Staten Island':[], 'Queens':[], 'Bronx':[], 'Manhattan':[], 'EMR':[], 'Brooklyn':[]}
    reg_y = {'Staten Island':[], 'Queens':[], 'Bronx':[], 'Manhattan':[], 'EMR':[], 'Brooklyn':[]}

    # colorbar
    if len(heat) == 0:
        norm = mpl.colors.Normalize(vmin=math.sqrt(min(heat.values()))), vmax=math.sqrt(max(heat.values())))
        cm=plt.cm.get_cmap('Wistia')
        sm = plt.cm.ScalarMappable(cmap=cm, norm=norm)
        sm.set_array([])
        plt.colorbar(sm, ticks=np.linspace(min(heat.values()),max(heat.values()),8), 
                    boundaries=np.arange(min(heat.values())-10,max(heat.values())+10,.1))

    ax.set_facecolor(ocean)
    for sr in sf.shapeRecords():
        shape = sr.shape
        rec = sr.record
        reg_name = rec[shp_dic['borough']]

        if len(heat) == 0:
            norm = mpl.colors.Normalize(vmin=1,vmax=6) #norm = mpl.colors.LogNorm(vmin=1,vmax=max(heat))
            cm=plt.cm.get_cmap('Pastel1')
            R,G,B,A = cm(norm(reg_list[reg_name]))
            col = [R,G,B]
        else:
            R,G,B,A = cm(norm(math.sqrt(heat[reg_name])))
            col = [R,G,B]

        # check number of parts (could use MultiPolygon class of shapely?)
        nparts = len(shape.parts) # total parts
        if nparts == 1:
            polygon = Polygon(shape.points)
            patch = PolygonPatch(polygon, facecolor=col, alpha=1.0, zorder=2)
            ax.add_patch(patch)
        else: # loop over parts of each shape, plot separately
            for ip in range(nparts): # loop over parts, plot separately
                i0 = shape.parts[ip]
                if ip < nparts-1:
                    i1 = shape.parts[ip+1]-1
                else:
                    i1 = len(shape.points)

                polygon = Polygon(shape.points[i0:i1+1])
                patch = PolygonPatch(polygon, facecolor=col, alpha=1.0, zorder=2)
                ax.add_patch(patch)

        reg_x[reg_name].append((shape.bbox[0]+shape.bbox[2])/2)
        reg_y[reg_name].append((shape.bbox[1]+shape.bbox[3])/2)

    for k in reg_list:
        if len(heat)==0:
            plt.text(np.mean(reg_x[k]), np.mean(reg_y[k]), k, horizontalalignment='center', verticalalignment='center',
                    bbox=dict(facecolor='black', alpha=0.5), color='white', fontsize=12)
        else:
            plt.text(np.mean(reg_x[k]), np.mean(reg_y[k]), "{}\n({})".format(k, heat[k]/1000), horizontalalignment='center',
                    verticalalignment='center',bbox=dict(facecolor='black', alpha=0.5), color='white', fontsize=12)

    # display
    limits = get_boundaries(sf)
    plt.xlim(limits[0],limits[1])
    plt.ylim(limits[2],limits[3])

def draw_zone_map(ax,sf,shp_dic,heat={}, text=[], arrows[]):
```

```

limits = get_boundaries(sf)
continent = [235/256, 151/256, 78/256] #color continente
ocean = (89/256, 171/256, 227/256) #color oceano
theta = np.linspace(0, 2*np.pi, len(text)+1).tolist() #separar el mapa en angulos iguales
ax.set_facecolor(ocean)
#Barra de colores
if len(heat) != 0:
    norm = mpl.colors.Normalize(vmin=min(heat.values()), vmax=max(heat.values()))
    #Normaliza la barra dependiendo de los valores
    #maximo y minimo de los conteos de viajes por zona
    #norm = mpl.colors.LogNorm(vmin=1, vmax=max(heat)) <- para tener una escala logaritmica
    cm=plt.get_cmap('rainbow') #color map
    sm=plt.cm.ScalarMappable(cmap=cm, norm=norm)
    sm.set_array([])
    plt.colorbar(sm, ticks=np.linspace(min(heat.values()),max(heat.values()),8), #Barra
                 boundaries=np.arange(min(heat.values())-0.00001,max(heat.values())+0.00001,0.000001))

for sr in sf.shapeRecords(): #Contiene la informacion de cada zona
    shape = sr.shape #Datos de la forma de la zona
    rec = sr.record #Datos cualitativos de la zona
    loc_id = rec[shp_dic['LocationID']] #ID de la zona
    zone = rec[shp_dic['zone']] #Nombre de la zona

