All the dependencies are imported herewith

In [6]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
import math
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_squared_error, mean_absolute_error,

from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
import xgboost as xgb
```

MILESTONE - 1: DATA SANITY

1. Use the PRS dataset to create a dataframe

In [7]:

```
df = pd.DataFrame(pd.read_csv('Final.csv'))
```

2. Check the description of the dataframe

In [8]:
```
1 df.describe()
```

Out[8]:

	CUSTOMER_ORDER_ID	SALES_ORG	COMPANY_CODE	ORDER_CREATION_DATE	
count	1.101925e+06	1.101925e+06	1.101925e+06	1.101925e+06	
mean	8.763187e+08	3.304891e+03	3.431364e+03	2.022032e+07	
std	7.088594e+07	6.958920e+02	5.483805e+02	1.426756e+02	
min	7.534520e+08	2.100000e+03	5.900000e+01	2.022010e+07	
25%	8.149522e+08	2.702000e+03	3.260000e+03	2.022021e+07	
50%	8.763963e+08	3.305000e+03	3.660000e+03	2.022032e+07	
75%	9.376832e+08	3.908000e+03	3.670000e+03	2.022042e+07	
max	9.990063e+08	4.510000e+03	4.260000e+03	2.022060e+07	

In [9]:
```
1 df.head()          #IT WILL PRINT THE FIRST FEW ROWS OF DATASET
```

Out[9]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEASED_C
0	946851639	3537	United States of America	South-Region	
1	963432061	3449	Martinique	South-Region	
2	971991639	3238	Moldova	South-Region	
3	754349803	3911	United Arab Emirates	South-Region	
4	930253442	2381	Greece	South-Region	

In [10]:
```
1 df.tail()          #IT WILL PRINT THE LAST FEW ROWS OF DATASET
```

Out[10]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEA
1101920	853605710	2498	Germany	South-Region	
1101921	998890898	4509	Armenia	South-Region	
1101922	983330221	3951	Nepal	South-Region	
1101923	926668029	3235	Panama	South-Region	
1101924	921701000	2968	Nicaragua	South-Region	

3. Check the shape of the dataframe

In [11]:
```
1 df.shape
```

Out[11]: (1101925, 16)

4. Check the data frame informations

In [12]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1101925 entries, 0 to 1101924
Data columns (total 16 columns):
 #   Column                   Non-Null Count    Dtype  
---  ------                   --------------    -----  
 0   CUSTOMER_ORDER_ID        1101925 non-null  int64  
 1   SALES_ORG                1101925 non-null  int64  
 2   DISTRIBUTION_CHANNEL     1101925 non-null  object 
 3   DIVISION                 1101925 non-null  object 
 4   RELEASED_CREDIT_VALUE    1101925 non-null  object 
 5   PURCHASE_ORDER_TYPE      1083233 non-null  object 
 6   COMPANY_CODE             1101925 non-null  int64  
 7   ORDER_CREATION_DATE      1101925 non-null  int64  
 8   ORDER_CREATION_TIME      1101925 non-null  int64  
 9   CREDIT_CONTROL_AREA      1101925 non-null  object 
 10  SOLD_TO_PARTY            1101925 non-null  int64  
 11  ORDER_AMOUNT             1101925 non-null  object 
 12  REQUESTED_DELIVERY_DATE  1101925 non-null  int64  
 13  ORDER_CURRENCY           1101925 non-null  object 
 14  CREDIT_STATUS            219478 non-null   float64
 15  CUSTOMER_NUMBER          1101925 non-null  int64  
dtypes: float64(1), int64(8), object(7)
memory usage: 134.5+ MB
```

5. Check for the Null values in the dataframe

In [13]:
```
df.isnull()
# It will return boolean values for each data present in dataset
# If Null returns true ,and Not null return False
```

Out[13]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEA
0	False	False	False	False	
1	False	False	False	False	
2	False	False	False	False	
3	False	False	False	False	
4	False	False	False	False	
...	...	...	...	...	
1101920	False	False	False	False	
1101921	False	False	False	False	
1101922	False	False	False	False	
1101923	False	False	False	False	
1101924	False	False	False	False	

1101925 rows × 16 columns

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
CUSTOMER_ORDER_ID              0
SALES_ORG                      0
DISTRIBUTION_CHANNEL           0
DIVISION                       0
RELEASED_CREDIT_VALUE          0
PURCHASE_ORDER_TYPE        18692
COMPANY_CODE                   0
ORDER_CREATION_DATE            0
ORDER_CREATION_TIME            0
CREDIT_CONTROL_AREA            0
SOLD_TO_PARTY                  0
ORDER_AMOUNT                   0
REQUESTED_DELIVERY_DATE        0
ORDER_CURRENCY                 0
CREDIT_STATUS             882447
CUSTOMER_NUMBER                0
dtype: int64
```

6. Replace all the null values with "NaN"

In [15]:
```
# We marked the Null values as missing by using np.nan

import numpy as np
df.fillna(np.nan, inplace = True)
```

In [16]:
```
# As CREDIT_STATUS is a categorical column, and having Null values so
# and Null values will be filled using suitable category column 'Unkn
# So that in future we can perform Label Encoding without any hindran

# As we are using RandomForestRegressor to extract important features,

df['CREDIT_STATUS'].fillna('Unknown', inplace=True)

df['CREDIT_STATUS'] = df['CREDIT_STATUS'].astype(str)

df['PURCHASE_ORDER_TYPE'] = df['PURCHASE_ORDER_TYPE'].fillna('Unknown
```

In [17]:
```
df
```

Out[17]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEA
0	946851639	3537	United States of America	South-Region	
1	963432061	3449	Martinique	South-Region	
2	971991639	3238	Moldova	South-Region	
3	754349803	3911	United Arab Emirates	South-Region	
4	930253442	2381	Greece	South-Region	
...	...	...	...	...	
1101920	853605710	2498	Germany	South-Region	
1101921	998890898	4509	Armenia	South-Region	
1101922	983330221	3951	Nepal	South-Region	
1101923	926668029	3235	Panama	South-Region	
1101924	921701000	2968	Nicaragua	South-Region	

1101925 rows × 16 columns

7. Change the format of date columns - "ORDER_CREATION_DATE" to datetime[64] with the format as "%Y%m%d"

In [18]:
```
1 print(df['ORDER_CREATION_DATE'])
```

```
0          20220101
1          20220101
2          20220101
3          20220101
4          20220101
             ...
1101920    20220601
1101921    20220601
1101922    20220601
1101923    20220601
1101924    20220601
Name: ORDER_CREATION_DATE, Length: 1101925, dtype: int64
```

In [19]:
```
1 df['ORDER_CREATION_DATE'] = pd.to_datetime(df['ORDER_CREATION_DATE'],
```

In [20]:
```
1 print(df['ORDER_CREATION_DATE'])
```

```
0          2022-01-01
1          2022-01-01
2          2022-01-01
3          2022-01-01
4          2022-01-01
              ...
1101920    2022-06-01
1101921    2022-06-01
1101922    2022-06-01
1101923    2022-06-01
1101924    2022-06-01
Name: ORDER_CREATION_DATE, Length: 1101925, dtype: datetime64[ns]
```

8. Do the same activity for the other date field i.e. "REQUESTED_DELIVERY_DATE" to datetime[64] with the format as "%Y%m%d"

In [21]:
```
1 df['REQUESTED_DELIVERY_DATE'] = pd.to_datetime(df['REQUESTED_DELIVERY_
```

In [22]:
```
1 print(df['REQUESTED_DELIVERY_DATE'])
```

```
0          2022-01-13
1          2022-01-11
2          2022-01-12
3          2022-01-06
4          2022-01-06
              ...
1101920    2022-06-01
1101921    2022-06-01
1101922    2022-06-01
1101923    2022-06-01
1101924    2022-06-01
Name: REQUESTED_DELIVERY_DATE, Length: 1101925, dtype: datetime64[ns]
```

9. Sanity check - Check how many records are having order date greater than the delivery date

In [23]:
```
1 print(len(df[df['ORDER_CREATION_DATE'] > df['REQUESTED_DELIVERY_DATE'
```

```
27142
```

10. Remove those records where order date is greater than the delivery date

In [24]:
```
1 df = df.drop(df[df['ORDER_CREATION_DATE'] > df['REQUESTED_DELIVERY_DA
```

In [25]:
```
1 print(len(df[df['ORDER_CREATION_DATE'] > df['REQUESTED_DELIVERY_DATE'
```

```
0
```

11. Check the number of records where the “ORDER_AMOUNT” field is having “-” in it..

In [26]:
```
1 no_of_records = df['ORDER_AMOUNT'].str.contains('-').sum()
2 print(no_of_records)
```

```
32
```

12. Replace “-” with “” from the “ORDER_AMOUNT” field.

In [27]:
```
1 df['ORDER_AMOUNT'] = df['ORDER_AMOUNT'].str.replace('-', '')
```

In [28]:
```
1 print(df['ORDER_AMOUNT'].str.contains('-').sum())
```

```
0
```

13. Check the number of records where the “ORDER_AMOUNT” field is having “,” in it..

In [29]:
```
1 print(df['ORDER_AMOUNT'].str.contains(',').sum())
```

```
1073406
```

14. Replace “,” with “.” from the “ORDER_AMOUNT” field.

In [30]:
```
1 df['ORDER_AMOUNT'] = df['ORDER_AMOUNT'].str.replace(',', '.')
```

In [31]:
```
1 print(df['ORDER_AMOUNT'].str.contains(',').sum())
```

```
0
```

15. Count the number of records where the order date and the delivery date are same

In [32]:
```
1 print(len(df[df['ORDER_CREATION_DATE'] == df['REQUESTED_DELIVERY_DATE
```

```
100437
```

16. Count the number of records for each currency type by using the field “'ORDER_CURRENCY”

In [33]:
```
1 print(df['ORDER_CURRENCY'].value_counts())
```

