Freight Cost Prediction In Supply chain Management

Optimizing Freight Costs in Supply Chain Management

Freight cost plays a vital role in supply chain management, directly impacting logistics expenses and operational efficiency. Accurate cost prediction enables businesses to optimize transportation expenses, enhance decision-making, and maximize profitability.

Role of Machine Learning in Cost Prediction

- Traditional methods often require human intervention for rate adjustments, increasing the risk of errors and inconsistencies.Manual or rule-based cost estimation methods struggle to handle large datasets, leading to oversimplified calculations and inaccurate predictions.
- Machine learning models can analyze large datasets and identify patterns to predict freight costs more effectively.

Tools and Methods

Language: Python
Libraries:Pandas, Scikit- Learn, Matplotlib, Seaborn
Models: Linear Regression, SVM, Decision Trees, Random Forest,Catboost
Evaluation Metrics: R^2 score, Hyperparameter tuning

Model Comparison

Algorithm	R Square value
Linear Regression	0.60
SVM Linear	0.86
Random Forest	0.84
Cat Boost	0.84

Best Model :

SVM Linear (High accuracy and interpretability)

Supply Chain Freight Cost Analytics

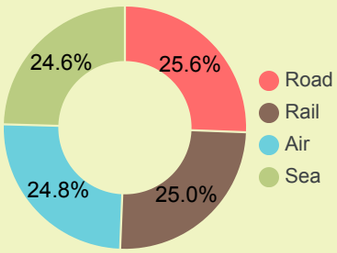
Total Cost in USD

Freight cost
7,778,032.25

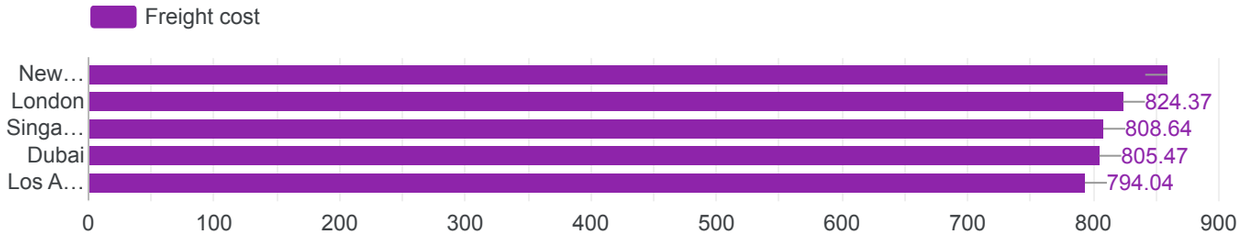
Average Weight

Weight
2,495.04

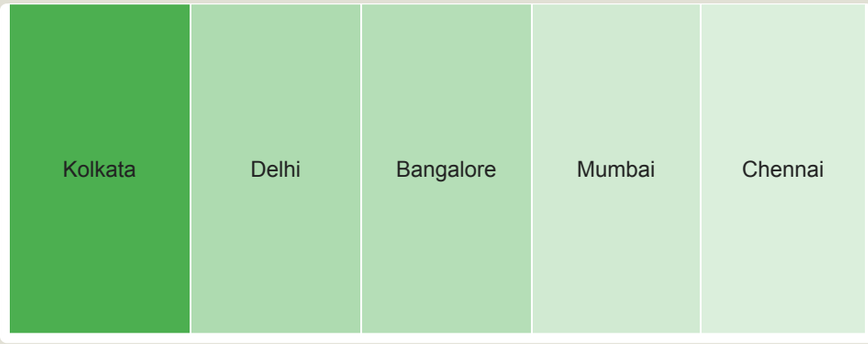
Mode of Transport



Average Cost per Destination



Point of Origin

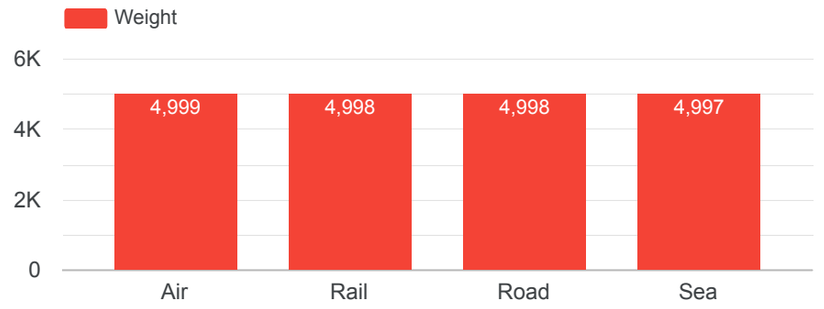


Average Cost

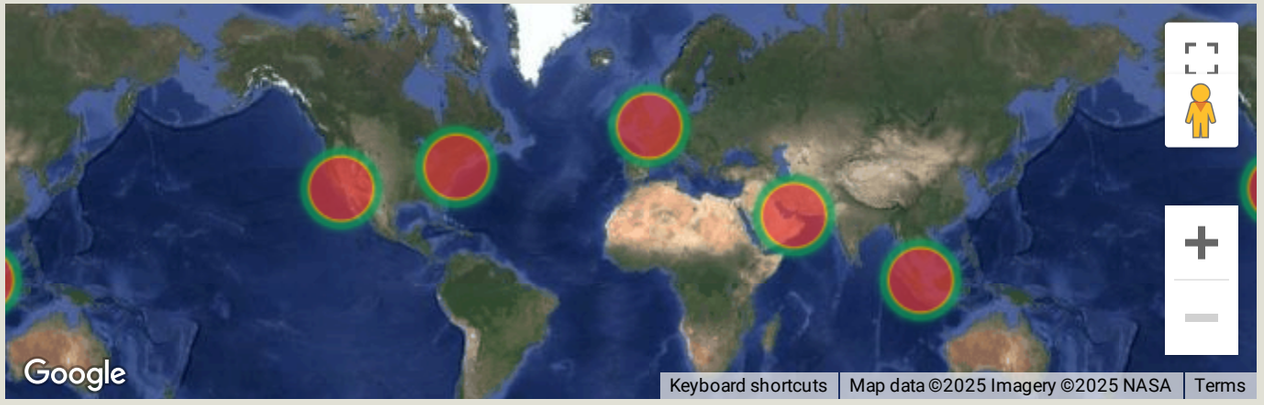
	Mode_of_Transport	Freight cost
1.	Air	3,472,880.75
2.	Sea	1,737,412.51
3.	Rail	1,377,164.78
4.	Road	1,190,574.21

1 - 4 / 4

Maximum Weight in each transport



Destination Heat Map



Total Shipments

Mode_of_Transport	Count of Shipment
Sea	12,184,139
Rail	12,428,085
Air	12,602,693
Road	12,790,083
Grand total	**50,005,000**

In [1]:
```python
import pandas as pd
import numpy as np
```

Data Collection

In [2]:
```python
data=pd.read_csv("Raw data.csv")
