```
import numpy as np
import pandas as pd
```

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

```
df
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40
...	...	...	...	...	...	...	...	
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73.986525	40
		2014-03-14		2014-03-14				

```
df.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitud
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.72321
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.75032
2	44984355	2009-08-24	12.9	2009-08-24	-74.005043	40.740770	-73.962565	40.77264

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

```
df.shape
```

```
(200000, 9)
```

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

	0
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

```
cdf=df
```

```
cdf['dropoff_latitude'].fillna(cdf['dropoff_latitude'].mean(),inplace=True)
cdf['dropoff_longitude'].fillna(cdf['dropoff_longitude'].mean(),inplace=True)
```

```
<ipython-input-93-a95f59353937>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are settin

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[c


  cdf['dropoff_latitude'].fillna(cdf['dropoff_latitude'].mean(),inplace=True)
<ipython-input-93-a95f59353937>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are settin

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[c


  cdf['dropoff_longitude'].fillna(cdf['dropoff_longitude'].mean(),inplace=True)
```

```
cdf.isnull().sum()
```

	0
Unnamed: 0	0
key	0
fare_amount	0
pickup_datetime	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
passenger_count	0

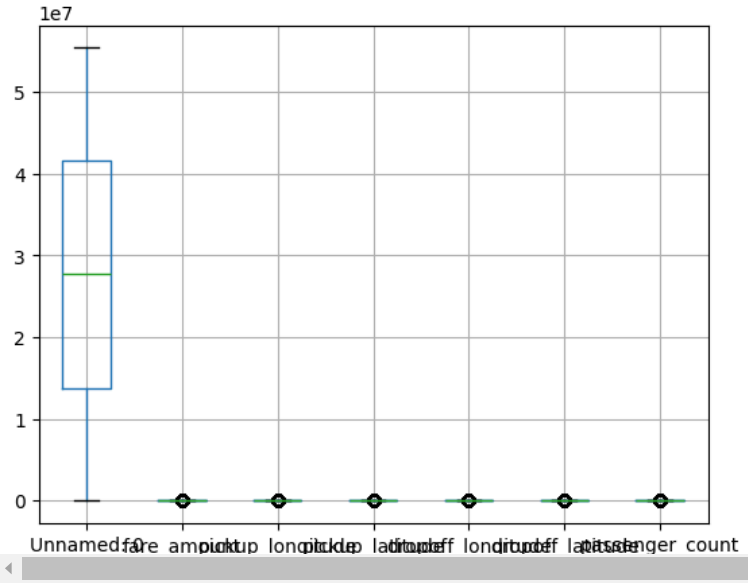
```
cdf.shape
```

```
(200000, 9)
```

Outliers Removal

```
cdf.boxplot()
```

```
<Axes: >
```



Harvesian Formula

```
plong=np.radians(cdf['pickup_longitude'])
plat=np.radians(cdf['pickup_latitude'])
dlong=np.radians(cdf['dropoff_longitude'])
dlat=np.radians(cdf['dropoff_latitude'])
```

```
d1=(dlat-plat)/2
d2=(dlong-plong)/2
```

```
exp=np.sqrt(d1**2+np.cos(plat)*np.cos(dlat)*d2**2)
```

```
dlat
```

	dropoff_latitude
0	0.710754
1	0.711227
2	0.711617
3	0.712153
4	0.711418
...	...
199995	0.711052
199996	0.711041
199997	0.710220
199998	0.710269
199999	0.711550

200000 rows × 1 columns

```
d=2*6356.7*np.arcsin(exp)
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:399: RuntimeWarning: invalid value encountered in arcsin
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

```
d
```

	0
0	1.679544
1	2.452074
2	5.025073
3	1.657954
4	4.465405
...	...
199995	0.111958
199996	1.870842
199997	12.821478
199998	3.531770
199999	5.405623

200000 rows × 1 columns

```
cdf.insert(len(cdf.columns),'distance',d)
```

```
cdf
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40
...	...	...	...	...	...	...	...	
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73.986525	40
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74.006672	40
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73.858957	40
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-73.983215	40
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-73.985508	40