```
USD    622835
EUR    253196
AUD     64200
CAD     55065
GBP     22028
MYR     13946
PLN     11861
AED      7852
HKD      6198
CHF      5259
RON      4731
SGD      3909
CZK      2178
HU1      1377
NZD        79
BHD        32
SAR        14
QAR        12
KWD         7
SEK         4
Name: ORDER_CURRENCY, dtype: int64
```

17. Create a new column in the existing dataframe as "'amount_in_usd'" and convert all the non-USD currencies in USD and store them in the same column.

In [34]:
```
exchange_rates = {
    'USD':1,
    'EUR': 1.08,
    'AUD': 0.65,
    'CAD': 0.74,
    'GBP': 1.24,
    'MYR': 0.22,
    'PLN': 0.24,
    'AED': 0.27,
    'HKD': 0.13,
    'CHF': 1.11,
    'RON': 0.22,
    'SGD': 0.74,
    'CZK': 0.045,
    'HU1': 0.0029,
    'NZD': 0.61,
    'BHD': 2.65,
    'SAR': 0.27,
    'QAR': 0.27,
    'KWD': 3.25,
    'SEK': 0.094
}
df['AMOUNT_IN_USD'] = df['ORDER_AMOUNT'].astype(float) * df['ORDER_CUI
```

In [35]:
```
df['AMOUNT_IN_USD']
```

Out[35]:
```
0           1030.9788
1            850.3488
2          72892.9368
3           1517.9832
4              0.0000
                ...
1101920     3267.2406
1101921        0.0000
1101922        0.0000
1101923        0.0000
1101924     3267.2406
Name: AMOUNT_IN_USD, Length: 1074783, dtype: float64
```

18. Check for values "0" in the "'amount_in_usd" column.

In [36]:
```
df[df['AMOUNT_IN_USD'] ==0.0].shape[0]
```

Out[36]:
```
237821
```

19. Create a new column in the existing dataframe “unique_cust_id” by adding 'CUSTOMER_NUMBER' and 'COMPANY_CODE'

In [37]:
```
df['UNIQUE_CUST_ID'] = df['CUSTOMER_NUMBER'].astype(str) + df['COMPAN
```

In [38]:
```
df['UNIQUE_CUST_ID']
```

Out[38]:
```
0            123118073220
1            123118073220
2            121187583260
3          12104997703290
4          12103514003290
                  ...
1101920    12103318044260
1101921    12103318044260
1101922    12103318114260
1101923    12103318114260
1101924    12103318044260
Name: UNIQUE_CUST_ID, Length: 1074783, dtype: object
```

In [ ]:
```

```

In [ ]:
```

```

MILESTONE -2: Exploratory Data Analysis

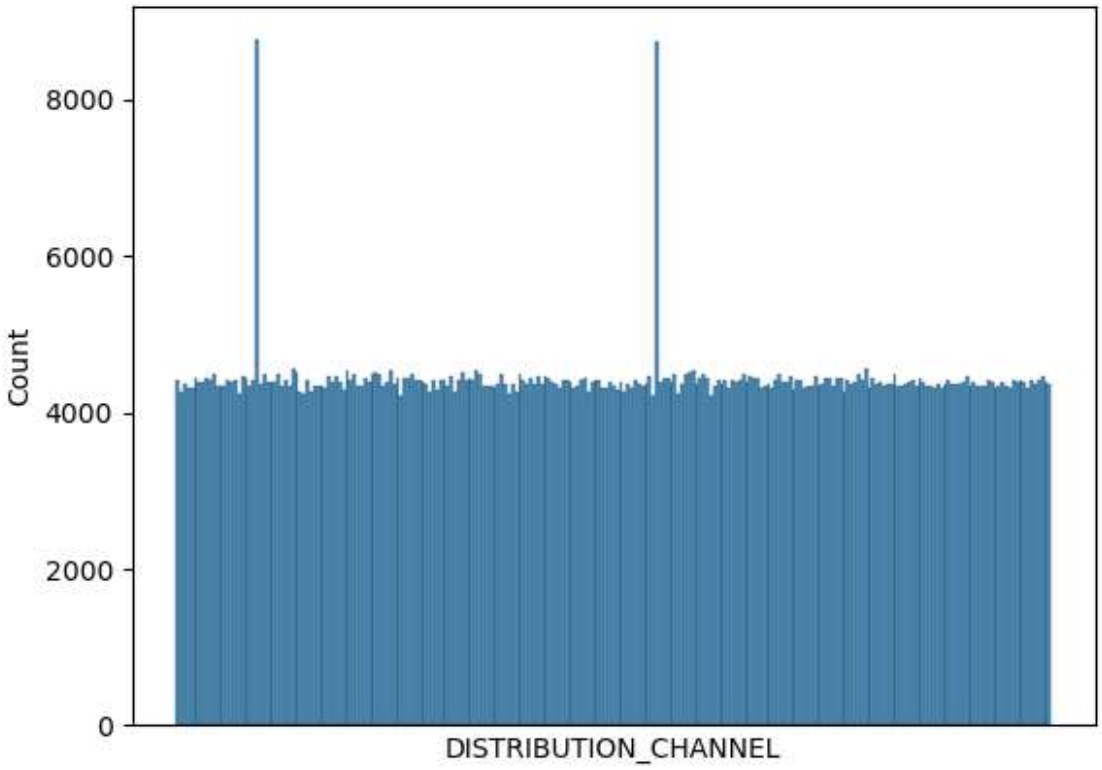
1. Create a Histogram on DISTRIBUTION_CHANNEL

In [39]:
```
df['DISTRIBUTION_CHANNEL']
```

Out[39]:
```
0          United States of America
1                         Martinique
2                            Moldova
3                United Arab Emirates
4                             Greece
                       ...
1101920                      Germany
1101921                      Armenia
1101922                        Nepal
1101923                       Panama
1101924                    Nicaragua
Name: DISTRIBUTION_CHANNEL, Length: 1074783, dtype: object
```

In [40]:

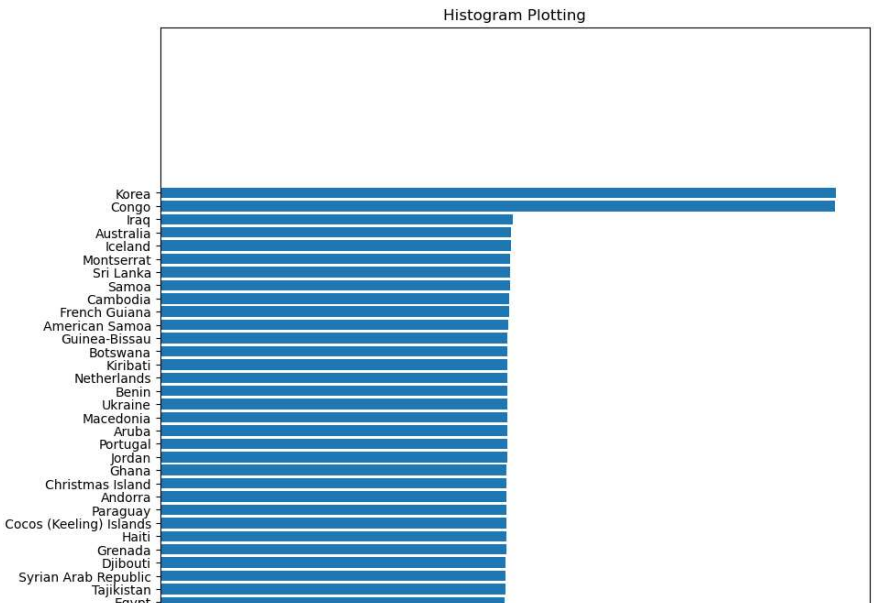
```
1 sns.histplot(df['DISTRIBUTION_CHANNEL'], bins = 400)
2 plt.xticks([])
3 plt.show()
```



In [41]:

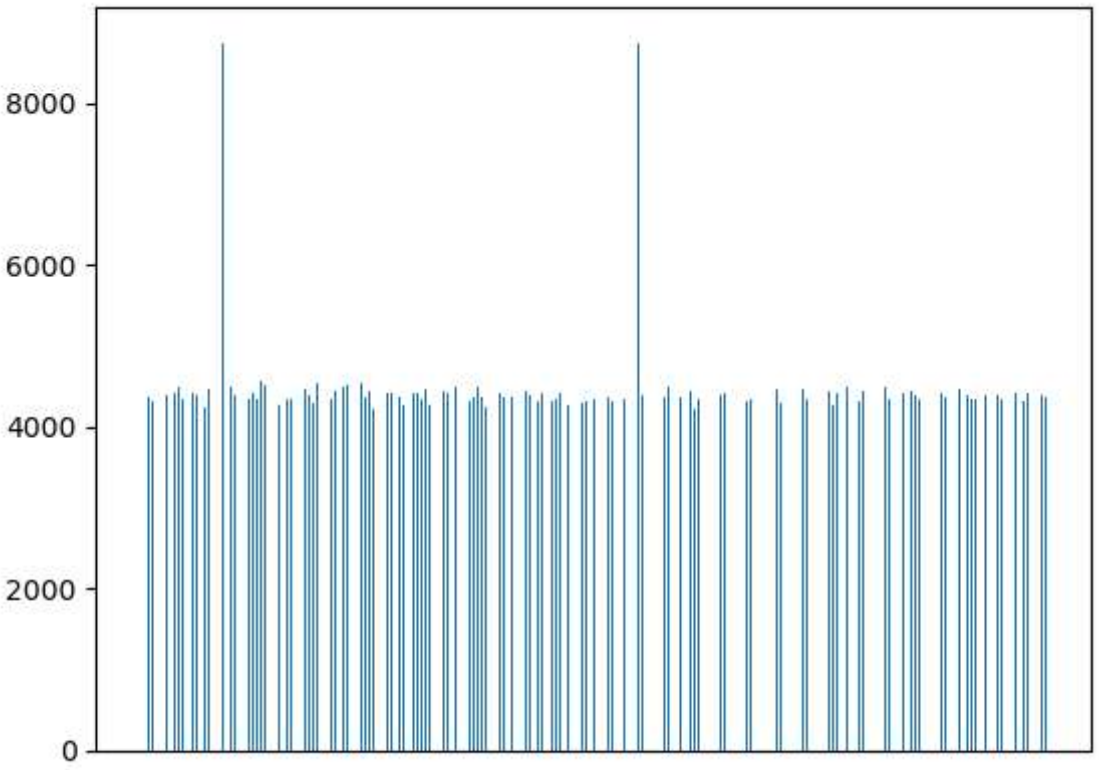
```
# Histogram is drawn for each Distribution Channel taking it according

x047 = df['DISTRIBUTION_CHANNEL'].value_counts()
y047 = x047.sort_values()
plt.figure(figsize=(10, 50))
plt.barh(y047.index,y047.values)
plt.xlabel('COUNT VALUE')
plt.ylabel('DISTRIBUTION CHANNEL')
plt.title('Histogram Plotting')
```

Out[41]: Text(0.5, 1.0, 'Histogram Plotting')



In [42]:

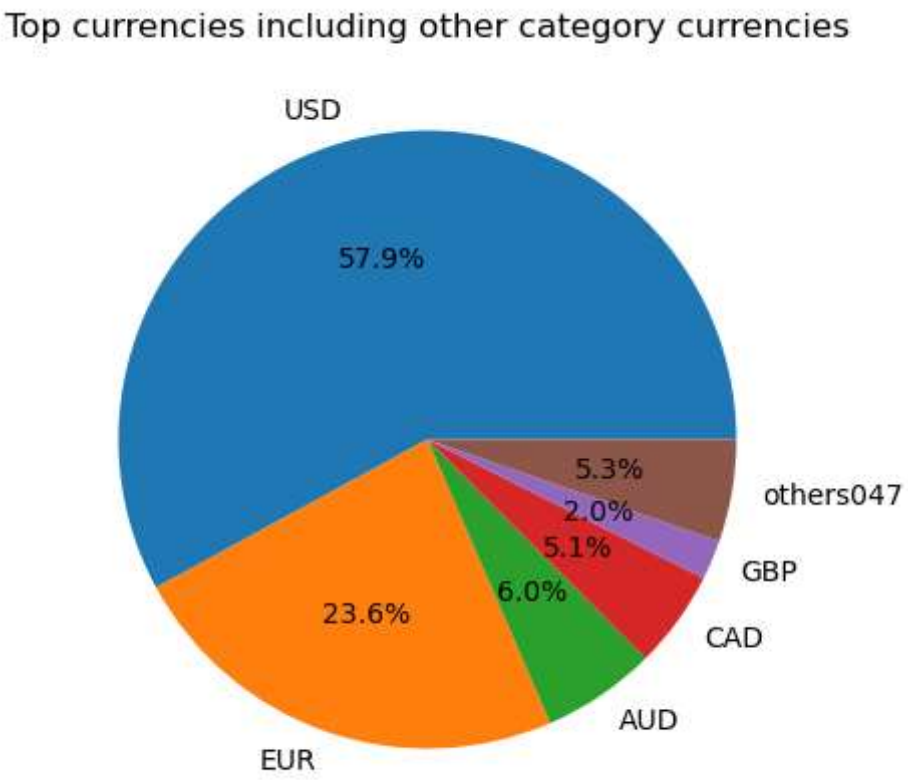
```
plt.hist(df['DISTRIBUTION_CHANNEL'], bins = 1000)
plt.xticks([])              # Removing the label marks along x- axis
plt.show()
```



2. Create a Pie Chart on ORDER_CURRENCY

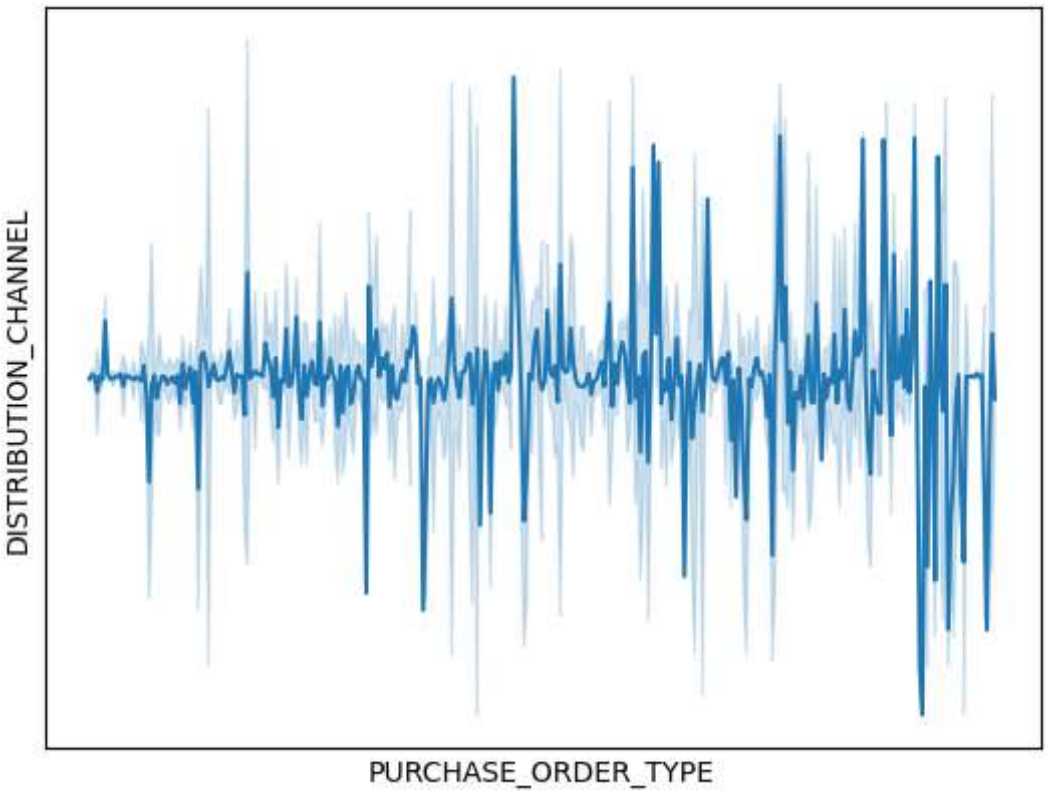
In [43]:

```
count047 = df['ORDER_CURRENCY'].value_counts()

# Here the top 5 category of currencies are taken in count047_top and

count047_top = count047[:5].index
others047 = count047[5:].sum()

threshold047 = 0.05              # As top 5 currencies are taken separ

# Create labels for the top currencies and 'Others'
labels = list(count047_top) + ['others047']

sizes = list(count047[:5])+[others047]

