Estimating Air Quality Index(AQI) of Indian Cities using Machine Learniing algorithms to leverage Pollution Prediction

Air pollution refers to the presence of harmful substances in the atmoshpere of this Earth which can be affect the human health and overall environenments. These pollutants can be in any form especcially gases and mixutre of solid or liquid substances in air or atmosphere which can cause potential impact on human health.

Air Quality Index (AQI) is a numerical scale used to measure and which is crucial indicator used to assess the air pollution levels in a specific area and it impacts on humanity and the surrounding environment. As environmental enthusiast and a budding Data Scientist, I delved into this project aimed to develop an accurate and most reliable AQI estimation model using Machine Learning Algorithms.

In this project, I used my Skills involved extensive data analysis and model developing, with a focus on understanding the influential factors affecting the air quality as well as environment. I collected a comprehensive dataset comprising various environmental parameters ad meterological factors, such as particulate matter(PM2.5, PM10), harmfull gases emittied by vechicles and industries such as NO2, NOx, SO2, CO, O3, Benzene, Toluene, Xylene etc which potentialy affects the air quality and cause several health issues. These parameters along with meteorological variables, provided valuable insights into the dynamics of the air pollution.

Import the Necessary Libraries

In [108]:
```
#Libraries required to Load, manipulate and visualize the data and to perform the Expl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Import the Data and Check the Features present in the Data

In [2]:
```
# 'import the CSV data file using pandas library'
# I assigned a variable 'data_set' for the CSV file
data_set= pd.read_csv('city_day.csv') # read the 'city_day.csv' file using pandas libr
data_set.head(10) # present the first 10 rows of the all the columns in 'data_set'.
```

Out[2]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene
0	Ahmedabad	2015-01-01	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.00	0.02
1	Ahmedabad	2015-01-02	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.68	5.50
2	Ahmedabad	2015-01-03	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.80	16.40
3	Ahmedabad	2015-01-04	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.43	10.14
4	Ahmedabad	2015-01-05	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.01	18.89
5	Ahmedabad	2015-01-06	NaN	NaN	45.41	38.48	81.50	NaN	45.41	45.76	46.51	5.42	10.83
6	Ahmedabad	2015-01-07	NaN	NaN	112.16	40.62	130.77	NaN	112.16	32.28	33.47	0.00	0.00
7	Ahmedabad	2015-01-08	NaN	NaN	80.87	36.74	96.75	NaN	80.87	38.54	31.89	0.00	0.00
8	Ahmedabad	2015-01-09	NaN	NaN	29.16	31.00	48.00	NaN	29.16	58.68	25.75	0.00	0.00
9	Ahmedabad	2015-01-10	NaN	NaN	NaN	7.04	0.00	NaN	NaN	8.29	4.55	0.00	0.00

These are the first 10 rows of the 'data_set' and it consists of particulate matters values and harmful gases values of several cities of india.

But it seems like our data set consist of lot of missing values

First see the list of cities present in our data set and the Lets check the counts of missing vaues in our data set

EDA(Exploratoy Data Anaysis)

Doing this to get insights from the data and vizualize the key insights

See the list of cities present in the data_set

In [3]:
```
cities= data_set["City"].unique() # by using unique() to check the unique values prese
print(cities) # call the variable 'cities' to see the list of cities
print(len(cities)) # count the numer of cities present in 'data_set'
```

```
['Ahmedabad' 'Aizawl' 'Amaravati' 'Amritsar' 'Bengaluru' 'Bhopal'
 'Brajrajnagar' 'Chandigarh' 'Chennai' 'Coimbatore' 'Delhi' 'Ernakulam'
 'Gurugram' 'Guwahati' 'Hyderabad' 'Jaipur' 'Jorapokhar' 'Kochi' 'Kolkata'
 'Lucknow' 'Mumbai' 'Patna' 'Shillong' 'Talcher' 'Thiruvananthapuram'
 'Visakhapatnam']
26
```

There are 26 cities totally present in the data_set.

Next we'll check the count of data present in each the cities

In [107]:
```
data_set['City'].value_counts()
```

Out[107]:
```
0     2009
10    2009
20    2009
4     2009
19    2009
8     2009
14    2006
21    1858
12    1679
25    1462
3     1221
16    1169
15    1114
24    1112
2      951
6      938
23     925
18     814
13     502
9      386
22     310
7      304
5      289
11     162
17     162
1      113
Name: City, dtype: int64
```

The 'City' column in this data_set holds significant importance as it represent the geographical location for which the AQI(Air Quality Index) is estimated. As we all know that the there are huge variations in the air quality across different cities due to several facotrs suchh as population, weather condition, public activities, transportation facilities, Industrial facilities,etc. Allowing for a localized understanding of the pollution levels in different cities

ohhhh! This cities have highly imbalanced data as well and with lot of missing values too.

First of all, Let's see the what should we do with missing values first.

Check the missing values and its percentage

In [5]:
```
#Lets make a dataframe of  each columns and  see the
null_values=pd.DataFrame({
    #'Columns':data_set.columns,
    'Null counts': data_set.isnull().sum(), # Gives the counts of missing values in ea
    'Null Percentage' : (data_set.isnull().sum()/ len(data_set) *100).map('{:.2f}%'.fo
})

print(null_values)
```

```
            Null counts Null Percentage
City                  0           0.00%
Date                  0           0.00%
PM2.5              4598          15.57%
PM10              11140          37.72%
NO                 3582          12.13%
NO2                3585          12.14%
NOx                4185          14.17%
NH3               10328          34.97%
CO                 2059           6.97%
SO2                3854          13.05%
O3                 4022          13.62%
Benzene            5623          19.04%
Toluene            8041          27.23%
Xylene            18109          61.32%
AQI                4681          15.85%
AQI_Bucket         4681          15.85%
```

By the look of it our data set consist lot of missing data points

Let see what we can do with the missing values

Choices we have

1. To fill the missing values with mean or median.
2. To drop the missing values.

I choose to fill the missing values with the mean (It is the most simple, straightforward and effecient method)

But, due to geographical and meteorological factors we can't normally fill the missing values with over all mean.

because of this reason, I choose to fill with the averaged mean for each city. So, it will maintain the order of missing values and minize the disortion of data

Group the data by city and check the mean for each city

In [6]:
```
avg_by_city = data_set.groupby('City').transform('mean') # Reference from stack overfl
#group the data by city and calculate the mean of each city seperately by using transf
avg_by_city.describe()
```

```
C:\Users\RASCAL\AppData\Local\Temp\ipykernel_4528\1480119980.py:1: FutureWarning: Th
e default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
version, numeric_only will default to False. Either specify numeric_only or select o
nly columns which should be valid for the function.
  avg_by_city = data_set.groupby('City').transform('mean') # Reference from stack ov
erflow
```

Out[6]:

	PM2.5	PM10	NO	NO2	NOx	NH3	CO
count	29531.000000	27522.000000	29531.000000	29531.000000	28362.000000	27522.000000	29531.000000
mean	66.234841	116.228087	18.226745	28.641768	32.855058	23.024137	2.602122
std	31.709480	47.911677	10.974596	13.766171	15.903958	14.939010	5.321265
min	17.130000	23.352857	0.917092	0.388496	1.002202	2.813625	0.239675
25%	41.130242	83.243287	9.409136	16.857623	19.717092	11.988493	0.663049
50%	55.468335	114.584029	15.233330	27.995042	29.377311	18.371518	1.067349
75%	109.714059	126.747958	26.846916	37.193188	45.952471	29.322199	1.838634
max	123.498562	232.809229	71.771625	59.025496	68.412901	63.452612	22.193407

Now we'll check the overall mean of the data set

In [7]:
```
data_set.describe()
```

Out[7]:

	PM2.5	PM10	NO	NO2	NOx	NH3	CO
count	24933.000000	18391.000000	25949.000000	25946.000000	25346.000000	19203.000000	27472.000000
mean	67.450578	118.127103	17.574730	28.560659	32.309123	23.483476	2.248598
std	64.661449	90.605110	22.785846	24.474746	31.646011	25.684275	6.962884
min	0.040000	0.010000	0.020000	0.010000	0.000000	0.010000	0.000000
25%	28.820000	56.255000	5.630000	11.750000	12.820000	8.580000	0.510000
50%	48.570000	95.680000	9.890000	21.690000	23.520000	15.850000	0.890000
75%	80.590000	149.745000	19.950000	37.620000	40.127500	30.020000	1.450000
max	949.990000	1000.000000	390.680000	362.210000	467.630000	352.890000	175.810000

There is definetly a difference between the overall mean and mean of each city.

Now fill the missing values with the mean of each cities.

Filling the missing values of each columns by the mean of each city

In [8]:
```
data_set['PM2.5'].fillna(avg_by_city['PM2.5'].mean().round(2), inplace=True)
data_set['PM10'].fillna(avg_by_city['PM10'].mean().round(2), inplace=True)
data_set['NO'].fillna(avg_by_city['NO'].mean().round(2), inplace=True)
data_set['NO2'].fillna(avg_by_city['NO2'].mean().round(2), inplace=True)
data_set['NOx'].fillna(avg_by_city['NOx'].mean().round(2), inplace=True)
data_set['NH3'].fillna(avg_by_city['NH3'].mean().round(2), inplace=True)
data_set['CO'].fillna(avg_by_city['CO'].mean().round(2), inplace=True)
data_set['SO2'].fillna(avg_by_city['SO2'].mean().round(2), inplace=True)
data_set['O3'].fillna(avg_by_city['O3'].mean().round(2), inplace=True)
data_set['Benzene'].fillna(avg_by_city['Benzene'].mean().round(2), inplace=True)
data_set['Toluene'].fillna(avg_by_city['Toluene'].mean().round(2), inplace=True)
data_set['Xylene'].fillna(avg_by_city['Xylene'].mean().round(2), inplace=True)
data_set['AQI'].fillna(avg_by_city['AQI'].mean().round(2), inplace=True)

