laSalle ENG Universitat Ramon Llull

Máster en Big Data

Asignatura: Casos de analítica
Fecha: De 19/03/2024 hasta 29/04/2024

Apellidos: Ripoll Batlló Nombre: Albert Curso: **2023/24**

Caso 1: New York City Taxi

Este primer caso gira entorno al sistema de transporte público de la Ciudad de New York (USA). Desde 2009 la *New York City Taxi & Limousine Commission* ha publicado datos acerca de los recorridos hechos por los distintos servicios de Taxi de la ciudad[1]. Para resolver este caso tendréis suficiente con analizar los datos de dos meses cualquiera del año 2023. Tened en cuenta que los Yellow Cabs y los Green Cabs se rigen por normativas distintas en cuanto a recogida de pasajeros, por lo que deberéis combinar los datos de ambas flotas. Con estos datos se os propone que apliquéis técnicas de estadística, analítica, minería de datos y visualización para responder a las siguientes preguntas. No hay restricciones acerca de las técnicas ni tecnologías a utilizar siempre y cuando los resultados sean reproducibles y estén debidamente justificados. Explicitad y detallad todos los pasos hechos para responder a cada pregunta.

PARTE 0: PROPUESTA DE DISEÑO DEL *PIPELINE* DE DATOS.

0. Propón y justifica una arquitectura completa (end-to-end) que permita soportar todo el ciclo de vida de los datos de este caso de uso:
    a. Adquisición de datos: ¿Qué dispositivo se podría utilizar para capturar la información (origen, destino, tiempo, ocupantes, precio, …) acerca de los trayectos de cada uno de los taxis de New York?

Para capturar la información de los trayectos de los taxis de New York se podría utilizar una aplicación móvil común en toda la empresa. Esa aplicación se comunicaría y alimentaría una base de datos a través de tecnología IOT (Internet Of Things).

    b. Infraestructura para el almacenamiento y procesamiento de datos: Detalla qué tecnologías y servicios (*cloud!*) se podrían utilizar para almacenar esta información. Estima el coste mensual que tendría tu solución.

Para almacenar información la aplicación móvil mencionada, se podrían utilizar los siguientes servicios:

• Base de datos en la nube:

- Amazon DynamoDB: Base de datos NoSQL totalmente administrada que proporciona un rendimiento rápido y escalabilidad automática.
- Google Cloud Firestore: Base de datos NoSQL en tiempo real que facilita el almacenamiento y la sincronización de datos entre dispositivos.
- Microsoft Azure Cosmos DB: Base de datos distribuida que admite varios modelos de datos y proporciona una escalabilidad horizontal automática.

• Almacenamiento de datos en la nube:

[1] Los datos están disponibles aquí: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[2] Nueva York tiene en la actualidad una flota de unos 14.000 taxis. Sin embargo, un estudio del MIT asegura que se podría cubrir toda la demanda de la ciudad con solo 3.000 vehículos. https://www.heraldodiariodesoria.es/soria/170103/57950/estudio-reduce-80-taxis-necesarios-grandes-ciudades-nueva-york.html

- Amazon S3 (Simple Storage Service): Servicio de almacenamiento en la nube que proporciona escalabilidad, disponibilidad y durabilidad.
- Google Cloud Storage: Servicio de almacenamiento en la nube que ofrece almacenamiento escalable y seguro para datos de la aplicación.
- Microsoft Azure Blob Storage: Servicio de almacenamiento en la nube que permite almacenar y recuperar grandes cantidades de datos de forma económica.

• Procesamiento de datos:

- Amazon Kinesis: Servicio para la transmisión de datos en tiempo real que permite recopilar, procesar y analizar datos de transmisión en tiempo real.
- Google Cloud Dataflow: Servicio de procesamiento de datos en tiempo real y por lotes que permite la ejecución de pipelines de datos escalables y flexibles.
- Microsoft Azure Stream Analytics: Servicio de análisis de datos en tiempo real que permite el procesamiento de datos en streaming con facilidad y a escala.

• Servicios de IoT:

- AWS IoT Core: Servicio de IoT que facilita la conexión segura de dispositivos IoT a la nube y permite la administración de flotas de dispositivos.
- Google Cloud IoT Core: Servicio de gestión de dispositivos IoT que permite conexión segura y administración de aplicaciones IoT a escala.
- Azure IoT Hub: Servicio de IoT totalmente administrado que facilita conectividad, administración y implementación de aplicaciones IoT a gran escala.

Para estimar el precio mensual de dicho servicio hace falta consultar calculadoras de AWS, Google, Microsoft. Una posible aproximación del coste sería:

• Coste almacenamiento de datos en la nube:

Un servicio de almacenamiento de objetos en la nube tiene un costo aproximado de $0.03 por GB al mes. Por lo que si se supone un requerimiento de unos 100 GB son $3 al mes.

• Coste procesamiento de datos:

Un servicio de procesamiento de datos en tiempo real tiene un costo aproximado de entre $0.05 a $0.10 por GB al mes. Por lo que si se supone un requerimiento de unos 100 GB son de $5 a $10 al mes.

• Coste servicios de IoT:

Un servicio de administración y gestión de dispositivos IoT tiene un costo de entre $0.05 a $0.10 por dispositivo al mes.

Si suponemos que se requiere gestionar una flota de unos 10 000 dispositivos (fuente[2]) son de $50 a $100 al mes.

• Costos adicionales:

Se debe tener en cuenta también otros posibles costos adicionales, como transferencia de datos, consultas a la base de datos, almacenamiento de respaldo, etc.

Con estas suposiciones y rangos, podemos calcular un costo mensual aproximado entre $500 y $1000.

a. Tecnologías para el almacenamiento y procesamiento de datos: Detalla qué tecnologías de almacenamiento y procesado de datos se podrían utilizar para almacenar toda esta información.

Para almacenar información la aplicación móvil mencionada, se podrían utilizar las siguientes tecnologías:

• Almacenamiento de Datos:

- Bases de Datos NoSQL: Como MongoDB, Cassandra o Firebase Realtime Database.
- Bases de Datos SQL: Como PostgreSQL, MySQL o Amazon RDS.
- Almacenamiento: Como Amazon S3, Google Cloud Storage o Azure Blob Storage.

• Procesamiento de Datos:

- Apache Kafka, Apache Spark o Apache Flink.
- Servicios de Transmisión de Datos en la Nube: Plataformas como Amazon Kinesis o Azure Event Hubs pueden capturar, procesar y cargar grandes volúmenes de datos en tiempo real desde dispositivos IoT o aplicaciones móviles.

• Almacenamiento y Procesamiento de Big Data:

- Apache Hadoop, Amazon EMR, Azure HDInsight.

b. Tecnologías para la visualización. ¿Qué tecnología permitiría mostrar un *dashboard* a tiempo real con la ubicación de cada taxi?

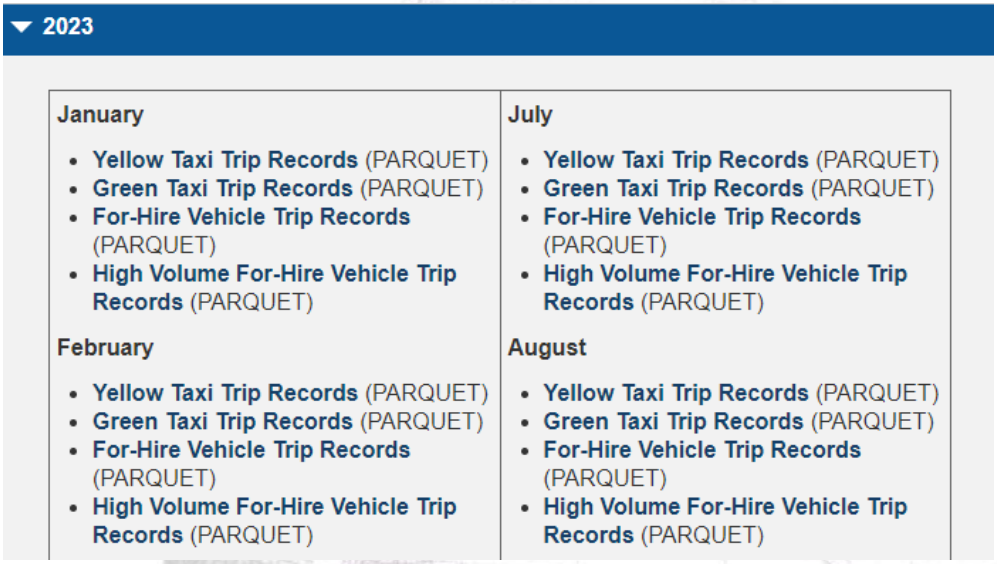
Podríamos usar tecnologías como Power BI, Qlik, Mapbox, Plotly o Leafet.

