#데이터사이언스를 위한 파이썬 프로그래밍

머신러닝 기법을 활용한 택시 우버 요금 예측

Classic Blue

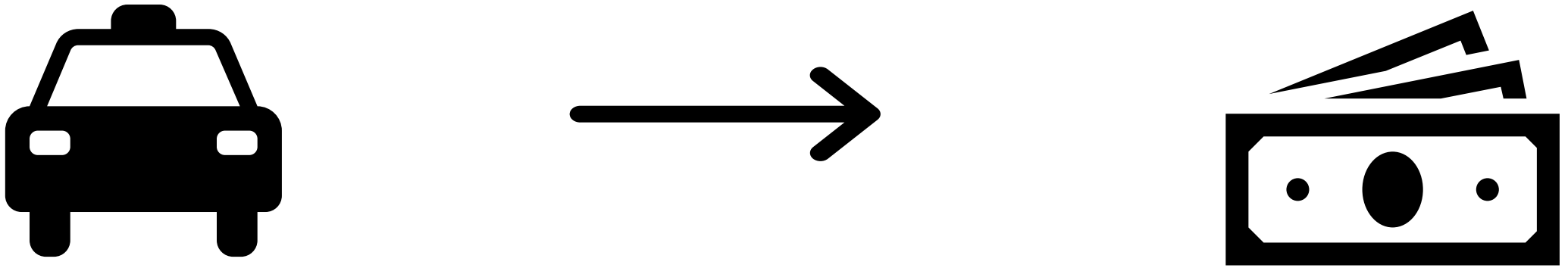
목차

A table of Contents

프로젝트 개요

프로젝트 목표

우버 요금제의 분석을 통한 우버 요금 예측



데이터 설명

데이터 출처: Kaggle uber-fares-dataset

데이터 설명

- key - a unique identifier for each trip
- fare_amount - the cost of each trip in usd
- pickup_datetime - date and time when the meter was engaged
- passenger_count - the number of passengers in the vehicle (driver entered value)
- pickup_longitude - the longitude where the meter was engaged
- pickup_latitude - the latitude where the meter was engaged
- dropoff_longitude - the longitude where the meter was disengaged
- dropoff_latitude - the latitude where the meter was disengaged

데이터 전처리

필요없는 열 삭제
Null값 확인

['Unnamed: 0','key'] 중복되는 열로 삭제 가능
차원을 줄일 수 있음

```
[5]  1 #study how many missing data are there ?
     2 uber_fare_df_1.isnull().sum()
```

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

데이터 전처리

Null값 삭제

Null값이
dropoff_longitude 1개
dropoff_latitude 1개
=> 해당 행 삭제해도 많은 데이터 존재 따라서 삭제가능

```
[6]  1 #since there are 2 values which are missing , we can remove it ....
     2 uber_fare_df_1.dropna(axis=0,inplace=True)
     3 uber_fare_df_1
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5
...	...	...	...	...	...	...	...
199995	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73.986525	40.740297	1
199996	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74.006672	40.739620	1
199997	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73.858957	40.692588	2
199998	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-73.983215	40.695415	1
199999	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-73.985508	40.768793	1

199999 rows × 7 columns

데이터 전처리

시간단위 쪼개기

연도/날짜/시간 단위로 열 쪼개기
시간단위 또한 아침 후 저녁 밤으로 쪼개기

```
1 #extract the time out of date and bin it to the morning, afternoon, evening, night
2 uber_fare_df_1['pickup_datetime'] = pd.to_datetime(uber_fare_df_1.pickup_datetime,utc=True)
```

```
[8]  1 uber_fare_df_1['pickup_timehour']= uber_fare_df_1['pickup_datetime'].apply(lambda x: x.hour)
     2 uber_fare_df_1['Day']= uber_fare_df_1['pickup_datetime'].dt.day_name()
     3 uber_fare_df_1['Day Part']= pd.cut(uber_fare_df_1['pickup_timehour'],bins=[-1,4,12,17,21,23],labels=['Night','Morning','Afternoon','Evening','Night'],ordered=False)
     4 uber_fare_df_1['pickup_month']= uber_fare_df_1['pickup_datetime'].apply(lambda x: x.month)
     5 uber_fare_df_1['pickup_year']= uber_fare_df_1['pickup_datetime'].apply(lambda x: x.year)
```

데이터 전처리

위도와 경도 이용해서 움직인 거리 km단위로 바꾸기

```python
#import libraries
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import datetime
#from geopy.geocoders import Nominatim
#import plotly.express as px
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
#create a function for distance calculation between the two locations latitudes and logitudes
from math import radians, cos, sin, asin, sqrt
def calculate_distance(lat1, lat2, lon1, lon2):

  # The math module contains a function named
  # radians which converts from degrees to radians.
  lon1 = radians(lon1)
  lon2 = radians(lon2)
  lat1 = radians(lat1)
  lat2 = radians(lat2)

  # Haversine formula
  dlon = lon2 - lon1
  dlat = lat2 - lat1
  a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2

  c = 2 * asin(sqrt(a))

  # Radius of earth in kilometers. Use 3956 for miles
  r = 6371

  # calculate the result
  return(c * r)

uber_fare_df_1['distance_travelled (KM)']= uber_fare_df_1.apply(lambda x: calculate_distance(x['pickup_latitude'],x['dropoff_latitude'],x['pickup_longitude'],x['dropoff_longitude']),axis=1)
```

데이터 전처리

이상치 값 수정 및 범위 제한하기

위도와 경도가 0으로 움직이지 않는 값 존재 => 해당 행 범위에 넣지 않음
요금이 0인 행 해당 범위에 넣지 않음

```
1 print(f'Before dropping {uber_fare_df_1.shape}')
```

```
Before dropping (199999, 13)
```

```
[13]  1 #remove the data
      2 #remove the latitude and longitude ===000, passenger_count=0, fare<=0
      3
      4 uber_fare_df_1=uber_fare_df_1[~((uber_fare_df_1['pickup_latitude']==0.000000)&(uber_fare_df_1['pickup_longitude']==0.000000))]
      5 print(f'After dropping the data, the final shape is {uber_fare_df_1.shape}')
```

```
After dropping the data, the final shape is (196220, 13)
```

```
[14]  1 uber_fare_df_1=uber_fare_df_1[~(uber_fare_df_1['fare_amount']<=0)]
      2 uber_fare_df_1=uber_fare_df_1[~(uber_fare_df_1['passenger_count']==0)]
      3 print(f'After dropping the data, the final shape is {uber_fare_df_1.shape}')
```

```
After dropping the data, the final shape is (195511, 13)
```

```
[15]  1 uber_fare_df_1=uber_fare_df_1[~(uber_fare_df_1['passenger_count']==208)]
      2 print(f'After dropping the data, the final shape is {uber_fare_df_1.shape}')
```

```
After dropping the data, the final shape is (195510, 13)
```

데이터 전처리

이상치 값 수정 및 범위 제한하기

위도와 경도가 0으로 움직이지 않는 값 존재 => 해당 행 범위에 넣지 않음
요금이 0인 행 해당 범위에 넣지 않음

```
1 print(f'Before dropping {uber_fare_df_1.shape}')
```

```
Before dropping (199999, 13)
```

```
[13]  1 #remove the data
      2 #remove the latitude and longitude ===000, passenger_count=0, fare<=0
      3
      4 uber_fare_df_1=uber_fare_df_1[~((uber_fare_df_1['pickup_latitude']==0.000000)&(uber_fare_df_1['pickup_longitude']==0.000000))]
      5 print(f'After dropping the data, the final shape is {uber_fare_df_1.shape}')
```

```
After dropping the data, the final shape is (196220, 13)
```

```
[14]  1 uber_fare_df_1=uber_fare_df_1[~(uber_fare_df_1['fare_amount']<=0)]
      2 uber_fare_df_1=uber_fare_df_1[~(uber_fare_df_1['passenger_count']==0)]
      3 print(f'After dropping the data, the final shape is {uber_fare_df_1.shape}')
```

```
After dropping the data, the final shape is (195511, 13)
```

```
[15]  1 uber_fare_df_1=uber_fare_df_1[~(uber_fare_df_1['passenger_count']==208)]
      2 print(f'After dropping the data, the final shape is {uber_fare_df_1.shape}')
```

```
After dropping the data, the final shape is (195510, 13)
```

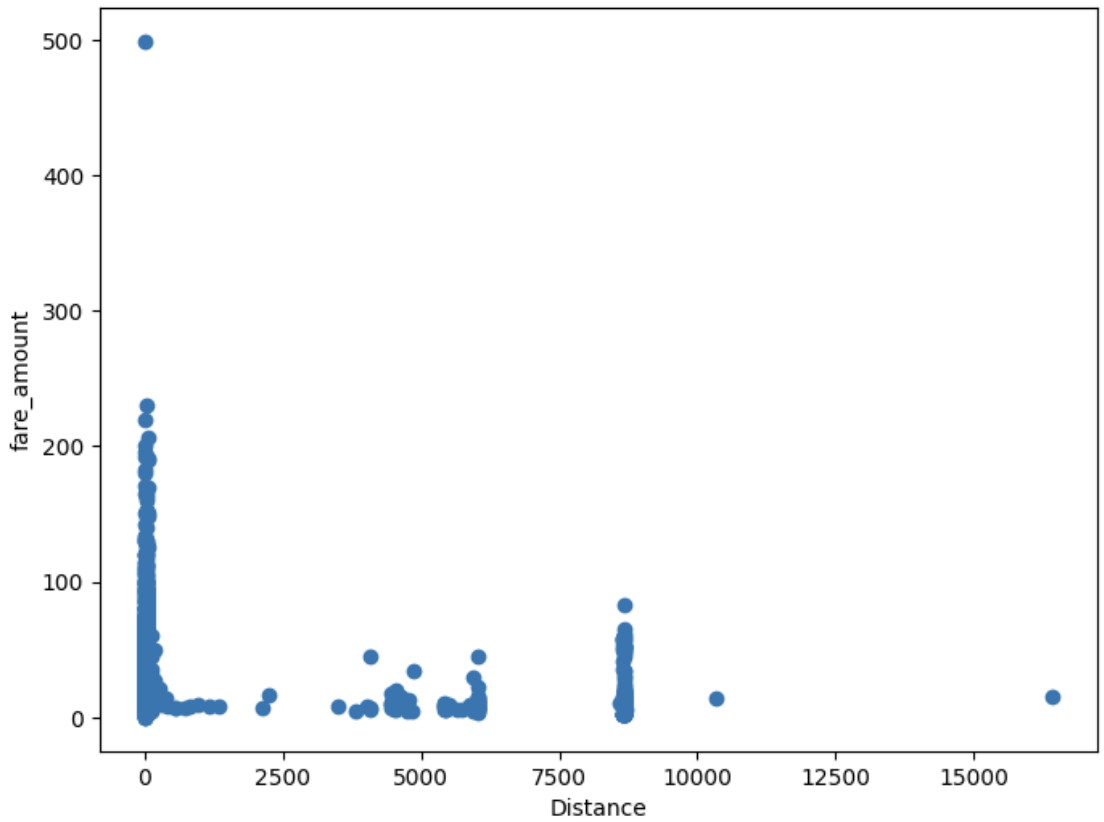
데이터 전처리

이상치 값 수정 및 범위 제한하기

['distance_travelled (KM)']
를 스캐터 플롯으로 살펴본 결과 데이터 범위 축소해 보이는게 좋겠음.

```
1 plt.figure(figsize=(8, 6))
2 plt.scatter(uber_fare_df_1['distance_travelled (KM)'],uber_fare_df_1['fare_amount'])
3 plt.xlabel("Distance")
4 plt.ylabel("fare_amount")
```

```
Text(0, 0.5, 'fare_amount')
```



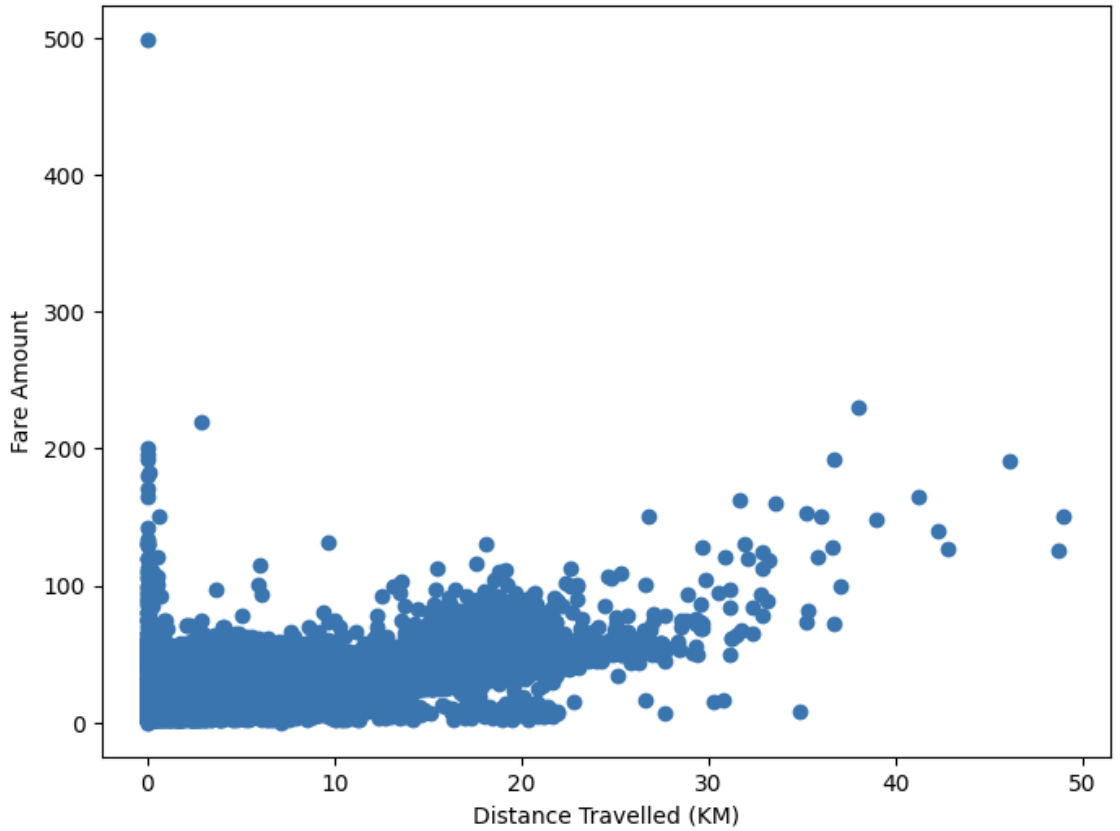
데이터 전처리

이상치 값 수정 및 범위 제한하기

데이터 범위 제한 결과
총 데이터의 개수(195209, 13)

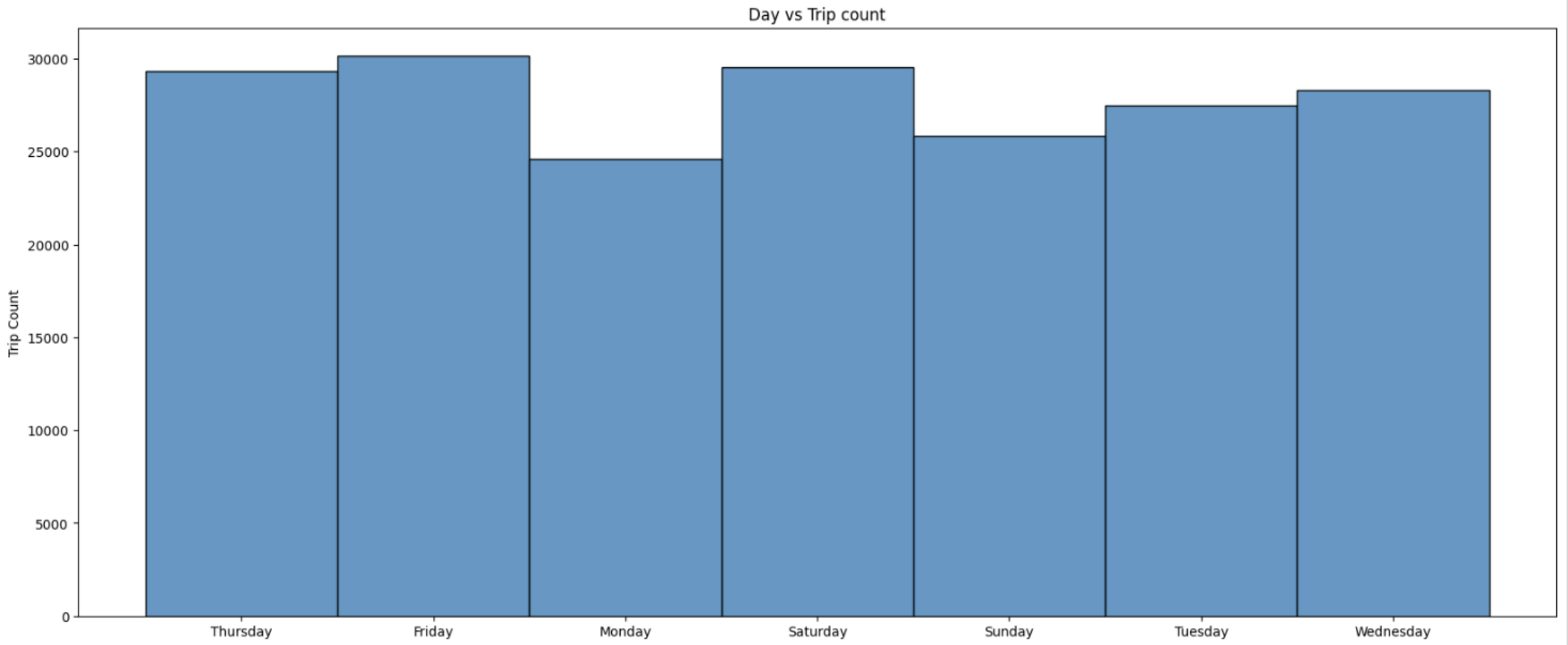
해당 플롯을 살펴본 결과 선형성을 띄우고 있는듯 함

```
[18]  1 plt.figure(figsize=(8, 6))
      2 plt.scatter(uber_fare_df_1['distance_travelled (KM)'],uber_fare_df_1['fare_amount'])
      3 plt.xlabel('Distance Travelled (KM)')
      4 plt.ylabel('Fare Amount')
      5 plt.show()
```

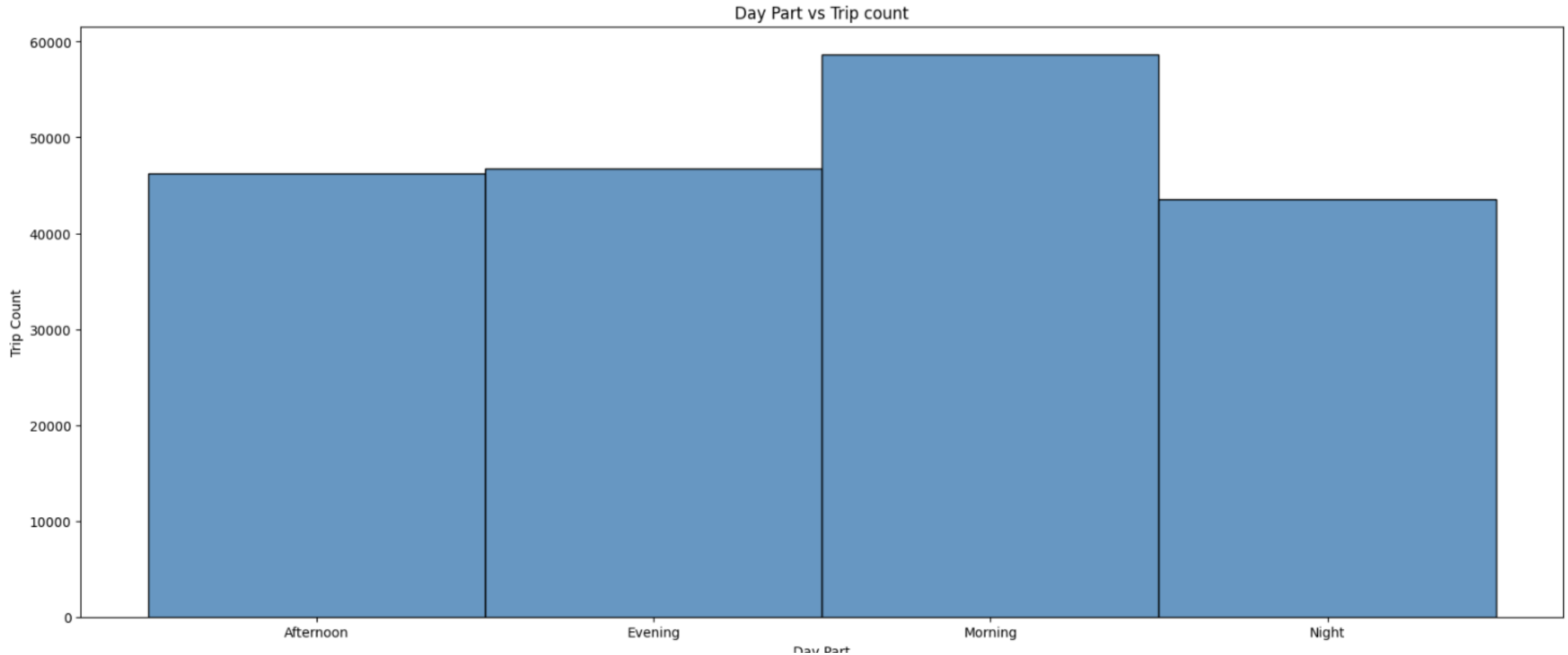


데이터 분석 및 시각화

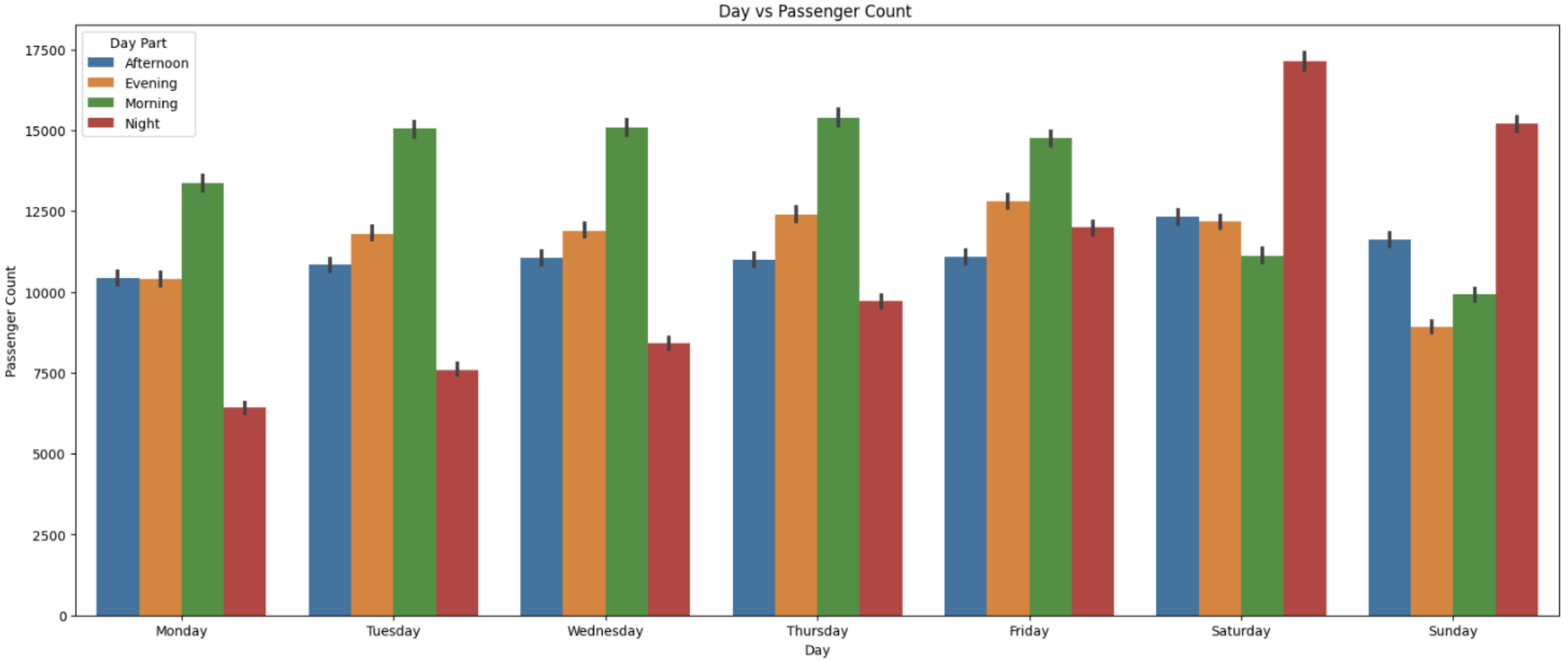
```
#study how much fare for each day in different parts of day
plt.figure(figsize=(20,8))
sb.histplot(uber_fare_df_1['Day'])
plt.ylabel('Trip Count')
plt.title('Day vs Trip count')
plt.show()
```



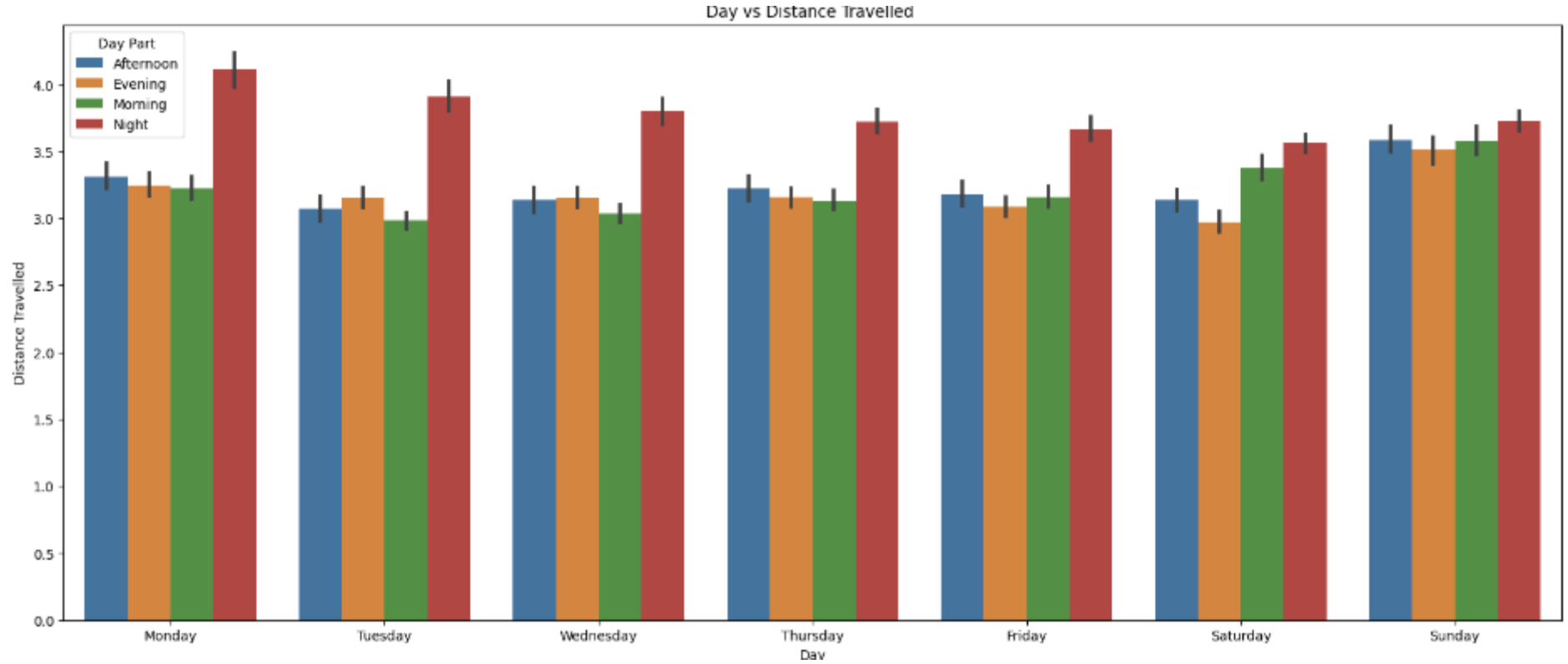
데이터 분석 및 시각화



데이터 분석 및 시각화



데이터 분석 및 시각화



데이터 분석 및 시각화

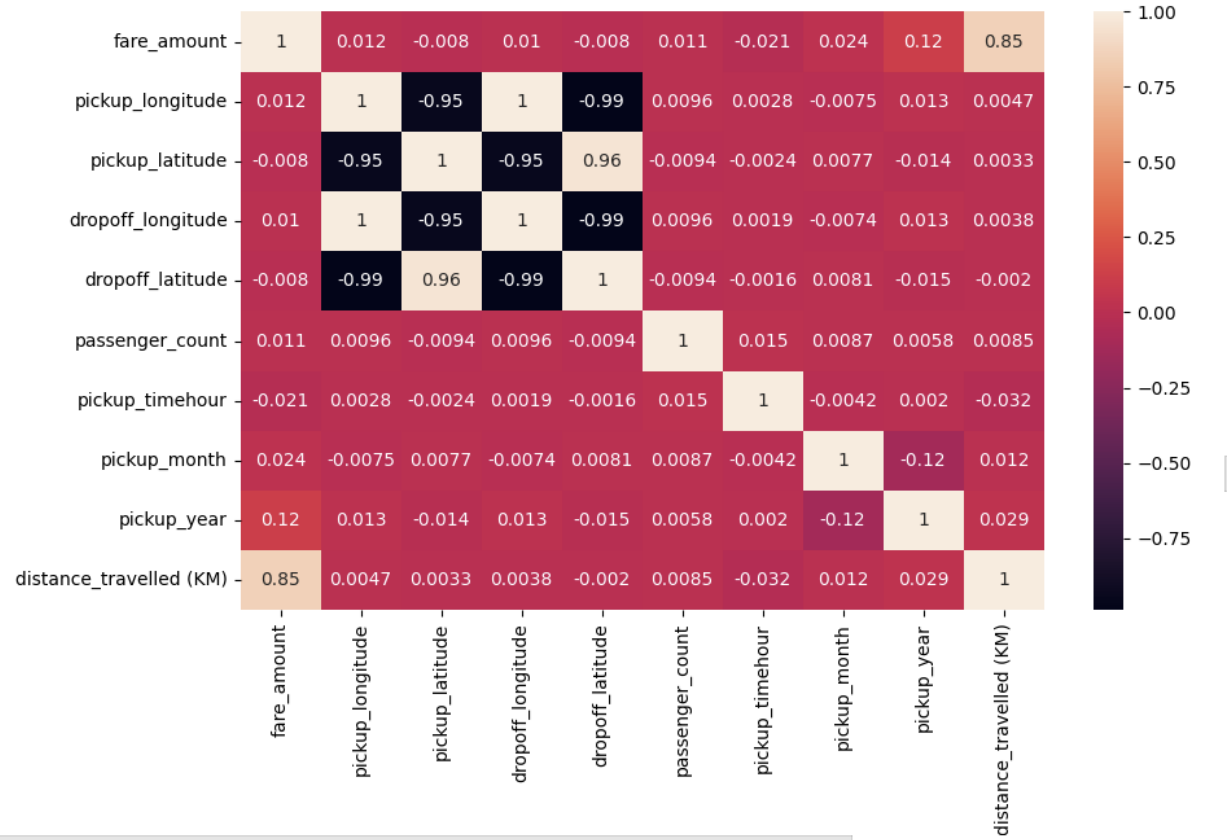
데이터 상관관계 확인

위도와 경도가 서로 강한 상관관계를 보이고 있으나
이는 distant로 차원축소하여 볼예정.

Distance_travelled(km)와 fare_amount가
가장 강한 상관관계를 보이고 있음을 알 수 있음

```
1 fig,axis = plt.subplots(figsize = (10,6))
2 sns.heatmap(uber_fare_df_1.corr(),annot = True) #Correlation Heatmap (Light values means highly correlated)
```

```
<ipython-input-30-5198f6808a1b>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versi
  sns.heatmap(uber_fare_df_1.corr(),annot = True) #Correlation Heatmap (Light values means highly correlated)
<Axes: >
```



데이터 전처리

이상치 값 수정 및 범위 제한하기

사이킷 런을 이용해
변수들의 값을 표준화 시키기

Standardization

For more accurate results on our linear regression model

Assigning the dependent and independent variable

```
[37]  1 X = uber_fare_df_1['distance_travelled (KM)'].values.reshape(-1, 1)        #Independent Variable
      2 y = uber_fare_df_1['fare_amount'].values.reshape(-1, 1)     #Dependent Variable
```

```
[38]  1 from sklearn.preprocessing import StandardScaler
      2 std = StandardScaler()
      3 y_std = std.fit_transform(y)
      4 print(y_std)
      5
      6 x_std = std.fit_transform(X)
      7 print(x_std)
```

```
[[-0.3936848 ]
 [-0.37322228]
 [ 0.15880317]
 ...
 [ 2.00042975]
 [ 0.32250331]
 [ 0.28157828]]
[[-0.45434404]
 [-0.23896147]
 [ 0.47839537]
 ...
 [ 2.65204678]
 [ 0.06205993]
 [ 0.58449346]]
```

분류모델 선정

Simple linear regression

앞선 요금과 거리와의 스캐터 플롯을 살펴본 결과
선형성을 띄고 있어 회귀문제의 가장 적절한 모델인
Linear regression사용

Linear regresiion 사용시
Train accuracy: 0.73
Test accuracy:0.74
Over fitting되지 않고
좋은 성능을 보이는 것이 확인됨

simple linear regression

```
[39]  1 from sklearn.model_selection import train_test_split
      2 X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2, random_state=0)
```

```
[40]  1 from sklearn.linear_model import LinearRegression
      2 l_reg = LinearRegression()
      3 l_reg.fit(X_train, y_train)
      4
      5 print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))
      6 print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

```
Training set score: 0.73
Test set score: 0.7495038
```

분류모델 선정

Simple linear regression

Accuracy 및
Mean squared error

Co-efficient확인

Accuracy Checking Finding the MSE,MAE, RMSE, etc.

```
1 from sklearn import metrics
2 print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
3 #print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
4 print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
5 print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 0.2479472187951637
Mean Squared Error: 0.24604492543779535
Root Mean Squared Error: 0.49602915785041846
```

Intercept and Co-efficient

```
[ ]  1 print(l_reg.intercept_)
     2 print(l_reg.coef_)
```

```
[-2.61492955e-05]
[[0.85329361]]
```

분류모델 선정

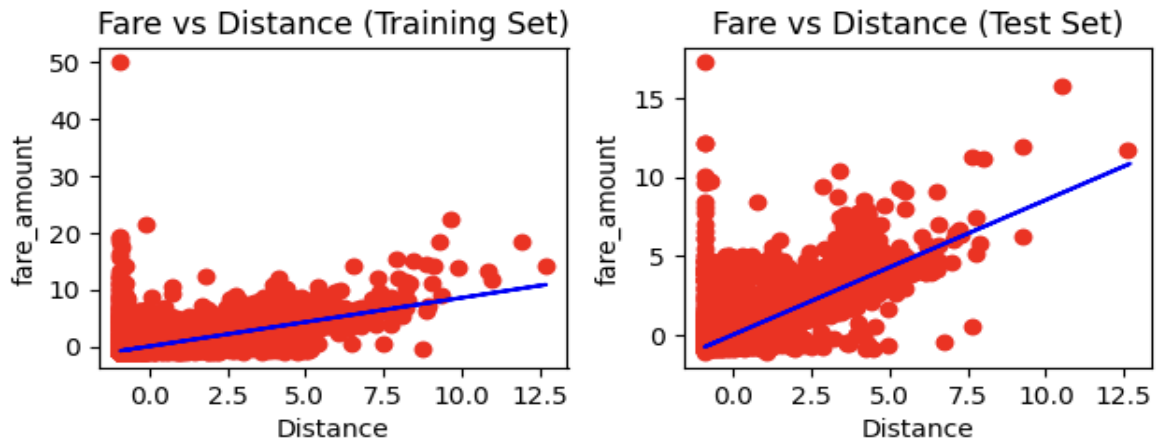
Simple linear regression

Training set과 test set
스캐터 플롯으로 확인

```
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")


plt.tight_layout()
plt.rcParams["figure.figsize"] = (32,22)
plt.show()
```



분류모델 선정

GridSearchCV를 활용하여 파라미터 그리드 탐색

```
Best Parameters: {'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 200}
Mean Squared Error: 28.903319986189512 Mean Absolute Error: 2.411841868645587
```

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
import numpy as np

# 데이터 준비
X = uber_fare_df_1['distance_travelled (KM)'].values.reshape(-1, 1)
y = uber_fare_df_1['fare_amount'].values.reshape(-1, 1)

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 모델 정의
model = RandomForestRegressor()

# 탐색할 파라미터 그리드 설정
param_grid = {
    'n_estimators': [100, 200],  # 의사 결정 트리의 개수
    'max_depth': [None, 5],  # 의사 결정 트리의 최대 깊이
    'min_samples_split': [2, 5]  # 내부 노드를 분할하기 위한 최소 샘플 수
}

# GridSearchCV를 사용하여 파라미터 그리드 탐색
grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)

# 최적의 모델과 파라미터 출력
print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

# 테스트 데이터에 대한 예측
y_pred = best_model.predict(X_test)

# 평가 지표 출력
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

모델의 결과 해석

프로젝트 결과

우버 요금제의 분석을 통한 우버 요금 예측

데이터 시각화를 통해
상관 관계 확인후
가장 강한 상관관계를 보이는
거리와 요금을 각각 독립변수와 종속변수 설정
⇒ 좋은 accuracy를 보임
⇒ 또한 overfitting도 나타나지 않음
⇒ 전처리를 확실히 했기 때문

결론

데이터 전처리
- 결측지 삭제
- 모델에 큰 상관 관계를 보이는 위도와 경도를 거리로 환산하여 km로 차원을 감소

변수선택
- 상관 관계가 높은 변수들에 대한 분석

적합한 모델 사용을 통한 성능 확인

파라미터 설정을 통한 성능 변화
- Gridsearch를 이용해 최적 모델 파라미터 교차 확인

해당 프로젝트 적용
- 콜택시 앱을 이용시
- 높은 정확도의 요금 계산 확인 가능

감사합니다.