BIG DATA

PROYECTO FINAL

Samuel Porcel Rodríguez

New Technology School

Tokio.

New
Technology
School

SAMUEL PORCEL RODRÍGUEZ

RESPONSABLE DEL DESARROLLO DEL PROYECTO FINAL

Fecha: 13/11/2024

ÍNDICE

DESARROLLO

PROYECTO FINAL: BIG DATA

Documento técnico

TAREAS:

1. Categorizar según el tipo de dato y su estructura:

Nombre del campo	Tipo de dato
1. Activity Period	Numérico discreto
2. Operating Airline	Categórico nominal
3. Operating Airline IATA Code	Categórico nominal
4. Published Airline	Categórico nominal
5. Published Airline IATA Code	Categórico nominal
6. GEO Summary	Categórico nominal
7. GEO Region	Categórico nominal
8. Activity Type Code	Categórico nominal
9. Price Category Code	Categórico nominal
10. Terminal	Categórico ordinal
11. Boarding Area	Categórico ordinal
12. Passenger Count	Numérico discreto
13. Adjusted Activity Type Code	Categórico nominal
14. Adjusted Passenger Count	Numérico discreto
15. Year	Numérico discreto
16. Month	Categórico ordinal

2. Cassandra:

- Crear espacio y tabla:

```
CREATE KEYSPACE statistics
WITH replication = {'class':'SimpleStrategy', 'replication_factor': '1'}
AND durable_writes = TRUE;


CREATE TABLE statistics.traffic (
"Activity Period" int,
"Operating Airline" text,
"Operating Airline IATA Code" text,
"Published Airline" text,
"Published Airline IATA Code" text,
"GEO Summary" text,
"GEO Region" text,
"Activity Type Code" text,
"Price Category Code" text,
"Terminal" text,
"Boarding Area" text,
"Passenger Count" int,
"Adjusted Activity Type Code" text,
"Adjusted Passenger Count" int,
"Year" int,
"Month" text,
PRIMARY KEY ("Operating Airline", "Year")
) WITH CLUSTERING ORDER BY ("Year" ASC);
```

- Insertar datos en las columnas de la tabla:

```
•	Insertar datos en las columnas de la tabla:

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (200507, 'Asiana Airlines', 'OZ', 'Asiana Airlines', 'OZ', 'International', 'Asia', 'Deplaned', 'Other', 'International', 'A', 5041, 'Deplaned', 5041, 2005, 'July');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (200509, 'Miami Air International', 'GL', 'Miami Air International', 'GL', 'Domestic', 'US', 'Deplaned', 'Other', 'International', 'A', 166, 'Deplaned', 166, 2015, 'September');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201603, 'Air France', 'AF', 'Air France', 'AF', 'International', 'Europe', 'Enplaned', 'Other', 'International', 'A', 10574, 'Enplaned', 10574, 2016, 'March');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201403, 'LAN Peru', 'LP', 'LAN Peru', 'LP', 'International', 'South America', 'Enplaned', 'Other', 'International', 'A', 2886, 'Enplaned', 2886, 2014, 'March');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201602, 'Air China', 'CA', 'Air China', 'CA', 'International', 'Asia', 'Enplaned', 'Other', 'International', 'G', 7731, 'Enplaned', 7731, 2016, 'February');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201601, 'Air China', 'CA', 'Air China', 'CA', 'International', 'Asia', 'Enplaned', 'Other', 'International', 'G', 7125, 'Enplaned', 7125, 2016, 'January');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201512, 'Air China', 'CA', 'Air China', 'CA', 'International', 'Asia', 'Enplaned', 'Other', 'International', 'G', 9341, 'Enplaned', 9341, 2015, 'December');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201010, 'Air Berlin', 'AB', 'Air Berlin', 'AB', 'International', 'Europe', 'Enplaned', 'Other', 'International', 'G', 1689, 'Enplaned', 1689, 2010, 'October');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201010, 'Air Berlin', 'AB', 'Air Berlin', 'AB', 'International', 'Europe', 'Deplaned', 'Other', 'International', 'G', 1455, 'Deplaned', 1455, 2010, 'October');

INSERT INTO statistics.traffic ("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", "Passenger Count", "Adjusted Activity Type Code", "Adjusted Passenger Count", "Year", "Month")
VALUES (201210, 'Air Berlin', 'AB', 'Air Berlin', 'AB', 'International', 'Europe', 'Enplaned', 'Other', 'International', 'A', 1487, 'Enplaned', 1487, 2012, 'October');
```

```
•	Verificar la importación visualizando la tabla:
SELECT * FROM statistics.traffic;

a)	RECUPERAR TODOS LOS REGISTROS DE LA AEROLÍNEA “AIR CHINA”:
SELECT * FROM statistics.traffic WHERE "Operating Airline" = 'Air China' ALLOW FILTERING;
```

```
b)	RECUPERAR TODOS LOS VUELOS DE LA COMPAÑÍA “AIR BERLÍN” EMBARCADOS POR LA PUERTA “G”:
SELECT * FROM statistics.traffic
WHERE "Operating Airline" = 'Air Berlin'
AND "Boarding Area" = 'G' ALLOW FILTERING;
```

