TATA STEEL INTERNSHIP

PREDICTION MODEL (REGRESSION)

BY - NIDA FARNAZ
(KIIT UNIVERSITY)

CONTENT

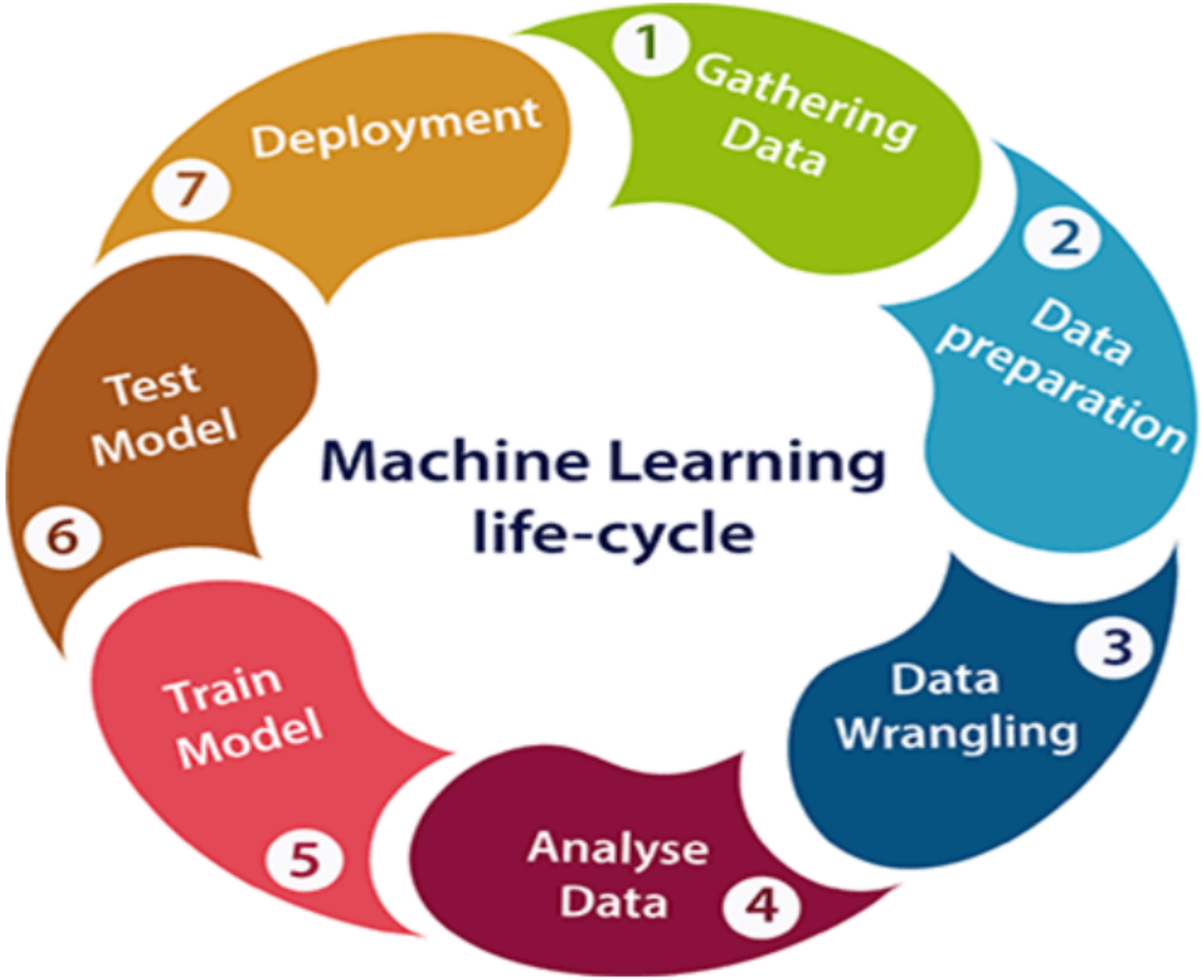
WHAT IS MACHINE LEARNING AND TATA STEEL DATA SET	1
DATA COLLECTION	2
DATA PREPROCESSING	3
FEATURE SELECTION	4
MODEL USED FOR PREDICTION	5
CONCLUSION	6

WHAT IS MACHINE LEARNING

MACHINE LEARNING IS A SUBFIELD OF ARTIFICIAL INTELLIGENCE (AI) THAT FOCUSES ON DEVELOPING ALGORITHMS AND MODELS THAT ENABLE COMPUTERS TO LEARN AND MAKE PREDICTIONS OR DECISIONS WITHOUT BEING EXPLICITLY PROGRAMMED. IT IS CONCERNED WITH THE DEVELOPMENT OF COMPUTATIONAL SYSTEMS THAT CAN LEARN FROM AND ADAPT TO DATA, ALLOWING THEM TO IMPROVE THEIR PERFORMANCE OR BEHAVIOR OVER TIME

- **SUPERVISED LEARNING**: IN SUPERVISED LEARNING, THE MODEL IS TRAINED USING LABELED DATA, WHERE BOTH INPUT DATA AND CORRESPONDING OUTPUT LABELS ARE PROVIDED. THE MODEL LEARNS TO MAP INPUT DATA TO OUTPUT LABELS, ENABLING IT TO MAKE PREDICTIONS OR CLASSIFICATIONS ON NEW, UNSEEN DATA.
- **UNSUPERVISED LEARNING**: IN UNSUPERVISED LEARNING, THE MODEL IS TRAINED ON UNLABELED DATA, WITHOUT ANY SPECIFIC OUTPUT LABELS. THE GOAL IS TO DISCOVER PATTERNS, STRUCTURES, OR RELATIONSHIPS WITHIN THE DATA, SUCH AS CLUSTERING SIMILAR DATA POINTS OR IDENTIFYING LATENT FACTORS.
- **REINFORCEMENT LEARNING:** IN REINFORCEMENT LEARNING, THE MODEL LEARNS BY INTERACTING WITH AN ENVIRONMENT AND RECEIVING FEEDBACK IN THE FORM OF REWARDS OR PENALTIES. THE GOAL IS TO LEARN OPTIMAL ACTIONS OR POLICIES TO MAXIMIZE THE CUMULATIVE REWARD OVER TIME.

MACHINE LEARNING LIFE CYCLE



SUPERVISED LEARNING (REGRESSION MODEL)

A REGRESSION MODEL IS A TYPE OF SUPERVISED LEARNING MODEL USED FOR PREDICTING A CONTINUOUS NUMERICAL VALUE OR A QUANTITY. IT ESTABLISHES A RELATIONSHIP BETWEEN A SET OF INPUT VARIABLES (ALSO CALLED INDEPENDENT VARIABLES, FEATURES, OR PREDICTORS) AND A SINGLE OUTPUT VARIABLE (ALSO CALLED THE DEPENDENT VARIABLE OR TARGET).

