Netaji Subhas University of Technology

A STATE UNIVERSITY UNDER DELHI ACT 06 OF 2018, GOVT. OF NCT OF DELHI

Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi-110078

LABORATORY FILE

Computer Hardware and Software

Submitted By:

Name: Amogh Garg

Roll Number: 2020UCO1688

Branch: COE

Section: 3

INDEX

EXPERIMENT-1

Create a Car Parking Prediction Model

Description:

In this, you need to find the Car Park Distance Prediction for Smart Vehicles. Most of the parking lots around the world make sure that the cars are parked in parallel. However, many people face difficulties when it comes to parallel parking, especially due to various assumptions one has to make regarding spatial arrangements. A machine learning algorithm needs to be implemented to calculate whether the car would be able to successfully park in the cavity or not, based on the distances measured by the sensor.

Approach:

Since there is no relevant dataset on the internet, we have to create our own synthetic dataset. The synthetic dataset contains 3 columns. The description of features is as followed:

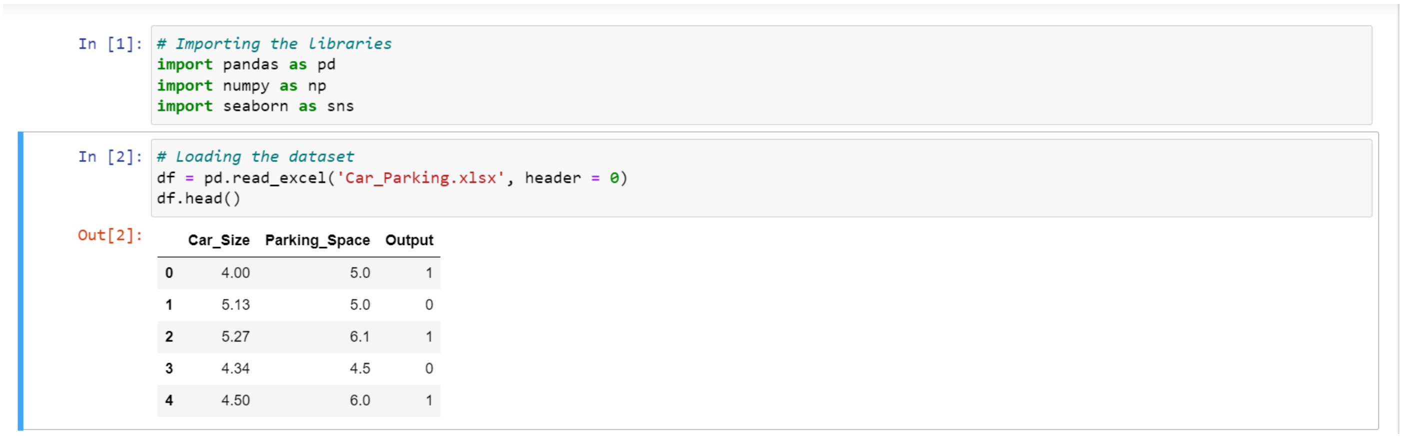
1. `Car_Size` : The length of the car in meters.
2. `Parking_Space` : The space available (in meters) for the car to be parked as detected by the sensor in the smart car.
3. `Output` : This is the output feature (binary). `0` indicates that car can not be parked in the available space. `1` indicates that car can be parked in the given space.

   More relevant features can also be added in the dataset like whether the car supports power-cutting ability and other relevant specifications of the car in order to make accurate predictions. But for now to keep the model simple, we will be considering only `Car_Size` and `Parking_Space` as input features.

We will be creating a classifier from scratch using *Tensorflow* (Neural Networks using Keras) and then use *TensorFlow Lite* to convert the normal ML model into a Tiny ML Model (compressed form) which can then be embedded on devices with less memory like micro-controllers or egde devices. In our case this Tiny ML Model can be loaded into the micro-controller in the smart car which takes the input from the sensors installed in the car, processes the input and generates the relevant output

DATASET : https://docs.google.com/spreadsheets/d/1rFzDI75nt0BiyPX1qJxdxVz6ZSkS-weV/edit?usp=share_link&ouid=103681875594479144076&rtpof=true&sd=true

CODE:



In [1]:
```python
# Importing the Libraries
import pandas as pd
import numpy as np
import seaborn as sns
```

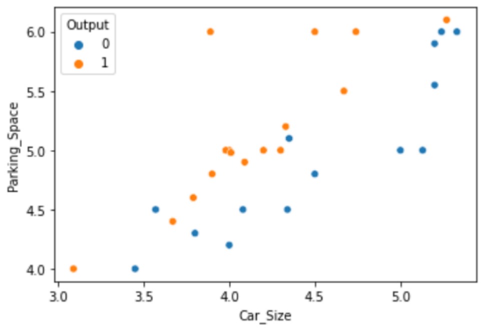
In [2]:
```python
# Loading the dataset
df = pd.read_excel('Car_Parking.xlsx', header = 0)
df.head()
```

Out[2]:

	Car_Size	Parking_Space	Output
0	4.00	5.0	1
1	5.13	5.0	0
2	5.27	6.1	1
3	4.34	4.5	0
4	4.50	6.0	1

In [3]:
```
# Plotting the scatter plot of Car_Size vs Parking_Space
sns.scatterplot(x = df['Car_Size'], y = df['Parking_Space'], hue = 'Output', data = df)
```

Out[3]:
```
<AxesSubplot:xlabel='Car_Size', ylabel='Parking_Space'>
```



In [4]:
```
df.shape
```

Out[4]:
```
(30, 3)
```

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

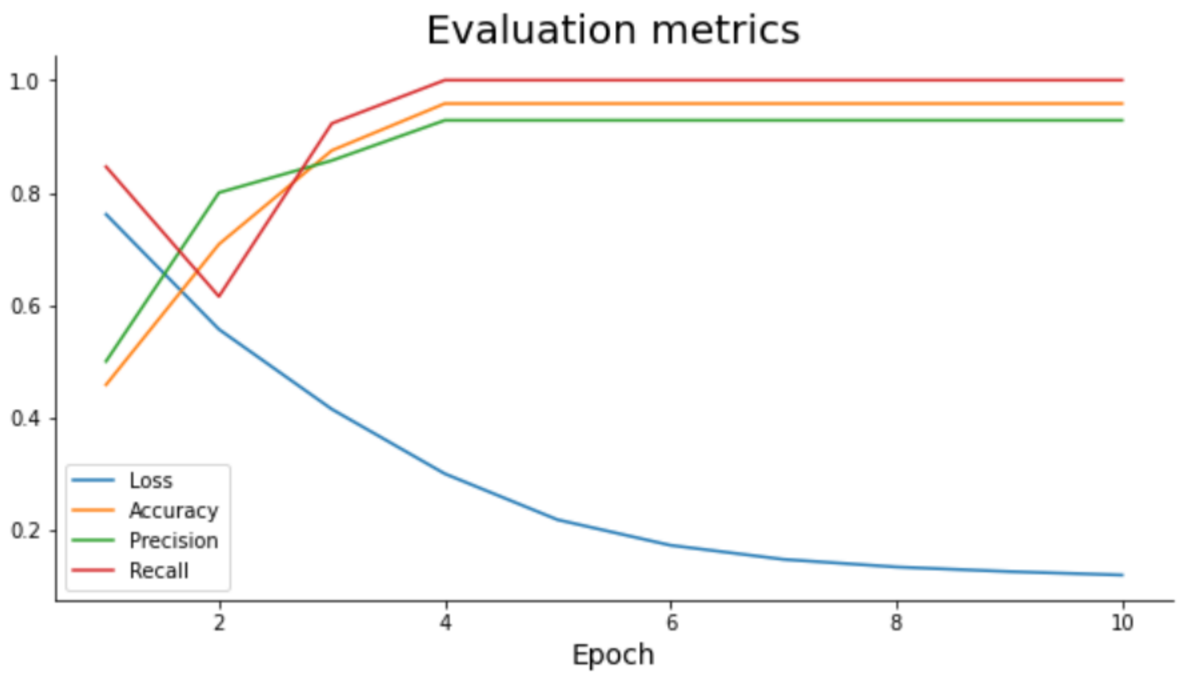
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Size       30 non-null     float64
 1   Parking_Space  30 non-null     float64
 2   Output         30 non-null     int64  
dtypes: float64(2), int64(1)
memory usage: 848.0 bytes
```

In [6]:
```
# Loading the x AND y features
x = df.drop('Output', axis = 1)
y = df['Output']
```

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [8]:
```
import tensorflow as tf
tf.random.set_seed(42)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')])

model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(lr=0.05),
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
)
#We kept track of loss, accuracy, precision, and recall during training, and saved them to history.
#We can now visualize these metrics to get a sense of how the model is doing
history = model.fit(X_train_scaled, y_train, epochs=10)
```

In [9]:
```python
import matplotlib.pyplot as plt
from matplotlib import rcParams

rcParams['figure.figsize'] = (10,5)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
plt.plot(
    np.arange(1, 11),
    history.history['loss'], label='Loss'
)
plt.plot(
    np.arange(1, 11),
    history.history['accuracy'], label='Accuracy'
)
plt.plot(
    np.arange(1, 11),
    history.history['precision'], label='Precision'
)
plt.plot(
    np.arange(1, 11),
    history.history['recall'], label='Recall'
)
plt.title('Evaluation metrics', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```



In [10]:
```python
# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('car_model.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
INFO:tensorflow:Assets written to: C:\Users\AMOGH GARG\AppData\Local\Temp\tmp935heenk\assets
```

The model can further be improved if we are able to provide as input a more comprehensive dataset. Also, more layers can be added to neural-network, but since the dataset is small as of now we are not adding more layers to avoid over-fitting.

`car_model.tflite` can be loaded and used for inference with the help of following documentation: Inference Guide

Inference refers to the process of executing a TensorFlow Lite model on-device to make predictions based on input data.

EXPERIMENT-2

DESCRIPTION: Data Visualization and Analytics using R Programming.

DATASET: https://drive.google.com/file/d/1C5kJ_FPTuFTvDF-xTbJ8h4lITCsNFVR9/view?usp=share_link

CODE:

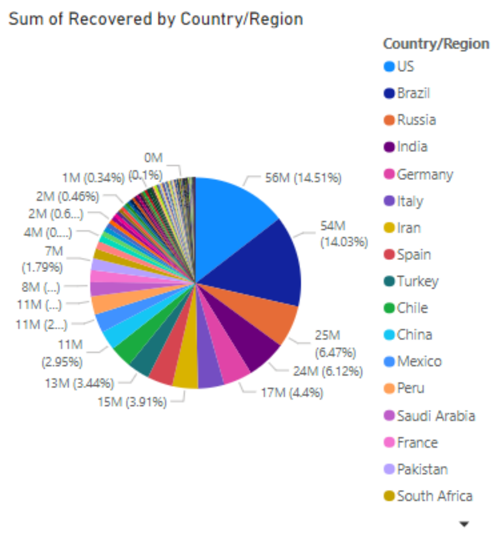
```
# Importing the dataset
dataset = read.csv('Data.csv')

# Importing the dataset
dataset = read.csv('Data.csv')

# Taking care of missing data
dataset$Age = ifelse(is.na(dataset$Age),
                     ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)),
                     dataset$Age)
dataset$Salary = ifelse(is.na(dataset$Salary),
                        ave(dataset$Salary, FUN = function(x) mean(x, na.rm = TRUE)),
                        dataset$Salary)

# Importing the dataset
dataset = read.csv('Data.csv')

# Taking care of missing data
dataset$Age = ifelse(is.na(dataset$Age),
                     ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)),
                     dataset$Age)
dataset$Salary = ifelse(is.na(dataset$Salary),
                        ave(dataset$Salary, FUN = function(x) mean(x, na.rm = TRUE)),
                        dataset$Salary)

# Encoding categorical data
dataset$Country = factor(dataset$Country,
                         levels = c('France', 'Spain', 'Germany'),
                         labels = c(1, 2, 3))
dataset$Purchased = factor(dataset$Purchased,
                           levels = c('No', 'Yes'),
                           labels = c(0, 1))

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$DependentVariable, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
training_set = scale(training_set)
test_set = scale(test_set)
```

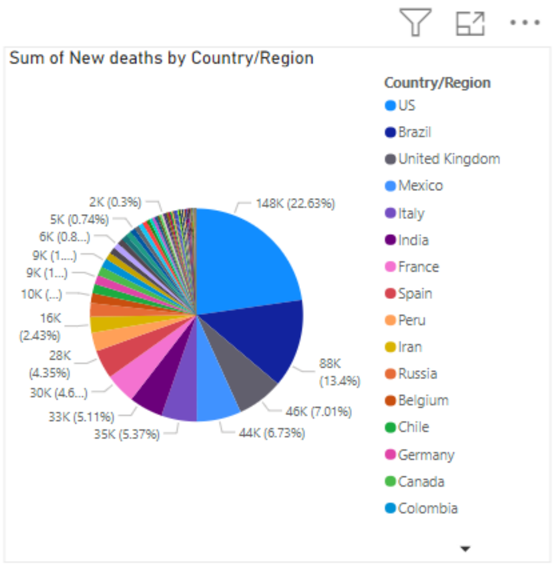
EXPERIMENT-3

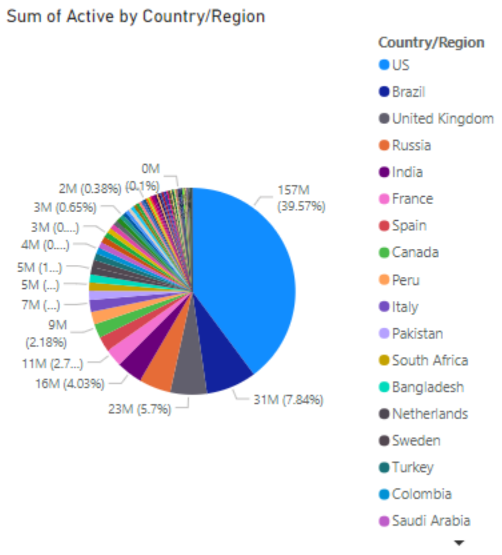
DESCRIPTION: Data Visualization using PowerBI.

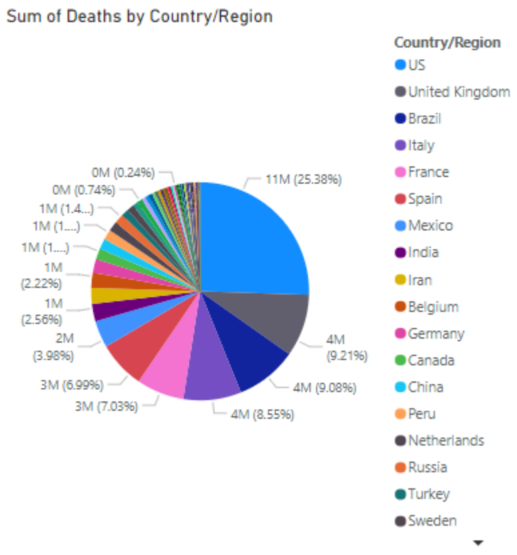
DATASET: https://www.kaggle.com/datasets/imdevskp/corona-virus-report?select=country_wise_latest.csv

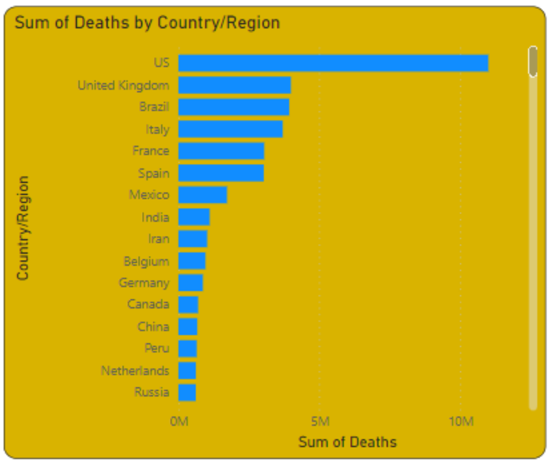
POWER BI REPORT:

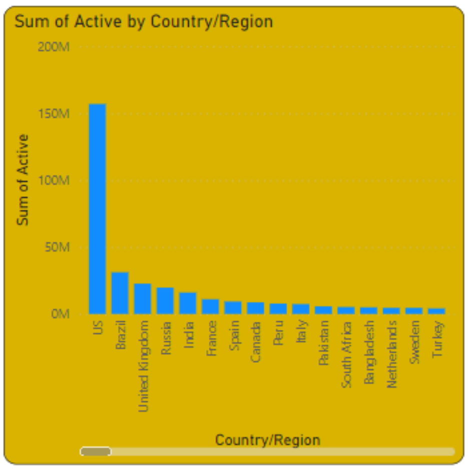


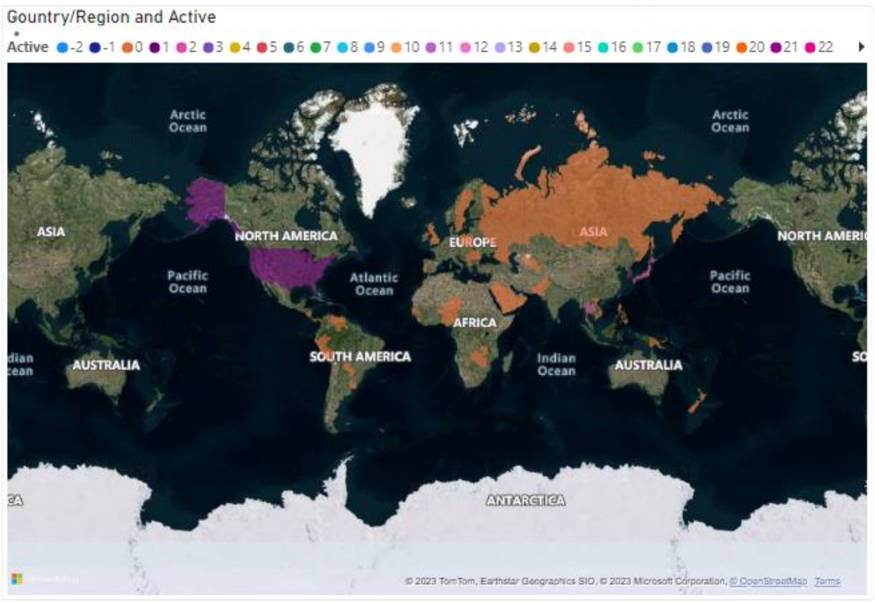


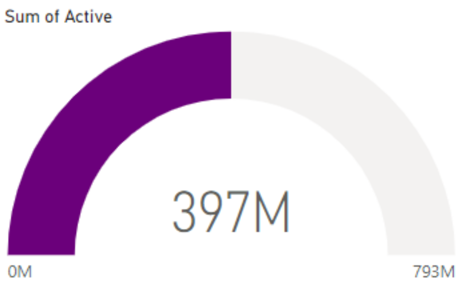


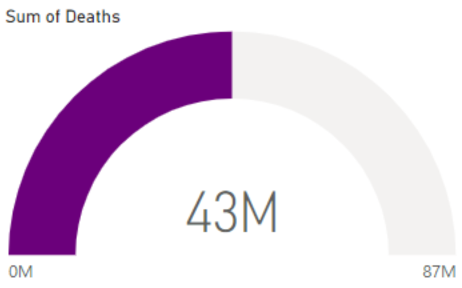



Sum of Deaths by Country/Region
Country/Region
US
United Kingdom
Brazil
Italy
France
Spain
Mexico
India
Iran
Belgium
Germany
Canada
China
Peru
Netherlands
Russia
0M
5M
10M
Sum of Deaths


Sum of Active by Country/Region
200M
150M
100M
50M
0M
Sum of Active
US
Brazil
United Kingdom
Russia
India
France
Spain
Canada
Peru
Italy
Pakistan
South Africa
Bangladesh
Netherlands
Sweden
Turkey
Country/Region

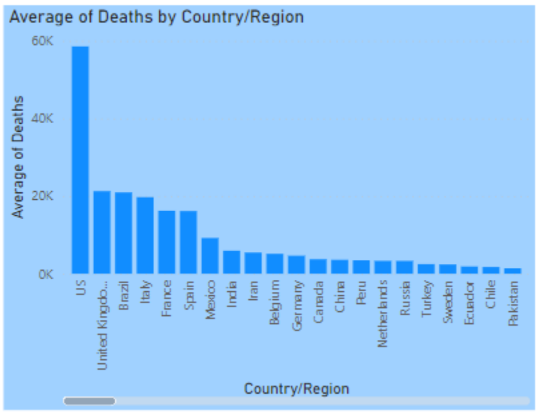

Country/Region and Active
Active ● -2 ● -1 ● 0 ● 1 ● 2 ● 3 ● 4 ● 5 ● 6 ● 7 ● 8 ● 9 ● 10 ● 11 ● 12 ● 13 ● 14 ● 15 ● 16 ● 17 ● 18 ● 19 ● 20 ● 21 ● 22
Arctic Ocean
ASIA
NORTH AMERICA
Pacific Ocean
Atlantic Ocean
EUROPE
AFRICA
SOUTH AMERICA
AUSTRALIA
Indian Ocean
ANTARCTICA
© 2023 TomTom, Earthstar Geographics SIO, © 2023 Microsoft Corporation, © OpenStreetMap Terms

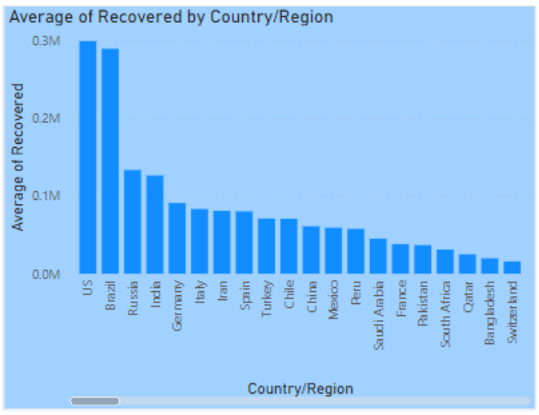

Sum of Active
397M
0M
793M


Sum of Deaths
43M
0M
87M

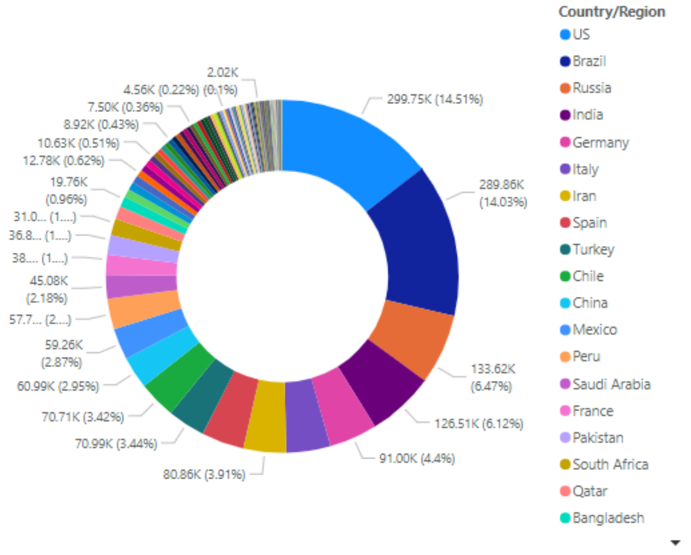
Sum of Deaths by WHO Region

Arctic Ocean
ASIA
NORTH AMERICA
EUROPE
ASIA
Pacific Ocean
Atlantic Ocean
AFRICA
SOUTH AMERICA
Indian Ocean
AUSTRALIA
ANTARCTICA
© 2023 TomTom, © 2023 Microsoft Corporation, © OpenStreetMap Terms





Average of Recovered by Country/Region



EXPERIMENT-4

DESCRIPTION: Create and manipulate database in spark
- Install Spark in Google Collaboratory.
- Use RDD
- Use Dataframe
- Implement linear regression using sparkMlib

CODE AND OUTPUT:

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-
hadoop3.2.tgz
!tar xf spark-3.1.1-bin-hadoop3.2.tgz
!pip install -q findspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.1-bin-hadoop3.2"

import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True) # Property used to format
output tables better
spark

# Load data from csv to a dataframe.
# header=True means the first row is a header
# sep=';' means the column are seperated using ''
df = spark.read.csv('cars.csv', header=True, sep=";")
df.show(5)
```

```
+--------------------+----+---------+------------+----------+------+------------+--
---+-------+
|                 Car|
MPG|Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|
+--------------------+----+---------+------------+----------+------+------------+--
---+-------+
|Chevrolet Chevell...|18.0|        8|       307.0|     130.0| 3504.|        12.0|
70|     US|
|   Buick Skylark 320|15.0|        8|       350.0|     165.0| 3693.|        11.5|
70|     US|
|  Plymouth Satellite|18.0|        8|       318.0|     150.0| 3436.|        11.0|
70|     US|
|       AMC Rebel SST|16.0|        8|       304.0|     150.0| 3433.|        12.0|
70|     US|
|         Ford Torino|17.0|        8|       302.0|     140.0| 3449.|        10.5|
70|     US|
+--------------------+----+---------+------------+----------+------+------------+--
---+-------+
only showing top 5 rows
```

```
df = spark.read.csv('cars.csv', header=True, sep=";", inferSchema=True)
df.printSchema()
```

```
root
 |-- Car: string (nullable = true)
 |-- MPG: double (nullable = true)
 |-- Cylinders: integer (nullable = true)
 |-- Displacement: double (nullable = true)
 |-- Horsepower: double (nullable = true)
 |-- Weight: decimal(4,0) (nullable = true)
 |-- Acceleration: double (nullable = true)
 |-- Model: integer (nullable = true)
 |-- Origin: string (nullable = true)
```

```
cars = spark.sparkContext.textFile('cars.csv')
print(cars.first())
cars_header = cars.first()
cars_rest = cars.filter(lambda line: line!=cars_header)
print(cars_rest.first())
```

```
Car;MPG;Cylinders;Displacement;Horsepower;Weight;Acceleration;Model;Origin
Chevrolet Chevelle Malibu;18.0;8;307.0;130.0;3504.;12.0;70;US
```

```
cars_rest.map(lambda line: line.split(";")).count()
406
```

```
# Car name is column  0
(cars_rest.filter(lambda line: line.split(";")[8]=='Europe').
 map(lambda line: (line.split(";")[0],
    line.split(";")[1],
    line.split(";")[2],
    line.split(";")[5],
    line.split(";")[8])).collect())
```

```
[('Citroen DS-21 Pallas', '0', '4', '3090.', 'Europe'), ('Volkswagen 1131 Deluxe
Sedan', '26.0', '4', '1835.', 'Europe'), ('Peugeot 504', '25.0', '4', '2672.',
'Europe'), ('Audi 100 LS', '24.0', '4', '2430.', 'Europe'), ('Saab 99e', '25.0',
'4', '2375.', 'Europe'), ('BMW 2002', '26.0', '4', '2234.', 'Europe'), ('Volkswagen
Super Beetle 117', '0', '4', '1978.', 'Europe'), ('Opel 1900', '28.0', '4',
'2123.', 'Europe'), ('Peugeot 304', '30.0', '4', '2074.', 'Europe'), ('Fiat 124B',
'30.0', '4', '2065.', 'Europe'), ('Volkswagen Model 111', '27.0', '4', '1834.',
'Europe'), ('Volkswagen Type 3', '23.0', '4', '2254.', 'Europe'), ('Volvo 145e
(sw)', '18.0', '4', '2933.', 'Europe'), ('Volkswagen 411 (sw)', '22.0', '4',
'2511.', 'Europe'), ('Peugeot 504 (sw)', '21.0', '4', '2979.', 'Europe'), ('Renault
12 (sw)', '26.0', '4', '2189.', 'Europe'), ('Volkswagen Super Beetle', '26.0', '4',
'1950.', 'Europe'), ('Fiat 124 Sport Coupe', '26.0', '4', '2265.', 'Europe'),
('Fiat 128', '29.0', '4', '1867.', 'Europe'), ('Opel Manta', '24.0', '4', '2158.',
'Europe'), ('Audi 100LS', '20.0', '4', '2582.', 'Europe'), ('Volvo 144ea', '19.0',
'4', '2868.', 'Europe'), ('Saab 99le', '24.0', '4', '2660.', 'Europe'), ('Audi
Fox', '29.0', '4', '2219.', 'Europe'), ('Volkswagen Dasher', '26.0', '4', '1963.',
'Europe'), ('Opel Manta', '26.0', '4', '2300.', 'Europe'), ('Fiat 128', '24.0',
'4', '2108.', 'Europe'), ('Fiat 124 TC', '26.0', '4', '2246.', 'Europe'), ('Fiat
x1.9', '31.0', '4', '2000.', 'Europe'), ('Volkswagen Dasher', '25.0', '4', '2223.',
'Europe'), ('Volkswagen Rabbit', '29.0', '4', '1937.', 'Europe'), ('Audi 100LS',
'23.0', '4', '2694.', 'Europe'), ('Peugeot 504', '23.0', '4', '2957.', 'Europe'),
('Volvo 244DL', '22.0', '4', '2945.', 'Europe'), ('Saab 99LE', '25.0', '4',
'2671.', 'Europe'), ('Fiat 131', '28.0', '4', '2464.', 'Europe'), ('Opel 1900',
'25.0', '4', '2220.', 'Europe'), ('Renault 12tl', '27.0', '4', '2202.', 'Europe'),
('Volkswagen Rabbit', '29.0', '4', '1937.', 'Europe'), ('Volkswagen Rabbit',
'29.5', '4', '1825.', 'Europe'), ('Volvo 245', '20.0', '4', '3150.', 'Europe'),
('Peugeot 504', '19.0', '4', '3270.', 'Europe'), ('Mercedes-Benz 280s', '16.5',
'6', '3820.', 'Europe'), ('Renault 5 GTL', '36.0', '4', '1825.', 'Europe'),
('Volkswagen Rabbit Custom', '29.0', '4', '1940.', 'Europe'), ('Volkswagen Dasher',
'30.5', '4', '2190.', 'Europe'), ('BMW 320i', '21.5', '4', '2600.', 'Europe'),
('Volkswagen Rabbit Custom Diesel', '43.1', '4', '1985.', 'Europe'), ('Audi 5000',
'20.3', '5', '2830.', 'Europe'), ('Volvo 264gl', '17.0', '6', '3140.', 'Europe'),
('Saab 99gle', '21.6', '4', '2795.', 'Europe'), ('Peugeot 604sl', '16.2', '6',
'3410.', 'Europe'), ('Volkswagen Scirocco', '31.5', '4', '1990.', 'Europe'),
('Volkswagen Rabbit Custom', '31.9', '4', '1925.', 'Europe'), ('Mercedes Benz
300d', '25.4', '5', '3530.', 'Europe'), ('Peugeot 504', '27.2', '4', '3190.',
'Europe'), ('Fiat Strada Custom', '37.3', '4', '2130.', 'Europe'), ('Volkswagen
Rabbit', '41.5', '4', '2144.', 'Europe'), ('Audi 4000', '34.3', '4', '2188.',
'Europe'), ('Volkswagen Rabbit C (Diesel)', '44.3', '4', '2085.', 'Europe'),
('Volkswagen Dasher (diesel)', '43.4', '4', '2335.', 'Europe'), ('Audi 5000s
(diesel)', '36.4', '5', '2950.', 'Europe'), ('Mercedes-Benz 240d', '30.0', '4',
'3250.', 'Europe'), ('Renault Lecar Deluxe', '40.9', '4', '1835.', 'Europe'),
('Volkswagen Rabbit', '29.8', '4', '1845.', 'Europe'), ('Triumph TR7 Coupe',
'35.0', '4', '2500.', 'Europe'), ('Volkswagen Jetta', '33.0', '4', '2190.',
'Europe'), ('Renault 18i', '34.5', '4', '2320.', 'Europe'), ('Peugeot 505s Turbo
Diesel', '28.1', '4', '3230.', 'Europe'), ('Saab 900s', '0', '4', '2800.',
'Europe'), ('Volvo Diesel', '30.7', '6', '3160.', 'Europe'), ('Volkswagen Rabbit
l', '36.0', '4', '1980.', 'Europe'), ('Volkswagen Pickup', '44.0', '4', '2130.',
'Europe')]
```

```
(cars_rest.filter(lambda line: line.split(";")[8] in ['Europe','Japan']).
 map(lambda line: (line.split(";")[0],
    line.split(";")[1],
    line.split(";")[2],
    line.split(";")[5],
    line.split(";")[8])).collect())
```

```
[('Citroen DS-21 Pallas', '0', '4', '3090.', 'Europe'), ('Toyota Corolla Mark ii',
'24.0', '4', '2372.', 'Japan'), ('Datsun PL510', '27.0', '4', '2130.', 'Japan'),
('Volkswagen 1131 Deluxe Sedan', '26.0', '4', '1835.', 'Europe'), ('Peugeot 504',
'25.0', '4', '2672.', 'Europe'), ('Audi 100 LS', '24.0', '4', '2430.', 'Europe'),
('Saab 99e', '25.0', '4', '2375.', 'Europe'), ('BMW 2002', '26.0', '4', '2234.',
'Europe'), ('Datsun PL510', '27.0', '4', '2130.', 'Japan'), ('Toyota Corolla',
'25.0', '4', '2228.', 'Japan'), ('Volkswagen Super Beetle 117', '0', '4', '1978.',
'Europe'), ('Opel 1900', '28.0', '4', '2123.', 'Europe'), ('Peugeot 304', '30.0',
'4', '2074.', 'Europe'), ('Fiat 124B', '30.0', '4', '2065.', 'Europe'), ('Toyota
Corolla 1200', '31.0', '4', '1773.', 'Japan'), ('Datsun 1200', '35.0', '4',
'1613.', 'Japan'), ('Volkswagen Model 111', '27.0', '4', '1834.', 'Europe'),
('Toyota Corolla Hardtop', '24.0', '4', '2278.', 'Japan'), ('Volkswagen Type 3',
'23.0', '4', '2254.', 'Europe'), ('Mazda RX2 Coupe', '19.0', '3', '2330.',
'Japan'), ('Volvo 145e (sw)', '18.0', '4', '2933.', 'Europe'), ('Volkswagen 411
(sw)', '22.0', '4', '2511.', 'Europe'), ('Peugeot 504 (sw)', '21.0', '4', '2979.',
'Europe'), ('Renault 12 (sw)', '26.0', '4', '2189.', 'Europe'), ('Datsun 510 (sw)',
'28.0', '4', '2288.', 'Japan'), ('Toyota Corolla Mark II (sw)', '23.0', '4',
'2506.', 'Japan'), ('Toyota Corolla 1600 (sw)', '27.0', '4', '2100.', 'Japan'),
('Volkswagen Super Beetle', '26.0', '4', '1950.', 'Europe'), ('Toyota Camry',
'20.0', '4', '2279.', 'Japan'), ('Datsun 610', '22.0', '4', '2379.', 'Japan'),
('Mazda RX3', '18.0', '3', '2124.', 'Japan'), ('Fiat 124 Sport Coupe', '26.0', '4',
'2265.', 'Europe'), ('Fiat 128', '29.0', '4', '1867.', 'Europe'), ('Opel Manta',
'24.0', '4', '2158.', 'Europe'), ('Audi 100LS', '20.0', '4', '2582.', 'Europe'),
('Volvo 144ea', '19.0', '4', '2868.', 'Europe'), ('Saab 99le', '24.0', '4',
'2660.', 'Europe'), ('Toyota Mark II', '20.0', '6', '2807.', 'Japan'), ('Datsun
B210', '31.0', '4', '1950.', 'Japan'), ('Toyota Corolla 1200', '32.0', '4',
'1836.', 'Japan'), ('Audi Fox', '29.0', '4', '2219.', 'Europe'), ('Volkswagen
Dasher', '26.0', '4', '1963.', 'Europe'), ('Opel Manta', '26.0', '4', '2300.',
'Europe'), ('Toyota Corolla', '31.0', '4', '1649.', 'Japan'), ('Datsun 710',
'32.0', '4', '2003.', 'Japan'), ('Fiat 128', '24.0', '4', '2108.', 'Europe'),
('Fiat 124 TC', '26.0', '4', '2246.', 'Europe'), ('Honda Civic', '24.0', '4',
'2489.', 'Japan'), ('Subaru', '26.0', '4', '2391.', 'Japan'), ('Fiat x1.9', '31.0',
'4', '2000.', 'Europe'), ('Toyota Corolla', '29.0', '4', '2171.', 'Japan'),
('Toyota Corolla', '24.0', '4', '2702.', 'Japan'), ('Volkswagen Dasher', '25.0',
'4', '2223.', 'Europe'), ('Datsun 710', '24.0', '4', '2545.', 'Japan'),
('Volkswagen Rabbit', '29.0', '4', '1937.', 'Europe'), ('Audi 100LS', '23.0', '4',
'2694.', 'Europe'), ('Peugeot 504', '23.0', '4', '2957.', 'Europe'), ('Volvo
244DL', '22.0', '4', '2945.', 'Europe'), ('Saab 99LE', '25.0', '4', '2671.',
'Europe'), ('Honda Civic CVCC', '33.0', '4', '1795.', 'Japan'), ('Fiat 131',
'28.0', '4', '2464.', 'Europe'), ('Opel 1900', '25.0', '4', '2220.', 'Europe'),
('Renault 12tl', '27.0', '4', '2202.', 'Europe'), ('Volkswagen Rabbit', '29.0',
'4', '1937.', 'Europe'), ('Honda Civic', '33.0', '4', '1795.', 'Japan'),
('Volkswagen Rabbit', '29.5', '4', '1825.', 'Europe'), ('Datsun B-210', '32.0',
'4', '1990.', 'Japan'), ('Toyota Corolla', '28.0', '4', '2155.', 'Japan'), ('Volvo
245', '20.0', '4', '3150.', 'Europe'), ('Peugeot 504', '19.0', '4', '3270.',
'Europe'), ('Toyota Mark II', '19.0', '6', '2930.', 'Japan'), ('Mercedes-Benz
280s', '16.5', '6', '3820.', 'Europe'), ('Honda Accord CVCC', '31.5', '4', '2045.',
'Japan'), ('Renault 5 GTL', '36.0', '4', '1825.', 'Europe'), ('Datsun F-10
Hatchback', '33.5', '4', '1945.', 'Japan'), ('Volkswagen Rabbit Custom', '29.0',
'4', '1940.', 'Europe'), ('Toyota Corolla Liftback', '26.0', '4', '2265.',
'Japan'), ('Subaru DL', '30.0', '4', '1985.', 'Japan'), ('Volkswagen Dasher',
'30.5', '4', '2190.', 'Europe'), ('Datsun 810', '22.0', '6', '2815.', 'Japan'),
('BMW 320i', '21.5', '4', '2600.', 'Europe'), ('Mazda RX-4', '21.5', '3', '2720.',
'Japan'), ('Volkswagen Rabbit Custom Diesel', '43.1', '4', '1985.', 'Europe'),
('Mazda GLC Deluxe', '32.8', '4', '1985.', 'Japan'), ('Datsun B210 GX', '39.4',
'4', '2070.', 'Japan'), ('Honda Civic CVCC', '36.1', '4', '1800.', 'Japan'),
('Toyota Corolla', '27.5', '4', '2560.', 'Japan'), ('Datsun 510', '27.2', '4',
'2300.', 'Japan'), ('Toyota Celica GT Liftback', '21.1', '4', '2515.', 'Japan'),
('Datsun 200-SX', '23.9', '4', '2405.', 'Japan'), ('Audi 5000', '20.3', '5',
'2830.', 'Europe'), ('Volvo 264gl', '17.0', '6', '3140.', 'Europe'), ('Saab 99gle',
'21.6', '4', '2795.', 'Europe'), ('Peugeot 604sl', '16.2', '6', '3410.', 'Europe'),
('Volkswagen Scirocco', '31.5', '4', '1990.', 'Europe'), ('Honda Accord LX',
'29.5', '4', '2135.', 'Japan'), ('Volkswagen Rabbit Custom', '31.9', '4', '1925.',
'Europe'), ('Mazda GLC Deluxe', '34.1', '4', '1975.', 'Japan'), ('Mercedes Benz
300d', '25.4', '5', '3530.', 'Europe'), ('Peugeot 504', '27.2', '4', '3190.',
'Europe'), ('Datsun 210', '31.8', '4', '2020.', 'Japan'), ('Fiat Strada Custom',
'37.3', '4', '2130.', 'Europe'), ('Volkswagen Rabbit', '41.5', '4', '2144.',
'Europe'), ('Toyota Corolla Tercel', '38.1', '4', '1968.', 'Japan'), ('Datsun 310',
'37.2', '4', '2019.', 'Japan'), ('Audi 4000', '34.3', '4', '2188.', 'Europe'),
('Toyota Corolla Liftback', '29.8', '4', '2711.', 'Japan'), ('Mazda 626', '31.3',
'4', '2542.', 'Japan'), ('Datsun 510 Hatchback', '37.0', '4', '2434.', 'Japan'),
('Toyota Corolla', '32.2', '4', '2265.', 'Japan'), ('Mazda GLC', '46.6', '4',
'2110.', 'Japan'), ('Datsun 210', '40.8', '4', '2110.', 'Japan'), ('Volkswagen
Rabbit C (Diesel)', '44.3', '4', '2085.', 'Europe'), ('Volkswagen Dasher (diesel)',
'43.4', '4', '2335.', 'Europe'), ('Audi 5000s (diesel)', '36.4', '5', '2950.',
'Europe'), ('Mercedes-Benz 240d', '30.0', '4', '3250.', 'Europe'), ('Honda Civic
1500 gl', '44.6', '4', '1850.', 'Japan'), ('Renault Lecar Deluxe', '40.9', '4',
'1835.', 'Europe'), ('Subaru DL', '33.8', '4', '2145.', 'Japan'), ('Volkswagen
Rabbit', '29.8', '4', '1845.', 'Europe'), ('Datsun 280-ZX', '32.7', '6', '2910.',
'Japan'), ('Mazda RX-7 GS', '23.7', '3', '2420.', 'Japan'), ('Triumph TR7 Coupe',
'35.0', '4', '2500.', 'Europe'), ('Honda Accord', '32.4', '4', '2290.', 'Japan'),
('Toyota Starlet', '39.1', '4', '1755.', 'Japan'), ('Honda Civic 1300', '35.1',
'4', '1760.', 'Japan'), ('Subaru', '32.3', '4', '2065.', 'Japan'), ('Datsun 210
MPG', '37.0', '4', '1975.', 'Japan'), ('Toyota Tercel', '37.7', '4', '2050.',
'Japan'), ('Mazda GLC 4', '34.1', '4', '1985.', 'Japan'), ('Volkswagen Jetta',
'33.0', '4', '2190.', 'Europe'), ('Renault 18i', '34.5', '4', '2320.', 'Europe'),
('Honda Prelude', '33.7', '4', '2210.', 'Japan'), ('Toyota Corolla', '32.4', '4',
'2350.', 'Japan'), ('Datsun 200SX', '32.9', '4', '2615.', 'Japan'), ('Mazda 626',
'31.6', '4', '2635.', 'Japan'), ('Peugeot 505s Turbo Diesel', '28.1', '4', '3230.',
'Europe'), ('Saab 900s', '0', '4', '2800.', 'Europe'), ('Volvo Diesel', '30.7',
'6', '3160.', 'Europe'), ('Toyota Cressida', '25.4', '6', '2900.', 'Japan'),
('Datsun 810 Maxima', '24.2', '6', '2930.', 'Japan'), ('Volkswagen Rabbit l',
'36.0', '4', '1980.', 'Europe'), ('Mazda GLC Custom l', '37.0', '4', '2025.',
'Japan'), ('Mazda GLC Custom', '31.0', '4', '1970.', 'Japan'), ('Nissan Stanza XE',
'36.0', '4', '2160.', 'Japan'), ('Honda Accord', '36.0', '4', '2205.', 'Japan'),
('Toyota Corolla', '34.0', '4', '2245', 'Japan'), ('Honda Civic', '38.0', '4',
'1965.', 'Japan'), ('Honda Civic (auto)', '32.0', '4', '1965.', 'Japan'), ('Datsun
310 GX', '38.0', '4', '1995.', 'Japan'), ('Toyota Celica GT', '32.0', '4', '2665.',
'Japan'), ('Volkswagen Pickup', '44.0', '4', '2130.', 'Europe')]
```

Implementation of Linear Regression:

```python
df = spark.read.csv("/databricks-datasets/Rdatasets/data-
001/csv/ggplot2/diamonds.csv", header="true", inferSchema="true")
display(df)

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder
cat_cols= ["cut", "color", "clarity"]
stages = [] # Stages in Pipeline

for c in cat_cols:
    stringIndexer = StringIndexer(inputCol=c, outputCol=c + "_index")
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], \
            outputCols=[c + "_vec"])
    stages += [stringIndexer, encoder] # Stages will be run later on

from pyspark.ml.feature import VectorAssembler

# Transform all features into a vector
num_cols = ["carat", "depth", "table", "x", "y", "z"]
assemblerInputs = [c + "_vec" for c in cat_cols] + num_cols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]

# Create pipeline and use on dataset
pipeline = Pipeline(stages=stages)
df = pipeline.fit(df).transform(df)

train, test = df.randomSplit([0.90, 0.1], seed=123)
print('Train dataset count:', train.count())
print('Test dataset count:', test.count())

from pyspark.ml.feature import StandardScaler

# Fit scaler to train dataset
scaler = StandardScaler().setInputCol('features') \
        .setOutputCol('scaled_features')
scaler_model = scaler.fit(train)

# Scale train and test features
train = scaler_model.transform(train)
test = scaler_model.transform(test)

from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol='scaled_features', labelCol='price')
lr_model = lr.fit(train)

train_predictions = lr_model.transform(train)
test_predictions = lr_model.transform(test)

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(predictionCol="prediction", \
                 labelCol="price", metricName="r2")

print("Train R2:", evaluator.evaluate(train_predictions))
print("Test R2:", evaluator.evaluate(test_predictions))
```

EXPERIMENT-5

DESCRIPTION: MLops using Heroku and Github actions.
MLops in Text Classification
Step 1. Dataset and Preprocessing
Step 2. Encoding and Machine Learning Model
Step 3. Deployment on Heroku
Step 4: Incorporating GitHub actions

GITHUB LINK: miguelfzafra/Latest-News-Classifier: Master in Data Science Final Project (github.com)

APPROACH:
The whole process of the development of this project has been divided into three different posts:

- Classification model training (link)
- News articles web scraping (link)
- App creation and deployment (this post)

HEROKU DEPLOYMENT:

Steps to be followed to deploy on Heroku:

```
# after signing in to Heroku and opening the anaconda prompt
# we create a new folder
$ mkdir dash-app-lnclass
$ cd dash-app-lnclass# initialize the folder with git
$ git init
```

After that, we create an environment file (environment.yml) in which we will indicate the dependencies we are going to need:

```
name: dash_app_lnclass #Environment name
dependencies:
  - python=3.6
  - pip:
    - dash
    - dash-renderer
    - dash-core-components
    - dash-html-components
    - dash-table
    - plotly
    - gunicorn # for app deployment
    - nltk
    - scikit-learn
    - beautifulsoup4
    - requests
    - pandas
    - numpy
    - lxml
```

And activate the environment:

```
$ conda env create
$ activate dash_app_lnclass
```

Then, we initialize the folder with app.py, requirements.txt and a Procfile:

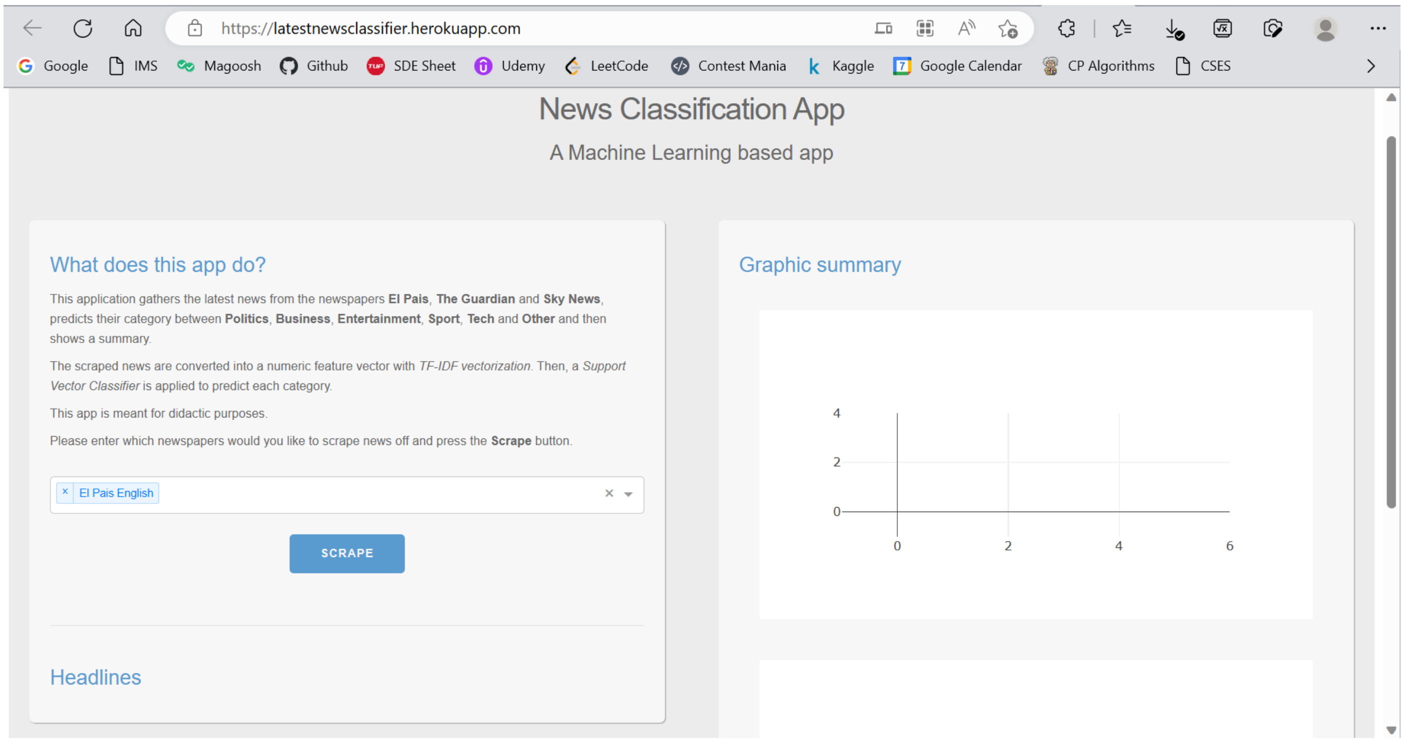
```
# the procfile must contain the following line of code
web: gunicorn app:server# to create the requirements.txt file, we run the
following:
$ pip freeze > requirements.txt
```

Finally, we initialize Heroku, add the files to Git and deploy:

```
$ heroku create lnclass # change my-dash-app to a unique name
$ git add . # add all files to git
$ git commit -m 'Comment'
$ git push heroku master # deploy code to heroku
$ heroku ps:scale web=1  # run the app with a 1 heroku "dyno"
```

DEPLOYMENT LINK: Dash (latestnewsclassifier.herokuapp.com)

OUTPUT SCREENSHOT:



News Classification App

A Machine Learning based app

What does this app do?

This application gathers the latest news from the newspapers **El Pais**, **The Guardian** and **Sky News**, predicts their category between **Politics**, **Business**, **Entertainment**, **Sport**, **Tech** and **Other** and then shows a summary.

The scraped news are converted into a numeric feature vector with *TF-IDF vectorization*. Then, a *Support Vector Classifier* is applied to predict each category.

This app is meant for didactic purposes.

Please enter which newspapers would you like to scrape news off and press the **Scrape** button.