3. Dataframe de Pyspark:

DESARROLLO, ejercicio 3:

Cargar el conjunto de datos en un dataframe

```
[ ] data_path = '/content/drive/MyDrive/TokioSchool/data_curso/'

    df = spark.read.options(header=True, inferSchema=True).csv(data_path + 'Air_Traffic_Passenger_Statistics.csv')
```

¿Cuántas compañías diferentes aparecen en el fichero?

```
[ ] df.select('Operating Airline').distinct().count()
```

¿Cuántos pasajeros tienen de media los vuelos de cada compañía?

```
[ ] df_compañia = df.groupBy('Operating Airline').agg (F.mean('Passenger Count').alias('mean_passenger_count'))
    df_compañia.show()
```

```
[ ] # Visualizamos con collect para poder mostrar la media de las 77 compañias, ya que con .show() es más visual pero solo muestra 20 filas.
    resultados = df_compañia.collect()

    # Mostrar los resultados
    for fila in resultados:
        print(fila)
```

Eliminaremos los registros duplicados por el campo "GEO Región", manteniendo únicamente aquel con mayor número de pasajeros.

```
[ ] max_passengers = df.groupBy("GEO Region").agg(F.max("Passenger Count").alias("max_passenger_count"))
    max_passengers.show()
```

Volcaremos los resultados de los dos puntos anteriores a un CSV.

```
df_compañia.coalesce(1).write \
  .mode("overwrite") \
  .format("csv") \
  .option("header", "true") \
  .save("/content/drive/MyDrive/TokioSchool/data/")
```

```
[ ] max_passengers.coalesce(1).write \
      .mode("overwrite") \
      .format("csv") \
      .option("header", "true") \
      .save("/content/drive/MyDrive/TokioSchool/data_1/")

    spark.stop()
```

4. Análisis descriptivo, matriz de correlación y logaritmo:

DESARROLLO, ejercicio 4:

Cargar el conjunto de datos en un dataframe

```
[13] data_path = '/content/drive/MyDrive/TokioSchool/data_curso/'

     df = spark.read.options(header=True, inferSchema=True).csv(data_path + 'Air_Traffic_Passenger_Statistics.csv')
```

ANÁLISIS DESCRIPTIVO

- MEDIA DE CADA ELEMENTO

Media del número de pasajeros según la región, categoria de precio y el mes

```
[14] df_region = df.groupBy('GEO Region').agg (F.mean('Passenger Count').alias('mean_passenger_count')).show()

     df_price = df.groupBy('Price Category Code').agg (F.mean('Passenger Count').alias('mean_passenger_count')).show()

     df_month = df.groupBy('Month').agg (F.mean('Passenger Count').alias('mean_passenger_count')).show()
```

