BIG DATA

Elías Ramos Calderón

New Technology School

Tokio.

New
Technology
School

Elías Ramos Calderón

RESPONSABLE DEL DESARROLLO DE LA APLICACIÓN

Fecha: 21, Julio, 2024

BIG DATA
PROYECTO FINAL
Elías Ramos Calderón

Tokio.

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[2] !pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 2.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=1aba4f5f5e38c79c6b58c00f4fa197bfe8d2bebeb21c1c13fc96d1801b27ec62
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

```
[3] from pyspark.sql import SparkSession
    from pyspark.sql.functions import avg
    from pyspark.sql.functions import col
    import matplotlib.pyplot as plt
    import seaborn as sns

    spark = SparkSession.builder.appName("Proyecto_Final").getOrCreate()
```

```
data_path = '/content/drive/MyDrive/Proyecto Final/Proyecto BigData/'

data = spark.read.options(inferSchema='True',delimiter=',', header=True).csv(data_path + 'Air_Traffic_Passenger_Statistics.csv')
data.take(2)
```

```
[Row(Activity Period=200507, Operating Airline='ATA Airlines', Operating Airline IATA Code='TZ', Published Airline='ATA Airlines', Published Airline IATA Code='TZ', GEO Summary='Domestic', GEO Region='US',
Activity Type Code='Deplaned', Price Category Code='Low Fare', Terminal='Terminal 1', Boarding Area='B', Passenger Count=27271, Adjusted Activity Type Code='Deplaned', Adjusted Passenger Count=27271,
Year=2005, Month='July'),
 Row(Activity Period=200507, Operating Airline='ATA Airlines', Operating Airline IATA Code='TZ', Published Airline='ATA Airlines', Published Airline IATA Code='TZ', GEO Summary='Domestic', GEO Region='US',
Activity Type Code='Enplaned', Price Category Code='Low Fare', Terminal='Terminal 1', Boarding Area='B', Passenger Count=29131, Adjusted Activity Type Code='Enplaned', Adjusted Passenger Count=29131,
Year=2005, Month='July')]
```

```
[5] data.show(5)
```

```
+---------------+----------------+--------------------------+----------------+--------------------------+-------------+----------+-----------------+--------------------+---------+-------------+--------
|Activity Period|Operating Airline|Operating Airline IATA Code|Published Airline|Published Airline IATA Code|  GEO Summary|GEO Region|Activity Type Code|Price Category Code|  Terminal|Boarding Area|Passenge
+---------------+----------------+--------------------------+----------------+--------------------------+-------------+----------+-----------------+--------------------+---------+-------------+--------
|         200507|     ATA Airlines|                        TZ|     ATA Airlines|                        TZ|     Domestic|        US|          Deplaned|           Low Fare|Terminal 1|            B|
|         200507|     ATA Airlines|                        TZ|     ATA Airlines|                        TZ|     Domestic|        US|          Enplaned|           Low Fare|Terminal 1|            B|
|         200507|     ATA Airlines|                        TZ|     ATA Airlines|                        TZ|     Domestic|        US|    Thru / Transit|           Low Fare|Terminal 1|            B|
|         200507|      Air Canada |                        AC|      Air Canada |                        AC|International|    Canada|          Deplaned|               Other|Terminal 1|            B|
|         200507|      Air Canada |                        AC|      Air Canada |                        AC|International|    Canada|          Enplaned|               Other|Terminal 1|            B|
+---------------+----------------+--------------------------+----------------+--------------------------+-------------+----------+-----------------+--------------------+---------+-------------+--------
only showing top 5 rows
```

```
# 1. ¿Cuántas compañías diferentes aparecen en el fichero?
unique_companies_count = data.select("Operating Airline").distinct().count()
print("Número de compañías diferentes:", unique_companies_count)

# Mostrar todas las compañías únicas
unique_companies = data.select("Operating Airline").distinct()
unique_companies.show(unique_companies.count(), truncate=False)
```

```python
# 2. ¿Cuántos pasajeros tienen de media los vuelos de cada compañía?
average_passengers_per_airline = data.groupBy("Operating Airline").agg(avg("Passenger Count").alias("Average Passenger Count"))
#average_passengers_per_airline.show(average_passengers_per_airline.count(), truncate=False)

# Convertir el DataFrame de PySpark a un DataFrame de Pandas
average_passengers_per_airline_pd = average_passengers_per_airline.toPandas()

# Configurar el estilo de los gráficos
sns.set(style="whitegrid")

# Crear el gráfico de barras
plt.figure(figsize=(14, 14))
ax = sns.barplot(x="Average Passenger Count", y="Operating Airline", data=average_passengers_per_airline_pd)

# Configurar el título y las etiquetas de los ejes
plt.title('Media de Pasajeros por Aerolínea')
plt.xlabel('Media de Pasajeros')
plt.ylabel('Aerolínea')

# Mostrar el gráfico
plt.tight_layout()
plt.show()
```

```python
# 3. Eliminar registros duplicados por el campo “GEO Región”, manteniendo sólo aquel con mayor número de pasajeros
df_sorted = data.orderBy(col("GEO Region"), col("Passenger Count").desc())
df_deduplicated = df_sorted.dropDuplicates(["GEO Region"])

# Mostrar el DataFrame deduplicado
df_deduplicated.show(df_deduplicated.count(), truncate=False)
```

```python
# 4. Volcar los resultados a un archivo CSV
output_path_avg = "/content/drive/MyDrive/Proyecto Final/Proyecto BigData/average_passengers_per_airline.csv"
output_path_deduplicated = "/content/drive/MyDrive/Proyecto Final/Proyecto BigData/deduplicated_geo_region.csv"

# Escribir los resultados en archivos CSV
average_passengers_per_airline.write.csv(output_path_avg, header=True, mode="overwrite")
df_deduplicated.write.csv(output_path_deduplicated, header=True, mode="overwrite")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install pyspark
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import mean, stddev, col, count
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
import matplotlib.ticker as ticker

# Iniciar una sesión de Spark
spark = SparkSession.builder.appName("AirTrafficAnalysis").getOrCreate()

# Cargar el conjunto de datos
data_path = '/content/drive/MyDrive/Proyecto Final/Proyecto BigData/'

data = spark.read.options(inferSchema='True',delimiter=',', header=True).csv(data_path + 'Air_Traffic_Passenger_Statistics.csv')
data.take(2)

# Mostrar el esquema del dataframe
data.printSchema()

# Mostrar los primeros registros del dataframe
data.show(5)
```

```
# Modificar los tipos de datos de las columnas si es necesario
df = data.withColumn("Activity Period", col("Activity Period").cast("integer"))
df = data.withColumn("Year", col("Year").cast("integer"))
df = data.withColumn("Passenger Count", col("Passenger Count").cast("integer"))
df = data.withColumn("Adjusted Passenger Count", col("Adjusted Passenger Count").cast("integer"))
```

```python
# Relacionar Passenger Count con Operating Airline
passenger_airline_stats = df.groupBy("Operating Airline").agg(
    mean("Passenger Count").alias("mean_passenger_count"),
    stddev("Passenger Count").alias("stddev_passenger_count"),
    count("Passenger Count").alias("count_passenger_count")
).toPandas()

# Redondear las estadísticas sin decimales
passenger_airline_stats['mean_passenger_count'] = passenger_airline_stats['mean_passenger_count'].round(0).astype(int)
passenger_airline_stats['stddev_passenger_count'] = passenger_airline_stats['stddev_passenger_count'].fillna(0).round(0).astype(int)
passenger_airline_stats['count_passenger_count'] = passenger_airline_stats['count_passenger_count'].round(0).astype(int)

# Ordenar las estadísticas alfabéticamente por Operating Airline
passenger_airline_stats = passenger_airline_stats.sort_values('Operating Airline')

# Crear una figura para la tabla
fig, ax = plt.subplots(figsize=(10, 6))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=passenger_airline_stats.values, colLabels=passenger_airline_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()
```

```python
# Configurar el estilo de los gráficos
sns.set(style="whitegrid")

# Gráfico de Barras para Passenger Count con Operating Airline (mean)
plt.figure(figsize=(16, 6))
ax = sns.barplot(data=passenger_airline_stats, x='Operating Airline', y='mean_passenger_count')
plt.title('Mean Passenger Count by Operating Airline')
plt.xlabel('Operating Airline')
plt.ylabel('Mean Passenger Count')
plt.xticks(rotation=90)
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
plt.show()

# Gráfico de Barras para Passenger Count con Operating Airline (stddev)
plt.figure(figsize=(16, 6))
ax = sns.barplot(data=passenger_airline_stats, x='Operating Airline', y='stddev_passenger_count')