THE GENERAL FORM OF A REGRESSION MODEL CAN BE EXPRESSED AS:

$$Y = F(X) + E$$

WHERE:

- Y REPRESENTS THE DEPENDENT VARIABLE OR TARGET VARIABLE.
- X REPRESENTS THE SET OF INDEPENDENT VARIABLES OR FEATURES.
- F(.) REPRESENTS THE REGRESSION FUNCTION OR MODEL, WHICH CAPTURES THE RELATIONSHIP BETWEEN THE INPUT VARIABLES AND THE OUTPUT VARIABLE.
- E REPRESENTS THE ERROR TERM OR RESIDUAL, ACCOUNTING FOR THE VARIABILITY OR NOISE THAT CANNOT BE EXPLAINED BY THE MODEL.

TATA STEEL DATASET

THE DATASET I HAVE GOT BELONGS TO THE SUPERVISED LEARNING (REGRESSION) TYPE OF MACHINE LEARNING WHICH IS REGARDING IINFORMATION OF STACK EMISSIONS OF TATA STEEL FACTORIES WHERE I PERFORMAED DIFFERENT MACHINE LEARNING STEPS THAT BELONGS TO THE MACHINE LEARNING LIFECYCLE

THE STEPS ARE

- DATA COLLECTION -WHERE I UPLOADED THE DATASET USING PANDAS AND GOT THE BASIC INFORMATION LIKE COLUMNS ,ROWS AND THE DATA TYPES OF EACH COLUMN
- DATA PREPROCESSING- WHICH INCLUDES CLEANING OF DATASET DO THAT IT GIVES

ACCURATE PREDICTION

- FEATURE SELECTION- WHERE I HAVE PERFORMED FEATURE SCALING BY NORMALIZATION ,FOUND THE CORRELATION,FEATURE SELECTION BY MUTUAL INFORMATION METHOD
- TRAINING AND TESTING MODELS-WHERE I HAVE USED VARIOUS PREDICTION MODELS LIKE MULTIPLE REGRESSION MODEL,XG BOOST,SUPPORT VECTOR MACHINE AND RANDOM FOREST REGRESSOR TO GET THE RMSE VALUES WHICH SHOWS THE ACCURACY OF THE MODEL.

PLATFORM USED -GOOGLE COLAB

DATA COLLECTION

GATHER RELEVANT DATA THAT WILL BE USED TO TRAIN AND EVALUATE YOUR MACHINE LEARNING MODEL. THE QUALITY AND QUANTITY OF DATA ARE CRUCIAL FOR THE SUCCESS OF THE MODEL.

```
from google.colab import files
import io
import pandas as pd
data = files.upload()
```

Choose Files intern_data_spm.csv

- **intern_data_spm.csv**(text/csv) - 1046715 bytes, last modified: 6/6/2023 - 100% done

```
Saving intern_data_spm.csv to intern_data_spm.csv
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
%matplotlib inline
```

```
df=pd.read_csv(io.StringIO(data['intern_data_spm.csv'].decode('utf-8')))
```

IMPORTING ALL THE LIBRARIES

- **NUMPY** FOR ALL THE NUMERICAL OPERATIONS
- **PANDAS** FOR DATASET
- **SEABORN AND MATPLOTLIB** FOR DATA VISUALIZATION.

BASIC INFORMATION ABOUT THE DATA

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   CLOCK   5000 non-null   object 
 1   X1      5000 non-null   float64
 2   X2      5000 non-null   float64
 3   X3      5000 non-null   float64
 4   X4      5000 non-null   float64
 5   X5      5000 non-null   float64
 6   X6      5000 non-null   float64
 7   X7      5000 non-null   float64
 8   X8      5000 non-null   float64
 9   X9      5000 non-null   float64
 10  X10     5000 non-null   float64
 11  X11     5000 non-null   float64
 12  X12     5000 non-null   float64
 13  X13     5000 non-null   float64
 14  X14     5000 non-null   float64
 15  output  5000 non-null   float64
dtypes: float64(15), object(1)
memory usage: 625.1+ KB
```

```
df.head()
```

	CLOCK	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	output
0	2022-12-18 14:50:00	48.65105	38.97555	35.33245	36.00630	63.58980	10.96650	145.15495	384.60800	919.95	2.05030	2.66385	3.5270	3.5270	7.69330	37.87175
1	2022-12-31 12:10:00	46.30760	36.69085	32.61530	34.56650	62.49190	10.48560	153.96970	389.11030	880.05	1.59045	1.83155	1.9555	1.9835	7.69370	35.35060
2	2022-12-23 18:10:00	48.56460	38.86065	34.98840	35.50790	65.73585	19.72395	161.72370	383.47905	712.95	1.06160	1.54015	3.5815	0.0000	7.69370	37.73035
3	2022-12-19 05:30:00	47.59175	38.35315	34.27105	35.68830	64.44080	11.15725	136.59115	379.70615	920.05	2.04615	2.74185	3.6125	3.5880	7.69310	39.72865
4	2022-12-10 14:00:00	49.20060	38.82830	35.15390	36.69245	63.76605	4.46650	149.19130	403.56375	919.95	2.18880	2.68110	3.7480	3.7475	7.69395	52.24655

```
df.tail()
```

	CLOCK	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	outpu
4995	2022-12-29 17:50:00	48.970250	38.739500	35.964150	36.376850	65.717900	8.880750	152.399100	408.228700	909.900000	1.79045	2.555800	2.760500	2.788500	7.693900	37.10480
4996	2022-12-29 00:40:00	47.482000	38.186750	34.010450	35.813450	64.686100	17.171500	134.963450	368.411750	919.950000	1.89235	2.610600	2.799000	2.839500	7.693900	38.26550
4997	2022-12-19 03:20:00	47.177000	38.093600	34.074500	35.629350	63.249700	11.424150	135.836050	378.533050	920.050000	2.04950	2.748300	3.621500	3.631500	7.693300	38.43215
4998	2022-12-12 13:50:00	48.468167	39.066556	34.451111	36.946222	62.825278	10.619111	135.960722	368.527111	919.944444	2.13950	2.801833	3.478889	3.467222	7.693833	35.84133
4999	2022-12-23 13:30:00	44.971650	35.831900	33.488650	34.230950	64.126850	22.199800	139.359300	329.820850	731.000000	0.99180	1.251100	1.882000	1.848500	7.693800	33.49360

DATA PREPROCESSING

CLEAN AND PREPROCESS THE DATA TO REMOVE ANY NOISE, HANDLE MISSING VALUES, AND TRANSFORM THE DATA INTO A SUITABLE FORMAT FOR TRAINING THE MODEL.

- **NULL VALUES DETECTION**

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

```
CLOCK     0
X1        0
X2        0
X3        0
X4        0
X5        0
X6        0
X7        0
X8        0
X9        0
X10       0
X11       0
X12       0
X13       0
X14       0
output    0
dtype: int64
```

```
[9]  [features for features in df.columns if df[features].isnull().sum()>0]
```