data_set.describe()
```

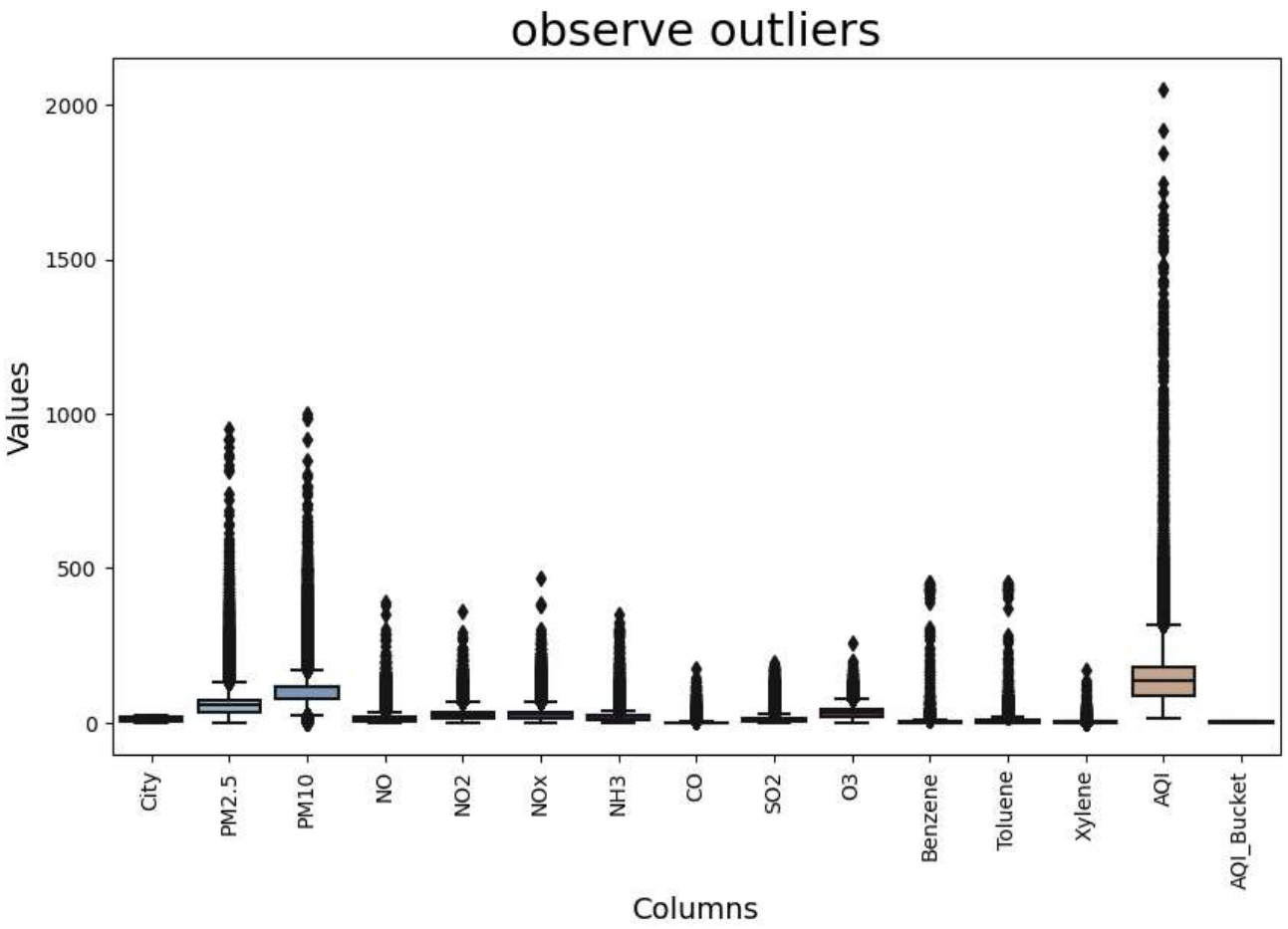
Out[8]:

	PM2.5	PM10	NO	NO2	NOx	NH3	CO
count	29531.000000	29531.000000	29531.000000	29531.000000	29531.000000	29531.000000	29531.000000
mean	67.260533	117.411457	17.654212	28.570291	32.387191	23.321383	2.273099
std	59.416124	71.506866	21.360291	22.941065	29.318566	20.712549	6.716349
min	0.040000	0.010000	0.020000	0.010000	0.000000	0.010000	0.000000
25%	32.150000	79.315000	6.210000	12.980000	14.670000	12.040000	0.540000
50%	58.030000	116.230000	11.530000	25.240000	27.550000	23.020000	0.950000
75%	72.450000	116.230000	18.230000	34.665000	36.015000	23.020000	1.710000
max	949.990000	1000.000000	390.680000	362.210000	467.630000	352.890000	175.810000

Check For outliers

We will now see the Distribution of data of each columns using box plot

In [123]:
```
plt.figure(figsize=(10,6))
sns.boxplot(data_set, palette='twilight')
plt.xlabel('Columns', fontsize=14)
plt.ylabel('Values', fontsize=14)
plt.title('observe outliers', fontsize=22)
plt.xticks(rotation=90)
plt.show()
```

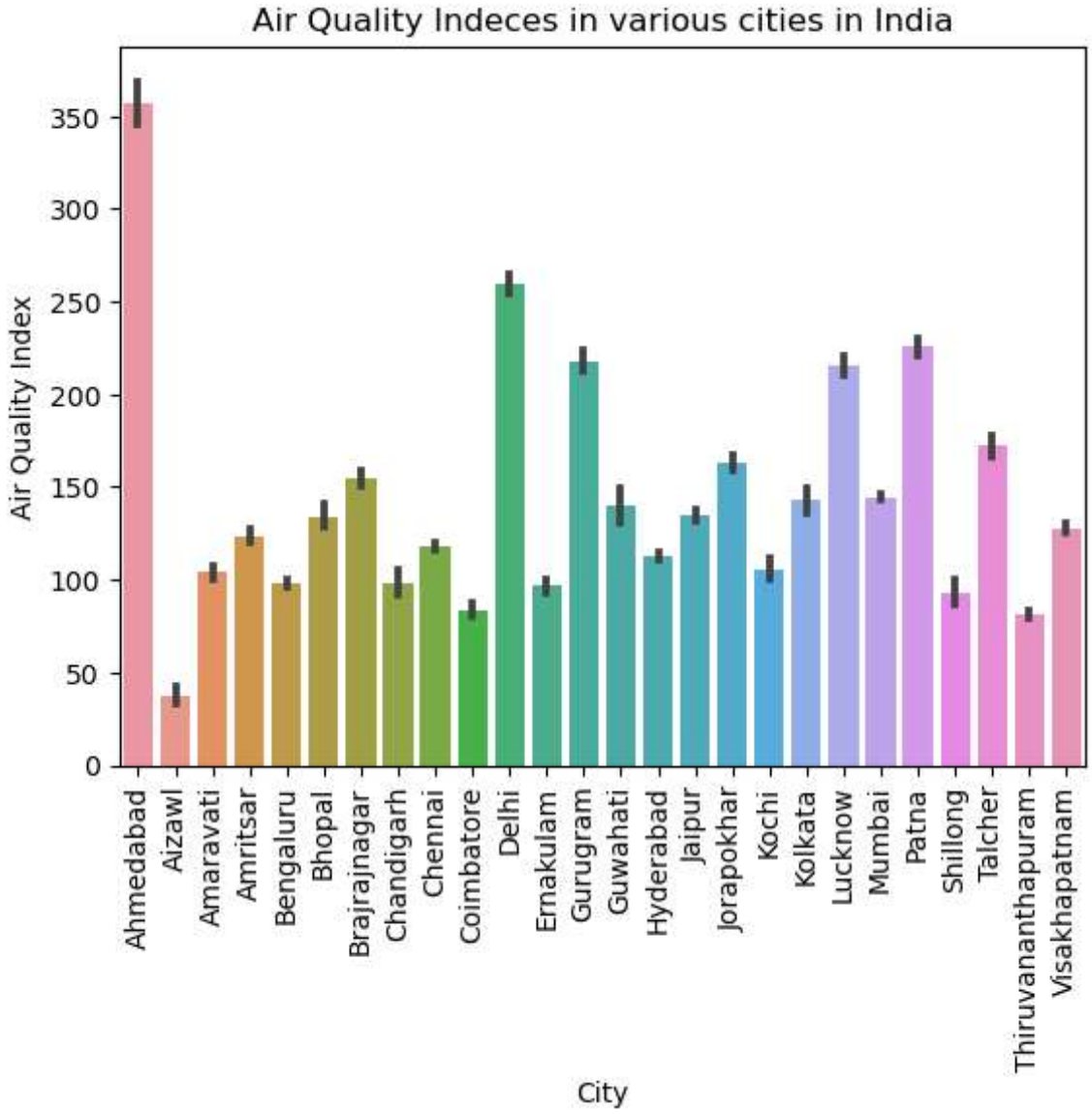


There are lot of outliers in each and every columns, but outliers in Air Quality data may also represents the Extreme pollution events(Celebrations with high amounts crackers such as diwali festival, marriage occasions, public celebrations of success and birth day of iconic stars, political leaders and sportsmen, etc.) or also be represents unusual environmental condiotions such as unpredictable Climatic changes. So for this project outliers also considered for the contirbuting for the model development

Plot the AQI of each cities

Let's visualize the distribution of AQI for each cities

In [9]:
```
sns.barplot(x='City', y= 'AQI', data=data_set)
plt.xlabel('City')
plt.ylabel('Air Quality Index')
plt.title('Air Quality Indeces in various cities in India')
plt.xticks(rotation=90)
plt.show()
```



Now Let's Encode the object data type columns such as Cty and AQI Bucket using label encoder

In [10]:
```
#import the label encoder from scikit learn library
from sklearn.preprocessing import LabelEncoder
lab=LabelEncoder()
data_set['City']=lab.fit_transform(data_set['City']) #fit and transform to encode the
data_set['AQI_Bucket']= lab.fit_transform(data_set['AQI_Bucket']) #fit and transform t

data_set['City'].unique() #check the encoded data of city column
```

Out[10]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25])
```

In [11]:
```
data_set['AQI_Bucket'].unique()
```

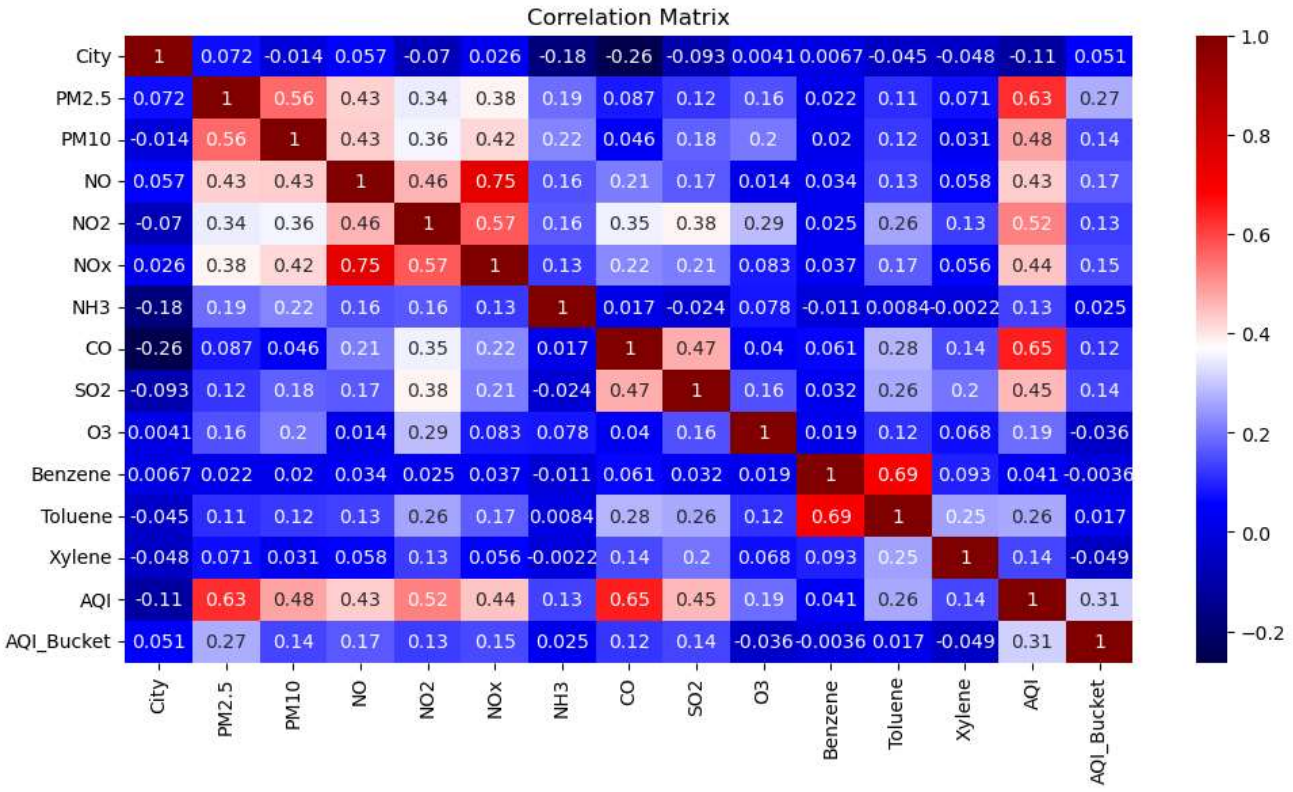
Out[11]:
```
array([6, 2, 5, 4, 1, 3, 0])
```

Correlation of the dataset

In this project AQI is the target variable and Let see what are all the features have relationship with our target by using correlation and visualize it with the help of seaborn heatmap

In [12]:
```
dt=data_set
correlation=dt.corr()
plt.figure(figsize=(12,6))
sns.heatmap(correlation, annot=True,cmap='seismic')
plt.title('Correlation Matrix')
plt.show()
```

```
C:\Users\RASCAL\AppData\Local\Temp\ipykernel_4528\4045941223.py:2: FutureWarning: Th
e default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of numer
ic_only to silence this warning.
  correlation=dt.corr()
```



By checking correlation, it seems that not all the column has high correlate with our target but even the low correlated features can play crucial role

Let's see the feature importance from training model

In [13]:
```
dt= dt.drop(columns='Date')
dt.shape
```

Out[13]:
```
(29531, 15)
```

Developing the Model

Split data for model building

Split the data into thte target (y) and features (X)

In [14]:
```
y=dt['AQI']
X=dt.drop(columns=['AQI'])
print(X.shape)
y.shape
```

```
(29531, 14)
```

Out[14]:
```
(29531,)
```

Experiment with various models

To create most accurate and robust model, I planned to experiment with several models and compare the performance and accracy of those models and choose the successful and most accurate model

Import the libraries and algoritms to create regression model

In [15]:
```
from sklearn.model_selection import GridSearchCV, train_test_split  # to select best p
from sklearn.preprocessing import StandardScaler #use this to scale dta
from sklearn.linear_model import LinearRegression, Ridge, Lasso # basic regression mod
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor #import
from xgboost import XGBRegressor #import xgboost regression algorithm
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from catboost import CatBoostRegressor #cat boost regression algorithm #got reference
```

Split the Training and Test data for the X and y(Features and Target)

In [16]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(23624, 14)
(5907, 14)
(23624,)
(5907,)
```

Scale the training and test features

Use Standard Scaler to rescale the training and test features of data but avoid scaling target variables.

In [17]:
```
scaler= StandardScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.fit_transform(X_test)

print(scaled_X_train.shape)
print(scaled_X_test.shape)
```

```
(23624, 14)
(5907, 14)
```

1. Linear Regression Model

Instantiate Linear Regression

In [18]:
```
linear= LinearRegression()
linear
```

Out[18]:
```
▾ LinearRegression