data
```

Out[2]:

	Shipment_ID	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freig
0	SHIP00001	NaN	1341.0	Rail	Kolkata	London	1.89	1.042210	Medium	8	2	326.35	
1	SHIP00002	3822.0	3538.0	Sea	Kolkata	Los Angeles	0.75	1.075883	High	8	3	293.82	
2	SHIP00003	3142.0	4777.0	Road	Chennai	London	1.99	1.170771	Low	3	2	52.03	
3	SHIP00004	516.0	3704.0	Air	Chennai	Dubai	1.58	1.129244	Low	8	3	421.87	
4	SHIP00005	4476.0	578.0	Sea	Mumbai	Dubai	NaN	0.998045	High	5	4	428.14	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	SHIP09996	284.0	633.0	Air	Bangalore	Los Angeles	0.52	0.977983	Medium	2	4	309.59	
9996	SHIP09997	NaN	2859.0	Rail	Kolkata	Los Angeles	1.39	1.091401	Medium	8	1	192.02	
9997	SHIP09998	1958.0	705.0	Air	Chennai	New York	NaN	0.971334	Medium	8	1	125.19	
9998	SHIP09999	1467.0	4931.0	Rail	Kolkata	Singapore	1.56	0.973844	High	1	1	107.21	
9999	SHIP10000	4969.0	NaN	Road	Chennai	Dubai	1.20	1.058222	High	7	1	209.60	

10000 rows × 14 columns

In [3]:
```python
data.shape
```

Out[3]:
```
(10000, 14)
```

In [4]:
```python
data.head()
```

Out[4]:

	Shipment_ID	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freight_In
0	SHIP00001	NaN	1341.0	Rail	Kolkata	London	1.89	1.042210	Medium	8	2	326.35	
1	SHIP00002	3822.0	3538.0	Sea	Kolkata	Los Angeles	0.75	1.075883	High	8	3	293.82	
2	SHIP00003	3142.0	4777.0	Road	Chennai	London	1.99	1.170771	Low	3	2	52.03	
3	SHIP00004	516.0	3704.0	Air	Chennai	Dubai	1.58	1.129244	Low	8	3	421.87	
4	SHIP00005	4476.0	578.0	Sea	Mumbai	Dubai	NaN	0.998045	High	5	4	428.14	

In [5]:
```python
data.tail()
```

Out[5]:

	Shipment_ID	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freig
9995	SHIP09996	284.0	633.0	Air	Bangalore	Los Angeles	0.52	0.977983	Medium	2	4	309.59	
9996	SHIP09997	NaN	2859.0	Rail	Kolkata	Los Angeles	1.39	1.091401	Medium	8	1	192.02	
9997	SHIP09998	1958.0	705.0	Air	Chennai	New York	NaN	0.971334	Medium	8	1	125.19	
9998	SHIP09999	1467.0	4931.0	Rail	Kolkata	Singapore	1.56	0.973844	High	1	1	107.21	
9999	SHIP10000	4969.0	NaN	Road	Chennai	Dubai	1.20	1.058222	High	7	1	209.60	

In [6]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Shipment_ID                10000 non-null  object 
 1   Distance_km                9500 non-null   float64
 2   Weight_kg                  9500 non-null   float64
 3   Mode_of_Transport          10000 non-null  object 
 4   Origin                     10000 non-null  object 
 5   Destination                10000 non-null  object 
 6   Fuel_Price_per_Liter       9500 non-null   float64
 7   Demand_Fluctuation         10000 non-null  float64
 8   Traffic_Conditions         10000 non-null  object 
 9   Warehouse_Processing_Time  10000 non-null  int64  
 10  Customs_Clearance_Delay    10000 non-null  int64  
 11  Packaging_Cost             10000 non-null  float64
 12  Freight_Insurance_Cost     10000 non-null  float64
 13  Freight_Cost_USD           9500 non-null   float64
dtypes: float64(7), int64(2), object(5)
memory usage: 1.1+ MB
```

In [7]:
```python
data.isna().sum()
```

Out[7]:
```
Shipment_ID                    0
Distance_km                  500
Weight_kg                    500
Mode_of_Transport              0
Origin                         0
Destination                    0
Fuel_Price_per_Liter         500
Demand_Fluctuation             0
Traffic_Conditions             0
Warehouse_Processing_Time      0
Customs_Clearance_Delay        0
Packaging_Cost                 0
Freight_Insurance_Cost         0
Freight_Cost_USD             500
dtype: int64
```

In [8]:
```python
data.columns
```

Out[8]:
```
Index(['Shipment_ID', 'Distance_km', 'Weight_kg', 'Mode_of_Transport',
       'Origin', 'Destination', 'Fuel_Price_per_Liter', 'Demand_Fluctuation',
       'Traffic_Conditions', 'Warehouse_Processing_Time',
       'Customs_Clearance_Delay', 'Packaging_Cost', 'Freight_Insurance_Cost',
       'Freight_Cost_USD'],
      dtype='object')
```

In [9]:
```python
data.describe()
```

Out[9]:

	Distance_km	Weight_kg	Fuel_Price_per_Liter	Demand_Fluctuation	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freight_Insurance_Cost	Freight_Cost_USD
count	9500.000000	9500.000000	9500.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	9500.000000
mean	2555.968211	2495.039895	1.256268	1.001966	4.978900	1.986600	255.671338	524.575038	818.740237
std	1428.546573	1426.040820	0.433669	0.115134	2.594056	1.421978	141.012448	273.959922	707.782778
min	51.000000	50.000000	0.500000	0.800052	1.000000	0.000000	10.160000	50.220000	9.720000
25%	1307.750000	1267.000000	0.880000	0.902636	3.000000	1.000000	132.777500	287.195000	367.452500
50%	2588.500000	2470.000000	1.260000	1.002316	5.000000	2.000000	257.350000	525.070000	633.185000
75%	3792.000000	3706.250000	1.640000	1.102649	7.000000	3.000000	375.425000	759.562500	1039.437500
max	4998.000000	4999.000000	2.000000	1.199945	9.000000	4.000000	499.960000	999.920000	16798.727885

Data Cleaning

Checking for Duplicates in a dataset

In [10]:
```python
data.duplicated().sum()
```

Out[10]:
```
0
```

In [11]:
```python
data['Mode_of_Transport'].value_counts()
```

Out[11]:
```
Road    2560
Rail    2503
Air     2481
Sea     2456
Name: Mode_of_Transport, dtype: int64
```

In [12]:
```python
median_counts = data['Freight_Cost_USD'].median()
median_counts
```

Out[12]:
```
633.185
```

In [13]:
```python
data['Freight_Cost_USD'].fillna(median_counts, inplace=True)
```

In [14]:
```python
median_counts_km = data['Distance_km'].median()
median_counts_km
```

Out[14]:
```
2588.5
```

In [15]:
```python
data['Distance_km'].fillna(median_counts_km, inplace=True)
```

In [16]:
```python
data['Distance_km'].value_counts()
```

Out[16]:
```
2588.5    500
2758.0      8
4830.0      8
1148.0      8
68.0        8
         ... 
4991.0      1
2985.0      1
3518.0      1
3942.0      1
4969.0      1
Name: Distance_km, Length: 4182, dtype: int64
```

In [17]:
```python
median_counts_fuel = data['Fuel_Price_per_Liter'].median()
median_counts_fuel
```

Out[17]:
```
1.26
```

In [18]:
```python
data['Fuel_Price_per_Liter'].fillna(median_counts_fuel,inplace=True)
```

In [19]:
```python
data['Fuel_Price_per_Liter'].value_counts()
```

Out[19]:
```
1.26    561
1.48     83
1.27     82
1.78     82
1.34     82
       ... 
1.82     48
1.50     46
1.49     44
2.00     31
0.50     21
Name: Fuel_Price_per_Liter, Length: 151, dtype: int64
```

In [20]:
```python
data["Weight_kg"].fillna(data["Weight_kg"].mean(), inplace=True)
```

In [21]:
```python
data.isna().sum()
```

Out[21]:
```
Shipment_ID                  0
Distance_km                  0
Weight_kg                    0
Mode_of_Transport            0
Origin                       0
Destination                  0
Fuel_Price_per_Liter         0
Demand_Fluctuation           0
Traffic_Conditions           0
Warehouse_Processing_Time    0
Customs_Clearance_Delay      0
Packaging_Cost               0
Freight_Insurance_Cost       0
Freight_Cost_USD             0
dtype: int64
```

In [22]:
```python
data.isna().sum()
```

Out[22]:
```
Shipment_ID                  0
Distance_km                  0
Weight_kg                    0