PRIMERA PARTE: ANÁLISIS CUANTITATIVO.

1.1 Primer examen preliminar del *dataset*. ¿En qué formato está el dataset y qué tiene que ver este formato con Big Data? ¿Qué parámetros hay en el *dataset*? ¿Cuál es su significado? ¿Existen valores aparentemente incorrectos?

• Formato del dataset:

El dataset está dividido por años y después por meses. Dentro de los meses hay 4 tipos: Yellow, Green, For-Hire, High Volume. El formato de cada dataset es .parquet es un fichero tipo zip que sirve para BigData para almacenar grandes volúmenes de datos en forma tabular. Pandas tiene un módulo para leer ficheros .parquet.



• Parámetros del dataset de Yellow por orden de aparición: En negrita los de mayor interés

- **VendorID: Identificador del proveedor del servicio de transporte. [tipo entero e identificador único]**
- **tpep_pickup_datetime: Fecha y hora en que se registró la recogida del pasajero. [formato yyy-mm-dd hh:mm:ss]**
- **tpep_dropoff_datetime: Fecha y hora en que se registró la entrega del pasajero. [formato yyy-mm-dd hh:mm:ss]**
- **passenger_count: Número de pasajeros en el viaje. [Entero (número de pasajeros)].**
- **trip_distance: Distancia recorrida en millas. [Float (millas)]**
- RatecodeID: Identifica el código de tarifa utilizado para calcular la tarifa del viaje. Puede haber diferentes códigos de tarifa para diferentes tipos de viajes o circunstancias especiales. [Entero (código de tarifa)].
- store_and_fwd_flag: Indica si los datos del viaje fueron almacenados temporalmente antes de ser enviados a través de la red.
- **PULocationID: Identificación de la ubicación de recogida. [Entero (código de identificación de la ubicación de recogida que se relaciona con el diccionario asociado)]**
- **DOLocationID: Identificación de la ubicación de entrega. [Entero (código de identificación de la ubicación de entrega que se relaciona con el diccionario asociado)]**
- payment_type: Método de pago utilizado (efectivo, tarjeta de crédito, etc.). [Float]
- **fare_amount: Tarifa del viaje. [Float (cantidad de tarifa)]**

- extra: Cargos adicionales que pueden aplicarse como tarifas nocturnas o tarifas de servicio premium. [Float (cantidad de tarifa extra)]
- mta_tax: Impuesto de la Metropolitan Autority Transport (MTA) aplicado en los viajes de Nueva York.
- tip_amount: Cantidad de propina. [Float (cantidad de propina)]
- tolls_amount: Tarifa de los peajes. [Float (cantidad de pago por peaje)]
- improvement_surcharge: Es un cargo adicional que se puede aplicar para cubrir mejoras en el servicio de transporte público
- **total_amount: Tarifa total cobrada al pasajero, incluyendo tarifas, peajes y propinas. [Float (cantidad total)]**
- congestion_surcharge: Cargo adicional que se aplica en circunstancias de áreas de congestión de tráfico. [Flotante (cantidad de cargo por congestión).]
- Airport_fee: Indica si se aplicó una tarifa adicional debido a que el viaje comenzó o terminó en un aeropuerto. [Flotante (cantidad de tarifa por viaje a aeropuerto).]

- Parámetros del dataset de Green por orden de aparición: En negrita los de mayor interés
  - **VendorID: Identificador del proveedor del servicio de transporte. [tipo entero e identificador único]**
  - **tpep_pickup_datetime: Fecha y hora en que se registró la recogida del pasajero. [formato yyy-mm-dd hh:mm:ss]**
  - **tpep_dropoff_datetime: Fecha y hora en que se registró la entrega del pasajero. [formato yyy-mm-dd hh:mm:ss]**
  - store_and_fwd_flag: Indica si los datos del viaje fueron almacenados temporalmente antes de ser enviados a través de la red.
  - RatecodeID: Identifica el código de tarifa utilizado para calcular la tarifa del viaje. Puede haber diferentes códigos de tarifa para diferentes tipos de viajes o circunstancias especiales. [Entero (código de tarifa)].
  - **PULocationID: Identificación de la ubicación de recogida. [Entero (código de identificación de la ubicación de recogida que se relaciona con el diccionario asociado)]**
  - **DOLocationID: Identificación de la ubicación de entrega. [Entero (código de identificación de la ubicación de entrega que se relaciona con el diccionario asociado)]**
  - **passenger_count: Número de pasajeros en el viaje. [Entero (número de pasajeros)].**
  - **trip_distance: Distancia recorrida en millas. [Float (millas)]**
  - **fare_amount: Tarifa del viaje. [Float (cantidad de tarifa)]**
  - extra: Cargos adicionales que pueden aplicarse como tarifas nocturnas o tarifas de servicio premium. [Float (cantidad de tarifa extra)]
  - mta_tax: Impuesto de la Metropolitan Autority Transport (MTA) aplicado en los viajes de Nueva York.
  - tip_amount: Cantidad de propina. [Float (cantidad de propina)]
  - tolls_amount: Tarifa de los peajes. [Float (cantidad de pago por peaje)]
  - ehail_fee: Tarifa de eHail. [Float]
  - improvement_surcharge: Cargo de mejora. [Float]
  - **total_amount: Tarifa total cobrada al pasajero, incluyendo tarifas, peajes y propinas. [Float (cantidad total)]**
  - payment_type: Método de pago utilizado (efectivo, tarjeta de crédito, etc.) [Float]
  - trip_type: Tipo de viaje. Solo ida (1) o ida y vuelta (2). [Entero]
  - congestion_surcharge: Cargo adicional que se aplica en circunstancias de áreas de congestión de tráfico. [Flotante (cantidad de cargo por congestión).]

1.2 Empezamos por visualizar el *dataset*. Haced un *plot* de las zonas de recogida y otro con las zonas de llegada del *dataset* de los cuatro ficheros y extrae conclusiones preliminares. ¿Se aprecian diferencias entre los puntos de los Yellow Cabs y los de los Green Cabs? ¿Se aprecian diferencias entre un mes y otro?

Las zonas de recogida de los parámetros PULocationID y DOLocationID que son valores enteros están relacionados con su posición según los siguientes diccionarios de la web.

- **Taxi Zone Lookup Table** (CSV)
- **Taxi Zone Shapefile** (PARQUET)

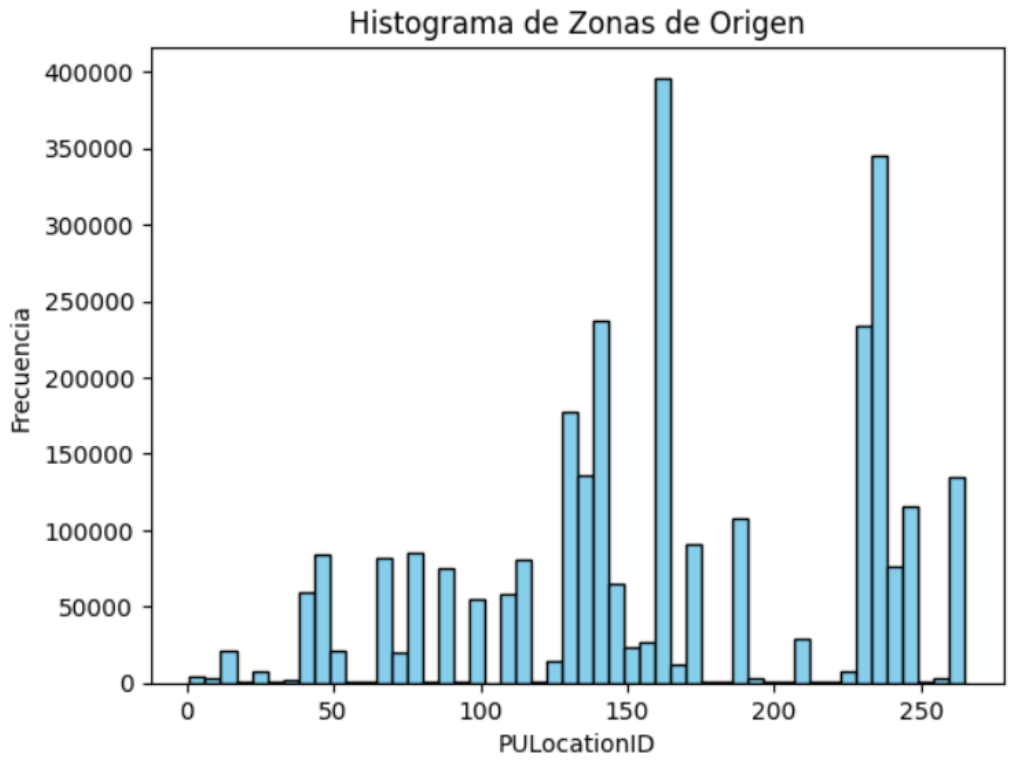
• Histogramas zonas de recogida (origen)