```
[]
```

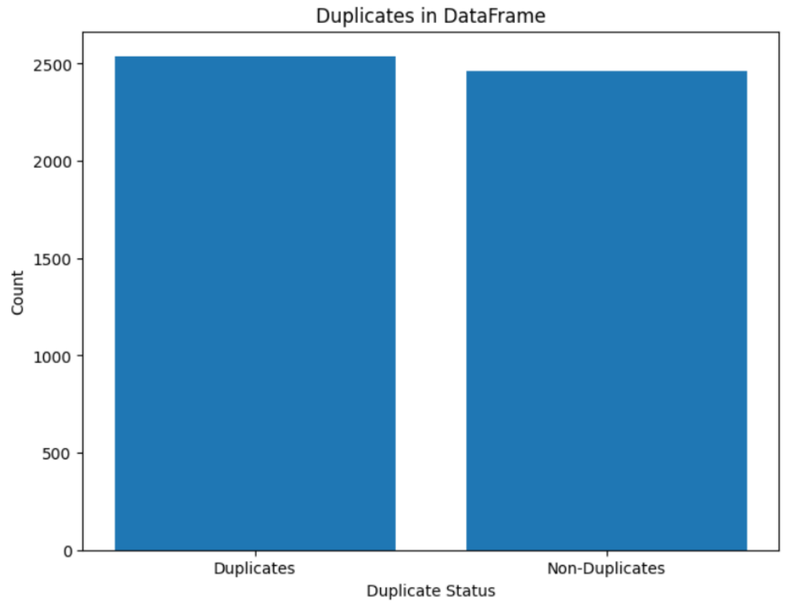
NO NULL VALUES SO THERE ARE NO MISSING VALUES

• DUPLICATE DETECTION AND REMOVAL

```
cols=df.columns
print(cols)
duplicates=df[['CLOCK', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'output']].duplicated()
duplicated_rows=df[duplicates]
print(duplicated_rows)
print(duplicates)
```

```
Index(['CLOCK', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'output'],
      dtype='object')
                 CLOCK        X1        X2        X3        X4  \
52    2022-12-13 06:00:00  48.111000  39.059250  34.437700  37.116600
107   2022-12-17 01:20:00  46.827900  37.523900  33.374050  35.422150
108   2022-12-19 17:00:00  47.433500  38.433700  34.218200  35.896800
118   2022-12-15 10:00:00  47.569650  38.379900  34.173000  36.184050
145   2022-12-13 06:00:00  48.111000  39.059250  34.437700  37.116600
...                   ...        ...        ...        ...        ...
4994  2022-12-25 07:40:00  46.379550  36.759050  33.223450  34.295950
4995  2022-12-29 17:50:00  48.970250  38.739500  35.964150  36.376850
4996  2022-12-29 00:40:00  47.482000  38.186750  34.010450  35.813450
4997  2022-12-19 03:20:00  47.177000  38.093600  34.074500  35.629350
4998  2022-12-12 13:50:00  48.468167  39.066556  34.451111  36.946222

            X5         X6          X7          X8          X9      X10  \
52    62.575800  17.052400  132.687400  358.084200  919.950000  2.17830
107   65.338500  10.199800  136.478150  369.819700  919.950000  1.93795
108   64.227100   8.185350  136.866000  363.454600  920.000000  2.05755
118   66.840300   8.052250  138.086150  379.072150  919.950000  1.80095
145   62.575800  17.052400  132.687400  358.084200  919.950000  2.17830
...         ...        ...         ...         ...         ...      ...
4994  64.590650   6.571300  156.399550  413.743950  920.000000  1.68955
```

DETECTING DUPLICATES



DATA VISUALIZATION

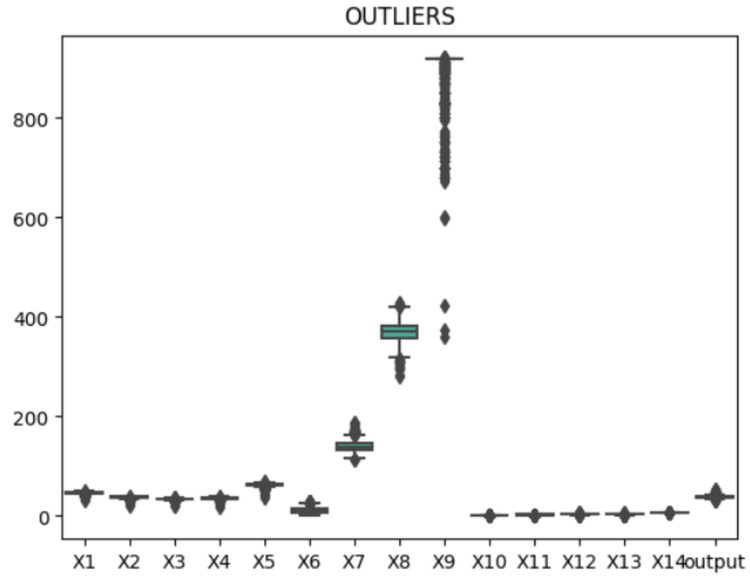
```
#REMOVING DUPLICATES
df=df.drop_duplicates()
print(df)
```

```
                 CLOCK        X1        X2        X3        X4        X5  \
0     2022-12-18 14:50:00  48.65105  38.97555  35.33245  36.00630  63.58980
1     2022-12-31 12:10:00  46.30760  36.69085  32.61530  34.56650  62.49190
2     2022-12-23 18:10:00  48.56460  38.86065  34.98840  35.50790  65.73585
3     2022-12-19 05:30:00  47.59175  38.35315  34.27105  35.68830  64.44080
4     2022-12-10 14:00:00  49.20060  38.82830  35.15390  36.69245  63.76605
...                   ...       ...       ...       ...       ...       ...
4987  2022-12-21 17:10:00  46.74265  37.91030  33.77445  35.46620  62.01295
4988  2022-12-20 13:30:00  46.81005  37.49205  34.34830  35.62205  62.44480
4989  2022-12-24 04:50:00  46.55275  38.00450  32.84530  35.70645  64.54105
4990  2022-12-13 22:20:00  49.12230  39.84660  35.99265  37.92645  65.72715
4999  2022-12-23 13:30:00  44.97165  35.83190  33.48865  34.23095  64.12685
```

REMOVING DUPLICATES

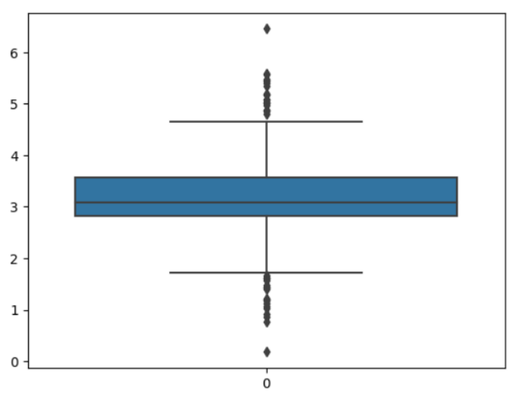
• OUTLIER DETECTION AND REMOVAL

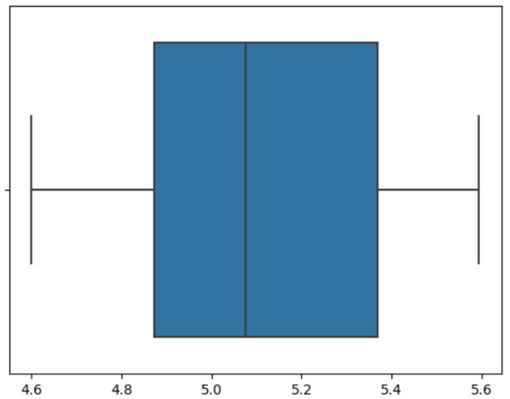
```
def find_outliers(data, threshold=3):   #defining a funciton to check wheather the dataset have outlier or not

    mean = data.mean()
    std = data.std()
    z_scores = (data - mean) / std
    outliers = data[np.abs(z_scores) > threshold]

    return outliers
numerical_cols = ['CLOCK', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'output']
numeric_data = df[numerical_cols]


outliers = find_outliers(numeric_data)
print(outliers)
```



DETECTING OUTLIERS BY Z-SCORE METHOD