plt.figure(figsize=(10, 5))
plt.pie(sizes, labels=labels,autopct='%1.1f%%')
plt.title('Top currencies including other category currencies')
plt.show()
```



3. Create a line chart PURCHASE_ORDER_TYPE and DISTRIBUTION_CHANNEL

In [44]:
```
1 sns.lineplot(x = df['PURCHASE_ORDER_TYPE'], y = df['DISTRIBUTION_CHANN
2 plt.yticks([])
3 plt.xticks([])
4 plt.show()
```

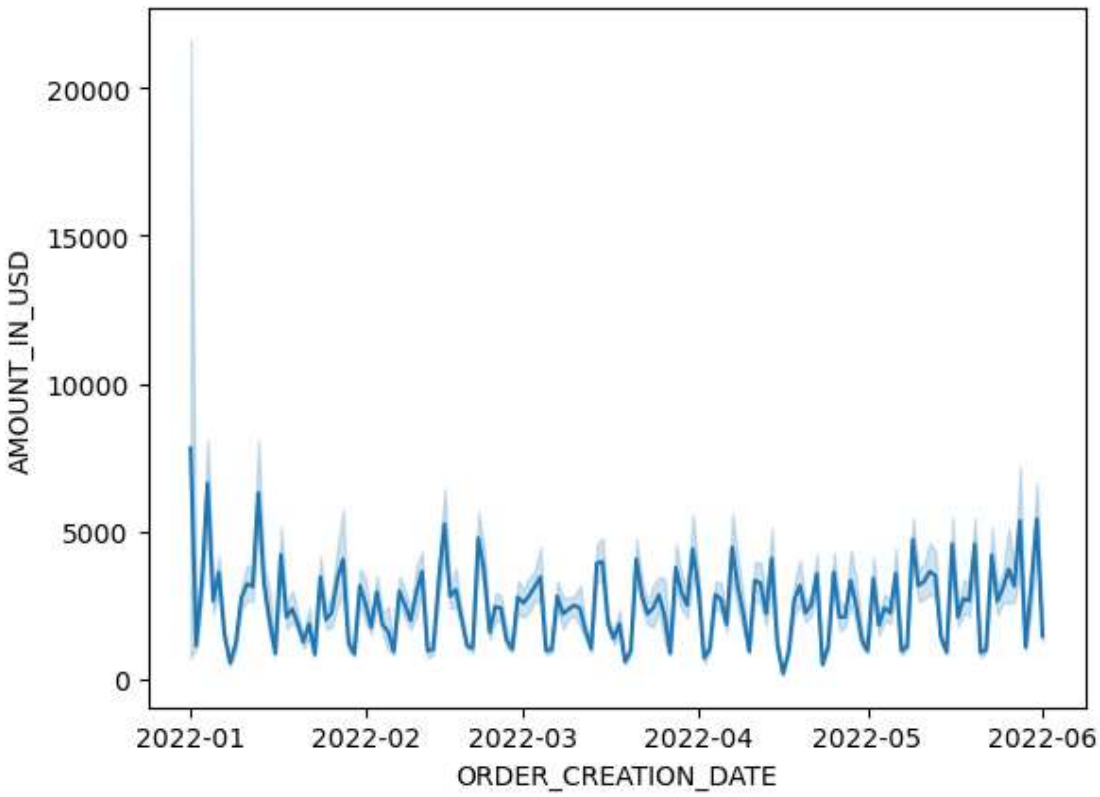


4. Create a line plot on ORDER_CREATION_DATE and amount_in_usd

In [45]:
```
1 df['AMOUNT_IN_USD']
```

Out[45]:
```
0           1030.9788
1            850.3488
2          72892.9368
3           1517.9832
4              0.0000
                ...
1101920     3267.2406
1101921        0.0000
1101922        0.0000
1101923        0.0000
1101924     3267.2406
Name: AMOUNT_IN_USD, Length: 1074783, dtype: float64
```

In [46]:
```
1 sns.lineplot(x=df['ORDER_CREATION_DATE'],y=df['AMOUNT_IN_USD'],data=d
2 plt.show()
```



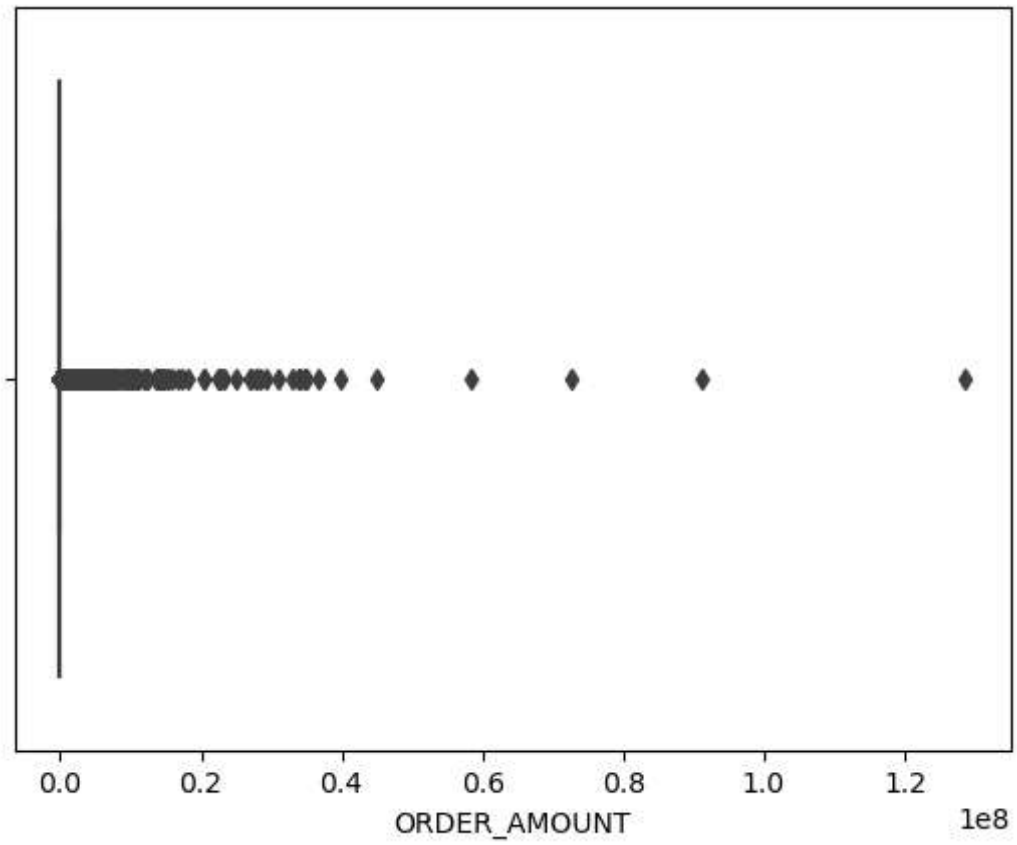
5. Create a boxplot on ORDER_AMOUNT to find out the outliers

In [47]:
```
1 df['ORDER_AMOUNT'].dtype
```

Out[47]: dtype('O')

In [48]:

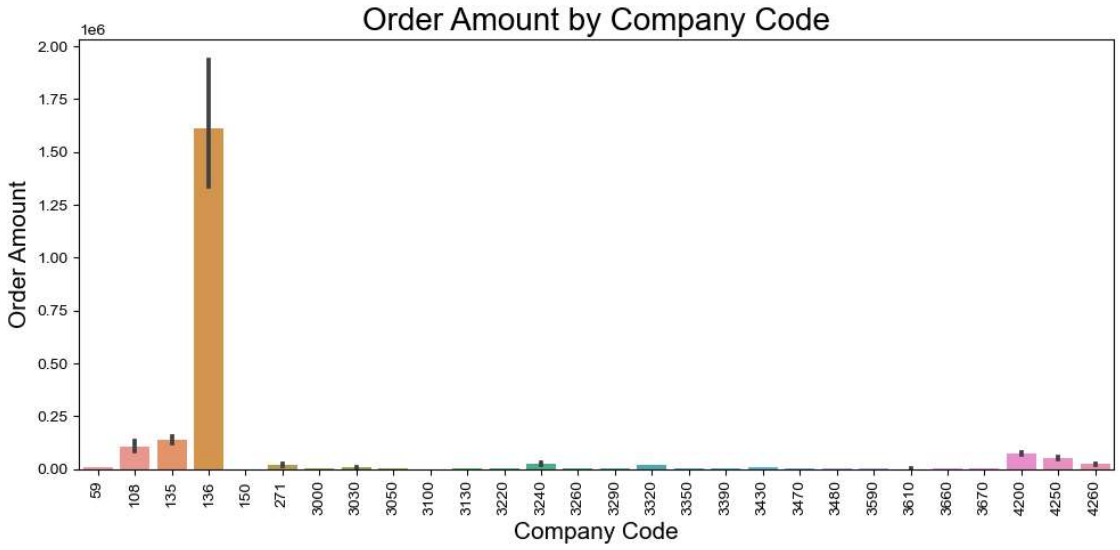
```
# As the datatype of ORDER_AMOUNT column is not numeric, so we need to

df['ORDER_AMOUNT'] = pd.to_numeric(df['ORDER_AMOUNT'])

sns.boxplot(x=df['ORDER_AMOUNT'],data=df)
plt.show()
```



6. Create a barchart on COMPANY_CODE and ORDER_AMOUNT

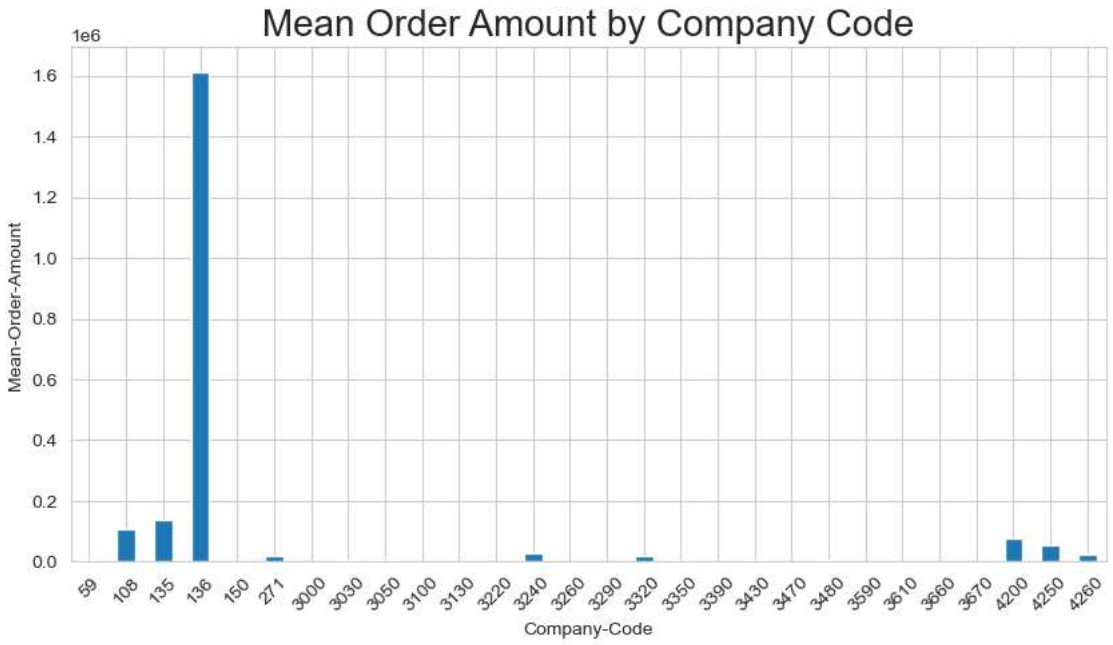
In [49]:

```
plt.figure(figsize=(10, 5))  # Set the figure size

sns.barplot(x='COMPANY_CODE', y='ORDER_AMOUNT', data=df)

plt.xlabel('Company Code', fontsize=15)
plt.ylabel('Order Amount', fontsize=15)
plt.title('Order Amount by Company Code', fontsize=20)

sns.set_style('whitegrid')
plt.xticks(fontsize=10, rotation=90)
plt.yticks(fontsize=10)

plt.tight_layout()  # Adjust spacing between elements

plt.show()
```



In [50]:

```
# Mean order amount for each company code

mean_order_amount047 = df.groupby('COMPANY_CODE')['ORDER_AMOUNT'].mean
plt.figure(figsize=(10, 5))
mean_order_amount047.plot(kind='bar')
plt.xlabel('Company-Code')
plt.ylabel('Mean-Order-Amount')
plt.title('Mean Order Amount by Company Code', fontsize = 20)
plt.xticks(rotation=45,fontsize = 10)
plt.grid(True)
plt.show()
```



In [ ]:

```
1
```

In [ ]:

```
1
```

MILESTONE -3 [FEATURE ENGINEERING & SELECTION]

1. Check for the outliers in the “amount_in_usd” column and replace the outliers with appropriate values, discussed in the sessions.

In [51]:

```
# So for detecting outliers from the specific column, we can use stat
# Here, Z-Score or IQR can be calculated for Outlier checking.
```

In [52]:
```
# IQR Calculation

q1 = df['AMOUNT_IN_USD'].quantile(0.25)
q3 = df['AMOUNT_IN_USD'].quantile(0.75)
IQR_047 = q3 - q1

lower_bound_047 = q1 - 1.5 * IQR_047
upper_bound_047 = q3 + 1.5 * IQR_047

outliers = df[(df['AMOUNT_IN_USD'] < lower_bound_047) | (df['AMOUNT_I
```

In [53]:
```
print(len(outliers))       #No of outliers present in dataset
```

```
101250
```

In [54]:
```
outliers.index              # Indices with correspondence to outliers
```

Out[54]:
```
Int64Index([      2,       8,      34,      59,      68,      70,      7
1,
                 72,      73,      85,
            ...
            1101897, 1101906, 1101910, 1101913, 1101915, 1101916, 110191
7,
            1101918, 1101920, 1101924],
           dtype='int64', length=101250)
```

In [55]:
```
# Outlier identification using Z-score

# Calculate the Z-score for each data point in the "amount_in_usd" col
mean_047 = np.mean(df['AMOUNT_IN_USD'])
std_047 = np.std(df['AMOUNT_IN_USD'])
z_scores_047 = (df['AMOUNT_IN_USD'] - mean_047) / std_047



# Find rows with Z-scores greater than a certain threshold (e.g., 3)
threshold_val = 3
outlier_count_047 = df[z_scores_047 > threshold_val]

print(len(outlier_count_047))
print()
print(outlier_count_047.index)
```

```
6086

Int64Index([     59,     133,     149,     349,     361,     439,     48
8,
                501,     520,     525,
            ...
            1099584, 1099585, 1101152, 1101161, 1101163, 1101182, 110127
8,
            1101473, 1101598, 1101628],
           dtype='int64', length=6086)
```

In [56]:
```
# Replacing Outliers of Z-score with median value of the column

median_047 = df['AMOUNT_IN_USD'].median()
df.loc[z_scores_047 > threshold_val, 'AMOUNT_IN_USD'] = median_047
```

In [57]:
```
df['AMOUNT_IN_USD'].describe()
```

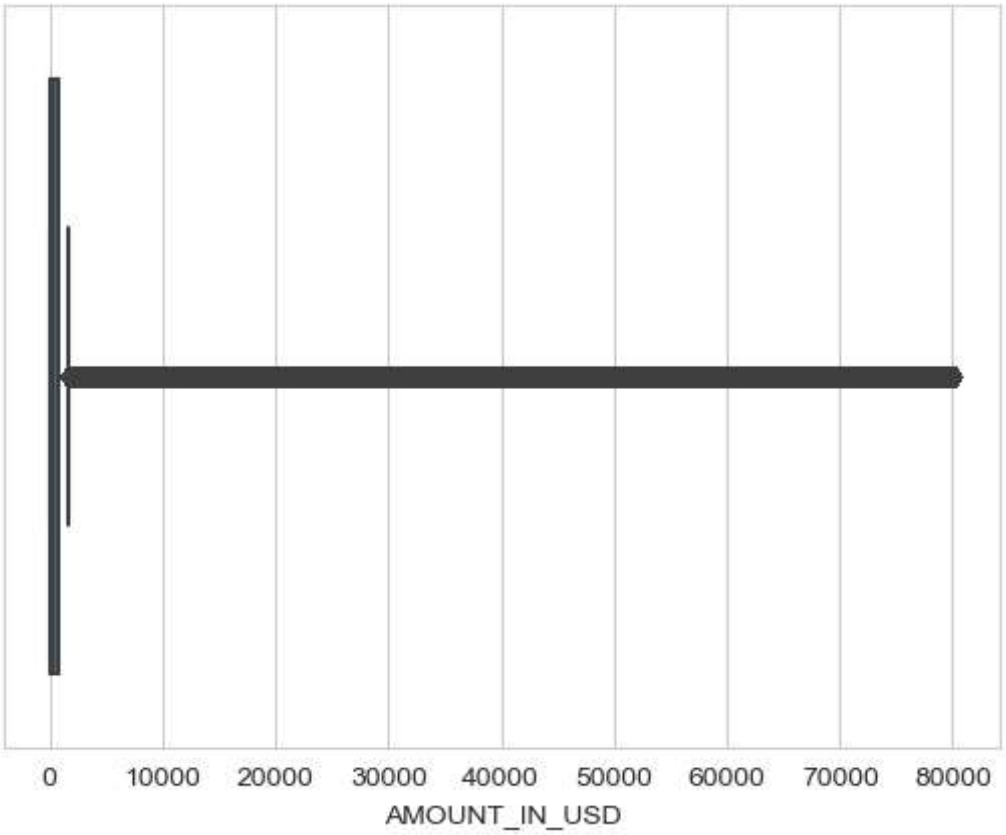
Out[57]:
```
count    1.074783e+06
mean     1.225215e+03
std      5.099231e+03
min      0.000000e+00
25%      1.360800e+01
50%      2.610000e+02
75%      7.140306e+02
max      8.017078e+04
Name: AMOUNT_IN_USD, dtype: float64
```

In [58]:
```
df['AMOUNT_IN_USD'].info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 1074783 entries, 0 to 1101924
Series name: AMOUNT_IN_USD
Non-Null Count    Dtype
--------------    -----
1074783 non-null  float64
dtypes: float64(1)
memory usage: 16.4 MB
```

In [59]:

```
1 sns.boxplot( x = df['AMOUNT_IN_USD'])
2 median_047 = df['AMOUNT_IN_USD'].median()
3 df['AMOUNT_IN_USD'] = df['AMOUNT_IN_USD'].mask(df['AMOUNT_IN_USD'] >
```



In [60]:

```
1 # We can also replace the outliers using Mean, Mode and maximum or mi
```

2. Label encoding or One hot Encoding on all the categorical columns

In [61]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1074783 entries, 0 to 1101924
Data columns (total 18 columns):
 #   Column                   Non-Null Count    Dtype         
---  ------                   --------------    -----         
 0   CUSTOMER_ORDER_ID        1074783 non-null  int64         
 1   SALES_ORG                1074783 non-null  int64         
 2   DISTRIBUTION_CHANNEL     1074783 non-null  object        
 3   DIVISION                 1074783 non-null  object        
 4   RELEASED_CREDIT_VALUE    1074783 non-null  object        
 5   PURCHASE_ORDER_TYPE      1074783 non-null  object        
 6   COMPANY_CODE             1074783 non-null  int64         
 7   ORDER_CREATION_DATE      1074783 non-null  datetime64[ns]
 8   ORDER_CREATION_TIME      1074783 non-null  int64         
 9   CREDIT_CONTROL_AREA      1074783 non-null  object        
 10  SOLD_TO_PARTY            1074783 non-null  int64         
 11  ORDER_AMOUNT             1074783 non-null  float64       
 12  REQUESTED_DELIVERY_DATE  1074783 non-null  datetime64[ns]
 13  ORDER_CURRENCY           1074783 non-null  object        
 14  CREDIT_STATUS            1074783 non-null  object        
 15  CUSTOMER_NUMBER          1074783 non-null  int64         
 16  AMOUNT_IN_USD            1074783 non-null  float64       
 17  UNIQUE_CUST_ID           1074783 non-null  object        
dtypes: datetime64[ns](2), float64(2), int64(6), object(8)
memory usage: 155.8+ MB
```

In [62]:
```
categorical_columns_047 = df.select_dtypes(include=['object', 'categor

# Print the categorical columns
print(len(categorical_columns_047))
print("Categorical columns:")
for col in categorical_columns_047:
    print(col)
```

```
8
Categorical columns:
DISTRIBUTION_CHANNEL
DIVISION
RELEASED_CREDIT_VALUE
PURCHASE_ORDER_TYPE
CREDIT_CONTROL_AREA
ORDER_CURRENCY
CREDIT_STATUS
UNIQUE_CUST_ID
```

In [63]:

```
# Select the categorical columns for label encoding

from sklearn.preprocessing import LabelEncoder

# Applying label encoding to the categorical columns

for col in categorical_columns_047:
    label_encoder047 = LabelEncoder()
    df[col] = label_encoder047.fit_transform(df[col])

```

In [64]:

```
# Unique values in each encoded columns
for col in categorical_columns_047:
    encoded_values = df[col].unique()
    print(f"Unique values in {col}: {encoded_values}")

```

```
Unique values in DISTRIBUTION_CHANNEL: [232 135 141 228  82   9  10 230
193 224 113  64 138  24  29  83 185  42
 213  94  97 109 114 234  92 131 121 120  58  16  40 154 102   3 122 173
  21  89  35  52 162 177  72 172 161   2  71  98 215  22  78 159 150 226
  80  90 152 210  25 145  81  41  69 196  91 110  56 132  37 149 205  77
 146   8 168 124  47 235 155  73 242  99 108 204 227 170  86  48 157 183
 169  20 237  57  93  84  34  17  33  55  31 163 128 107 143 111  32 118
  30  61 238  14 236 233 223  53 221 202 119 116 178 201 166  13  51 192
 153  50 147 142 206 198 212  49 137 115  76 136  45 190 207   4  36 188
 180  79 127 191  66 186  96   7  87 140 217 214 216  46 130 211  75  19
  68  88  54   0 160  11 231   5  62 144 156 220 134  15 197 125 106 240
 103 171 189 151 101 184 112  63 187 174  74  12 200  26 182 158 117 123
  67  27 181  70 105 133 195  18 208 100  38 164   6 194 175  43  28 167
 219  39 209 176 239 179  65 126  95 203  44 139  60 222  85 225 129 218
 241 148  23 229  59   1 199 165 104]
Unique values in DIVISION: [1 0]
Unique values in RELEASED_CREDIT_VALUE: [    1 43496  7852 ... 26040 147
09  7738]
Unique values in PURCHASE_ORDER_TYPE: [  0 204 255 270 291  56 264 156 3
42 307 334 158   3  75 101 104  35 340
 105 126 207 203 253 202  74  51 103 247  11  16  17  18  52  32  58  19
  15  13  50  84 172 157 169 164 171 174 244 127 331  47  71  76 200  98
  81  69 201  80 214 296 211  10 206 205 168 338 254  12 165  55  14 267
 266 170 215   4 265 132  36  83 310 137 134 138 133  73 109 113 118 257
  39 335 124 314 258 131 315  57 269 263  70 108 125 115 107 110 116 167
 112 117 175 341 199  34   1 111 336 163 311 271 272 273  79  67 213  59
  40  41 248 251 166 308  53 252 298 299 303 302 159 274 318 320 102 339
 268 292  66  33 136  82 332 289 223 212 313 317 312  64 220 114 219 216
 230 319  54 129 160 210 139 140 227 162  43  45 294 135 246  46  44 276
  65  68 337 225  48 231 316   2 222 218 278 145  49 176 250 224 228 177
 173 181 179  89 180 280 229  92  87  42  62 242  21  20 143 128  61  63
 208 178 333  95 217 297  37 152 233 141 304  85  86 275 209 281 144  38
 240 142  88 322 234 321  96  25  26  23  24  22 183  77 106 282 279 182
 277  60 245 288 120 121 123  94 260 119 122  97 290 328 329 323 324 190
 330 327 189 261 221 226 184 326 325 256  90   8 283  27 100  91 186 185
 191 295 262 146 192 301 147 300  78  93 193 187 198 149 293 197 284 196
 148 241 195 235 161 243 236  99 188 305 150 286 259 309 237 249 194 285
 130 232 238 239 153  31   5   9  29  30   7  28  72   6 151 287 154 344
 343 306 155 350 345 346 347 348 349]
Unique values in CREDIT_CONTROL_AREA: [5 2 0 4 3 1 7 6]
Unique values in ORDER_CURRENCY: [ 6  0  4  5  7 15 19 13  9 16  2 10 14
17  3  8  1 11 18 12]
Unique values in CREDIT_STATUS: [3 2 1 0]
Unique values in UNIQUE_CUST_ID: [6608 4405 4059 ... 3144 3096   56]
```

In [65]:
```
1 df.head()
```

Out[65]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEASED_C
0	946851639	3537	232	1	
1	963432061	3449	135	1	
2	971991639	3238	141	1	
3	754349803	3911	228	1	
4	930253442	2381	82	1	

In [66]:
```
1 from pandas.api.types import CategoricalDtype
2 for column in categorical_columns_047:
3     df[column] = df[column].astype(CategoricalDtype())
4
5 # Perform one-hot encoding on the categorical columns using SparseDat
6 one_hot_encoded_df_047 = pd.get_dummies(df, columns=categorical_colum
```

In [67]:
```
1 one_hot_encoded_df_047
```

Out[67]:

	CUSTOMER_ORDER_ID	SALES_ORG	COMPANY_CODE	ORDER_CREATION_DATE
0	946851639	3537	3220	2022-01-01
1	963432061	3449	3220	2022-01-01
2	971991639	3238	3260	2022-01-01
3	754349803	3911	3290	2022-01-01
4	930253442	2381	3290	2022-01-01
...	...	...	...	...
1101920	853605710	2498	4260	2022-06-01
1101921	998890898	4509	4260	2022-06-01
1101922	983330221	3951	4260	2022-06-01
1101923	926668029	3235	4260	2022-06-01
1101924	921701000	2968	4260	2022-06-01

1074783 rows × 54799 columns

In [68]:
```
1 df.shape
```

Out[68]:
```
(1074783, 18)
```

3. Log Transformations on continuous columns

In [69]:
```
# select_dtypes used for selecting only continuous values present in 
# Continuous values means numeric values like int64, float64

continuous_cols_047 = df.select_dtypes(include=['int64', 'float64']).
```

In [70]:
```
# Continuous columns present in our dataset
print(len(continuous_cols_047))
for i in continuous_cols_047:
    print(i)
```

```
8
CUSTOMER_ORDER_ID
SALES_ORG
COMPANY_CODE
ORDER_CREATION_TIME
SOLD_TO_PARTY
ORDER_AMOUNT
CUSTOMER_NUMBER
AMOUNT_IN_USD
```

In [71]:
```
# Apply log transformation to each continuous column

for col in continuous_cols_047:
    df[col] = np.log1p(df[col])
```

In [72]:
```
df
```

Out[72]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEA
0	20.668653	8.171317	232	1	
1	20.686013	8.146130	135	1	
2	20.694858	8.083020	141	1	
3	20.441367	8.271804	228	1	
4	20.650968	7.775696	82	1	
...	...	...	...	...	
1101920	20.564980	7.823646	79	1	
1101921	20.722156	8.414052	10	1	
1101922	20.706456	8.281977	151	1	
1101923	20.647106	8.082093	167	1	
1101924	20.641731	7.995980	156	1	

1074783 rows × 18 columns

4. Try to extract new features by grouping existing columns

In [73]:
```
1 df['total_order_amount'] = df.groupby('COMPANY_CODE')['ORDER_AMOUNT']
```

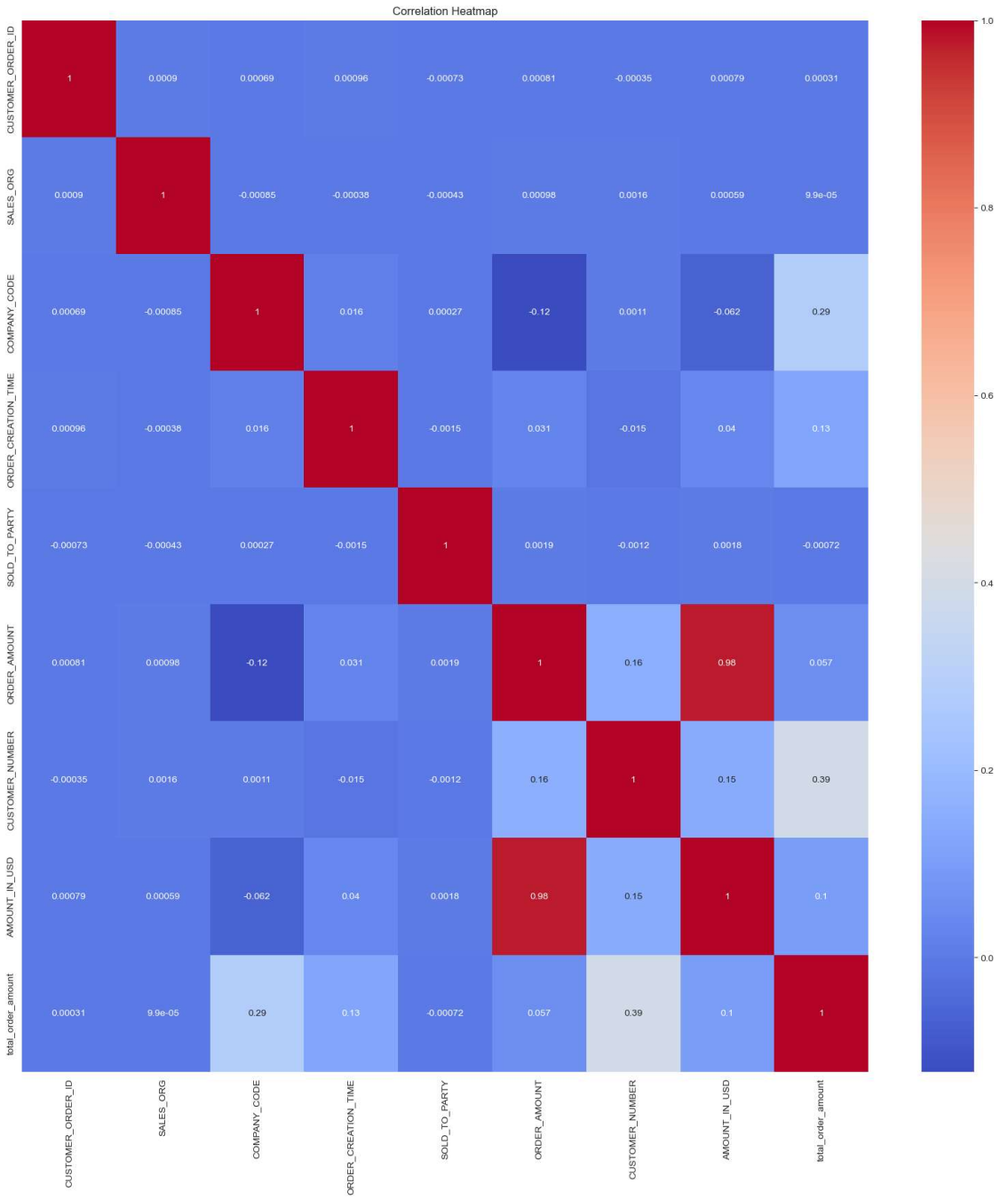
In [74]:
```
1 df['total_order_amount']
```

Out[74]:
```
0          367131.217302
1          367131.217302
2           80193.039645
3          107583.612158
4          107583.612158
               ...
1101920     34972.349714
1101921     34972.349714
1101922     34972.349714
1101923     34972.349714
1101924     34972.349714
Name: total_order_amount, Length: 1074783, dtype: float64
```

5. Create a heatmap to find correlation between the columns

In [75]:

```
plt.figure(figsize=(20,20))
# sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f", linew
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



6. Try to identify important or relevant columns for feature extraction

In [76]:
```
# So by calculating the correlation matrix we can extract highly corre

# It creates a list of tuples (high_corr_var047) where each tuple repr

corr_matrix047 = df.corr().abs()

high_corr_var047 = np.where(corr_matrix047 > 0.7)

high_corr_var047 = [(corr_matrix047.columns[x], corr_matrix047.columns

print(high_corr_var047)
```

```
[('ORDER_AMOUNT', 'AMOUNT_IN_USD')]
```

In [77]:
```
print(corr_matrix047.shape)
```

```
(9, 9)
```

In [78]:
```
# It's not recommended to take the highly correlated features as the m
# As if we take them then during training model may be overfitted.
# So we will try to check all the columns and estimate the required co
```

In [79]:

```
import pandas as pd

# Calculate the correlation matrix
corr_matrix047 = df.corr().abs()

# Set the correlation threshold
threshold_val = 0.5

# Create a list to store the relevant columns
relevant_cols047 = []

# Iterate over the columns
for col in corr_matrix047.columns:
    if col != 'AMOUNT_IN_USD':            # Exclude the target variab
        if any(corr_matrix047[col] >= threshold_val):  # Check if any
            relevant_cols047.append(col)

# Include the highly correlated features as well
# relevant_cols047.extend(['ORDER_AMOUNT'])

# Create a new DataFrame with only the relevant columns
extracted_features047 = df[relevant_cols047]

# Print the relevant columns
print('Relevant columns for feature extraction:', relevant_cols047)

```

```
Relevant columns for feature extraction: ['CUSTOMER_ORDER_ID', 'SALES_OR
G', 'COMPANY_CODE', 'ORDER_CREATION_TIME', 'SOLD_TO_PARTY', 'ORDER_AMOUN
T', 'CUSTOMER_NUMBER', 'total_order_amount']
```

In [ ]:

```

```

In [80]:

```
# We have to convert the 'Timestamp' column to numerical representati
```

In [81]:

```
# Convert 'REQUESTED_DELIVERY_DATE' column to numerical values
df['REQUESTED_DELIVERY_DATE'] = pd.to_numeric(df['REQUESTED_DELIVERY_

# Convert 'ORDER_CREATION_DATE' column to numerical values
df['ORDER_CREATION_DATE'] = pd.to_numeric(df['ORDER_CREATION_DATE'],
```

In [82]:
```
1 df
```

Out[82]:

	CUSTOMER_ORDER_ID	SALES_ORG	DISTRIBUTION_CHANNEL	DIVISION	RELEA
0	20.668653	8.171317	232	1	
1	20.686013	8.146130	135	1	
2	20.694858	8.083020	141	1	
3	20.441367	8.271804	228	1	
4	20.650968	7.775696	82	1	
...	...	...	...	...	
1101920	20.564980	7.823646	79	1	
1101921	20.722156	8.414052	10	1	
1101922	20.706456	8.281977	151	1	
1101923	20.647106	8.082093	167	1	
1101924	20.641731	7.995980	156	1	

1074783 rows × 19 columns

In [83]:

```
# Split the dataset into X (input features) and y (target variable)

x = df.drop('AMOUNT_IN_USD', axis=1)
y = df['AMOUNT_IN_USD']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0

# Perform feature selection using SelectKBest and f_regression
selector = SelectKBest(f_regression)
selector.fit(x_train, y_train)

# Get the selected features
selected_features047 = x_train.columns[selector.get_support()]

# Create an instance of the linear regression model
model = LinearRegression()

# Define the parameter grid for grid search
param_grid = {
    'fit_intercept': [True, False]
}

# Create an instance of GridSearchCV
grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squar

# Fit the GridSearchCV to the training data
grid_search.fit(x_train[selected_features047], y_train)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions on the test data
y_pred = best_model.predict(x_test[selected_features047])

# Calculate error metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Best Features:", selected_features047)
print("Best Parameters:", grid_search.best_params_)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
```

```
Best Features: Index(['DIVISION', 'RELEASED_CREDIT_VALUE', 'PURCHASE_ORD
ER_TYPE',
       'COMPANY_CODE', 'ORDER_CREATION_TIME', 'ORDER_AMOUNT', 'ORDER_CUR
RENCY',
       'CREDIT_STATUS', 'CUSTOMER_NUMBER', 'total_order_amount'],
      dtype='object')
Best Parameters: {'fit_intercept': True}
Mean Squared Error: 0.3118298006685517
Root Mean Squared Error: 0.558417228126561
Mean Absolute Error: 0.23566797298096293
R2 Score: 0.9609444355571939
```

So here from GridSearchCV and using SelectKBest, we got the best features among the whole dataset.

Now we will check each selected feature importance in this prediction task like how they are matured to be fit with the target variable Based on that we will take them accordingly for model training

In [84]:

```
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder

selected_features047 = ['DIVISION', 'RELEASED_CREDIT_VALUE', 'COMPANY_
       'ORDER_CREATION_TIME', 'REQUESTED_DELIVERY_DATE', 'CREDIT_STATU
       'CUSTOMER_NUMBER', 'AMOUNT_IN_USD', 'UNIQUE_CUST_ID',
       'total_order_amount']

df_selected = df[selected_features047].copy()

# Convert categorical columns to "category" dtype
categorical_columns = ['DIVISION', 'RELEASED_CREDIT_VALUE', 'CREDIT_ST
for col in categorical_columns:
    df_selected[col] = df_selected[col].astype('category')

# Apply label encoding to the categorical columns
label_encoder = LabelEncoder()
for col in df_selected.columns:
    if df_selected[col].dtype.name == 'category':
        df_selected[col] = label_encoder.fit_transform(df_selected[col

x = df_selected.drop('AMOUNT_IN_USD',axis=1)
y = df_selected['AMOUNT_IN_USD']

# XGBoost Model
model = xgb.XGBRegressor()
model.fit(x, y)

# Feature importance
importance = model.feature_importances_

# Get the names of the features
feature_names = x.columns

feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Impor

# Sorting the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importa

print(feature_importance_df)

```

	Feature	Importance
0	DIVISION	0.307417
8	total_order_amount	0.170724
5	CREDIT_STATUS	0.121894
1	RELEASED_CREDIT_VALUE	0.090676
7	UNIQUE_CUST_ID	0.089152
6	CUSTOMER_NUMBER	0.065589
2	COMPANY_CODE	0.055264
4	REQUESTED_DELIVERY_DATE	0.051235
3	ORDER_CREATION_TIME	0.048049

In [85]:
```
print(feature_names)
```

```
Index(['DIVISION', 'RELEASED_CREDIT_VALUE', 'COMPANY_CODE',
       'ORDER_CREATION_TIME', 'REQUESTED_DELIVERY_DATE', 'CREDIT_STATU
S',
       'CUSTOMER_NUMBER', 'UNIQUE_CUST_ID', 'total_order_amount'],
      dtype='object')
```

In [86]:
```
print(selected_features047)
```

```
['DIVISION', 'RELEASED_CREDIT_VALUE', 'COMPANY_CODE', 'ORDER_CREATION_TI
ME', 'REQUESTED_DELIVERY_DATE', 'CREDIT_STATUS', 'CUSTOMER_NUMBER', 'AMO
UNT_IN_USD', 'UNIQUE_CUST_ID', 'total_order_amount']
```

In [ ]:
```

```

In [ ]:
```

```

MILESTONE - 4 [MODEL TRAINING & EVALUATIONS]

In [87]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.utils import all_estimators
from sklearn.metrics import accuracy_score, precision_score, recall_s
from sklearn.model_selection import GridSearchCV

%matplotlib inline
```

1-2-3. Try different machine learning models Linear Regression, Support Vector Machine, Decision Tree,

Random Forest, AdaBoost and lightGBM etc. as well as Calculate the Error Metrics like MSE, RMSE, MAE,

and R2 Score for checking best model.

IN THIS QUESTION, BASICALLY I CHECKED FOR NORMAL TRAIN TEST SPLITTING DATA AND THEN IN NEXT PHASE I SPLITTED THE DATASET USING TIME BASED ANALYSIS.

BELOW, 6 MODELS ARE PERFORMED FOR NORMAL TRAIN TEST SPLIT DATA.

NORMAL TRAIN TEST SPLIT

In [88]:
```
# All Dependencies imported at the first of this .ipnyb
```

In [89]:
```
# Splitting the data into relevant features(x) and target variable(y)

x = df[feature_names]
y = df['AMOUNT_IN_USD']
```

In [90]:
```
# Splitting the data into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = (
```

Linear Regression

In [91]:
```
model_lr = LinearRegression()
model_lr.fit(x_train, y_train)

y_pred = model_lr.predict(x_test)        # Prediction on test data


# Calculation of Error metrics
mse_linear = mean_squared_error(y_test, y_pred)
rmse_linear = np.sqrt(mse_linear)
mae_linear = mean_absolute_error(y_test, y_pred)
r2_linear = r2_score(y_test, y_pred)

print("Mean Square Error: ", mse_linear)
print("Root Means Square Error: ", rmse_linear)
print("Mean Absolute Error: ", mae_linear)
print("R2 Score: ", r2_linear)
```

```
Mean Square Error:  7.596969645792412
Root Means Square Error:  2.756260083118502
Mean Absolute Error:  2.251946232024614
R2 Score:  0.04872669239148886
```

In [92]:
```
train_pred_lr = model_lr.predict(x_train)
r2_linear_train = r2_score(y_train, train_pred_lr)
print(r2_linear_train)
print(r2_linear)
```

```
0.05816417974578303
0.04872669239148886
```

SVM

In [93]:

```
# As all the columns present in x variable have linear relationship Sc
from sklearn.svm import LinearSVR

model_svm = LinearSVR()
model_svm.fit(x_train, y_train)
y_pred = model_svm.predict(x_test)

# Error metrics
mse_svm = mean_squared_error(y_test, y_pred)
rmse_svm = np.sqrt(mse)
mae_svm = mean_absolute_error(y_test, y_pred)
r2_svm = r2_score(y_test, y_pred)

print("Mean Squared Error: ", mse_svm)
print("Root Mean Sqaured Error: ", rmse_svm)
print("Mean Absolute Error: ", mae_svm)
print("R2 score: ", r2_svm)
```

```
Mean Squared Error:  29.466621030530437
Root Mean Sqaured Error:  0.558417228126561
Mean Absolute Error:  4.6347076331999295
R2 score:  -2.689735691820763

C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\svm\_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of it
erations.
  warnings.warn(
```

In [94]:

```
train_pred_svm = model_svm.predict(x_train)
r2_svm_train = r2_score(y_train, train_pred_svm)
print(r2_svm_train)
print(r2_svm)
```

```
-2.697086814130243
-2.689735691820763
```

Decision Tree Regression

In [95]:

```
# Create an instance of the Decision Tree Regression model
model_dt = DecisionTreeRegressor()

# Train the model on the training data
model_dt.fit(x_train, y_train)

# Make predictions on the test data
y_pred = model_dt.predict(x_test)

# Evaluate the model's performance
mse_dt = mean_squared_error(y_test, y_pred)
rmse_dt = np.sqrt(mse_dt)
mae_dt = mean_absolute_error(y_test, y_pred)
r2_dt = r2_score(y_test, y_pred)


print("Mean Squared Error: ", mse_dt)
print("Root Means Square error: ", rmse_dt)
print("Mean Absolute error: ", mae_dt)
print("r2 score: ", r2_dt)
```

```
Mean Squared Error:  2.6089869689691447
Root Means Square error:  1.6152358864788587
Mean Absolute error:  0.8874706433969508
r2 score:  0.6733092562962442
```

In [96]:

```
train_pred_dt = model_dt.predict(x_train)
r2_dt_train = r2_score(y_train, train_pred_dt)
print(r2_dt_train)
print(r2_dt)
```

```
0.9447359875666728
0.6733092562962442
```

Random Forest Regression

In [97]:

```
# # Create an instance of the Random Forest Regression model
model_rfc = RandomForestRegressor()

# Train the model on the training data
model_rfc.fit(x_train, y_train)

# Make predictions on the test data
y_pred = model_rfc.predict(x_test)

# Evaluate the model's performance
mse_rfc = mean_squared_error(y_test, y_pred)
rmse_rfc = np.sqrt(mse_rfc)
mae_rfc = mean_absolute_error(y_test, y_pred)
r2_rfc = r2_score(y_test, y_pred)


print("Mean Squared Error: ", mse_rfc)
print("Root Means Square error: ", rmse_rfc)
print("Mean Absolute error: ", mae_rfc)
print("r2 score: ", r2_rfc)
```

```
Mean Squared Error:  1.9230466818174103
Root Means Square error:  1.3867395868790255
Mean Absolute error:  0.8001208474780986
r2 score:  0.759200962621826
```

In [98]:

```
train_pred_rfc = model_rfc.predict(x_train)
r2_rfc_train = r2_score(y_train, train_pred_rfc)
print(r2_rfc_train)
print(r2_rfc)
```

```
0.9243298106655135
0.759200962621826
```

AdaBoost

In [99]:

```
model_ab = AdaBoostRegressor()

# Train the model on the training data
model_ab.fit(x_train, y_train)

# Make predictions on the test data
y_pred = model_ab.predict(x_test)


# Evaluate the model's performance
mse_ab = mean_squared_error(y_test, y_pred)
rmse_ab = np.sqrt(mse_ab)
mae_ab = mean_absolute_error(y_test, y_pred)
r2_ab = r2_score(y_test, y_pred)


print("Mean Squared Error: ", mse_ab)
print("Root Means Square error: ", rmse_ab)
print("Mean Absolute error: ", mae_ab)
print("r2 score: ", r2_ab)
```

```
Mean Squared Error:  6.560119778457941
Root Means Square error:  2.5612730776818666
Mean Absolute error:  2.1031820982779625
r2 score:  0.1785584080333944
```

In [100]:

```
train_pred_ab = model_ab.predict(x_train)
r2_ab_train = r2_score(y_train, train_pred_ab)
print(r2_ab_train)
print(r2_ab)
```

```
0.17841485853015815
0.1785584080333944
```

LightGBM Method

In [101]:

```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\kiit\anaconda3\lib\site-packages (3.3.5)
Requirement already satisfied: wheel in c:\users\kiit\anaconda3\lib\site-packages (from lightgbm) (0.38.4)
Requirement already satisfied: numpy in c:\users\kiit\anaconda3\lib\site-packages (from lightgbm) (1.23.5)
Requirement already satisfied: scipy in c:\users\kiit\anaconda3\lib\site-packages (from lightgbm) (1.10.1)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\users\kiit\anaconda3\lib\site-packages (from lightgbm) (1.2.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\kiit\anaconda3\lib\site-packages (from scikit-learn!=0.22.0->lightgbm) (2.2.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\kiit\anaconda3\lib\site-packages (from scikit-learn!=0.22.0->lightgbm) (1.2.0)

WARNING: Ignoring invalid distribution -cikit-learn (c:\users\kiit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -cikit-learn (c:\users\kiit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -cikit-learn (c:\users\kiit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -cikit-learn (c:\users\kiit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -cikit-learn (c:\users\kiit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -cikit-learn (c:\users\kiit\anaconda3\lib\site-packages)
```

In [102]:

```
import lightgbm as lgb
import math

# Create and train the LightGBM regressor model
model_lgbm = lgb.LGBMRegressor()
model_lgbm.fit(x_train, y_train)

y_pred = model_lgbm.predict(x_test)

# Evaluation metrics
mse_lgbm = mean_squared_error(y_test, y_pred)
rmse_lgbm = np.sqrt(mse_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred)
r2_lgbm = r2_score(y_test, y_pred)

print("MSE:", mse_lgbm)
print("RMSE:", rmse_lgbm)
print("MAE:", mae_lgbm)
print("R2 score:", r2_lgbm)
```

```
MSE: 2.565912392263618
RMSE: 1.6018465570283622
MAE: 1.0811287880942089
R2 score: 0.6787029457496694
```

In [103]:

```
train_pred_lgbm = model_lgbm.predict(x_train)
r2_lgbm_train = r2_score(y_train, train_pred_lgbm)
print(r2_lgbm_train)
print(r2_lgbm)
```

```
0.6839296007841718
0.6787029457496694
```

In [ ]:

```

```

In [ ]:

```

```

TIME BASED TRAIN TEST SPLIT

Now here we will try to optimize or train_test_split by using time series data analysis. Because our dataset is time based So we have to predict future unknown data on according to the past data.

In [104]:

```
df_sorted = df.sort_values('ORDER_CREATION_TIME')

# Define the train-test split ratio
train_size = 0.8  # 80% of the data for training, 20% for testing

# Calculate the index to split the data
split_index = int(train_size * len(df_sorted))

# Split the data into training and testing sets
train_data = df_sorted[:split_index]
test_data = df_sorted[split_index:]

# Separate the features and target variables
x_train_time = train_data.drop(['AMOUNT_IN_USD'], axis=1)
y_train_time = train_data['AMOUNT_IN_USD']

x_test_time = test_data.drop(['AMOUNT_IN_USD'], axis=1)
y_test_time = test_data['AMOUNT_IN_USD']

```

In [105]:

```
# Testing only for LinearRegression

model_time_lr = LinearRegression()
model_time_lr.fit(x_train_time, y_train_time)

# Make predictions on the test set
y_pred_time = model_time_lr.predict(x_test_time)

# Calculate the evaluation metrics
mse_time_lr = mean_squared_error(y_test_time, y_pred_time)
rmse_time_lr = mean_squared_error(y_test_time, y_pred_time, squared=Fa
mae_time_lr = mean_absolute_error(y_test_time, y_pred_time)
r2_time_lr = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("MSE:", mse_time_lr)
print("RMSE:", rmse_time_lr)
print("MAE:", mae_time_lr)
print("R2 score:", r2_time_lr)
```

```
MSE: 0.221365556993468
RMSE: 0.470495012719017
MAE: 0.2248728252208013
R2 score: 0.9701739332576491
```

In [106]:

```
train_pred = model_time_lr.predict(x_train_time)

r2_train = r2_score(y_train_time, train_pred)

print(r2_train)
print(r2_time_lr)
```

```
0.9559416337552149
0.9701739332576491
```

As the r2_train and r2_test value are coming closer to each other And difference is very less, testing score is slighlt higher than training ,So model is performing well in liner regression.

In [107]:

```
# Testing only for svm

model_time_svm = LinearSVR()
model_time_svm.fit(x_train_time, y_train_time)

# Make predictions on the test set
y_pred_time = model_time_svm.predict(x_test_time)

# Calculate the evaluation metrics
mse_time_svm = mean_squared_error(y_test_time, y_pred_time)
rmse_time_svm = mean_squared_error(y_test_time, y_pred_time, squared=
mae_time_svm = mean_absolute_error(y_test_time, y_pred_time)
r2_time_svm = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("MSE:", mse_time_svm)
print("RMSE:", rmse_time_svm)
print("MAE:", mae_time_svm)
print("R2 score:", r2_time_svm)
```

```
MSE: 28.32586372594996
RMSE: 5.322204780535033
MAE: 4.5720872061447375
R2 score: -2.8165336717213343

C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\svm\_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of it
erations.
  warnings.warn(
```

In [108]:

```
train_pred = model_time_svm.predict(x_train_time)

r2_train = r2_score(y_train_time, train_pred)

print(r2_train)
print(r2_time_svm)
```

```
-2.668714382520475
-2.8165336717213343
```

In case of SVM, training and testing accuracy of model is very poor. So model is not properly fitted and performing worse in this scenario.

In [109]:

```python
# Testing only for Decision tree

model_time_dt = DecisionTreeRegressor()
model_time_dt.fit(x_train_time, y_train_time)

# Make predictions on the test set
y_pred_time = model_time_dt.predict(x_test_time)

# Calculate the evaluation metrics
mse_time_dt = mean_squared_error(y_test_time, y_pred_time)
rmse_time_dt = mean_squared_error(y_test_time, y_pred_time, squared=F
mae_time_dt = mean_absolute_error(y_test_time, y_pred_time)
r2_time_dt = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("MSE:", mse_time_dt)
print("RMSE:", rmse_time_dt)
print("MAE:", mae_time_dt)
print("R2 score:", r2_time_dt)
```

```
MSE: 0.001961133741027821
RMSE: 0.04428468969099615
MAE: 0.0010056815743453161
R2 score: 0.9997357632928763
```

In [110]:

```python
train_pred = model_time_dt.predict(x_train_time)

r2_train = r2_score(y_train_time, train_pred)

print(r2_train)
print(r2_time_dt)
```

```
0.9999999999999998
0.9997357632928763
```

Decision trees have the capability to learn intricate patterns and details from the training data, which can lead to high accuracy scores. I also checked for cross validation set performance in decision tree, and ensured model is not overfitted.

In [111]:

```python
# Testing only for Random Forest

model_time_rfc = RandomForestRegressor()
model_time_rfc.fit(x_train_time, y_train_time)

# Make predictions on the test set
y_pred_time = model_time_rfc.predict(x_test_time)

# Calculate the evaluation metrics
mse_time_rfc = mean_squared_error(y_test_time, y_pred_time)
rmse_time_rfc = mean_squared_error(y_test_time, y_pred_time, squared=
mae_time_rfc = mean_absolute_error(y_test_time, y_pred_time)
r2_time_rfc = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("MSE:", mse_time_rfc)
print("RMSE:", rmse_time_rfc)
print("MAE:", mae_time_rfc)
print("R2 score:", r2_time_rfc)
```

```
MSE: 0.0009991046518026216
RMSE: 0.03160861673345769
MAE: 0.0021972979188614856
R2 score: 0.9998653839267861
```

In [112]:

```python
# Random Forest Accuracy checking in time based

train_pred = model_time_rfc.predict(x_train_time)

r2_train = r2_score(y_train_time, train_pred)

print(r2_train)
print(r2_time_rfc)
```

```
0.9999701405696346
0.9998653839267861
```

In [113]:

```python
# Testing only for AdaBoost Model in time based

model_time_ab = AdaBoostRegressor()
model_time_ab.fit(x_train_time, y_train_time)

# Make predictions on the test set
y_pred_time = model_time_ab.predict(x_test_time)

# Calculate the evaluation metrics
mse_time_ab = mean_squared_error(y_test_time, y_pred_time)
rmse_time_ab = mean_squared_error(y_test_time, y_pred_time, squared=Fa
mae_time_ab = mean_absolute_error(y_test_time, y_pred_time)
r2_time_ab = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("MSE:", mse_time_ab)
print("RMSE:", rmse_time_ab)
print("MAE:", mae_time_ab)
print("R2 score:", r2_time_ab)
```

```
MSE: 0.22208422627642962
RMSE: 0.4712581312576258
MAE: 0.29949013477743636
R2 score: 0.9700771021232558
```

In [114]:

```python
train_pred = model_time_ab.predict(x_train_time)

r2_train = r2_score(y_train_time, train_pred)

print(r2_train)
print(r2_time_ab)
```

```
0.9642171369729288
0.9700771021232558
```

In [115]:

```
# Testing only for lightGBM model

import lightgbm as lgb
import math

model_time_l = lgb.LGBMRegressor()
model_time_l.fit(x_train_time, y_train_time)

# Make predictions on the test set
y_pred_time = model_time_l.predict(x_test_time)

# Calculate the evaluation metrics
mse_time_l = mean_squared_error(y_test_time, y_pred_time)
rmse_time_l = mean_squared_error(y_test_time, y_pred_time, squared=Fa
mae_time_l = mean_absolute_error(y_test_time, y_pred_time)
r2_time_l = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("MSE:", mse_time_l)
print("RMSE:", rmse_time_l)
print("MAE:", mae_time_l)
print("R2 score:", r2_time_l)

```

```
MSE: 0.009826615919778308
RMSE: 0.09912928890987924
MAE: 0.01586414632506444
R2 score: 0.9986759941056078
```

In [116]:

```
train_pred = model_time_l.predict(x_train_time)

r2_train = r2_score(y_train_time, train_pred)

print(r2_train)
print(r2_time_l)
```

```
0.9990601673411479
0.9986759941056078
```

In [ ]:

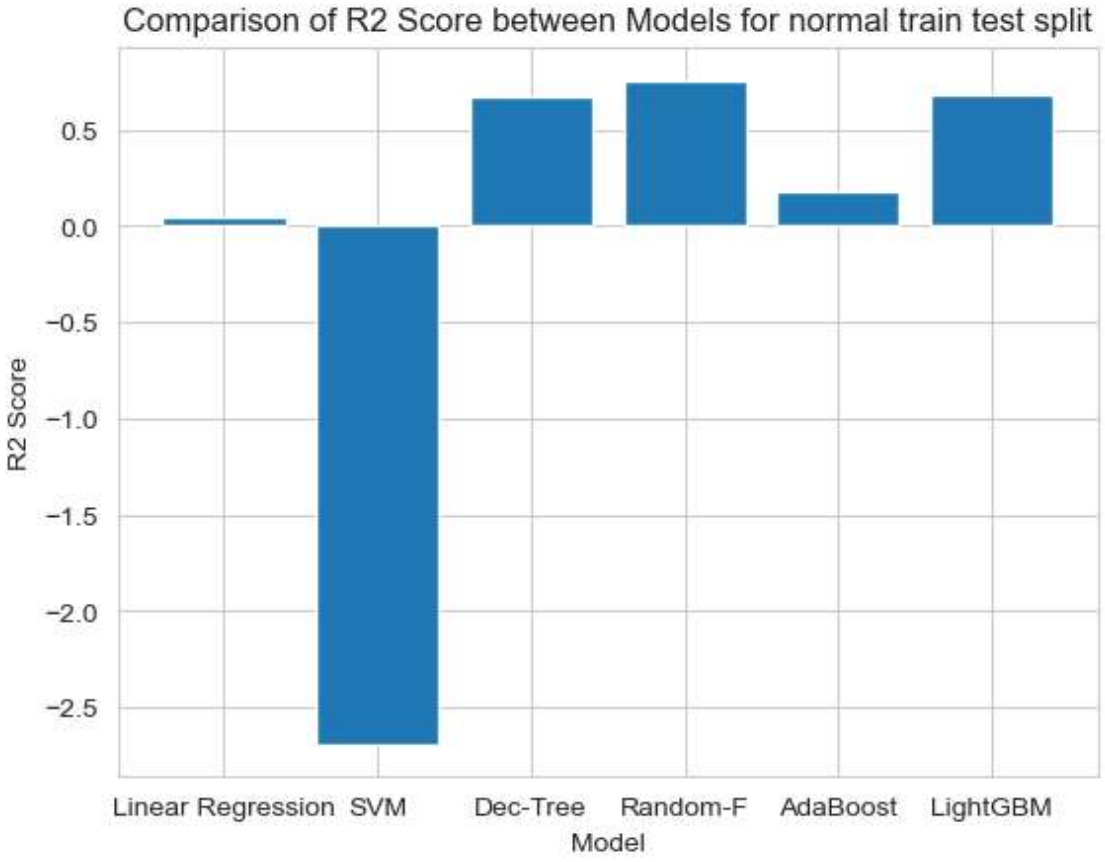
In [ ]:

4. Comparisional Analysis from visualisation of different Models for Model accuracy checking

Visualizations of different models error metrics for better insight gain in case of normal train test split

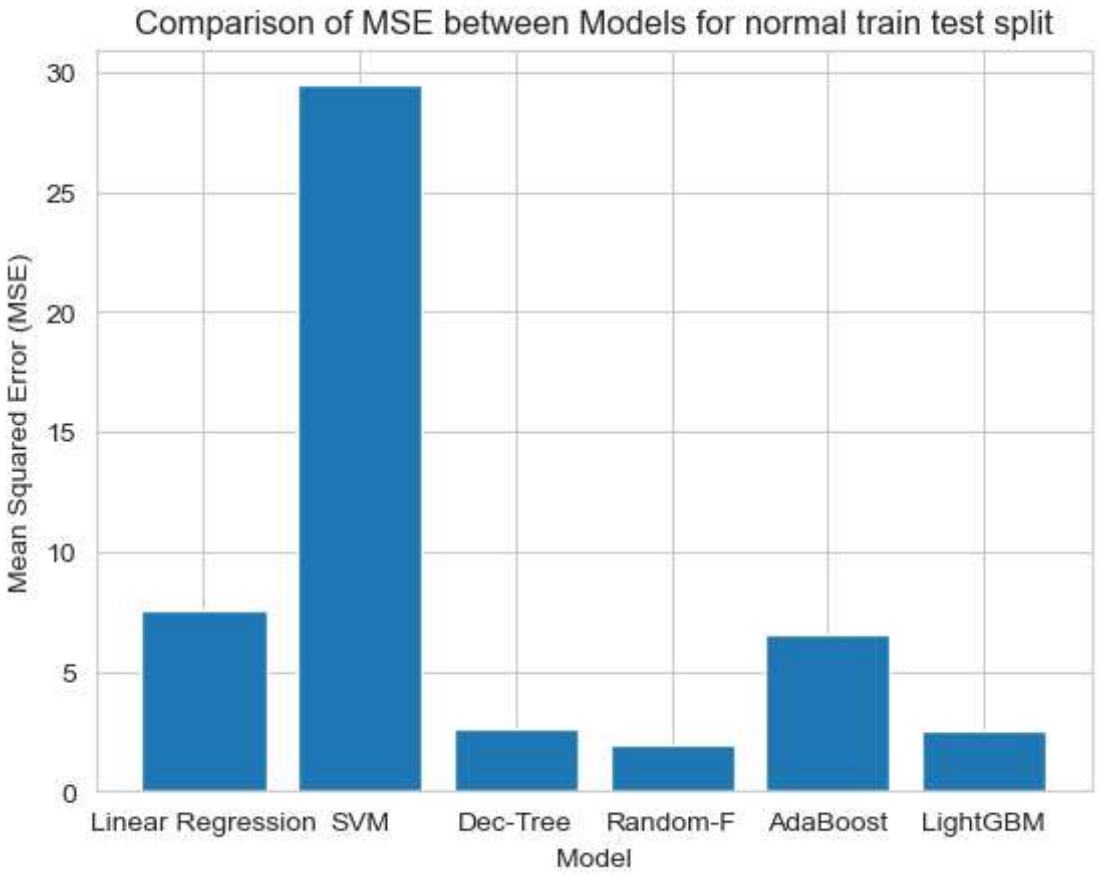
In [117]:

```
# Comparison of R2 score of diff models

import matplotlib.pyplot as plt

# Define the error metrics for each model
r2_values = [r2_linear, r2_svm, r2_dt, r2_rfc, r2_ab, r2_lgbm]

# Define the model names
model_names = ['Linear Regression', 'SVM', 'Dec-Tree', 'Random-F', 'A

# Create a bar plot
plt.bar(model_names, r2_values)
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Comparison of R2 Score between Models for normal train tes
plt.show()
```



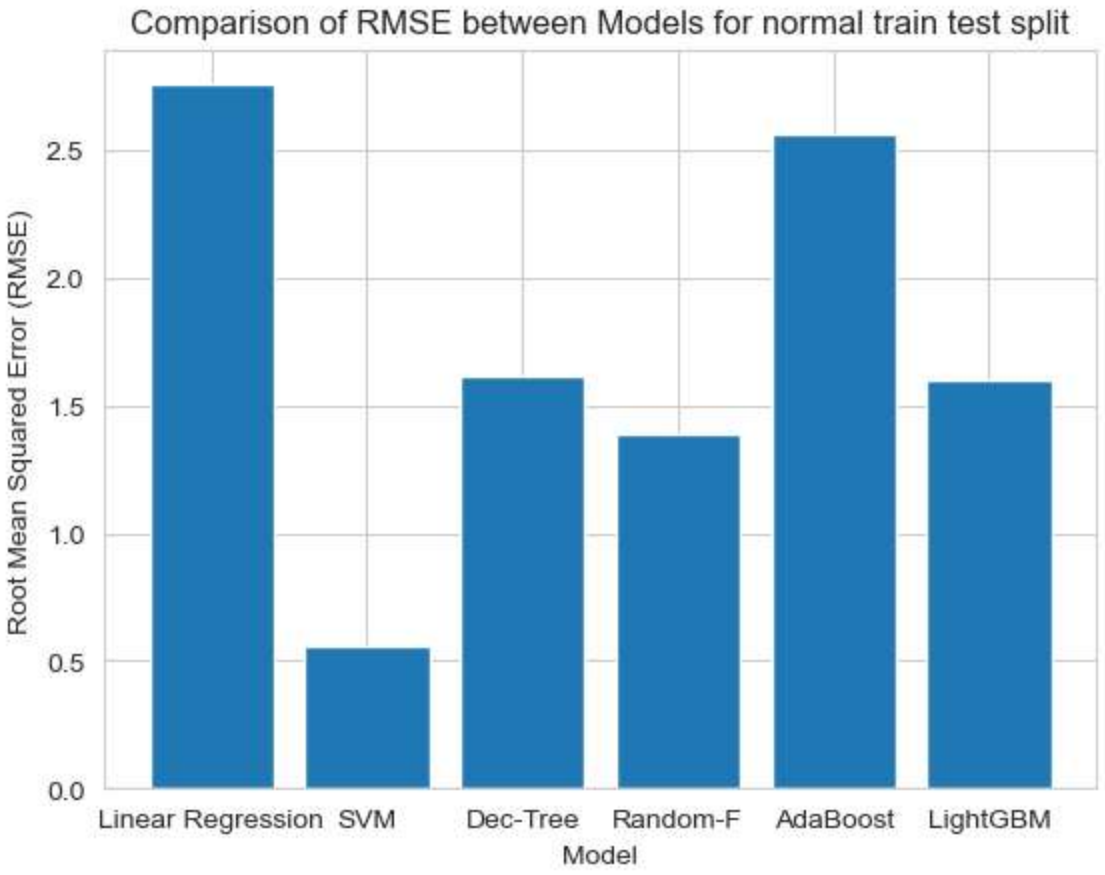
In [118]:

```
# Comparison of MSE of diff models
import matplotlib.pyplot as plt

# Define the error metrics for each model
mse_values = [mse_linear, mse_svm, mse_dt, mse_rfc, mse_ab, mse_lgbm]

# Define the model names
model_names = ['Linear Regression', 'SVM', 'Dec-Tree', 'Random-F', 'A

# Create a bar plot
plt.bar(model_names, mse_values)
plt.xlabel('Model')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Comparison of MSE between Models for normal train test spli
plt.show()
```



In [119]:

```
# Comparison of RMSE of diff models
import matplotlib.pyplot as plt

# Define the error metrics for each model
rmse_values = [rmse_linear, rmse_svm, rmse_dt, rmse_rfc, rmse_ab, rms

# Define the model names
model_names = ['Linear Regression', 'SVM', 'Dec-Tree', 'Random-F', 'A

# Create a bar plot
plt.bar(model_names, rmse_values)
plt.xlabel('Model')
plt.ylabel('Root Mean Squared Error (RMSE)')
plt.title('Comparison of RMSE between Models for normal train test sp
plt.show()
```



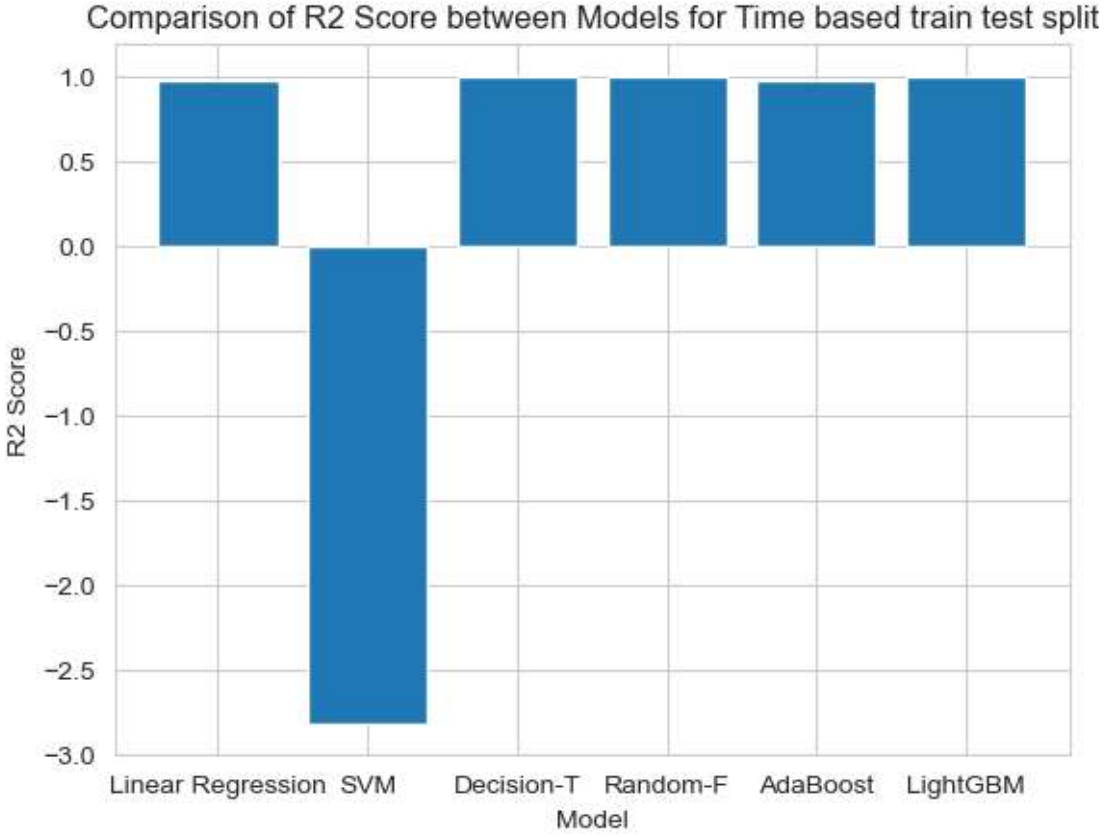
In [ ]:

```
1
```

Visualizations of different models error metrics for better insight gain in case of Time based train test split

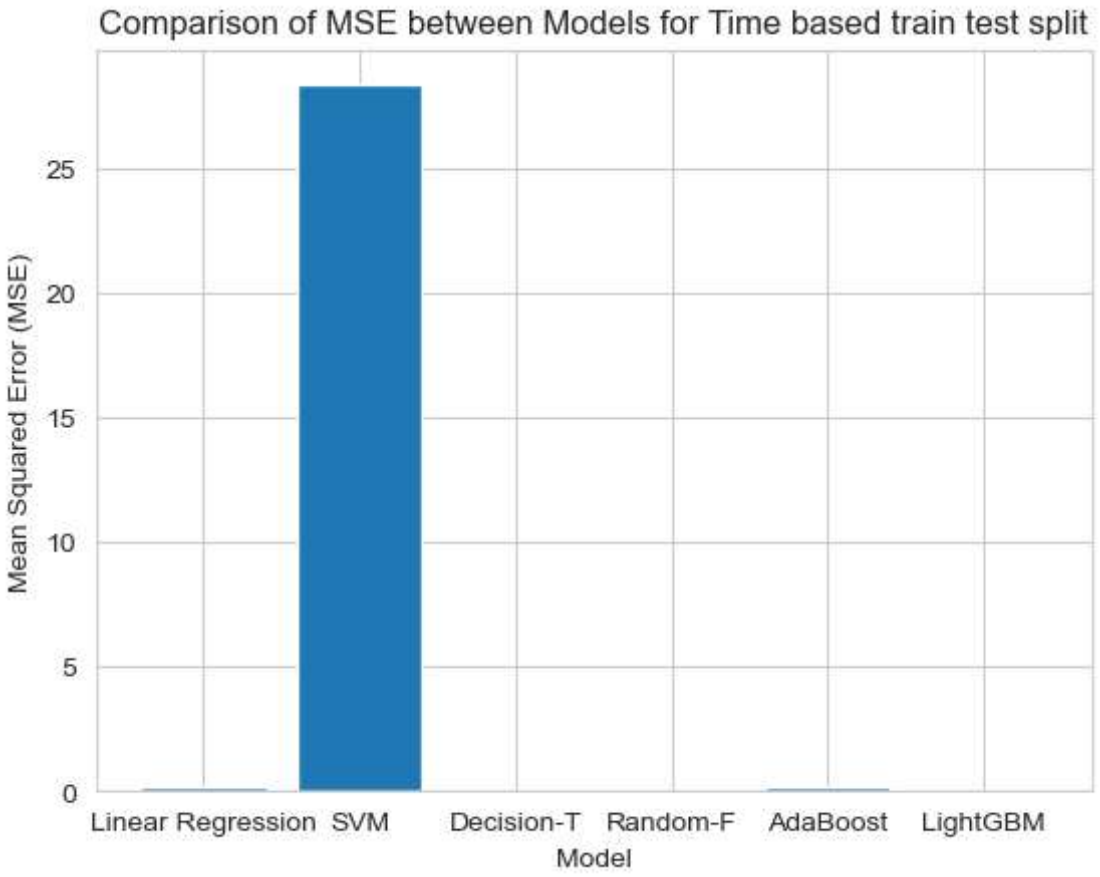
In [120]:

```
# Comparison of R2 score of diff models

import matplotlib.pyplot as plt

# Define the error metrics for each model
r2_values = [r2_time_lr, r2_time_svm, r2_time_dt, r2_time_rfc, r2_time

# Define the model names
model_names = ['Linear Regression', 'SVM', 'Decision-T', 'Random-F',

# Create a bar plot
plt.bar(model_names, r2_values)
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Comparison of R2 Score between Models for Time based train
plt.show()
```



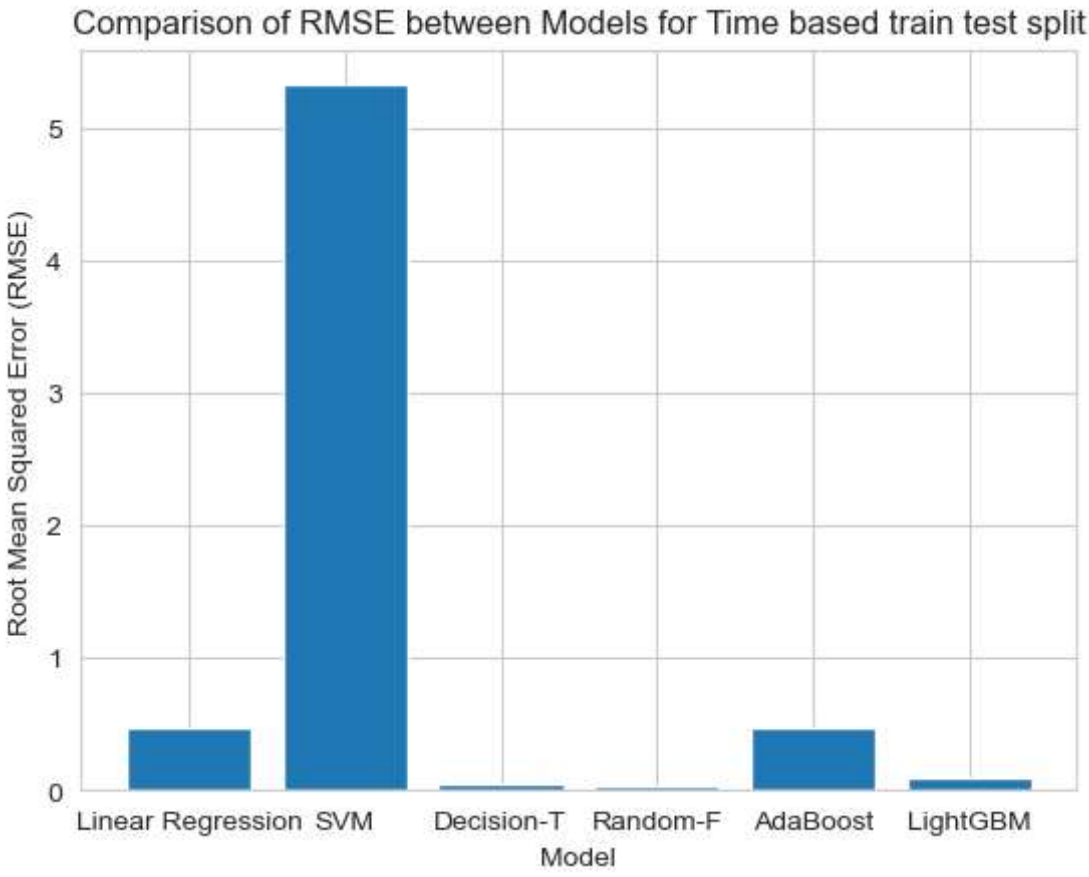
In [121]:

```
# Comparison of MSE of diff models
import matplotlib.pyplot as plt

# Define the error metrics for each model
mse_values_time = [mse_time_lr,mse_time_svm, mse_time_dt, mse_time_rf

# Define the model names
model_names = ['Linear Regression', 'SVM', 'Decision-T', 'Random-F',

# Create a bar plot
plt.bar(model_names, mse_values_time)
plt.xlabel('Model')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Comparison of MSE between Models for Time based train test
plt.show()
```



In [122]:

```
# Comparison of RMSE of diff models
import matplotlib.pyplot as plt

# Define the error metrics for each model
rmse_values = [rmse_time_lr, rmse_time_svm, rmse_time_dt, rmse_time_r

# Define the model names
model_names = ['Linear Regression', 'SVM', 'Decision-T', 'Random-F',

# Create a bar plot
plt.bar(model_names, rmse_values)
plt.xlabel('Model')
plt.ylabel('Root Mean Squared Error (RMSE)')
plt.title('Comparison of RMSE between Models for Time based train tes
plt.show()
```



In [ ]:

```
```

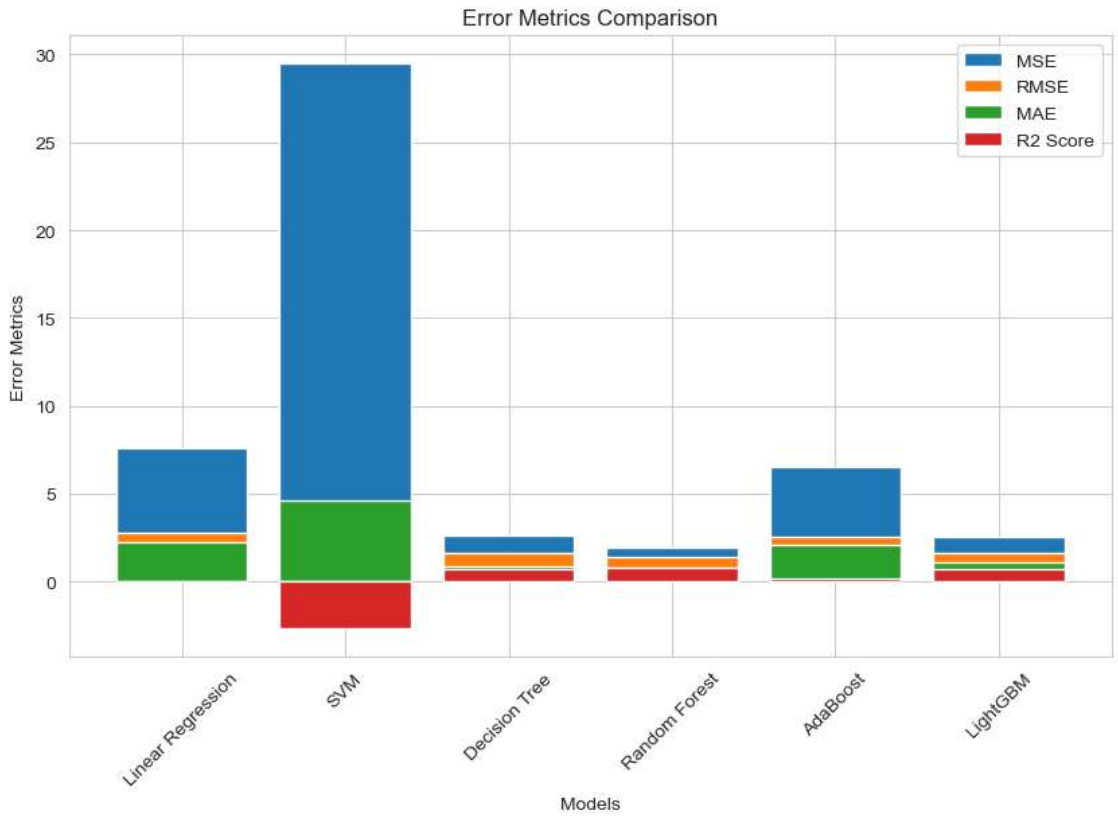
In [123]:

```
# Comparisional analysis between Normal Train test split accuracy and
```

NORMAL TRAIN TEST SPLIT ACCURACY CHECKING FOR DIFFERENT MODELS (taking all errors together)

In [124]:

```
import matplotlib.pyplot as plt

# Define the models and their corresponding error metrics
models = ['Linear Regression','SVM', 'Decision Tree', 'Random Forest',
mse = [mse_linear, mse_svm, mse_dt, mse_rfc, mse_ab, mse_lgbm]
rmse = [rmse_linear, rmse_svm, rmse_dt, rmse_rfc, rmse_ab, rmse_lgbm]
mae = [mae_linear, mae_svm, mae_dt, mae_rfc, mae_ab, mae_lgbm]
r2 = [r2_linear, r2_svm, r2_dt, r2_rfc, r2_ab, r2_lgbm]

# Plot the error metrics for each model
plt.figure(figsize=(10, 6))
plt.bar(models, mse, label='MSE')
plt.bar(models, rmse, label='RMSE')
plt.bar(models, mae, label='MAE')
plt.bar(models, r2, label='R2 Score')
plt.xlabel('Models')
plt.ylabel('Error Metrics')
plt.title('Error Metrics Comparison')
plt.legend()
plt.xticks(rotation=45)
plt.show()

```



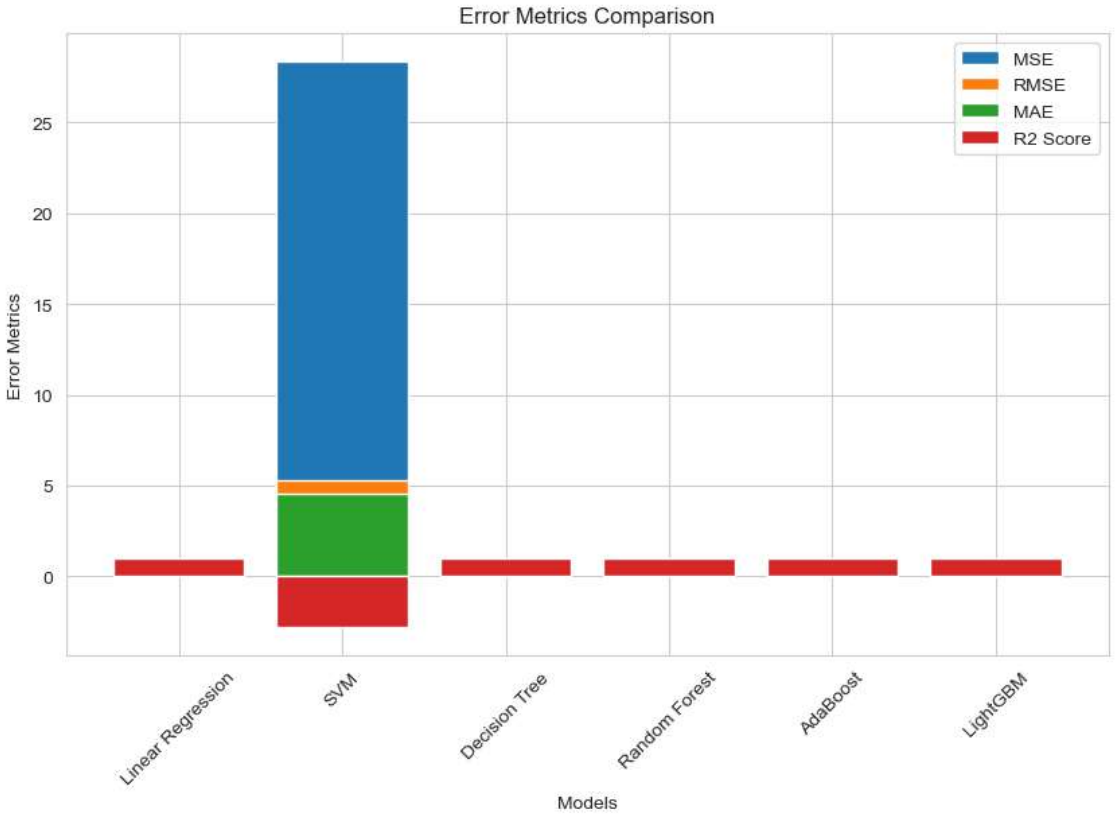
In [ ]:

```
1
```

TIME BASED TRAIN TEST SPLIT ACCURACY CHECKING FOR DIFFERENT MODELS
(Taking all errors together)

In [125]:

```
import matplotlib.pyplot as plt

# Define the models and their corresponding error metrics
models = ['Linear Regression','SVM', 'Decision Tree', 'Random Forest',

mse = [mse_time_lr, mse_time_svm, mse_time_dt, mse_time_rfc, mse_time_
rmse = [rmse_time_lr, rmse_time_svm, rmse_time_dt, rmse_time_rfc, rmse
mae = [mae_time_lr, mae_time_svm, mae_time_dt, mae_time_rfc, mae_time_
r2 = [r2_time_lr, r2_time_svm, r2_time_dt, r2_time_rfc, r2_time_ab, r2

# Plot the error metrics for each model
plt.figure(figsize=(10, 6))
plt.bar(models, mse, label='MSE')
plt.bar(models, rmse, label='RMSE')
plt.bar(models, mae, label='MAE')
plt.bar(models, r2, label='R2 Score')
plt.xlabel('Models')
plt.ylabel('Error Metrics')
plt.title('Error Metrics Comparison')
plt.legend()
plt.xticks(rotation=45)
plt.show()

```



In [ ]:

```
1
```

5. Select the best possible model

FROM THE TRAINING AND TESTING R2 SCORE CHECKING, AND BARPLOT VISUALIZATION OF DIFFERENT MODELS, WE GOT THESE FOLLOWING MAIN COMPARISIONAL DATA - NOrmal Train Test Split:

1. In case of Normal train test split, LinearRegression, SVM are performing totally bad.
2. In the case of SVM, where the accuracy is negative, it suggests that the model is not able to capture the patterns in the data and is performing poorly.
3. Decision Tree, and Random Forest are showing overfitting as huge difference between training and testing also the training data accuracy is very high over testing. So model is unable to collect patterns and to predict output based on unseen data.
4. AdaBoost can be considered performing slightly better than other models but very low accuracy level 15.78% So it can't be taken.
5. LightGBM is showing better performance rather than all the other 5 models, testing accuracy = 67.87%. So we can take it.

Time Based Train Test Split:

1. In the case of SVM, where the accuracy is negative, it suggests that the model is not able to capture the patterns in the data. So it can't be considered.
2. All models excluding SVM are performing well as their training and testing accuracy is more closer as well as the accuracy score is also high above 95%.
3. Also I checked for overfitting, no overfitting is there in case of time based analysis.
4. Based on the accuracy score, Random Forest, Decision Tree, LightGBM are performing more betterto capture computational patterns and can predict better outcome in future for our project.
5. By the way, all models including Linear Regression and AdaBoost also performing well.

Type *Markdown* and LaTeX: α^2

So based on the two kinds of train test split value testing accuracy , I will use Random Forest for hyperparameter tuning.

6. Perform Hyperparameter tuning, select best hyperparameters by using appropriate algorithms

In [127]:

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt'],
    'random_state': [42]
}

# Create the Random Forest Regressor model
model_hyper_rfc = RandomForestRegressor(n_jobs = -1)

# Create the RandomizedSearchCV object
random_search = RandomizedSearchCV(
    estimator = model_hyper_rfc,
    param_distributions = param_grid,
    scoring='r2',
    n_iter= 3,
    cv= 3,
    verbose=1,
    random_state=42
)

# Perform the random search for hyperparameter tuning
random_search.fit(x_train_time, y_train_time)

# Get the best hyperparameters and the corresponding model
best_params = random_search.best_params_
best_model = random_search.best_estimator_

# Train the best model on the full training set
best_model.fit(x_train_time, y_train_time)

# Make predictions on the test set using the best model
y_pred_time = best_model.predict(x_test_time)

# Calculate the evaluation metrics for the best model
mse_hyper_rfc = mean_squared_error(y_test_time, y_pred_time)
rmse_hyper_rfc = np.sqrt(mse_hyper_rfc)
mae_hyper_rfc = mean_absolute_error(y_test_time, y_pred_time)
r2_hyper_rfc = r2_score(y_test_time, y_pred_time)

# Print the evaluation metrics
print("Best Parameters:", best_params)
print("MSE:", mse_hyper_rfc)
print("RMSE:", rmse_hyper_rfc)
print("MAE:", mae_hyper_rfc)
print("R2 score:", r2_hyper_rfc)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
```

```
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:41
3: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and w
ill be removed in 1.3. To keep the past behaviour, explicitly set `max_f
eatures=1.0` or remove this parameter as it is also the default value fo
r RandomForestRegressors and ExtraTreesRegressors.
  warn(
```

```
Best Parameters: {'random_state': 42, 'n_estimators': 100, 'min_samples_
split': 10, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth':
None}
MSE: 0.0010291930614371655
RMSE: 0.032081038970662494
MAE: 0.0017612331121665836
R2 score: 0.9998613299134784
```

7. Come up with the best possible model accuracy.

The hyperparameter tuning is taking very huge time. Best Model Accuracy is coming 99.99%. It can be predicted.

THANK YOU!!!