INNOMATICS-Data-Analysis-and-Machine-Learning-HACKATHON

September 4, 2023

1 INNOMATICS Data Analysis and Machine Learning HACKATHON

2 1. Importing Necessary Libraries:

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import os
import warnings

from sklearn.preprocessing import StandardScaler, MinMaxScaler,RobustScaler
from sklearn.model_selection import  train_test_split, GridSearchCV,␣
 ↪cross_validate, cross_val_score

from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,␣
 ↪accuracy_score

warnings.filterwarnings("ignore")
%matplotlib inline
```

2.1 2. Read Data in Jupyter Notebook:

[3]:
```
# Load csv file
df = pd.read_csv('uber_rides_data.csv')
df.head()
```

[3]:
```
   ride_id  fare_amount          pickup_datetime  pickup_longitude  \
0  24238194          7.5  2015-05-07 19:52:06 UTC        -73.999817
1  27835199          7.7  2009-07-17 20:04:56 UTC        -73.994355
2  44984355         12.9  2009-08-24 21:45:00 UTC        -74.005043
3  25894730          5.3  2009-06-26 08:22:21 UTC        -73.976124
4  17610152         16.0  2014-08-28 17:47:00 UTC        -73.925023

   pickup_latitude  dropoff_longitude  dropoff_latitude  passenger_count
0        40.738354         -73.999512         40.723217                1
1        40.728225         -73.994710         40.750325                1
2        40.740770         -73.962565         40.772647                1
3        40.790844         -73.965316         40.803349                3
4        40.744085         -73.973082         40.761247                5
```

3 3. Exploratory Data Analysis [EDA]

[4]:
```
# Shape of given dataset:
df.shape
```

[4]:
```
(200000, 8)
```

[5]:
```
# Data type:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   ride_id            200000 non-null  int64
 1   fare_amount        200000 non-null  float64
 2   pickup_datetime    200000 non-null  object
 3   pickup_longitude   200000 non-null  float64
 4   pickup_latitude    200000 non-null  float64
 5   dropoff_longitude  199999 non-null  float64
 6   dropoff_latitude   199999 non-null  float64
 7   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(1)
memory usage: 12.2+ MB
```

[6]:
```
# Exploring Null Values:
df.isna().sum()
```

[6]:
```
ride_id              0
fare_amount          0
pickup_datetime      0
```

```
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

[7]:
```
# convert 'pickup_datetime' to datetime datatype
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
```

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   ride_id            200000 non-null  int64
 1   fare_amount        200000 non-null  float64
 2   pickup_datetime    200000 non-null  datetime64[ns, UTC]
 3   pickup_longitude   200000 non-null  float64
 4   pickup_latitude    200000 non-null  float64
 5   dropoff_longitude  199999 non-null  float64
 6   dropoff_latitude   199999 non-null  float64
 7   passenger_count    200000 non-null  int64
dtypes: datetime64[ns, UTC](1), float64(5), int64(2)
memory usage: 12.2 MB
```

[9]:
```
df.head()
```

[9]:

	ride_id	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	1
1	27835199	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	1
2	44984355	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	1
3	25894730	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	5

[10]:
```
# Remove null values:
df.dropna(inplace=True)
```

[11]:
```
# Checking again null values:
df.isna().sum()
```

[11]:
```
ride_id              0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

[12]:
```
# Calculate the average fare amount
average_fare = df['fare_amount'].mean()
print("The average fare amount is:", average_fare)
```

```
The average fare amount is: 11.359891549457748
```

[13]:
```
# Function to calculate Haversine distance
def haversine(lat1, lon1, lat2, lon2):
    # Radius of the Earth in kilometers
    R = 6371

    # Convert latitude and longitude from degrees to radians
    lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    distance = R * c

    return distance
```

[14]:
```
df['haversine_distance'] = df.apply(lambda row:␣
 ↪haversine(row['pickup_latitude'], row['pickup_longitude'],␣
 ↪row['dropoff_latitude'], row['dropoff_longitude']), axis=1)

# Calculate the median Haversine distance
median_haversine_distance = df['haversine_distance'].median()

print("The median Haversine distance between pickup and dropoff locations is:",␣
 ↪median_haversine_distance, "kilometers")
```

```
The median Haversine distance between pickup and dropoff locations is:
```

```
2.1209923961833708 kilometers
```

[15]:
```
max_haversine_distance = df['haversine_distance'].max()

print("The maximum Haversine distance between pickup and dropoff locations is:
 ↪", max_haversine_distance, "kilometers")
```

```
The maximum Haversine distance between pickup and dropoff locations is:
16409.239135313168 kilometers
```

[16]:
```
# Rides have 0.0 haversine distance between pickup and dropoff location
rides_with_zero_distance = df[df['haversine_distance'] == 0.0]
num_rides_with_zero_distance = len(rides_with_zero_distance)

print("The number of rides with a Haversine distance of 0.0 between pickup and␣
 ↪dropoff locations is:", num_rides_with_zero_distance)
```

```
The number of rides with a Haversine distance of 0.0 between pickup and dropoff
locations is: 5632
```

[17]:
```
# The mean 'fare_amount' for rides with 0 haversine distance

# Filter the DataFrame to include only rides with 0.0 Haversine distance
rides_with_zero_distance = df[df['haversine_distance'] == 0.0]

# Calculate the mean 'fare_amount' for these rides
mean_fare_for_zero_distance = rides_with_zero_distance['fare_amount'].mean()

print("The mean 'fare_amount' for rides with 0.0 Haversine distance is:",␣
 ↪mean_fare_for_zero_distance)
```

```
The mean 'fare_amount' for rides with 0.0 Haversine distance is:
11.585317826704546
```

3.0.1 Analytical Question:

- Do you sense something fishy? Try to analyze, and give your expert opinion in Jupyter Notebook.

3.0.2 Reply:

1. An error in the data, where the coordinates for both pickup and dropoff are the same or extremely close, resulting in a zero distance.
2. Short trips where the passenger may have been picked up and dropped off at the same location (e.g., the passenger entered the cab but then decided not to take the ride).

[18]:
```
# Maximum 'fare_amount' for a ride:
max_fare_amount = df['fare_amount'].max()
print("The maximum 'fare_amount' for a ride is:", max_fare_amount)
```

```
The maximum 'fare_amount' for a ride is: 499.0
```

[19]:
```
# Haversine distance between pickup and dropoff location for the costliest ride:

# Find the row with the highest 'fare_amount'
costliest_ride = df[df['fare_amount'] == df['fare_amount'].max()]

# Calculate the Haversine distance for the costliest ride
haversine_distance_costliest_ride = costliest_ride['haversine_distance'].
 ↪values[0]
print("The Haversine distance between pickup and dropoff locations for the␣
 ↪costliest ride is:", haversine_distance_costliest_ride, "kilometers")
```

```
The Haversine distance between pickup and dropoff locations for the costliest
ride is: 0.0007899213191009993 kilometers
```

3.0.3 Analytical Question:

- Do you sense something fishy? Try to analyze, and give your expert opinion in Jupyter Notebook.

3.0.4 Reply:

1. Data Anomalies: Extremely high fare amounts could be indicative of data entry errors or anomalies.
2. Outliers: It's possible that the costliest ride is an outlier.

[20]:
```
# How many rides were recorded in the year 2014?

df['pickup_year'] = df['pickup_datetime'].dt.year

# Count the number of rides recorded in the year 2014
rides_in_2014 = len(df[df['pickup_year'] == 2014])

print("The number of rides recorded in the year 2014 is:", rides_in_2014)
```

```
The number of rides recorded in the year 2014 is: 29968
```

[21]:
```
# How many rides were recorded in the first quarter of 2014?

# Filter rides for the first quarter of 2014
q1_2014_rides = df[(df['pickup_datetime'] >= '2014-01-01') &␣
 ↪(df['pickup_datetime'] <= '2014-03-31')]

# Count the number of rides in the first quarter of 2014
num_rides_q1_2014 = len(q1_2014_rides)
```

```
print("The number of rides recorded in the first quarter of 2014 is:",␣
 ↪num_rides_q1_2014)
```

The number of rides recorded in the first quarter of 2014 is: 7617

[22]:
```
# On which day of the week in September 2010, maximum rides were recorded ?

# Filter rides for the month of September 2010
september_2010_rides = df[(df['pickup_datetime'] >= '2010-09-01') &␣
 ↪(df['pickup_datetime'] <= '2010-09-30')]

# Group by day of the week and count the number of rides
day_of_week_counts = september_2010_rides['pickup_datetime'].dt.day_name().
 ↪value_counts()

# Find the day with the maximum number of rides
max_rides_day = day_of_week_counts.idxmax()

print("The day of the week in September 2010 with the maximum number of rides␣
 ↪recorded is:", max_rides_day)
```

The day of the week in September 2010 with the maximum number of rides recorded is: Wednesday

[23]:
```
# Creating Distance and ride_week_day columns:

df['distance'] = df.apply(lambda row: haversine(row['pickup_latitude'],␣
 ↪row['pickup_longitude'], row['dropoff_latitude'], row['dropoff_longitude']),␣
 ↪axis=1)

# Extract 'ride_week_day' from 'pickup_datetime'
df['ride_week_day'] = df['pickup_datetime'].dt.day_name()
```

[24]:
```
# Define a mapping of days to numerical values
day_mapping = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}
```

[25]:
```
# Map 'ride_week_day' to numerical values
df['ride_week_day'] = df['ride_week_day'].map(day_mapping)
```

[26]:
```
df.head()
```

[26]:

	ride_id	fare_amount	pickup_datetime	pickup_longitude	\
0	24238194	7.5	2015-05-07 19:52:06+00:00	-73.999817	
1	27835199	7.7	2009-07-17 20:04:56+00:00	-73.994355	
2	44984355	12.9	2009-08-24 21:45:00+00:00	-74.005043	
3	25894730	5.3	2009-06-26 08:22:21+00:00	-73.976124	
4	17610152	16.0	2014-08-28 17:47:00+00:00	-73.925023	

	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	\
0	40.738354	-73.999512	40.723217	1	
1	40.728225	-73.994710	40.750325	1	
2	40.740770	-73.962565	40.772647	1	
3	40.790844	-73.965316	40.803349	3	
4	40.744085	-73.973082	40.761247	5	

	haversine_distance	pickup_year	distance	ride_week_day
0	1.683323	2015	1.683323	4
1	2.457590	2009	2.457590	5
2	5.036377	2009	5.036377	1
3	1.661683	2009	1.661683	5
4	4.475450	2014	4.475450	4

[28]:
```
# Applying ML algorithms:

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import pandas as pd

# Input features (passenger_count, distance, ride_week_day),
# Target variable (fare_amount)

X = df[['passenger_count', 'distance', 'ride_week_day']]
y = df['fare_amount']

# Split the data into training and testing sets (70-30 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Train and evaluate Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
linear_reg_predictions = linear_reg.predict(X_test)
linear_reg_r2 = r2_score(y_test, linear_reg_predictions)
```

```
# Calculate adjusted R-squared for Linear Regression
n = len(y_test)  # Number of observations
p = X_test.shape[1]  # Number of predictors
adjusted_r2_linear = 1 - (1 - linear_reg_r2) * (n - 1) / (n - p - 1)

# Train and evaluate Random Forest Regression
random_forest_reg = RandomForestRegressor()
random_forest_reg.fit(X_train, y_train)
random_forest_reg_predictions = random_forest_reg.predict(X_test)
random_forest_reg_r2 = r2_score(y_test, random_forest_reg_predictions)

# Calculate adjusted R-squared for Random Forest Regression
adjusted_r2_random_forest = 1 - (1 - random_forest_reg_r2) * (n - 1) / (n - p -␣
 ↪1)

print("Adjusted R-squared for Linear Regression:", adjusted_r2_linear)
print("Adjusted R-squared for Random Forest Regression:",␣
 ↪adjusted_r2_random_forest)
```

```
Adjusted R-squared for Linear Regression: 0.00038413228805733723
Adjusted R-squared for Random Forest Regression: 0.6289517318801318
```

[29]:
```
# Create a comparison table
comparison_table = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest Regression'],
    'Adjusted R-squared': [adjusted_r2_linear, adjusted_r2_random_forest]
})

print(comparison_table)
```

```
                     Model  Adjusted R-squared
0        Linear Regression            0.000384
1  Random Forest Regression            0.628952
```

3.0.5 Observations:

- The Random Forest Regression model has a significantly higher adjusted R-squared score (approximately 0.629) compared to the Linear Regression model (approximately 0.000384).
- This suggests that the Random Forest Regression model explains a larger portion of the variance in the target variable and likely performs better in this particular context.

[ ]: