Assignment No. 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

Step 1: Download the dataset from Kaggle

1. **Dataset link:** Uber Fares Dataset
2. Go to the Kaggle dataset page.
3. Click on the **Download** button to download the uber.csv file.
4. After downloading, make sure the dataset is placed in the same directory where you plan to open and run your Jupyter Notebook.
   - If the dataset is located in a different directory, update the file path in the code where pd.read_csv() is used (e.g., pd.read_csv('path/to/uber.csv')).

Step 2: Open Jupyter Notebook

1. **Open Jupyter Notebook:**
   - Launch Jupyter Notebook from your system (either through **Anaconda** or by typing jupyter notebook in the command line).
   - This will open the Jupyter environment in your web browser.
2. **Create a new notebook:**
   - Once Jupyter opens, navigate to the directory where you placed the dataset.
   - Click **New** -> **Python 3** to create a new Python notebook.

Step 3: Paste the code into Jupyter Notebook

Now you will start adding the code in chunks. Here's how each step of the code works.

3.1: Import necessary libraries

```
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
```

- **Explanation**: This step imports essential libraries:
  - pandas for handling data and creating DataFrames.
  - numpy for numerical operations.
  - matplotlib and seaborn for visualization.
  - train_test_split from sklearn.model_selection to split the data into training and testing sets.
- **Execute this code block**: Place this code in the first cell of your notebook and run it by pressing **Shift + Enter**.

3.2: Load the dataset

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

- **Explanation**:
  - pd.read_csv('uber.csv') reads the dataset into a pandas DataFrame.
  - df.head() displays the first 5 rows of the dataset so you can verify it loaded correctly.
- **Note**: If the dataset is saved in another folder, you need to provide the full path, like pd.read_csv('C:/path/to/uber.csv').

3.3: Check dataset shape

```
df.shape
```

- **Explanation**: This line checks the number of rows and columns in the dataset.
  - The result will be something like (n_rows, n_columns).

3.4: Check for missing values

```
df.isnull()
```

- **Explanation**: This line checks for missing values (null values) in the dataset. It will return True for each cell that contains a missing value.

3.5: Drop unnecessary columns

```
df.drop(columns=["Unnamed: 0", "key"], inplace=True)
df.head()
```

- **Explanation**:
  - df.drop() removes columns that are not useful for analysis.
  - inplace=True ensures that the changes are made directly to the DataFrame without needing to reassign it.
  - After dropping the columns, we use df.head() again to confirm that the columns are removed.

3.6: Count missing values in each column

```
df.isnull().sum()
```

- **Explanation**: This counts the total number of missing values in each column. This is important for determining which columns need further cleaning.

3.7: Fill missing values in specific columns

```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(), inplace=True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(), inplace=True)
```

- **Explanation**:
  - This fills missing values in the dropoff_latitude column using the column's mean value.
  - The dropoff_longitude column is filled using the column's median value. These choices (mean and median) are typical strategies for handling missing data.
  - inplace=True ensures that the changes are made directly to the DataFrame.

3.8: Check the data types of each column

```
df.dtypes
```

- **Explanation**: This displays the data types (e.g., integer, float, object) of each column. This is useful to identify if any column needs conversion, especially datetime columns.

3.9: Convert 'pickup_datetime' to datetime format

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime)

df.dtypes
```

- **Explanation**: This converts the pickup_datetime column to a proper datetime format, allowing for easier manipulation and feature extraction.
  - After conversion, you can check the data types again to confirm the change.

3.10: Create new columns based on 'pickup_datetime'

```
df = df.assign(hour=df.pickup_datetime.dt.hour,
               day=df.pickup_datetime.dt.day,
               month=df.pickup_datetime.dt.month,
               year=df.pickup_datetime.dt.year,
               dayofweek=df.pickup_datetime.dt.dayofweek)
```

- **Explanation**:
  - This extracts various components (hour, day, month, year, day of the week) from the pickup_datetime column and adds them as new columns to the DataFrame.

3.11: Drop the 'pickup_datetime' column

```
df = df.drop(["pickup_datetime"], axis=1)

df
```

- **Explanation**: Since we've extracted useful information from pickup_datetime, we no longer need the original column, so it's dropped.

3.12: Define a function to calculate travel distance

```
from math import *

def distance_formula(longitude1, latitude1, longitude2, latitude2):
    travel_dist = []

    for pos in range(len(longitude1)):
        lon1, lan1, lon2, lan2 = map(radians, [longitude1[pos], latitude1[pos], longitude2[pos], latitude2[pos]])
```

```
        dist_lon = lon2 - lon1

        dist_lan = lan2 - lan1

        a = sin(dist_lan/2)**2 + cos(lan1) * cos(lan2) * sin(dist_lon/2)**2

        c = 2 * asin(sqrt(a)) * 6371

        travel_dist.append(c)


    return travel_dist
```

- **Explanation**:
  - This function calculates the distance between two geographical points using the **Haversine formula**, which accounts for the curvature of the Earth.
  - It converts the latitude and longitude values from degrees to radians and then applies trigonometry to compute the distance in kilometers.

3.13: Add the calculated travel distance to the DataFrame

```
df['dist_travel_km'] = distance_formula(df.pickup_longitude.to_numpy(), df.pickup_latitude.to_numpy(), df.dropoff_longitude.to_numpy(), df.dropoff_latitude.to_numpy())
```

- **Explanation**: This uses the distance_formula function to calculate the distance between the pickup and dropoff points for each row in the DataFrame, and stores the result in the new dist_travel_km column.

3.14: Define features (X) and target (y)

```
df_x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','passenger_count','hour','day','month','year','dayofweek','dist_travel_km']]

df_y = df['fare_amount']
```

- **Explanation**:
  - df_x includes the input features (longitude, latitude, passenger count, etc.) used for predicting the target.
  - df_y is the target variable (fare amount) that we want to predict.

3.15: Split the data into training and testing sets

```
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state=1)
```

- **Explanation**:
  - train_test_split splits the data into training (80%) and testing (20%) sets.
  - random_state=1 ensures that the split is reproducible.

3.16: Train and predict using Linear Regression

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()

reg.fit(x_train, y_train)

y_pred_lin = reg.predict(x_test)

print(y_pred_lin)
```

- **Explanation**:
  - This initializes a linear regression model and trains it using the training data.
  - After training, it predicts the fare amounts for the test set and prints the predictions.

3.17: Train and predict using Random Forest

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100)

rf.fit(x_train, y_train)

y_pred_rf = rf.predict(x_test)

print(y_pred_rf)
```