step: 1

```
import pandas as pd
import csv.
house = pd.read_csv("Ames_House_sales_Cropped.csv")
house
house.head()
house.shape.
df = pd.read_csv("Ames_House_sales_Cropped.csv")
df
d = df.columns
d
df.Pnl(00)
df.dtypes.value_counts
```

step: 3

```
.house.drop(['Bldg Type'],axis=1)
df.pop('Central Air')
```

Lab4. House Price Prediction using LR with Regularization

Objectives

In this lab, you will learn how to build Regression Models to predict the Sales Price of houses using Ames, Iowa dataset.

Learning Outcomes

After completing this lab, you will be able to

- Understand data and build baseline LR model
- Apply One Hot Encoding to categorical features and rebuild LR model
- Apply scaling using StandardScaler and MinMaxScaler and rebuild LR model
- Compare the performance of LR model with SGD Regressor, RidgeCV and LassoCV
- Compute RMSE for all models.

Business Use Case

Ames house sales price dataset contains past sales price details of 1379 homes from Ames, USA. This dataset has 79 feature columns (independent variables) and the dependent variable, sale price ("**SalePrice**"). There are three different types: integers (int64), floats (float64), and strings (object, categoricals).

In this lab, you will use a cropped dataset where we have removed all object columns, except two object columns – building type ("**BldgType**") and centralized air conditioning ("**CentralAir**"). Therefore, it contains only 38 independent variables.

With the help of the regression model you will build on this reduced dataset, we can predict the sale price of a house given the values for all independent variables.

Step1. [Import dataset]. Using Pandas, import "**Ames_House_Sales_Cropped.csv**" file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Predict Sale Price without Categorical features].

- Drop both categorical features – BldgType and CentralAir (USE drop() and pop() methods)
- Prepare X matrix (36 feature columns) and y vector (ie., SalePrice column)
- Split dataset for training and testing as X_train, X_test, y_train, y_test (use **25%** test size).
- Create LinearRegression model, fit on training set and predict on test set
- Compute Mean Squared Error (MSE) on actual values and predicted values (you will get output as 1474827326.0).

Step3. [Create Scatter Plot]. Plot Scatterplot between **y_test** and **y_pred**.

Step4. [Encode Categorical columns]. Using **get_dummies()** method, perform one hot encoding on the two categorical columns, *BldgType* and *CentralAir*. Now, you will get 5 columns for BldgType variable and 2 columns for CentralAir column. So, now you have 43 independent variables and 1 dependent variable.

Step5. [Predict Sale Price with Categorical features]

- Prepare X matrix (43 feature columns) and y vector (ie., SalePrice column)
- Split dataset for training and testing
- Create LinearRegression model, fit on training set and predict on test set
- Compute Mean Squared Error on actual values and predicted values (you will get output as 1461036570.0).

Step6. [Normalize using StandardScaler and Predict Sale Price]

- Using StandardScaler, perform **fit_transform()** on X_train and **transform()** on X_test matrix that you already splitted.

```
new_df = df.drop(['Bldg Type'], axis=1)
new_df.
y = new_df['Sales Price']
y
inp = ['1st FLrSF', '2ndFlrSF', '3SSn porch',
       'Bed roomAbvGr', 'BsmtFinSF1']
x = new_df[inp]
x
from sklearn.model selection import train_test_split.
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
from sklearn.linear_model import linear Regression
```

- Create a new Linear Regression model, fit on *scaled X_train* and *y_train* and predict on *scaled X_test*.
- Compute Mean Squared Error (MSE) on actual values and predicted values (you will get output as 1461036570.0).

Step7. [Normalize using MinMaxScaler and Predict Sale Price]

- Repeat Step6 using MinMaxScaler
- Mean Squared Error will be: 1461036570.0

Step8. [Predict using SGD Regressor]

- Use scaled X_train and X_test using StandardScaler that you computed before
- Create SGDRegressor, fit and predict
- Compute MSE on y_test and y_pred this time. You will get output as 1592430104.0.

Step8. [Predict using Ridge Regression]

- Use scaled X_train and X_test using StandardScaler that you computed before
- Create RidgeCV, fit and predict
- Compute MSE on y_test and y_pred this time. You will get output as 1442196000.3367693.

Step8. [Predict using Lasso Regression]

- Use scaled X_train and X_test using StandardScaler that you computed before
- Create LassoCV, fit and predict
- Compute MSE on y_test and y_pred this time. You will get output as 1409368613.5329669.

Step9.[RMSE]. Print Root Mean Squared Error values (use numpy.sqrt() method) as below and compare error values.

RMSE without one hot encoding: 38403.0
RMSE with One hot encoding: 38224.0
RMSE with OHE and Standard Scaling: 38224.0
RMSE with OHE and MinMax Scaling: 38224.0
RMSE of SGDRegressor with OHE and Standard Scaler: 38528.0
RMSE of RidgeCV with OHE and Standard Scaler: 37976.0
RMSE of LassoCV with OHE and Standard Scaler: 37542.0

```
model = LinearRegression()
model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_pred

from sklearn.metrics, import mean_squared_error.
    mean_squared_error(y_test, y_pred)
```

Step 3:

```
plt.scatter(y_test y_pred)
```

Step 4:

```
en_df = pd.get_dummies(df, columns=["Central Air", "Bldg Type"])
    en_df.head()
    en.df.shape
    en_df.columns
```

Step 5

```
e_y = en_df["Sales price"]
    e_y
e_inp = ["1st FlrSF", "2nd FlrSF", "3SSnporch",
    "Bedroom AbvGr", "BsmtFins", "Enclosedporch",
    "Fireplaces", "Full Bath", "GrLivArea",
    "LotArea", "LotFrontage", "OverallQual", "FinSF",
    "HSSUbclass", "OverallCond", "PoolArea", "Screen
    Porch", "TotRmsAbvGrd", "Central Air_N"
```

NOTE

'CentralAir_Y', 'BldgType_1Fam', 'BldgType_2Fam']

```
e.X = en_df[e_inp]

e_X_train, ex_test, ey_train, ey_test = train_
                    test_split(e_X, e_y)

model1 = LinearRegression()
model1.fit(ex_train, ey_train)
ey_pred = model1.predict(ex_test)
ey_pred
```

Step6:

```
from sklearn.preprocessing import standard
                                  scaler.
scales = standard scaler()
ss = scale.fit.transform(ex_train)
ss
ss1 = scale.transform(ex_test)
ss1
model2 = Linear.Regression()
model2.fit(ss, ey_train)
se_X_pred = model12.predict(ss1)
se_X_pred
mean_squared_error(ey_test, se_y_pred)
```

Step 7:

```
from sklearn.preprocessing import MinMaxScaler
m_scaler = MinMaxScaler()
m_ss = m_scaler.fit_transform(ex_train)
m_ss
m_ss1 = m_scaler.transform(ex_test)
m_ss1
model3 = LinearRegression()
model3.fit(m_ss, ey_train)
me_y_pred = model3.predict(m_ss1)
me_y_pred
mean_squared_error(ey_test, me_y_pred)
```

Step 8:

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
SGD = make_pipeline(StandardScaler(), SGDRegressor
    (max_iter=1000, t
SGD.fit(X, X)
r_y_pred = SGD.predict(X)
r_y_pred
mean_squared_error(ey_train, ridge_y_pred)
```

Step 9:

```
from math import sqrt
print("RMSE without One hot encoding", sqrt(mean_squared
    _error))
```

NOTE

```
print("RMSE with one and standard Scaling: ", sqrt(s-me))
print("RMSE with one and MinMax Scaling: ", sqrt(mse-mm))
print("RMSE of SGDRegressor with one and standard scaler:", sqrt(mse-sag))
print("RMSE of Ridge cv with one and standard scaler:", sqrt(mse-rid))
print("RMSE of Lassocv with one and standard scaler:", sqrt(mse-las))
```