What is regularization:

- Regularization refers to a set of different techniques that lower the complexity of a neural network model during training,
  and thus prevent the overfitting.

Lasso:

- Least Absolute Shrinkage and Selection Operator
- If a model uses the L1 regularization technique, then it is called lasso regression.
- Adds the “absolute value of magnitude” of the coefficient as a penalty term to the loss function
- Also performs automatic feature selection

Ridge:

- Ridge regression is also referred to as L2 Regularization.
- Adds the "squared magnitude" of the coefficient as the penalty term to the loss function.

Type *Markdown* and LaTeX: α^2

Importing Packages

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from scipy import stats
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

Storing the dataset

In [2]:

```
df_train = pd.read_csv(r"C:\Users\hemaachandhan\OneDrive\Desktop\Notebooks\AML-lab\AML-1
```

In [3]:

```
df_test = pd.read_csv(r"C:\Users\hemaachandhan\OneDrive\Desktop\Notebooks\AML-lab\AML-1\
```

In [ ]:

```

```

Basic Analysis

In [4]:

```
df_train.head()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl

5 rows × 81 columns

In [5]:

```
df_train.shape
```

Out[5]:

```
(1460, 81)
```

In [6]:

```
df_test.shape
```

Out[6]:

```
(1459, 80)
```

In [7]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
```

```
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
 58  GarageType     1379 non-null   object 
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object 
 61  GarageCars     1460 non-null   int64  
 62  GarageArea     1460 non-null   int64  
 63  GarageQual     1379 non-null   object 
 64  GarageCond     1379 non-null   object 
 65  PavedDrive     1460 non-null   object 
 66  WoodDeckSF     1460 non-null   int64  
 67  OpenPorchSF    1460 non-null   int64  
 68  EnclosedPorch  1460 non-null   int64  
 69  3SsnPorch      1460 non-null   int64  
 70  ScreenPorch    1460 non-null   int64  
 71  PoolArea       1460 non-null   int64  
 72  PoolQC         7 non-null      object 
 73  Fence          281 non-null    object 
 74  MiscFeature    54 non-null     object 
 75  MiscVal        1460 non-null   int64  
 76  MoSold         1460 non-null   int64  
 77  YrSold         1460 non-null   int64  
 78  SaleType       1460 non-null   object 
 79  SaleCondition  1460 non-null   object 
 80  SalePrice      1460 non-null   int64  
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [8]:

```
df_train.dtypes.value_counts()
```

Out[8]:

```
object     43
int64      35
float64     3
dtype: int64
```

In [9]:

```
for i in df_train.columns:
    if df_train[i].isnull().any() == True:
        print(i,end = " ")
        print(df_train.loc[lambda df_train : df_train[i].isnull() == True].shape[0])
```

```
LotFrontage 259
Alley 1369
MasVnrType 8
MasVnrArea 8
BsmtQual 37
BsmtCond 37
BsmtExposure 38
BsmtFinType1 37
BsmtFinType2 38
Electrical 1
FireplaceQu 690
GarageType 81
GarageYrBlt 81
GarageFinish 81
GarageQual 81
GarageCond 81
PoolQC 1453
Fence 1179
MiscFeature 1406
```

In [10]:

```
df_train["Electrical"].fillna(df_train["Electrical"].mode()[0],inplace = True)
```

In [11]:

```
df_test.columns
```

Out[11]:

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Stree
t',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgTyp
e',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemo
dAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrTyp
e',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heatin
g',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'Full
Bath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'Garage
Type',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'Garage
Qual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition'],
      dtype='object')
```

In [12]:

```
for i in df_test.columns:
    if df_train[i].isnull().sum() > 500:
        print("Dropped Columns: ",i)
        df_train.drop(columns = [i], axis=1, inplace=True)
        df_test.drop(columns= [i], axis=1, inplace = True)
```

```
Dropped Columns:  Alley
Dropped Columns:  FireplaceQu
Dropped Columns:  PoolQC
Dropped Columns:  Fence
Dropped Columns:  MiscFeature
```

In [13]:

```
df_train.shape
```

Out[13]:

```
(1460, 76)
```

In [ ]:

Encoding

In [14]:

```
numeric_feature = df_train.select_dtypes(include=["int64","float64"]).columns
```

In [15]:

```
numeric_feature
```

Out[15]:

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFin
SF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'Full
Bath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeck
SF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolA
rea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

In [16]:

```
categorical_feature = df_train.select_dtypes(exclude=["int64","float64"]).columns
```

In [17]:

```
categorical_feature
```

Out[17]:

```
Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition
2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundatio
n',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinTy
pe2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageC
ond',
       'PavedDrive', 'SaleType', 'SaleCondition'],
      dtype='object')
```

In [18]:

```
Ord_encoder = OrdinalEncoder()
```

In [19]:

```
for i in categorical_feature:
    df_train[[i]] = Ord_encoder.fit_transform(df_train[[i]])
    df_test[[i]] = Ord_encoder.fit_transform(df_test[[i]])
```

In [20]:

```
df_train.dtypes.value_counts()
```

Out[20]:

```
float64    41
int64      35
dtype: int64
```

In [21]:

```
df_test.dtypes.value_counts()
```

Out[21]:

```
float64    49
int64      26
dtype: int64
```

No Objects...

In [ ]:

```

```

In [ ]:

```

```

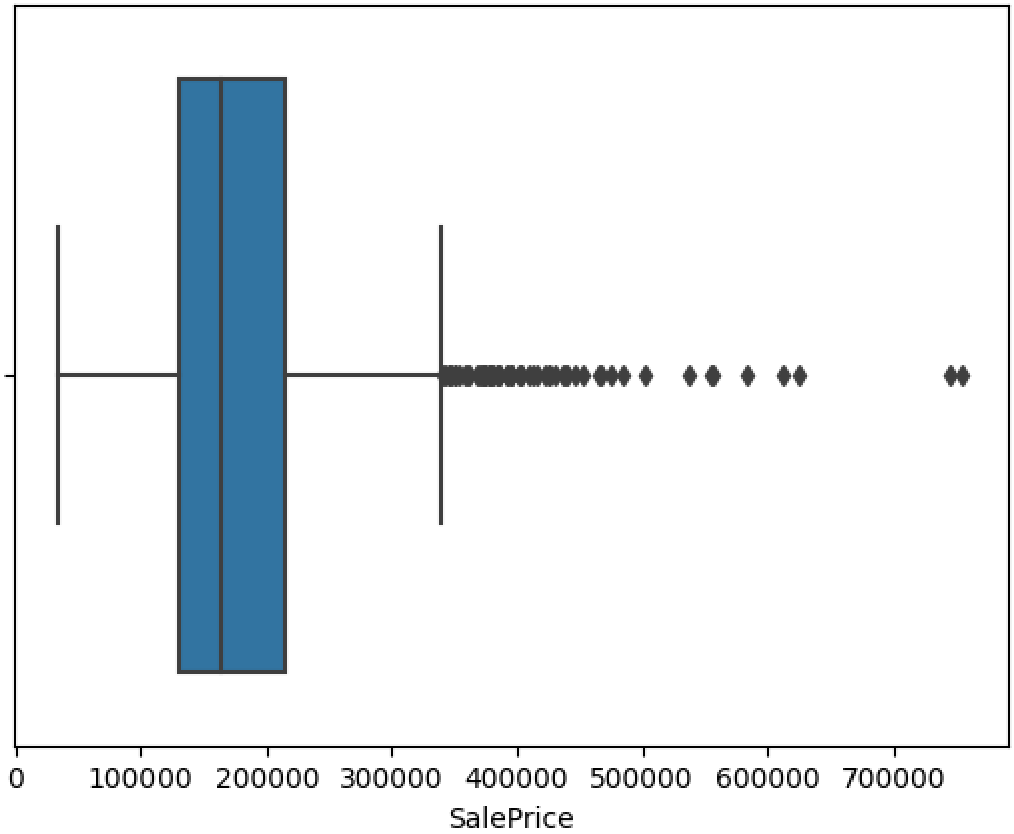
Outlier Analysis & Removal

In [22]:

```
sns.boxplot(x= df_train.SalePrice)
```

Out[22]:

```
<AxesSubplot: xlabel='SalePrice'>
```



In [23]:

```
df_train.SalePrice.quantile([.25,.5,.75])
```

Out[23]:

```
0.25    129975.0
0.50    163000.0
0.75    214000.0
Name: SalePrice, dtype: float64
```

In [24]:

```
z = np.abs(stats.zscore(df_train.SalePrice))
```

In [25]:

```
df_train.drop(np.where(z>3)[0],axis = 0,inplace = True)
```

In [26]:

```
df_train.shape
```

Out[26]:

```
(1438, 76)
```

In [27]:

```
df_train.fillna(method = "bfill", inplace = True )
```

In [28]:

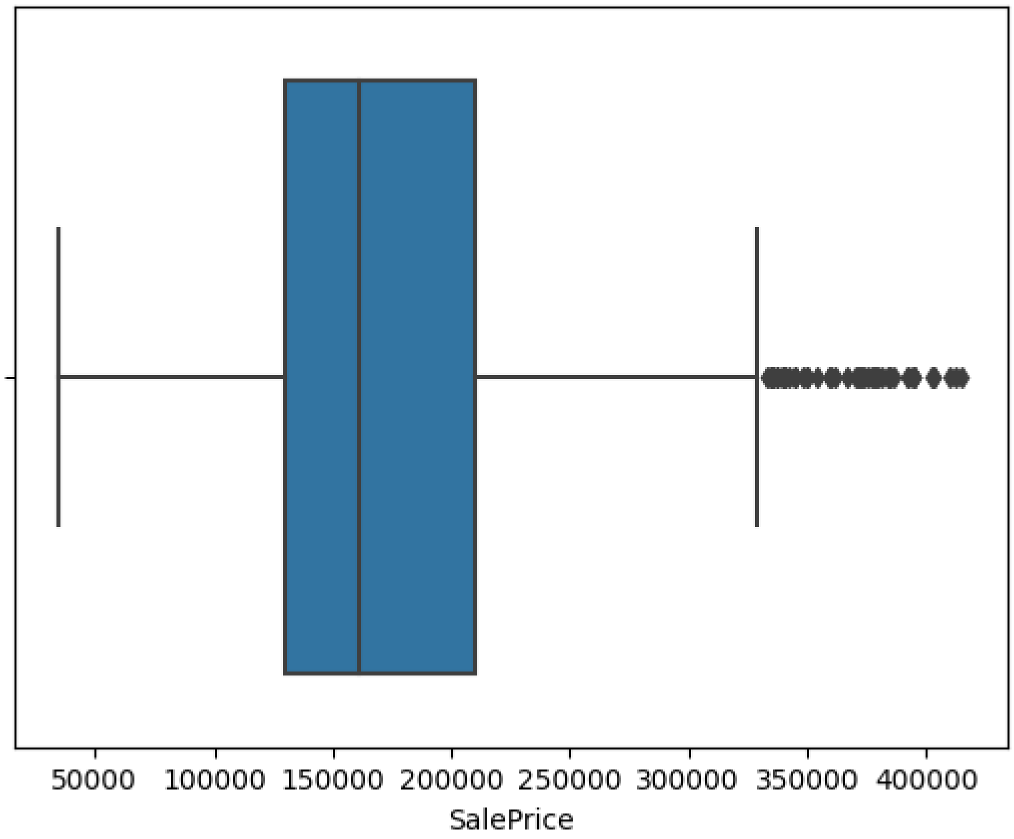
```
df_test.fillna(method = "bfill", inplace = True )
```

In [29]:

```
sns.boxplot(x= df_train.SalePrice)
```

Out[29]:

```
<AxesSubplot: xlabel='SalePrice'>
```



In [30]:

```
df_train.SalePrice.quantile([.25,.5,.75])
```

Out[30]:

```
0.25    129500.0
0.50    161500.0
0.75    210000.0
Name: SalePrice, dtype: float64
```

In [ ]:

```
```

In [31]:

```
# Q1=df_train.SalePrice.quantile(.25)
# Q2=df_train.SalePrice.quantile(0.5)
# Q3=df_train.SalePrice.quantile(0.75)
```

In [32]:

```
# IQR = Q3-Q1
# IQR
```

In [33]:

```
# lower = Q1-1.5*IQR
# upper = Q3+1.5*IQR
```

In [34]:

```
# print(lower,"and",upper)
```

In [35]:

```
# df_train[(df_train.SalePrice>lower) & (df_train.SalePrice<upper)].shape[0]
```

In [ ]:

```
```

Type *Markdown* and LaTeX: α^2

Splitting of data

In [36]:

```
X = df_train.drop("SalePrice",axis = 1)
```

In [37]:

```
Y = df_train.SalePrice
```

In [ ]:

```

```

Important Feature Selection

In [38]:

```
fs = SelectKBest(score_func=f_regression, k=50)
```

In [39]:

```
X.fillna(method = "bfill", inplace = True )
```

In [40]:

```
X_selected = fs.fit_transform(X, Y)
```

In [41]:

```
X_selected
```

Out[41]:

```
array([[3.000e+00, 6.500e+01, 8.450e+03, ..., 6.100e+01, 0.000e+00,
        4.000e+00],
       [3.000e+00, 8.000e+01, 9.600e+03, ..., 0.000e+00, 0.000e+00,
        4.000e+00],
       [3.000e+00, 6.800e+01, 1.125e+04, ..., 4.200e+01, 0.000e+00,
        4.000e+00],
       ...,
       [3.000e+00, 6.600e+01, 9.042e+03, ..., 6.000e+01, 0.000e+00,
        4.000e+00],
       [3.000e+00, 6.800e+01, 9.717e+03, ..., 0.000e+00, 1.120e+02,
        4.000e+00],
       [3.000e+00, 7.500e+01, 9.937e+03, ..., 6.800e+01, 0.000e+00,
        4.000e+00]])
```

In [42]:

```
X_selected.shape
```

Out[42]:

```
(1438, 50)
```

In [ ]:

```

```

Model Selection

In [43]:

```
x_train,x_test, y_train, y_test = train_test_split(X_selected,Y)
```

In [ ]:

```

```

Lasso with cv

In [44]:

```
la = LassoCV()
```

In [45]:

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

Out[45]:

```
▾ LassoCV
LassoCV()
```

In [46]:

```
yl = la.predict(x_test)
```

In [47]:

```
r2_score(y_test,yl)
```

Out[47]:

```
0.7203969041650569
```

In [ ]:

```

```

Ridge with cv

In [48]:

```
ri = RidgeCV()
```

In [49]:

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

Out[49]:

```
▾ RidgeCV
RidgeCV()
```

In [50]:

```
yr = ri.predict(x_test)
```

In [51]:

```
r2_score(y_test,yr)
```

Out[51]:

```
0.8045605272308718
```

In [ ]:

```

```

Linear Regression

In [52]:

```
lr = LinearRegression()
```

In [53]:

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

Out[53]:

```
▾ LinearRegression
LinearRegression()
```

In [54]:

```
yp = lr.predict(x_test)
```

In [55]:

```
r2_score(y_test,yp)
```

Out[55]:

```
0.8041996302976034
```

In [ ]:

```

```

Random Forest

In [56]:

```
rf = RandomForestRegressor(max_depth=100,n_estimators=500)
```

In [57]:

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

Out[57]:

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=100, n_estimators=500)
```

In [58]:

```
ypr = rf.predict(x_test)
```

In [59]:

```
r2_score(y_test,ypr)
```

Out[59]:

```
0.8685917190533834
```

In [ ]:

```

```

In [ ]:

```

```

Hyperparameter Tuning

In [60]:

```
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 30, 40, 50, 60, 70, 80, 90, 100,200]}
```

In [61]:

```
lasso = Lasso()
folds = 5
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'r2',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 33 candidates, totalling 165 fits
```

Out[61]:

```
▸ GridSearchCV
▸ estimator: Lasso
  ▸ Lasso
```

In [62]:

```
model_cv.best_params_
```

Out[62]:

```
{'alpha': 200}
```

In [63]:

```
model_cv.best_estimator_
```

Out[63]:

```
▾ Lasso
Lasso(alpha=200)
```

In [64]:

```
alpha = model_cv.best_estimator_.alpha
# max_iter = model_cv.best_estimator_.max_iter
lasso = Lasso(alpha=alpha)

lasso.fit(x_train, y_train)
lasso.coef_
```

Out[64]:

```
array([-1.48371917e+03, -6.88416325e+01,  4.18480010e-01, -1.37922075e+0
3,
        3.82950030e+02, -2.17550782e+02, -2.48981520e+03,  1.25214119e+0
4,
        1.10557108e+02,  2.35769483e+02,  2.64180011e+03, -8.50513336e+0
2,
        6.26752454e+02,  1.41943284e+01, -6.06419523e+03, -0.00000000e+0
0,
       -0.00000000e+00, -5.14882000e+03, -2.93559517e+03, -6.39627517e-0
1,
       -6.55930363e+00,  3.26025889e+00, -0.00000000e+00, -1.07157416e+0
3,
        3.77185692e+03,  1.53531301e+02,  4.71321726e+01,  3.98109986e+0
1,
       -2.05276287e+00,  7.41881581e+03,  2.44901518e+02, -0.00000000e+0
0,
        0.00000000e+00, -1.30215951e+04, -6.60248821e+03,  1.42625318e+0
3,
        4.37576499e+03,  5.13444349e+03, -7.18194584e+02, -1.28716595e+0
2,
       -2.75362466e+03,  1.08876150e+04,  6.82193135e+00,  2.08602769e+0
2,
        4.93038289e+02,  3.64222187e+03,  2.54717553e+01,  1.97511806e+0
1,
        1.44662283e+00,  3.38375110e+03])
```

In [65]:

```
y_pred = lasso.predict(x_test)
print(r2_score(y_pred, y_test))
```

```
0.7907529083887381
```

In [ ]:

```

```

In [66]:

```
ridge = Ridge()
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'neg_mean_squared_error',
                        cv = 10,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 33 candidates, totalling 330 fits
```

Out[66]:

```
▸ GridSearchCV
▸ estimator: Ridge
    ▸ Ridge
```

In [67]:

```
model_cv.best_estimator_
```

Out[67]:

```
▾ Ridge
Ridge(alpha=70)
```

In [68]:

```
alpha = model_cv.best_estimator_.alpha
ridge= Ridge(alpha=alpha)

ridge.fit(x_train, y_train)
lasso.coef_
```

Out[68]:

```
array([-1.48371917e+03, -6.88416325e+01,  4.18480010e-01, -1.37922075e+0
3,
        3.82950030e+02, -2.17550782e+02, -2.48981520e+03,  1.25214119e+0
4,
        1.10557108e+02,  2.35769483e+02,  2.64180011e+03, -8.50513336e+0
2,
        6.26752454e+02,  1.41943284e+01, -6.06419523e+03, -0.00000000e+0
0,
       -0.00000000e+00, -5.14882000e+03, -2.93559517e+03, -6.39627517e-0
1,
       -6.55930363e+00,  3.26025889e+00, -0.00000000e+00, -1.07157416e+0
3,
        3.77185692e+03,  1.53531301e+02,  4.71321726e+01,  3.98109986e+0
1,
       -2.05276287e+00,  7.41881581e+03,  2.44901518e+02, -0.00000000e+0
0,
        0.00000000e+00, -1.30215951e+04, -6.60248821e+03,  1.42625318e+0
3,
        4.37576499e+03,  5.13444349e+03, -7.18194584e+02, -1.28716595e+0
2,
       -2.75362466e+03,  1.08876150e+04,  6.82193135e+00,  2.08602769e+0
2,
        4.93038289e+02,  3.64222187e+03,  2.54717553e+01,  1.97511806e+0
1,
        1.44662283e+00,  3.38375110e+03])
```

In [69]:

```
y_r = ridge.predict(x_test)
```

In [70]:

```
r2_score(y_test,y_r)
```

Out[70]:

```
0.8044679913523669
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: