7avpxvvuh

February 17, 2024

IMPORTING LIBRARIES

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.preprocessing import PolynomialFeatures
```

LOADING DATASET

[6]:
```
df = pd.read_csv(r'/winequality-red.csv',sep=",")
df
```

[6]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…             …                …             …              …        …
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
```

```
1                    25.0                 67.0  0.99680  3.20       0.68
2                    15.0                 54.0  0.99700  3.26       0.65
3                    17.0                 60.0  0.99800  3.16       0.58
4                    11.0                 34.0  0.99780  3.51       0.56
…                    …                    …     …        …          …
1594                 32.0                 44.0  0.99490  3.45       0.58
1595                 39.0                 51.0  0.99512  3.52       0.76
1596                 29.0                 40.0  0.99574  3.42       0.75
1597                 32.0                 44.0  0.99547  3.57       0.71
1598                 18.0                 42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
…         …        …
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]
```

SUMMARIZING THE DATA

[7]:
```
df.info() #no na values present
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
```

```
memory usage: 150.0 KB
```

[8]:
```python
df.describe()
```

[8]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

PART A The summary of attributes is shown above,It can be observed that 1.there are no null values in the given dataset 2.All the values are continuous (float) and quality which is our target variable is int (discrete) 3.No categorical values are present in the dataset

[9]:
```python
wine_df=df.drop(['quality'],axis=1)
```

[10]:
```python
y=df['quality']
```

[11]:
```python
wine_df
```

[11]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
```

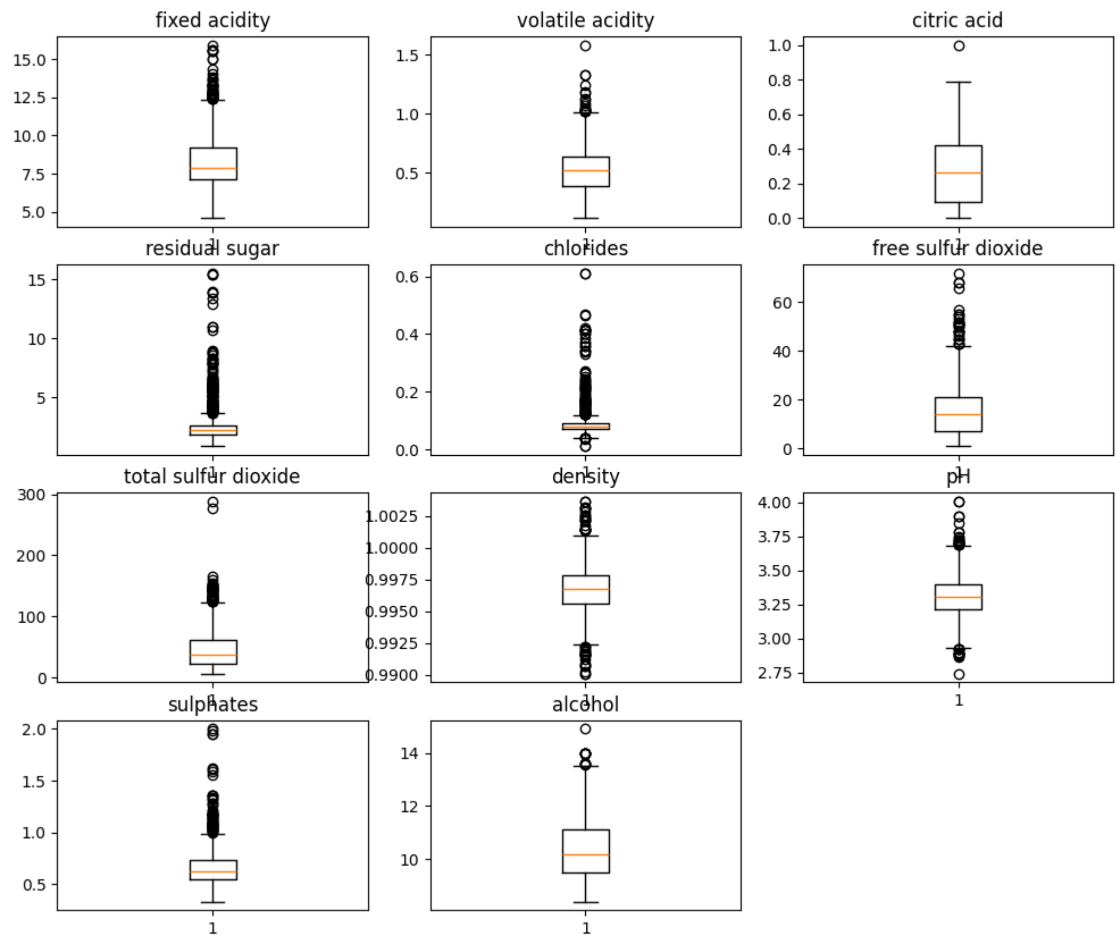
```
3           11.2           0.280      0.56            1.9      0.075
4            7.4           0.700      0.00            1.9      0.076
…             …              …          …             …         …
1594         6.2           0.600      0.08            2.0      0.090
1595         5.9           0.550      0.10            2.2      0.062
1596         6.3           0.510      0.13            2.3      0.076
1597         5.9           0.645      0.12            2.0      0.075
1598         6.0           0.310      0.47            3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                     …                   …       …      …         …
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol
0         9.4
1         9.8
2         9.8
3         9.8
4         9.4
…        …
1594     10.5
1595     11.2
1596     11.0
1597     10.2
1598     11.0

[1599 rows x 11 columns]
```

[12]:
```
y.value_counts()  #counting unique values of target variable shows imbalance
```

[12]:
```
quality
5    681
6    638
7    199
4     53
8     18
3     10
Name: count, dtype: int64
```

[13]:
```
#applying boxcox transformation to the target variable for normalizing
↪distribution
from scipy.stats import boxcox
y_transformed, lambda_value = boxcox(y)
y1 = pd.Series(y_transformed)
```

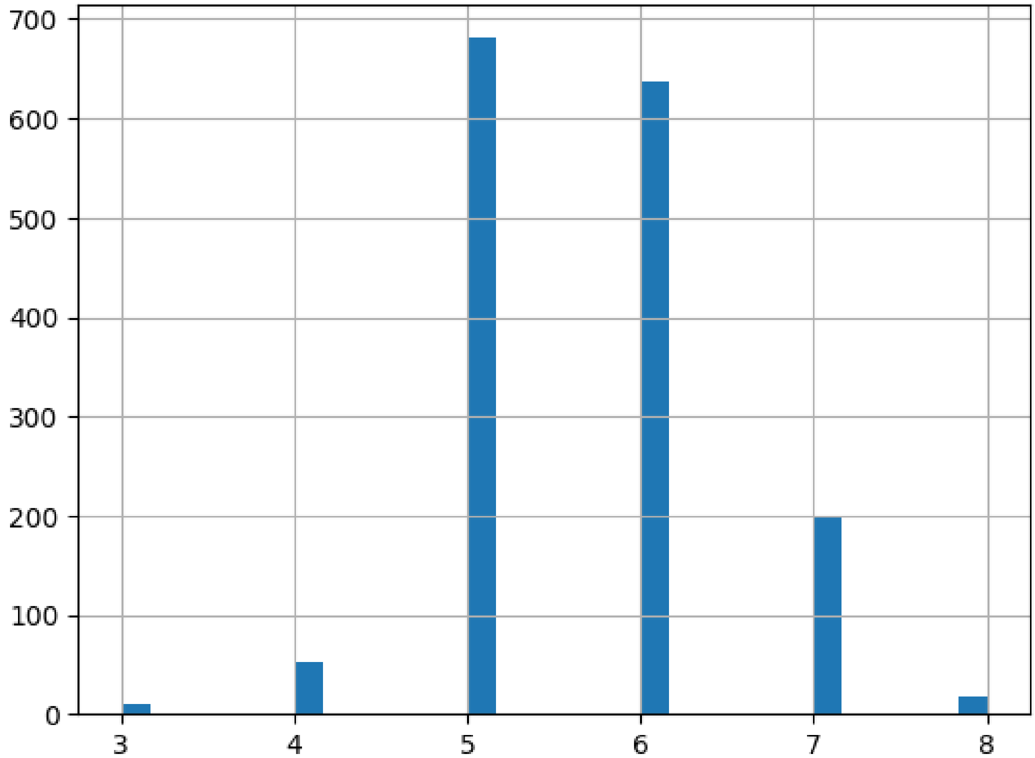
[14]:
```
#boxplot shows the outliers and data distribution of each attribute

%matplotlib inline
plt.figure(figsize=(12,10))
for i,c in enumerate(wine_df.columns):
  if i<=11:
    plt.subplot(4,3,i+1)
    plt.title(c)
    plt.boxplot(wine_df[c])
plt.show()
```



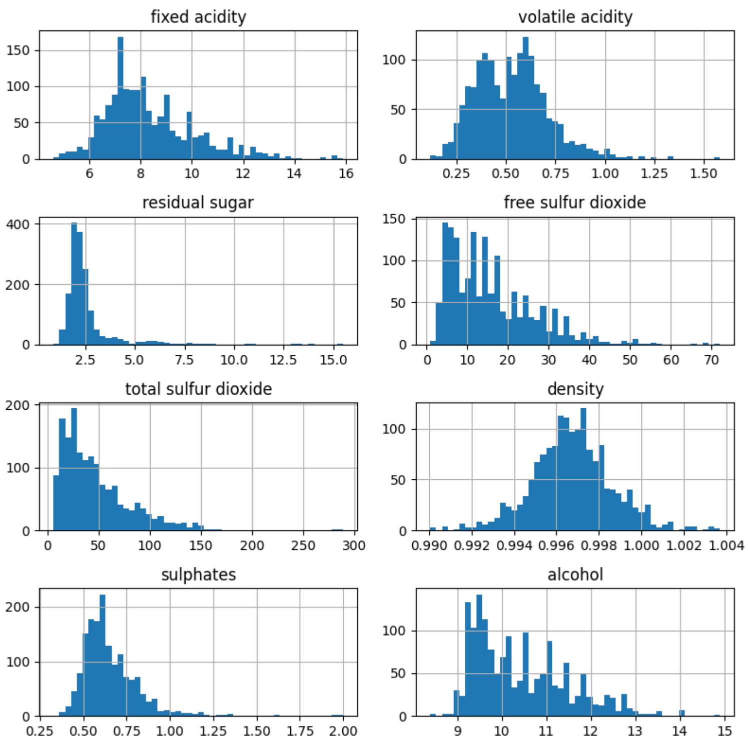
HISTOGRAM

[15]:
```
df['quality'].hist(bins=30)
```

[15]: <Axes: >



[16]:
```
wine_df[['fixed acidity','volatile acidity','citric acid','residual␣
↪sugar','free sulfur dioxide','chlorides','total sulfur␣
↪dioxide','density','pH','sulphates','alcohol']].hist(bins=50, figsize=(12,␣
↪8), layout=(4, 3))
plt.tight_layout()
plt.show()
```



PART B Histogram of various attributes including the target variable depicting the distribution of data From the figure most of the data is uniformly distributed , the target variable has uneven distribution of target variable since all the attributes are continuous valued with no null values they dont require any special treatment Outliers were not removed to prevent data loss and they tell they provide the true nature of dataset

SCATTER PLOT

[ ]:
```
sns.pairplot(df,kind='scatter')

# Show the plot
plt.show()
```

[ ]:
```
corr=df.corr(method ='pearson',numeric_only=True)
f, ax = plt.subplots(figsize=(9, 8))
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True,ax=ax)
```

[ ]:
```
corr
```

[ ]:
```
corr["quality"].sort_values(ascending=False) #correlation of target variable with the attributes
```

```
[ ]: #wine_df=wine_df.drop(['residual sugar'],axis=1)
```

PART C scatter plots showing relationship between various attributes which are generated using pearson coefficients.The above table data shows that residual sugar is least correlated with the target variable i.e 0.01 while attributes such as alcohol and volatile acidity contributes the most towards determining wine quality

SPLITTING DATASET FOR TRAINING

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(wine_df, y1, test_size=0.2,␣
      ↪random_state=42)
```

```
[ ]: X_test.describe()
```

```
[ ]: wine_df.describe()
```

PART D Comparing the statistical values of the dataset and the test set we can observe that the test set represents the entire dataset.The dataset is splitted using random sampling which ensures that the random split is reproducible and that the test set is representative of the entire dataset

PART E PART 1 TRAINING MODEL USING NORMAL EQUATION AND K FOLD CROSS VALIDATION

```
[ ]: kf = KFold(shuffle=True,n_splits=4,random_state=42)
     rmse_values = []
     r2_values = []

     for train_index, test_index in kf.split(wine_df):
         X_train, X_test = wine_df.iloc[train_index], wine_df.iloc[test_index]
         y_train, y_test = y1.iloc[train_index], y1.iloc[test_index]

         model = LinearRegression()
         model.fit(X_train, y_train)

         y_pred = model.predict(X_test)

         rmse = np.sqrt(mean_squared_error(y_test, y_pred))
         rmse_values.append(rmse)

         r2 = r2_score(y_test, y_pred)
         r2_values.append(r2)
         average_rmse = np.mean(rmse_values)
     average_r2 = np.mean(r2_values)

     print("Average RMSE:", average_rmse)
     print("Average R^2:", average_r2)
```

SGD REGRESSOR

[ ]:
```
model_sgd = SGDRegressor(max_iter=100, tol=1e-3, penalty='l2', alpha = 0.1)
tloss=[]
vloss=[]

for i in range(100):
        model_sgd.partial_fit(X_train, y_train)
        tloss.append(mean_squared_error(y_train, model_sgd.predict(X_train)))
        vloss.append(mean_squared_error(y_test, model_sgd.predict(X_test)))

plt.figure(figsize=(10,6))
plt.plot(tloss, label='Training Loss', marker='o')
plt.plot(vloss, label='Validation Loss')
plt.legend()
plt.show()
```

From the graph above it is evident that the model overfits the data depicted by the high validation loss and very less training loss.

[ ]:
```
#DROPPING ONE FEATURE TO CHECK IF IT IMPROVES THE MODEL PERFORMANCE AND␣
↪RESIDUAL SUGAR HAS LEAST CORRELATION COEFFICIENT
wine_df1 = wine_df.drop(['residual sugar'],axis=1)
X_train, X_test, y_train, y_test = train_test_split(wine_df1, y, test_size=0.2,␣
↪random_state=42)
rmse_values = []
r2=[]
for train_index, test_index in kf.split(wine_df1):
    X_train, X_test = wine_df1.iloc[train_index], wine_df1.iloc[test_index]
    y_train, y_test = y1.iloc[train_index], y1.iloc[test_index]
    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    rmse_values.append(rmse)
    r2=r2_score(y_test, y_pred)

average_rmse = np.mean(rmse_values)
print(average_rmse)
r2
```

[ ]:
```
model = SGDRegressor(max_iter=100, tol=1e-3, penalty='l2', alpha = 0.1)
tloss=[]
vloss=[]

for i in range(100):
    model.partial_fit(X_train, y_train)
    tloss.append(mean_squared_error(y_train, model.predict(X_train)))
```

```
    vloss.append(mean_squared_error(y_test, model.predict(X_test)))

plt.figure(figsize=(10,6))
plt.plot(tloss, label='Training Loss', marker='o')
plt.plot(vloss, label='Validation Loss')
plt.legend()
plt.show()
```

PART E RIDGE LASSO AND ELASTIC NET WITH DIFFERENT ALPHA VALUES

[ ]:
```
alphas = [0.1, 1.0, 10.0]  # Example alpha values
```

[ ]:
```
# Train Ridge regression with different alpha values
for alpha in alphas:
    ridge = Ridge(alpha=alpha,max_iter=1000)
    ridge.fit(X_train, y_train)
    y_pred_ridge = ridge.predict(X_test)
    mse_ridge = mean_squared_error(y_test, y_pred_ridge)
    print(f"Mean Squared Error (Ridge) with alpha={alpha}: {mse_ridge}")
```

[ ]:
```
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    y_pred_lasso = lasso.predict(X_test)
    mse_lasso = mean_squared_error(y_test, y_pred_lasso)
    print(f"Mean Squared Error (Lasso) with alpha={alpha}: {mse_lasso}")
```

[ ]:
```
elastic_net = ElasticNetCV(cv=3) #evaluating elasticnet using 4-fold cross␣
 ↪validation
elastic_net.fit(X_train, y_train)
y_pred_elastic_net = elastic_net.predict(X_test)
mse_elastic_net = mean_squared_error(y_test, y_pred_elastic_net)
print("Mean Squared Error (Elastic Net):", mse_elastic_net)
```

PART E the lasso,ridge and elastic net regularization with different alpha values are given. RIDGE–>These MSE values represent the model's performance in terms of prediction accuracy for the respective alpha values. In this case, ridge with alpha=0.1 has the lowest MSE, suggesting that it provides the most accurate predictions among the three cases of alpha. Lasso with the learning rate of 0.1 also provides the least MSE among all the other values of alpha. For the above dataset Elastic net or ridge is best for regularization

POLYNOMIAL REGRESSION

[ ]:
```
degree = 4
k = 4

train_errors = []
val_errors = []
```

```
kf = KFold(n_splits=k, shuffle=True)

for train_idx, val_idx in kf.split(wine_df):

    X_train, X_val = wine_df.iloc[train_idx], wine_df.iloc[val_idx]
    y_train, y_val = y1.iloc[train_idx], y1.iloc[val_idx]

    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    X_val_poly = poly.transform(X_val)

    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    y_train_pred = model.predict(X_train_poly)
    y_val_pred = model.predict(X_val_poly)

    train_mse = mean_squared_error(y_train, y_train_pred)
    val_mse = mean_squared_error(y_val, y_val_pred)
    r2=r2_score(y_val, y_val_pred)

    train_errors.append(train_mse)
    val_errors.append(val_mse)

plt.plot(range(1, k+1), train_errors, label='Train')
plt.plot(range(1, k+1), val_errors, label='Validation')
plt.xlabel('Fold')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

[ ]:
```
model_sgd = SGDRegressor(max_iter=100, tol=1e-3, penalty='l2', alpha = 0.1)
tloss=[]
vloss=[]

for i in range(100):
        model_sgd.partial_fit(X_train, y_train)
        tloss.append(mean_squared_error(y_train, model_sgd.predict(X_train)))
        vloss.append(mean_squared_error(y_test, model_sgd.predict(X_test)))

plt.figure(figsize=(10,6))
plt.plot(tloss, label='Training Loss', marker='o')
plt.plot(vloss, label='Validation Loss')
plt.legend()
plt.show()
```

The above polynomial model overfits the data which is depicted by the high validation loss and low training loss for both closed form and SGD method

RIDGE LASSO AND ELASTICNET FOR POLYNOMIAL REGRESSION

[ ]:
```
for alpha in alphas:
    ridge = Ridge(alpha=alpha,max_iter=1000)
    ridge.fit(X_train, y_train)
    y_pred_ridge = ridge.predict(X_test)
    mse_ridge = mean_squared_error(y_test, y_pred_ridge)
    print(f"Mean Squared Error (Ridge) with alpha={alpha}: {mse_ridge}")
```

[ ]:
```
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    y_pred_lasso = lasso.predict(X_test)
    mse_lasso = mean_squared_error(y_test, y_pred_lasso)
    print(f"Mean Squared Error (Lasso) with alpha={alpha}: {mse_lasso}")
```

[ ]:
```
elastic_net = ElasticNetCV(cv=3) #evaluating elasticnet using 4-fold cross␣
 ↪validation
elastic_net.fit(X_train, y_train)
y_pred_elastic_net = elastic_net.predict(X_test)
mse_elastic_net = mean_squared_error(y_test, y_pred_elastic_net)
print("Mean Squared Error (Elastic Net):", mse_elastic_net)
```

the lasso,ridge and elastic net regularization with different alpha values are given. RIDGE–>These MSE values represent the model's performance in terms of prediction accuracy for the respective alpha values. In this case, ridge with alpha=0.1 has the lowest MSE, suggesting that it provides the most accurate predictions among the three cases of alpha. Lasso with the learning rate of 0.1 also provides the least MSE among all the other values of alpha. For the above dataset Elastic net or ridge is best for regularization

PART G PREDICTION ON TEST LABELS

[ ]:
```
ridge = Ridge(alpha=0.1,max_iter=1000)
ridge.fit(X_train,y_train)
y_pred_ridge = ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2=r2_score(y_test, y_pred_ridge)
print("MSE VALUE:",mse_ridge)
print("R2 VALUE:",r2)
```

The best model for regression on the above dataset is using the ridge regularizaion with alpha value=0.1 which reduces the MSE value to 0.1034389407556793.MSE is the best evaluation metric for the regression model above Future scope could include improving the R2 value of the model and increasing the dataset to prevent overfitting and evaluating more parameters using grid search or randomized search

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```