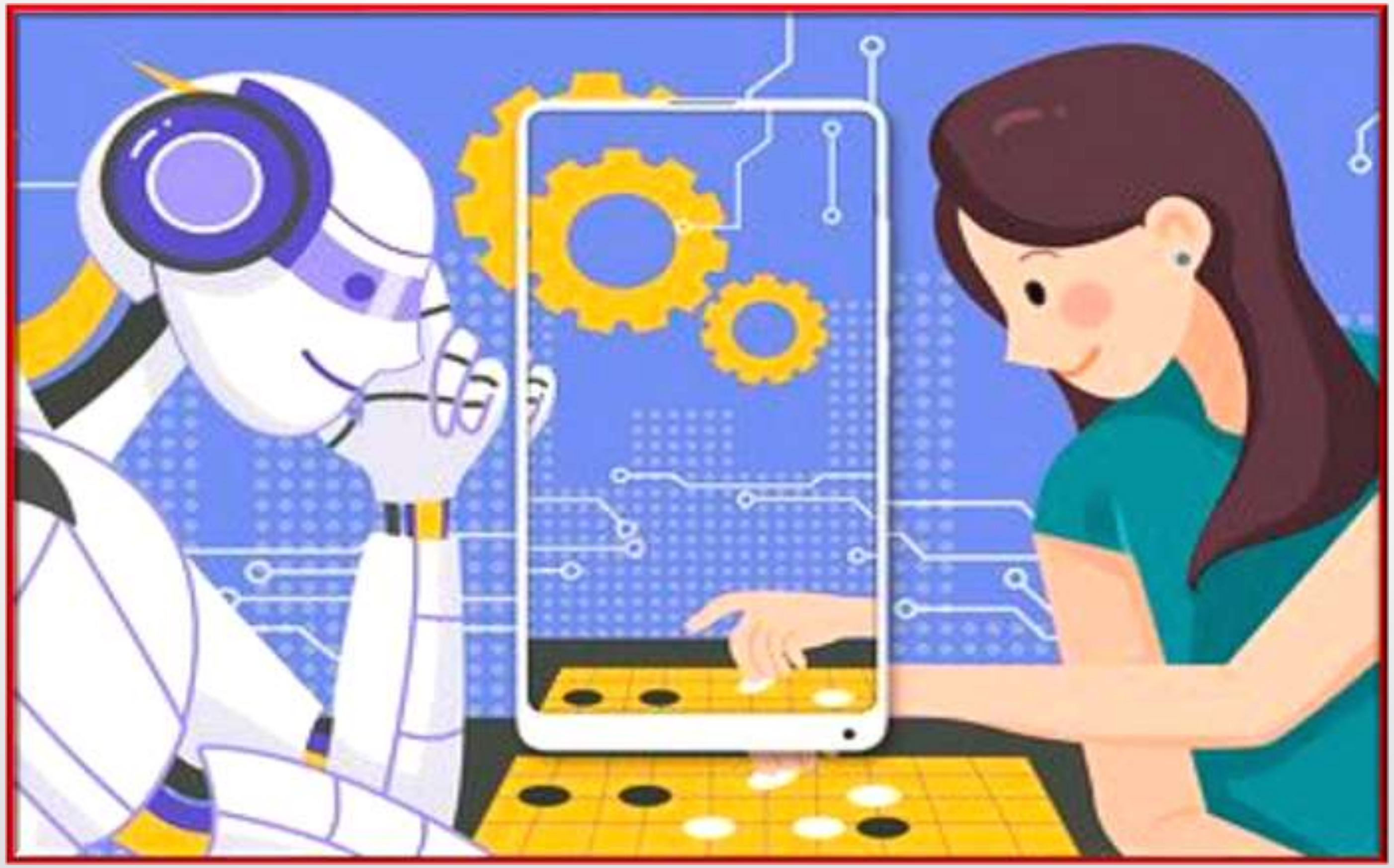
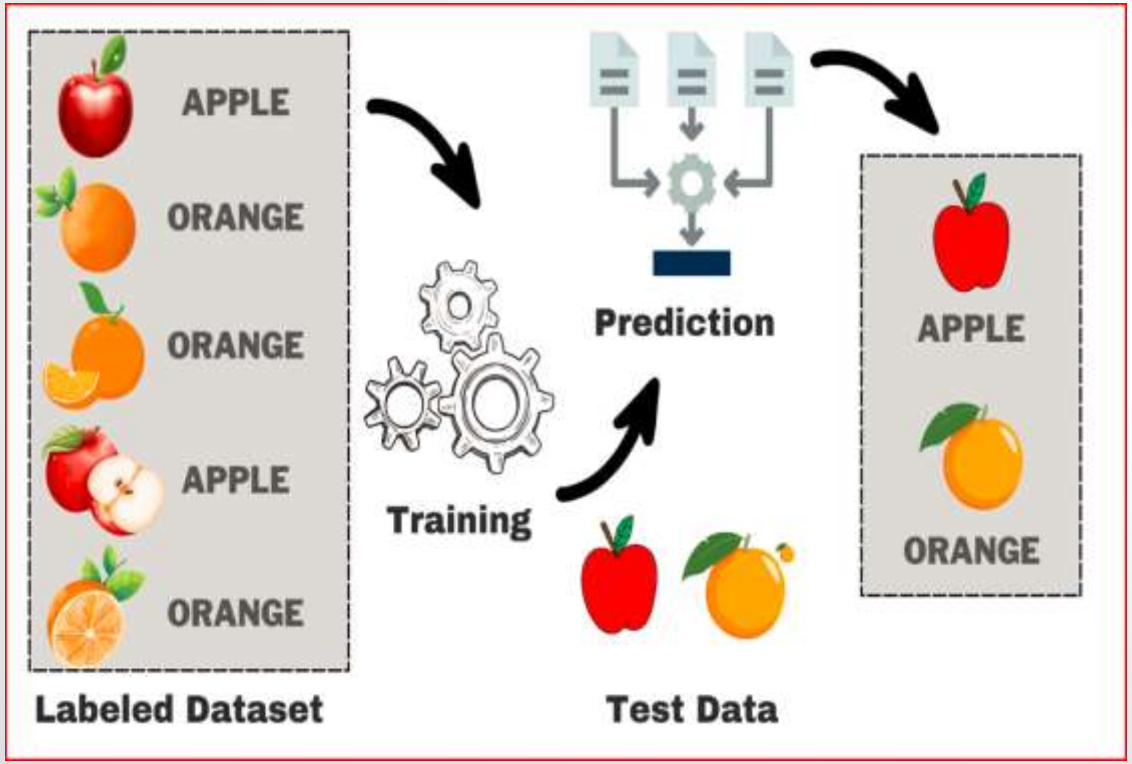
MACHINE LEARNING

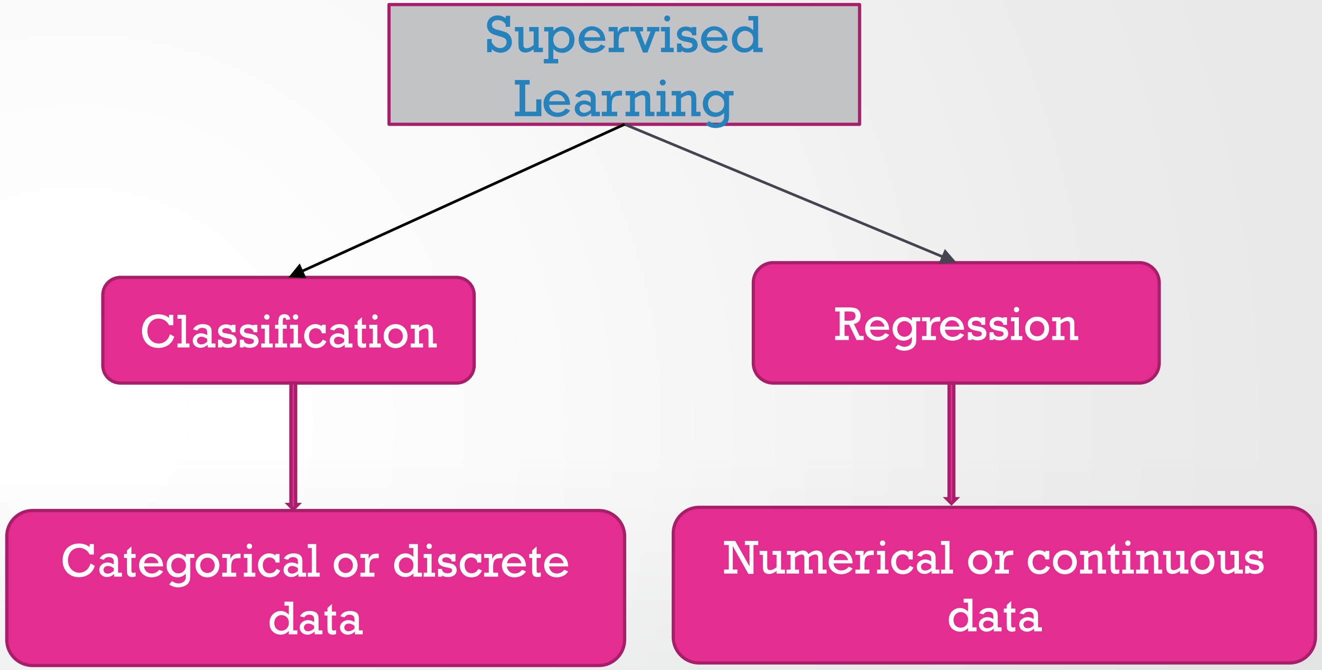


PRESENTED BY

K.SELVALAKSHMI

SUPERVISED LEARNING





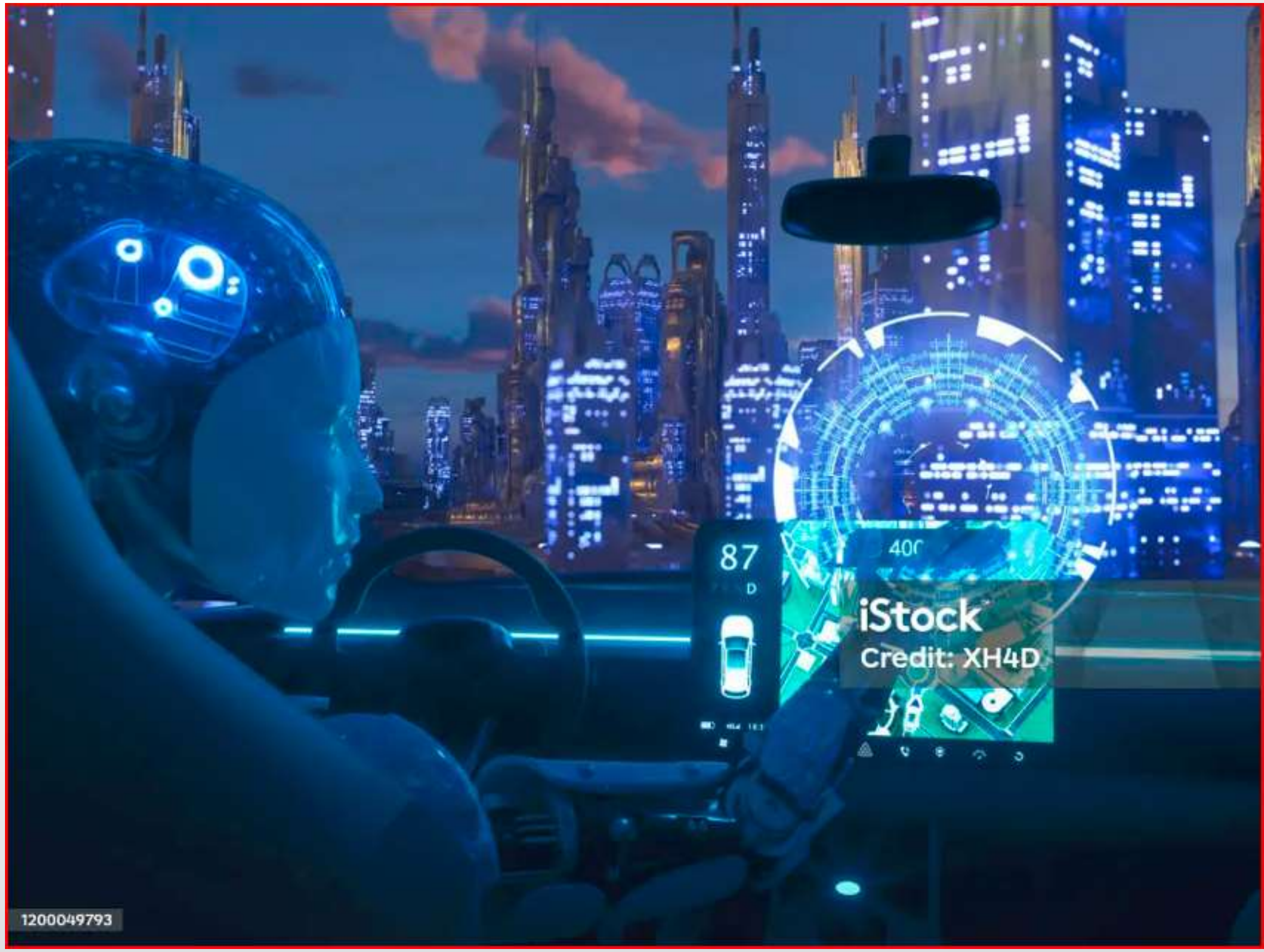
Classifier Algorithms

- Logistic Regression
- k-Nearest Neighbours
- Naive Bayes
- Decision Tree Classifier
- Support Vector Classifier
- Random Forest Classifier

Regressor Algorithms

- Linear Regression
- Ridge Regressor
- Lasso Regressor
- Decision Tree Regressor
- Support Vector Regressor
- Random Forest Regressor

HEART DISEASE AND CAR PRICE FORECASTING



DATA PREPROCESSING FOR HEART DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,mean_squared_error,r2_score
import joblib
```

df.head
df.tail
df.describe
df.info()
df.columns

```
df=pd.read_csv('C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/heart.csv')
df
```

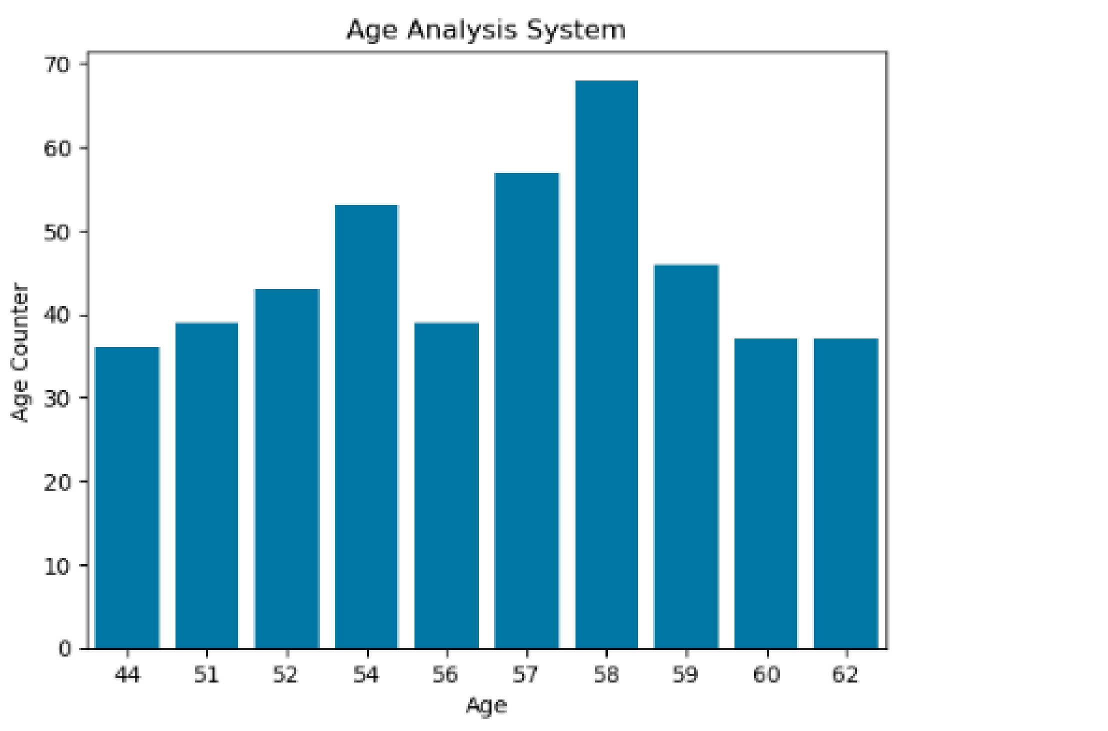
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0

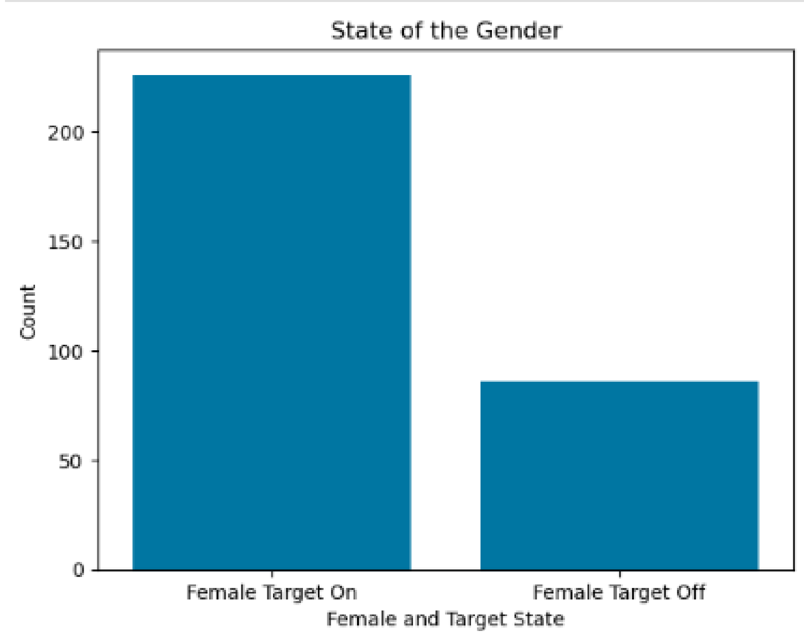
```
df.isnull().sum()
```

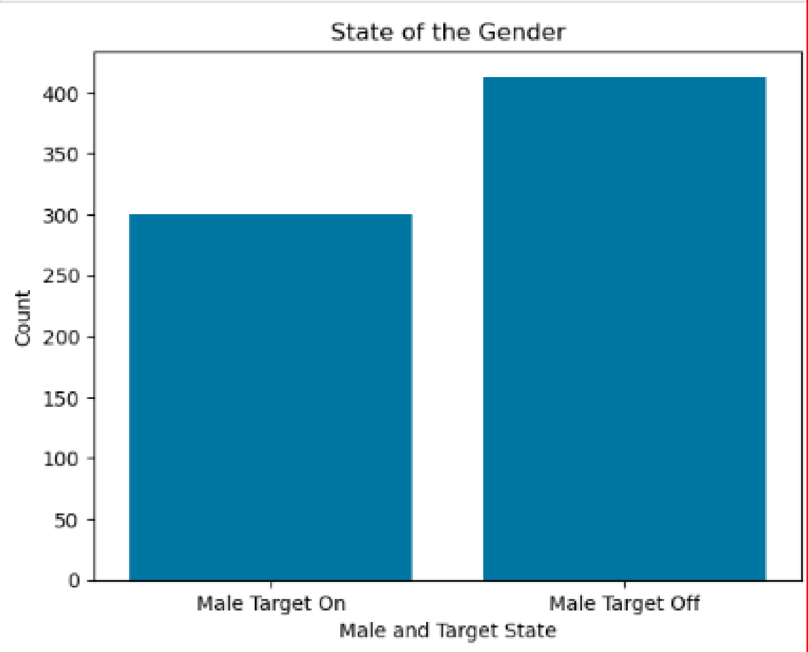
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

DATA VISUALIZATION

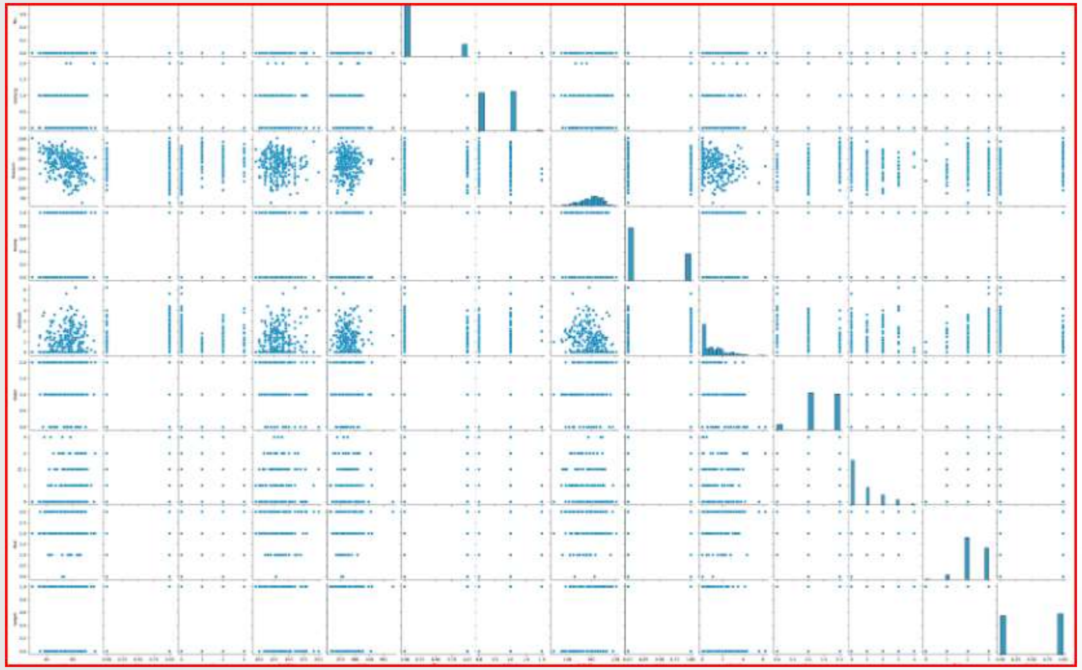
```
sns.barplot(x=df.age.value_counts()[:10].index,y=df.age.value_counts()[:10].values)
plt.xlabel('Age')
plt.ylabel('Age Counter')
plt.title('Age Analysis System')
plt.show()
```

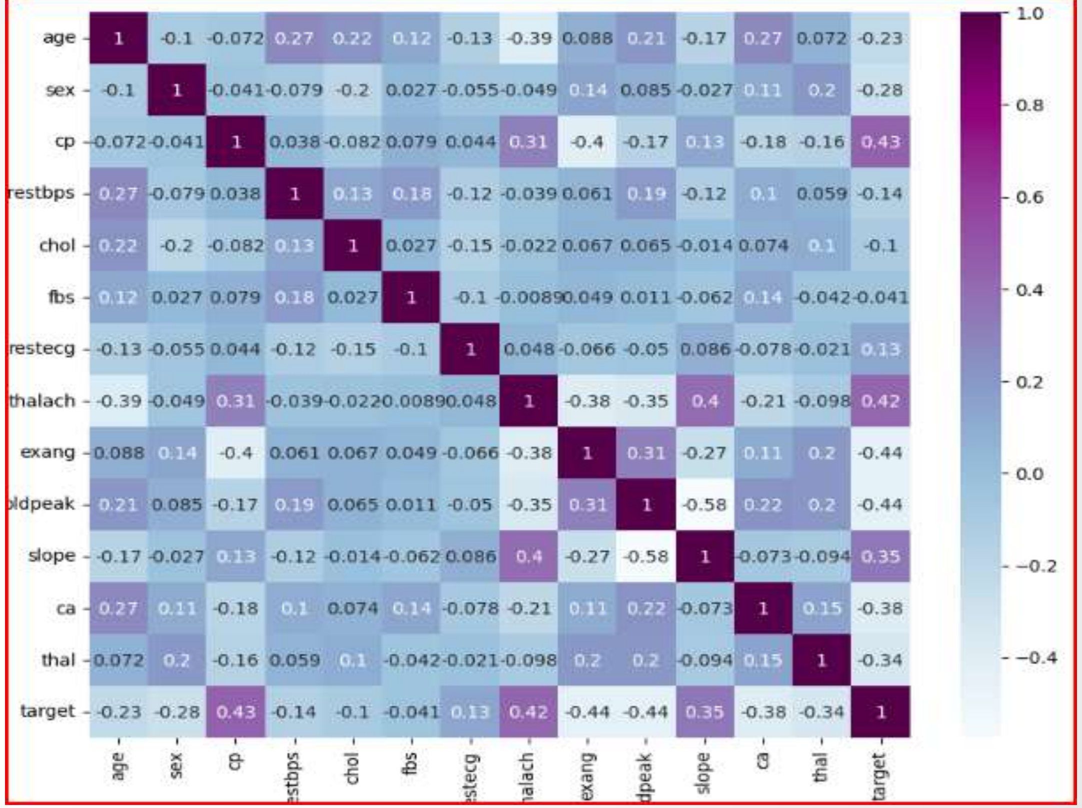


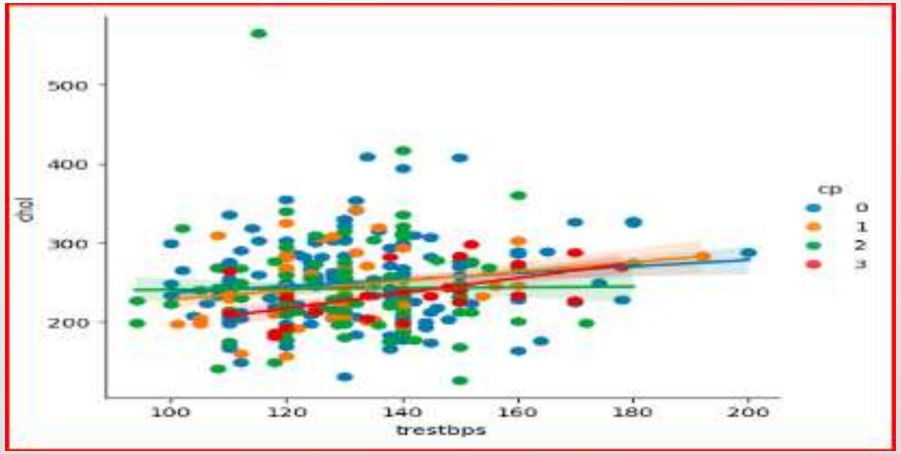




VISUALIZING CORRELATION BETWEEN VARIABLES








```
from scipy import stats
zscore=np.abs(stats.zscore(a))zscore
b=zscore>3
b=zscore<-3
```

SPLITTING DATA FOR MACHINE LEARNING: TRAIN/TEST SETS

```
x=df.iloc[:,:-1]
x
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3

025 rows × 13 columns

```
y=df.iloc[:,-1]
y
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

```
[95]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
      x_train,x_test,y_train,y_test
```

```
[95]: (      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
 835   49    1   2       118   149    0        0      126      0      0.8
 137   64    0   0       180   325    0        1      154      1      0.0
 534   54    0   2       108   267    0        0      167      0      0.0
 495   59    1   0       135   234    0        1      161      0      0.5
 244   51    1   2       125   245    1        0      166      0      2.4
 ..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
 700   41    1   2       130   214    0        0      168      0      2.0
 71    61    1   0       140   207    0        0      138      1      1.9
 106   51    1   0       140   299    0        1      173      1      1.6
 270   43    1   0       110   211    0        1      161      0      0.0
 860   52    1   0       112   230    0        1      160      0      0.0

      slope  ca  thal
 835      2   3     2
 137      2   0     2
 534      2   0     2
 495      1   0     3
 244      1   0     2
 ..     ...  ..   ...
 700      1   0     2
 71       2   1     3
 106      2   0     3
 270      2   0     3
 860      2   1     2

 [820 rows x 13 columns],
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
 527   62    0   0       124   209    0        1      163      0      0.0
 359   53    0   2       128   216    0        0      115      0      0.0
 447   55    1   0       160   289    0        0      145      1      0.8
 31    50    0   1       120   244    0        1      162      0      1.1
 621   48    1   0       130   256    1        0      150      1      0.0
 ..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
 832   68    1   2       118   277    0        1      151      0      1.0
 796   41    1   1       135   203    0        1      132      0      0.0
 644   44    1   2       120   226    0        1      169      0      0.0
 404   61    1   0       140   207    0        0      138      1      1.9
 842   58    1   2       112   230    0        0      165      0      2.5

      slope  ca  thal
 527      2   0     2
 359      2   0     0
 447      1   1     3
 31       2   0     2
 621      2   2     3
 ..     ...  ..   ...
 832      2   1     3
 796      1   0     1
 644      2   0     2
 404      2   1     3
 842      1   1     3

 [205 rows x 13 columns],
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((820, 13), (205, 13), (820,), (205,))
```

CLASSIFIERS

WITHOUT HP TUNNING

WITH HYPER PARAMETER TUNNING

```
logistc=LogisticRegression()
logistc
```

```
▾ LogisticRegression
LogisticRegression()
```

```
logistc.fit(x_train,y_train)
```

```
logistc_MSE=mean_squared_error(y_test,pred)
logistc_MSE
```

```
0.2146341463414634
```

```
performance=r2_score(y_test,pred)
performance
```

```
0.14144298496097452
```

```
accuracy_score(y_test,y_pred)
```

```
0.7853658536585366
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.70      0.76       102
           1       0.74      0.87      0.80       103

    accuracy                           0.79       205
   macro avg       0.79      0.78      0.78       205
weighted avg       0.79      0.79      0.78       205
```

```
Knn=KNeighborsClassifier()
Knn
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
Knn.fit(x_train,y_train)
```

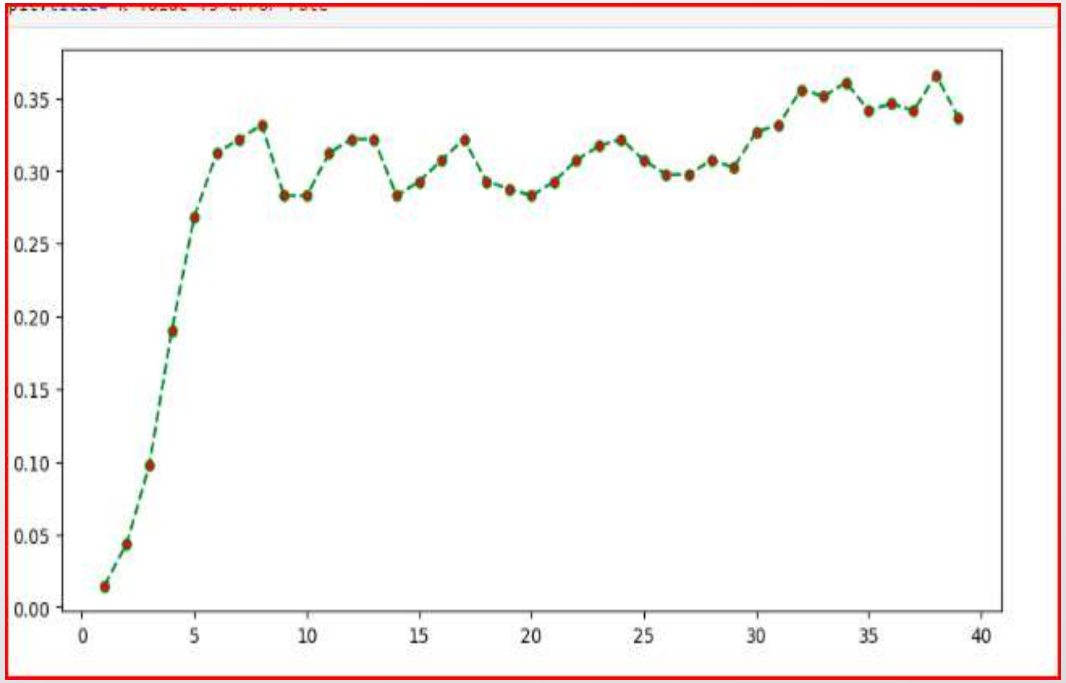
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
Knn_MSE=mean_squared_error(y_test,pred)
Knn_MSE
```

```
0.2682926829268293
```

```
Perf=r2_score(y_test,pred)
Perf
```

```
-0.07319626879878194
```



(n_neighbours=38)

```
knn=KNeighborsClassifier(n_neighbors=38)
knn.fit(x_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=38)
```

```
pr=knn.predict(x_test)
accuracy_score(y_test,pr)
```

```
0.6341463414634146
```

CLASSIFIERS

```
GNB=GaussianNB()
GNB
```

```
▾ GaussianNB
GaussianNB()
```

```
GNB.fit(x_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

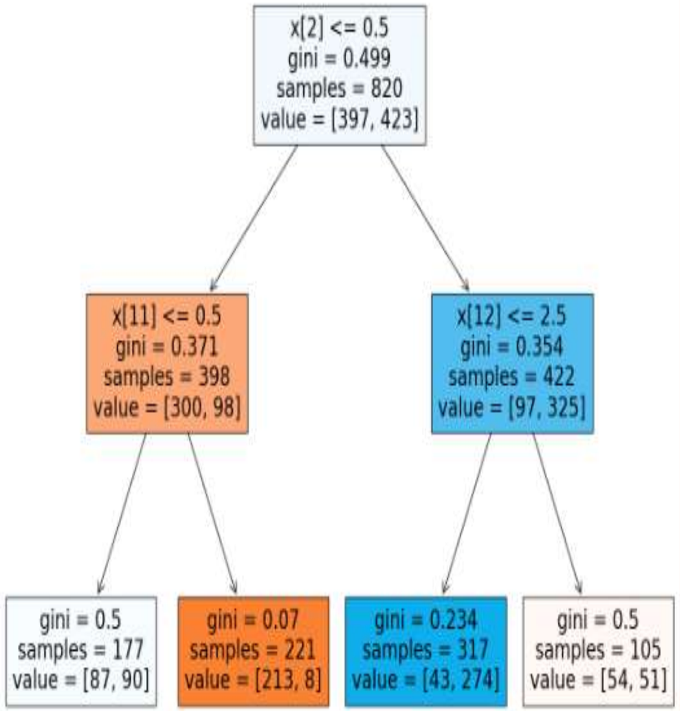
```
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.71	0.78	102
1	0.75	0.89	0.82	103
accuracy			0.80	205
macro avg	0.81	0.80	0.80	205
weighted avg	0.81	0.80	0.80	205

```
from sklearn import tree
plt.figure(figsize=(15,12))
tree.plot_tree(DT,filled=True)
```

```
[Text(0.5, 0.8333333333333334, 'x[2] <= 0.5\ngini = 0.499\nsamples = 820\nvalue = [397, 423]'),
 Text(0.25, 0.5, 'x[11] <= 0.5\ngini = 0.371\nsamples = 398\nvalue = [300, 98]'),
 Text(0.125, 0.16666666666666666, 'gini = 0.5\nsamples = 177\nvalue = [87, 90]'),
 Text(0.375, 0.16666666666666666, 'gini = 0.07\nsamples = 221\nvalue = [213, 8]'),
 Text(0.75, 0.5, 'x[12] <= 2.5\ngini = 0.354\nsamples = 422\nvalue = [97, 325]'),
 Text(0.625, 0.16666666666666666, 'gini = 0.234\nsamples = 317\nvalue = [43, 274]'),
 Text(0.875, 0.16666666666666666, 'gini = 0.5\nsamples = 105\nvalue = [54, 51]')]
```




```
DT=DecisionTreeClassifier(max_depth=2)
DT.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.71	0.60	0.65	102
1	0.66	0.76	0.70	103
accuracy			0.68	205
macro avg	0.68	0.68	0.68	205
weighted avg	0.68	0.68	0.68	205

CLASSIFIERS

WITHOUT HP TUNNING

WITH HYPER PARAMETER TUNNING

```
svm=SVC(kernel='linear',C=1)
svm
```

```
SVC
SVC(C=1, kernel='linear')
```

```
svm.fit(x_train,y_train)
```

```
SVC
SVC(C=1, kernel='linear')
```

```
accuracy=accuracy_score(y_test,pred)
print(f"Accuracy:{accuracy}")
```

```
Accuracy:0.6829268292682927
```

SVM Classification Report (With Tuning):

	precision	recall	f1-score	support
0	0.88	0.68	0.77	102
1	0.74	0.91	0.82	103
accuracy			0.80	205
macro avg	0.81	0.79	0.79	205
weighted avg	0.81	0.80	0.79	205

```
# Train the Random Forest Classifier
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

Classification Report:

	precision	recall	f1-score	support
0	0.97	1.00	0.99	102
1	1.00	0.97	0.99	103
accuracy			0.99	205
macro avg	0.99	0.99	0.99	205
weighted avg	0.99	0.99	0.99	205

Random Forest Classification Report (With Tuning):

	precision	recall	f1-score	support
0	0.97	1.00	0.99	102
1	1.00	0.97	0.99	103
accuracy			0.99	205
macro avg	0.99	0.99	0.99	205
weighted avg	0.99	0.99	0.99	205

JOBLIB

```
joblib.dump(rf,'C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/Milestone3_classifier.pkl')
```

```
['C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/Milestone3_classifier.pkl']
```

```
Load_model=joblib.load('C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/Milestone3_classifier.pkl')
Load_model
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

CONCLUSION :

The **Random Forest classifier** outperforms other models in predicting the heart dataset, achieving a remarkable accuracy of 99%.

DATA PREPROCESSING FOR CAR DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,mean_squared_error,r2_score
import joblib
import re
```

df.head
df.tail
df.describe
df.info()
df.columns

```
mode_eng=df['Engine'].mode()[0]
mode_eng
```

```
1197.0
```

```
df['Engine'].fillna(mode_eng,inplace=True)
```

```
mode_maxp=df['Max Power'].mode()[0]
mode_maxp
```

```
89.0
```

```
df['Max Power'].fillna(mode_maxp,inplace=True)
```

```
mode_maxt=df['Max Torque'].mode()[0]
mode_maxt
```

```
200.0
```

```
df['Max Torque'].fillna(mode_maxt,inplace=True)
```

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

```
Make             0
Model            0
Price            0
Year             0
Kilometer        0
Fuel Type        0
Transmission     0
Location         0
Color            0
Owner            0
Seller Type      0
Engine           0
Max Power        0
Max Torque       0
Drivetrain       0
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
```

```
Label=LabelEncoder()
```

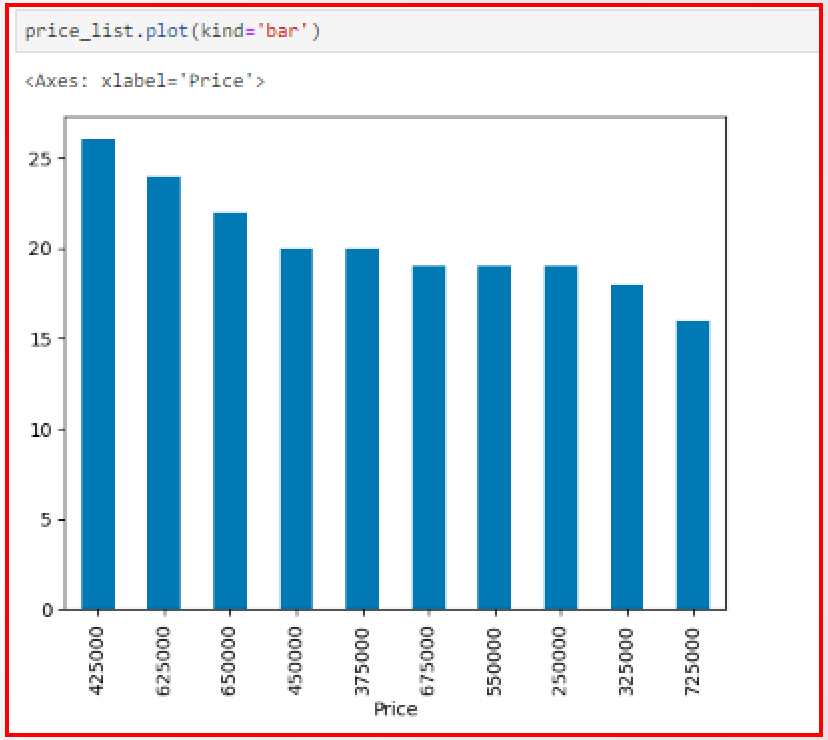
```
df['Make']=Label.fit_transform(df['Make'])
df['Model']=Label.fit_transform(df['Model'])
df['Fuel Type']=Label.fit_transform(df['Fuel Type'])
df['Transmission']=Label.fit_transform(df['Transmission'])
df['Location']=Label.fit_transform(df['Location'])
df['Color']=Label.fit_transform(df['Color'])
df['Owner']=Label.fit_transform(df['Owner'])
df['Seller Type']=Label.fit_transform(df['Seller Type'])
df['Drivetrain']=Label.fit_transform(df['Drivetrain'])
```

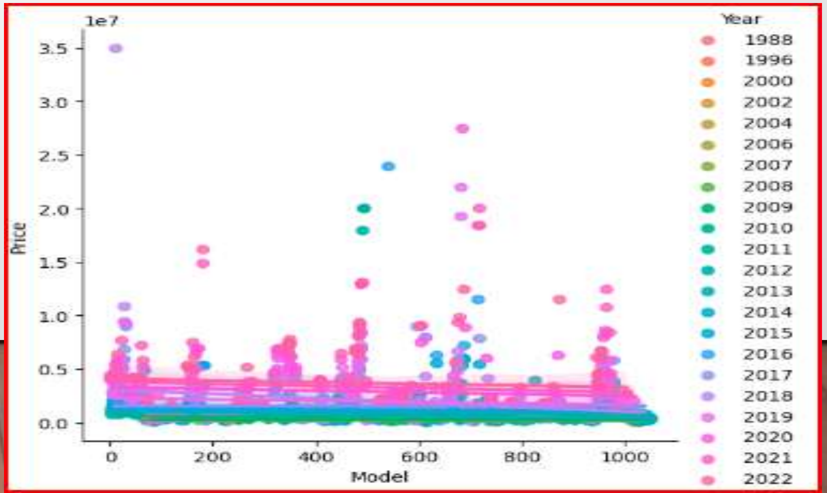
```
def get_value(val):
    if val != val:
        return val
    return float(re.split(' |@', val)[0])
df['Engine'] = df['Engine'].apply(get_value)
df['Max Power'] = df['Max Power'].apply(get_value)
df['Max Torque'] = df['Max Torque'].apply(get_value)
```

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

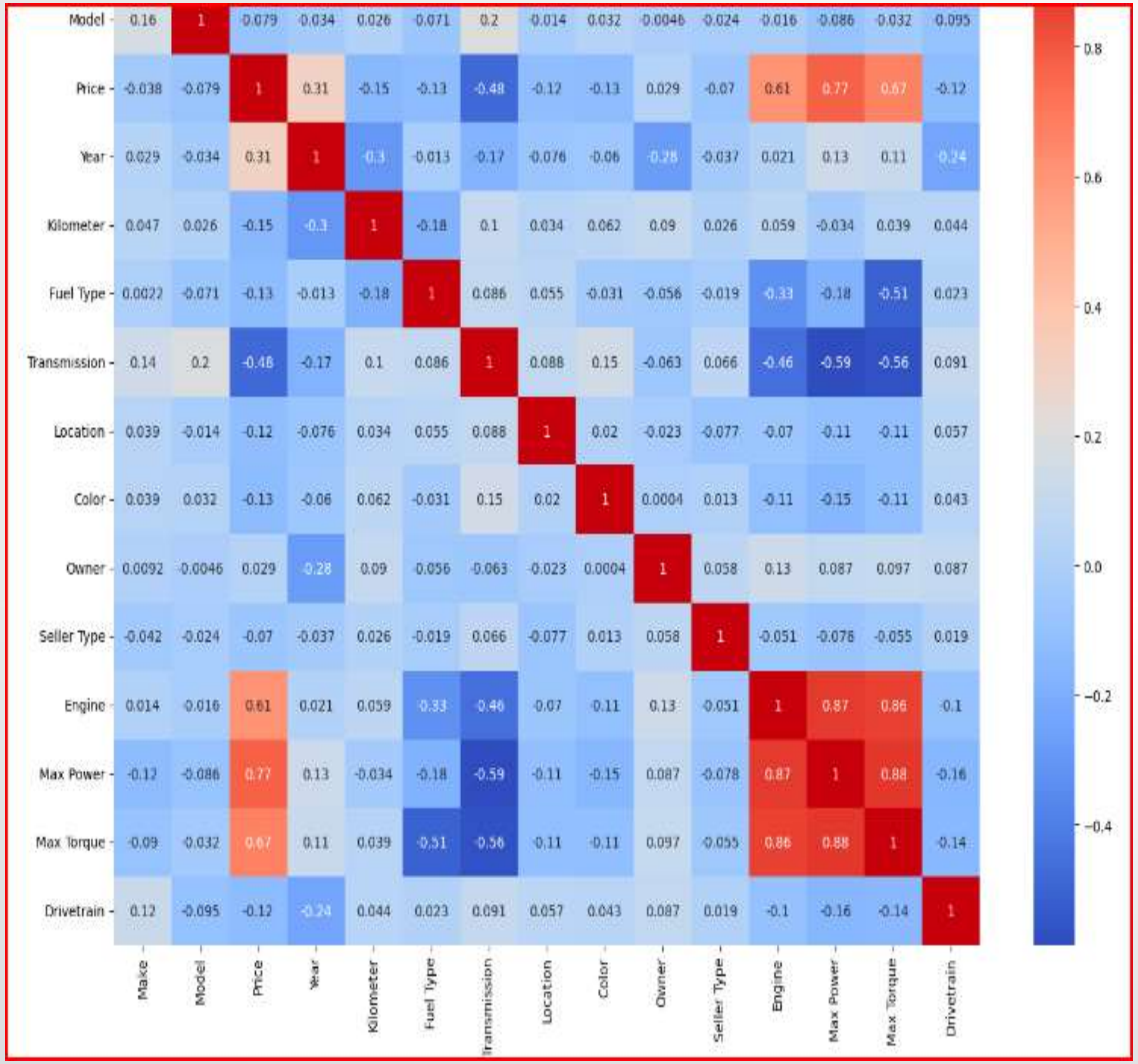
```
Make                   0
Model                  0
Price                  0
Year                   0
Kilometer              0
Fuel Type              0
Transmission           0
Location               0
Color                  0
Owner                  0
Seller Type            0
Engine                80
Max Power             80
Max Torque            80
Drivetrain             0
Length                64
Width                 64
Height                64
Seating Capacity      64
Fuel Tank Capacity   113
dtype: int64
```

DATA VISUALIZATION





HEAT MAP



IQR ANALYSIS

```
IQR=q3-q1
```

```
C=((df1<q1-1.5*IQR)|(df1>q1+1.5*IQR))
```

```
df2=df[~C.any(axis=1)]
```

```
df2
```

	Make	Model	Price	Year	Kilometer	Fuel Type	Transmission	Location	Color	Owner	Seller Type	Engine	Max Power	Max Torque	Drivetrain
2	8	1030	220000	2011	67000	6	1	39	8	1	2	1197.0	79.0	112.7619	1
5	19	216	675000	2017	73315	6	1	56	7	1	2	1373.0	91.0	130.0000	1
8	27	624	1390000	2017	56000	6	0	45	15	1	2	1798.0	177.0	250.0000	1
9	23	834	575000	2015	85000	2	1	45	15	1	2	1461.0	84.0	200.0000	1
10	8	397	591000	2017	20281	6	1	45	13	1	2	1197.0	82.0	115.0000	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2042	2	743	299000	2014	32000	6	1	14	14	1	2	1199.0	85.0	113.0000	1
2046	19	917	850000	2018	85000	2	1	27	8	1	2	1248.0	89.0	200.0000	1
2047	7	233	480000	2015	49000	6	1	36	7	1	2	1497.0	117.0	145.0000	1
2050	8	303	891000	2016	47000	6	1	15	15	1	2	1591.0	122.0	154.0000	1
2051	19	916	925000	2021	48000	6	1	6	15	1	2	1462.0	103.0	138.0000	1

809 rows × 15 columns

SPLITTING DATA FOR MACHINE LEARNING: TRAIN/TEST SETS

```
x=df2.drop(columns=['Price'])
x
```

	Make	Model	Year	Kilometer	Fuel Type	Transmission	Location	Color	Owner	Seller Type	Engine	Max Power	Max Torque	Drivetrain
2	8	1030	2011	67000	6	1	39	8	1	2	1197.0	79.0	112.7619	1
5	19	216	2017	73315	6	1	56	7	1	2	1373.0	91.0	130.0000	1
8	27	624	2017	56000	6	0	45	15	1	2	1798.0	177.0	250.0000	1
9	23	834	2015	85000	2	1	45	15	1	2	1461.0	84.0	200.0000	1
10	8	397	2017	20281	6	1	45	13	1	2	1197.0	82.0	115.0000	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2042	2	743	2014	32000	6	1	14	14	1	2	1199.0	85.0	113.0000	1
2046	19	917	2018	85000	2	1	27	8	1	2	1248.0	89.0	200.0000	1
2047	7	233	2015	49000	6	1	36	7	1	2	1497.0	117.0	145.0000	1
2050	8	303	2016	47000	6	1	15	15	1	2	1591.0	122.0	154.0000	1
2051	19	916	2021	48000	6	1	6	15	1	2	1462.0	103.0	138.0000	1

809 rows × 14 columns

```
y=df2['Price']
y
```

```
2        220000
5        675000
8       1390000
9        575000
10       591000
         ...
2042     299000
2046     850000
2047     480000
2050     891000
2051     925000
Name: Price, Length: 809, dtype: int64
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
x_train,x_test,y_train,y_test
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((647, 14), (162, 14), (647,), (162,))
```

REGRESSOR

```
linearreg=LinearRegression()
linearreg
```

```
LinearRegression
LinearRegression()
```

```
ridge=Ridge()
ridge
```

```
Ridge
Ridge()
```

```
las=Lasso()
```

```
las.fit(x_train,y_train)
```

```
Lasso
Lasso()
```

WITHOUT HP TUNNING

```
MSE=mean_squared_error(y_test,pred)
MSE
```

```
38359911771.41415
```

```
perf=r2_score(y_test,pred)
perf
```

```
0.8312597057058213
```

```
MSE=mean_squared_error(y_test,pred)
MSE
```

```
38377039270.111595
```

```
perf=r2_score(y_test,pred)
perf
```

```
0.8311843640525883
```

```
MSE=mean_squared_error(y_test,y_prediction)
MSE
```

```
38359968392.71922
```

```
performance=r2_score(y_test,y_prediction)
performance
```

```
0.831259456636018
```

WITH HYPER PARAMETER TUNNING

```
autohp=Ridge()
```

```
parameter={'alpha':[0.1, 1, 10, 100]}
```

```
Ridgehp=GridSearchCV(autohp,parameter,scoring='neg_mean_squared_error',cv=5)
```

```
print(Ridgehp.best_params_)
```

```
{'alpha': 1}
```

```
print(Ridgehp.best_score_)
```

```
-38141802822.46461
```

```
MSE=mean_squared_error(y_test,predictionhp)
MSE
```

```
38359912337.533775
```

```
performance=r2_score(y_test,predictionhp)
performance
```

```
0.8312597032155342
```

DECISION TREE,SUPPORT VECTOR AND RANDOM FOREST REGRESSOR

```
DT_regressor = DecisionTreeRegressor()
DT_regressor.fit(x_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
MSE=mean_squared_error(y_test,y_pred)
MSE
```

```
50154720975.4321
```

```
perf=r2_score(y_test,y_pred)
perf
```

```
0.779375864364121
```

```
best_model = gs.best_estimator_
y_pre = best_model.predict(x_test)
mse = mean_squared_error(y_test, y_pre)
mae = mean_absolute_error(y_test, y_pre)
r2 = r2_score(y_test, y_pre)
print(f"MSE: {mse:.2f}, MAE: {mae:.2f}, R2: {r2:.2f}")
```

```
MSE: 49792650680.33, MAE: 147157.64, R2: 0.78
```

```
svm = SVC()
svm.fit(x_train, y_train)
```

```
▾ SVC
SVC()
```

```
print(classification_report(y_test, y_pred))
```

```
      2048999       0.00      0.00      0.00         1
      2400000       0.00      0.00      0.00         1
      2425000       0.00      0.00      0.00         1

     accuracy                           0.01       162
    macro avg       0.00      0.01      0.00       162
 weighted avg       0.00      0.01      0.00       162
```

```
r2_svr = r2_score(y_test, y_pred_svr)
mse_svr = mean_squared_error(y_test, y_pred_svr)
print(f"SVR R2: {r2_svr}")
print(f"SVR MSE: {mse_svr}")
```

```
SVR R2: -0.18066232842573693
SVR MSE: 268401231251.17792
```

```
RFG=RandomForestRegressor()
RFG
```

```
RFG.fit(x_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
mse= mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Without Hyperparameter Tuning:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Without Hyperparameter Tuning:
Mean Squared Error: 23234057379.935448
R-squared: 0.897796384326939
```

```
mse=mean_squared_error(y_test,y_pred)
mse
```

```
24908558037.100098
```

```
per=r2_score(y_test,y_pred)
per
```

```
0.890430472346497
```

CONCLUSION :

The Random Forest Regressor outperforms other models in predicting the car dataset, achieving a remarkable accuracy of 89%.

JOBLIB

```
joblib.dump(RFG,'C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/Milestone3_regressor.pkl')
```

```
['C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/Milestone3_regressor.pkl']
```

```
Load_model=joblib.load('C:/Users/ADMIN/Desktop/DA andDS/Excel(ML)/Milestone3_regressor.pkl')
Load_model
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

OVERALL CONCLUSION :

Random Forest has proven to be the best algorithm for both datasets in our analysis.

Thank You