plt.title('Stddev Passenger Count by Operating Airline')
plt.xlabel('Operating Airline')
plt.ylabel('Stddev Passenger Count')
plt.xticks(rotation=90)
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
plt.show()
```

```python
# Relacionar Passenger Count con GEO Region
passenger_geo_region_stats = df.groupBy("GEO Region").agg(
    mean("Passenger Count").alias("mean_passenger_count"),
    stddev("Passenger Count").alias("stddev_passenger_count"),
    count("Passenger Count").alias("count_passenger_count")
).toPandas()

# Redondear las estadísticas sin decimales
passenger_geo_region_stats['mean_passenger_count'] = passenger_geo_region_stats['mean_passenger_count'].round(0).astype(int)
passenger_geo_region_stats['stddev_passenger_count'] = passenger_geo_region_stats['stddev_passenger_count'].fillna(0).round(0).astype(int)
passenger_geo_region_stats['count_passenger_count'] = passenger_geo_region_stats['count_passenger_count'].round(0).astype(int)

# Ordenar las estadísticas alfabéticamente por GEO Region
passenger_geo_region_stats = passenger_geo_region_stats.sort_values('GEO Region')

# Configurar el estilo de los gráficos
sns.set(style="whitegrid")
```

```python
# Visualizar la tabla y el gráfico lado a lado (mean)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))
sns.barplot(data=passenger_geo_region_stats, x='GEO Region', y='mean_passenger_count', ax=ax1)
ax1.set_title('Mean Passenger Count by GEO Region')
ax1.set_xlabel('GEO Region')
ax1.set_ylabel('Mean Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_geo_region_stats.iterrows():
    ax1.annotate(f"{row['mean_passenger_count']:,.0f}".replace(',', '.'), (row['GEO Region'], row['mean_passenger_count']), textcoords="offset points", xytext=(0,5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_geo_region_stats.values, colLabels=passenger_geo_region_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()

# Visualizar la tabla y el gráfico lado a lado (stddev)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))
sns.barplot(data=passenger_geo_region_stats, x='GEO Region', y='stddev_passenger_count', ax=ax1)
ax1.set_title('Stddev Passenger Count by GEO Region')
ax1.set_xlabel('GEO Region')
ax1.set_ylabel('Stddev Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_geo_region_stats.iterrows():
    ax1.annotate(f"{row['stddev_passenger_count']:,.0f}".replace(',', '.'), (row['GEO Region'], row['stddev_passenger_count']), textcoords="offset points", xytext=(0,5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_geo_region_stats.values, colLabels=passenger_geo_region_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()
```

```python
# Relacionar Passenger Count con Activity Type Code
passenger_activity_type_stats = df.groupBy("Activity Type Code").agg(
    mean("Passenger Count").alias("mean_passenger_count"),
    stddev("Passenger Count").alias("stddev_passenger_count"),
    count("Passenger Count").alias("count_passenger_count")
).toPandas()

# Redondear las estadísticas sin decimales
passenger_activity_type_stats['mean_passenger_count'] = passenger_activity_type_stats['mean_passenger_count'].round(0).astype(int)
passenger_activity_type_stats['stddev_passenger_count'] = passenger_activity_type_stats['stddev_passenger_count'].fillna(0).round(0).astype(int)
passenger_activity_type_stats['count_passenger_count'] = passenger_activity_type_stats['count_passenger_count'].round(0).astype(int)

# Ordenar las estadísticas alfabéticamente por Activity Type Code
passenger_activity_type_stats = passenger_activity_type_stats.sort_values('Activity Type Code')

# Configurar el estilo de los gráficos
sns.set(style="whitegrid")
```

```python
# Visualizar la tabla y el gráfico lado a lado (mean)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
sns.barplot(data=passenger_activity_type_stats, x='Activity Type Code', y='mean_passenger_count', ax=ax1)
ax1.set_title('Mean Passenger Count by Activity Type Code')
ax1.set_xlabel('Activity Type Code')
ax1.set_ylabel('Mean Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_activity_type_stats.iterrows():
    ax1.annotate(f"{row['mean_passenger_count']:,.0f}".replace(',', '.'), (row['Activity Type Code'], row['mean_passenger_count']), textcoords="offset points", xytext=(0,5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_activity_type_stats.values, colLabels=passenger_activity_type_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()

# Visualizar la tabla y el gráfico lado a lado (stddev)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
sns.barplot(data=passenger_activity_type_stats, x='Activity Type Code', y='stddev_passenger_count', ax=ax1)
ax1.set_title('Stddev Passenger Count by Activity Type Code')
ax1.set_xlabel('Activity Type Code')
ax1.set_ylabel('Stddev Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_activity_type_stats.iterrows():
    ax1.annotate(f"{row['stddev_passenger_count']:,.0f}".replace(',', '.'), (row['Activity Type Code'], row['stddev_passenger_count']), textcoords="offset points", xytext=(0,5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_activity_type_stats.values, colLabels=passenger_activity_type_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()
```

```python
# Relacionar Passenger Count con Terminal
passenger_terminal_stats = df.groupBy("Terminal").agg(
    mean("Passenger Count").alias("mean_passenger_count"),
    stddev("Passenger Count").alias("stddev_passenger_count"),
    count("Passenger Count").alias("count_passenger_count")
).toPandas()

# Redondear las estadísticas sin decimales
passenger_terminal_stats['mean_passenger_count'] = passenger_terminal_stats['mean_passenger_count'].round(0).astype(int)
passenger_terminal_stats['stddev_passenger_count'] = passenger_terminal_stats['stddev_passenger_count'].fillna(0).round(0).astype(int)
passenger_terminal_stats['count_passenger_count'] = passenger_terminal_stats['count_passenger_count'].round(0).astype(int)

# Ordenar las estadísticas alfabéticamente por Terminal
passenger_terminal_stats = passenger_terminal_stats.sort_values('Terminal')

# Configurar el estilo de los gráficos
sns.set(style="whitegrid")
```

```python
# Visualizar la tabla y el gráfico lado a lado (mean)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
sns.barplot(data=passenger_terminal_stats, x='Terminal', y='mean_passenger_count', ax=ax1)
ax1.set_title('Mean Passenger Count by Terminal')
ax1.set_xlabel('Terminal')
ax1.set_ylabel('Mean Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_terminal_stats.iterrows():
    ax1.annotate(f"{row['mean_passenger_count']:,.0f}".replace(',', '.'), (row['Terminal'], row['mean_passenger_count']), textcoords="offset points", xytext=(0, 5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_terminal_stats.values, colLabels=passenger_terminal_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()

# Visualizar la tabla y el gráfico lado a lado (stddev)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
sns.barplot(data=passenger_terminal_stats, x='Terminal', y='stddev_passenger_count', ax=ax1)
ax1.set_title('Stddev Passenger Count by Terminal')
ax1.set_xlabel('Terminal')
ax1.set_ylabel('Stddev Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_terminal_stats.iterrows():
    ax1.annotate(f"{row['stddev_passenger_count']:,.0f}".replace(',', '.'), (row['Terminal'], row['stddev_passenger_count']), textcoords="offset points", xytext=(0, 5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_terminal_stats.values, colLabels=passenger_terminal_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()
```

```python
# Relacionar Passenger Count con Boarding Area
passenger_boarding_area_stats = df.groupBy("Boarding Area").agg(
    mean("Passenger Count").alias("mean_passenger_count"),
    stddev("Passenger Count").alias("stddev_passenger_count"),
    count("Passenger Count").alias("count_passenger_count")
).toPandas()

# Redondear las estadísticas sin decimales
passenger_boarding_area_stats['mean_passenger_count'] = passenger_boarding_area_stats['mean_passenger_count'].round(0).astype(int)
passenger_boarding_area_stats['stddev_passenger_count'] = passenger_boarding_area_stats['stddev_passenger_count'].fillna(0).round(0).astype(int)
passenger_boarding_area_stats['count_passenger_count'] = passenger_boarding_area_stats['count_passenger_count'].round(0).astype(int)

# Ordenar las estadísticas alfabéticamente por Boarding Area
passenger_boarding_area_stats = passenger_boarding_area_stats.sort_values('Boarding Area')

# Configurar el estilo de los gráficos
sns.set(style="whitegrid")
```

```python
# Visualizar la tabla y el gráfico lado a lado (mean)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
sns.barplot(data=passenger_boarding_area_stats, x='Boarding Area', y='mean_passenger_count', ax=ax1)
ax1.set_title('Mean Passenger Count by Boarding Area')
ax1.set_xlabel('Boarding Area')
ax1.set_ylabel('Mean Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_boarding_area_stats.iterrows():
    ax1.annotate(f"{row['mean_passenger_count']:,.0f}".replace(',', '.'), (row['Boarding Area'], row['mean_passenger_count']), textcoords="offset points", xytext=(0, 5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=0)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_boarding_area_stats.values, colLabels=passenger_boarding_area_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()

# Visualizar la tabla y el gráfico lado a lado (stddev)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
sns.barplot(data=passenger_boarding_area_stats, x='Boarding Area', y='stddev_passenger_count', ax=ax1)
ax1.set_title('Stddev Passenger Count by Boarding Area')
ax1.set_xlabel('Boarding Area')
ax1.set_ylabel('Stddev Passenger Count')
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
for i, row in passenger_boarding_area_stats.iterrows():
    ax1.annotate(f"{row['stddev_passenger_count']:,.0f}".replace(',', '.'), (row['Boarding Area'], row['stddev_passenger_count']), textcoords="offset points", xytext=(0, 5), ha='center')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=0)
ax2.axis('tight')
ax2.axis('off')
table = ax2.table(cellText=passenger_boarding_area_stats.values, colLabels=passenger_boarding_area_stats.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()
```

```python
# Columnas categóricas que necesitamos convertir
categorical_columns = ['Operating Airline', 'GEO Region', 'Activity Type Code', 'Price Category Code', 'Terminal', 'Boarding Area', 'Month']

# Crear un índice para cada columna categórica
for column in categorical_columns:
    indexer = StringIndexer(inputCol=column, outputCol=column + "_Index")
    df = indexer.fit(df).transform(df)

# Seleccionar solo las columnas necesarias para la matriz de correlación
selected_columns = ['Activity Period', 'Passenger Count', 'Adjusted Passenger Count', 'Year'] + [col + "_Index" for col in categorical_columns]
df_selected = df.select(selected_columns)

# Convertir a DataFrame de Pandas para la matriz de correlación
pandas_df = df_selected.toPandas()
```

```python
# Calcular la matriz de correlación
correlation_matrix = pandas_df.corr()

# Visualizar la matriz de correlación
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Matriz de Correlación')
plt.show()
```

```python
# Seleccionar las 10 correlaciones más altas (excluyendo 1.0 que es la correlación de una variable consigo misma)
corr_pairs = correlation_matrix.unstack()
sorted_pairs = corr_pairs.sort_values(kind="quicksort", ascending=False)

# Filtrar las correlaciones más altas excluyendo los 1.0
high_corr_pairs = sorted_pairs[(sorted_pairs < 1) & (sorted_pairs > -1)].head(10)

# Convertir las correlaciones más altas en un DataFrame para visualización
high_corr_pairs_df = high_corr_pairs.reset_index()
high_corr_pairs_df.columns = ['Variable 1', 'Variable 2', 'Correlation']
fig, ax = plt.subplots(figsize=(10, 6))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=high_corr_pairs_df.values, colLabels=high_corr_pairs_df.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.tight_layout()
plt.show()
```

```python
# Configurar el modelo K-Means
kmeans = KMeans().setK(3).setSeed(1)  # K=3 para agrupar en 3 clusters

# Entrenar el modelo
model = kmeans.fit(df_features)

# Hacer predicciones
predictions = model.transform(df_features)

# Mostrar las predicciones
predictions.select('features', 'prediction').show(5)
```

```
# Evaluar el modelo
wssse = model.summary.trainingCost
print(f"Within Set Sum of Squared Errors = {wssse}")
```

```
# Convertir a DataFrame de Pandas para la visualización
predictions_pd = predictions.select('Passenger Count', 'Adjusted Passenger Count', 'prediction').toPandas()

print(predictions_pd)

# Crear el gráfico de dispersión
plt.figure(figsize=(12, 8))
sns.scatterplot(data=predictions_pd, x='Passenger Count', y='Adjusted Passenger Count', hue='prediction', palette='viridis')
plt.title('Clustering de Pasajeros por K-Means')
plt.xlabel('Passenger Count')
plt.ylabel('Adjusted Passenger Count')
ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x).replace(',', '.')))
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda y, pos: '{:,.0f}'.format(y).replace(',', '.')))
plt.legend(title='Cluster')
plt.show()
```