```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing
```

## Loading the Dataset

First we load the dataset and find out the number of columns, rows, NULL values, etc.

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17848 entries, 0 to 17847
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         17848 non-null  int64
 1   key                17848 non-null  object
 2   fare_amount        17847 non-null  float64
 3   pickup_datetime    17847 non-null  object
 4   pickup_longitude   17847 non-null  float64
 5   pickup_latitude    17847 non-null  float64
 6   dropoff_longitude  17847 non-null  float64
 7   dropoff_latitude   17847 non-null  float64
 8   passenger_count    17847 non-null  float64
dtypes: float64(6), int64(1), object(2)
memory usage: 1.2+ MB
```

```
df.head()
```

| | Unnamed: 0 | key | fare_amount | pickup_datetime | pickup_longitude | pickup_l |
|---|---|---|---|---|---|---|
| 0 | 24238194 | 2015-05-07 19:52:06.0000003 | 7.5 | 2015-05-07 19:52:06 UTC | -73.999817 | 4( |
| 1 | 27835199 | 2009-07-17 20:04:56.0000002 | 7.7 | 2009-07-17 20:04:56 UTC | -73.994355 | 4( |
| 2 | 44984355 | 2009-08-24 21:45:00.00000061 | 12.9 | 2009-08-24 21:45:00 UTC | -74.005043 | 4( |
| 3 | 25894730 | 2009-06-26 08:22:21.0000001 | 5.3 | 2009-06-26 08:22:21 UTC | -73.976124 | 4( |
| 4 | 17610152 | 2014-08-28 17:47:00.000000188 | 16.0 | 2014-08-28 17:47:00 UTC | -73.925023 | 4( |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.describe()
```

| | Unnamed: 0 | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude |
|---|---|---|---|---|---|
| count | 1.784800e+04 | 17847.000000 | 17847.000000 | 17847.000000 | 17847.000000 |
| mean | 2.765310e+07 | 11.417429 | -72.595005 | 39.951854 | -72.580938 |
| std | 1.599173e+07 | 10.173691 | 11.458450 | 6.095753 | 10.197475 |
| min | 4.800000e+01 | 2.500000 | -748.016667 | -74.009697 | -75.350437 |
| 25% | 1.383501e+07 | 6.000000 | -73.992000 | 40.734977 | -73.991591 |
| 50% | 2.755475e+07 | 8.500000 | -73.981823 | 40.752377 | -73.980073 |
| 75% | 4.140304e+07 | 12.500000 | -73.967328 | 40.767152 | -73.963307 |
| max | 5.542169e+07 | 350.000000 | 40.770667 | 41.366138 | 40.828377 |

# Cleaning

```
df = df.drop(['Unnamed: 0', 'key'], axis=1)
```

```
df.isna().sum()
```

| | 0 |
|---|---|
| **fare_amount** | 1 |
| **pickup_datetime** | 1 |
| **pickup_longitude** | 1 |
| **pickup_latitude** | 1 |
| **dropoff_longitude** | 1 |
| **dropoff_latitude** | 1 |
| **passenger_count** | 1 |

**dtype:** int64

## Remove null rows

```
df.dropna(axis=0,inplace=True)
```

```
df.dtypes
```

| | 0 |
|---|---|
| **fare_amount** | float64 |
| **pickup_datetime** | object |
| **pickup_longitude** | float64 |
| **pickup_latitude** | float64 |
| **dropoff_longitude** | float64 |
| **dropoff_latitude** | float64 |
| **passenger_count** | float64 |

**dtype:** object

## Fix data type of `pickup_datetime` from `Object` to `DateTime`

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```

## Separating the date and time into separate columns for more usability.

```
df= df.assign(
    second = df.pickup_datetime.dt.second,
    minute = df.pickup_datetime.dt.minute,
    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
)
df = df.drop('pickup_datetime',axis=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 17847 entries, 0 to 17846
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   fare_amount        17847 non-null  float64
 1   pickup_longitude   17847 non-null  float64
 2   pickup_latitude    17847 non-null  float64
 3   dropoff_longitude  17847 non-null  float64
 4   dropoff_latitude   17847 non-null  float64
 5   passenger_count    17847 non-null  float64
 6   second             17847 non-null  int32
 7   minute             17847 non-null  int32
 8   hour               17847 non-null  int32
 9   day                17847 non-null  int32
 10  month              17847 non-null  int32
 11  year               17847 non-null  int32
 12  dayofweek          17847 non-null  int32
dtypes: float64(6), int32(7)
memory usage: 1.4 MB
```

```
df.head()
```

| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude |
|---|---|---|---|---|---|
| **0** | 7.5 | -73.999817 | 40.738354 | -73.999512 | 40.723217 |
| **1** | 7.7 | -73.994355 | 40.728225 | -73.994710 | 40.750325 |
| **2** | 12.9 | -74.005043 | 40.740770 | -73.962565 | 40.772647 |
| **3** | 5.3 | -73.976124 | 40.790844 | -73.965316 | 40.803349 |
| **4** | 16.0 | -73.925023 | 40.744085 | -73.973082 | 40.761247 |

Next steps: Generate code with df | View recommended plots | New interactive sheet

## Haversine Formula

Calculatin the distance between the pickup and drop co-ordinates using the Haversine formual for accuracy.

$$d = 2rsin^{-1}\left(\sqrt{sin^2\left(\frac{\Phi_2-\Phi_1}{2}\right) + cos(\Phi_1)cos(\Phi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\ \right)$$

```
incorrect_coordinates = df.loc[
    (df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
    (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
    (df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
    (df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')


def distance_transform(longitude1, latitude1, longitude2, latitude2):
    long1, lati1, long2, lati2 = map(np.radians, [longitude1, latitude1, longitude2, latituc
    dist_long = long2 - long1
    dist_lati = lati2 - lati1
    a = np.sin(dist_lati/2)**2 + np.cos(lati1) * np.cos(lati2) * np.sin(dist_long/2)**2
    c = 2 * np.arcsin(np.sqrt(a)) * 6371
    # long1,lati1,long2,lati2 = longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos
    # c = sqrt((long2 - long1) ** 2 + (lati2 - lati1) ** 2)asin

    return c


df['Distance'] = distance_transform(
    df['pickup_longitude'],
    df['pickup_latitude'],
    df['dropoff_longitude'],
    df['dropoff_latitude']
)


df.head()
```

| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude |
|---|---|---|---|---|---|
| **0** | 7.5 | -73.999817 | 40.738354 | -73.999512 | 40.723217 |
| **1** | 7.7 | -73.994355 | 40.728225 | -73.994710 | 40.750325 |
| **2** | 12.9 | -74.005043 | 40.740770 | -73.962565 | 40.772647 |
| **3** | 5.3 | -73.976124 | 40.790844 | -73.965316 | 40.803349 |
| **4** | 16.0 | -73.925023 | 40.744085 | -73.973082 | 40.761247 |