```
df['X1'].skew()
percentile25=df['X1'].quantile(0.05)
percentile75=df['X1'].quantile(0.95)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X1']<upper_limit)&(df['X1']>lower_limit)]
print(len(df))
print(len(newdf))
```





BEFORE REMOVAL

AFTER REMOVAL

REMOVING OUTLIERS BY IQR METHOD

DATA VISUALIZATION

• CHECKING STATIONARITY

```
[69] from statsmodels.tsa.stattools import adfuller

     result = adfuller(newdf['X1'])
     p_value = result[1]
     print("ADF test p-value:", p_value)

     if p_value <= 0.05:
         print("The time series is stationary.")
```

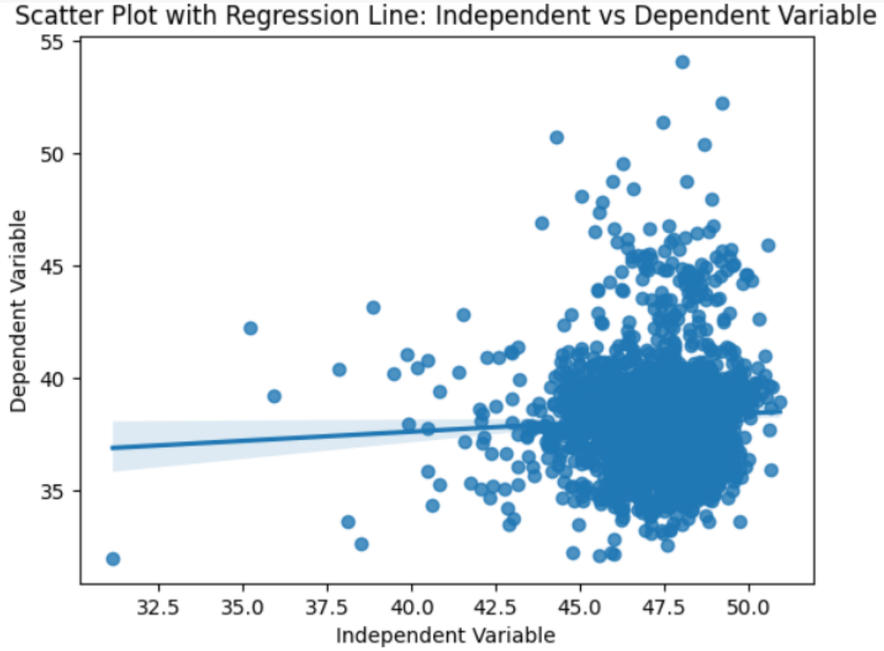
```
ADF test p-value: 0.0006258385649311937
The time series is stationary.
```

```
     result = adfuller(newdf['X2'])
     p_value = result[1]
     print("ADF test p-value:", p_value)

     if p_value <= 0.05:
         print("The time series is stationary.")
```

```
ADF test p-value: 0.7365018642254951
```

• SCATTER PLOT WITH REGRESSION LINE

```
for col in df.select_dtypes(include='number'):
  sns.regplot(x=col, y='output', data=df)
  plt.xlabel('Independent Variable')
  plt.ylabel('Dependent Variable')
  plt.title('Scatter Plot with Regression Line: Independent vs Dependent Variable')
  plt.show()
```



FEATURE SELECTION

- **VARIANCE THRESHOLD**

VARIANCE THRESHOLD IS A FEATURE SELECTION TECHNIQUE USED TO REMOVE FEATURES WITH LOW VARIANCE FROM A DATASET.

```
[88] #VARIANCE THRESHOLD
     from sklearn.feature_selection import VarianceThreshold
     var_thres=VarianceThreshold(threshold=0)
     var_thres.fit(X_train)
     sum(var_thres.get_support())
     constant_columns=[column for column in X_train.columns
                       if column not in X_train.columns[var_thres.get_support()]]

     print(len(constant_columns))
```

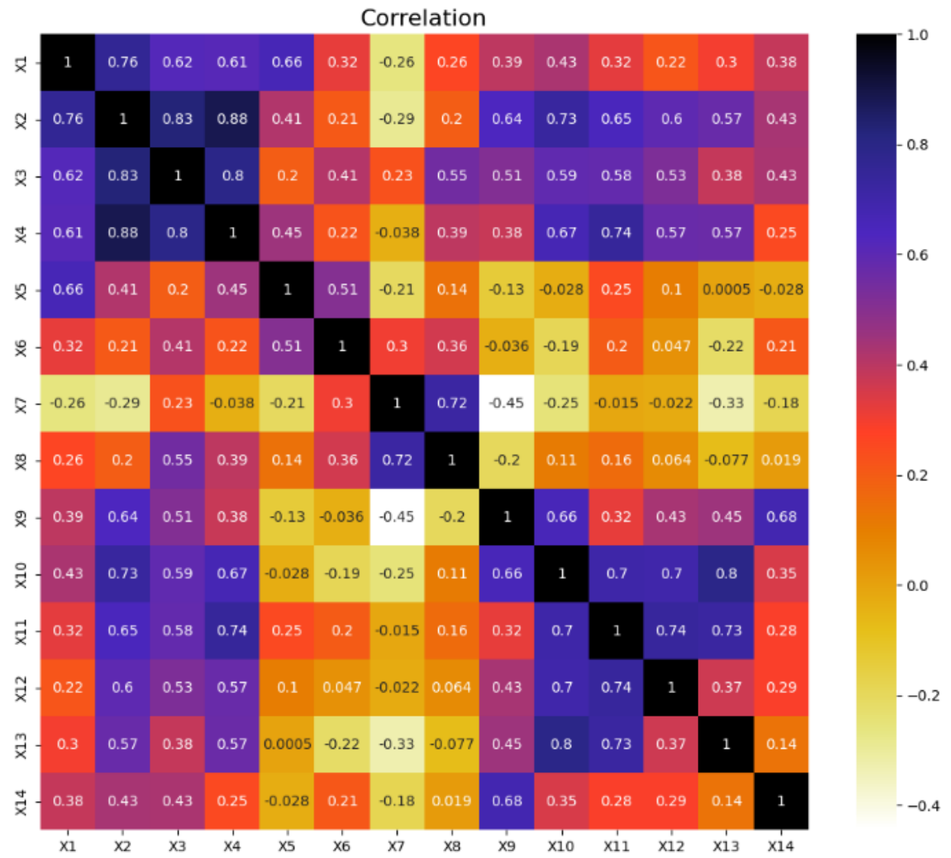
```
0
```

THIS SOWS THAT THERE ARE NO VALUES WITH LOW VARIANCE

• CORRELATION

CORRELATION IS A STATISTICAL MEASURE THAT QUANTIFIES THE DEGREE OF RELATIONSHIP OR ASSOCIATION BETWEEN TWO VARIABLES.

```
plt.figure(figsize=(12,10))
cor=X_train.corr()
sns.heatmap(cor,annot=True,cmap=plt.cm.CMRmap_r)
plt.title('Correlation',y=1,size=16)
plt.show()
```



DATA VISUALIZATION

```
[90] #selectiong higly correlated values
     def correlation(newdf,threshold):
       col_corr=set()
       corr_matrix=newdf.corr()
       for i in range(len(corr_matrix.columns)):
         for j in range(i):
           if abs(corr_matrix.iloc[i,j])>threshold:
             colname=corr_matrix.columns[i]
             col_corr.add(colname)
       return col_corr
     corr_features=correlation(X_train,0.7)
     len(set(corr_features))
```

```
8
```

SELECTING CORRELATED VALUES

```
[91] corr_features
     X_train.drop(corr_features,axis=1)
     X_test.drop(corr_features,axis=1)
```

	X1	X5	X6	X7	X9	X14
968	48.13655	61.14210	3.91925	144.55780	920.10	7.69365
2110	45.05445	55.27700	16.17585	177.59975	836.10	7.69400
4	49.20060	63.76605	4.46650	149.19130	919.95	7.69395
4512	48.81600	64.98010	9.06215	161.62080	883.95	7.69385
1225	44.29340	62.59475	13.92400	152.64485	850.45	7.69350
1051	50.58130	66.68180	8.69540	148.68040	920.10	7.69395
1255	46.41550	62.25225	3.63800	128.27790	919.90	7.69355
86	46.59040	63.99565	6.69065	154.72790	919.95	7.69375

REMOVING CORREALTED VALUES

• FEATURE SELECTION BY MUTUAL INFORMATION

MUTUAL INFORMATION CAN BE USED TO ESTIMATE THE RELEVANCE OR IMPORTANCE OF FEATURES WITH RESPECT TO THE TARGET VARIABLE.

```
[93] from sklearn.feature_selection import mutual_info_regression
     #to determine mutual information
     mutual_info=mutual_info_regression(X_train.fillna(0),y_train)
     mutual_info=pd.Series(mutual_info)
     mutual_info.index=X_train.columns
     mutual_info.sort_values(ascending=False)
```

```
X8     0.153901
X7     0.124133
X4     0.103617
X6     0.066129
X10    0.065412
X3     0.064835
X1     0.000253
X2     0.000000
X5     0.000000
X9     0.000000
X11    0.000000
X12    0.000000
X13    0.000000
X14    0.000000
dtype: float64
```

```
from sklearn.feature_selection import SelectPercentile
selected_top_columns=SelectPercentile(mutual_info_regression,percentile=20)
selected_top_columns.fit(X_train.fillna(0),y_train)
X_train.columns[selected_top_columns.get_support()]
```

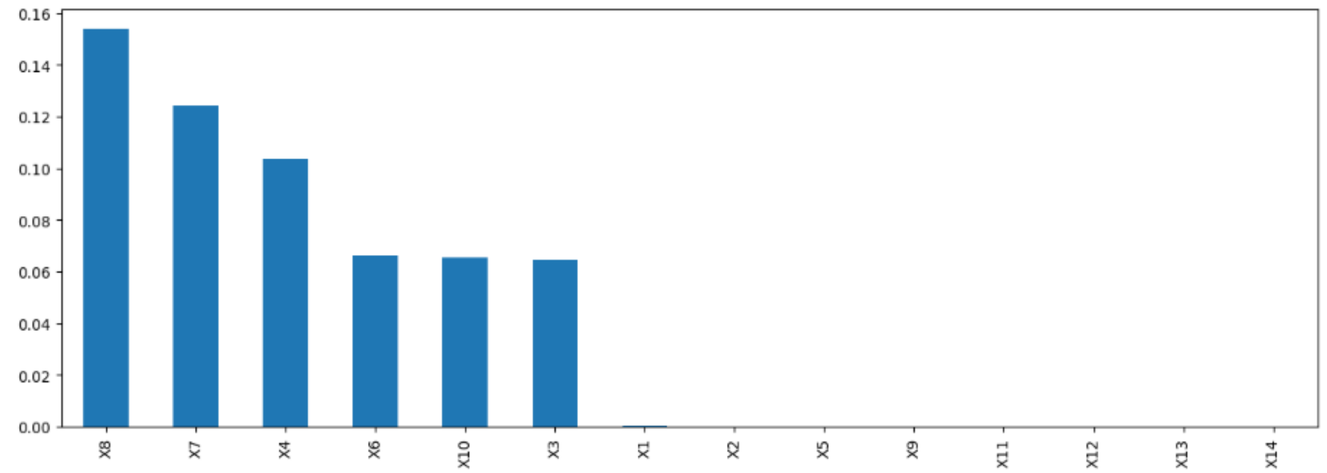
```
Index(['X4', 'X7', 'X8'], dtype='object')
```

SELECTED FEATURES

BAR PLOTT

```
] mutual_info.sort_values(ascending=False).plot.bar(figsize=(15,5))
```

```
<Axes: >
```



• DIRECTIONAL MATRIX

```
# directional matrix
correlation_matrix = newdf.corr()
directional_matrix = np.zeros(correlation_matrix.shape, dtype=str)

for i in range(correlation_matrix.shape[0]):
    for j in range(correlation_matrix.shape[1]):
        if i == j:
            directional_matrix[i, j] = 'self'
        elif correlation_matrix.iloc[i, j] > 0:
            directional_matrix[i, j] = 'positive'
        elif correlation_matrix.iloc[i, j] < 0:
            directional_matrix[i, j] = 'negative'
        else:
            directional_matrix[i, j] = 'no relationship'