- DESVIACIÓN TÍPICA DE CADA ELEMENTO

Desviación estandar del número de pasajeros según la región, categoria de precio y el mes

```
[15] df_region = df.groupBy('GEO Region').agg (F.std('Passenger Count').alias('std_passenger_count')).show()

     df_price = df.groupBy('Price Category Code').agg (F.std('Passenger Count').alias('std_passenger_count')).show()

     df_month = df.groupBy('Month').agg (F.std('Passenger Count').alias('std_passenger_count')).show()
```

- Análisis de correlación
- Matriz de correlación en el resultado

```
[16] # Transformo todas las variables para poder calcular la correlacion. Crear nuevas columnas
     operating_airline_indexer = StringIndexer(inputCol="Operating Airline", outputCol="Operating Airline Index")
     df = operating_airline_indexer.fit(df).transform(df)

     geo_summary_indexer = StringIndexer(inputCol="GEO Summary", outputCol="GEO Summary Index")
     df = geo_summary_indexer.fit(df).transform(df)

     geo_region_indexer = StringIndexer(inputCol="GEO Region", outputCol="GEO Region Index")
     df = geo_region_indexer.fit(df).transform(df)

     activity_type_code_indexer = StringIndexer(inputCol="Activity Type Code", outputCol="Activity Type Code Index")
     df = activity_type_code_indexer.fit(df).transform(df)

     price_category_code_indexer = StringIndexer(inputCol="Price Category Code", outputCol="Price Category Code Index")
     df = price_category_code_indexer.fit(df).transform(df)

     terminal_indexer = StringIndexer(inputCol="Terminal", outputCol="Terminal Index")
     df = terminal_indexer.fit(df).transform(df)

     boarding_area_indexer = StringIndexer(inputCol="Boarding Area", outputCol="Boarding Area Index")
     df = boarding_area_indexer.fit(df).transform(df)
```

```
[18] # Mapeo de meses a números
     df = df.withColumn ('month_numero',
         when (df['Month'] == 'January', 1)
         .when (df['Month'] == 'February', 2)
         .when (df['Month'] == 'March', 3)
         .when (df['Month'] == 'April', 4)
         .when (df['Month'] == 'May', 5)
         .when (df['Month'] == 'June', 6)
         .when (df['Month'] == 'July', 7)
         .when (df['Month'] == 'August', 8)
         .when (df['Month'] == 'September', 9)
         .when (df['Month'] == 'October', 10)
         .when (df['Month'] == 'November', 11)
         .when (df['Month'] == 'December', 12)
     )
```

```
[19] # Generar matriz de correlación y calcular la correlación entre todas las variables numericas
     numeric_cols = ['Operating Airline Index', 'GEO Summary Index', 'GEO Region Index', 'Activity Type Code Index', 'Price Category Code Index',
                     'Terminal Index', 'Boarding Area Index', 'Passenger Count', 'Year', 'month_numero']

     for col1 in numeric_cols:
         for col2 in numeric_cols:
             corr_value = df.stat.corr(col1, col2)
             print(f"Correlación entre {col1} y {col2}: {corr_value}")
```

- Algoritmo: Árbol de decisión

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.regression import GBTRegressor

# Crear una sesión de Spark
spark = SparkSession.builder.appName("Air Traffic Passenger Prediction").getOrCreate()

# Cargar los datos
df = spark.read.option("header", True).csv("/content/drive/MyDrive/TokioSchool/data_curso/Air_Traffic_Passenger_Statistics.csv")

# Transformar columnas a indice
operating_airline_indexer = StringIndexer(inputCol="Operating Airline", outputCol="Operating Airline Index")
df = operating_airline_indexer.fit(df).transform(df)

published_airline_indexer = StringIndexer(inputCol="Published Airline", outputCol="Published Airline Index")
df = published_airline_indexer.fit(df).transform(df)

GEO_summary_indexer = StringIndexer(inputCol="GEO Summary", outputCol="GEO Summary Index")
df = GEO_summary_indexer.fit(df).transform(df)

activity_type_code_indexer = StringIndexer(inputCol="Activity Type Code", outputCol="Activity Type Code Index")
df = activity_type_code_indexer.fit(df).transform(df)
```

```
df = operating_airline_indexer.fit(df).transform(df)

published_airline_indexer = StringIndexer(inputCol="Published Airline", outputCol="Published Airline Index")
df = published_airline_indexer.fit(df).transform(df)

GEO_summary_indexer = StringIndexer(inputCol="GEO Summary", outputCol="GEO Summary Index")
df = GEO_summary_indexer.fit(df).transform(df)

activity_type_code_indexer = StringIndexer(inputCol="Activity Type Code", outputCol="Activity Type Code Index")
df = activity_type_code_indexer.fit(df).transform(df)

price_category_code_indexer = StringIndexer(inputCol="Price Category Code", outputCol="Price Category Code Index")
df = price_category_code_indexer.fit(df).transform(df)

terminal_indexer = StringIndexer(inputCol="Terminal", outputCol="Terminal Index")
df = terminal_indexer.fit(df).transform(df)

boarding_area_indexer = StringIndexer(inputCol="Boarding Area", outputCol="Boarding Area Index")
df = boarding_area_indexer.fit(df).transform(df)

adjusted_activity_type_code_indexer = StringIndexer(inputCol="Adjusted Activity Type Code", outputCol="Adjusted Activity Type Code Index")
df = adjusted_activity_type_code_indexer.fit(df).transform(df)

month_indexer = StringIndexer(inputCol="Month", outputCol="Month Index")
df = month_indexer.fit(df).transform(df)
```

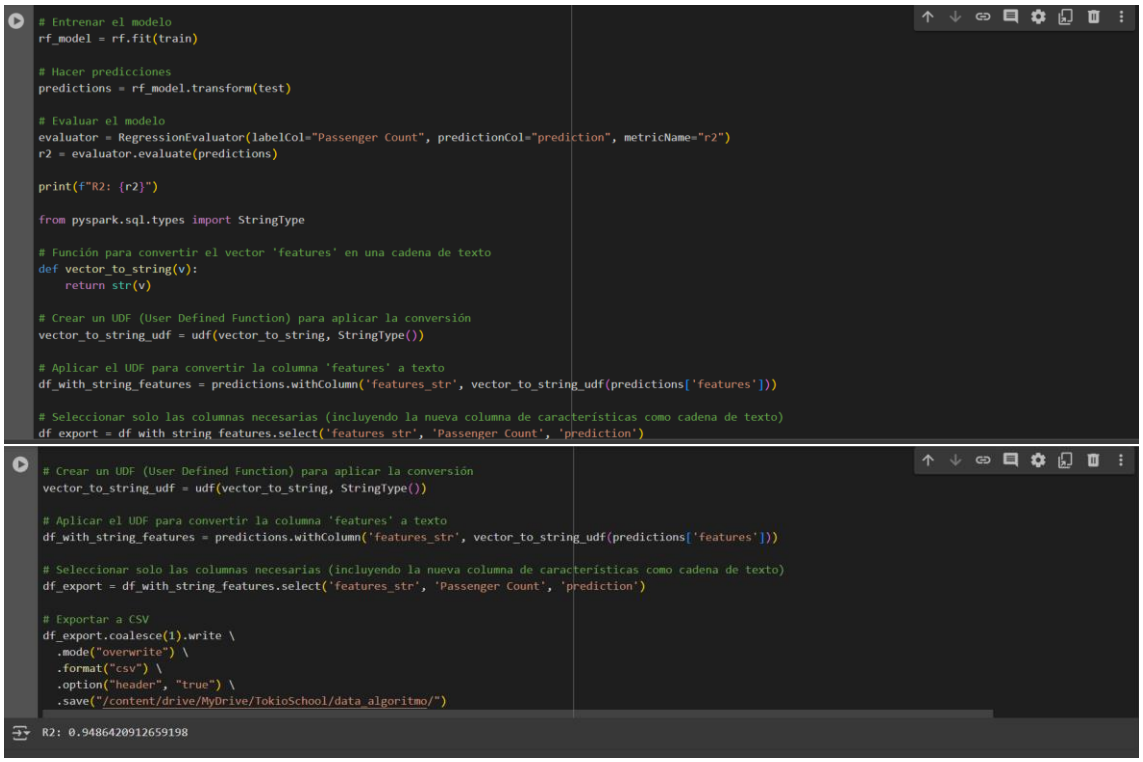
```
# Modificar tipos de datos
df = df.withColumn('Activity Period', df['Activity Period'].cast('float'))
df = df.withColumn('Operating Airline Index', df['Operating Airline Index'].cast('float'))
df = df.withColumn('Published Airline Index', df['Published Airline Index'].cast('float'))
df = df.withColumn('GEO Summary Index', df['GEO Summary Index'].cast('float'))
df = df.withColumn('Activity Type Code Index', df['Activity Type Code Index'].cast('float'))
df = df.withColumn('Price Category Code Index', df['Price Category Code Index'].cast('float'))
df = df.withColumn('Terminal Index', df['Terminal Index'].cast('float'))
df = df.withColumn('Boarding Area Index', df['Boarding Area Index'].cast('float'))
df = df.withColumn('Passenger Count', df['Passenger Count'].cast('float'))
df = df.withColumn('Adjusted Activity Type Code Index', df['Adjusted Activity Type Code Index'].cast('float'))
df = df.withColumn('Year', df['Year'].cast('float'))
df = df.withColumn('Month Index', df['Month Index'].cast('float'))

# Crear la columna 'features' utilizando VectorAssembler
va = VectorAssembler(inputCols=['Activity Period','Operating Airline Index', 'Published Airline Index', 'GEO Summary Index','Activity Type Code Index',
                                'Price Category Code Index','Terminal Index','Boarding Area Index','Year', 'Adjusted Activity Type Code Index','Month Index'], outpu

df = va.transform(df)

# Dividir los datos en entrenamiento y prueba
train, test = df.randomSplit([0.8, 0.2])

# Definir el modelo de regresión basado en árboles de decisión
rf = GBTRegressor(featuresCol='features', labelCol='Passenger Count', maxIter=100)
```



```python
# Entrenar el modelo
rf_model = rf.fit(train)

# Hacer predicciones
predictions = rf_model.transform(test)

# Evaluar el modelo
evaluator = RegressionEvaluator(labelCol="Passenger Count", predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)

print(f"R2: {r2}")

from pyspark.sql.types import StringType

# Función para convertir el vector 'features' en una cadena de texto
def vector_to_string(v):
    return str(v)

# Crear un UDF (User Defined Function) para aplicar la conversión
vector_to_string_udf = udf(vector_to_string, StringType())

# Aplicar el UDF para convertir la columna 'features' a texto
df_with_string_features = predictions.withColumn('features_str', vector_to_string_udf(predictions['features']))

# Seleccionar solo las columnas necesarias (incluyendo la nueva columna de características como cadena de texto)
df_export = df_with_string_features.select('features_str', 'Passenger Count', 'prediction')
```

```python
# Crear un UDF (User Defined Function) para aplicar la conversión
vector_to_string_udf = udf(vector_to_string, StringType())

# Aplicar el UDF para convertir la columna 'features' a texto
df_with_string_features = predictions.withColumn('features_str', vector_to_string_udf(predictions['features']))

# Seleccionar solo las columnas necesarias (incluyendo la nueva columna de características como cadena de texto)
df_export = df_with_string_features.select('features_str', 'Passenger Count', 'prediction')

# Exportar a CSV
df_export.coalesce(1).write \
  .mode("overwrite") \
  .format("csv") \
  .option("header", "true") \
  .save("/content/drive/MyDrive/TokioSchool/data_algoritmo/")
```

```
R2: 0.9486420912659198
```