EXPERIMENT - 9B

Aim: Visualization using Power BI and Performing EDA,logistic and linear regression using Apache Spark.

About Dataset:

Link to our dataset: https://www.kaggle.com/datasets/divyansh22/flight-delay-prediction

This is the first part of flight delay prediction i.e. for the month of January. This data is collected from the Bureau of Transportation Statistics, Govt. of the USA. This data is open-sourced under U.S. Govt. Works. This dataset contains all the flights in the month of January 2019 and January 2020.

There are more than 400,000 flights in the month of January itself throughout the United States. The features were manually chosen to do a primary time series analysis. There are several other features available on their website.

Theory:

Spark is an Apache project advertised as "lightning fast cluster computing". It has a thriving open-source community and is the most active Apache project at the moment. Spark provides a faster and more general data processing platform. Spark lets you run programs up to 100x faster in memory, or 10x faster on disk, than Hadoop. Last year, Spark took over Hadoop by completing the 100 TB Daytona GraySort contest 3x faster on one tenth the number of machines and it also became the fastest open source engine for sorting a petabyte. Spark also makes it possible to write code more quickly as you have over 80 high-level operators at your disposal. Another important aspect when learning how to use Apache Spark is the interactive shell (REPL) which it provides out-of-the box. Using REPL, one can test the outcome of each line of code without first needing to code and execute the entire job. The path to working code is thus much shorter and ad-hoc data analysis is made possible.

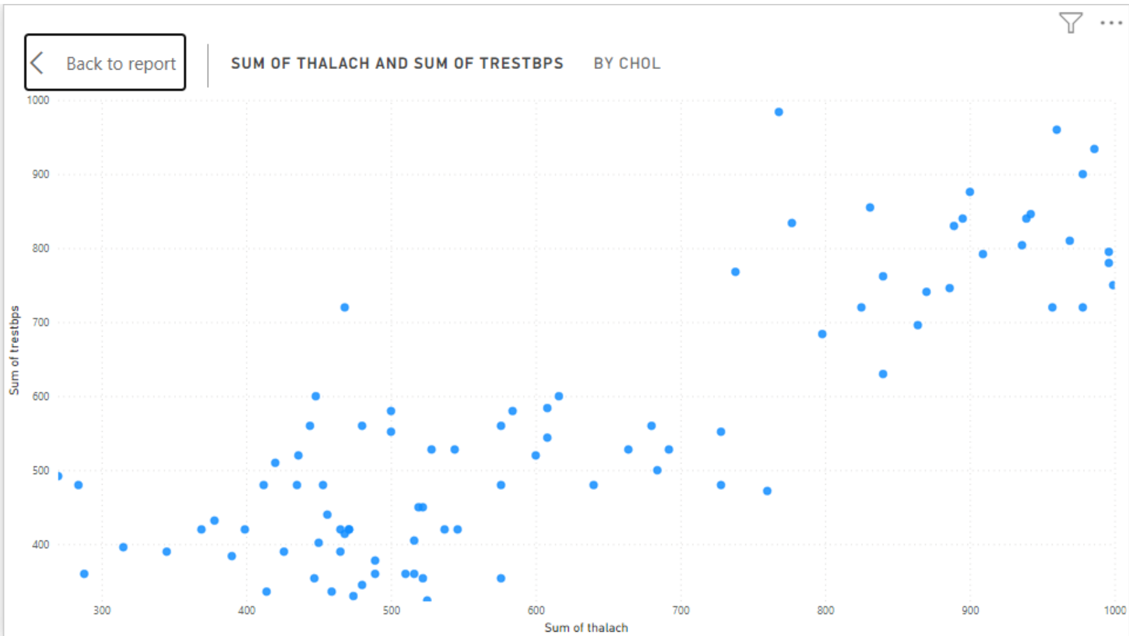
Power BI is a data visualization platform that is primarily used for business intelligence. Power BI's dashboard, which is intended for use by business professionals with varying levels of data knowledge, is capable of reporting and visualizing data in a variety of formats, including graphs, maps, charts, scatter plots, and more. Power BI is made up of several interconnected applications, including Power BI Desktop, Pro, Premium, Mobile, Embedded, and Report Server. While some of these apps are free to use, paid subscriptions to the pro and premium versions offer enhanced analytics capabilities.

<u>Implementation</u>:

<u>Power BI</u>

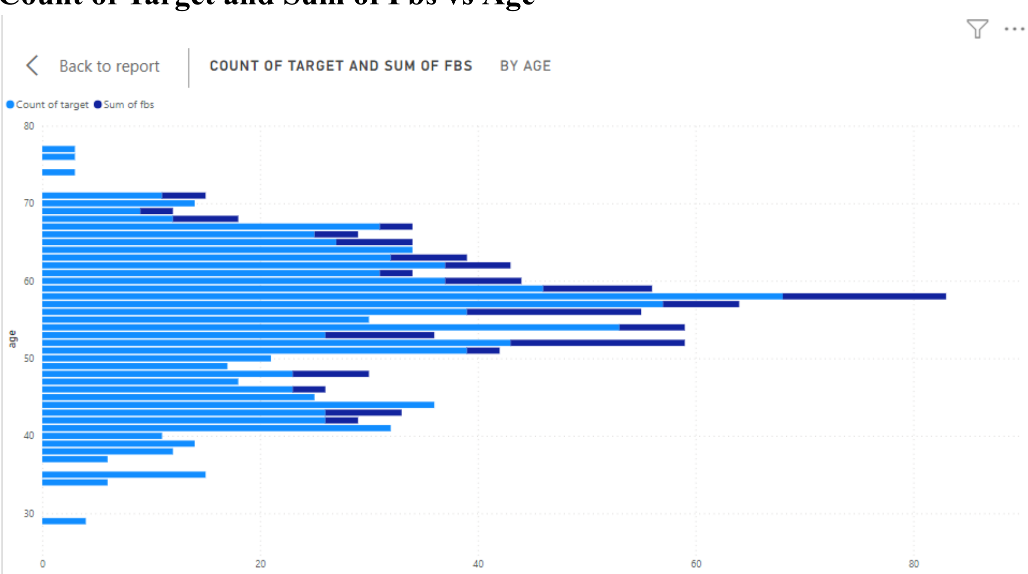
1) Scatterplot

Sum of thalach and trestbps vs chol



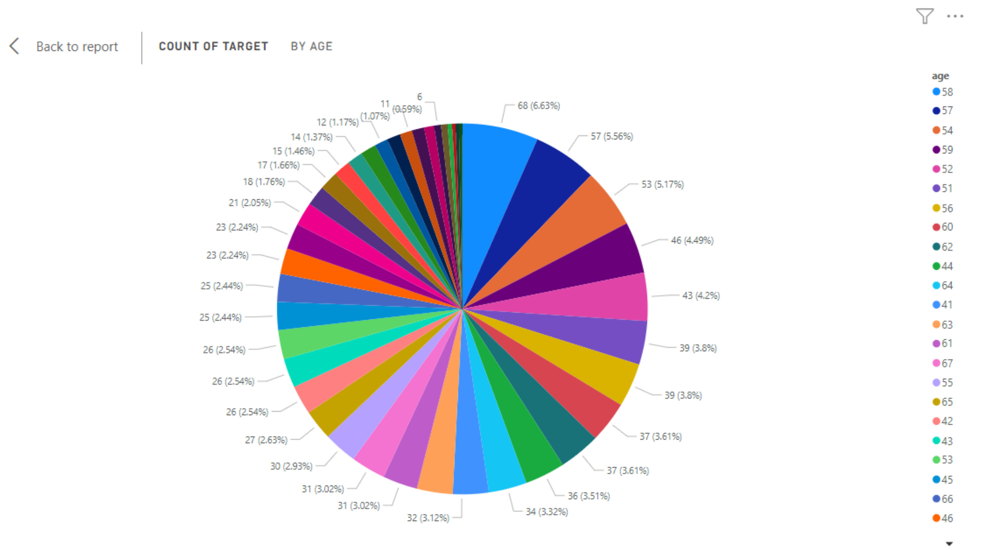
2) Bar graph

Count of Target and Sum of Fbs vs Age

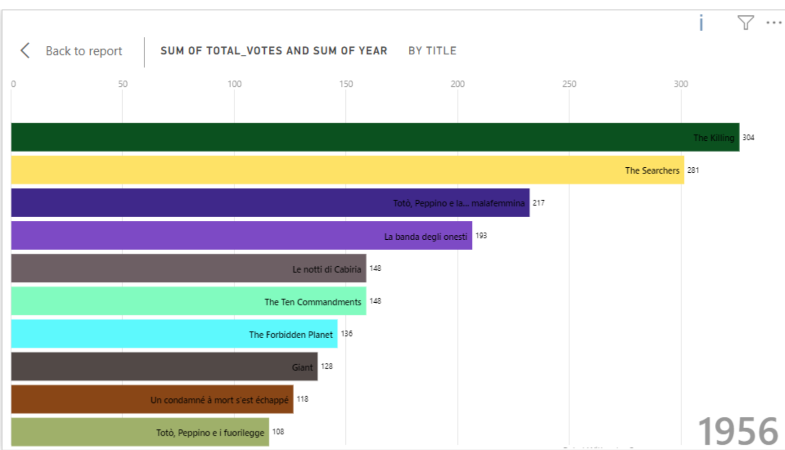


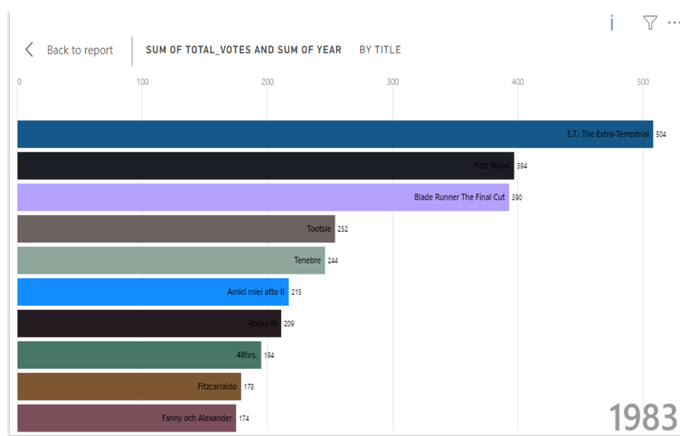
3) Pie chart

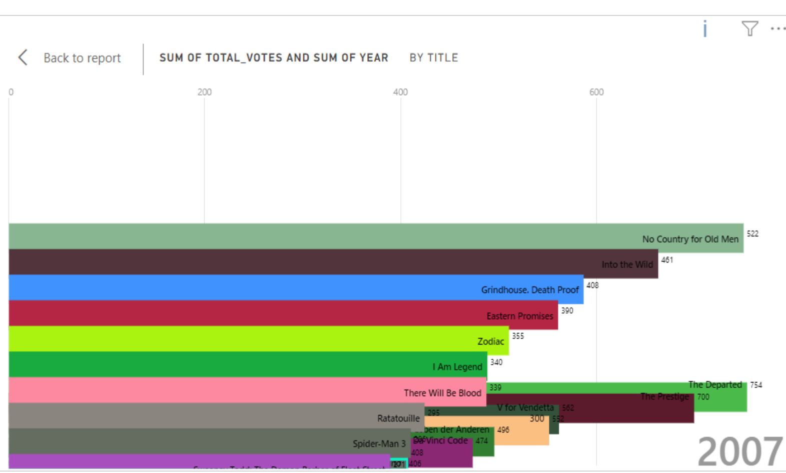
Count of Target by age: It shows the percentage of each age which are likely to get heart disease.

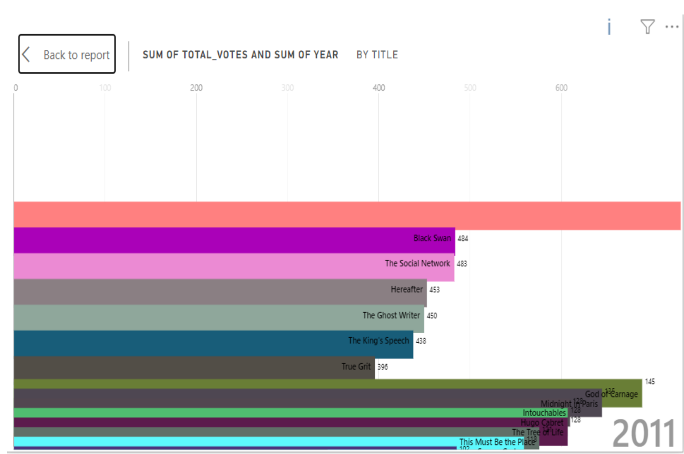


4) Animated visualization









5) Waterfall graph

Sum of trestbps vs thal and age

<u>Apache Spark:-</u>

Step 1: Loading the dataset

Cmd 1

```
from pyspark.sql import SparkSession
```

Command took 2.22 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 4:44:46 PM on My Cluster 5

Cmd 2

```
spark = SparkSession.builder.appName('flight').getOrCreate()
```

Command took 0.36 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 4:44:46 PM on My Cluster 5

Cmd 3

```
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/2020.mayuri.yerande@ves.ac.in/Jan_2020_ontime-2.csv")
```

▸ (1) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [DAY_OF_MONTH: string, DAY_OF_WEEK: string ... 20 more fields]

Command took 18.60 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 4:44:48 PM on My Cluster 5

Step 2: Describing the data

```
df.printSchema()
```

```
root
 |-- DAY_OF_MONTH: string (nullable = true)
 |-- DAY_OF_WEEK: string (nullable = true)
 |-- OP_UNIQUE_CARRIER: string (nullable = true)
 |-- OP_CARRIER_AIRLINE_ID: string (nullable = true)
 |-- OP_CARRIER: string (nullable = true)
 |-- TAIL_NUM: string (nullable = true)
 |-- OP_CARRIER_FL_NUM: string (nullable = true)
 |-- ORIGIN_AIRPORT_ID: string (nullable = true)
 |-- ORIGIN_AIRPORT_SEQ_ID: string (nullable = true)
 |-- ORIGIN: string (nullable = true)
 |-- DEST_AIRPORT_ID: string (nullable = true)
 |-- DEST_AIRPORT_SEQ_ID: string (nullable = true)
 |-- DEST: string (nullable = true)
 |-- DEP_TIME: string (nullable = true)
 |-- DEP_DEL15: string (nullable = true)
 |-- DEP_TIME_BLK: string (nullable = true)
 |-- ARR_TIME: string (nullable = true)
 |-- ARR_DEL15: string (nullable = true)
 |-- CANCELLED: string (nullable = true)
 |-- DIVERTED: string (nullable = true)
Command took 0.23 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 4:48:32 PM on My Cluster 5
Cmd 5
```

```
df.groupBy('DEST').count().show()
```

(2) Spark Jobs

```
+----+-----+
|DEST|count|
+----+-----+
| BGM|   61|
| INL|   54|
| PSE|   68|
| MSY| 4708|
| PPG|    9|
| GEG| 1111|
| DRT|   59|
| BUR| 2775|
| SNA| 3296|
| GRB|  404|
| GTF|  159|
| IDA|  139|
```

```
df.show()
```

(1) Spark Jobs

```
|DAY_OF_MONTH|DAY_OF_WEEK|OP_UNIQUE_CARRIER|OP_CARRIER_AIRLINE_ID|OP_CARRIER|TAIL_NUM|OP_CARRIER_FL_NUM|ORIGIN_AIRPORT_ID|ORIGIN_AIRPORT_SEQ_ID|ORI
GIN|DEST_AIRPORT_ID|DEST_AIRPORT_SEQ_ID|DEST|DEP_TIME|DEP_DEL15|DEP_TIME_BLK|ARR_TIME|ARR_DEL15|CANCELLED|DIVERTED|DISTANCE|_c21|
|           1|          3|               EV|                20366|        EV|  N48901|             4397|            13930|              1393007|
ORD|          11977|            1197705| GRB|    1003|     0.00|   1000-1059|    1117|     0.00|     0.00|     0.00|  174.00|null|
|           1|          3|               EV|                20366|        EV|  N16976|             4401|            15370|              1537002|
TUL|          13930|            1393007| ORD|    1027|     0.00|   1000-1059|    1216|     0.00|     0.00|     0.00|  585.00|null|
|           1|          3|               EV|                20366|        EV|  N12167|             4404|            11618|              1161802|
EWR|          15412|            1541205| TYS|    1848|     0.00|   1800-1859|    2120|     0.00|     0.00|     0.00|  631.00|null|
|           1|          3|               EV|                20366|        EV|  N14902|             4405|            10781|              1078105|
BTR|          12266|            1226603| IAH|    1846|     0.00|   1800-1859|    2004|     0.00|     0.00|     0.00|  253.00|null|
|           1|          3|               EV|                20366|        EV|  N606UX|             4407|            14524|              1452401|
RIC|          12266|            1226603| IAH|    1038|     0.00|   1000-1059|    1330|     0.00|     0.00|     0.00| 1157.00|null|
|           1|          3|               EV|                20366|        EV|  N605UX|             4408|            13871|              1387102|
OMA|          12266|            1226603| IAH|    0555|     0.00|   0600-0659|    0838|     0.00|     0.00|     0.00|  781.00|null|
|           1|          3|               EV|                20366|        EV|  N14991|             4410|            12266|              1226603|
IAH|          14814|            1481402| SHV|    1610|     0.00|   1600-1659|    1718|     0.00|     0.00|     0.00|  192.00|null|
|           1|          3|               EV|                20366|        EV|  N15986|             4411|            12266|              1226603|
Command took 1.42 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 4:48:33 PM on My Cluster 5
```

Step 3: Converting needed columns into integer type

```python
#converting needed columns into inter type
from pyspark.sql import functions as f
from pyspark.sql.types import IntegerType

numeric_columns = ['DAY_OF_MONTH',
'DAY_OF_WEEK',
'OP_UNIQUE_CARRIER',
'OP_CARRIER_AIRLINE_ID',
'OP_CARRIER',
'TAIL_NUM' ,
'OP_CARRIER_FL_NUM','ORIGIN_AIRPORT_ID' ,'ORIGIN_AIRPORT_SEQ_ID']
for column in numeric_columns:
    df = df.withColumn(column,f.col(column).cast(IntegerType()))
df.printSchema()
```

df: pyspark.sql.dataframe.DataFrame = [DAY_OF_MONTH: integer, DAY_OF_WEEK: integer ... 20 more fields]

```
root
 |-- DAY_OF_MONTH: integer (nullable = true)
 |-- DAY_OF_WEEK: integer (nullable = true)
 |-- OP_UNIQUE_CARRIER: integer (nullable = true)
 |-- OP_CARRIER_AIRLINE_ID: integer (nullable = true)
 |-- OP_CARRIER: integer (nullable = true)
 |-- TAIL_NUM: integer (nullable = true)
 |-- OP_CARRIER_FL_NUM: integer (nullable = true)
 |-- ORIGIN_AIRPORT_ID: integer (nullable = true)
 |-- ORIGIN_AIRPORT_SEQ_ID: integer (nullable = true)
 |-- ORIGIN: string (nullable = true)
 |-- DEST_AIRPORT_ID: string (nullable = true)
```

Step 4: Statistical measures of data

```
#mean
from pyspark.sql.functions import mean, col
df_stats = df.select(
    mean(col('DAY_OF_MONTH')).alias('Mean DAY_OF_MONTH'),
    mean(col('ARR_TIME')).alias('Mean ARR_TIME'),
    mean(col('DEP_TIME')).alias('DEP_TIME'),


).collect()
for i in df_stats:
  row = i.asDict()
  for k in row:
    print(k," - ", row[k])
```

▸ (2) Spark Jobs

```
Mean DAY_OF_MONTH  -  16.014354256058326
Mean ARR_TIME  -  1477.9689240359771
DEP_TIME  -  1331.512559057871
```

```
print("Median DAY_OF_MONTH - ",df.approxQuantile("DAY_OF_MONTH", [0.5], 0.25))
```

▸ (1) Spark Jobs

```
Median DAY_OF_MONTH -  [9.0]
```

Command took 4.17 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 5:05:19 PM on My Cluster 5

Step 5: Correlation analysis - Pearson

```
import pandas as pd
from pyspark.mllib.stat import Statistics

features = df.select(numeric_columns).rdd.map(lambda row: row[0:])

corr_mat=Statistics.corr(features, method="pearson")

corr_df = pd.DataFrame(corr_mat,index=numeric_columns, columns=numeric_columns)
corr_df
```

▸ (4) Spark Jobs

	DAY_OF_MONTH	DAY_OF_WEEK	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE_ID	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	OR
DAY_OF_MONTH	1.000000	-0.053947	NaN	-0.001769	NaN	NaN	-0.020052	-0.004940	
DAY_OF_WEEK	-0.053947	1.000000	NaN	0.007127	NaN	NaN	0.056448	0.005209	
OP_UNIQUE_CARRIER	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	
OP_CARRIER_AIRLINE_ID	-0.001769	0.007127	NaN	1.000000	NaN	NaN	0.425973	-0.046500	
OP_CARRIER	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	
TAIL_NUM	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	
OP_CARRIER_FL_NUM	-0.020052	0.056448	NaN	0.425973	NaN	NaN	1.000000	-0.052829	
ORIGIN_AIRPORT_ID	-0.004940	0.005209	NaN	-0.046500	NaN	NaN	-0.052829	1.000000	
ORIGIN_AIRPORT_SEQ_ID	-0.004940	0.005209	NaN	-0.046500	NaN	NaN	-0.052830	1.000000	

```python

# get a boolean dataframe where true means that a pair of variables is highly correlated
highly_correlated_df = (abs(corr_df) > .5) & (corr_df < 1.0)

# get the names of the variables so we can use them to slice the dataframe
correlated_vars_index = (highly_correlated_df==True).any()
correlated_var_names = correlated_vars_index[correlated_vars_index==True].index

# slice it
highly_correlated_df.loc[correlated_var_names,correlated_var_names]
```

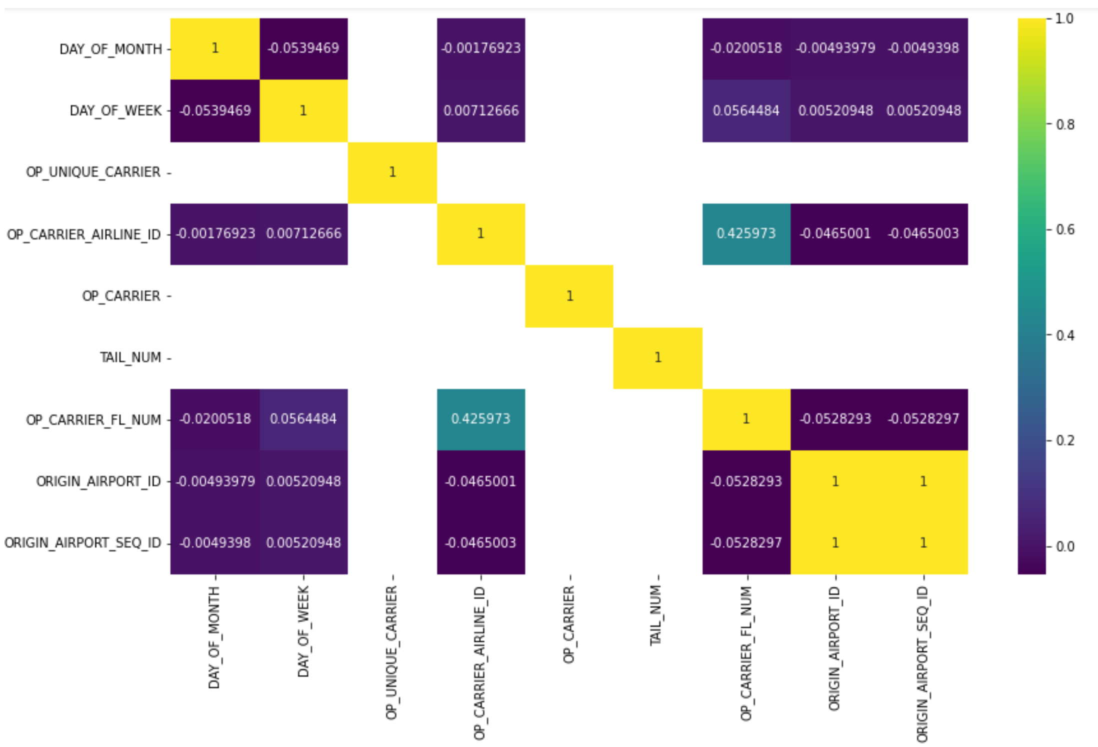
	ORIGIN_AIRPORT_ID	ORIGIN_AIRPORT_SEQ_ID
ORIGIN_AIRPORT_ID	False	True
ORIGIN_AIRPORT_SEQ_ID	True	False

Command took 0.15 seconds -- by 2020.mayuri.yerande@ves.ac.in at 4/5/2023, 5:06:59 PM on My Cluster 5

```python

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14,8))
sns.heatmap(corr_df, annot=True, fmt="g", cmap='viridis')
```

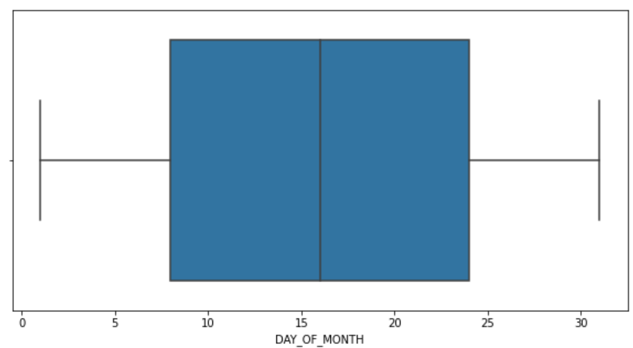


Step 6: Visualization

```
# BOXPLOT
x = df.select('DAY_OF_MONTH').toPandas()
plt.figure(figsize=(10,5))
sns.boxplot('DAY_OF_MONTH',data=x)
```

▸ (1) Spark Jobs

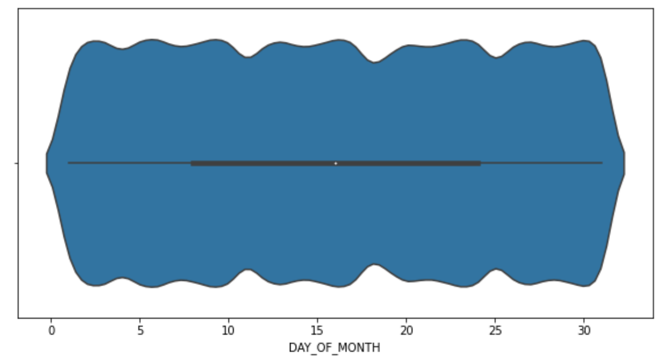
```
/databricks/python/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWar
on 0.12, the only valid positional argument will be `data`, and passing other argur
terpretation.
  warnings.warn(
Out[27]: <AxesSubplot:xlabel='DAY_OF_MONTH'>
```




```
# VIOLIN PLOT
x = df.select('DAY_OF_MONTH').toPandas()
plt.figure(figsize=(10,5))
sns.violinplot('DAY_OF_MONTH',data=x)
```

▸ (1) Spark Jobs

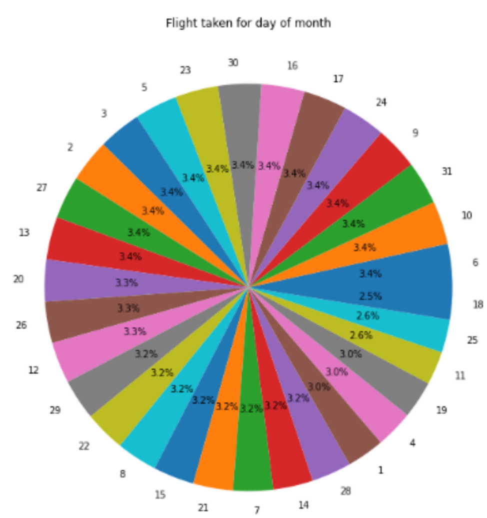
```
Out[23]: <AxesSubplot:xlabel='DAY_OF_MONTH'>
```




```
plot_df = df.toPandas()
labels=plot_df['DAY_OF_MONTH'].value_counts().keys()
size=plot_df['DAY_OF_MONTH'].value_counts()
plt.figure(figsize=(15,10))
plt.pie(size,labels=labels,autopct="%.1f%%",)
plt.title("Flight taken for day of month")
```

▸ (1) Spark Jobs

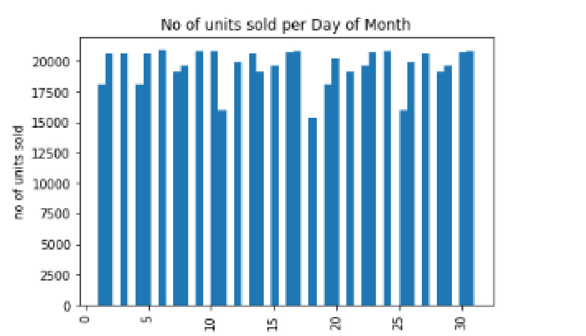
```
Out[42]: Text(0.5, 1.0, 'Flight taken for day of month')
```




```
plot_df = df.toPandas()
plt.hist(plot_df["DAY_OF_MONTH"],bins=50)
plt.xticks(rotation=90)
plt.ylabel("no of units sold")
plt.title("No of units sold per Day of Month")
```

▸ (1) Spark Jobs

```
Out[43]: Text(0.5, 1.0, 'No of units sold per Day of Month')
```



Linear Regression using Apache Spark
Dataset:https://github.com/LeondraJames/Hyundai-Cruise-Ship-Crew-Prediction/blob/master/cruise_ship_info.csv

- Import library

```
from pyspark.sql import SparkSession
```

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

- Create a new SparkSession using the builder pattern.

```
spark = SparkSession.builder.appName('cruise').getOrCreate()
```

- Read the dataset

```
df1 = spark.read.format("csv").option("header",
"true").load("dbfs:/FileStore/shared_uploads/2020.nandana.nair@ves.ac.in/cruise_ship_info.csv",inferSchema=True,
                               header=True)
```

- Printing the schema of dataset

```
df1.printSchema()
```

```
root
 |-- Ship_name: string (nullable = true)
 |-- Cruise_line: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Tonnage: double (nullable = true)
 |-- passengers: double (nullable = true)
 |-- length: double (nullable = true)
 |-- cabins: double (nullable = true)
 |-- passenger_density: double (nullable = true)
 |-- crew: double (nullable = true)
```

- Understanding the contents of dataset

```
1  df1.show()
```

Python

▸ (1) Spark Jobs

```
+-----------+-----------+---+------------------+----------+------+------+-----------------+----+
|  Ship_name|Cruise_line|Age|           Tonnage|passengers|length|cabins|passenger_density|crew|
+-----------+-----------+---+------------------+----------+------+------+-----------------+----+
|    Journey|    Azamara|  6|30.276999999999997|      6.94|  5.94|  3.55|            42.64|3.55|
|      Quest|    Azamara|  6|30.276999999999997|      6.94|  5.94|  3.55|            42.64|3.55|
|Celebration|   Carnival| 26|            47.262|     14.86|  7.22|  7.43|             31.8| 6.7|
|   Conquest|   Carnival| 11|             110.0|     29.74|  9.53| 14.88|            36.99|19.1|
|    Destiny|   Carnival| 17|           101.353|     26.42|  8.92| 13.21|            38.36|10.0|
|    Ecstasy|   Carnival| 22|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|    Elation|   Carnival| 15|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|    Fantasy|   Carnival| 23|            70.367|     20.56|  8.55| 10.22|            34.23| 9.2|
|Fascination|   Carnival| 19|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|    Freedom|   Carnival|  6|110.23899999999999|      37.0|  9.51| 14.87|            29.79|11.5|
|      Glory|   Carnival| 10|             110.0|     29.74|  9.51| 14.87|            36.99|11.6|
|    Holiday|   Carnival| 28|            46.052|     14.52|  7.27|  7.26|            31.72| 6.6|
|Imagination|   Carnival| 18|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|Inspiration|   Carnival| 17|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|     Legend|   Carnival| 11|              86.0|     21.24|  9.63| 10.62|            40.49| 9.3|
|   Liberty*|   Carnival|  8|             110.0|     29.74|  9.51| 14.87|            36.99|11.6|
|    Miracle|   Carnival|  9|              88.5|     21.24|  9.63| 10.62|            41.67|10.3|
|   Paradise|   Carnival| 15|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
```

```
1  df1.describe().show()
```

Python

▸ (2) Spark Jobs

```
+-------+---------+-----------+------------------+------------------+------------------+------------------+------------------+-----
------------+-----------------+
|summary|Ship_name|Cruise_line|               Age|           Tonnage|        passengers|            length|            cabins|passe
nger_density|             crew|
+-------+---------+-----------+------------------+------------------+------------------+------------------+------------------+-----
------------+-----------------+
|  count|      158|        158|               158|               158|               158|               158|               158|
158|              158|
|   mean| Infinity|       null|15.689873417721518| 71.28467088607599|18.45740506329114|8.130632911392404| 8.830000000000005|39.90
094936708861|7.794177215189873|
| stddev|     null|       null| 7.615691058751413|37.229540025907866|9.677094775143416|1.793473548054825|4.4714172221480615| 8.63
921711391542|3.503486564627034|
|    min|Adventure|    Azamara|                 4|             2.329|              0.66|              2.79|              0.33|
17.7|             0.59|
|    max|Zuiderdam|   Windstar|                48|             220.0|              54.0|             11.82|              27.0|
71.43|             21.0|
+-------+---------+-----------+------------------+------------------+------------------+------------------+------------------+-----
------------+-----------------+
```

