LP3 Practical - 1

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.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
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 mean_squared_error, r2_score
```

```
df = pd.read_csv('/content/sample_data/uber.csv')
df.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043

```
# Check for null values
print("Missing values:\n", df.isnull().sum())

# Drop rows with missing values
df.dropna(inplace=True)

# Check datatypes
print("\nData types:\n", df.dtyp
```

```
# Convert pickup_datetime to datetime if it's a string
if 'pickup_datetime' in df.columns and df['pickup_datetime'].dtype == 'object':
    df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])

# Extract datetime features
if 'pickup_datetime' in df.columns:
    df['hour'] = df['pickup_datetime'].dt.hour
    df['day'] = df['pickup_datetime'].dt.day
    df['month'] = df['pickup_datetime'].dt.month
    df['dayofweek'] = df['pickup_datetime'].dt.dayofweek

# Drop irrelevant columns
df.drop(['key', 'pickup_datetime'], axis=1, errors='ignore', inplace=True)

# Show final columns
df.head()
```

```
Missing values:
 Unnamed: 0           0
key                   0
fare_amount           0
pickup_datetime       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     1
dropoff_latitude      1
passenger_count       0
dtype: int64

Data types:
 Unnamed: 0             int64
key                    object
fare_amount           float64
pickup_datetime        object
pickup_longitude      float64
pickup_latitude       float64
dropoff_longitude     float64
dropoff_latitude      float64
passenger_count         int64
dtype: object
```

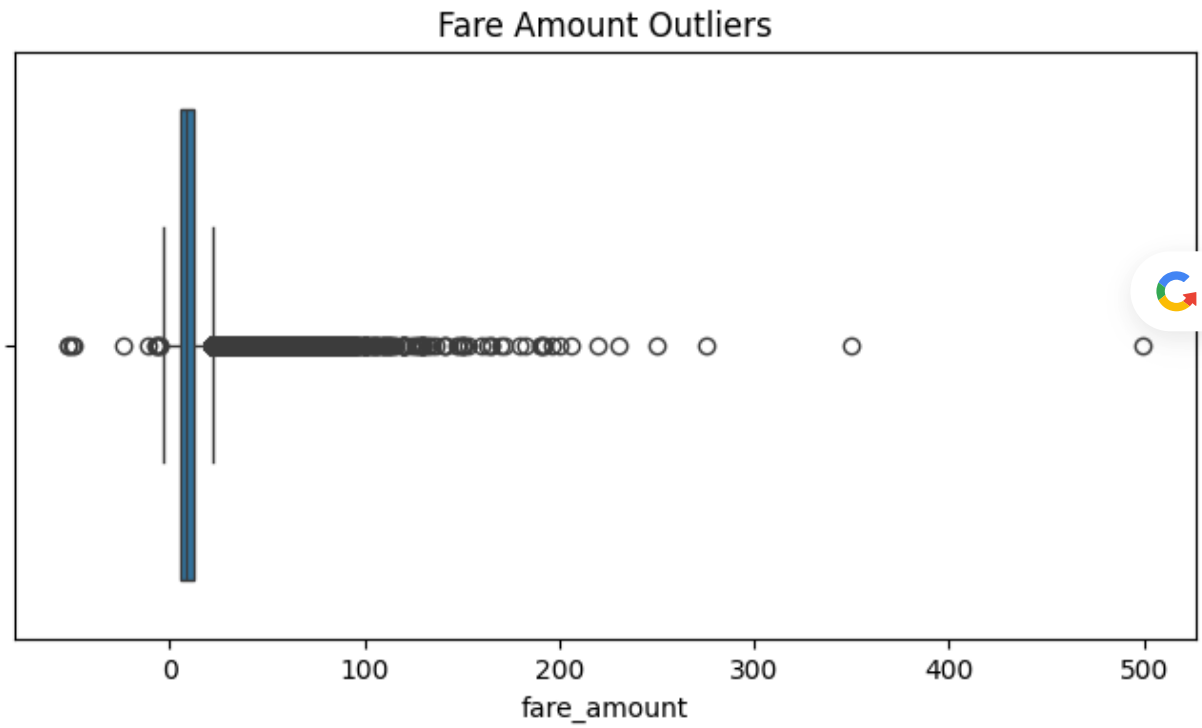
	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude
0	24238194	7.5	-73.999817	40.738354	-73.999512
1	27835199	7.7	-73.994355	40.728225	-73.994710
2	44984355	12.9	-74.005043	40.740770	-73.962565
3	25894730	5.3	-73.976124	40.790844	-73.965316

```
# Visualize outliers in 'fare_amount'
plt.figure(figsize=(8,4))
sns.boxplot(x=df['fare_amount'])
plt.title('Fare Amount Outliers')
```

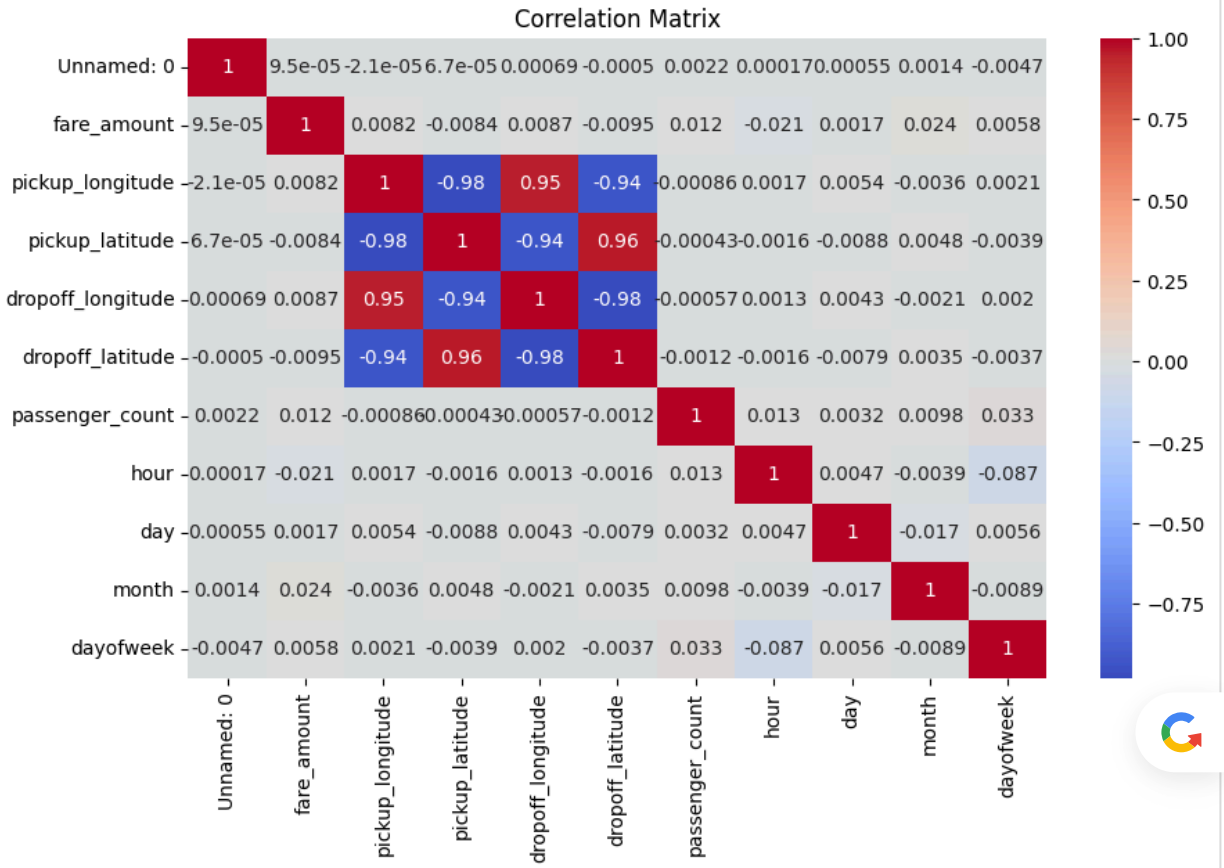
```
plt.show()

# Remove fare_amount outliers
df = df[(df['fare_amount'] > 0) & (df['fare_amount'] < 100)]

# Optional: remove unrealistic coordinates
df = df[(df['pickup_latitude'] >= -90) & (df['pickup_latitude'] <= 90)]
df = df[(df['pickup_longitude'] >= -180) & (df['pickup_longitude'] <= 180)]
df = df[(df['dropoff_latitude'] >= -90) & (df['dropoff_latitude'] <= 90)]
df = df[(df['dropoff_longitude'] >= -180) & (df['dropoff_longitude'] <= 180)]
```




```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




```
X = df.drop('fare_amount', axis=1)
y = df['fare_amount']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
```

```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict
y_pred_lr = lr_model.predict(X_test)

# Evaluate
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)
```

```
print(f"Linear Regression RMSE: {rmse_lr}")
print(f"Linear Regression R2 Score: {r2_lr}")
```

```
Linear Regression RMSE: 9.288902442406064
Linear Regression R2 Score: 0.0011294335113416487
```

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict
y_pred_rf = rf_model.predict(X_test)

# Evaluate
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest RMSE: {rmse_rf}")
print(f"Random Forest R2 Score: {r2_rf}")
```

```
Random Forest RMSE: 4.089661435272358
Random Forest R2 Score: 0.8063778112891907
```

```
comparison = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest'],
    'RMSE': [rmse_lr, rmse_rf],
    'R2 Score': [r2_lr, r2_rf]
})

print(comparison)
```

```
               Model      RMSE  R2 Score
0  Linear Regression  9.288902  0.001129
1      Random Forest  4.089661  0.806378
```