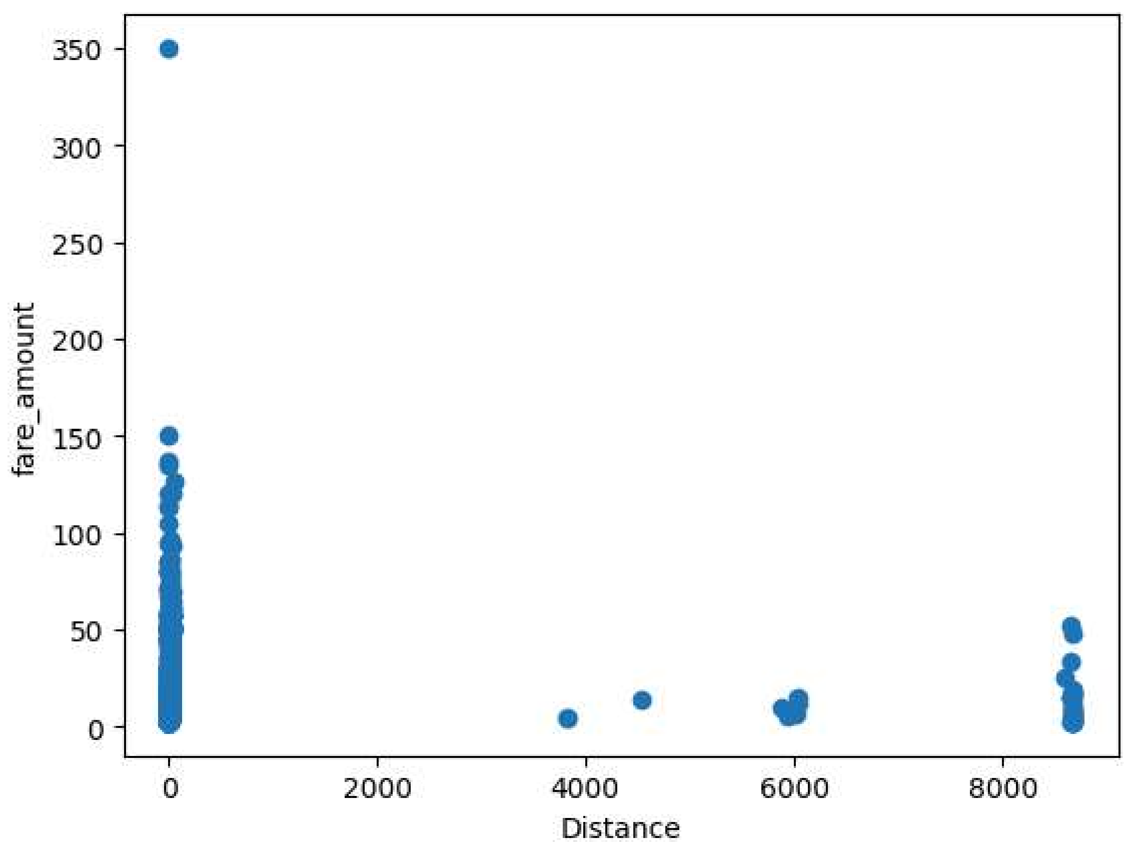
Next steps: Generate code with `df` View recommended plots New interactive sheet

## Outliers

We can get rid of the trips with very large distances that are outliers as well as trips with 0 distance.

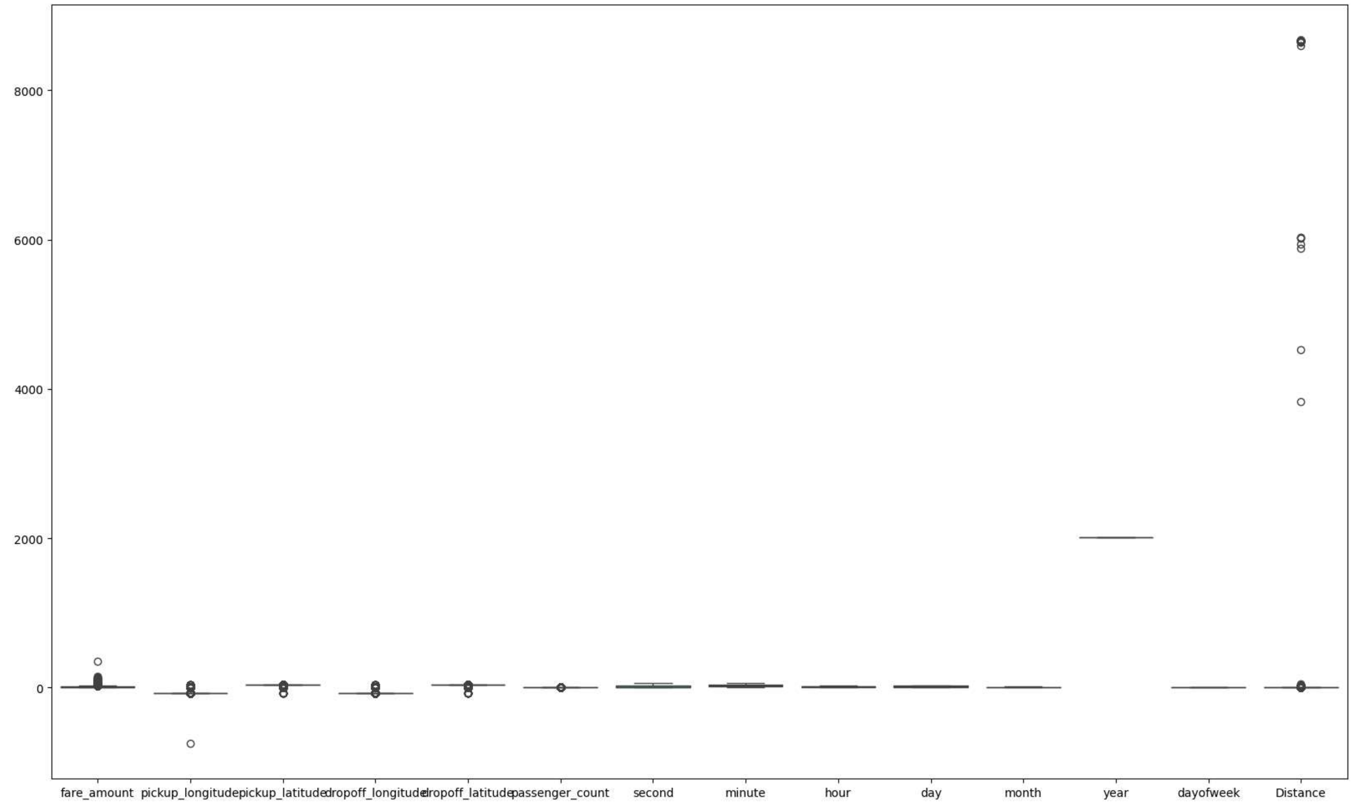
```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

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