print(directional_matrix)
```

```
[['s' 'p' 'p' 'p' 'p' 'p' 'n' 'p' 'p' 'p' 'p' 'p' 'p' 'n']
 ['p' 's' 'p' 'p' 'p' 'p' 'n' 'p' 'p' 'p' 'p' 'p' 'p' 'p']
 ['p' 'p' 's' 'p' 'p' 'p' 'p' 'p' 'p' 'p' 'p' 'p' 'p' 'n']
 ['p' 'p' 'p' 's' 'p' 'p' 'n' 'p' 'p' 'p' 'p' 'p' 'p' 'p']
 ['p' 'p' 'p' 'p' 's' 'p' 'n' 'p' 'p' 'p' 'p' 'p' 'n' 'p']
 ['p' 'p' 'p' 'p' 'p' 's' 'p' 'p' 'n' 'n' 'n' 'p' 'n' 'p']
 ['n' 'n' 'p' 'n' 'n' 'p' 's' 'p' 'n' 'n' 'n' 'n' 'n' 'p']
 ['p' 'p' 'p' 'p' 'p' 'p' 'p' 's' 'n' 'n' 'p' 'n' 'n' 'p']
 ['p' 'p' 'p' 'p' 'p' 'n' 'n' 'n' 's' 'p' 'p' 'p' 'p' 'n']
 ['p' 'p' 'p' 'p' 'p' 'n' 'n' 'n' 'p' 's' 'p' 'p' 'p' 'n']
 ['p' 'p' 'p' 'p' 'p' 'n' 'n' 'p' 'p' 'p' 's' 'p' 'p' 'n']
 ['p' 'p' 'p' 'p' 'p' 'p' 'n' 'n' 'p' 'p' 'p' 's' 'p' 'p']
 ['p' 'p' 'p' 'p' 'p' 'n' 'n' 'n' 'p' 'p' 'p' 'p' 's' 'n']
 ['p' 'p' 'p' 'p' 'n' 'p' 'p' 'p' 'p' 'p' 'p' 'p' 's' 'p']
 ['n' 'p' 'n' 'p' 'p' 'p' 'p' 'n' 'n' 'n' 'p' 'n' 'p' 's']]
<ipython-input-96-61b765aea716>:2: FutureWarning: The default value of numeric_only in DataFrame.cor
  correlation_matrix = newdf.corr()
```

• APPLYING NORMALISATION BY MINMAXSCALER AND SPLITTING THE DATA

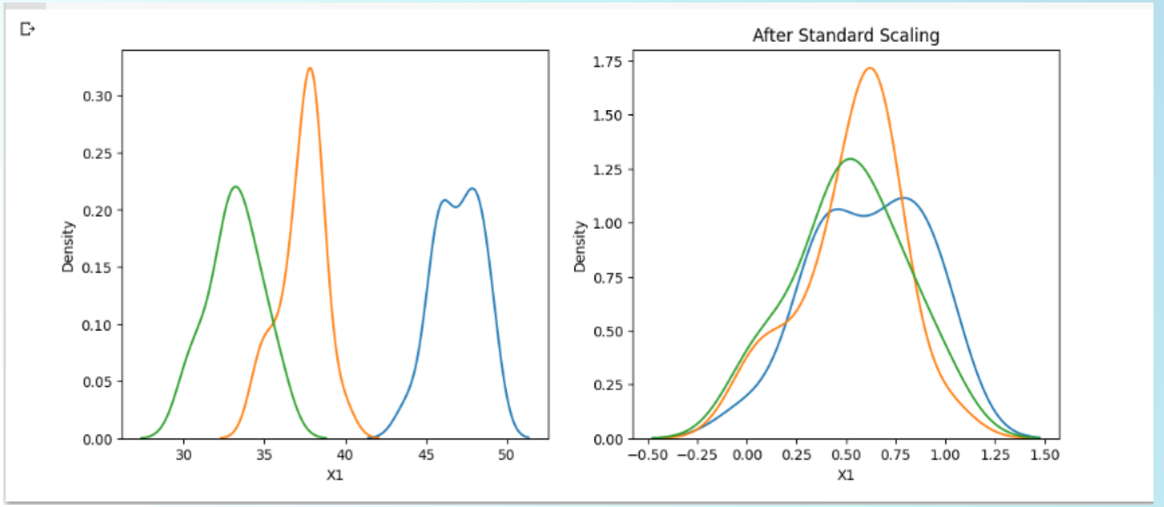
MIN-MAX SCALING (NORMALIZATION): THIS TECHNIQUE SCALES THE FEATURES TO A SPECIFIC RANGE, TYPICALLY BETWEEN 0 AND 1. IT IS ACHIEVED BY SUBTRACTING THE MINIMUM VALUE OF THE FEATURE AND DIVIDING IT BY THE RANGE (MAXIMUM VALUE MINUS MINIMUM VALUE). THE FORMULA FOR MIN-MAX SCALING IS:

$$X\_SCALED = (X - X\_MIN) / (X\_MAX - X\_MIN)$$

X_SCALED = (X - X_MIN) / (X_MAX - X_MIN)

```
[ ] from sklearn.preprocessing import MinMaxScaler
    scaler=MinMaxScaler()
    scaler.fit(X_train)
    X_train_scaled=scaler.transform(X_train)
    X_test_scaled=scaler.transform(X_test)
```

```
X_train_scaled=pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_test_scaled=pd.DataFrame(X_test_scaled,columns=X_test.columns)
```



MODELS USED FOR PREDICTION

- **MULTIVARIATE REGRESSION MODEL WITH HYPERPARAMETER TUNING**

LINEAR REGRESSION IS USED FOR REGRESSION TASKS, WHERE THE GOAL IS TO PREDICT A CONTINUOUS NUMERICAL VALUE. IT ASSUMES A LINEAR RELATIONSHIP BETWEEN THE INPUT FEATURES AND THE TARGET VARIABLE.

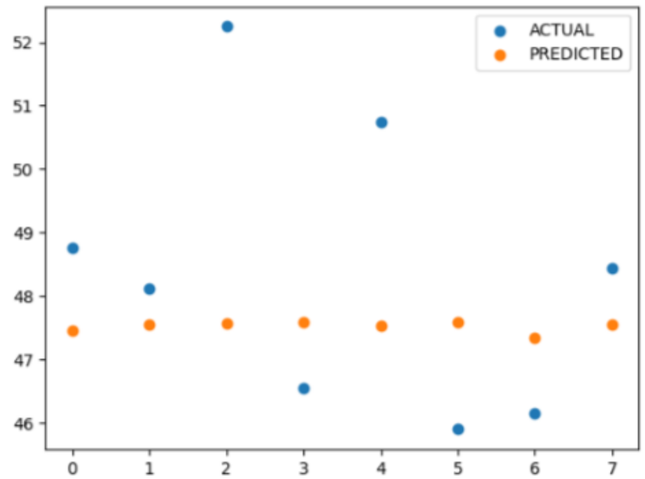
ML MODEL USING LINEAR REGRESSION

```
[ ]  from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     from sklearn.model_selection import train_test_split
     import numpy as np
     from sklearn.model_selection import  GridSearchCV
     from sklearn.linear_model import Ridge
     model = LinearRegression()

     param_grid = {'alpha': [0.1, 1.0, 10.0], 'max_iter': [10, 100, 1000],'fit_intercept':[True,False]}
     ridge = Ridge(param_grid)

     #cross-validation
     grid_search = GridSearchCV(ridge, param_grid, cv=5)
     grid_search.fit(X_train_scaled,y_train)

     print("hyperparameters:", grid_search.best_params_)
     print("score:", grid_search.best_score_)

     model = grid_search.best_estimator_
     y_pred = model.predict(X_test_scaled)

     mse = mean_squared_error(y_test, y_pred)
     rmse1 = np.sqrt(mse)
     coefficients =model.coef_
     intercept =model.intercept_

     print("Mean Squared Error:", mse)
     print("Root Mean Squared Error:", rmse1)
     print("Coefficients:", coefficients)
     print("Intercept:", intercept)
     print(y_pred)
```

```
hyperparameters: {'alpha': 10.0, 'fit_intercept': True, 'max_iter': 10}
score: -4.0244334465974365
Mean Squared Error: 5.048998427024806
Root Mean Squared Error: 2.2469976473118094
Coefficients: [ 0.03540168 -0.00746005 -0.03035131  0.00650708  0.04756259  0.1035437
  0.00727843  0.08593501  0.06916387 -0.01895683 -0.02358537  0.04285931
 -0.09678309  0.22221922]
Intercept: 47.18707633855994
[47.45448098 47.55088246 47.56592632 47.58537703 47.52846727 47.59133999
 47.34566342 47.5416299 ]
```

HYPERPARAMETERS AND ACURRACY

```
[ ]  import matplotlib.pyplot as plt

     plt.scatter(range(len(y_test)), y_test)
     plt.scatter(range(len(y_pred)), y_pred)
     plt.legend(['ACTUAL', 'PREDICTED'])
     plt.show()
```



ACTUAL VS PREDICTED SCATTERED PLOT

• XGBOOST MODEL WITH HYPERPARAMETER TUNING

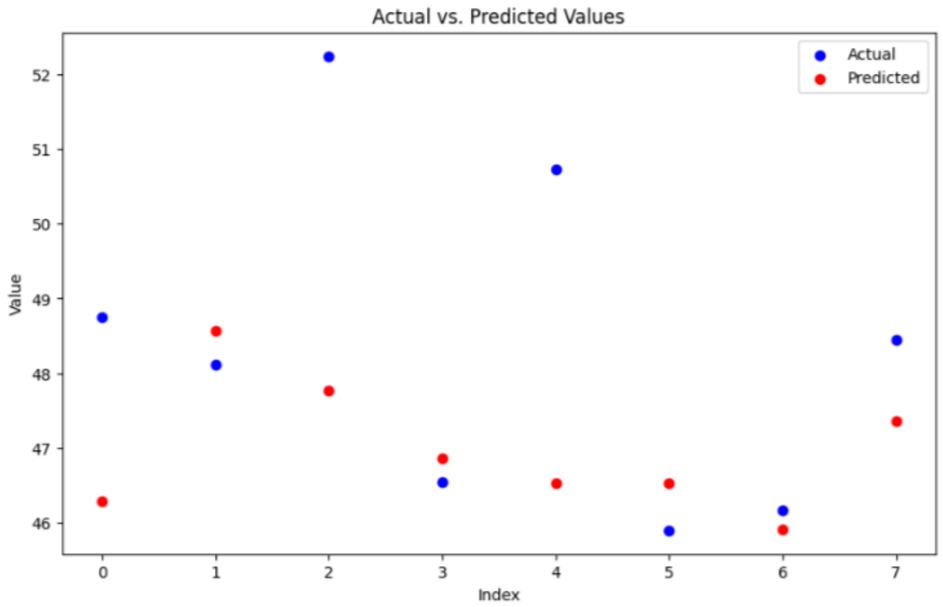
GRADIENT BOOSTING MODELS, SUCH AS GRADIENT BOOSTED TREES OR XGBOOST, ARE ENSEMBLE MODELS THAT COMBINE WEAK LEARNERS (E.G., DECISION TREES) IN A SEQUENTIAL MANNER, WHERE EACH SUBSEQUENT LEARNER CORRECTS THE MISTAKES MADE BY THE PREVIOUS LEARNERS.

```python
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': list(range(100, 1000, 100))
}
model = xgb.XGBRegressor(objective='reg:squarederror', max_depth=3, learning_rate=0.1, n_estimators=100)


grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train_scaled, y_train)

print("Hyperparameters:", grid_search.best_params_)
print("Best score:", -grid_search.best_score_)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse2= np.sqrt(mse)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse2)

r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)
```

```
Hyperparameters: {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 600}
Best score: 4.403528439173089
Mean Squared Error: 5.711810170616317
Mean Absolute Error: 1.7362405120849589
Root Mean Squared Error: 2.3899393654685714
R-squared Score: -0.2984468814728565
```

HYPERPARAMETERS AND ACURRACY

```python
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(y_pred)), y_pred, color='r', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```



ACTUAL VS PREDICTED SCATTERED PLOT

• SUPPORT VECTOR REGRESSION MODEL WITH HYPERPARAMETER TUNING

SVM IS A POWERFUL ALGORITHM USED FOR BOTH CLASSIFICATION AND REGRESSION TASKS. IT CONSTRUCTS A HYPERPLANE OR SET OF HYPERPLANES THAT MAXIMIZE THE SEPARATION BETWEEN CLASSES OR MINIMIZE THE REGRESSION ERROR.

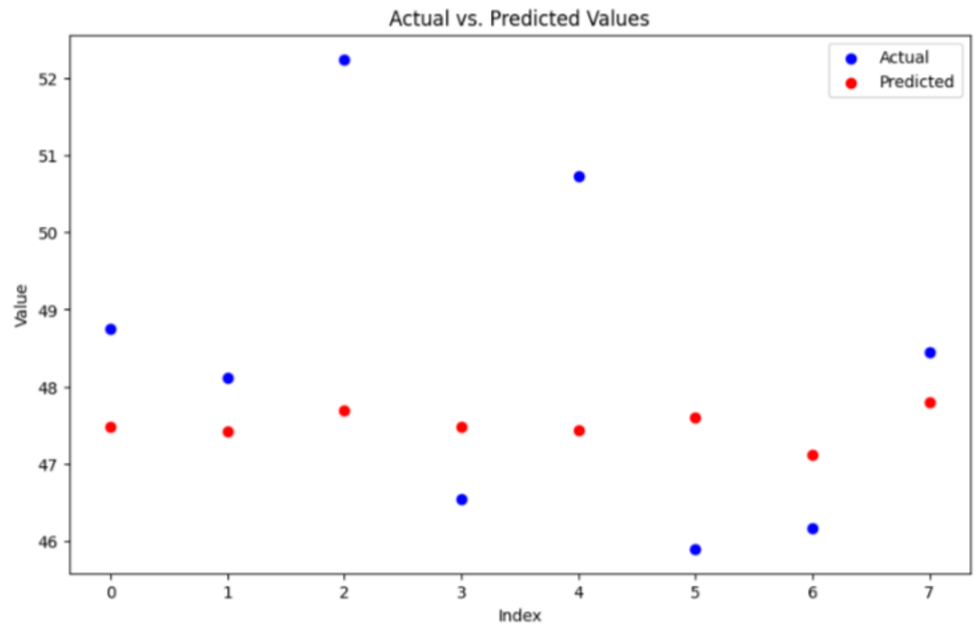
ML MODEL USING SUPPORT VECTOR REGRESSION

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import GridSearchCV

model = SVR(kernel='rbf')
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.01, 0.1, 1],
    'epsilon': [0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}
# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse3 = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Hyperparameters:", grid_search.best_params_)
print("Best score:", -grid_search.best_score_)
print("Root Mean Squared Error:", rmse3)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 1.760160533457226
Hyperparameters: {'C': 1, 'epsilon': 1, 'gamma': 1, 'kernel': 'rbf'}
Best score: 2.484413222634023
Root Mean Squared Error: 2.2044486319739613
Mean Squared Error: 4.859593771011871
R-squared: -0.1047153509505423
```

HYPERPARAMETERS AND ACURRACY

```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(y_pred)), y_pred, color='r', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```



ACTUAL VS PREDICTED SCATTERED PLOT

• RANDOM FOREST REGRESSOR MODEL WITH HYPERPARAMETER TUNING

RANDOM FOREST IS AN ENSEMBLE LEARNING METHOD THAT COMBINES MULTIPLE DECISION TREES. IT IMPROVES UPON DECISION TREES BY REDUCING OVERFITTING AND INCREASING ROBUSTNESS AND ACCURACY.

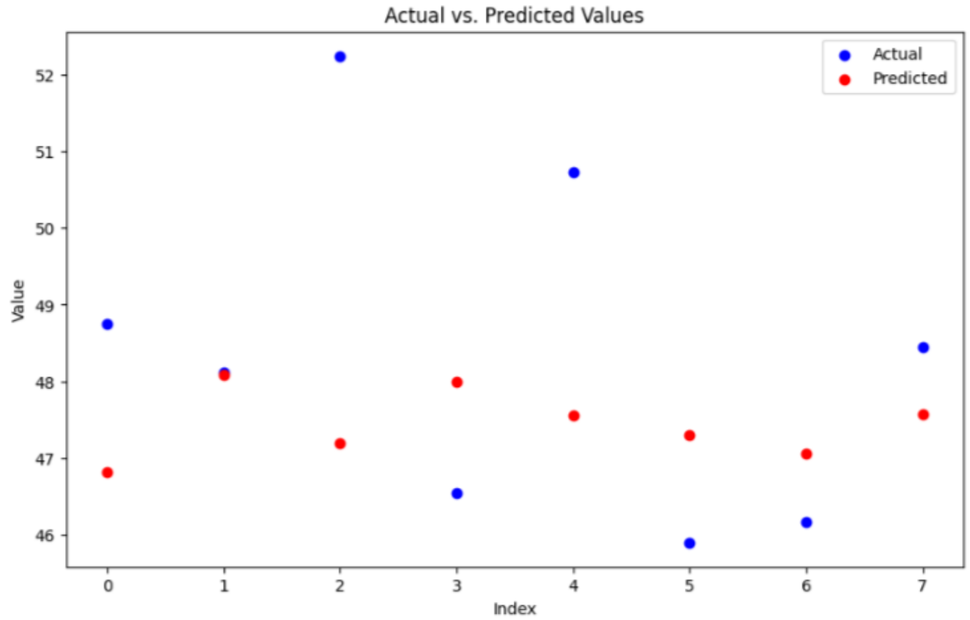
ML MODEL USING RANDOM FOREST REGRESSOR

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
param_grid = {
    'n_estimators': list(range(100, 1000, 100)),
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
model = RandomForestRegressor(random_state=42)
# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse4 = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Hyperparameters:", grid_search.best_params_)
print("Best score:", -grid_search.best_score_)
print("Root Mean Squared Error:", rmse4)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 1.8541532208282652
Hyperparameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 100}
Best score: 2.862188392956128
Root Mean Squared Error: 2.3745221579299147
Mean Squared Error: 5.638355478500138
R-squared: -0.28174866968730794
```

HYPERPARAMETERS AND ACURRACY

```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(y_pred)), y_pred, color='r', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```

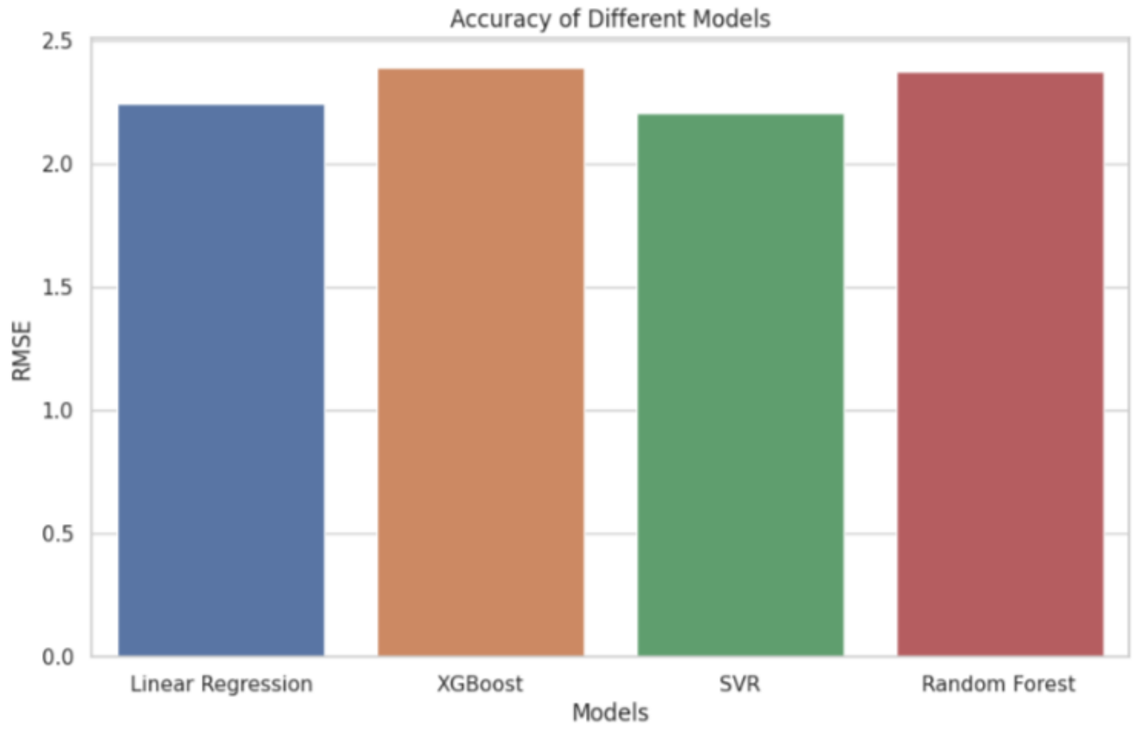


ACTUAL VS PREDICTED SCATTERED PLOT

CONCLUSION

MODEL ACCURACY COMPARISION

```
models = ['Linear Regression','XGBoost','SVR','Random Forest']
rmse= [rmse1,rmse2,rmse3,rmse4]
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=rmse)
plt.title('Accuracy of Different Models')
plt.xlabel('Models')
plt.ylabel('RMSE')
plt.show()
```



From the following bar plots we got to know that XGBOOST gives the most accurate prediction

AFTER TESTING AND TRAINING THE DATASET ON DIFFETRENT PREDICTION MODELS AND PLOTTING THE GRAPH OF HTEIR RMSE VALUES WHICH SHOES THE ACCURACY WE GOT TO KNOW THAT **XGBOOST REGRSSION MODEL** GIVES THE MOST ACCURATE PREDICTION FOR THE GIVE DATASET.