    #Asignacion de color a cada zona
    if len(heat) == 0:
        col = continent
    else:
        if loc_id not in heat:
            R,G,B,A = cm(norm(0))
        else:
            R,G,B,A = cm(norm(heat[loc_id]))
        col = [R,G,B]

    #Dibujar el poligono de la zona
    nparts = len(shape.parts) #partes de conforman la zona
    if nparts == 1: #Si esta conformado por un solo poligono
        polygon = Polygon(shape.points) #Dibujo de la forma
        patch = PolygonPatch(polygon, facecolor=col, alpha=1.0, zorder=2)
        ax.add_patch(patch)
    else: #Confirmado or varios poligonos
        for i in range(nparts):
            u = shape.parts[i]
            if i < nparts-1:
                v = shape.parts[i+1]-1
            else:
                v = len(shape.points)
            polygon = Polygon(shape.points[u:v+1])
            patch = PolygonPatch(polygon, facecolor=col, alpha=1.0, zorder=2)
            ax.add_patch(patch)

    x = (shape.bbox[0]+shape.bbox[2])/2 #Coordenadas centrales de la zona
    y = (shape.bbox[1]+shape.bbox[3])/2

    if len(text) != 0 and loc_id in text: #Etiquetar las zonas de interes
        plt.text(x+0.01, y-0.01, str(loc_id), fontsize=12, color="white", bbox=dict(facecolor='black', alpha=0.5))
        factor=abs(limits[2]-limits[3])/7.6361
        dx = factor*np.cos(theta[text.index(loc_id)])
        dy = factor*np.sin(theta[text.index(loc_id)])
        ax.annotate('{[{}]} {}'.format(loc_id, zone), xy=(x,y), xytext=(x+dx,y+dy),
                   bbox=dict(facecolor='black',alpha=0.5), color='white', fontsize=12, #Caja
                   arrowprops=dict(facecolor='white',width=3,shrink=0.05))#Flechas

if len(arrows)!=0:
    for arr in arrows:
        ax.annotate('', xy = arr['dest'], xytext = arr['src'], size = arr['cnt'],
                    arrowprops=dict(arrowstyle="fancy", fc="0.6", ec="none"))

#Limites del grafico
plt.xlim(limits[0], limits[1])
plt.ylim(limits[2], limits[3])

```

## 8.4. Muestras

### Clase muestras

Modulo para crear muestras aleatorias.

```
In [ ]: import pandas as pd
import random as rd

class muestra_aleatoria:
    _path='/home/osvaldo/Descargas/nyc_taxi/prueba.csv'

    def __init__(self, size=100000):
        """
        :param size: int, tamaño de la muestra deseada.
        """
        self._size=size
    def _lista_de_aleatorios(self):
        aux=rd.sample(range(1,1000001),self._size)
        aux.insert(0,0)
        return aux
    def crear_muestra(self):
        a=open(self._path,'rb')
        lista=a.readlines()
        a.close()

        b=open('muestra','wb')
        for i in self._lista_de_aleatorios():
            b.write(b'\n'.join(lista[i]))
        b.close()
        x=pd.read_csv('muestra.csv')
        return x
    def distribucion_muestra(self):
        d=pd.DataFrame({'viaje':self._lista_de_aleatorios()})
        print(d.hist())
        print(d.describe())
    def tamaño(self):
        print(self._size)
```

## 8.5. Ingenería y Modelos

### Lectura de datos

```
In [1]: # matplotlib
import pandas as pd
import numpy as np
import random as rd
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_selection import VarianceThreshold
```

### Ruta de datos

```
In [2]: ruta = "/home/christo/Downloads/Compressed"
archivo = "2018_Yellow_Taxi_Trip_Data.csv"
```

### Carga y procesamiento de los datos

En principio solo es necesario conocer la cantidad de viajes por hora durante el 2018, y construiremos la matriz predictiva a partir de esos datos. Creamos una función que nos da la hora en formato 24 horas, esto para que sea más rápido el código.

```
In [3]: def hora(hora, meridiano):
    if meridiano == "AM":
        return hora % 12
    elif meridiano == "PM":
        return 12 + (hora % 12)
```

```
In [4]: # Manipulación de datos por chunks
chunk_list = []
contador = 0
for chunk in pd.read_csv(os.path.join(ruta,archivo), usecols = ["tpep_pickup_datetime"], chunksize = 500000, iterator = True):
    tiempo = pd.DataFrame()
    tiempo["fecha"] = chunk["tpep_pickup_datetime"].map(lambda x: x[:5]) # solo le falta el año
    tiempo["hora"] = chunk["tpep_pickup_datetime"].map(lambda x: hora(int(x[11:13]), x[-2:]))
    tiempo["viajes"] = 1 # Creamos contador de filas(viajes)
    tiempo = tiempo.groupby(["fecha", "hora"], as_index = False, sort = False).aggregate("count")
    contador += 1
    print(contador)
    chunk_list.append(tiempo)
```

```
In [5]: # Concatenando los chunks
datos = pd.concat(chunk_list, axis = 0).groupby(["fecha", "hora"], as_index = False, sort = False).aggregate("sum")
```

```
In [6]: # Guardando datos para no correr la manipulación
datos.to_pickle("./datos.pkl")
```

```
In [7]: datos = pd.read_pickle("./datos.pkl")
```

```
In [8]: # Dandole formato a la fecha con pd.to_datetime
datos["fecha"] = pd.to_datetime(datos["fecha"].map(lambda x: x + "/2018"), format = "%m/%d/%Y")
datos = datos.sort_values(["fecha", "hora"]).reset_index(drop = True) # Ordenamos
```

```
In [9]: datos
```

```
Out[9]:   fecha hora viajes
0 2018-01-01 0 16783
1 2018-01-01 1 19061
2 2018-01-01 2 16801
3 2018-01-01 3 12631
4 2018-01-01 4 8740
...
8755 2018-12-31 19 14878
8756 2018-12-31 20 14435
8757 2018-12-31 21 14117
8758 2018-12-31 22 10741
8759 2018-12-31 23 8642
8760 rows × 3 columns
```

### Catálogo de fechas(por unidad muestral)

Los ID's serán por cada hora del día del año puesto que queremos predecir el volumen de viajes de la siguiente hora.

```
In [10]: cat = datos[["fecha", "hora"]].drop_duplicates().sort_values(by = ["fecha", "hora"], reset_index(drop = True))
cat["hora_dia"] = cat["fecha"].map(lambda x: x.strftime("%Ww")) + cat["hora"].astype(str)
cat["idh"] = cat.index
cat
```

```
Out[10]:      fecha  hora  hora_dia  idh
0  2018-01-01    0    0110      0
1  2018-01-01    1    0111      1
2  2018-01-01    2    0112      2
3  2018-01-01    3    0113      3
4  2018-01-01    4    0114      4
...
8755 2018-12-31   19    53119    8755
8756 2018-12-31   20    53120    8756
8757 2018-12-31   21    53121    8757
8758 2018-12-31   22    53122    8758
8759 2018-12-31   23    53123    8759
```

8760 rows × 4 columns

```
In [11]: datos = datos.merge(cat.drop("hora_dia", axis = 1), on = ["fecha", "hora"],
                           how = "left").drop(["fecha", "hora"], axis = 1)
```

```
In [12]: datos
```

```
Out[12]:      viajes  idh
0    16783    0
1    19061    1
2    16601    2
3    12631    3
4     8740    4
...
8755  14876  8755
8756  14435  8756
8757  14117  8757
8758  10741  8758
8759   8642  8759
```

8760 rows × 2 columns

## Ingeniería de datos (crear matriz $\mathcal{X}$ y vector $\vec{y}$ )

```
In [13]: # Matriz predictiva usando los viajes de horas anteriores
n = 24 # NÚMERO DE HORAS ANTERIORES
matriz_pred = pd.DataFrame([[datos["idh"]][i + n]] +
                           list(datos["viajes"])[(0 + i):(n + 1 + i)])  for i in range(len(datos) - n))

# Renombrando columnas
varx = ["Ant_" + str(25-int(v)) for v in matriz_pred.columns[1:-1]]
new_columns = ["idh"]+ varx + ["Y"]
matriz_pred.columns = new_columns

# Matriz X,y
X = matriz_pred.iloc[:,0:-1].copy()
y = matriz_pred.iloc[:,[-1]].copy()

matriz_pred.tail()
```

```
Out[13]:      idh  Ant_24  Ant_23  Ant_22  Ant_21  Ant_20  Ant_19  Ant_18  Ant_17  Ant_16 ...  Ant_9  Ant_8  Ant_7  Ant_6  Ant_5  Ant_4  Ant_3  Ant_2
8731  8755    10710   9092   8950   8146   7166   5878   3876   2565   1806 ...   8197   9516   10632   12619   14485   13307   13010   13057
8732  8756    9092   8950   8146   7166   5878   3876   2565   1806   1527 ...   9516   10632   12619   14485   13307   13010   13057   14265
8733  8757    8950   8146   7166   5878   3876   2565   1806   1527   1684 ...   10632   12619   14485   13307   13010   13057   14265   14876
8734  8758    8146   7166   5878   3876   2565   1806   1527   1684   3312 ...   12619   14485   13307   13010   13057   14265   14876   14435
8735  8759    7166   5878   3876   2565   1806   1527   1684   3312   4403 ...   14485   13307   13010   13057   14265   14876   14435   14117
```

5 rows × 26 columns

```
In [46]: y
```

```
Out[46]:      idh      Y
0     24    3584
1     25    1724
2     26    1205
3     27     918
4     52   1368
```

```

    idh      Y
    ...     ...
8441  8755  14876
8442  8756  14435
8443  8757  14117
8444  8758  10741
8445  8759  8642

```

8446 rows × 2 columns

In [47]: X

```

Out[47]:   idh  Ant_24  Ant_23  Ant_22  Ant_21  Ant_20  Ant_19  Ant_18  Ant_17  Ant_16  ...  Ant_10  Ant_9  Ant_8  Ant_7  Ant_6  Ant_5  Ant_4  Ant_3
0    24    16783   19061   16601   12631    8740    4229    3414    3433    3643  ...   12415   13277   12552   12759   11821   10458   8357   8852
1    25    19061   16601   12631    8740    4229    3414    3433    3643    4940  ...   13277   12552   12759   11821   10458   8357   8852   7906
2    26    16601   12631    8740    4229    3414    3433    3643    4940    6974  ...   12552   12759   11821   10458   8357   8852   7906   5411
3    27    12631    8740    4229    3414    3433    3643    4940    6974    9984  ...   12759   11821   10458   8357   8852   7906   5411   3584
4    52    1214     2865    7050   10914   13732   13129   12049   12284   12901  ...   17349   14390   12371   11812   10276   6590   4547   2614
...   ...     ...     ...     ...     ...     ...     ...     ...     ...     ...  ...   ...     ...     ...     ...     ...     ...     ...     ...
8441  8755  10710   9092   8950   8146   7166   5878   3876   2565  ...   1806     ...   7044   8197   9516   10632   12619   14485   13307   13010
8442  8756  9092   8950   8146   7166   5878   3876   2565   1806  ...   1527     ...   8197   9516   10632   12619   14485   13307   13010   13057
8443  8757  8950   8146   7166   5878   3876   2565   1806   1527  ...   1684     ...   9516   10632   12619   14485   13307   13010   13057   14265
8444  8758  8146   7166   5878   3876   2565   1806   1527   1684  ...   3312     ...   10632   12619   14485   13307   13010   13057   14265   14876
8445  8759   7166   5878   3876   2565   1806   1527   1684   3312  ...   4403     ...   12619   14485   13307   13010   13057   14265   14876   14435

```

8446 rows × 25 columns

## Análisis Exploratorio

### Datos Ausentes

In [16]: X.isna().any().sum() # No hay datos ausentes

Out[16]: 0

### Varianza Nula

No hay variables con varianza nula.

In [17]: vt = VarianceThreshold()  
vt.fit(X[varx])  
varx = [x for x,y in zip(varx, vt.get\_support()) if y]

### Datos Extremos

Método de percentil  $P_1$ ,  $P_{99}$

In [18]: lim\_ext = X[varx].describe(percentiles = [0.001, 0.999]).T[['0.1%', "99.9%"]].reset\_index().values.tolist()

In [19]: for col,li,ls in lim\_ext:  
 X[f"ev\_{col}"] = (X[col] < li)|(ls < X[col]).astype(int)  
var\_ext = [v for v in X.columns if v[:2] == "ev"]

In [20]: X["extremos"] = X[var\_ext].max(axis = 1)

In [21]: X["extremos"].value\_counts(True) # No borramos datos extremos

Out[21]: False 0.966804  
True 0.033196  
Name: extremos, dtype: float64

In [22]: X = X.loc[X['extremos']!=1].reset\_index(drop=True).drop(var\_ext +['extremos'],axis=1)

In [23]: y = y.merge(X["idh"],on="idh",how='inner')

In [24]: X

```

Out[24]:   idh  Ant_24  Ant_23  Ant_22  Ant_21  Ant_20  Ant_19  Ant_18  Ant_17  Ant_16  ...  Ant_10  Ant_9  Ant_8  Ant_7  Ant_6  Ant_5  Ant_4  Ant_3
0    24    16783   19061   16601   12631    8740    4229    3414    3433    3643  ...   12415   13277   12552   12759   11821   10458   8357   8852
1    25    19061   16601   12631    8740    4229    3414    3433    3643    4940  ...   13277   12552   12759   11821   10458   8357   8852   7906
2    26    16601   12631    8740    4229    3414    3433    3643    4940    6974  ...   12552   12759   11821   10458   8357   8852   7906   5411
3    27    12631    8740    4229    3414    3433    3643    4940    6974    9984  ...   12759   11821   10458   8357   8852   7906   5411   3584
4    52    1214     2865    7050   10914   13732   13129   12049   12284   12901  ...   17349   14390   12371   11812   10276   6590   4547   2614

```

	idh	Ant_24	Ant_23	Ant_22	Ant_21	Ant_20	Ant_19	Ant_18	Ant_17	Ant_16	...	Ant_10	Ant_9	Ant_8	Ant_7	Ant_6	Ant_5	Ant_4	Ant_3
8441	8755	10710	9092	8950	8146	7166	5878	3876	2565	1806	...	7044	8197	9516	10632	12619	14485	13307	13010
8442	8756	9092	8950	8146	7166	5878	3876	2565	1806	1527	...	8197	9516	10632	12619	14485	13307	13010	13057
8443	8757	8950	8146	7166	5878	3876	2565	1806	1527	1684	...	9516	10632	12619	14485	13307	13010	13057	14265
8444	8758	8146	7166	5878	3876	2565	1806	1527	1684	3312	...	10632	12619	14485	13307	13010	13057	14265	14876
8445	8759	7166	5878	3876	2565	1806	1527	1684	3312	4403	...	12619	14485	13307	13010	13057	14265	14876	14435

8446 rows × 25 columns

y

In [25]:

Out[25]: **idh** **Y**

<b>0</b>	24	3584
<b>1</b>	25	1724
<b>2</b>	26	1205
<b>3</b>	27	918
<b>4</b>	52	1368
...	...	...
<b>8441</b>	8755	14876
<b>8442</b>	8756	14435
<b>8443</b>	8757	14117
<b>8444</b>	8758	10741
<b>8445</b>	8759	8642

8446 rows × 2 columns

Modelos

## Árbol de Regresión

```
In [26]: from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, SCORERS, r2_score

# Setting parameters
hyper_param = dict(criterion = ["mse", "mae"], # Puede agregarse friedman_mse, son metricas.
                    splitter = ["best"],
                    max_depth = range(2,50,5),
                    min_samples_leaf = [50,100,150]) # Si de las carac. mira a todos o toma una aleatoria

# Splitting data
Xt, Xv, yt, yv = train_test_split(X[varx],y["Y"],train_size = 0.7)

# Shape
print(Xt.shape, Xv.shape, yt.shape, yv.shape)

# Creating model
modelo = DecisionTreeRegressor()

# Creating grid
# https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation
grid = GridSearchCV(estimator = modelo, #Modelo sujeto de la hiperparametrización
                     param_grid = hyper_param, #Espacio paramétrico
                     cv = 3, #Número de dobleces (Folds)
                     n_jobs = -1, #Procesadores a usar (-1 == todos)
                     scoring = 'neg_mean_absolute_error', #Métrica para seleccionar el mejor modelo
                     verbose = True) #Nos informa del estado del entrenamiento
```

```
In [27]: grid.fit(Xt,yt)
```

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks   | elapsed:   4.2s
[Parallel(n_jobs=-1)]: Done 180 out of 180   | elapsed:  54.8s finished
```

```
[4]: fitter(n_jobs=1), bins=100, cut=0, n_estimators=100, max_depth=5, min_samples_leaf=50, min_samples_split=100, criterion='mse', splitter='best'), scoring='neg_mean_absolute_error', verbose=True)
```

```
In [28]: print("Mejores parametros: ", grid.best_params_)
print("Mejor std: ", pd.DataFrame(grid.cv_results_).sort_values("rank_test_score")["std_test_score"][0])
print("Mejor score: ", np.abs(grid.best_score_))
```

```
Mejores parametros:  {'criterion': 'mae', 'max_depth': 17, 'min_samples_leaf': 50, 'splitter': 'best'}
Mejor std:  20.48152463657
Mejor score: 10597.628192320243
```

```
In [29]: #modelo = DecisionTreeRegressor(criterion = "mae", max_depth = 13, min_samples_leaf=2, splitter = "best")
modelo = grid.best_estimator_ # Elegimos el mejor modelo
modelo.fit(Xt,yt) # Entrenamos el modelo
yt_predict = np.round(modelo.predict(Xt)) # Nos interesan enteros por eso redondeamos
yv_predict = np.round(modelo.predict(Xv))
```

```
In [30]: print("MAE para entrenamiento: ", mean_absolute_error(y_true = yt, y_pred = yt_predict))
print("MAE para validación: ", mean_absolute_error(y_true = yv, y_pred = yv_predict))

MAE para entrenamiento: 869.9599120433018
MAE para validación: 967.8425414364641
```

```
In [31]: print("MSE para entrenamiento: ", mean_squared_error(y_true = yt, y_pred = yt_predict))
print("MSE para validación: ", mean_squared_error(y_true = yv, y_pred = yv_predict))

MSE para entrenamiento: 2418206.4521312583
MSE para validación: 2442184.043804262
```

```
In [32]: print("El error relativo para validación es: ", np.abs((yv-yv_predict)/yv).mean())
print("El error relativo para entrenamiento es: ", np.abs((yt-yt_predict)/yt).mean())

El error relativo para validación es: 0.10183077879001022
El error relativo para entrenamiento es: 1.3289728438031254
```

```
In [ ]: # Gráfica metricas
mae = [mean_absolute_error(y_true = yt, y_pred = yt_predict),
       mean_absolute_error(y_true = yv, y_pred = yv_predict)]

mse = [mean_squared_error(y_true = yt, y_pred = yt_predict),
       mean_squared_error(y_true = yv, y_pred = yv_predict)]

nombres = ["entrenamiento", "validación"]

# Gráficas MAE y MSE
plt.get_cmap("cool")
plt.figure(figsize=(15,7))

plt.subplot(1, 2, 1)
plt.bar(nombres, mae, color = (0.2, 0.4, 0.6, 0.6))
for i in range(len(mae)):
    plt.annotate(str(round(mae[i])), xy = (nombres[i], mae[i]), ha='center', va='bottom')
plt.ylabel("Error")
plt.title("MAE")

plt.subplot(1, 2, 2)
plt.bar(nombres, mse, color = (0.2, 0.4, 0.6, 0.6))
for i in range(len(mse)):
    plt.annotate(str(round(mse[i])), xy = (nombres[i], mse[i]), ha='center', va='bottom')
plt.ylabel("Error")
plt.title("MSE")

plt.show()
```

```
In [ ]: # Gráficas ajuste
plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)
plt.scatter(yt.reset_index(drop= True), yt_predict, c='crimson')
p1 = max(max(yt_predict), max(yt))
p2 = min(min(yt_predict), min(yt))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.title('Training Data')

plt.subplot(1, 2, 2)
plt.scatter(yv.reset_index(drop= True), yv_predict, c='crimson')
p1 = max(max(yv_predict), max(yv))
p2 = min(min(yv_predict), min(yv))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.title('Validation Data')

plt.show()
```

```
In [ ]: # Gráficas densidad y distribución acumulada
plt.figure()
sns.distplot(yt,hist=False,kde_kws={'cumulative':True})
sns.distplot(yt_predict,hist=False,kde_kws={'cumulative':True})
plt.title("Distribución acumulada: Entrenamiento")
plt.figure()
sns.distplot(yv,hist=False,kde_kws={'cumulative':True})
sns.distplot(yv_predict,hist=False,kde_kws={'cumulative':True})
plt.title("Distribución acumulada: Validación")
plt.figure()
sns.distplot(yt,hist=False,kde_kws={'cumulative':False})
sns.distplot(yt_predict,hist=False,kde_kws={'cumulative':False})
plt.title("Densidad: Entrenamiento")
plt.figure()
sns.distplot(yv,hist=False,kde_kws={'cumulative':False})
sns.distplot(yv_predict,hist=False,kde_kws={'cumulative':False})
plt.title("Densidad: Validación")
```

```
In [36]: from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, SCORERS, r2_score

# Setting parameters
hyper_param = dict(hidden_layer_sizes=[(a,b,c,) for a in range(20,30,1) for b in range(30,40,1) for c in range(20,30,1)],
                   activation = ['identity', 'logistic', 'tanh', 'relu'],
                   alpha = np.arange(0.07,0.8,0.0001),
                   learning_rate = ['constant', 'invscaling', 'adaptive'],
                   )
# Splitting data
Xt, Xv, yt, yv = train_test_split(X[varx],y["Y"],train_size = 0.7)

# Shape
print(Xt.shape, Xv.shape, yt.shape, yv.shape)

# Creating model
modelo = MLPRegressor()

# Creating grid
# https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation
grid = RandomizedSearchCV(estimator = modelo, #Modelo sujeto de la hiperparametrización
                           n_iter = 500,
                           param_distributions = hyper_param, #Espacio paramétrico
                           cv = 3, #Número de dobleces (Folds)
                           n_jobs = 4, #Procesadores a usar (-1 == todos)
                           scoring = 'neg_mean_absolute_error', #Métrica para seleccionar el mejor modelo
                           verbose = True) #Nos informa del estado del entrenamiento
```

(5912, 24) (2534, 24) (5912,) (2534,)

```
In [37]: grid.fit(Xt,yt)
```

Fitting 3 folds for each of 500 candidates, totalling 1500 fits

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 42 tasks      | elapsed:  44.8s
[Parallel(n_jobs=4)]: Done 192 tasks     | elapsed:  3.1min
[Parallel(n_jobs=4)]: Done 442 tasks     | elapsed:  7.2min
[Parallel(n_jobs=4)]: Done 792 tasks     | elapsed: 12.5min
[Parallel(n_jobs=4)]: Done 1242 tasks    | elapsed: 20.0min
[Parallel(n_jobs=4)]: Done 1500 out of 1500 | elapsed: 24.6min finished
```

```
Out[37]: RandomizedSearchCV(cv=3, estimator=MLPRegressor(), n_iter=500, n_jobs=4,
                            param_distributions={'activation': ['identity', 'logistic',
                                                               'tanh', 'relu'],
                                                 'alpha': array([0.07 , 0.0701, 0.0702, ..., 0.7997, 0.7998, 0.7999]),
                                                 'hidden_layer_sizes': [(20, 30, 20),
                                                       (20, 30, 21),
                                                       (20, 30, 22),
                                                       (20, 30, 23),
                                                       (20, 30, 24),
                                                       (20, 30, 25),
                                                       (20, 30, 26),
                                                       (20, 30, 27),
                                                       (20, 30, 28),
                                                       (20, 30, 29),
                                                       (20, 31, 20),
                                                       (20, 31, 21),
                                                       (20, 31, 22),
                                                       (20, 31, 23),
                                                       (20, 31, 24),
                                                       (20, 31, 25),
                                                       (20, 31, 26),
                                                       (20, 31, 27),
                                                       (20, 31, 28),
                                                       (20, 31, 29),
                                                       (20, 32, 20),
                                                       (20, 32, 21),
                                                       (20, 32, 22),
                                                       (20, 32, 23),
                                                       (20, 32, 24),
                                                       (20, 32, 25),
                                                       (20, 32, 26),
                                                       (20, 32, 27),
                                                       (20, 32, 28),
                                                       (20, 32, 29), ...],
                                                 'learning_rate': ['constant',
                                                       'invscaling',
                                                       'adaptive']},
                            scoring='neg_mean_absolute_error', verbose=True)
```

```
In [38]: print("Mejores parametros: ", grid.best_params_)
print("Desviación estándar: ", pd.DataFrame(grid.cv_results_).sort_values(by = "rank_test_score").reset_index()["std_test_s"]
print("Mejor score: ", np.abs(grid.best_score_))
```

Mejores parametros: {'learning\_rate': 'adaptive', 'hidden\_layer\_sizes': (23, 38, 22), 'alpha': 0.3809000000000089, 'activation': 'relu'}  
Desviación estándar: 24.858579602225234  
Mejor score: 634.3589660281356

```
In [39]: #modelo = MLPRegressor(hidden_layer_sizes = 98, activation = "identity", solver = "lbfgs", alpha = 0.837699999999995, learn.
modelo = grid.best_estimator_ # Elegí mos el mejor modelo
modelo.max_iter = len(matriz_pred)*2 # Max iteraciones
modelo.fit(Xt,yt) # Entrenamos el modelo
yt_predict = np.round(modelo.predict(Xt)) # Nos interesan enteros por eso redondeamos
yv_predict = np.round(modelo.predict(Xv))
```

```
In [40]: print("MAE para entrenamiento: ", mean_absolute_error(y_true = yt, y_pred = yt_predict))
print("MAE para validación: ", mean_absolute_error(y_true = yv, y_pred = yv_predict))
```

```
MAE para entrenamiento: 587.0487144790258
MAE para validación: 613.0481452249408
```

```
In [41]: print("MSE para entrenamiento: ", mean_squared_error(y_true = yt, y_pred = yt_predict))
print("MSE para validación: ", mean_squared_error(y_true = yv, y_pred = yv_predict))

MSE para entrenamiento: 836515.9076454669
MSE para validación: 870110.4191002367
```

```
In [42]: print("El error relativo para validación es: ", np.abs((yv-yv_predict)/yv).mean())
print("El error relativo para entrenamiento es: ", np.abs((yt-yt_predict)/yt).mean())

El error relativo para validación es: 0.06449642841577301
El error relativo para entrenamiento es: 0.9973951320188218
```

```
In [ ]: # Gráfica métricas
mae = [mean_absolute_error(y_true = yt, y_pred = yt_predict),
       mean_absolute_error(y_true = yv, y_pred = yv_predict)]

mse = [mean_squared_error(y_true = yt, y_pred = yt_predict),
       mean_squared_error(y_true = yv, y_pred = yv_predict)]

nombres = ["entrenamiento", "validación"]

# Gráficas MAE y MSE
plt.get_cmap("cool")
plt.figure(figsize=(15,7))

plt.subplot(1, 2, 1)
plt.bar(nombres, mae, color = (0.2, 0.4, 0.6, 0.6))
for i in range(len(mae)):
    plt.annotate(str(round(mae[i])), xy = (nombres[i], mae[i]), ha='center', va='bottom')
plt.ylabel("Error")
plt.title("MAE")

plt.subplot(1, 2, 2)
plt.bar(nombres, mse, color = (0.2, 0.4, 0.6, 0.6))
for i in range(len(mse)):
    plt.annotate(str(round(mse[i])), xy = (nombres[i], mse[i]), ha='center', va='bottom')
plt.ylabel("Error")
plt.title("MSE")

plt.show()
```

```
In [ ]: # Gráficas ajuste
plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)
plt.scatter(yt.reset_index(drop= True), yt_predict, c='crimson')
p1 = max(max(yt_predict), max(yt))
p2 = min(min(yt_predict), min(yt))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.title("Training Data")

plt.subplot(1, 2, 2)
plt.scatter(yv.reset_index(drop= True), yv_predict, c='crimson')
p1 = max(max(yv_predict), max(yv))
p2 = min(min(yv_predict), min(yv))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.title("Validation Data")

plt.show()
```

```
In [ ]: # Gráficas densidad y distribución acumulada
plt.figure()
sns.distplot(yt,hist=False,kde_kws={'cumulative':True})
sns.distplot(yt_predict,hist=False,kde_kws={'cumulative':True})
plt.title("Distribución acumulada: Entrenamiento")
plt.figure()
sns.distplot(yv,hist=False,kde_kws={'cumulative':True})
sns.distplot(yv_predict,hist=False,kde_kws={'cumulative':True})
plt.title("Distribución acumulada: Validación")
plt.figure()
sns.distplot(yt,hist=False,kde_kws={'cumulative':False})
sns.distplot(yt_predict,hist=False,kde_kws={'cumulative':False})
plt.title("Densidad: Entrenamiento")
plt.figure()
sns.distplot(yv,hist=False,kde_kws={'cumulative':False})
sns.distplot(yv_predict,hist=False,kde_kws={'cumulative':False})
plt.title("Densidad: Validación")
```