```
plt.figure(figsize=(20,12))
sns.boxplot(data = df)
```

```
<Axes: >
```

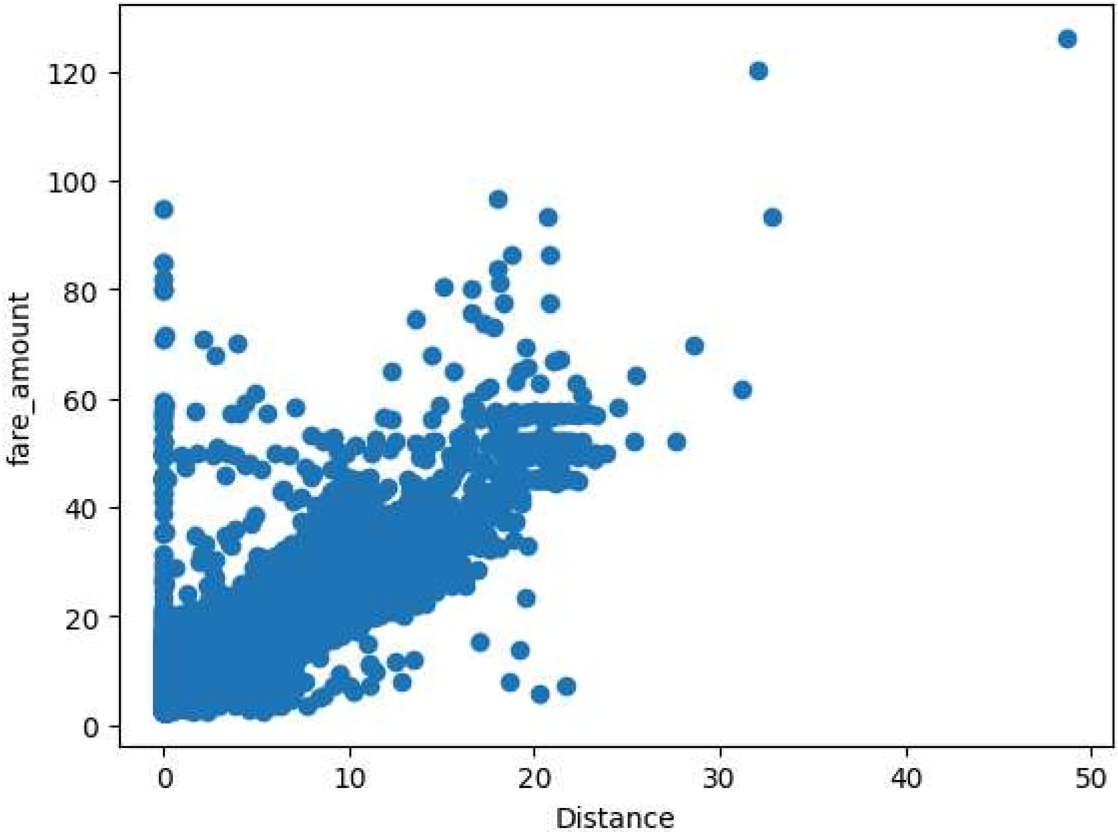



```
df.drop(df[df['Distance'] >= 60].index, inplace = True)
df.drop(df[df['fare_amount'] <= 0].index, inplace = True)

df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace = True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index, inplace = True )
```

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

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




## Coorelation Matrix

To find the two variables that have the most inter-dependence

```
corr = df.corr()

corr.style.background_gradient(cmap='BuGn')
```

| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dro |
|---|---|---|---|---|---|
| **fare_amount** | 1.000000 | 0.011058 | -0.010620 | 0.010744 | |
| **pickup_longitude** | 0.011058 | 1.000000 | -0.978635 | 0.999992 | |
| **pickup_latitude** | -0.010620 | -0.978635 | 1.000000 | -0.978642 | |
| **dropoff_longitude** | 0.010744 | 0.999992 | -0.978642 | 1.000000 | |
| **dropoff_latitude** | -0.010569 | -0.978618 | 0.999987 | -0.978626 | |
| **passenger_count** | 0.008514 | 0.005076 | -0.009071 | 0.005062 | |
| **second** | -0.006353 | -0.018881 | 0.021698 | -0.018794 | |
| **minute** | -0.007230 | 0.011759 | -0.011255 | 0.011740 | |
| **hour** | -0.003587 | -0.003357 | 0.007316 | -0.003647 | |
| **day** | -0.001046 | 0.007920 | -0.012252 | 0.007937 | |
| **month** | 0.029099 | -0.014126 | 0.014966 | -0.014099 | |
| **year** | 0.124357 | 0.002281 | -0.005746 | 0.002297 | |
| **dayofweek** | 0.011921 | -0.016376 | 0.012885 | -0.016308 | |
| **Distance** | 0.855590 | -0.111318 | 0.101323 | -0.111493 | |

# Standardization

For more accurate results on our linear regression model

```
X = df['Distance'].values.reshape(-1, 1)        #Independent Variable
y = df['fare_amount'].values.reshape(-1, 1)     #Dependent Variable


from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(y)
print(y_std)

x_std = std.fit_transform(X)
print(x_std)
```

```
[[-0.40180516]
 [-0.38094327]
 [ 0.16146609]
 ...
 [ 1.58007517]
 [-0.67300984]]
```

```
 [-0.03672194]]
[[-0.43899682]
 [-0.22366223]
 [ 0.49353483]
 ...
 [ 1.69990844]
 [-0.42274548]
 [-0.64312305]]
```

## Splitting the Dataset

Training and Test Set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2, random_stat
```

## Simple Linear Regression

Training the simple linear regression model on the training set

```
from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

```
Training set score: 0.73
Test set score: 0.7316693
```

```
y_pred = l_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result[['Predicted']] = y_pred

result.sample(10)
```

| | Actual | Predicted |
|---|---|---|
| **3101** | 1.079390 | 1.160677 |
| **540** | 0.453533 | 0.242715 |
| **1738** | -0.923353 | -0.775808 |
| **851** | -0.547838 | -0.369444 |
| **3192** | -0.506115 | -0.478435 |
| **407** | 1.162837 | 1.863157 |
| **1794** | -0.464391 | -0.379238 |
| **1199** | -0.766888 | -0.639975 |
| **3288** | -0.506115 | -0.368911 |
| **2955** | 0.328361 | -0.253541 |

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred)))
```

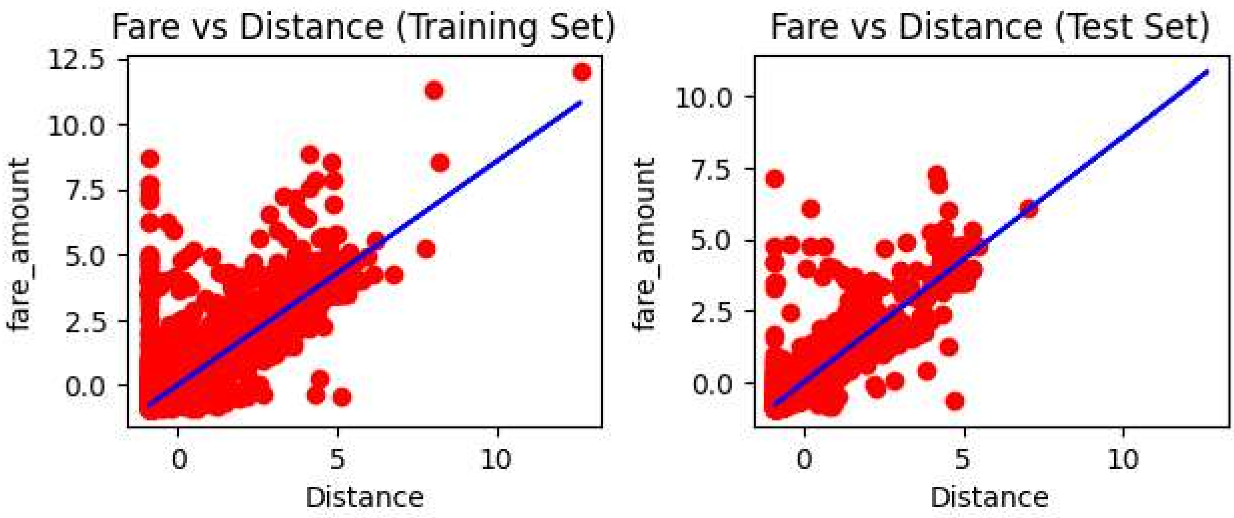
```
Mean Absolute Error: 0.26722036563873924
Mean Absolute % Error: 1.3303893172593673
Mean Squared Error: 0.2556415197241215
Root Mean Squared Error: 0.5056100470957055
R Squared (R²): 0.8553767201239891
```

## Visualization

```
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.show()
```




```
cols = ['Model', 'RMSE', 'R-Squared']

# create a empty dataframe of the colums
# columns: specifies the columns to be selected
result_tabulation = pd.DataFrame(columns = cols)

# compile the required information
linreg_metrics = pd.DataFrame([[
     "Linear Regresion model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
     np.sqrt(metrics.r2_score(y_test, y_pred))
]], columns = cols)

result_tabulation = pd.concat([result_tabulation, linreg_metrics], ignore_index=True)

result_tabulation
```

```
<ipython-input-30-2d9e8cc5ba0b>:14: FutureWarning: The behavior of DataFrame concatenati
  result_tabulation = pd.concat([result_tabulation, linreg_metrics], ignore_index=True)
```

| | Model | RMSE | R-Squared |
|---|---|---|---|
| **0** | Linear Regresion model | 0.50561 | 0.855377 |

## RandomForestRegressor

Training the RandomForestRegressor model on the training set

```
rf_reg = RandomForestRegressor(n_estimators=100, random_state=10)

# fit the regressor with training dataset
rf_reg.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A c
  return fit_method(estimator, *args, **kwargs)
```

RandomForestRegressor

RandomForestRegressor(random_state=10)

```
# predict the values on test dataset using predict()
y_pred_RF = rf_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result['Predicted'] = y_pred_RF

result.sample(10)
```

| | Actual | Predicted |
|---|---|---|
| **2077** | -0.339219 | 0.018979 |
| **404** | 0.286637 | -0.034636 |
| **821** | -0.401805 | -0.643803 |
| **1902** | 0.380516 | 0.043388 |
| **1699** | 0.078018 | 0.055099 |
| **1803** | -0.297496 | -0.202261 |
| **1461** | 1.663523 | 1.218121 |
| **1559** | -0.756457 | -0.558009 |
| **3388** | 0.161466 | 0.037964 |
| **2362** | 0.536980 | -0.680729 |

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_RF))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred_RF))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_RF))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred_RF)))
```

```
Mean Absolute Error: 0.3052581448929417
Mean Absolute % Error: 1.4662409641542689
Mean Squared Error: 0.3075490233637449
Root Mean Squared Error: 0.554571026437322
```

```
R Squared (R²): 0.8229127071359651
```

## Visualization

```
# Build scatterplot
plt.scatter(X_test, y_test, c = 'b', alpha = 0.5, marker = '.', label = 'Real')
plt.scatter(X_test, y_pred_RF, c = 'r', alpha = 0.5, marker = '.', label = 'Predicted')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.grid(color = '#D3D3D3', linestyle = 'solid')
plt.legend(loc = 'lower right')


plt.tight_layout()
plt.show()
```