200000 rows × 10 columns

```
odf=cdf
```

```
odf=odf.drop(['key','pickup_datetime'],axis=1)
```

```
odf.shape
```

```
(200000, 8)
```

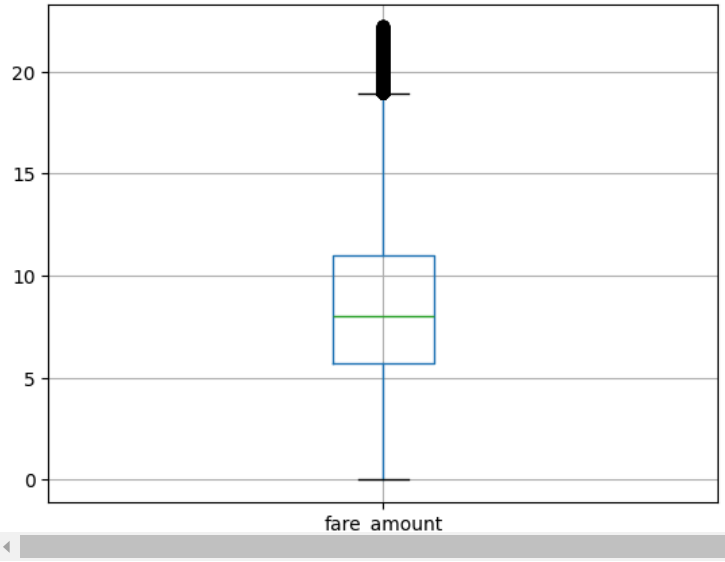
```
Q1 = odf['fare_amount'].quantile(0.25)
Q3 = odf['fare_amount'].quantile(0.75)
IQR = Q3 - Q1
Lower_limit = Q1 - 1.5 * IQR
Upper_limit = Q3 + 1.5 * IQR
print(f'Q1 = {Q1}, Q3 = {Q3}, IQR = {IQR}, Lower_limit = {Lower_limit},Upper_limit = {Upper_limit}')
```

```
Q1 = 6.0, Q3 = 12.5, IQR = 6.5, Lower_limit = -3.75,Upper_limit = 22.25
```

```
odf=odf[(odf['fare_amount']>=0) & (odf['fare_amount']<=Upper_limit)]
```

```
odf.boxplot(['fare_amount'])
```

```
<Axes: >
```




```
q1 = odf['distance'].quantile(0.25)
q3 = odf['distance'].quantile(0.75)
iqr = q3 - q1
Lower_limit = q1 - 1.5 * iqr
Upper_limit = q3 + 1.5 * iqr
print(f'Q1 = {q1}, Q3 = {q3}, IQR = {iqr}, Lower_limit = {Lower_limit},Upper_limit = {Upper_limit}')
```

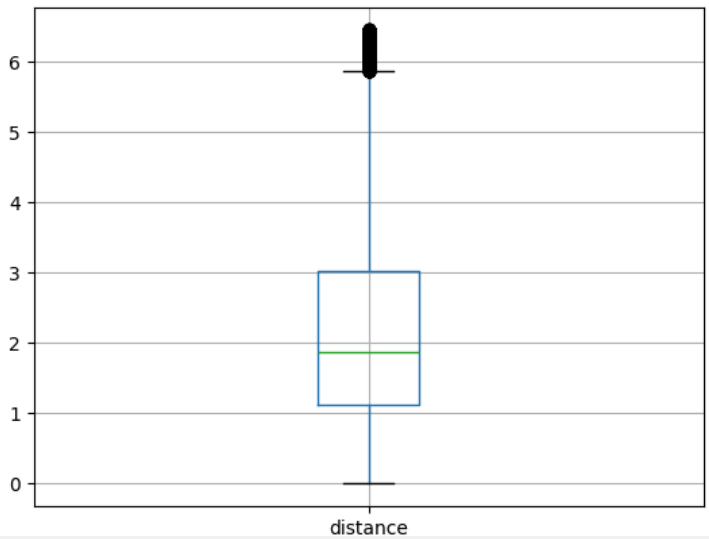
```
Q1 = 1.1598913695422408, Q3 = 3.275213239465648, IQR = 2.1153218699234073, Lower_limit = -2.01309143534287,Upper_limit = 6.4481960
```

```
import seaborn as sns
```

```
odf=odf[(odf['distance']>=0) & (odf['distance']<=Upper_limit)]
```