LinearRegression()
```

Fit the Linear Regression Model with Training Data

In [19]:
```
model_lr=linear.fit(scaled_X_train, y_train) #Linearreg doesn't have hyperparameters
```

Analyse Metrics for the Linear Regression Model

Check the mean absolute error, root mean squared error and r2_score or explained variance score of the developed model.

Validate the Prediction result with test data.

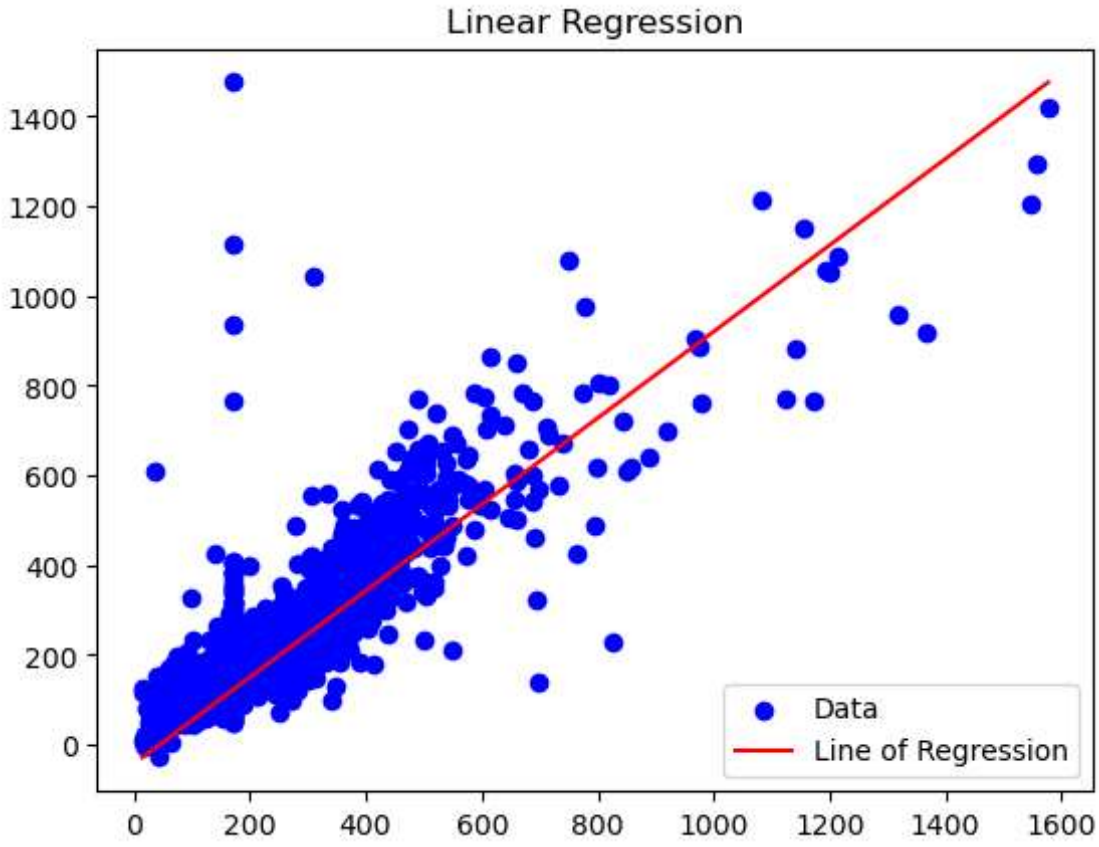
In [20]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score, explaine

y_pred_lr=model_lr.predict(scaled_X_test)

mae=mean_absolute_error(y_test,y_pred_lr)
rmse=np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_score= r2_score(y_test, y_pred_lr)
evs= explained_variance_score(y_test,y_pred_lr)

print('Mean Absolute Error of linear Regression:', mae)
print('Root Mean Squared Error of linear Regression:', rmse)
print('Coeffecient of Determination of linear Regression:', r2_score)
print('explained_variance_score of linear regression:', evs)
```

```
Mean Absolute Error of linear Regression: 30.5858223818957
Root Mean Squared Error of linear Regression: 53.86216954896833
Coeffecient of Determination of linear Regression: 0.8075738068339972
explained_variance_score of linear regression: 0.8075772454681636
```

Our linear regression model performs with nearly ~81% accuracy

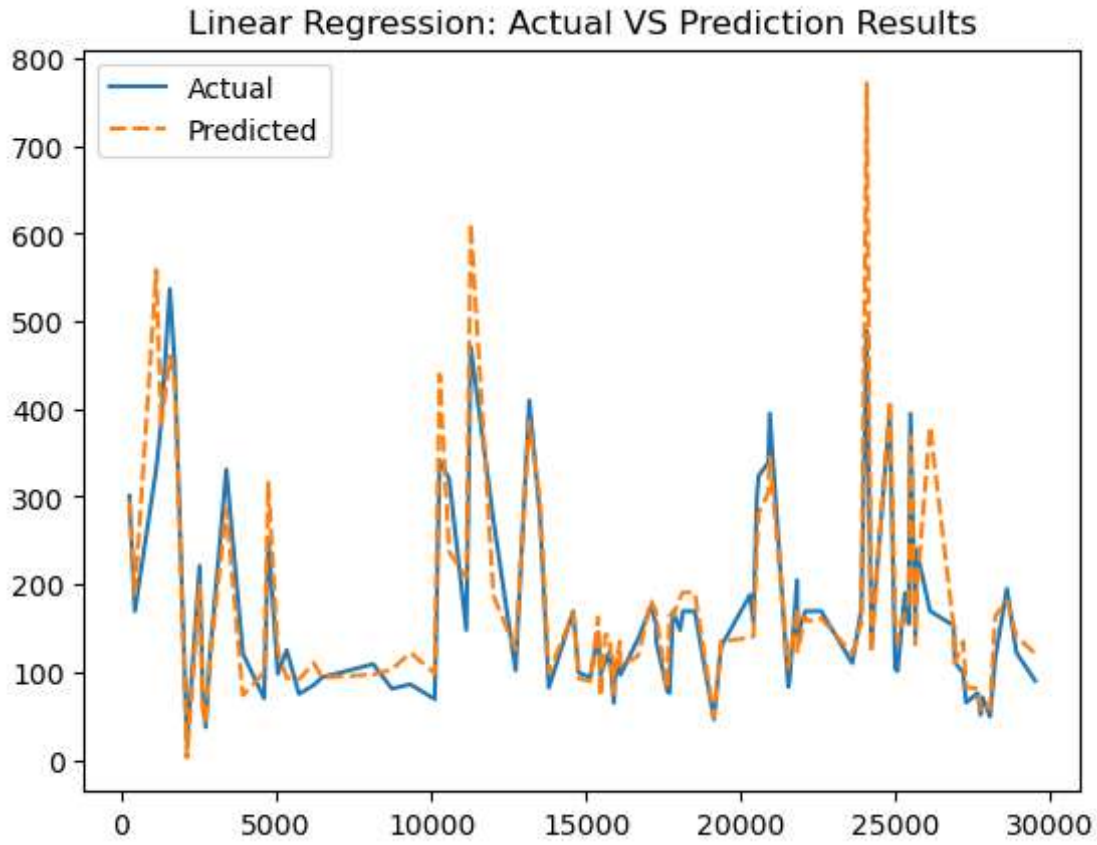
Plot Regression Line

In [21]:
```
plt.scatter(y_test, y_pred_lr, color='b', label='Data')
plt.plot([np.min(y_test),np.max(y_test)],[np.min(y_pred_lr),np.max(y_pred_lr)], color=
#plt.scatter(0, intercept, color='y', label='Y_Intercept')
plt.title('Linear Regression')
plt.legend()
plt.show()
```



Visualize the difference between the Actual test data and result of our prediction using Linear Regression

In [22]:
```
df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_lr})
sns.lineplot(df.head(100))
plt.title('Linear Regression: Actual VS Prediction Results')
plt.show()
```



By using the line plot we see how our predicted result varies from the actual test data

Ridge Regression for regularization

Instantiate Ridge regression

In [23]:
```
ridge=Ridge()
ridge
```

Out[23]:
```
Ridge
Ridge()
```

Assign a hyperparameters to the Ridge regression model to perform grid search and choose best parameters for ridge regression

In [24]:
```
ridge_param_grid ={'alpha':[0.1,1],
                   'solver':['auto']}
```

Perform Grid Search and fit the Ridge model to the best parameter

In [25]:
```
ridge_grid_search= GridSearchCV(ridge, ridge_param_grid, cv=5)
```

In [26]:
```
model_ridge=ridge_grid_search.fit(scaled_X_train, y_train)
```

Get the best parameters and best score for the model

In [27]:
```
ridge_best_params = ridge_grid_search.best_params_

ridge_best_score = ridge_grid_search.best_score_

print('Ridge best parameters:',ridge_best_params)
print('Ridge best score:',ridge_best_score)
```

```
Ridge best parameters: {'alpha': 1, 'solver': 'auto'}
Ridge best score: 0.7957439376369859
```

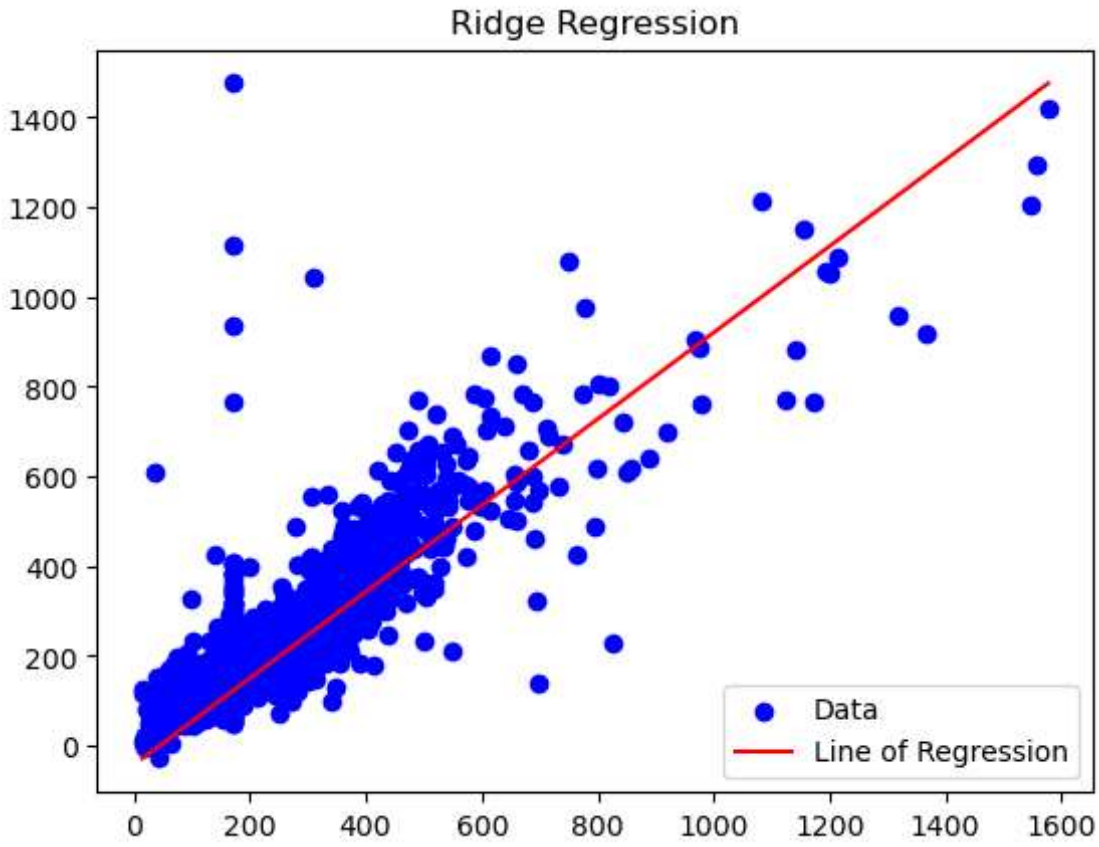
Analyze the metrics for Ridge Model

In [28]:
```
y_pred_ridge=model_ridge.predict(scaled_X_test)

mae_rid=mean_absolute_error(y_test,y_pred_ridge)
rmse_rid=(mean_squared_error(y_test, y_pred_ridge, squared=False))
evs_rid = explained_variance_score(y_test,y_pred_ridge)

print('Mean Absolute Error of Ridge Regression:', mae_rid)
print('Root Mean Squared Error of Ridge  Regression:', rmse_rid)
print('explained_variance_score of Ridge Regression:', evs_rid)
```

```
Mean Absolute Error of Ridge Regression: 30.586198345995438
Root Mean Squared Error of Ridge  Regression: 53.861768792362554
explained_variance_score of Ridge Regression: 0.8075801089169179
```