Mode_of_Transport            0
Origin                       0
Destination                  0
Fuel_Price_per_Liter         0
Demand_Fluctuation           0
Traffic_Conditions           0
Warehouse_Processing_Time    0
Customs_Clearance_Delay      0
Packaging_Cost               0
Freight_Insurance_Cost       0
Freight_Cost_USD             0
dtype: int64
```

In [23]:
```python
df=data
```

In [24]:
```python
df=data.copy()
```

Categorizing Numerical and categorical columns

In [25]:
```python
qual=[]
quan=[]
for columnname in df.columns:
    #print(columnname)
    if (df[columnname].dtype=='O'):
        #print("Qual")
        qual.append(columnname)
    else:
        #print("Quant")
        quan.append(columnname)
```

In [26]:
```python
qual
```

Out[26]:
```
['Shipment_ID',
 'Mode_of_Transport',
 'Origin',
 'Destination',
 'Traffic_Conditions']
```

In [27]:
```python
quan
```

Out[27]:
```
['Distance_km',
 'Weight_kg',
 'Fuel_Price_per_Liter',
 'Demand_Fluctuation',
 'Warehouse_Processing_Time',
 'Customs_Clearance_Delay',
 'Packaging_Cost',
 'Freight_Insurance_Cost',
 'Freight_Cost_USD']
```

Descriptive Statistics

In [28]:
```python
descriptive=pd.DataFrame(index=["Mean","Median","Mode","Q1:25%","Q2:50%","Q3:75%","99%","Q4:100%","IQR","1.5Rule",
                                "Lesser","Greater","Min","Max","kurtosis","skewness","Variance","Standard deviation"],columns=quan)
for columnname in quan:
    descriptive[columnname]["Mean"]=df[columnname].mean()
    descriptive[columnname]["Median"]=df[columnname].median()
    descriptive[columnname]["Mode"]=df[columnname].mode()[0]
    descriptive[columnname]["Q1:25%"]=df.describe()[columnname]["25%"]
    descriptive[columnname]["Q2:50%"]=df.describe()[columnname]["50%"]
    descriptive[columnname]["Q3:75%"]=df.describe()[columnname]["75%"]
    descriptive[columnname]["99%"]=np.percentile(df[columnname],99)
    descriptive[columnname]["Q4:100%"]=df.describe()[columnname]["max"]
    descriptive[columnname]["IQR"]=descriptive[columnname]["Q3:75%"]-descriptive[columnname]["Q1:25%"]
    descriptive[columnname]["1.5Rule"]=1.5*descriptive[columnname]["IQR"]
    descriptive[columnname]["Lesser"]=descriptive[columnname]["Q1:25%"]-descriptive[columnname]["1.5Rule"]
    descriptive[columnname]["Greater"]=descriptive[columnname]["Q3:75%"]+descriptive[columnname]["1.5Rule"]
    descriptive[columnname]["Min"]=df[columnname].min()
    descriptive[columnname]["Max"]=df[columnname].max()
    descriptive[columnname]["kurtosis"]=df[columnname].kurtosis()
    descriptive[columnname]["skewness"]=df[columnname].skew()
    descriptive[columnname]["Variance"]=df[columnname].var()
    descriptive[columnname]["Standard deviation"]=df[columnname].std()
```

In [29]:
```python
descriptive
```

Out[29]:

	Distance_km	Weight_kg	Fuel_Price_per_Liter	Demand_Fluctuation	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freight_Insurance_Cost	Freight_Cost_USD
Mean	2557.5948	2495.039895	1.256455	1.001966	4.9789	1.9866	255.671338	524.575038	809.462475
Median	2588.5	2495.039895	1.26	1.002316	5.0	2.0	257.35	525.07	633.185
Mode	2588.5	2495.039895	1.26	0.800052	1	0	73.53	85.05	633.185
Q1:25%	1366.0	1324.0	0.9	0.902636	3.0	1.0	132.7775	287.195	380.93
Q2:50%	2588.5	2495.039895	1.26	1.002316	5.0	2.0	257.35	525.07	633.185
Q3:75%	3727.5	3644.25	1.61	1.102649	7.0	3.0	375.425	759.5625	1009.0675
99%	4961.0	4955.01	1.98	1.195042	9.0	4.0	495.29	990.5142	3313.5317
Q4:100%	4998.0	4999.0	2.0	1.199945	9.0	4.0	499.96	999.92	16798.727885
IQR	2361.5	2320.25	0.71	0.200013	4.0	2.0	242.6475	472.3675	628.1375
1.5Rule	3542.25	3480.375	1.065	0.300019	6.0	3.0	363.97125	708.55125	942.20625
Lesser	-2176.25	-2156.375	-0.165	0.602617	-3.0	-2.0	-231.19375	-421.35625	-561.27625
Greater	7269.75	7124.625	2.675	1.402668	13.0	6.0	739.39625	1468.11375	1951.27375
Min	51.0	50.0	0.5	0.800052	1	0	10.16	50.22	9.72
Max	4998.0	4999.0	2.0	1.199945	9	4	499.96	999.92	16798.727885
kurtosis	-1.110873	-1.087657	-1.103063	-1.197287	-1.239082	-1.313119	-1.186346	-1.188018	52.770655
skewness	-0.026011	0.041274	-0.007337	-0.02783	-0.002417	0.017979	-0.011123	-0.002191	4.093521
Variance	1938748.115925	1931902.630651	0.178665	0.013256	6.729128	2.022023	19884.510531	75054.038921	477541.756588
Standard deviation	1392.389355	1389.929002	0.422688	0.115134	2.594056	1.421978	141.012448	273.959922	691.043961

Handling Outliers

In [30]:
```python
lesser=[]
greater=[]
for colname in quan:
    if(descriptive[colname]["Min"]<descriptive[colname]["Lesser"]):
        lesser.append(colname)
    if(descriptive[colname]["Max"]>descriptive[colname]["Greater"]):
        greater.append(colname)
```

In [31]:
```python
greater
```

Out[31]:
```
['Freight_Cost_USD']
```

In [32]:
```python
lesser
```

Out[32]:
```
[]
```

In [33]:
```python
def insertoutlier(data,greater):
    for columnName in greater:
        data[columnName][data[columnName]>descriptive[columnName]["Greater"]]=descriptive[columnName]["Greater"]
    return data
```

In [34]:
```python
insertoutlier(df,greater)
```

```
C:\Users\MOHANLAL\anaconda3\envs\aiml\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[34]:

	Shipment_ID	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Fre
0	SHIP00001	2588.5	1341.000000	Rail	Kolkata	London	1.89	1.042210	Medium	8	2	326.35	
1	SHIP00002	3822.0	3538.000000	Sea	Kolkata	Los Angeles	0.75	1.075883	High	8	3	293.82	
2	SHIP00003	3142.0	4777.000000	Road	Chennai	London	1.99	1.170771	Low	3	2	52.03	
3	SHIP00004	516.0	3704.000000	Air	Chennai	Dubai	1.58	1.129244	Low	8	3	421.87	
4	SHIP00005	4476.0	578.000000	Sea	Mumbai	Dubai	1.26	0.998045	High	5	4	428.14	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	SHIP09996	284.0	633.000000	Air	Bangalore	Los Angeles	0.52	0.977983	Medium	2	4	309.59	
9996	SHIP09997	2588.5	2859.000000	Rail	Kolkata	Los Angeles	1.39	1.091401	Medium	8	1	192.02	
9997	SHIP09998	1958.0	705.000000	Air	Chennai	New York	1.26	0.971334	Medium	8	1	125.19	
9998	SHIP09999	1467.0	4931.000000	Rail	Kolkata	Singapore	1.56	0.973844	High	1	1	107.21	
9999	SHIP10000	4969.0	2495.039895	Road	Chennai	Dubai	1.20	1.058222	High	7	1	209.60	

10000 rows × 14 columns

In [35]:
```python
# Assuming 'quan' is defined somewhere in your code
descriptive = pd.DataFrame(index=['mean','median','mode','Q1:25%','Q2:50%','Q3:75%','Q4:100%','IQR','1.5rule','lesser','greater','min','max'], columns=quan)

# Assuming 'dataset' is your DataFrame
for ColumnName in quan:
    descriptive.loc['mean', ColumnName] = df[ColumnName].mean()
    descriptive.loc['median', ColumnName] = df[ColumnName].median()
    descriptive.loc['mode', ColumnName] = df[ColumnName].mode()[0]
    descriptive.loc['Q1:25%', ColumnName] = df[ColumnName].quantile(0.25)
    descriptive.loc['Q2:50%', ColumnName] = df[ColumnName].quantile(0.50)
    descriptive.loc['Q3:75%', ColumnName] = df[ColumnName].quantile(0.75)
    descriptive.loc['99%', ColumnName] = np.percentile(df[ColumnName], 99)
    descriptive.loc['Q4:100%', ColumnName] = df[ColumnName].max()
    descriptive.loc['IQR', ColumnName] = descriptive.loc['Q3:75%', ColumnName] - descriptive.loc['Q1:25%', ColumnName]
    descriptive.loc['1.5rule', ColumnName] = 1.5 * descriptive.loc['IQR', ColumnName]
    descriptive.loc['lesser', ColumnName] = descriptive.loc['Q1:25%', ColumnName] - descriptive.loc['1.5rule', ColumnName]
    descriptive.loc['greater', ColumnName] = descriptive.loc['Q3:75%', ColumnName] + descriptive.loc['1.5rule', ColumnName]
    descriptive.loc['min', ColumnName] = df[ColumnName].min()
    descriptive.loc['max', ColumnName] = df[ColumnName].max()
```

In [36]:
```python
descriptive
```

Out[36]:

	Distance_km	Weight_kg	Fuel_Price_per_Liter	Demand_Fluctuation	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freight_Insurance_Cost	Freight_Cost_USD
mean	2557.5948	2495.039895	1.256455	1.001966	4.9789	1.9866	255.671338	524.575038	759.883205
median	2588.5	2495.039895	1.26	1.002316	5.0	2.0	257.35	525.07	633.185
mode	2588.5	2495.039895	1.26	0.800052	1	0	73.53	85.05	1951.27375
Q1:25%	1366.0	1324.0	0.9	0.902636	3.0	1.0	132.7775	287.195	380.93
Q2:50%	2588.5	2495.039895	1.26	1.002316	5.0	2.0	257.35	525.07	633.185
Q3:75%	3727.5	3644.25	1.61	1.102649	7.0	3.0	375.425	759.5625	1009.0675
Q4:100%	4998.0	4999.0	2.0	1.199945	9	4	499.96	999.92	1951.27375
IQR	2361.5	2320.25	0.71	0.200013	4.0	2.0	242.6475	472.3675	628.1375
1.5rule	3542.25	3480.375	1.065	0.300019	6.0	3.0	363.97125	708.55125	942.20625
lesser	-2176.25	-2156.375	-0.165	0.602617	-3.0	-2.0	-231.19375	-421.35625	-561.27625
greater	7269.75	7124.625	2.675	1.402668	13.0	6.0	739.39625	1468.11375	1951.27375
min	51.0	50.0	0.5	0.800052	1	0	10.16	50.22	9.72
max	4998.0	4999.0	2.0	1.199945	9	4	499.96	999.92	1951.27375
99%	4961.0	4955.01	1.98	1.195042	9.0	4.0	495.29	990.5142	1951.27375

In [37]:
```python
def find_outliers(quan):
    lesser = []
    greater = []

    for columnName in quan:
        if descriptive[columnName]["min"] < descriptive[columnName]["lesser"]:
            lesser.append(columnName)
        if descriptive[columnName]["max"] > descriptive[columnName]["greater"]:
            greater.append(columnName)

    return lesser, greater
```

In [38]:
```python
find_outliers(quan)
```

Out[38]:
```
([], [])
```

In [39]:
```python
df.to_csv('clean_data.csv',index=False)
```

In [40]:
```python
lesser, greater=find_outliers(quan)
```

In [41]:
```python
lesser
```

Out[41]:
```
[]
```

In [42]:
```python
greater
```

Out[42]:
```
[]
```

There are no outliers present in the dataset

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import RFE
import pickle
import warnings
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings("ignore")
```

Feature Selection

In [2]:
```
def selectkbest(indep_X,dep_Y,n):
        test = SelectKBest(score_func=f_regression, k=n)
        # We need to use the function f_regression for regression model.As this Data contains only classification data
        ##we are proceeding with chisquare
        fit1= test.fit(indep_X,dep_Y)
        selectk_features = fit1.transform(indep_X)
        selectk_features = fit1.transform(indep_X)
        selected_indices = fit1.get_support(indices=True)
        selected_features_names = indep_X.columns[selected_indices]
        return selectk_features,selected_features_names.to_list()
```

In [3]:
```
def split_scalar(indep_X,dep_Y):
        X_train, X_test, y_train, y_test = train_test_split(indep_X, dep_Y, test_size = 0.25, random_state = 0)
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
        return X_train, X_test, y_train, y_test
```

In [4]:
```
#Evaluation Metrics for Regression Models
def R2_prediction(regressor,x_test):
    y_pred = regressor.predict(x_test)

    from sklearn.metrics import r2_score
    R2_score = r2_score(y_test,y_pred)
    return regressor,R2_score,x_test,y_test
```

In [5]:
```
data=pd.read_csv("clean_data.csv")
data
```

Out[5]:

	Shipment_ID	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Fre
0	SHIP00001	2588.5	1341.000000	Rail	Kolkata	London	1.89	1.042210	Medium	8	2	326.35	
1	SHIP00002	3822.0	3538.000000	Sea	Kolkata	Los Angeles	0.75	1.075883	High	8	3	293.82	
2	SHIP00003	3142.0	4777.000000	Road	Chennai	London	1.99	1.170771	Low	3	2	52.03	
3	SHIP00004	516.0	3704.000000	Air	Chennai	Dubai	1.58	1.129244	Low	8	3	421.87	
4	SHIP00005	4476.0	578.000000	Sea	Mumbai	Dubai	1.26	0.998045	High	5	4	428.14	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	SHIP09996	284.0	633.000000	Air	Bangalore	Los Angeles	0.52	0.977983	Medium	2	4	309.59	
9996	SHIP09997	2588.5	2859.000000	Rail	Kolkata	Los Angeles	1.39	1.091401	Medium	8	1	192.02	
9997	SHIP09998	1958.0	705.000000	Air	Chennai	New York	1.26	0.971334	Medium	8	1	125.19	
9998	SHIP09999	1467.0	4931.000000	Rail	Kolkata	Singapore	1.56	0.973844	High	1	1	107.21	
9999	SHIP10000	4969.0	2495.039895	Road	Chennai	Dubai	1.20	1.058222	High	7	1	209.60	

10000 rows × 14 columns

In [6]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data["Origin"] = encoder.fit_transform(data["Origin"])
data["Destination"] = encoder.fit_transform(data["Destination"])
data["Mode_of_Transport"] = encoder.fit_transform(data["Mode_of_Transport"])
data["Traffic_Conditions"] = encoder.fit_transform(data["Traffic_Conditions"])
```

In [7]:
```
data
```

Out[7]:

	Shipment_ID	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freigh
0	SHIP00001	2588.5	1341.000000	1	3	1	1.89	1.042210	2	8	2	326.35	
1	SHIP00002	3822.0	3538.000000	3	3	2	0.75	1.075883	0	8	3	293.82	
2	SHIP00003	3142.0	4777.000000	2	1	1	1.99	1.170771	1	3	2	52.03	
3	SHIP00004	516.0	3704.000000	0	1	0	1.58	1.129244	1	8	3	421.87	
4	SHIP00005	4476.0	578.000000	3	4	0	1.26	0.998045	0	5	4	428.14	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	SHIP09996	284.0	633.000000	0	0	2	0.52	0.977983	2	2	4	309.59	
9996	SHIP09997	2588.5	2859.000000	1	3	2	1.39	1.091401	2	8	1	192.02	
9997	SHIP09998	1958.0	705.000000	0	1	3	1.26	0.971334	2	8	1	125.19	
9998	SHIP09999	1467.0	4931.000000	1	3	4	1.56	0.973844	0	1	1	107.21	
9999	SHIP10000	4969.0	2495.039895	2	1	0	1.20	1.058222	0	7	1	209.60	

10000 rows × 14 columns

In [8]:
```
data=data.drop('Shipment_ID',axis=1)
```

In [9]:
```
data=pd.get_dummies(data,drop_first=True)
data
```

Out[9]:

	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freight_Insurance_C
0	2588.5	1341.000000	1	3	1	1.89	1.042210	2	8	2	326.35	969
1	3822.0	3538.000000	3	3	2	0.75	1.075883	0	8	3	293.82	649
2	3142.0	4777.000000	2	1	1	1.99	1.170771	1	3	2	52.03	150
3	516.0	3704.000000	0	1	0	1.58	1.129244	1	8	3	421.87	181
4	4476.0	578.000000	3	4	0	1.26	0.998045	0	5	4	428.14	615
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	284.0	633.000000	0	0	2	0.52	0.977983	2	2	4	309.59	739
9996	2588.5	2859.000000	1	3	2	1.39	1.091401	2	8	1	192.02	386
9997	1958.0	705.000000	0	1	3	1.26	0.971334	2	8	1	125.19	323
9998	1467.0	4931.000000	1	3	4	1.56	0.973844	0	1	1	107.21	489
9999	4969.0	2495.039895	2	1	0	1.20	1.058222	0	7	1	209.60	854

10000 rows × 13 columns

In [10]:
```
indep_X=data.drop('Freight_Cost_USD',axis=1)
dep_Y=data['Freight_Cost_USD']
```

In [11]:
```
indep_X
```

Out[11]:

	Distance_km	Weight_kg	Mode_of_Transport	Origin	Destination	Fuel_Price_per_Liter	Demand_Fluctuation	Traffic_Conditions	Warehouse_Processing_Time	Customs_Clearance_Delay	Packaging_Cost	Freight_Insurance_C
0	2588.5	1341.000000	1	3	1	1.89	1.042210	2	8	2	326.35	969
1	3822.0	3538.000000	3	3	2	0.75	1.075883	0	8	3	293.82	649
2	3142.0	4777.000000	2	1	1	1.99	1.170771	1	3	2	52.03	150
3	516.0	3704.000000	0	1	0	1.58	1.129244	1	8	3	421.87	181
4	4476.0	578.000000	3	4	0	1.26	0.998045	0	5	4	428.14	615
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	284.0	633.000000	0	0	2	0.52	0.977983	2	2	4	309.59	739
9996	2588.5	2859.000000	1	3	2	1.39	1.091401	2	8	1	192.02	386
9997	1958.0	705.000000	0	1	3	1.26	0.971334	2	8	1	125.19	323
9998	1467.0	4931.000000	1	3	4	1.56	0.973844	0	1	1	107.21	489
9999	4969.0	2495.039895	2	1	0	1.20	1.058222	0	7	1	209.60	854

10000 rows × 12 columns

In [12]:
```
dep_Y
```

Out[12]:
```
0        818.740237
1        676.720000
2       1288.520000
3       1267.500000
4        463.680000
           ...
9995      91.620000
9996     472.500000
9997     599.220000
9998     716.910000
9999     648.270000
Name: Freight_Cost_USD, Length: 10000, dtype: float64
```

In [13]:
```
kbest,selected_features=selectkbest(indep_X,dep_Y,5)
```

In [14]:
```
selected_features
```

Out[14]:
```
['Distance_km',
 'Weight_kg',
 'Mode_of_Transport',
 'Fuel_Price_per_Liter',
 'Demand_Fluctuation']
```

In [15]:
```
kbest
```

Out[15]:
```
array([[2.58850000e+03, 1.34100000e+03, 1.00000000e+00, 1.89000000e+00,
        1.04220962e+00],
       [3.82200000e+03, 3.53800000e+03, 3.00000000e+00, 7.50000000e-01,
        1.07588253e+00],
       [3.14200000e+03, 4.77700000e+03, 2.00000000e+00, 1.99000000e+00,
        1.17077117e+00],
       ...,
       [1.95800000e+03, 7.05000000e+02, 0.00000000e+00, 1.26000000e+00,
        9.71333983e-01],
       [1.46700000e+03, 4.93100000e+03, 1.00000000e+00, 1.56000000e+00,
        9.73844311e-01],
       [4.96900000e+03, 2.49503989e+03, 2.00000000e+00, 1.20000000e+00,
        1.05822176e+00]])
```

In [16]:
```
def linear(x_train,y_train,x_test):
    from sklearn.linear_model import LinearRegression
    param_grid={'fit_intercept':[True,False],'copy_X':[True,False]}
    grid=GridSearchCV(LinearRegression(), param_grid, refit=True, verbose=1,n_jobs=1)
    grid.fit(x_train,y_train)
    R2_score=R2_prediction(grid,x_test)
    print ("The Rsquare value for the the best paramater ():".format(grid.best_params_))
    return grid,R2_score,x_test,y_test

def svm_linear(x_train,y_train,x_test):
    from sklearn.svm import SVR
    param_grid={'kernel':['linear', 'poly', 'rbf', 'sigmoid'],'C':[10,100],'gamma':['auto','scale']}
    grid=GridSearchCV(SVR(), param_grid, refit=True, verbose=1,n_jobs=1)
    grid.fit(x_train,y_train)
    R2_score=R2_prediction(grid,x_test)
    print ("The Rsquare value for the the best paramater ():".format(grid.best_params_))
    return grid,R2_score,x_test,y_test


def decision(x_train,y_train,x_test):
    from sklearn.tree import DecisionTreeRegressor
    param_grid={'criterion':['squared_error', 'friedman_mse', 'absolute_error'],'splitter':['best', 'random'],'max_features':['sqrt', 'log2']}
    grid=GridSearchCV(DecisionTreeRegressor(), param_grid, refit=True, verbose=1,n_jobs=1)
    grid.fit(x_train,y_train)
    R2_score=R2_prediction(grid,x_test)
    print ("The Rsquare value for the the best paramater ():".format(grid.best_params_))
    return grid,R2_score,x_test,y_test

def random(x_train,y_train,x_test):
    from sklearn.ensemble import RandomForestRegressor
    param_grid={'n_estimators':[10,50,100],'criterion':['squared_error', 'friedman_mse', 'absolute_error'],'max_features':['sqrt', 'log2']}
    grid=GridSearchCV(RandomForestRegressor(), param_grid, refit=True, verbose=1,n_jobs=1)
    grid.fit(x_train,y_train)
    R2_score=R2_prediction(grid,x_test)
    print ("The Rsquare value for the the best paramater ():".format(grid.best_params_))
    return grid,R2_score,x_test,y_test

def boost(x_train,y_train,x_test):
    from sklearn.ensemble import GradientBoostingRegressor
    param_grid={'n_estimators':[10,50,100],'loss':['squared_error', 'huber', 'absolute_error'],'max_features':['sqrt', 'log2']}
    grid=GridSearchCV(GradientBoostingRegressor(), param_grid, refit=True, verbose=1,n_jobs=1)
    grid.fit(x_train,y_train)
    R2_score=R2_prediction(grid,x_test)
    print ("The Rsquare value for the the best paramater ():".format(grid.best_params_))
    return grid,R2_score,x_test,y_test
```

In [17]:
```
def selectk_regression(r2lin,r2svml,r2dc,r2rf,r2bg):
    dataframe = pd.DataFrame(index=['R2_Square'], columns = ['Linear','SVM_L','Decision','Random','boost'])
    for number,index in enumerate(dataframe.index):
        dataframe['Linear'][index] = r2lin[number]
        dataframe['SVM_L'][index] = r2svml[number]
        dataframe['Decision'][index] = r2dc[number]
        dataframe['Random'][index] = r2rf[number]
        dataframe['boost'][index] = r2bg[number]
        return dataframe
```

In [18]:
```
r2lin = []
r2svml = []
r2dc = []
r2rf = []
r2bg=[]
```

In [19]:
```
x_train,x_test,y_train,y_test = split_scalar(kbest,dep_Y)
```

In [20]:
```
regressor,R2_score,x_test,y_test = linear(x_train,y_train,x_test)
r2lin.append(R2_score[1])

regressor,R2_score,x_test,y_test = svm_linear(x_train,y_train,x_test)
r2svml.append(R2_score[1])

regressor,R2_score,x_test,y_test = decision(x_train,y_train,x_test)
r2dc.append(R2_score[1])

regressor,R2_score,x_test,y_test = random(x_train,y_train,x_test)
r2rf.append(R2_score[1])

regressor,R2_score,x_test,y_test = boost(x_train,y_train,x_test)
r2bg.append(R2_score[1])

result = selectk_regression(r2lin,r2svml,r2dc,r2rf,r2bg)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
The Rsquare value for the the best paramater {'copy_X': True, 'fit_intercept': True}:
Fitting 5 folds for each of 16 candidates, totalling 80 fits
The Rsquare value for the the best paramater {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}:
Fitting 5 folds for each of 12 candidates, totalling 60 fits
The Rsquare value for the the best paramater {'criterion': 'squared_error', 'max_features': 'log2', 'splitter': 'best'}:
Fitting 5 folds for each of 18 candidates, totalling 90 fits
The Rsquare value for the the best paramater {'criterion': 'friedman_mse', 'max_features': 'sqrt', 'n_estimators': 100}:
Fitting 5 folds for each of 18 candidates, totalling 90 fits
The Rsquare value for the the best paramater {'loss': 'huber', 'max_features': 'sqrt', 'n_estimators': 100}:
```

In [21]:
```
result
#5
```

Out[21]:

	Linear	SVM_L	Decision	Random	boost
R2_Square	0.600909	0.866211	0.675884	0.848741	0.846843

The above results shows that SVR algorithms works the best for this data set with the parameters kernel=rbf,C=100,gamma scale

In [ ]:
```
result
#7
```

In [ ]:
```
! pip install scikit-learn
```

In [ ]:
```
conda update scikit-learn
```

Freight Cost Prediction Application

Django framework has been used to built this application

Input

Output