Carga de los datos y fusión de las zonas de recogida con código numérico con los diccionarios:

```
# Cargar el diccionario de zonas de recogida
taxi_zones = pd.read_csv(ruta_base + 'taxi_zone_lookup.csv')

# Fusionar los datos de los viajes con las zonas de recogida utilizando PULocationID
data_with_zones = yellow_tripdata_2023_07.merge(taxi_zones, left_on='PULocationID', right_on='LocationID', how='left')
```

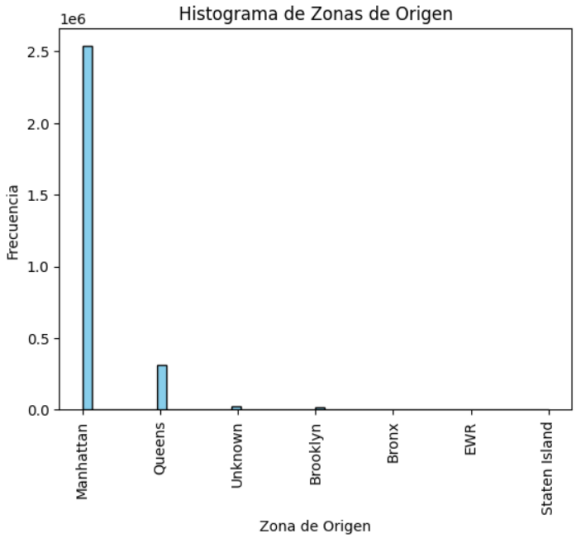
Histograma de la zona por su código numérico

```
plt.hist(data_with_zones['PULocationID'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histograma de Zonas de Origen')
plt.xlabel('PULocationID')
plt.ylabel('Frecuencia')
plt.show()
```



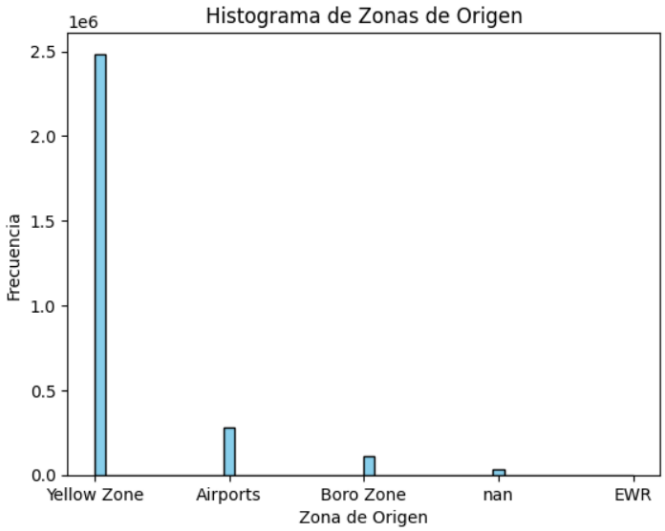
Histograma del diccionario Borough

```
# Eliminar filas con valores NaN en la columna 'Borough'
data_with_zones_clean = data_with_zones.dropna(subset=['Borough'])

# Crear el histograma
plt.hist(data_with_zones_clean['Borough'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histograma de Zonas de Origen')
plt.xlabel('Zona de Origen')
plt.ylabel('Frecuencia')
plt.xticks(rotation=90)
plt.show()
```



Histograma del diccionari service_zone

```
# Convertir los valores de la columna 'service_zone' a str
data_with_zones['service_zone'] = data_with_zones['service_zone'].astype(str)

# Crear el histograma
plt.hist(data_with_zones['service_zone'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histograma de Zonas de Origen')
plt.xlabel('Zona de Origen')
plt.ylabel('Frecuencia')
plt.show()
```



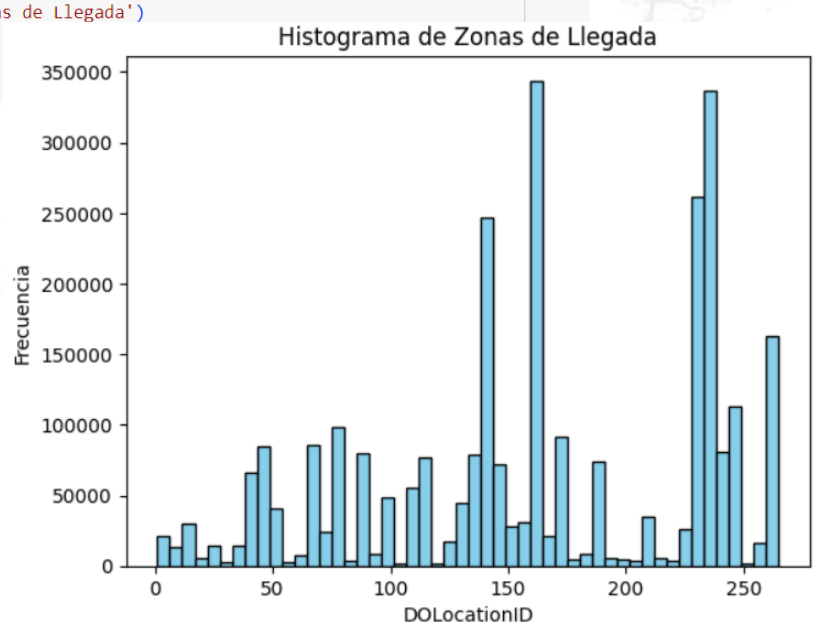
- ## Histogramas zonas de llegada (destino)

Carga de los datos y fusión de las zonas de llegada con código numérico con los diccionarios:

```
# Cargar el diccionario de zonas de llegada
taxi_zones = pd.read_csv(ruta_base + 'taxi_zone_lookup.csv')

# Fusionar los datos de los viajes con las zonas de llegada utilizando DOLocationID
data_with_zones = yellow_tripdata_2023_07.merge(taxi_zones, left_on='DOLocationID', right_on='LocationID', how='left')
```

Histograma de la zona por su código numérico

```
plt.hist(data_with_zones['DOLocationID'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histograma de Zonas de Llegada')
plt.xlabel('DOLocationID')
plt.ylabel('Frecuencia')
plt.show()
```

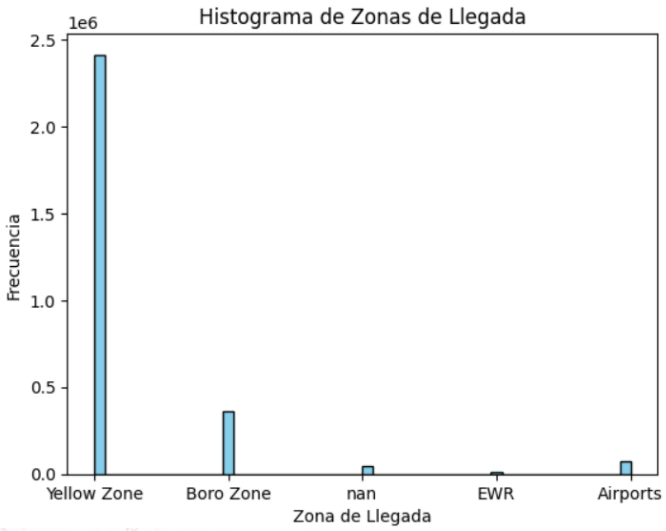


Histograma del diccionario Borough

```python
# Eliminar filas con valores NaN en la columna 'Borough'
data_with_zones_clean = data_with_zones.dropna(subset=['Borough'])

# Crear el histograma
plt.hist(data_with_zones_clean['Borough'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histograma de Zonas de Llegada')
plt.xlabel('Zona de Llegada')
plt.ylabel('Frecuencia')
plt.xticks(rotation=90)
plt.show()
```

Histograma del diccionari service_zone

```python
# Convertir los valores de la columna 'service_zone' a str
data_with_zones['service_zone'] = data_with_zones['service_zone'].astype(str)

# Crear el histograma
plt.hist(data_with_zones['service_zone'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histograma de Zonas de Llegada')
plt.xlabel('Zona de Llegada')
plt.ylabel('Frecuencia')
plt.show()
```



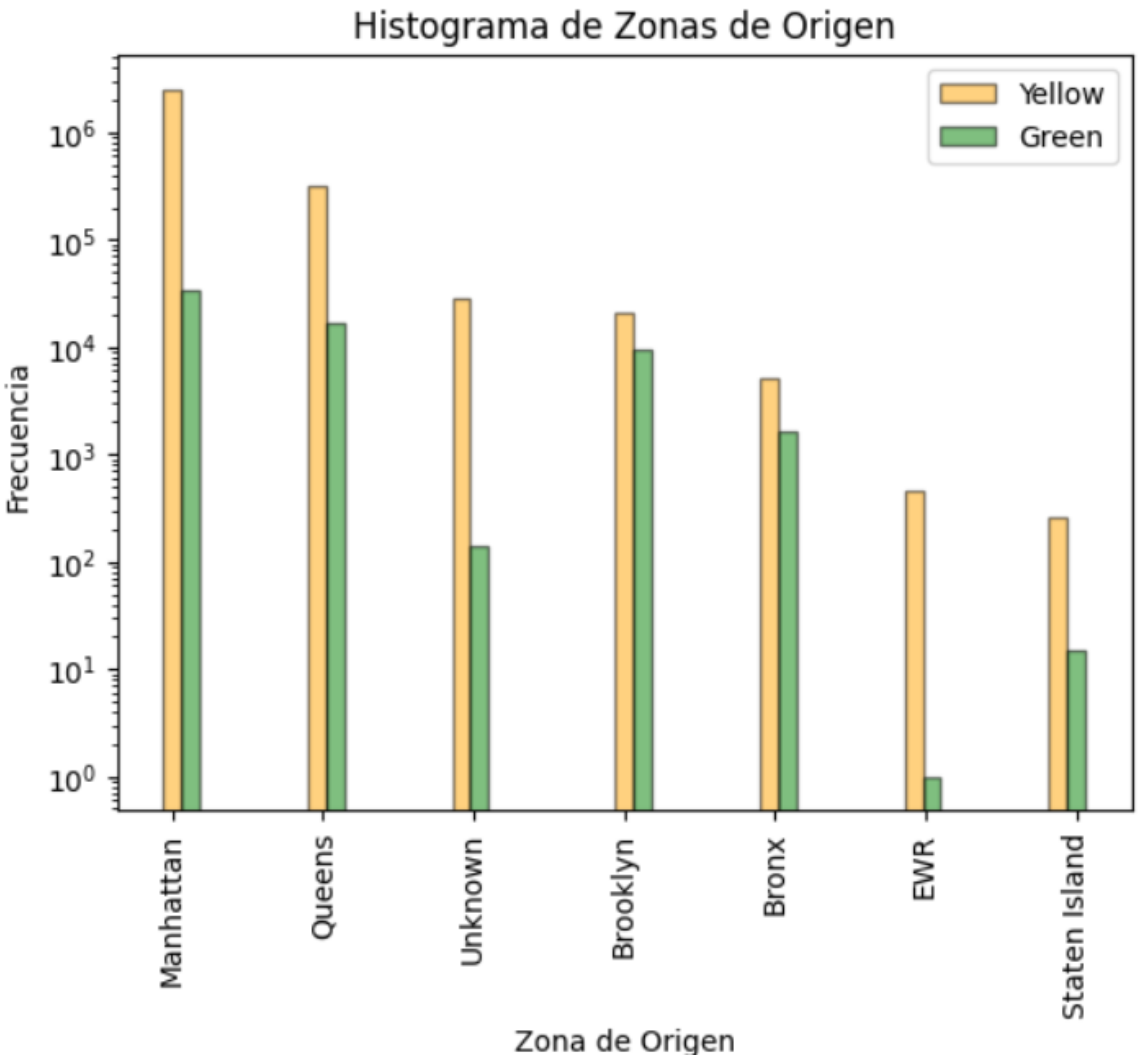
- **Diferencias entre Yellow y Green**

Representamos histogramas uno al lado del otro con escala logarítmica para comparar los viajes de los taxis Yellow que son mucho más numerosos que los viajes de los taxis Green.

```python
# Cargar el diccionario de zonas de recogida
taxi_zones = pd.read_csv(ruta_base + 'taxi_zone_lookup.csv')

# Fusionar los datos de los viajes con las zonas de recogida utilizando PULocationID
data_with_zones_y = yellow_tripdata_2023_07.merge(taxi_zones, left_on='PULocationID', right_on='LocationID', how='left')
data_with_zones_g = green_tripdata_2023_07.merge(taxi_zones, left_on='PULocationID', right_on='LocationID', how='left')
```

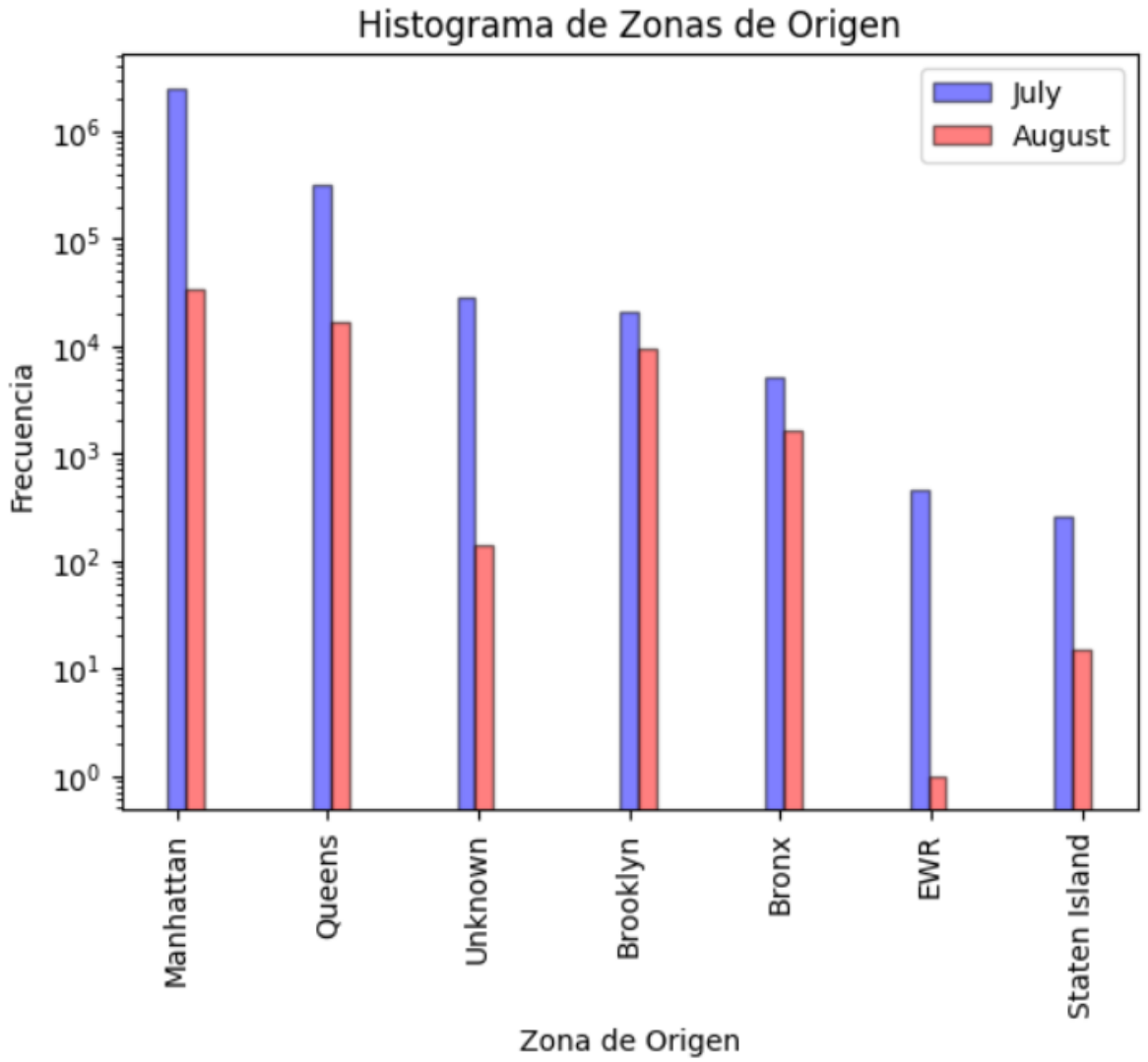
```python
# Eliminar filas con valores NaN en la columna 'Borough'
data_with_zones_clean_y = data_with_zones_y.dropna(subset=['Borough'])
data_with_zones_clean_g = data_with_zones_g.dropna(subset=['Borough'])
```

```python
# Definir las posiciones para los grupos de barras
positions_y = range(len(data_with_zones_clean_y['Borough'].unique()))
positions_g = [pos + 0.35 for pos in positions_y]

# Crear el histograma
plt.hist(data_with_zones_clean_y['Borough'], bins=50, color='orange', edgecolor='black',
         label='Yellow', alpha=0.5, histtype='bar', align='left')
plt.hist(data_with_zones_clean_g['Borough'], bins=50, color='green', edgecolor='black',
         label='Green', alpha=0.5, histtype='bar', align='right')
plt.yscale('log') # NEW!
plt.xticks(positions_y, data_with_zones_clean_y['Borough'].unique(), rotation=90)
plt.title('Histograma de Zonas de Origen')
plt.xlabel('Zona de Origen')
plt.ylabel('Frecuencia')
plt.legend()
plt.show()
```



- [Diferencias entre julio 2023 y agosto 2023]

Aplicamos el mismo código anterior para los meses de julio23 y agosto23 para los taxis yellow para apreciar diferencias.



1.3 Mejora la visualización con un *heat map*. Ayúdate de los *Zone Map…*

• Heatmap relación entre las zonas de recogida y de llegada.

Observamos que los valores son muy variados, por lo que realizamos un plot logarítmico para representar la frecuencia logarítmica de zonas de recogida y llegada

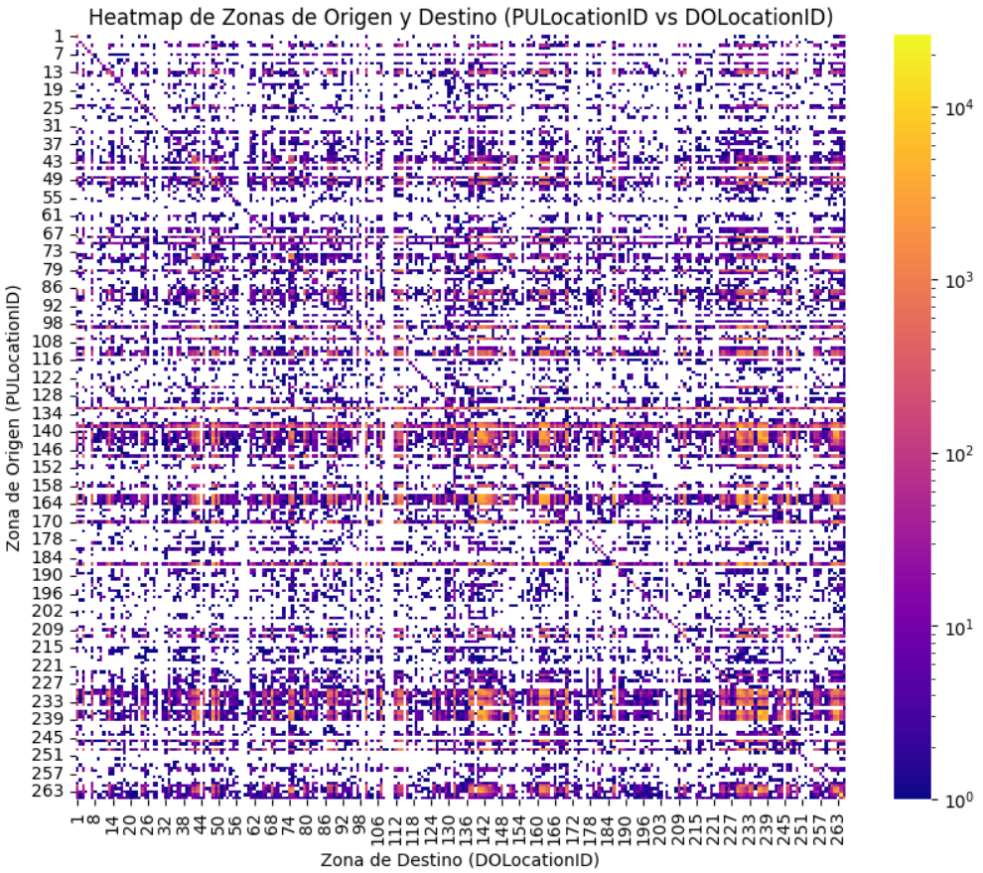
```
1 data_with_zones['PULocationID'].value_counts()
```

```
PULocationID
132    176228
161    136867
237    119947
186    107188
162    105676
        ...
111         3
59          2
187         2
245         1
30          1
Name: count, Length: 255, dtype: int64
```

```
1 data_with_zones['DOLocationID'].value_counts()
```

```
DOLocationID
161    115609
236    106858
237    105550
230     98011
170     89361
        ...
111        22
204        17
99          3
110         1
105         1
Name: count, Length: 261, dtype: int64
```

```python
from matplotlib.colors import LogNorm
zone_counts = data_with_zones.groupby(['PULocationID', 'DOLocationID']).size().unstack(fill_value=0)
plt.figure(figsize=(10, 8))
sns.heatmap(zone_counts, cmap='plasma', norm=LogNorm())
plt.title('Heatmap de Zonas de Origen y Destino (PULocationID vs DOLocationID)')
plt.xlabel('Zona de Destino (DOLocationID)')
plt.ylabel('Zona de Origen (PULocationID)')
plt.show()
```



SEGUNDA PARTE: ANÁLISIS CUALITATIVO.

2.1 ¿Cuál es el trayecto en el que la relación precio/km es más alta? ¿Cuál es el trayecto en el que la relación tiempo/km es más alta? ¿Cuál es el trayecto en el que la relación precio/tiempo es más alta?

Eliminamos los valores intrusos:

- Con 0 pasajeros.
- Con 0 distancia.
- Con tiempo de viaje superior a 1 día.

```python
# Eliminar los datos con passenger_count = 0.0
yellow_tripdata_2023_07 = yellow_tripdata_2023_07[yellow_tripdata_2023_07['passenger_count'] != 0.0]

# Eliminar los datos donde trip_distance es igual a 0
yellow_tripdata_2023_07 = yellow_tripdata_2023_07[yellow_tripdata_2023_07['trip_distance'] != 0]

# Calcular el tiempo de viaje (la diferencia entre tpep_pickup_datetime y tpep_dropoff_datetime)
yellow_tripdata_2023_07['tpep_pickup_datetime'] = pd.to_datetime(yellow_tripdata_2023_07['tpep_pickup_datetime'])
yellow_tripdata_2023_07['tpep_dropoff_datetime'] = pd.to_datetime(yellow_tripdata_2023_07['tpep_dropoff_datetime'])
time_difference = yellow_tripdata_2023_07['tpep_dropoff_datetime'] - yellow_tripdata_2023_07['tpep_pickup_datetime']

# Filtrar los datos donde la diferencia es menor o igual a un día
yellow_tripdata_2023_07 = yellow_tripdata_2023_07[time_difference <= pd.Timedelta(days=1)]
```

- **Trayecto con relación precio/km más alta.**

```python
# Calcular la división total_amount/trip_distance
yellow_tripdata_2023_07['total_amount_per_mile'] = yellow_tripdata_2023_07['total_amount'] / yellow_tripdata_2023_07['trip_distance']

# Encontrar el viaje con el valor total_amount/trip_distance más alto
viaje_max_total_amount_per_distance = yellow_tripdata_2023_07.loc[yellow_tripdata_2023_07['total_amount_per_mile'].idxmax()]

print("Viaje con la relación precio/km más alta")
print(viaje_max_total_amount_per_distance)
```

```
Viaje con la relación precio/km más alta
VendorID                                    2
tpep_pickup_datetime          2023-07-29 12:40:55
tpep_dropoff_datetime         2023-07-29 12:41:19
passenger_count                           1.0
trip_distance                            0.01
RatecodeID                                5.0
store_and_fwd_flag                          N
PULocationID                              246
DOLocationID                               50
payment_type                                1
fare_amount                             300.0
extra                                     0.0
mta_tax                                   0.0
tip_amount                                0.0
tolls_amount                              0.0
improvement_surcharge                     1.0
total_amount                            303.5
congestion_surcharge                      2.5
Airport_fee                               0.0
total_amount_per_distance             30350.0
time_per_distance              0 days 00:40:00
total_amount_per_mile                 30350.0
time_per_mile                  0 days 00:40:00
```

- **Trayecto con relación tiempo/km más alta.**

```python
# Calcular la relación entre el tiempo de trayecto y trip_distance
yellow_tripdata_2023_07['time_per_mile'] = time_difference / yellow_tripdata_2023_07['trip_distance']

# Encontrar el viaje con el valor tiempo de trayecto/trip_distance más alto
viaje_max_time_per_distance = yellow_tripdata_2023_07.loc[yellow_tripdata_2023_07['time_per_mile'].idxmax()]

print("\nViaje con la relación tiempo/km más alta")
print(viaje_max_time_per_distance)
```

```
Viaje con la relación tiempo/km más alta
VendorID                                    2
tpep_pickup_datetime          2023-07-20 08:11:39
tpep_dropoff_datetime         2023-07-20 12:20:45
passenger_count                             1.0
trip_distance                              0.01
RatecodeID                                  2.0
store_and_fwd_flag                            N
PULocationID                                 68
DOLocationID                                164
payment_type                                  2
fare_amount                                70.0
extra                                       0.0
mta_tax                                     0.5
tip_amount                                  0.0
tolls_amount                                0.0
improvement_surcharge                       1.0
total_amount                               74.0
congestion_surcharge                        2.5
Airport_fee                                 0.0
total_amount_per_distance                7400.0
time_per_distance               17 days 07:10:00
total_amount_per_mile                    7400.0
time_per_mile                   17 days 07:10:00
```

- **Trayecto con relación precio/tiempo más alta.**

```python
# Convertir 'time_difference' de microsegundos a segundos
yellow_tripdata_2023_07['time_difference_seconds'] = yellow_tripdata_2023_07['time_difference'].dt.total_seconds()

# Calcular el precio por hora (precio/tiempo)
yellow_tripdata_2023_07['price_per_hour'] = yellow_tripdata_2023_07['total_amount'] /
(yellow_tripdata_2023_07['time_difference_seconds'] / 3600)

# Encontrar el trayecto con la relación precio/tiempo más alta
max_price_per_time_trip = yellow_tripdata_2023_07.loc[yellow_tripdata_2023_07['price_per_hour'].idxmax()]

print("\nTrayecto con la relación precio/tiempo más alta:")
print(max_price_per_time_trip)
```

```
Trayecto con la relación precio/tiempo más alta:
VendorID                                    1
tpep_pickup_datetime       2023-07-02 09:30:19
tpep_dropoff_datetime      2023-07-02 09:30:19
passenger_count                            2.0
trip_distance                              1.1
RatecodeID                                 1.0
store_and_fwd_flag                           N
PULocationID                               186
DOLocationID                               186
payment_type                                 2
fare_amount                                6.5
extra                                      2.5
mta_tax                                    0.5
tip_amount                                 0.0
tolls_amount                               0.0
improvement_surcharge                      1.0
total_amount                              10.5
congestion_surcharge                       2.5
Airport_fee                                0.0
total_amount_per_mile                 9.545455
time_per_mile                0 days 00:00:00
time_difference              0 days 00:00:00
time_difference_seconds                    0.0
price_per_hour                             inf
Name: 93126, dtype: object
```

Este trayecto podría ser erróneo ya que el pasajero no se subió al taxi (pickup time = dropoff time). Podríamos pensar que se trata de un viaje que se pidió y no se usó. Si queremos quitar estos tipos de viajes podemos eliminar los viajes que tienen una duración menor a 10 segundos para no considerarlos en el análisi. Entonces encontramos un viaje de 12 segundos en el que se hicieron 70 metros.

```python
# Eliminamos los trayectos que tienen una time_difference_seconds menor a 10 segundos
yellow_tripdata_2023_07 = yellow_tripdata_2023_07[yellow_tripdata_2023_07['time_difference_seconds'] >= 10]

# Convertir 'time_difference' de microsegundos a segundos
yellow_tripdata_2023_07['time_difference_seconds'] = yellow_tripdata_2023_07['time_difference'].dt.total_seconds()

# Calcular el precio por hora (precio/tiempo)
yellow_tripdata_2023_07['price_per_hour'] = yellow_tripdata_2023_07['total_amount'] /
(yellow_tripdata_2023_07['time_difference_seconds'] / 3600)

# Encontrar el trayecto con la relación precio/tiempo más alta
max_price_per_time_trip = yellow_tripdata_2023_07.loc[yellow_tripdata_2023_07['price_per_hour'].idxmax()]

print("\nTrayecto con la relación precio/tiempo más alta:")
print(max_price_per_time_trip)
```

```
Trayecto con la relación precio/tiempo más alta:
VendorID                                    2
tpep_pickup_datetime       2023-07-16 16:36:25
tpep_dropoff_datetime      2023-07-16 16:36:37
passenger_count                            1.0
trip_distance                             0.07
RatecodeID                                 5.0
store_and_fwd_flag                           N
PULocationID                               138
DOLocationID                               226
payment_type                                 2
fare_amount                              700.0
extra                                      5.0
mta_tax                                    0.0
tip_amount                                 0.0
tolls_amount                               0.0
improvement_surcharge                      1.0
total_amount                            707.75
congestion_surcharge                       0.0
Airport_fee                               1.75
total_amount_per_mile                10110.714286
time_per_mile              0 days 00:02:51.428571
time_difference                  0 days 00:00:12
time_difference_seconds                     12.0
price_per_hour                          212325.0
Name: 1341491, dtype: object
```

2.2 ¿Cuál es el trayecto en el que la relación precio/km es más baja? ¿Cuál es el trayecto en el que la relación tiempo/km es más baja? ¿Cuál es el trayecto en el que la relación precio/tiempo es más baja?

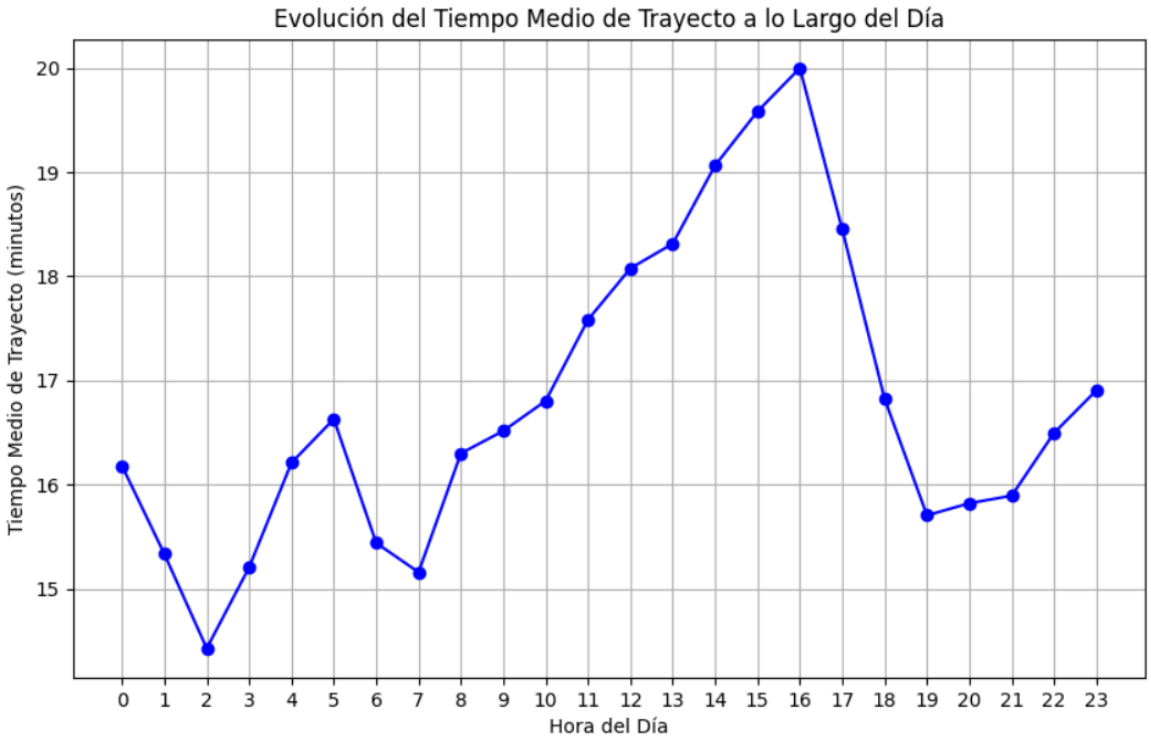
Igual que el 2.1 pero con el valor mínimo. En vez de idxmax() -> idxmin(). Aplicamos el valor absoluto para no encontrar viajes con precios negativos. Descartamos los valores donde el precio es 0 (viaje gratis o precio no anotado).

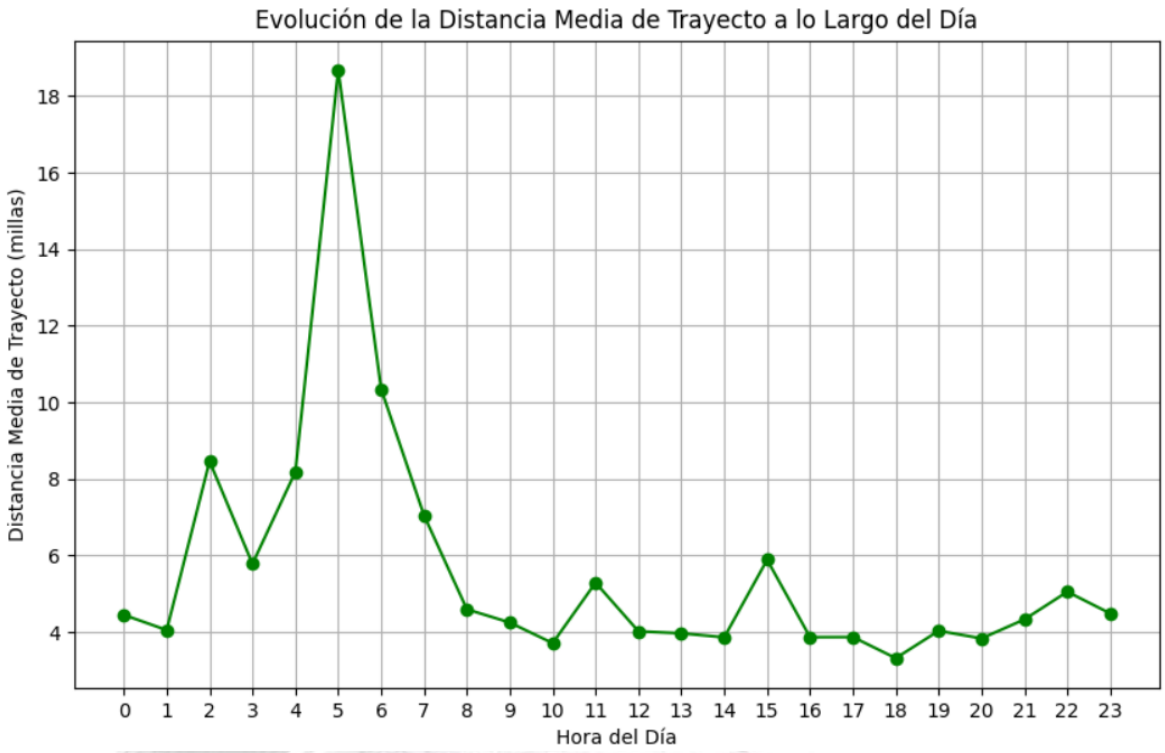
```python
# Filtrar los viajes donde fare_amount no es igual a 0
yellow_tripdata_2023_07 = yellow_tripdata_2023_07[yellow_tripdata_2023_07['fare_amount'] != 0]

# Calcular la división total_amount/trip_distance
yellow_tripdata_2023_07['total_amount_per_mile'] = yellow_tripdata_2023_07['total_amount'] / yellow_tripdata_2023_07['trip_distance']

# Encontrar el viaje con el valor total_amount/trip_distance más pequeño en valor absoluto
min_absolute_total_amount_per_distance_index = yellow_tripdata_2023_07['total_amount_per_mile'].abs().idxmin()
viaje_min_total_amount_per_distance = yellow_tripdata_2023_07.loc[min_absolute_total_amount_per_distance_index]

print("Viaje con la relación precio/km más baja (en valor absoluto) después de descartar los viajes con fare_amount = 0")
print(viaje_min_total_amount_per_distance)
```

```
Viaje con la relación precio/km más baja (en valor absoluto) después de descartar los viajes con fare_amount = 0
VendorID                                      2
tpep_pickup_datetime          2023-07-09 02:37:00
tpep_dropoff_datetime         2023-07-09 02:48:00
passenger_count                               NaN
trip_distance                          191944.96
RatecodeID                                    NaN
store_and_fwd_flag                           None
PULocationID                                  158
DOLocationID                                  137
payment_type                                    0
fare_amount                                 14.74
extra                                         0.0
mta_tax                                       0.5
tip_amount                                   3.75
tolls_amount                                  0.0
improvement_surcharge                         1.0
total_amount                                22.49
congestion_surcharge                          NaN
Airport_fee                                   NaN
total_amount_per_mile                    0.000117
time_per_mile              0 days 00:00:00.003438
time_difference                  0 days 00:11:00
time_difference_seconds                    660.0
price_per_hour                        122.672727
Name: 2837215, dtype: object
```

2.3 Muestra la evolución del tiempo medio de trayecto a lo largo del día. Muestra la evolución de la distancia media de trayecto a lo largo del día.

```python
# Convertir las columnas de fecha y hora a tipo datetime
yellow_tripdata_2023_07['tpep_pickup_datetime'] = pd.to_datetime(yellow_tripdata_2023_07['tpep_pickup_datetime'])
yellow_tripdata_2023_07['tpep_dropoff_datetime'] = pd.to_datetime(yellow_tripdata_2023_07['tpep_dropoff_datetime'])

# Extraer la hora del día de la columna de hora de recogida
yellow_tripdata_2023_07['hour'] = yellow_tripdata_2023_07['tpep_pickup_datetime'].dt.hour

# Calcular la duración de cada viaje en minutos
yellow_tripdata_2023_07['trip_duration_minutes'] = (yellow_tripdata_2023_07['tpep_dropoff_datetime']
                                                    - yellow_tripdata_2023_07['tpep_pickup_datetime']).dt.total_seconds() / 60

# Agrupar los datos por hora del día y calcular el tiempo medio de trayecto y la distancia media de trayecto
avg_trip_duration_by_hour = yellow_tripdata_2023_07.groupby('hour')['trip_duration_minutes'].mean()
avg_trip_distance_by_hour = yellow_tripdata_2023_07.groupby('hour')['trip_distance'].mean()

# Graficar la evolución del tiempo medio de trayecto a lo largo del día
plt.figure(figsize=(10, 6))
plt.plot(avg_trip_duration_by_hour.index, avg_trip_duration_by_hour.values, marker='o', color='blue')
plt.title('Evolución del Tiempo Medio de Trayecto a lo Largo del Día')
plt.xlabel('Hora del Día')
plt.ylabel('Tiempo Medio de Trayecto (minutos)')
plt.xticks(range(24))
plt.grid(True)
plt.show()

# Graficar la evolución de la distancia media de trayecto a lo largo del día
plt.figure(figsize=(10, 6))
plt.plot(avg_trip_distance_by_hour.index, avg_trip_distance_by_hour.values, marker='o', color='green')
plt.title('Evolución de la Distancia Media de Trayecto a lo Largo del Día')
plt.xlabel('Hora del Día')
plt.ylabel('Distancia Media de Trayecto (millas)')
plt.xticks(range(24))
plt.grid(True)
plt.show()
```





2.4 Elige dos zonas cualquiera de la ciudad y calcula la probabilidad de desplazarse de una zona a otra en menos de X minutos. (El valor X así como las zonas deben ser fácilmente modificables).

```
# Selección de zonas de la ciudad y el valor de X minutos
zona_origen = 50
zona_destino = 107
X = 30
```

```
# Filtrar los datos para incluir solo los viajes que se originan en la zona_origen y terminan en la zona_destino
viajes = yellow_tripdata_2023_07[(yellow_tripdata_2023_07['PULocationID'] == zona_origen) & (yellow_tripdata_2023_07['DOLocationID'] == zona_destino)]

# Calcular la proporción de viajes que duran menos de X minutos
probabilidad = (viajes['trip_duration_minutes'] < X).mean()

print(f"La probabilidad de desplazarse de {zona_origen} a {zona_destino} en menos de {X} minutos es: {probabilidad:.2%}")
```

```
La probabilidad de desplazarse de 50 a 107 en menos de 30 minutos es: 94.67%
```

2.5 Repite los apartados 1.1, 1.2, 1.3, 2.1, 2.2, 2.3 y 2.4 con un dataset del mismo mes pero del año 2009. Comenta todas las diferencias que vayas encontrando.

Las diferencias

TERCERA PARTE: ANÁLISIS PREDICTIVO.

3.1 ¿Cuáles son las zonas donde es más probable coger un taxi en función de la hora del día?

```python
# Cargar el diccionario de zonas de recogida
taxi_zones = pd.read_csv(ruta_base + 'taxi_zone_lookup.csv')

# Fusionar los datos de los viajes con las zonas de recogida utilizando PULocationID
data_with_zones = yellow_tripdata_2023_07.merge(taxi_zones, left_on='PULocationID', right_on='LocationID', how='left')

# Eliminar los NaN
data_with_zones_clean = data_with_zones.dropna(subset=['Borough'])

# Convertir la columna 'tpep_pickup_datetime' a tipo datetime y extraer la hora
data_with_zones['tpep_pickup_datetime'] = pd.to_datetime(data_with_zones['tpep_pickup_datetime'])
data_with_zones['hour'] = data_with_zones['tpep_pickup_datetime'].dt.hour

# Agrupar los datos por hora y calcular el máximo de veces que aparece una zona determinada en 'Borough'
max_zone_byhour = data_with_zones.groupby(['hour'])['Borough'].apply(lambda x: x.value_counts().idxmax())
print("Máximo de veces que aparece una zona determinada en 'Borough' por hora:", max_zone_byhour)
```

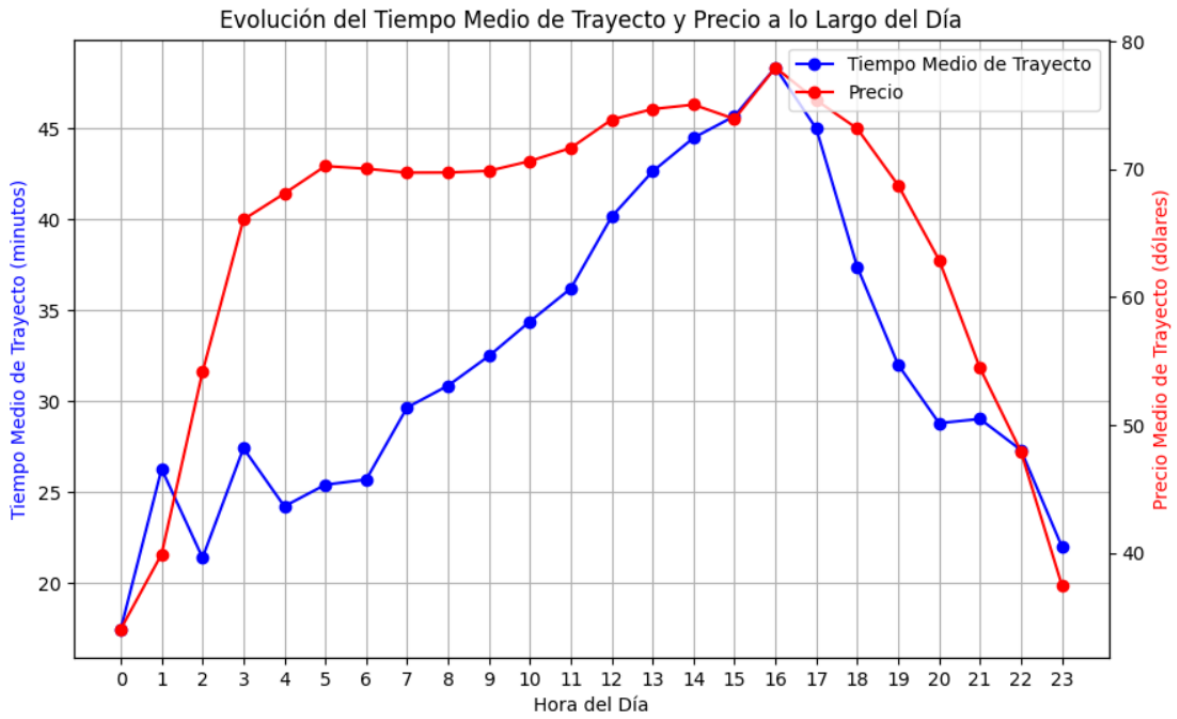
```
Máximo de veces que aparece una zona determinada en 'Borough' por hora: hour
0     Manhattan
1     Manhattan
2     Manhattan
3     Manhattan
4     Manhattan
5     Manhattan
6     Manhattan
7     Manhattan
8     Manhattan
9     Manhattan
10    Manhattan
11    Manhattan
12    Manhattan
13    Manhattan
14    Manhattan
15    Manhattan
16    Manhattan
17    Manhattan
18    Manhattan
19    Manhattan
20    Manhattan
21    Manhattan
22    Manhattan
23    Manhattan
Name: Borough, dtype: object
```

En Manhattan es donde más taxis se suben pasajeros para cualquier hora. El resultado no es de extrañar observando el gráfico logarítmico de las zonas de origen apartado 1.2 donde se ve que a la zona de Manhattan es donde más taxis se cogen.

3.2 ¿Cuál es la mejor hora del día para ir al aeropuerto?

```python
# Cargar el diccionario de zonas de llegada
taxi_zones = pd.read_csv(ruta_base + 'taxi_zone_lookup.csv')

# Fusionar los datos de los viajes con las zonas de llegada utilizando DOLocationID
data_with_zones = yellow_tripdata_2023_07.merge(taxi_zones, left_on='DOLocationID', right_on='LocationID', how='left')
data_with_zones['service_zone'] = data_with_zones['service_zone'].astype(str)

# Filtrar las zonas que son 'airport' como destinación
airport_end = data_with_zones[data_with_zones['service_zone'] == 'Airports']
```

```python
# Convertir las columnas de fecha y hora a tipo datetime y Extraer la hora del día de la columna de hora de recogida
airport_end['tpep_pickup_datetime'] = pd.to_datetime(airport_end['tpep_pickup_datetime'])
airport_end['tpep_dropoff_datetime'] = pd.to_datetime(airport_end['tpep_dropoff_datetime'])
airport_end['hour'] = airport_end['tpep_pickup_datetime'].dt.hour

# Calcular la duración de cada viaje en minutos
airport_end['trip_duration_minutes'] = (airport_end['tpep_dropoff_datetime'] - airport_end['tpep_pickup_datetime']).dt.total_seconds() / 60

# Agrupar los datos por hora del día. Calcular el tiempo y el precio medio de trayecto
avg_trip_duration_by_hour = airport_end.groupby('hour')['trip_duration_minutes'].mean()
avg_price_by_hour = airport_end.groupby('hour')['total_amount'].mean()

fig, ax1 = plt.subplots(figsize=(10, 6))
# Graficar el tiempo
ax1.plot(avg_trip_duration_by_hour.index, avg_trip_duration_by_hour.values, marker='o', color='blue', label='Tiempo Medio de Trayecto')
ax1.set_xlabel('Hora del Día')
ax1.set_ylabel('Tiempo Medio de Trayecto (minutos)', color='blue')
ax1.set_xticks(range(24))
ax1.grid(True)

# Graficar el precio
ax2 = ax1.twinx()
ax2.plot(avg_price_by_hour.index, avg_price_by_hour.values, marker='o', color='red', label='Precio')
ax2.set_ylabel('Precio Medio de Trayecto (dólares)', color='red')

# Mostrar leyendas
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc='upper right')
plt.title('Evolución del Tiempo Medio de Trayecto y Precio a lo Largo del Día')
plt.show()
```



Se observa que por la noche (entre 23h-2h) es cuando el precio y el tiempo de viaje al aeropuerto es menor. Téngase en cuenta que el siguiente gráfico toma todos los viajes con destino al aeropuerto sin discernir el origen del trayecto.

3.3 Diseña un modelo que, dada una hora, una zona origen, y una zona destino, predice la duración del trayecto y su coste. Muestra la relevancia de los atributos del *dataset.*

• Modelo de predicción a partir de la media de los datos

```
# Datos usados para la predicción
datos_yellow_julio_y_agosto = pd.concat([yellow_tripdata_2023_07, yellow_tripdata_2023_08], ignore_index=True)
```

```
# Modelo predictor
def predecir_duracion_coste(data, hour, origen, destino):
    # Filtrar los datos por hora, zona de origen y zona de destino
    viajes_filtrados = data[(data['hour'] == hour) & (data['PULocationID'] == origen) & (data['DOLocationID'] == destino)]

    # Calcular la media de la duración del trayecto y su coste
    duracion_media = viajes_filtrados['trip_duration_minutes'].mean()
    coste_medio = viajes_filtrados['total_amount'].mean()

    return duracion_media, coste_medio
```

```
hora = 11  # Hora del día (0-23)
zona_origen = 50  # ID de la zona de origen
zona_destino = 107  # ID de la zona de destino

duracion_predict, coste_predict = predecir_duracion_coste(datos_yellow_julio_y_agosto, hora, zona_origen, zona_destino)
print(f"Predicción de duración del trayecto: {round(duracion_predict)} minutos")
print(f"Predicción de coste del trayecto: ${round(coste_predict,2)}")
```

```
Predicción de duración del trayecto: 27 minutos
Predicción de coste del trayecto: $25.27
```

• Modelo de predicción con randomforest

```
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

# Eliminar filas con valores NaN en los datos de entrada
datos_yellow_julio_y_agosto_sin_nan = datos_yellow_julio_y_agosto.dropna()

# Separar los datos en características (X) y la variable objetivo (y)
X = datos_yellow_julio_y_agosto_sin_nan[['hour', 'PULocationID', 'DOLocationID']]
y_duracion = datos_yellow_julio_y_agosto_sin_nan['trip_duration_minutes']
y_coste = datos_yellow_julio_y_agosto_sin_nan['total_amount']

# Inicializar el modelo de bosque aleatorio
rf_duracion = RandomForestRegressor()
rf_coste = RandomForestRegressor()

# Entrenar el modelo
rf_duracion.fit(X, y_duracion)
rf_coste.fit(X, y_coste)
```

```python
# Obtener la importancia de las características
importancia_caracteristicas_duracion = rf_duracion.feature_importances_
importancia_caracteristicas_coste = rf_coste.feature_importances_

# Visualizar la importancia de las características
plt.figure(figsize=(10, 6))
plt.bar(X.columns, importancia_caracteristicas_duracion, color='blue', alpha=0.7)
plt.xlabel('Características')
plt.ylabel('Importancia')
plt.title('Importancia de las Características para la Duración del Trayecto')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(10, 6))
plt.bar(X.columns, importancia_caracteristicas_coste, color='green', alpha=0.7)
plt.xlabel('Características')
plt.ylabel('Importancia')
plt.title('Importancia de las Características para el Coste del Trayecto')
plt.xticks(rotation=45)
plt.show()
```