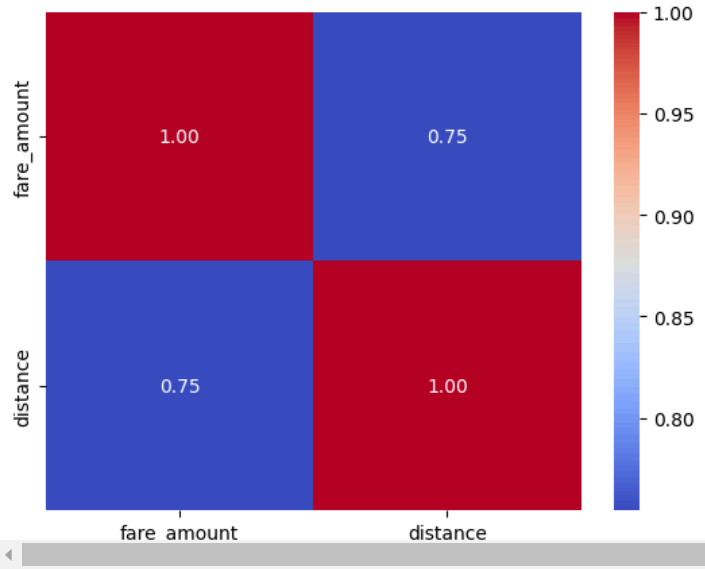
```
odf.boxplot(['distance'])
```

```
<Axes: >
```



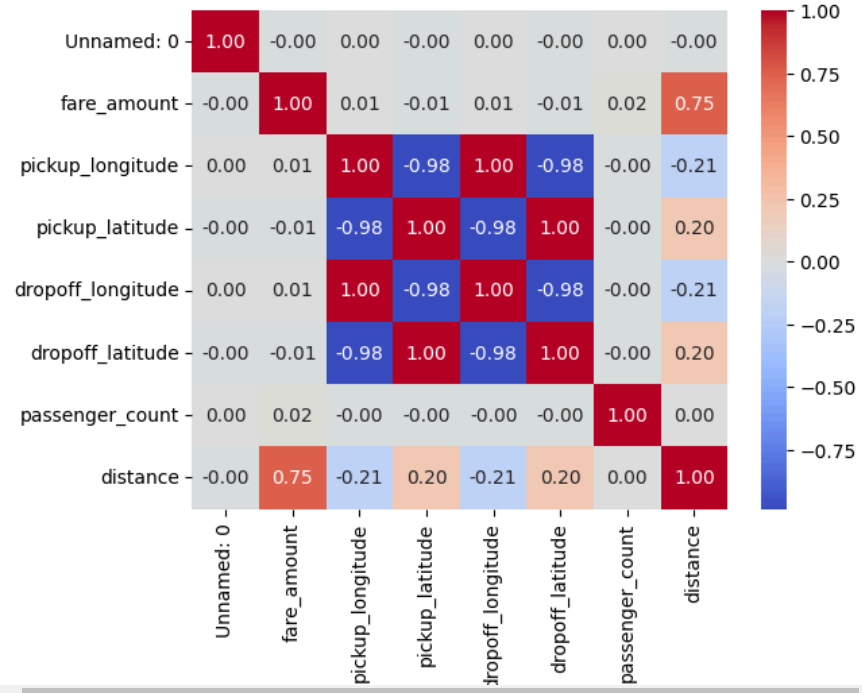

```
cm=odf[['fare_amount','distance']]
com=cm.corr()

print(com)
sns.heatmap(com,annot=True,cmap='coolwarm',fmt='.2f')
```

```
             fare_amount  distance
fare_amount     1.000000  0.754558
distance        0.754558  1.000000
<Axes: >
```




```
sns.heatmap(odf.corr(),annot=True,cmap='coolwarm',fmt='.2f')
```

```
<Axes: >
```




```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
X=odf['distance']
Y=odf['fare_amount']
```

```
x_train,x_test,y_train,y_test = train_test_split(X,Y,random_state=0,train_size=0.2)
```

```
lm=LinearRegression()
```

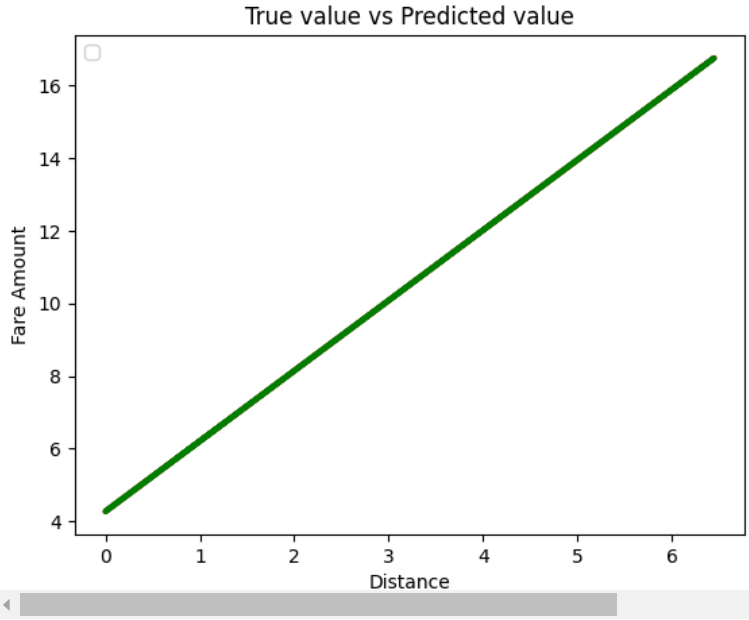
```
model=lm.fit(x_train.values.reshape(-1,1),y_train)
```

```
y_train_predict=lm.predict(x_train.values.reshape(-1,1))
y_test_predict=lm.predict(x_test.values.reshape(-1,1))
```

```
import matplotlib.pyplot as plt
```

```
plt.xlabel('Distance')
plt.ylabel('Fare Amount')
plt.title("True value vs Predicted value")
plt.legend(loc= 'upper left')
plt.plot(x_test,y_test_predict,color='red',linewidth=3)
plt.plot(x_train,y_train_predict,color='green',linewidth=3)
plt.xticks()
plt.yticks()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore a
```




```
#Evaluation Metrics
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('MSE is {}'.format(mse))
print("\n")
```

```
The model performance for training set
--------------------------------------
MSE is 6.0377253838299225
```

```
mse = mean_squared_error(y_test, y_test_predict)
print("The model performance for testing set")
print("--------------------------------------")
print('MSE is {}'.format(mse))
print("\n\n\n")
```

```
The model performance for testing set
--------------------------------------
MSE is 5.974305409153931
```

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('MAE is {}'.format(mae))
print("\n")

print("The model performance for testing set")
print("--------------------------------------")
mae = mean_absolute_error(y_test, y_test_predict)
print('MAE is {}'.format(mae))
print("\n\n\n")
```

```
The model performance for training set
--------------------------------------
MAE is 1.7506610309033295
```

```
The model performance for testing set
--------------------------------------
MAE is 1.7483451938415613
```

```
from sklearn.metrics import r2_score
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = r2_score(y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 2.4571783378155363
R2 score is 0.5643617767200362
```

```
rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
r2 = r2_score(y_test, y_test_predict)
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for testing set
--------------------------------------
RMSE is 2.4442392291168904
R2 score is 0.5705918454686051
```

Random Forest Regression

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

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(x_train.values.reshape(-1, 1), x_train.values.reshape(-1, 1))
y_pred_rf = rf_model.predict(x_test.values.reshape(-1, 1))
```