In [ ]:
```python
# import
import time
start_time = time.time()

import os
import numpy as np
import pandas as pd
import pyarrow.parquet as pa
from sklearn.feature_extraction import DictVectorizer

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import root_mean_squared_error

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```python
vectorise = DictVectorizer()
linear_regression = LinearRegression()
```

In [ ]:
```python
# setting path to the data directory
CURRENT_DIRECTORY = os.getcwd()
PARENT_DIRECTORY = os.path.dirname(CURRENT_DIRECTORY)
DATA_PATH = os.path.join(PARENT_DIRECTORY, '_data')
```

In [ ]:
```python
# read the data
def read_data(data):
    if data.endswith('.parquet'):
        data = pa.read_table(data)
        df = data.to_pandas() # converting to pandas df
        df.columns = df.columns.str.lower()
        return df
    elif data.endswith('.csv'):
        df = pd.read_csv(data)
        df.columns = df.columns.str.lower()
        return df

    else:
        return 'Not valid format'
```

In [ ]:
```python
# To calculate the standard deviation of the pick and drop time in minutes
def standard_deviation(data):
    data['duration'] = pd.to_datetime(data['tpep_dropoff_datetime']) - pd.to_datetime(data['tpep_pickup_datetim
    # Convert duration to total seconds
    data['duration'] = data['duration'].dt.total_seconds()
    # Convert seconds to hours and minutes
    data['duration'] = data['duration'] / 60
    # Standard deviation
    return data, data['duration'].std()
```

In [ ]:
```python
def outliers(data):
    data_outliers = data[(data['duration']>=1)&(data['duration']<=60)]
    return data_outliers, (data_outliers.shape[0] / data.shape[0]) * 100
```

In [ ]:
```python
# One-hot encoding
def one_hot_encoding(df, choice):
    # Converting pick up and drop off location id into strings
    df['pulocationid'] = df['pulocationid'].astype(str)
    df['dolocationid'] = df['dolocationid'].astype(str)

    # Converting DataFrame into a list of dictionaries
    df_dict = df[['pulocationid', 'dolocationid']].to_dict(orient='records')

    if choice == 0:
        X_train = vectorise.fit_transform(df_dict)
        return X_train
    elif choice == 1:
        X_val = vectorise.transform(df_dict)
        return X_val
    else:
        return 'Enter Choice 0 or 1'
```

In [ ]:
```python
# Define RMSE function
def rmse(y_, y_pred):
    return root_mean_squared_error(y_, y_pred)
```

In [ ]:
```python
def training(data, X_train):
    y_train = data['duration'].values
```

```python
    linear_regression.fit(X_train, y_train)
    y_prediction = linear_regression.predict(X_train)

    # Calculate the Root Mean Square Error
    RMSE = rmse(y_train, y_prediction)
    return y_train, y_prediction, RMSE
```

In [ ]:
```python
def evaluation(df_val, X_val):
    y_val = df_val['duration'].values
    y_prediction = linear_regression.predict(X_val)
    RMSE = rmse(y_val, y_prediction)
    return RMSE
```

Q1. Downloading the data

We'll use the same NYC taxi dataset, but instead of "Green Taxi Trip Records", we'll use "Yellow Taxi Trip Records".

Download the data for January and February 2023.

Read the data for January. How many columns are there?

a) 16

b) 17

c) 18

d) 19

Answer : d) 19

In [ ]:
```python
# File Name
january_file_name = 'yellow_tripdata_2023-01.parquet'
february_file_name = 'yellow_tripdata_2023-02.parquet'

# Join January data path
january_data_path = os.path.join(DATA_PATH, january_file_name)
# Join February data path
february_data_path = os.path.join(DATA_PATH, february_file_name)

# READ JANUARY DATA
df_train = read_data(january_data_path)
# READ FEBRUARY DATA
df_val = read_data(february_data_path)

print(f'January Data Shape = {df_train.shape}')
print(f'February Data Shape = {df_val.shape}')
```

```
January Data Shape = (3066766, 19)
February Data Shape = (2913955, 19)
```

Q2. Computing duration

Now let's compute the duration variable. It should contain the duration of a ride in minutes.

What's the standard deviation of the trips duration in January?

a) 32.59

b) 42.59

c) 52.59

d) 62.59

Answer: b) 42.59

In [ ]:
```python
df_train, january_duriation_std_dev = standard_deviation(df_train)
print('Standard Deviation of Pick and Drop time for the month of January (time in minutes)', january_duriation_s
print(f'January Data Shape after adding column minutes= {df_train.shape}')
```

```
Standard Deviation of Pick and Drop time for the month of January (time in minutes) 42.59435124195458
January Data Shape after adding column minutes= (3066766, 20)
```

In [ ]:
```python
df_val, february_duriation_std_dev = standard_deviation(df_val)
```

```
print('Standard Deviation of Pick and Drop time for the month of FEbruary (time in minutes)', february_duratio
print(f'February Data Shape after adding column minutes = {df_val.shape}')
```

```
Standard Deviation of Pick and Drop time for the month of FEbruary (time in minutes) 42.84210176105113
February Data Shape after adding column minutes = (2913955, 20)
```

Q3. Dropping outliers

Next, we need to check the distribution of the duration variable. There are some outliers. Let's remove them and keep only the records where the duration was between 1 and 60 minutes (inclusive).
What fraction of the records left after you dropped the outliers?
a) 90%
b) 92%
c) 95%
d) 98%

Answer: d) 98%

In [ ]:
```
df_train, jan_records_left = outliers(df_train)
print('Fraction of the records left after dropping the outliers = ', jan_records_left)
print(f'January Data Shape after removing outliers= {df_train.shape}')
```

```
Fraction of the records left after dropping the outliers =  98.1220282212598
January Data Shape after removing outliers= (3009173, 20)
```

In [ ]:
```
df_val, feb_records_left = outliers(df_val)
print('Fraction of the records left after dropping the outliers = ', feb_records_left)
print(f'February Data Shape after removing outliers = {df_val.shape}')
```

```
Fraction of the records left after dropping the outliers =  98.00944077722545
February Data Shape after removing outliers = (2855951, 20)
```

Q4. One-hot encoding

Let's apply one-hot encoding to the pickup and dropoff location IDs. We'll use only these two features for our model.

1. Turn the dataframe into a list of dictionaries (remember to re-cast the ids to strings - otherwise it will label encode them)
2. Fit a dictionary vectorizer
3. Get a feature matrix from it

What's the dimensionality of this matrix (number of columns)?

a) 2
b) 155
c) 345
d) 515
e) 715

Answer: d) 515

In [ ]:
```
# Fit DictVectorizer and transform January data
X_train = one_hot_encoding(df_train, choice=0)
print('*********************************************')
print(f'Feature Matrix size = {X_train.shape}')
```

```
*********************************************
Feature Matrix size = (3009173, 515)
```

RMSE

RMSE = sqrt [(Σ(Pi – Oi)²) / n]

Where
Pi==> Predicted Value
Oi==> Observed Value

Q5. Training a model

Now let's use the feature matrix from the previous step to train a model.

1. Train a plain linear regression model with default parameters, where duration is the response variable
2. Calculate the RMSE of the model on the training data

What's the RMSE on train?

a) 3.64
b) 7.64
c) 11.64
d) 16.64

Answer: b) 7.64

In [ ]:
```python
y_train, y_prediction, RMSE = training(df_train, X_train)
print(f'Training Root Mean Square Error = {RMSE}')
```

```
Training Root Mean Square Error = 7.649262092523899
```

Q6. Evaluating the model

Now let's apply this model to the validation dataset (February 2023).
What's the RMSE on validation?
a) 3.81
b) 7.81
c) 11.81
d) 16.81

Answer: b) 7.81

In [ ]:
```python
# Transform February data using the fitted DictVectorizer
X_val = one_hot_encoding(df_val, choice=1)
```

In [ ]:
```python
RMSE = evaluation(df_val, X_val)
print(f'Validation RMSE: {RMSE}')
```

```
Validation RMSE: 7.811817680839882
```

In [ ]:
```python
end_time = time.time()
print(f'Total time taken = {end_time-start_time}')
```

```
Total time taken = 76.92450189590454
```