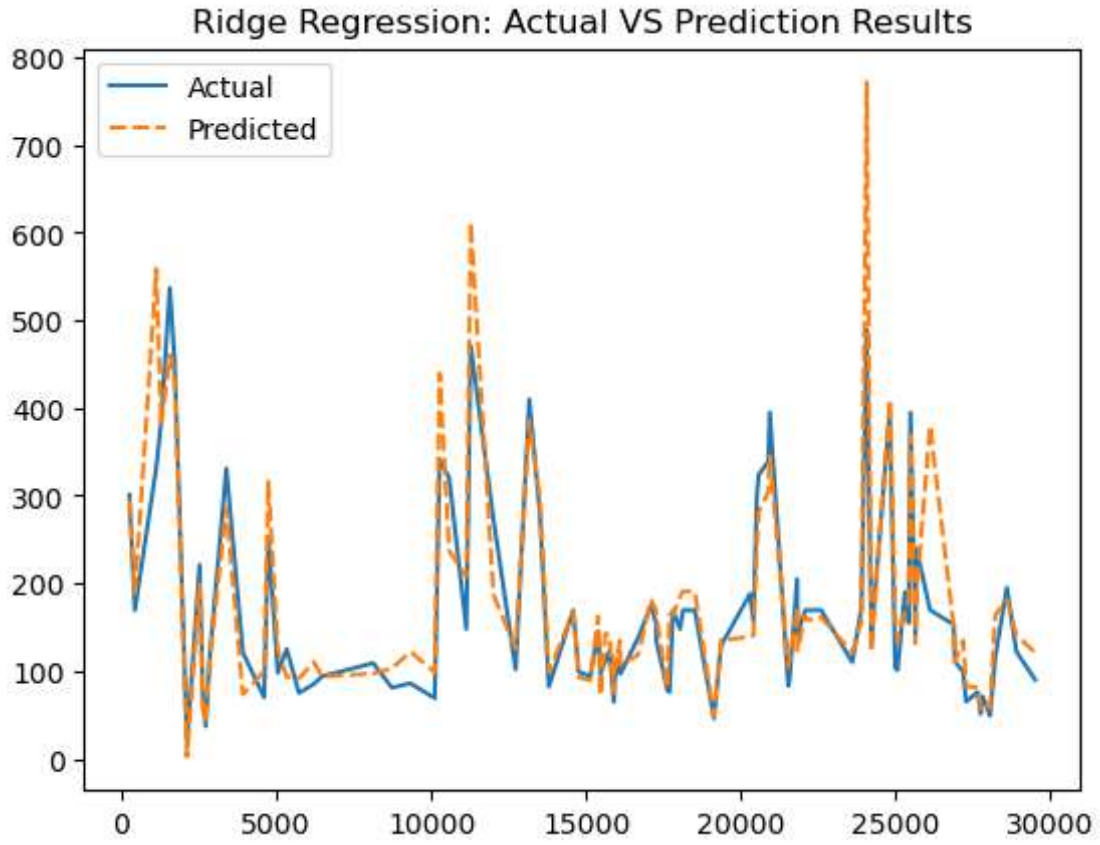
Plot th regression line of ridge model

In [29]:
```
plt.scatter(y_test, y_pred_ridge, color='b', label='Data')
plt.plot([np.min(y_test),np.max(y_test)],[np.min(y_pred_ridge),np.max(y_pred_ridge)],
#plt.scatter(0, intercept, color='y', label='Y_Intercept')
plt.title('Ridge Regression')
plt.legend()
plt.show()
```



Visualize the differencec between the Actual test data and the result of our prediction using Ridge Regression

In [30]:
```
df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_ridge})
sns.lineplot(df.head(100))
plt.title('Ridge Regression: Actual VS Prediction Results')
plt.show()
```



Lasso Regression

In [31]:
```
lasso=Lasso()
lasso
```

Out[31]:
```
Lasso()
```

In [32]:
```
lasso_param_grid ={'alpha':[0.1,1],
                   'selection':['cyclic','random']}
```

In [33]:
```
lasso_grid_search= GridSearchCV(lasso, lasso_param_grid, cv=5)
```

Fit Lasso Model

In [43]:
```
model_lasso=lasso.fit(scaled_X_train, y_train)
```

In [35]:
```
lasso_best_params = lasso_grid_search.best_params_

lasso_best_score = lasso_grid_search.best_score_


print('Lasso best parameters:', lasso_best_params)
print('Lasso best score:',lasso_best_score)
```

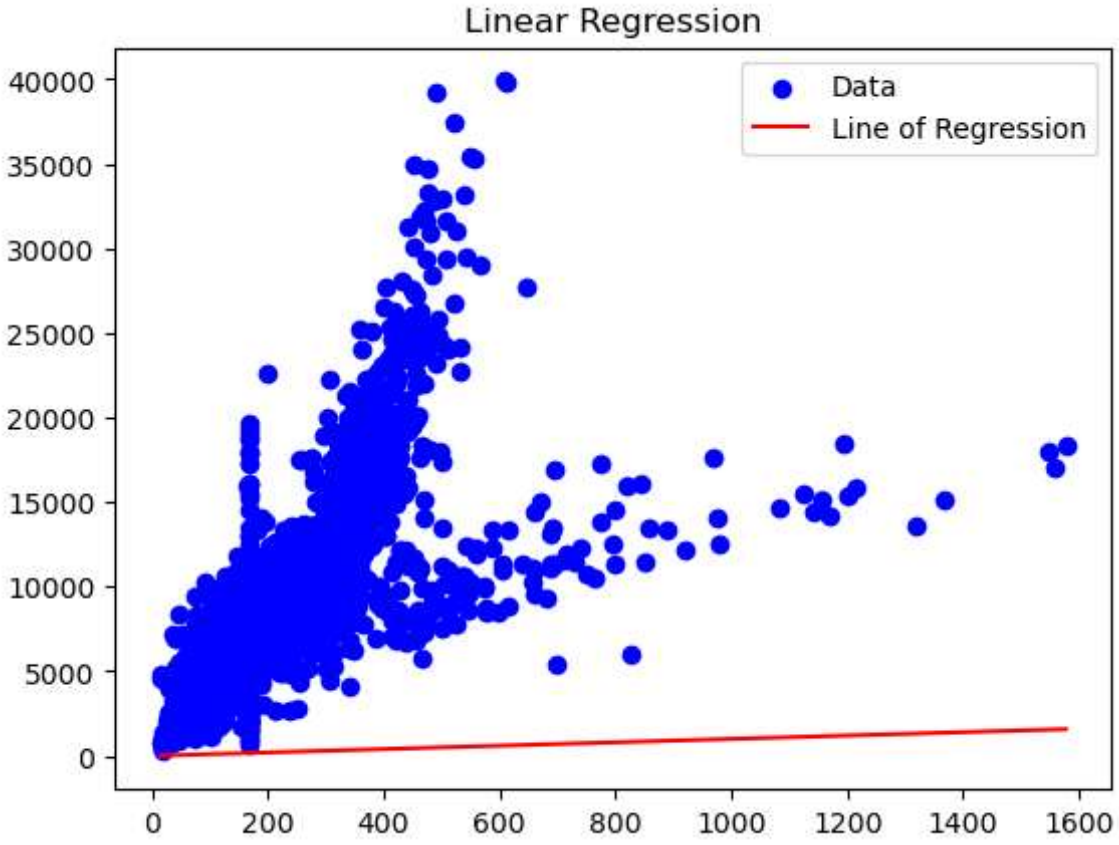
```
Lasso best parameters: {'alpha': 0.1, 'selection': 'cyclic'}
Lasso best score: 0.7957472132249057
```

Analyze the metrics for Lasso Model

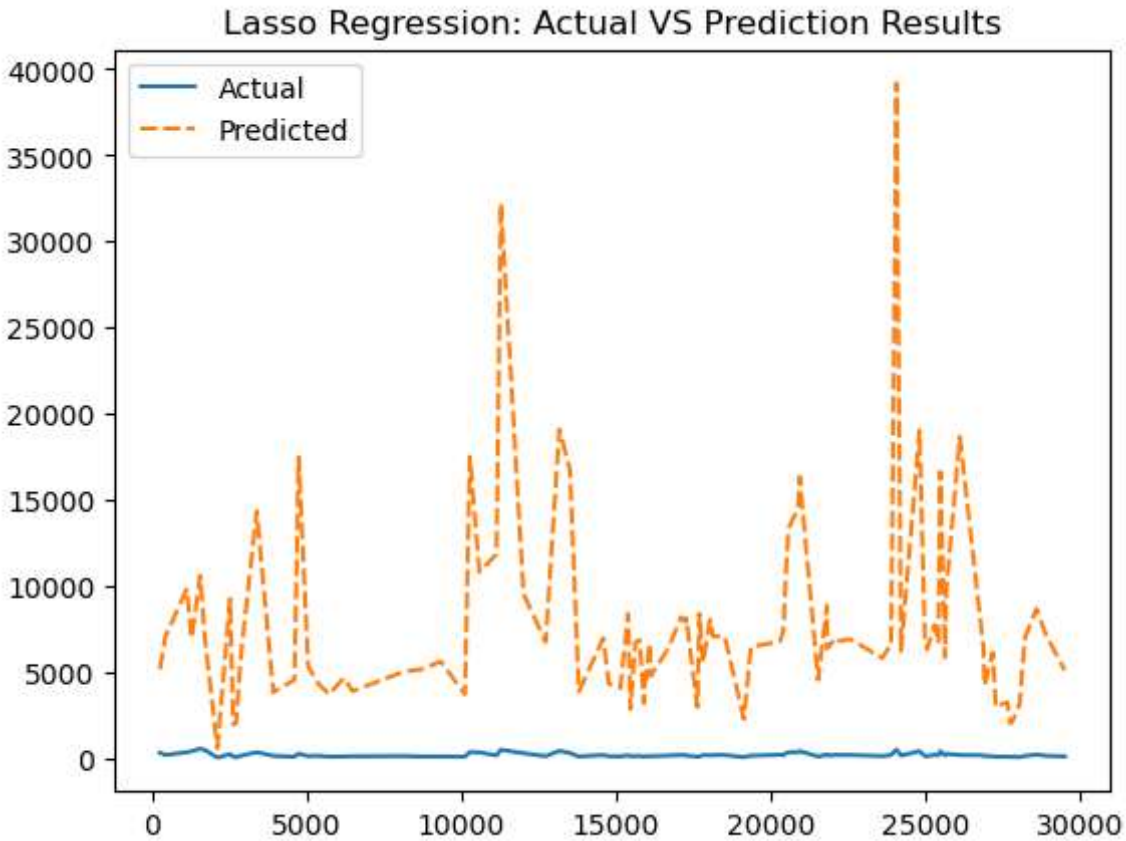
In [44]:
```
y_pred_las=model_lasso.predict(X_test)

mae_las=mean_absolute_error(y_test,y_pred_las)
rmse_las=np.sqrt(mean_squared_error(y_test, y_pred_las))
#r2s_las= r2_score(y_test, y_pred_las)
evs_las= explained_variance_score(y_test,y_pred_las)

print('Mean Absolute Error of lasso Regression:', mae_las)
print('Root Mean Squared Error of lasso Regression:', rmse_las)
print('Coeffecient of Determination of lasso Regression:', evs_las)
```

```
Mean Absolute Error of lasso Regression: 6813.250701811775
Root Mean Squared Error of lasso Regression: 8019.93637242945
Coeffecient of Determination of lasso Regression: -1186.2034018593697

C:\Users\RASCAL\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has
feature names, but Lasso was fitted without feature names
  warnings.warn(
```

In [37]:
```
plt.scatter(y_test, y_pred_las, color='b', label='Data')
plt.plot([np.min(y_test),np.max(y_test)],[np.min(y_test),np.max(y_test)], color='r', l
#plt.scatter(0, intercept, color='y', label='Y_Intercept')
plt.title('Linear Regression')
plt.legend()
plt.show()
```



In [38]:
```
df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_las})
sns.lineplot(df.head(100))
plt.title('Lasso Regression: Actual VS Prediction Results')
plt.show()
```



Our Lasso model didnt Perform Well with ur test data, consider it as an underfit model and ignore the model.

Random Forest Model

In [39]:
```
rf= RandomForestRegressor()
rf
```

Out[39]:
```
▸ RandomForestRegressor
```

In [40]:
```
param_grid_rf={'n_estimators': [100,200,300],
               'max_depth':[None, 5, 10],
               'min_samples_split':[2,5,10]}
```

In [45]:
```
rf_grid_search= GridSearchCV(rf, param_grid_rf, cv=5)
model_rf= rf_grid_search
```

In [46]:
```
model_rf.fit(scaled_X_train, y_train)
```

Out[46]:
```
▸ GridSearchCV
▸ estimator: RandomForestRegressor
    ▸ RandomForestRegressor
```

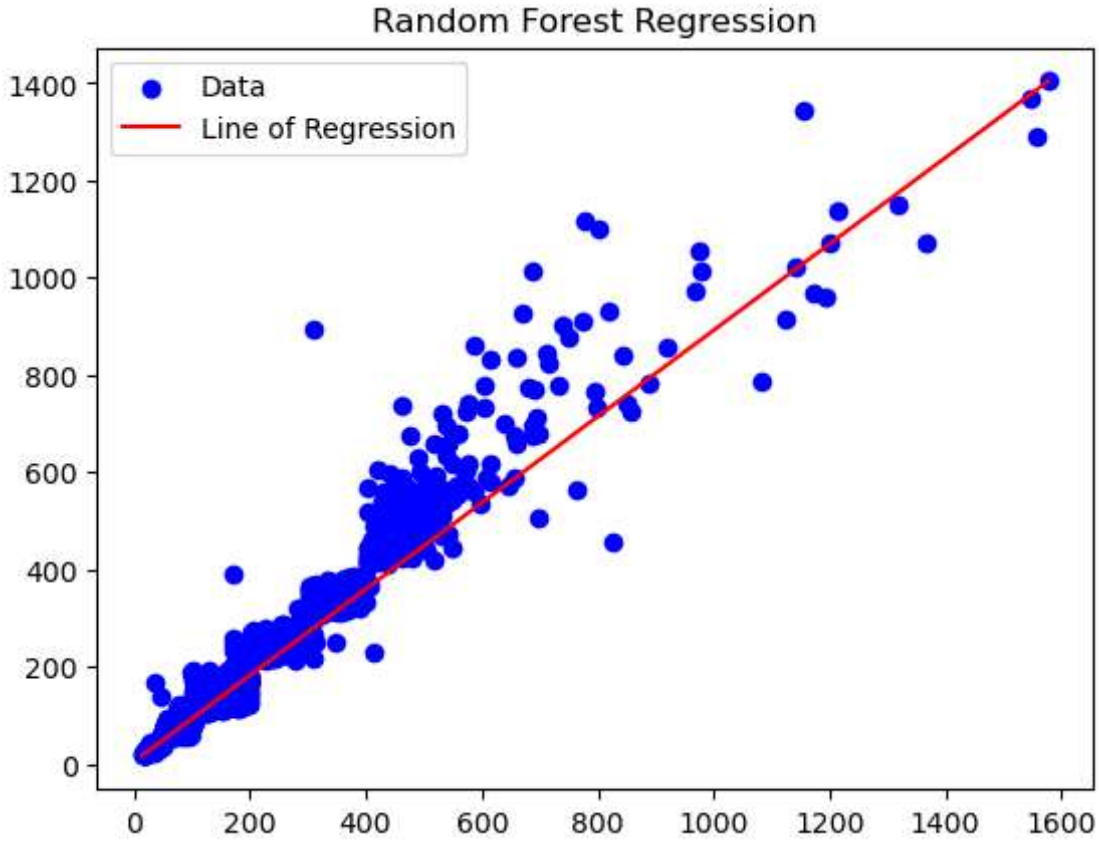
In [47]:
```
rf_best_params = rf_grid_search.best_params_

rf_best_score = rf_grid_search.best_score_

print('Random Forest best parameters:',rf_best_params)
print('Random Forest best score:',rf_best_score)
```

```
Random Forest best parameters: {'max_depth': None, 'min_samples_split': 10, 'n_estim
ators': 300}
Random Forest best score: 0.9303480555893335
```

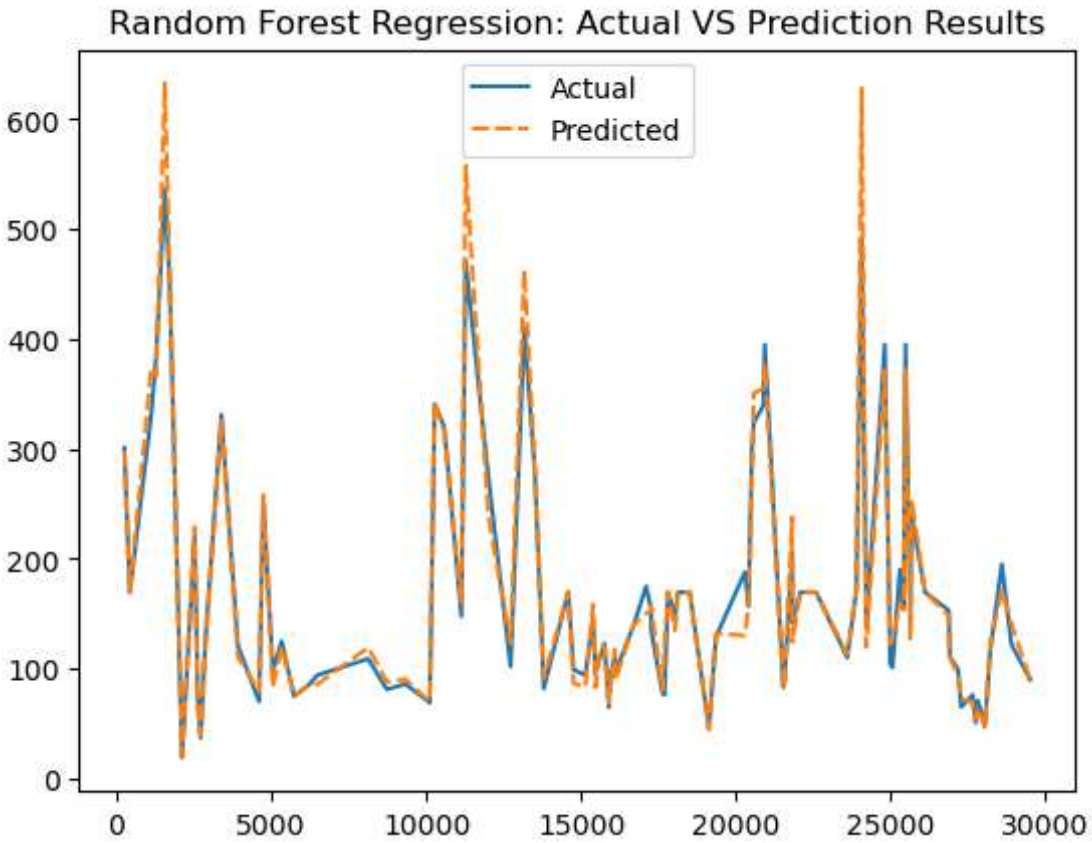
In [48]:
```
y_pred_rf=model_rf.predict(scaled_X_test)

mae_rf=mean_absolute_error(y_test,y_pred_rf)
rmse_rf=np.sqrt(mean_squared_error(y_test, y_pred_rf))
#r2s_rf=np.array([r2_score(y_test, y_pred_rf)])
evs_rf= explained_variance_score(y_test,y_pred_rf)

print('Mean Absolute Error of  Random Forest Regression:', mae_rf)
print('Root Mean Squared Error of  Random Forest Regression:', rmse_rf)
print('Coeffecient of Determination of Random Forest Regression:', evs_rf)
```

```
Mean Absolute Error of  Random Forest Regression: 12.959982878660279
Root Mean Squared Error of  Random Forest Regression: 26.5673063431299
Coeffecient of Determination of Random Forest Regression: 0.953324404351506
```

In [49]:
```
plt.scatter(y_test, y_pred_rf, color='b', label='Data')
plt.plot([np.min(y_test),np.max(y_test)],[np.min(y_pred_rf),np.max(y_pred_rf)], color=
#plt.scatter(0, intercept, color='y', label='Y_Intercept')
plt.title('Random Forest Regression')
plt.legend()
plt.show()
```



In [50]:
```
df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_rf})
sns.lineplot(df.head(100))
plt.title('Random Forest Regression: Actual VS Prediction Results')
plt.show()
```



Cat Boost Regression model

In [51]:
```
cat =  CatBoostRegressor()
cat
```

Out[51]:
```
<catboost.core.CatBoostRegressor at 0x1764e6f12d0>
```

In [52]:
```
cat_param_grid = {'iterations':[100,200,300],
                  'learning_rate':[0.1,0.01],
                  'depth':[3,5,7]}
```

In [53]:
```
cat_grid_search=GridSearchCV(cat, cat_param_grid, cv=5)
model_cat = cat_grid_search
```

In [54]:
```
model_cat.fit(scaled_X_train, y_train)
```

```
0:      learn: 122.6514869      total: 280ms    remaining: 27.7s
1:      learn: 114.4059527      total: 288ms    remaining: 14.1s
2:      learn: 107.2467465      total: 295ms    remaining: 9.53s
3:      learn: 101.0534905      total: 302ms    remaining: 7.25s
4:      learn: 95.1295759       total: 311ms    remaining: 5.9s
5:      learn: 90.2757016       total: 318ms    remaining: 4.98s
6:      learn: 86.0374077       total: 325ms    remaining: 4.32s
7:      learn: 82.2610542       total: 332ms    remaining: 3.81s
8:      learn: 78.6545342       total: 339ms    remaining: 3.43s
9:      learn: 75.6675616       total: 346ms    remaining: 3.12s
10:     learn: 72.8513470       total: 354ms    remaining: 2.86s
11:     learn: 70.4891609       total: 362ms    remaining: 2.65s
12:     learn: 68.1309865       total: 368ms    remaining: 2.46s
13:     learn: 65.0867275       total: 375ms    remaining: 2.3s
14:     learn: 63.1916134       total: 382ms    remaining: 2.16s
15:     learn: 61.5109830       total: 388ms    remaining: 2.04s
16:     learn: 60.0324547       total: 394ms    remaining: 1.92s
17:     learn: 57.8235676       total: 400ms    remaining: 1.82s
18:     learn: 55.9350501       total: 406ms    remaining: 1.73s
```

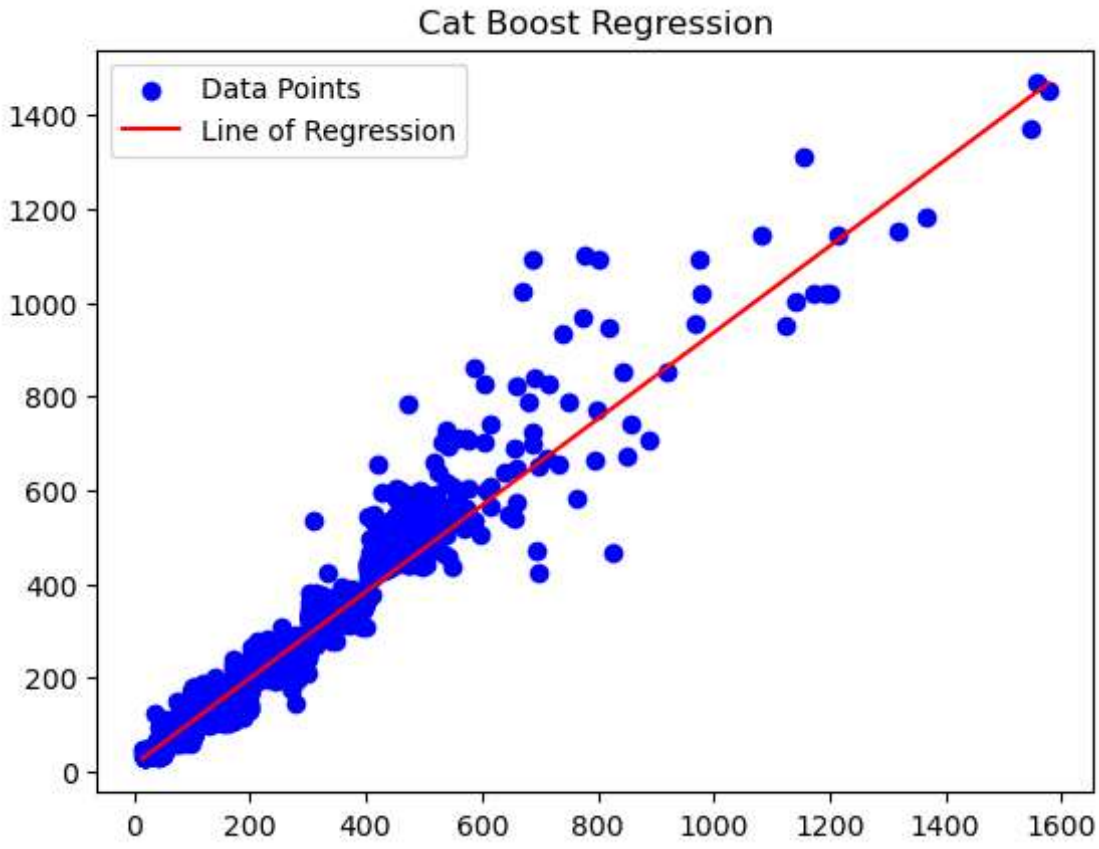
In [55]:
```
cat_best_params = cat_grid_search.best_params_

cat_best_score = cat_grid_search.best_score_

print('Cat Boost best parameters:',cat_best_params)
print('Cat Boost best score:',cat_best_score)
```

```
Cat Boost best parameters: {'depth': 7, 'iterations': 200, 'learning_rate': 0.1}
Cat Boost best score: 0.940528459199456
```

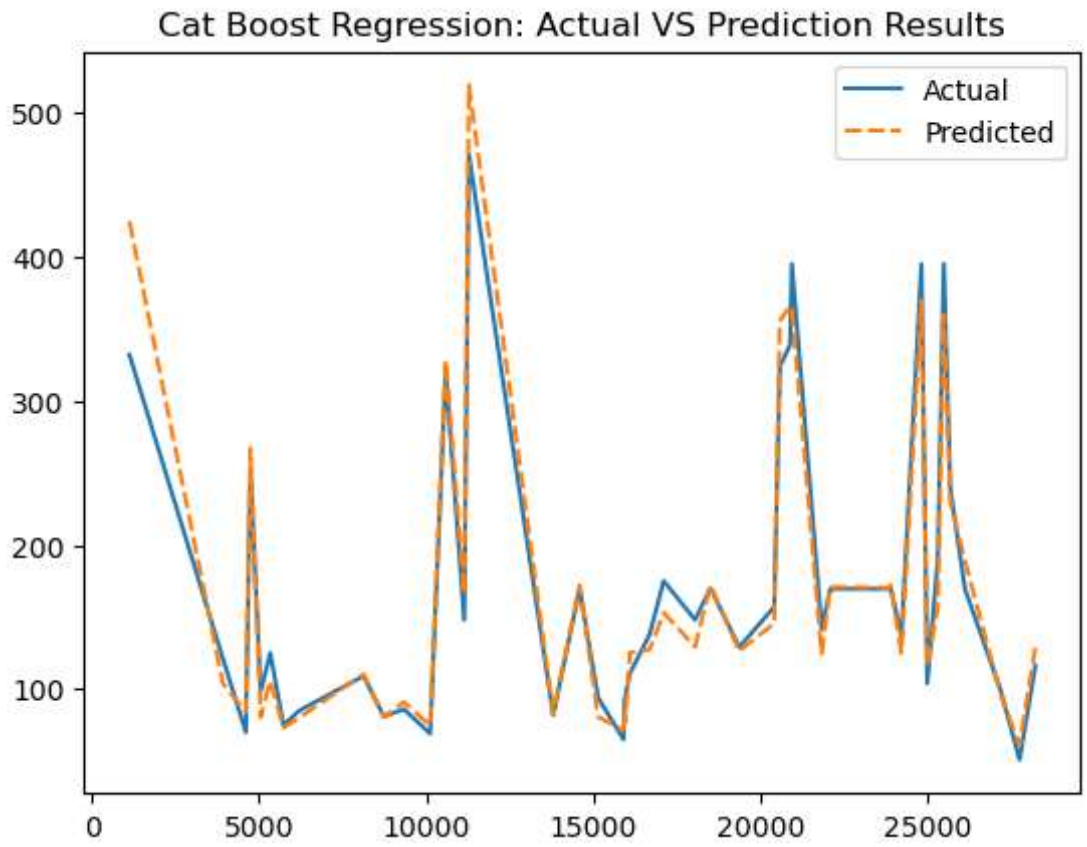
In [56]:
```
y_pred_cat=model_cat.predict(scaled_X_test)

mae_cat=mean_absolute_error(y_test,y_pred_cat)
rmse_cat=np.sqrt(mean_squared_error(y_test, y_pred_cat))

evs_cat= explained_variance_score(y_test,y_pred_cat)


print('Mean Absolute Error of Cat Boost Regression:', mae_cat)
print('Root Mean Squared Error of Cat Boost Regression:', rmse_cat)
print('Coeffecient of Determination of Cat Boost Regression:', evs_cat)
```

```
Mean Absolute Error of Cat Boost Regression: 14.23168918494042
Root Mean Squared Error of Cat Boost Regression: 26.262572273793715
Coeffecient of Determination of Cat Boost Regression: 0.9543602066796502
```

In [57]:
```
plt.scatter(y_test, y_pred_cat, color='b', label='Data Points')
plt.plot([np.min(y_test),np.max(y_test)],[np.min(y_pred_cat),np.max(y_pred_cat)], colo
#plt.scatter(0, intercept, color='y', label='Y_Intercept')
plt.title('Cat Boost Regression')
plt.legend()
plt.show()
```



In [58]:
```
df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_cat})
sns.lineplot(df.head(50))
plt.title('Cat Boost Regression: Actual VS Prediction Results')
plt.show()
```



XG Boost Regressor Model

In [59]:
```
xgb= XGBRegressor()
xgb
```

Out[59]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
```

In [60]:
```
xgb_param_grid = {'n_estimators':[100,200,300],
                  'learning_rate':[0.1,0.01],
                  'depth':[3,5,7]}
```

In [61]:
```
xgb_grid_search=GridSearchCV(xgb, xgb_param_grid, cv=5)
model_xgb= xgb_grid_search
```

In [62]:
```
model_xgb.fit(scaled_X_train, y_train)
```

```
oup-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "depth" } are not used.

[22:01:40] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gr
oup-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "depth" } are not used.

[22:01:44] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gr
oup-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "depth" } are not used.

[22:01:47] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gr
oup-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "depth" } are not used.

[22:01:55] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gr
oup-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "depth" } are not used.
```

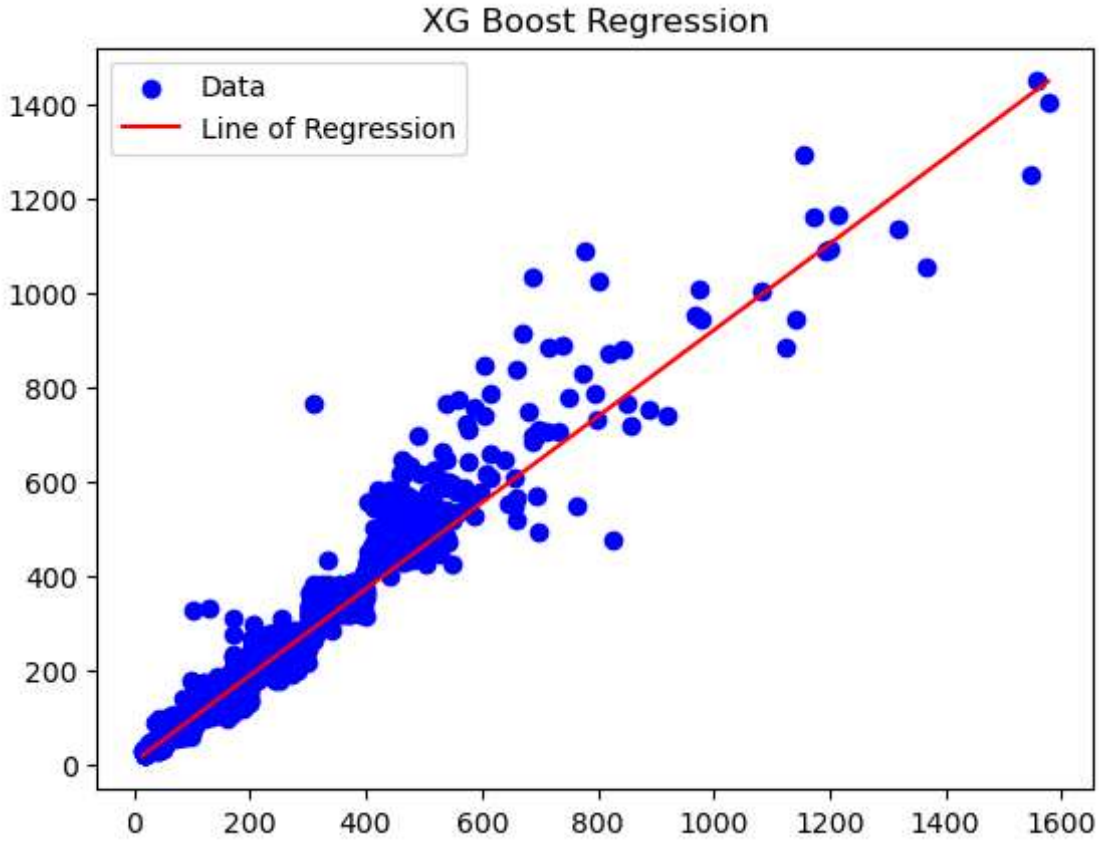
In [63]:
```
xgb_best_params = xgb_grid_search.best_params_

xgb_best_score = xgb_grid_search.best_score_

print('XGBoost best parameters:',xgb_best_params)
print('XGBoost best score:',xgb_best_score)
```

```
XGBoost best parameters: {'depth': 3, 'learning_rate': 0.1, 'n_estimators': 200}
XGBoost best score: 0.9261417833489588
```

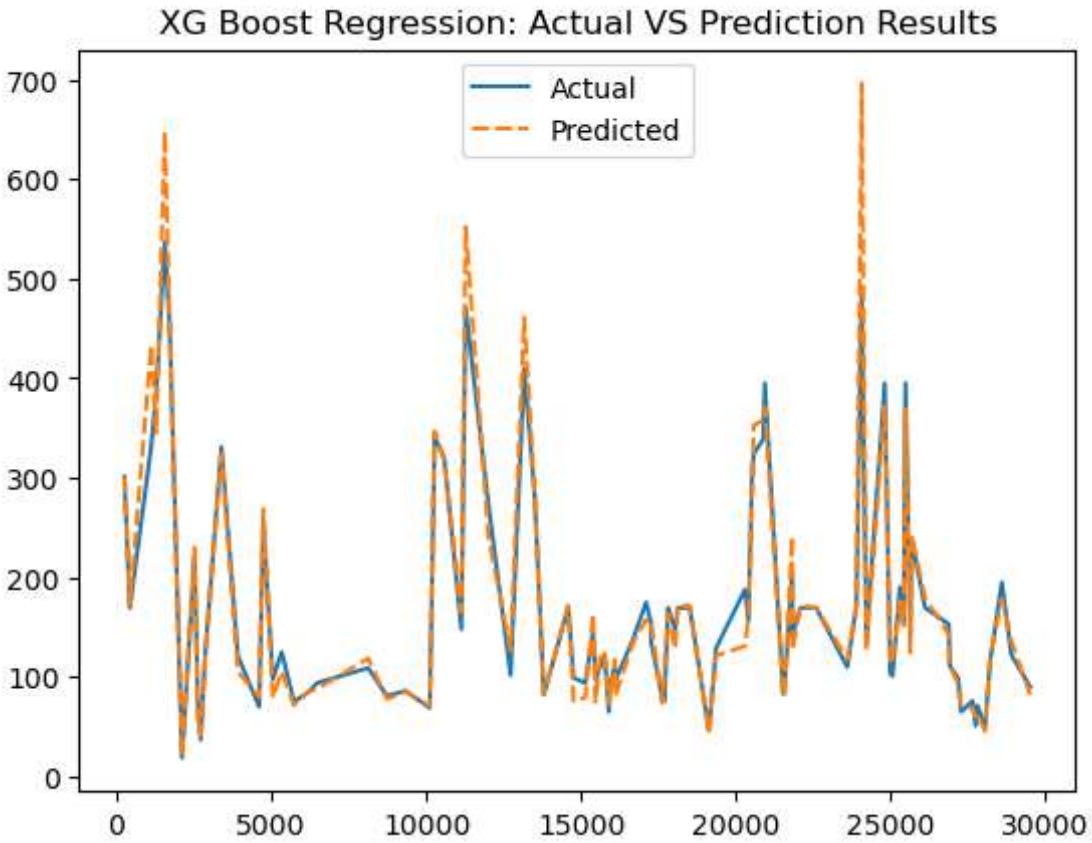
In [64]:
```
y_pred_xgb=model_xgb.predict(scaled_X_test)

mae_xgb=mean_absolute_error(y_test,y_pred_xgb)
rmse_xgb=np.sqrt(mean_squared_error(y_test, y_pred_xgb))

evs_xgb= explained_variance_score(y_test,y_pred_xgb)


print('Mean Absolute Error of Cat Boost Regression:', mae_xgb)
print('Root Mean Squared Error of Cat Boost Regression:', rmse_xgb)
print('Coeffecient of Determination of Cat Boost Regression:', evs_xgb)
```

```
Mean Absolute Error of Cat Boost Regression: 13.605039954907962
Root Mean Squared Error of Cat Boost Regression: 25.80108555148271
Coeffecient of Determination of Cat Boost Regression: 0.9558887330325706
```

In [65]:
```
plt.scatter(y_test, y_pred_xgb, color='b', label='Data')
plt.plot([np.min(y_test),np.max(y_test)],[np.min(y_pred_xgb),np.max(y_pred_xgb)], colo
#plt.scatter(0, intercept, color='y', label='Y_Intercept')
plt.title('XG Boost Regression')
plt.legend()
plt.show()
```



In [66]:
```
df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_xgb})
sns.lineplot(df.head(100))
plt.title('XG Boost Regression: Actual VS Prediction Results')
plt.show()
```

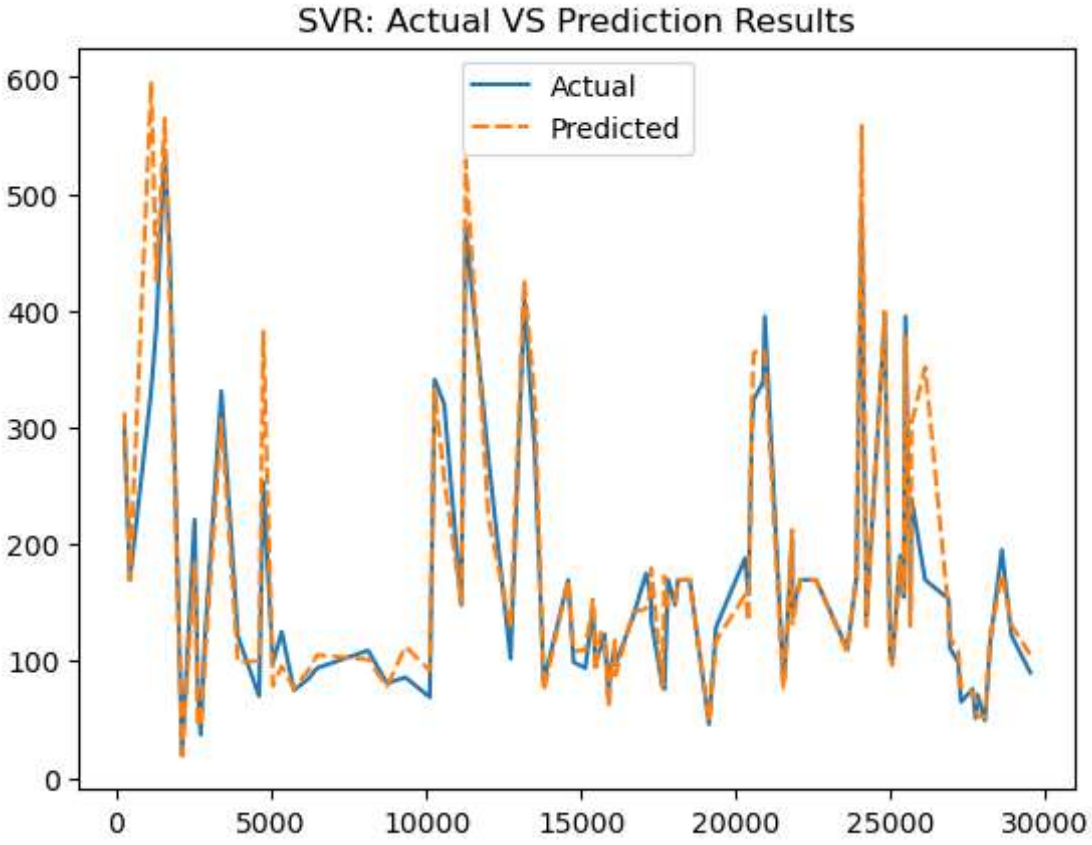


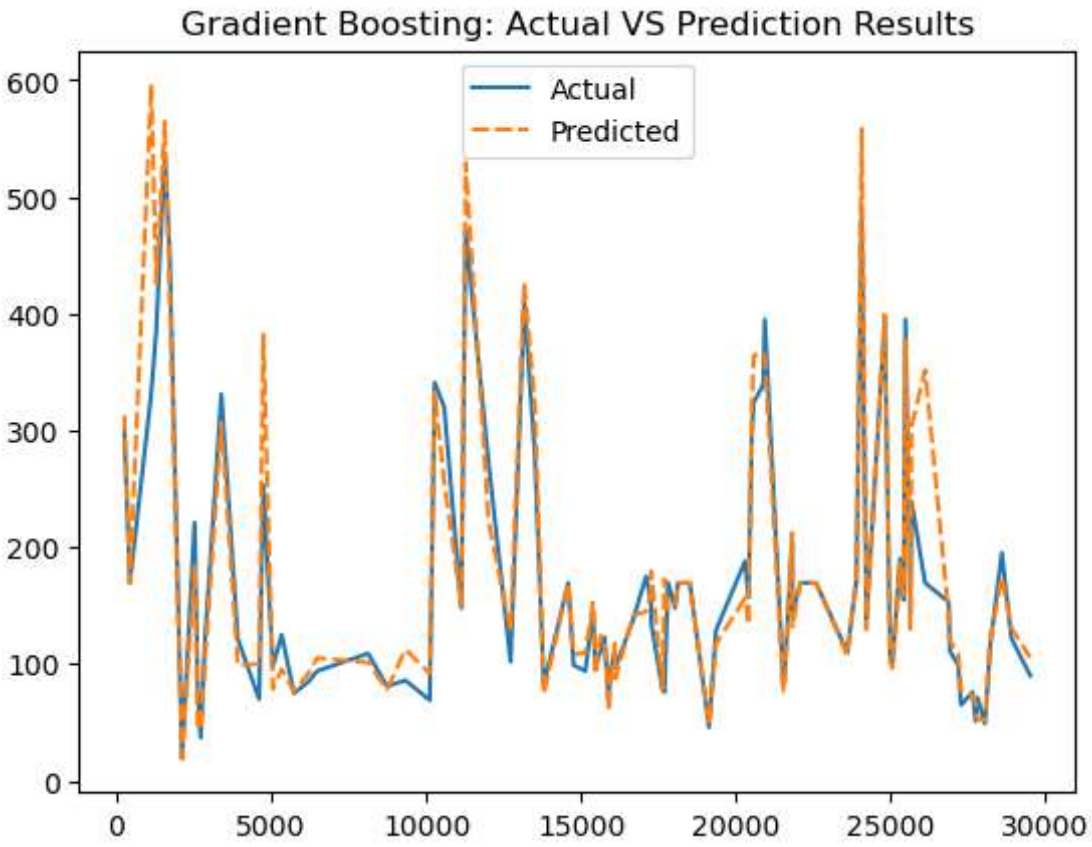
In [67]:
```
svr=SVR()
gb=GradientBoostingRegressor()
knn=KNeighborsRegressor()
```

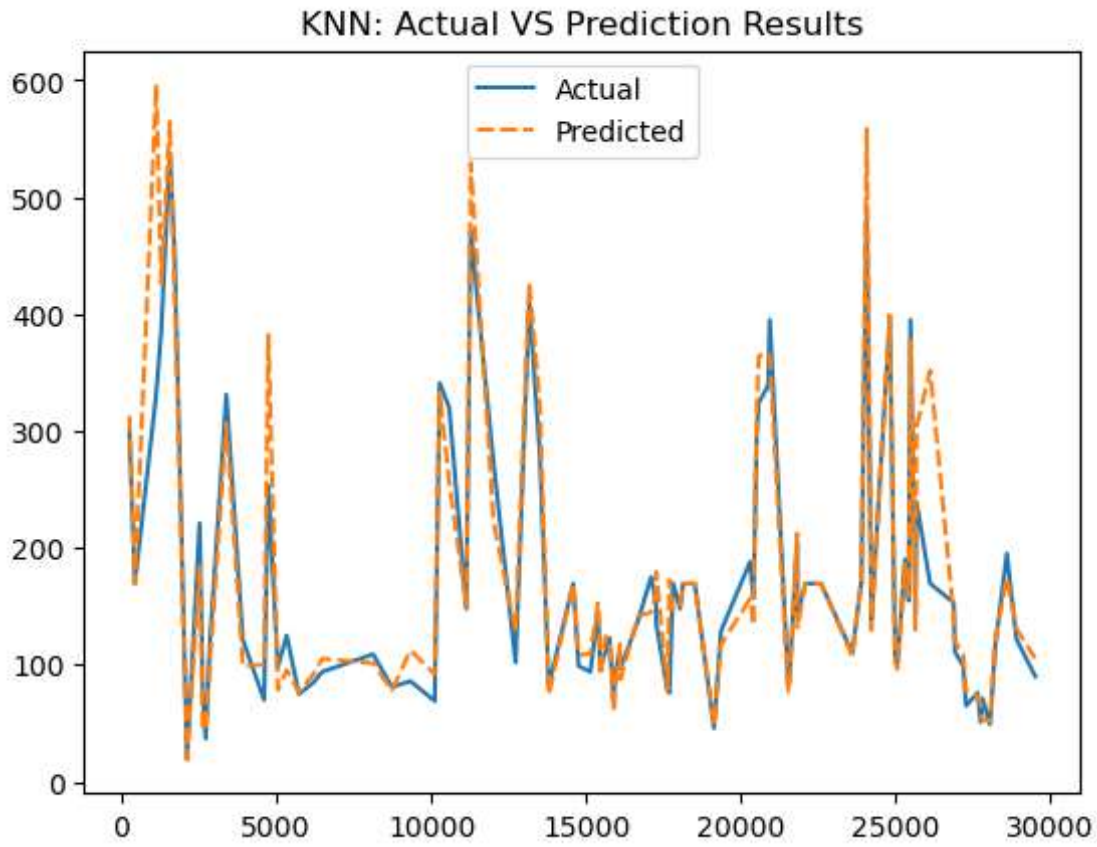
In [68]:
```
models=[svr, gb, knn]
model_names=["SVR", 'Gradient Boosting', 'KNN']
mae_list=[]
rmse_list=[]
evs_list=[]

for model, name in zip(models, model_names):
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mae=mean_absolute_error(y_test,y_pred)
    rmse=np.sqrt(mean_squared_error(y_test, y_pred))
    #r2s=r2_score(y_test, y_pred)
    evs= explained_variance_score(y_test, y_pred)
    mae_list.append(mae)
    rmse_list.append(rmse)
    evs_list.append(evs)
    print(f'{name}, MAE: {mae}')
    print(f'{name}, RMSE:{rmse}')
    #print(f'{name}, R2Score:{r2s}')
    print(f'{name},Explained Variance Score:{evs}')
    print(mae_list)
```

```
SVR, MAE: 32.605844290758704
SVR, RMSE:74.15426691974668
SVR,Explained Variance Score:0.6395302270632869
[32.605844290758704]
Gradient Boosting, MAE: 16.152569978709607
Gradient Boosting, RMSE:28.8469734198087
Gradient Boosting,Explained Variance Score:0.9448134754019115
[32.605844290758704, 16.152569978709607]
KNN, MAE: 22.887737938039614
KNN, RMSE:49.34201034955269
KNN,Explained Variance Score:0.8386524923386327
[32.605844290758704, 16.152569978709607, 22.887737938039614]
```

In [94]:
```
for model, name in zip(models, model_names):
    df=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    sns.lineplot(df.head(100))
    plt.title(f'{name}: Actual VS Prediction Results')
    plt.show()
```







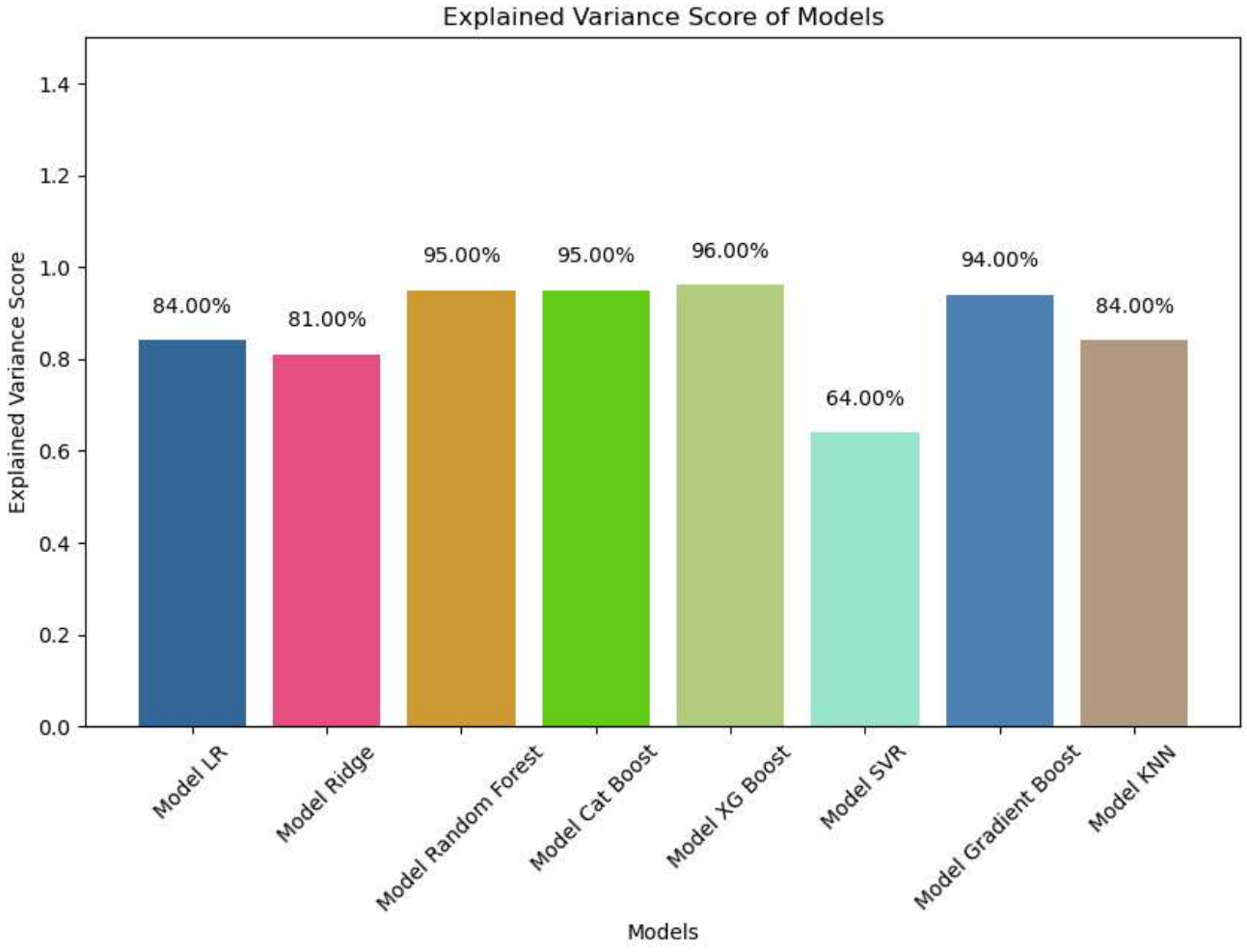
In [73]:
```
print(mae_list,rmse_list,evs_list)
```

```
[32.605844290758704, 16.152569978709607, 22.887737938039614] [74.15426691974668, 28.
8469734198087, 49.34201034955269] [0.6395302270632869, 0.9448134754019115, 0.8386524
923386327]
```

Visualize the Metric Scores of the Models Developed using different regression algorithms

The higher the explained variance score better accuracy of the model

In [74]:
```
evs_score = [evs, evs_rid, evs_rf, evs_cat, evs_xgb]
evs_scores= evs_score+ evs_list
Scores = [round(i,2) for i in evs_scores]

model_name=['Model LR','Model Ridge', 'Model Random Forest', 'Model Cat Boost', 'Model

colors= [(0.2,0.4,0.6),(0.9,0.3,0.5),(0.8,0.6,0.2), (0.4,0.8,0.1),(0.7,0.8,0.5),(0.6,

plt.figure(figsize=(10,6))
plt.bar(model_name, Scores, color=colors)
plt.xlabel('Models')
plt.ylabel('Explained Variance Score')
plt.title('Explained Variance Score of Models')
plt.xticks(rotation=45)
plt.ylim(bottom=0.0, top=1.50)

for i, score in enumerate(Scores):
    percentage = f'{score*100:.2f}%'
    plt.text(i, score+0.05, percentage, ha='center', va='bottom')

plt.show()
```

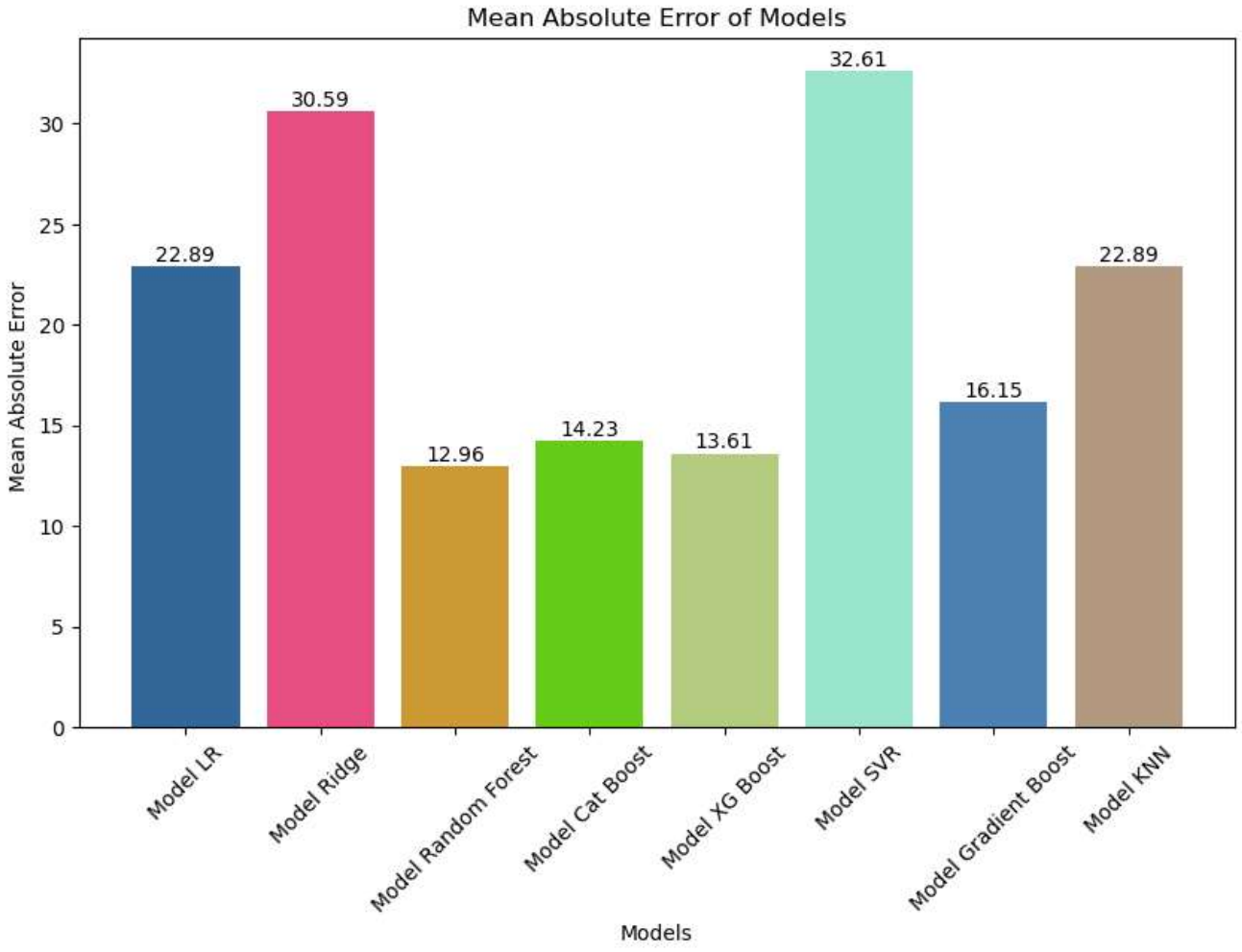


This graph shows that XGBoost Regression model performs well and good with highest accuracy(explained variance score) of 96% followed by Random forest regression and Cat Boost regression with accuracy(explained variance score) of 95% respectively.

MAE and RMSE of all the models

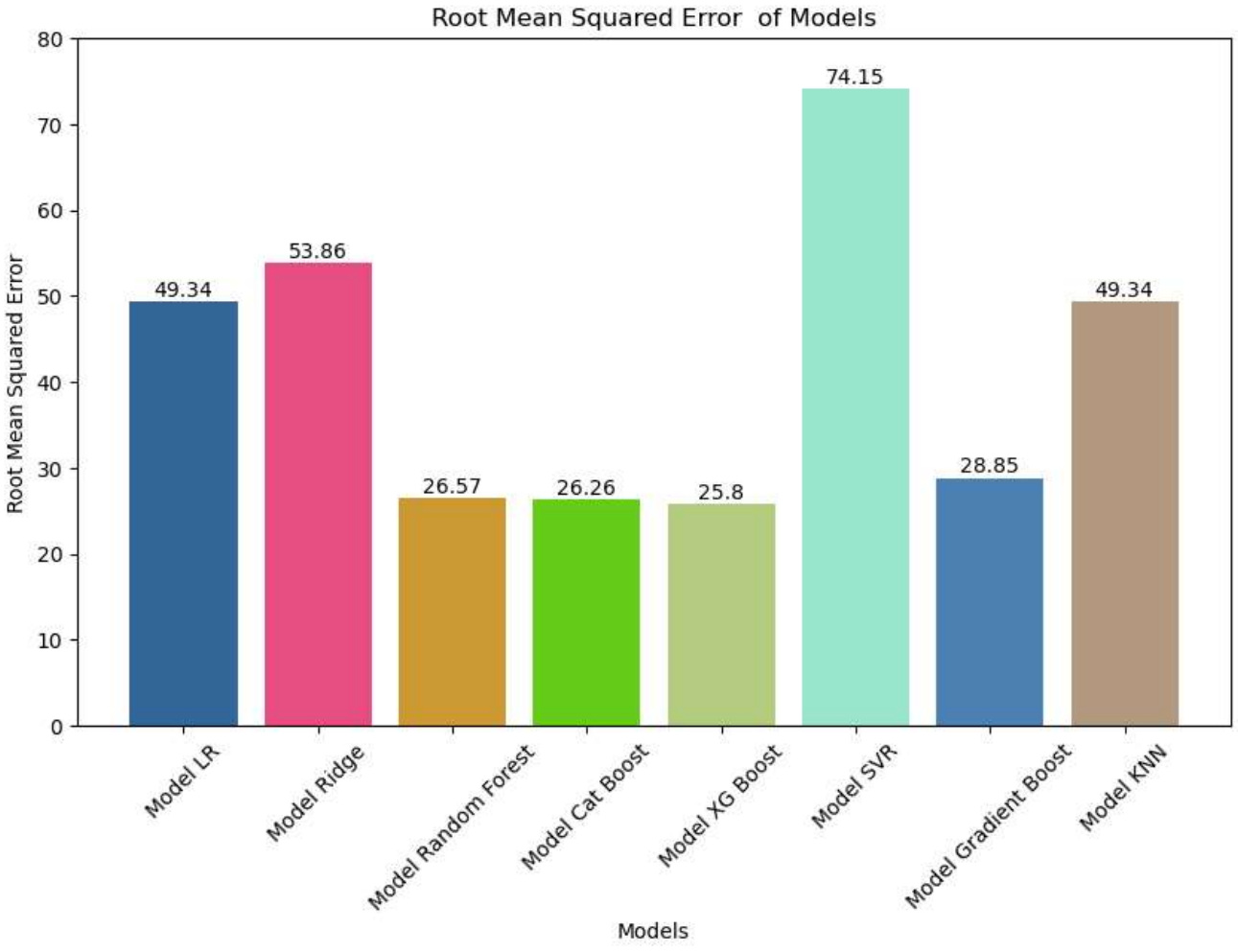
The lower the mean absolute error and root mean squared error th better the accuracy of the model

In [79]:
```
mae_s1 = [mae, mae_rid, mae_rf, mae_cat, mae_xgb]
mae_s= mae_s1 + mae_list
Scores_mae = [round(i,2) for i in mae_s]

model_name=['Model LR','Model Ridge', 'Model Random Forest', 'Model Cat Boost', 'Model

colors= [(0.2,0.4,0.6),(0.9,0.3,0.5),(0.8,0.6,0.2), (0.4,0.8,0.1),(0.7,0.8,0.5),(0.6,0

plt.figure(figsize=(10,6))
plt.bar(model_name, Scores_mae, color=colors)
plt.xlabel('Models')
plt.ylabel('Mean Absolute Error ')
plt.title('Mean Absolute Error of Models')
plt.xticks(rotation=45)

for i, score in enumerate(Scores_mae):
    plt.text(i, score+0.05,Scores_mae[i], ha='center', va='bottom')

plt.show()
```



XG Boost regressor has the Lowest MAE

In [89]:
```
rmse_s1 = [rmse, rmse_rid, rmse_rf, rmse_cat, rmse_xgb]
rmse_s= rmse_s1 + rmse_list
Scores_rmse = [round(i,2) for i in rmse_s]

model_name=['Model LR','Model Ridge', 'Model Random Forest', 'Model Cat Boost', 'Model

colors= [(0.2,0.4,0.6),(0.9,0.3,0.5),(0.8,0.6,0.2), (0.4,0.8,0.1),(0.7,0.8,0.5),(0.6,0

plt.figure(figsize=(10,6))
plt.bar(model_name, Scores_rmse, color=colors)
plt.xlabel('Models')
plt.ylabel('Root Mean Squared Error')
plt.title('Root Mean Squared Error  of Models')
plt.xticks(rotation=45)
plt.ylim(bottom=0,top=80)

for i, score in enumerate(Scores_rmse):
    plt.text(i, score+0.05,Scores_rmse[i],ha='center', va='bottom')

plt.show()
```



In [92]:
```
print(len(model_name))
print(len(Scores_mae))
print(len(Scores_rmse))
print(len(Scores))
```

```
8
8
8
8
```

In [93]:
```
dafr=pd.DataFrame({'Model': model_name,'MAE':Scores_mae,'RMSE':Scores_rmse,'ExpVarScor
dafr.set_index('Model',inplace=True)
print(dafr)
```

```
                          MAE    RMSE  ExVarScore
Model
Model LR                22.89  49.34        0.84
Model Ridge             30.59  53.86        0.81
Model Random Forest     12.96  26.57        0.95
Model Cat Boost         14.23  26.26        0.95
Model XG Boost          13.61  25.80        0.96
Model SVR               32.61  74.15        0.64
Model Gradient Boost    16.15  28.85        0.94
Model KNN               22.89  49.34        0.84
```

XG Boost model has the lowest MAE & RMSE score than other models.

Export The Model

In [72]:
```
import pickle

with open('ML_Model_XGB_AQI_Est.pkl','wb') as f:
    pickle.dump(model_xgb, f)
```

In [ ]: