Laboratory Practice III – Practical 1

Name: Anuj Sachin Dhole
Roll No: B21042
Class: BE CE A
Subject: Laboratory Practice III (Machine Learning)

Practical 1

Problem Statement:

Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

Tasks to Perform:

1. **Pre-process the dataset.**
2. **Identify outliers.**
3. **Check the correlation.**
4. **Implement Linear Regression and Random Forest Regression models.**
5. **Evaluate the models** and compare their respective scores like **R^2, RMSE**, etc.

Dataset:

Source: [Uber Fares Dataset on Kaggle (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)](https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)

In [126]:

```
# Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.
# Perform following tasks:
# 1. Pre-process the dataset.
# 2. Identify outliers.
# 3. Check the correlation.
# 4. Implement linear regression and random forest regression models.
# 5. Evaluate the models and compare their respective scores like R2, RMSE, etc.
# Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-datase
```

In [1]:

```
# 1. Load libraries and dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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, mean_squared_error
```

In [2]:

```
import os
print("Current directory:", os.getcwd())

# List files in current directory
print("Files here:", os.listdir())
```

```
Current directory: C:\Users\MBA1 PC-17\Desktop\Anuj Dhole
Files here: ['.ipynb_checkpoints', 'Prac1.ipynb', 'uber.csv']
```

In [3]:

```
df = pd.read_csv('uber.csv')
df.dropna(inplace=True)
```

In [4]:

```
print("Data sample:")
print(df.head())

print("\nData info:")
print(df.info())

print("\nMissing values:")
print(df.isnull().sum())
```

```
Data sample:
   Unnamed: 0                            key  fare_amount  \
0    24238194    2015-05-07 19:52:06.0000003          7.5
1    27835199    2009-07-17 20:04:56.0000002          7.7
2    44984355   2009-08-24 21:45:00.00000061         12.9
3    25894730    2009-06-26 08:22:21.0000001          5.3
4    17610152  2014-08-28 17:47:00.000000188         16.0

           pickup_datetime  pickup_longitude  pickup_latitude  \
0  2015-05-07 19:52:06 UTC        -73.999817        40.738354
1  2009-07-17 20:04:56 UTC        -73.994355        40.728225
2  2009-08-24 21:45:00 UTC        -74.005043        40.740770
3  2009-06-26 08:22:21 UTC        -73.976124        40.790844
4  2014-08-28 17:47:00 UTC        -73.925023        40.744085

   dropoff_longitude  dropoff_latitude  passenger_count
0         -73.999512         40.723217                1
1         -73.994710         40.750325                1
2         -73.962565         40.772647                1
3         -73.965316         40.803349                3
4         -73.973082         40.761247                5

Data info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199999 entries, 0 to 199999
Data columns (total 9 columns):
Unnamed: 0           199999 non-null int64
key                  199999 non-null object
fare_amount          199999 non-null float64
pickup_datetime      199999 non-null object
pickup_longitude     199999 non-null float64
pickup_latitude      199999 non-null float64
dropoff_longitude    199999 non-null float64
dropoff_latitude     199999 non-null float64
passenger_count      199999 non-null int64
dtypes: float64(5), int64(2), object(2)
memory usage: 15.3+ MB
None

Missing values:
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [5]:

```
# Convert pickup_datetime to datetime format
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'], errors='coerce')
# converts the pickup_datetime column from strings into actual datetime objects

# pd.to_datetime() tries to parse each value in df['pickup_datetime'] and turn it into a datetime64 type (date +
time).

# errors='coerce' means:

# If a value can’t be parsed into a valid date/time, instead of raising an error, it will convert that value into
a missing value (NaT).
```

In [6]:

```
df.head()
```

Out[6]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	

In [7]:

```
# Extract datetime features
df['hour'] = df['pickup_datetime'].dt.hour
df['day'] = df['pickup_datetime'].dt.day
df['weekday'] = df['pickup_datetime'].dt.weekday
df['month'] = df['pickup_datetime'].dt.month
```

In [8]:

```
df.head()
```

Out[8]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247

In [9]:

```
# Haversine distance function
def haversine_distance(lat1, lon1, lat2, lon2):
    R = 6371  # Earth's radius in km
    phi1 = np.radians(lat1)
    phi2 = np.radians(lat2)
    d_phi = np.radians(lat2 - lat1)
    d_lambda = np.radians(lon2 - lon1)

    a = np.sin(d_phi / 2.0)**2 + \
        np.cos(phi1) * np.cos(phi2) * np.sin(d_lambda / 2.0)**2

    c = 2 * np.arcsin(np.sqrt(a))
    return R * c

# Apply Haversine formula to get distance
df['trip_distance_km'] = haversine_distance(
    df['pickup_latitude'],
    df['pickup_longitude'],
    df['dropoff_latitude'],
    df['dropoff_longitude']
)
```

In [10]:

```
# df['trip_distance_approx'] = np.sqrt(
#     (df['pickup_latitude'] - df['dropoff_latitude'])**2 +
#     (df['pickup_longitude'] - df['dropoff_longitude'])**2
# )

# Alterantive to Haversin
# Didn't use Haversin as it is complicated
# This Works well if your coordinates are in a flat (planar) coordinate system.

# Haversin Formula calculates the great-circle distance between two points on the Earth's surface (accounts for E
arth's curvature).
```

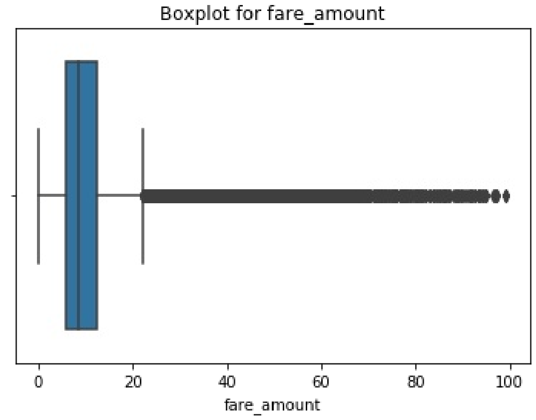
In [11]:

```
# Filter invalid values (NOT Outlier removal, just sanity check)
df = df[(df['fare_amount'] > 0) & (df['fare_amount'] < 100)]
df = df[(df['passenger_count'] > 0) & (df['passenger_count'] <= 6)]
df = df[(df['trip_distance_km'] > 0)]
```

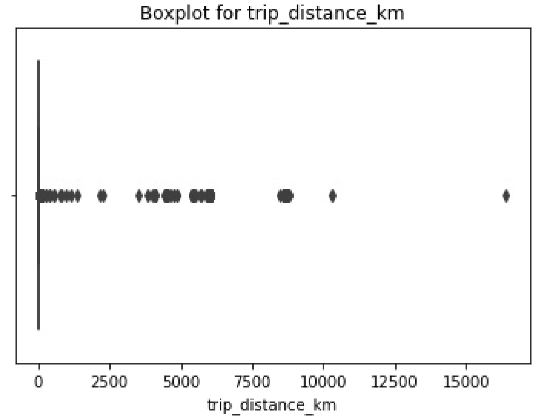
In [12]:

```
# 2. Outlier detection using IQR
def detect_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]

    print(f"\nOutliers in '{column}':")
    print(f"Q1 = {Q1:.2f}, Q3 = {Q3:.2f}, IQR = {IQR:.2f}")
    print(f"Lower Bound = {lower_bound:.2f}, Upper Bound = {upper_bound:.2f}")
    print(f"Number of Outliers: {len(outliers)}")

    # Boxplot
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=data[column])
    plt.title(f'Boxplot for {column}')
    plt.show()

    return outliers

outlier_columns = ['fare_amount', 'trip_distance_km', 'passenger_count']
for col in outlier_columns:
    detect_outliers_iqr(df, col)
```

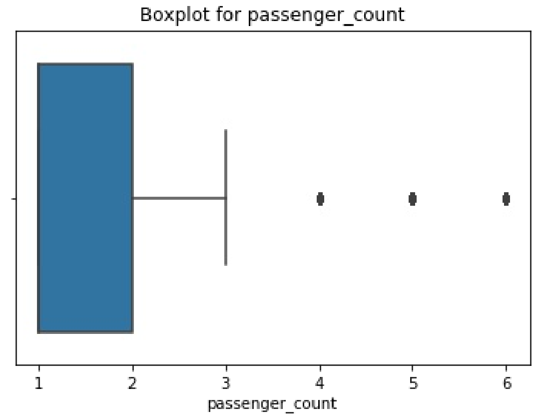
```
Outliers in 'fare_amount':
Q1 = 6.00, Q3 = 12.50, IQR = 6.50
Lower Bound = -3.75, Upper Bound = 22.25
Number of Outliers: 16544
```




```
Outliers in 'trip_distance_km':
Q1 = 1.28, Q3 = 3.96, IQR = 2.68
Lower Bound = -2.73, Upper Bound = 7.97
Number of Outliers: 16303
```




```
Outliers in 'passenger_count':
Q1 = 1.00, Q3 = 2.00, IQR = 1.00
Lower Bound = -0.50, Upper Bound = 3.50
Number of Outliers: 21908
```



In [13]:

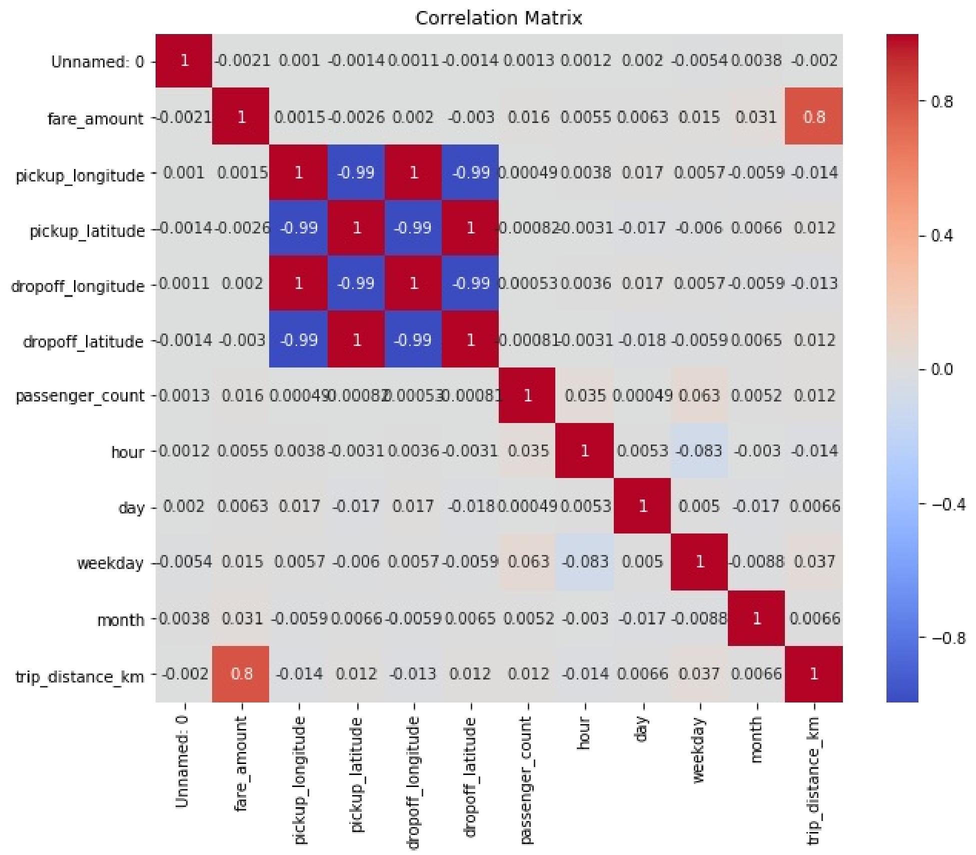
```
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return data[(data[column] >= lower) & (data[column] <= upper)]

# Apply to all outlier columns
for col in ['fare_amount', 'trip_distance_km', 'passenger_count']:
    df = remove_outliers_iqr(df, col)
```

In [14]:

```
# The calculated lower bound for outliers in 'fare_amount' is -3.75, which is below 0.

# However, the boxplot starts from 0, and you don’t see any data below 0.

# Why This Happens:
# This is not a bug — it's a result of your sanity filter before outlier detection:

# df = df[(df['fare_amount'] > 0) & (df['fare_amount'] < 100)]
# This line removes all rows where fare_amount ≤ 0, so:

# Even though the IQR lower bound is -3.75, there are no negative fare values left.

# Therefore, boxplot starts at 0 (the minimum in the filtered data).

# So the outliers are only on the upper side (above 22.25).
```

In [15]:

```
# Correlation Matrix

plt.figure(figsize=(10, 8))
corr = df.corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()

#if gives numeric error, use
# sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
```



In [16]:

```
# The reason this did not throw an error is because pandas.DataFrame.corr() automatically excludes non-numeric co
lumns (like pickup_datetime) when calculating the correlation matrix.

# Here's how it works:
# df.corr() only computes correlation between numeric columns (int or float).

# Columns with object, datetime, or bool types are ignored silently.

# That’s why the heatmap worked without error — pickup_datetime just wasn't included in the correlation matrix at
all.
# can verify using
# print(df.dtypes)  # shows the types of all columns
# print(df.corr().columns)  # shows only the numeric columns used in correlation
```

In [17]:

```
df.columns
```

Out[17]:

```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count', 'hour', 'day', 'weekday',
       'month', 'trip_distance_km'],
      dtype='object')
```

In [18]:

```
# 4. Prepare data for modeling
features = ['pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude',
            'passenger_count', 'hour', 'day', 'month', 'weekday', 'trip_distance_km']
# took all columns in features other than unnamed and key and pickup_datetime column.
X = df[features]
y = df['fare_amount']
```

In [31]:

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [32]:

```
# 5. Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
```

In [33]:

```
print(y_pred_lr)
```

```
[ 6.38876765  5.264624   12.34097931 ...  9.01298912 15.99987701
  9.0948439 ]
```

In [34]:

```
# 6. Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

In [35]:

```
# 7. Evaluation function
def evaluate_model(y_true, y_pred, model_name):
    r2 = r2_score(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"\n{model_name} Evaluation:")
    print(f"R² Score: {r2}")
    print(f"RMSE: {rmse}")

# Evaluate both models
evaluate_model(y_test, y_pred_lr, "Linear Regression")
evaluate_model(y_test, y_pred_rf, "Random Forest Regression")
```

```
Linear Regression Evaluation:
R² Score: 0.6467976913676975
RMSE: 2.1965875242556

Random Forest Regression Evaluation:
R² Score: 0.7080506226232166
RMSE: 1.9970572845215275
```

In [36]:

```
# Ideal R² and RMSE (for Uber fare prediction)
#  R² Score (Coefficient of Determination)
# R² ≈ 1.0 → Perfect prediction (model explains nearly all variance)

# R² > 0.95 → Excellent (usually with very clean or simulated data)

# R² ≈ 0.80–0.90 → Good, especially for real-world noisy data like Uber

# R² < 0.70 → Likely too much noise, missing key variables, or not enough filtering

# Goal: For your project with filtering, aim for R² ≥ 0.90 — this means your model is closely predicting actual f
ares.

#  RMSE (Root Mean Squared Error)
# This is in the same unit as fare_amount (USD).

# Ideal RMSE depends on your fare range.

# Fare Range    Ideal RMSE
# $5–$60 (your case)    RMSE < $3.00
# Narrower data RMSE < $2.00

# Goal: Try to get RMSE below $3, ideally close to $2 or less, which means average prediction error is small.
```