- Count the number of occurrences of each unique value in this column

```
df1.groupBy('Cruise_line').count().show()
```

▸ (2) Spark Jobs

```
+-----------------+-----+
|      Cruise_line|count|
+-----------------+-----+
|            Costa|   11|
|              P&O|    6|
|           Cunard|    3|
|Regent_Seven_Seas|    5|
|              MSC|    8|
|         Carnival|   22|
|          Crystal|    2|
|           Orient|    1|
|         Princess|   17|
|        Silversea|    4|
|          Seabourn|   3|
|  Holland_American|  14|
|          Windstar|   3|
|            Disney|   2|
|         Norwegian|  13|
|           Oceania|   3|
|           Azamara|   2|
|         Celebrity|  10|
```

- Convert the categorical column "Cruise_line" in the DataFrame df1 to a numerical index column "cruise_cat" in a new DataFrame indexed

```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="Cruise_line", outputCol="cruise_cat")
indexed = indexer.fit(df1).transform(df1)
indexed.head(5)
```

- Printing new dataframe

```
indexed.columns
```

```
Out[54]: ['Ship_name',
 'Cruise_line',
 'Age',
 'Tonnage',
 'passengers',
 'length',
 'cabins',
 'passenger_density',
 'crew',
 'cruise_cat']
```

- Create a VectorAssembler object from the pyspark.ml.feature module that combines multiple input columns in a DataFrame into a single output column.

```
assembler = VectorAssembler(
  inputCols=['Age',
             'Tonnage',
             'passengers',
             'length',
             'cabins',
             'passenger_density',
             'cruise_cat'],
    outputCol="features")
```

```
output = assembler.transform(indexed)
```

```
output.select("features", "crew").show()
```

(1) Spark Jobs

```
+--------------------+----+
|            features|crew|
+--------------------+----+
|[6.0,30.276999999...|3.55|
|[6.0,30.276999999...|3.55|
|[26.0,47.262,14.8...| 6.7|
|[11.0,110.0,29.74...|19.1|
|[17.0,101.353,26....|10.0|
|[22.0,70.367,20.5...| 9.2|
|[15.0,70.367,20.5...| 9.2|
|[23.0,70.367,20.5...| 9.2|
|[19.0,70.367,20.5...| 9.2|
|[6.0,110.23899999...|11.5|
|[10.0,110.0,29.74...|11.6|
|[28.0,46.052,14.5...| 6.6|
|[18.0,70.367,20.5...| 9.2|
|[17.0,70.367,20.5...| 9.2|
|[11.0,86.0,21.24,...| 9.3|
|[8.0,110.0,29.74,...|11.6|
|[9.0,88.5,21.24,9...|10.3|
|[15.0,70.367,20.5...| 9.2|
```

- Select the "features" column and the "crew" column from the output DataFrame

```
final_data = output.select("features", "crew")
```

- Splitting the dataset into train and test dataset

```
train_data,test_data = final_data.randomSplit([0.7,0.3])
```

- Model creation

```
from pyspark.ml.regression import LinearRegression
# Create a Linear Regression Model object
lr = LinearRegression(labelCol='crew')
```

- Fitting the model

```
# Fit the model to the data and call this model lrModel
lrModel = lr.fit(train_data)
```

- Printing coefficients and intercept

```python
# Print the coefficients and intercept for linear regression
print("Coefficients: {} Intercept: {}".format(lrModel.coefficients,lrModel.intercept))
```

```
Coefficients: [-0.007907524274939418,0.010549738042022238,-0.17312350509313318,0.38909053958896234,0.9215589936576951,-0.00709347
1571446334,0.05534458494471553] Intercept: -0.8749465017603469
```

```python
test_results = lrModel.evaluate(test_data)
```

```python
print("RMSE: {}".format(test_results.rootMeanSquaredError))
print("MSE: {}".format(test_results.meanSquaredError))
print("R2: {}".format(test_results.r2))
```

```
RMSE: 0.6588197285198434
MSE: 0.43404343468696016
R2: 0.9641523051695818
```

- Performing correlation

```python
# R2 of 0.86 is pretty good, let's check the data a little closer
from pyspark.sql.functions import corr
```

```python
df1.select(corr('crew','passengers')).show()
```

```
▸ (2) Spark Jobs
+----------------------+
|corr(crew, passengers)|
+----------------------+
|    0.9152341306065384|
+----------------------+

Command took 0.76 seconds -- by 2020.nandana.nair@ves.ac.in at 2/10/2023, 3:54:58 PM on Session
```

Cmd 26

```python
df1.select(corr('crew','cabins')).show()
```

```
▸ (2) Spark Jobs
+------------------+
|corr(crew, cabins)|
+------------------+
|0.9508226063578497|
+------------------+
```

Logistic regression using Apache spark

Dataset:https://github.com/SkalskiP/pySpark_Tutorial/blob/master/Sekcja_12_Logistic_Regression/new_customers.csv

- Importing pyspark

```python
from pyspark.sql import SparkSession
```

```python
spark = SparkSession.builder.appName('logregconsult').getOrCreate()
```

- Printing the schema for the dataset.

```
data.printSchema()
```

```
root
 |-- Names: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- Total_Purchase: double (nullable = true)
 |-- Account_Manager: integer (nullable = true)
 |-- Years: double (nullable = true)
 |-- Num_Sites: double (nullable = true)
 |-- Onboard_date: timestamp (nullable = true)
 |-- Location: string (nullable = true)
 |-- Company: string (nullable = true)
 |-- Churn: integer (nullable = true)
```

- Computes and displays descriptive statistics of a DataFrame

```
data.describe().show()
```

(2) Spark Jobs

```
+-------+-------------+-----------------+-----------------+------------------+-----------------+------------------+-------------
------+-------------------+-------------------+
|summary|        Names|              Age|   Total_Purchase|   Account_Manager|            Years|         Num_Sites|           Lo
cation|            Company|              Churn|
+-------+-------------+-----------------+-----------------+------------------+-----------------+------------------+-------------
------+-------------------+-------------------+
|  count|          900|              900|              900|               900|              900|               900|
900|                900|                900|
|   mean|         null|41.81666666666667|10062.82403333334|0.4811111111111111| 5.27315555555555| 8.587777777777777|
null|               null|0.16666666666666666|
| stddev|         null|6.127560416916251|2408.644531858096|0.4999208935073339|1.274449013194616|1.7648355920350969|
null|               null| 0.3728852122772358|
|    min|   Aaron King|             22.0|            100.0|                 0|              1.0|               3.0|00103 Jeffrey
Cre...|    Abbott-Thompson|                  0|
|    max|Zachary Walsh|             65.0|         18026.01|                 1|             9.15|              14.0|Unit 9800 Box
287...|Zuniga, Clark and...|                  1|
+-------+-------------+-----------------+-----------------+------------------+-----------------+------------------+-------------
------+-------------------+-------------------+
```

```
data.columns
```

```
Out[34]: ['Names',
 'Age',
 'Total_Purchase',
 'Account_Manager',
 'Years',
 'Num_Sites',
 'Onboard_date',
 'Location',
 'Company',
 'Churn']
```

- The VectorAssembler is a transformer in PySpark that is used to combine a given list of columns into a single vector column.

```
from pyspark.ml.feature import VectorAssembler
```

```python
assembler = VectorAssembler(inputCols=['Age',
 'Total_Purchase',
 'Account_Manager',
 'Years',
 'Num_Sites'],outputCol='features')
```

```python
output = assembler.transform(data)
```

output: pyspark.sql.dataframe.DataFrame = [Names: string, Age: double ... 9 more fields]

- This line of code creates a new DataFrame final_data by selecting two columns from the output DataFrame: the features column and the churn column.

```python
final_data = output.select('features','churn')
```

final_data: pyspark.sql.dataframe.DataFrame = [features: udt, churn: integer]

- Splitting the dataset into train and test set

```python
train_churn,test_churn = final_data.randomSplit([0.7,0.3])
```

train_churn: pyspark.sql.dataframe.DataFrame = [features: udt, churn: integer]
test_churn: pyspark.sql.dataframe.DataFrame = [features: udt, churn: integer]

- Importing LogisticRegression library

```python
from pyspark.ml.classification import LogisticRegression
```

- This line of code initializes a new LogisticRegression estimator in PySpark with the label column set to churn.

```python
lr_churn = LogisticRegression(labelCol='churn')
```

- Fitting the train set in the model.

```python
fitted_churn_model = lr_churn.fit(train_churn)
```

```python
training_sum = fitted_churn_model.summary
```

```
1   training_sum.predictions.describe().show()
```

▸ (2) Spark Jobs

```
+-------+-------------------+-------------------+
|summary|              churn|         prediction|
+-------+-------------------+-------------------+
|  count|                630|                630|
|   mean| 0.173015873015873|0.1523809523809524|
| stddev|0.37856156048067147|0.3596753273871411|
|    min|                0.0|                0.0|
|    max|                1.0|                1.0|
+-------+-------------------+-------------------+
```

```
1   from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
1   pred_and_labels = fitted_churn_model.evaluate(test_churn)
```

- Display the predictions made by the Logistic Regression model.

```
1   pred_and_labels.predictions.show()
```

Python

▸ (1) Spark Jobs

```
+--------------------+-----+--------------------+--------------------+----------+
|            features|churn|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[26.0,8787.39,1.0...|    1|[-0.2029268544860...|[0.44944166386318...|       1.0|
|[27.0,8628.8,1.0,...|    0|[5.64245690587166...|[0.99646836658134...|       0.0|
|[28.0,8670.98,0.0...|    0|[8.50981099862742...|[0.99979855868510...|       0.0|
|[28.0,9090.43,1.0...|    0|[1.02136548408360...|[0.73523849385903...|       0.0|
|[29.0,12711.15,0....|    0|[5.91023747386637...|[0.99729578969188...|       0.0|
|[29.0,13240.01,1....|    0|[6.97159023565538...|[0.99906271949514...|       0.0|
|[30.0,6744.87,0.0...|    0|[3.74110328125562...|[0.97682205409108...|       0.0|
|[30.0,10183.98,1....|    0|[2.59020233148786...|[0.93022835030179...|       0.0|
|[31.0,8829.83,1.0...|    0|[4.40191565403454...|[0.98789449567262...|       0.0|
|[31.0,10058.87,1....|    0|[4.85040392638843...|[0.99223554245866...|       0.0|
|[32.0,7896.65,0.0...|    0|[4.15406075598772...|[0.98454216537319...|       0.0|
|[32.0,10716.75,0....|    0|[4.81123885322278...|[0.99192791682664...|       0.0|
|[32.0,13630.93,0....|    0|[1.99417782817638...|[0.88018443073451...|       0.0|
|[33.0,8556.73,0.0...|    0|[3.91524217348165...|[0.98045394453372...|       0.0|
|[33.0,10306.21,1....|    0|[1.94600468809309...|[0.87501033984067...|       0.0|
|[33.0,13157.08,1....|    0|[0.84511283591903...|[0.69954094444479...|       0.0|
|[34.0,5447.16,1.0...|    0|[3.31603538178110...|[0.96497484046300...|       0.0|
|[34.0,6131.92,0.0...|    0|[4.10531645935957...|[0.98378254035572...|       0.0|
```

<u>CONCLUSION:</u> Thus we performed Visualization using Power BI and used its various features. We performed EDA,logistic and linear regression on Apache Spark and did some visualization. Thus we successfully learnt about power BI and Apache Spark