IT-641 Deep Learning

Lab 2

House Price Prediction: Predict house prices using advanced regression techniques

SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.

MSSubClass: The building class

MSZoning: The general zoning classification

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access

Alley: Type of alley access

LotShape: General shape of property

LandContour: Flatness of the property

Utilities: Type of utilities available

LotConfig: Lot configuration

LandSlope: Slope of property

Neighborhood: Physical locations within Ames city limits

Condition1: Proximity to main road or railroad

Condition2: Proximity to main road or railroad (if a second is present)

BldgType: Type of dwelling

HouseStyle: Style of dwelling

OverallQual: Overall material and finish quality

OverallCond: Overall condition rating

YearBuilt: Original construction date

YearRemodAdd: Remodel date

RoofStyle: Type of roof

RoofMatl: Roof material

Exterior1st: Exterior covering on house

Exterior2nd: Exterior covering on house (if more than one material)

MasVnrType: Masonry veneer type

MasVnrArea: Masonry veneer area in square feet

ExterQual: Exterior material quality

ExterCond: Present condition of the material on the exterior

Foundation: Type of foundation

BsmtQual: Height of the basement

BsmtCond: General condition of the basement

BsmtExposure: Walkout or garden level basement walls

BsmtFinType1: Quality of basement finished area

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Quality of second finished area (if present)

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

HeatingQC: Heating quality and condition

CentralAir: Central air conditioning

Electrical: Electrical system

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors) GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Number of bedrooms above basement level

Kitchen: Number of kitchens

KitchenQual: Kitchen quality

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality rating

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

GarageType: Garage location

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

GarageCond: Garage condition

PavedDrive: Paved driveway

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

Fence: Fence quality

MiscFeature: Miscellaneous feature not covered in other categories

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold

YrSold: Year Sold

SaleType: Type of sale

SaleCondition: Condition of sale

It seems quite a lot of information in columns.

Tasks:

For the given datasets perform the following tasks:

1. Load Data and Find out if it has any missing values
2. Correct the missing values
3. Identify and encode necessary features
4. Identify and normalize necessary features

5. Split the dataset into train set (75%) test set (15%) an validation set (10%)
6. For Regression Task: Write a code manually for Linear Regression and compare the results with sklearns linear regression model.
7. For Classification Task: Write manual code for logistic regression using Gradient Descent and compare it with sklearns Logistic Regression
8. Use all the respective model performance criteria and compare to model
9. Discuss under-fitting and overfitting based upon results

1.Loading Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
import warnings
warnings.filterwarnings(action = 'ignore')
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score,cross_val_predict
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

```

2. Load Dataset

```
data = pd.read_csv("https://raw.githubusercontent.com/Jatansahu/DEEP_LEARNING_ASSIGNMENTS/main/LAB_02/
```

```
# Previewing data
data.head()
```

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

5 rows × 81 columns

Looking at the above dataset our target variable is the column "SalePrice"

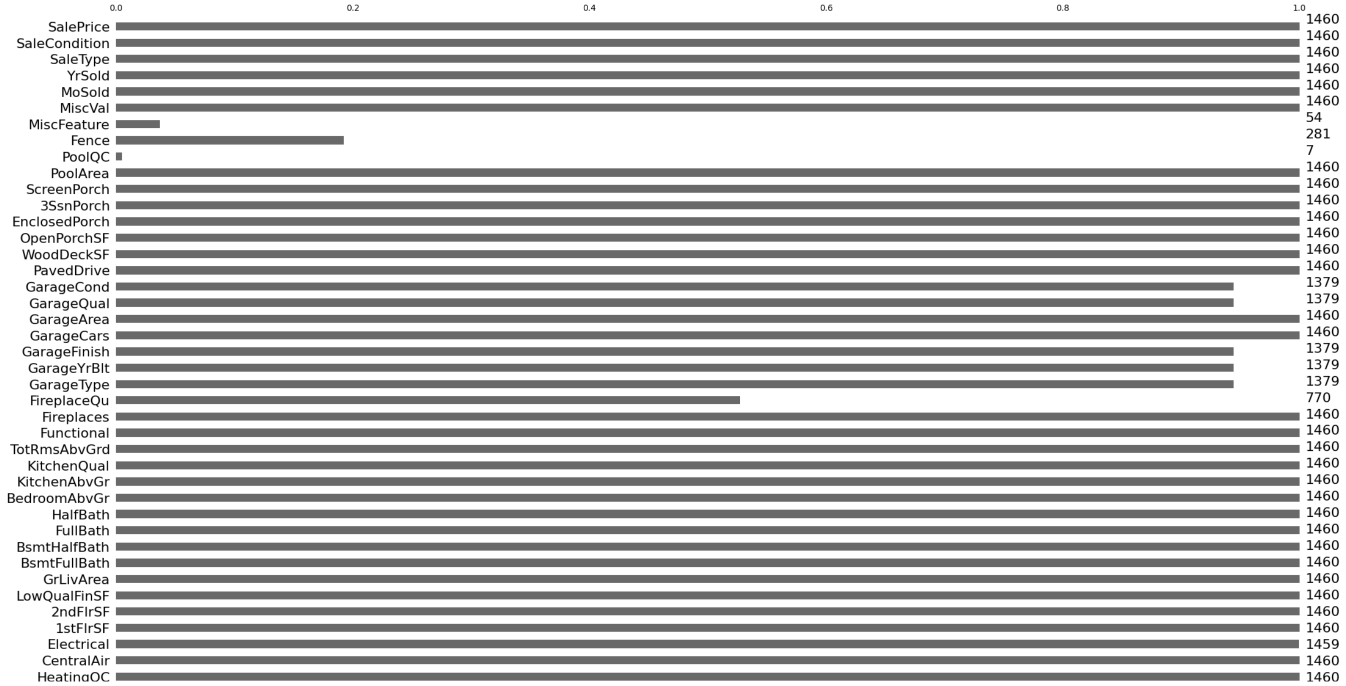
3.Looking for Null values

```
data.info()
```

```
 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
```

```
import missingno as msno
msno.bar(data)
```

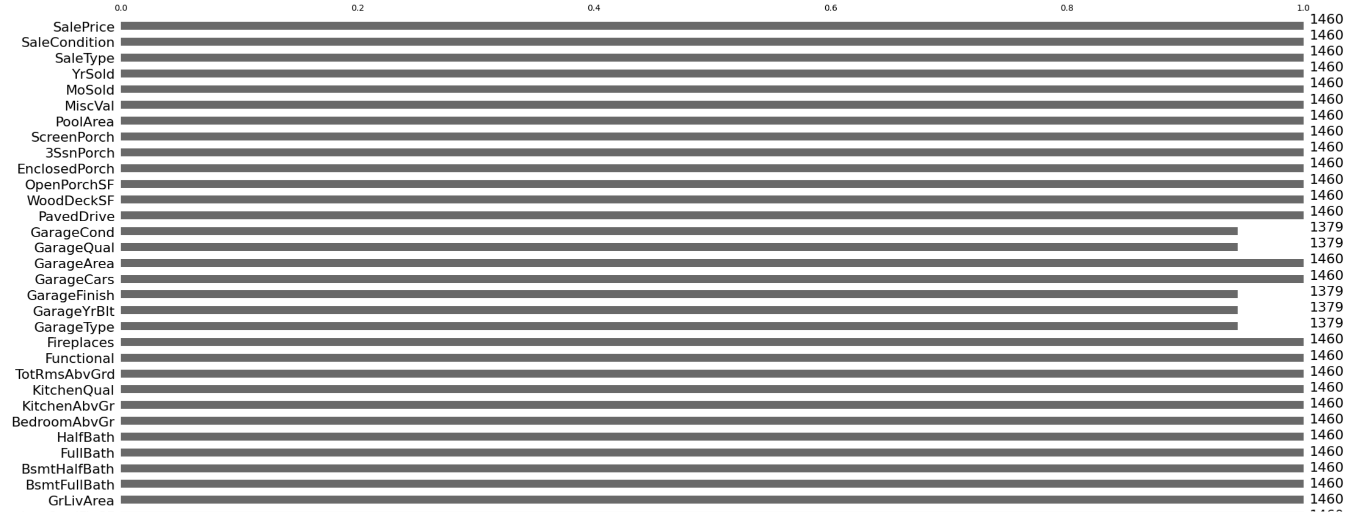
```
<Axes: >
```



Dropping those columns which have less than 25% values


```
# Calculate the threshold for null values
threshold = len(data) * 0.25  # 25% of total rows

# Iterate through the columns and drop those with null percentages greater than the threshold
for column in data.columns:
    if data[column].isnull().sum() > threshold:
        data.drop(column, axis=1, inplace=True)
print(msno.bar(data))
```

```
Axes(0.125,0.11;0.775x0.77)
```



Double-click (or enter) to edit

We still have some missing values in our dataset we will look into further

4.Basic EDA

1.Descriptive Analysis


```
data.describe().style.background_gradient()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFi
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.00
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.63
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.09
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.00
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.00
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.50
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.25
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.00

2 Correlation

```
correlation_matrix = data.corr()
print("Correlation Matrix:")
(correlation_matrix.style.background_gradient())
```

Correlation Matrix:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinS
Id	1.000000	0.011156	-0.010601	-0.033226	-0.028365	0.012609	-0.012713	-0.021998	-0.050298	-0.0050
MSSubClass	0.011156	1.000000	-0.386347	-0.139781	0.032628	-0.059316	0.027850	0.040581	0.022936	-0.0698
LotFrontage	-0.010601	-0.386347	1.000000	0.426095	0.251646	-0.059213	0.123349	0.088866	0.193458	0.2336
LotArea	-0.033226	-0.139781	0.426095	1.000000	0.105806	-0.005636	0.014228	0.013788	0.104160	0.2141
OverallQual	-0.028365	0.032628	0.251646	0.105806	1.000000	-0.091932	0.572323	0.550684	0.411876	0.2396
OverallCond	0.012609	-0.059316	-0.059213	-0.005636	-0.091932	1.000000	-0.375983	0.073741	-0.128101	-0.0462
YearBuilt	-0.012713	0.027850	0.123349	0.014228	0.572323	-0.375983	1.000000	0.592855	0.315707	0.2495
YearRemodAdd	-0.021998	0.040581	0.088866	0.013788	0.550684	0.073741	0.592855	1.000000	0.179618	0.1284
MasVnrArea	-0.050298	0.022936	0.193458	0.104160	0.411876	-0.128101	0.315707	0.179618	1.000000	0.2647
BsmtFinSF1	-0.005024	-0.069836	0.233633	0.214103	0.239666	-0.046231	0.249503	0.128451	0.264736	1.0000
BsmtFinSF2	-0.005968	-0.065649	0.049900	0.111170	-0.059119	0.040229	-0.049107	-0.067759	-0.072319	-0.0501
BsmtUnfSF	-0.007940	-0.140759	0.132644	-0.002618	0.308159	-0.136841	0.149040	0.181133	0.114442	-0.4952
TotalBsmtSF	-0.015415	-0.238518	0.392075	0.260833	0.537808	-0.171098	0.391452	0.291066	0.363936	0.5223
1stFlrSF	0.010496	-0.251758	0.457181	0.299475	0.476224	-0.144203	0.281986	0.240379	0.344501	0.4458
2ndFlrSF	0.005590	0.307886	0.080177	0.050986	0.295493	0.028942	0.010308	0.140024	0.174561	-0.1370
LowQualFinSF	-0.044230	0.046474	0.038469	0.004779	-0.030429	0.025494	-0.183784	-0.062419	-0.069071	-0.0645
GrLivArea	0.008273	0.074853	0.402797	0.263116	0.593007	-0.079686	0.199010	0.287389	0.390857	0.2081
BsmtFullBath	0.002289	0.003491	0.100949	0.158155	0.111098	-0.054942	0.187599	0.119470	0.085310	0.6492
BsmtHalfBath	-0.020155	-0.002333	-0.007234	0.048046	-0.040150	0.117821	-0.038162	-0.012337	0.026673	0.0674
FullBath	0.005587	0.131608	0.198769	0.126031	0.550600	-0.194149	0.468271	0.439046	0.276833	0.0585
HalfBath	0.006784	0.177354	0.053532	0.014259	0.273458	-0.060769	0.242656	0.183331	0.201444	0.0042
BedroomAbvGr	0.037719	-0.023438	0.263170	0.119690	0.101676	0.012980	-0.070651	-0.040581	0.102821	-0.1073
KitchenAbvGr	0.002951	0.281721	-0.006069	-0.017784	-0.183882	-0.087001	-0.174800	-0.149598	-0.037610	-0.0810
TotRmsAbvGrd	0.027239	0.040380	0.352096	0.190015	0.427452	-0.057583	0.095589	0.191740	0.280682	0.0443
Fireplaces	-0.019772	-0.045569	0.266639	0.271364	0.396765	-0.023820	0.147716	0.112581	0.249070	0.260C
GarageYrBlt	0.000072	0.085072	0.070250	-0.024947	0.547766	-0.324297	0.825667	0.642277	0.252691	0.1534
GarageCars	0.016570	-0.040110	0.285691	0.154871	0.600671	-0.185758	0.537850	0.420622	0.364204	0.2240
GarageArea	0.017634	-0.098672	0.344997	0.180403	0.562022	-0.151521	0.478954	0.371600	0.373066	0.2969
WoodDeckSF	-0.029643	-0.012579	0.088521	0.171698	0.238923	-0.003334	0.224880	0.205726	0.159718	0.2043

3.Feature selection (Numerical)

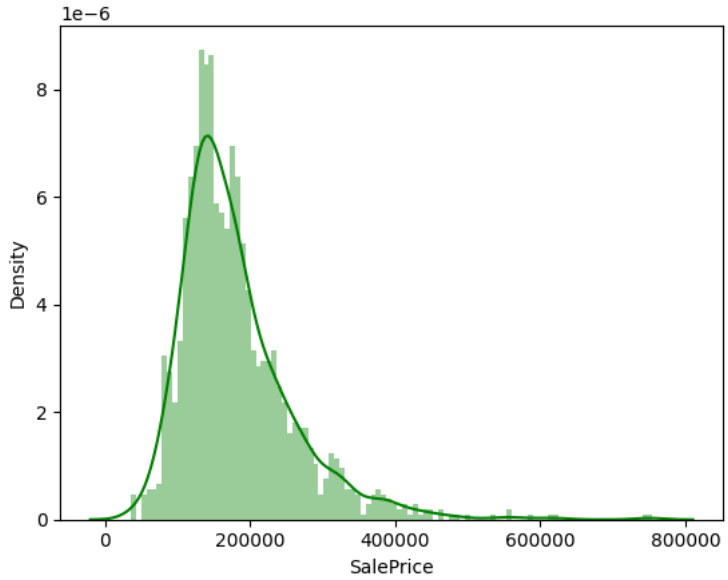
```
# Set the correlation threshold for feature selection
correlation_threshold = 0.3

print("Before Feature Selection ",data.shape)

# Select features with correlation above the threshold
selected_features = []
for column in correlation_matrix.columns:
    if abs(correlation_matrix['SalePrice'][column]) < correlation_threshold:
        data = data.drop(columns=column)
    else:
      selected_features.append(column)
print("Selected Features",selected_features)
print("After Feature Selection ",data.shape)
```

```
Before Feature Selection  (1460, 76)
Selected Features ['LotFrontage', 'OverallQual', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'TotalBsmtSF', '1stFlrSF'
After Feature Selection  (1460, 57)
```

4. Checking Skewness of Target Variable

```
import seaborn as sns
sns.distplot(data['SalePrice'], color='g', bins=100, hist_kws={'alpha' : 0.4})
```

```
<Axes: xlabel='SalePrice', ylabel='Density'>
```



Salesprice is right skewed means mean> median

5.Preprocessing

```
data.head()
```

	MSZoning	LotFrontage	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	...	GarageCars
0	RL	65.0	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	...	2
1	RL	80.0	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	...	2
2	RL	68.0	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	...	2
3	RL	60.0	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	...	3
4	RL	84.0	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	...	3

5 rows × 57 columns

1.Dropping Unnecessary columns

Droping Id column which is not usefull. We already deleted in correlation part

Scaling feature

```
SalePrice_data = data[["SalePrice"]]  # Extracting the "SalePrice" column as a separate DataFrame
scaler = StandardScaler()
scaled_SalePrice = scaler.fit_transform(SalePrice_data.values.reshape(-1,1))
```

```
# Droping profit column from dataset
data.drop(["SalePrice"],1,inplace = True)
data['scaled_SalePrice'] = scaled_SalePrice
```

```
# # Get the mean and standard deviation from the scaler
# mean_profit = scaler.mean_[0]
# std_dev_profit = scaler.scale_[0]

# scaled_prediction =  2.01120333
```

```

# # Reverse the scaling to get the prediction in the original units
# original_prediction = (scaled_prediction * std_dev_profit) + mean_profit

# print("Original prediction in dollars:", original_prediction)
```

```
sc = StandardScaler()
for col in data.columns:
    if data[col].dtype in ['int64' , 'float64']:
          data[col] = sc.fit_transform(data[col].values.reshape(-1,1))
```

2.Converting Categorical Variables into their corresponding form

A).Cardinality

"Cardinality" means the number of unique values in a column

```
for col in data.columns:
    if data[col].dtype=='object':
        print()
        print(col)
        print('Number of unique values :',data[col].nunique())
        print('Sample unique values :',data[col].unique()[:5])
```

```
SaleCondition
Number of unique values : 6
Sample unique values : ['Normal' 'Abnorml' 'Partial' 'AdjLand' 'Alloca']
```

B) Encoding all the categorical variable using labelencoder

```
categorical_cols = [cname for cname in data.columns
                    if data[cname].dtype == "object"]
# categorical_cols
```

```
#encoding Embarked column
le = LabelEncoder()
for col in data.columns:
    if data[col].dtype=='object':
      data[col] = le.fit_transform(data[col])
```

```
sc = StandardScaler()
for col in data.columns:
    if data[col].dtype in ['int64' , 'float64']:
          data[col] = sc.fit_transform(data[col].values.reshape(-1,1))
```

```
data.info()
```

```
 1   LotFrontage    1201 non-null   float64
 2   Street         1460 non-null   float64
 3   LotShape       1460 non-null   float64
 4   LandContour    1460 non-null   float64
 5   Utilities      1460 non-null   float64
 6   LotConfig      1460 non-null   float64
 7   LandSlope      1460 non-null   float64
 8   Neighborhood   1460 non-null   float64
 9   Condition1     1460 non-null   float64
 10  Condition2     1460 non-null   float64
 11  BldgType       1460 non-null   float64
 12  HouseStyle     1460 non-null   float64
 13  OverallQual    1460 non-null   float64
 14  YearBuilt      1460 non-null   float64
 15  YearRemodAdd   1460 non-null   float64
 16  RoofStyle      1460 non-null   float64
 17  RoofMatl       1460 non-null   float64
 18  Exterior1st    1460 non-null   float64
 19  Exterior2nd    1460 non-null   float64
 20  MasVnrType     1460 non-null   float64
 21  MasVnrArea     1452 non-null   float64
 22  ExterQual      1460 non-null   float64
 23  ExterCond      1460 non-null   float64
 24  Foundation     1460 non-null   float64
 25  BsmtQual       1460 non-null   float64
 26  BsmtCond       1460 non-null   float64
 27  BsmtExposure   1460 non-null   float64
 28  BsmtFinType1   1460 non-null   float64
 29  BsmtFinSF1     1460 non-null   float64
 30  BsmtFinType2   1460 non-null   float64
 31  TotalBsmtSF    1460 non-null   float64
 32  Heating        1460 non-null   float64
 33  HeatingQC      1460 non-null   float64
 34  CentralAir     1460 non-null   float64
 35  Electrical     1460 non-null   float64
 36  1stFlrSF       1460 non-null   float64
 37  2ndFlrSF       1460 non-null   float64
 38  GrLivArea      1460 non-null   float64
 39  FullBath       1460 non-null   float64
 40  KitchenQual    1460 non-null   float64
 41  TotRmsAbvGrd   1460 non-null   float64
 42  Functional     1460 non-null   float64
 43  Fireplaces     1460 non-null   float64
 44  GarageType     1460 non-null   float64
 45  GarageYrBlt    1379 non-null   float64
 46  GarageFinish   1460 non-null   float64
 47  GarageCars     1460 non-null   float64
 48  GarageArea     1460 non-null   float64
 49  GarageQual     1460 non-null   float64
 50  GarageCond     1460 non-null   float64
```

```
dtypes: float64(57)
memory usage: 650.3 KB
```

Data imputation

```
def auto_data_impute(data,get_rid_percent=2):
    for x,y in data.isnull().sum().items():
        percent = y/data.shape[0]
        if percent <= get_rid_percent/100:
            data[x]  = data[x].fillna(data[x].mean())
        else:
            print("removed column : ",x)
            data = data.drop([x],axis=1)
    print("Data imputation successfull")
    return data
```

```
data = auto_data_impute(data,get_rid_percent=2)
```

```
removed column :  LotFrontage
removed column :  GarageYrBlt
Data imputation successfull
```

6.Splitting the dataset

```
data
```

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	...	GarageC
0	-0.045532	0.064238	0.750731	0.314667	-0.02618	0.604670	-0.225716	-1.206215	-0.036289	-0.03174	...	0.311
1	-0.045532	0.064238	0.750731	0.314667	-0.02618	-0.628316	-0.225716	1.954302	-1.188074	-0.03174	...	0.311
2	-0.045532	0.064238	-1.378933	0.314667	-0.02618	0.604670	-0.225716	-1.206215	-0.036289	-0.03174	...	0.311
3	-0.045532	0.064238	-1.378933	0.314667	-0.02618	-1.861302	-0.225716	-1.039872	-0.036289	-0.03174	...	1.650
4	-0.045532	0.064238	-1.378933	0.314667	-0.02618	-0.628316	-0.225716	0.457215	-0.036289	-0.03174	...	1.650
...	...	...	...	...	...	...	...	...	...	...	...	
1455	-0.045532	0.064238	0.750731	0.314667	-0.02618	0.604670	-0.225716	-0.707186	-0.036289	-0.03174	...	0.311
1456	-0.045532	0.064238	0.750731	0.314667	-0.02618	0.604670	-0.225716	0.290872	-0.036289	-0.03174	...	0.311
1457	-0.045532	0.064238	0.750731	0.314667	-0.02618	0.604670	-0.225716	-1.039872	-0.036289	-0.03174	...	-1.026
1458	-0.045532	0.064238	0.750731	0.314667	-0.02618	0.604670	-0.225716	-0.041814	-0.036289	-0.03174	...	-1.026
1459	-0.045532	0.064238	0.750731	0.314667	-0.02618	0.604670	-0.225716	-0.873529	-0.036289	-0.03174	...	-1.026

1460 rows × 55 columns

```
x = data.iloc[:,:54]
y = data['scaled_SalePrice']
```

```
y
```

```
0       0.347273
1       0.007288
2       0.536154
3      -0.515281
4       0.869843
          ...
1455   -0.074560
1456    0.366161
1457    1.077611
1458   -0.488523
1459   -0.420841
Name: scaled_SalePrice, Length: 1460, dtype: float64
```

```
x.head()
```

Neighborhood	Condition1	Condition2	...	GarageFinish	GarageCars	GarageArea	GarageQual	GarageCond	PavedDrive	WoodDeckSF	OpenI
-1.206215	-0.036289	-0.03174	...	-0.318475	0.311725	0.351000	0.11211	0.0689	0.289745	-0.752176	0
1.954302	-1.188074	-0.03174	...	-0.318475	0.311725	-0.060731	0.11211	0.0689	0.289745	1.626195	-0
-1.206215	-0.036289	-0.03174	...	-0.318475	0.311725	0.631726	0.11211	0.0689	0.289745	-0.752176	-0
-1.039872	-0.036289	-0.03174	...	0.801942	1.650307	0.790804	0.11211	0.0689	0.289745	-0.752176	-0
0.457215	-0.036289	-0.03174	...	-0.318475	1.650307	1.698485	0.11211	0.0689	0.289745	0.780197	0

Training set - 75%

Testing set - 15%

Validation set - 10%

```
x_train,x_part,y_train,y_part = train_test_split(x,y,test_size = 0.25,random_state = 1)
x_test,x_valid,y_test,y_valid = train_test_split(x_part,y_part,test_size = 0.4,random_state = 1)
```

```
print(x_train.shape,x_test.shape,x_valid.shape)
print(y_train.shape,y_test.shape,y_valid.shape)
```

```
(1095, 54) (219, 54) (146, 54)
(1095,) (219,) (146,)
```

7.Model Selection

A) Linear regression from scratch

```
class LinearRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        # Gradient descent optimization
        for _ in range(self.num_iterations):
            predicted = np.dot(X, self.weights) + self.bias
            error = y - predicted

            # Update weights and bias using gradients
            self.weights += (self.learning_rate / num_samples) * np.dot(X.T, error)
            self.bias += (self.learning_rate / num_samples) * np.sum(error)

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

# Create a linear regression model
model = LinearRegression(learning_rate=0.01, num_iterations=1000)

# Fit the model to the data
model.fit(x_train, y_train)
```

```
# Calculate Mean Squared Error (MSE)
def mean_squared_error(y_test,y_pred ):
    return np.mean((y_test - y_pred) ** 2)

# Calculate Root Mean Squared Error (RMSE)
def root_mean_squared_error(y_test, y_pred):
    return np.sqrt(mean_squared_error(y_test, y_pred))
```

```
# Calculate R-squared (coefficient of determination)
def r_squared(y_test, y_pred):
    y_mean = np.mean(y_test)
    ss_total = np.sum((y_test - y_mean) ** 2)
    ss_residual = np.sum((y_test - y_pred) ** 2)
    return 1 - (ss_residual / ss_total)

```

```
# For testing
# Make predictions
y_pred = model.predict(x_test)
```

```
# Calculate metrics
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r_squared(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Squared Error: 0.18648684238029734
Root Mean Squared Error: 0.43184122357678795
R-squared: 0.8466170598362222
```

USING SCIKIT LEARN LIBRARY

```
def model_performance(model,model_name,x_train = x_train,y_train = y_train,x_test = x_test,y_test = y_

    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)
    y_val_pred = model.predict(x_valid)

    # Training_Score = np.round(model.score(x_train,y_train),3)
    # Testing_Score = np.round(model.score(x_test,y_test),3)
    # Validation_score = np.round(model.score(x_valid,y_valid))

    # mse_training = np.round(mean_squared_error(y_train,y_train_pred),3)
    mse_testing = np.round(mean_squared_error(y_test,y_test_pred),3)
    # mse_validation = np.round(mean_squared_error(y_valid,y_val_pred),3)

    # mae_training = np.round(mean_absolute_error(y_train,y_train_pred),3)
    mae_testing = np.round(mean_absolute_error(y_test,y_test_pred),3)
    # mae_valid = np.round(mean_absolute_error(y_valid,y_val_pred),3)

    # r2_training = np.round(r2_score(y_train,y_train_pred),3)
    r2_testing = np.round(r2_score(y_test,y_test_pred),3)
    # r2_valid = np.round(r2_score(y_valid,y_val_pred),3)

    print("Model Performance for:",model_name)
    print("")

    # print("Training Score:",Training_Score)
    # print("Testing Score:",Testing_Score)
    # print("Validation Score",Validation_score)
    # print("")

    # print("Training Data Mean Squared Error:",mse_training)
    print("Testing Data Mean Squared Error:",mse_testing)
    # print("Validation Data Mean Squared Error:",mse_validation)

    print("")

    # print("Training Data Mean Absolute Error:",mae_training)
    print("Testing Data Mean Absolute Error:",mae_testing)
    # print("Validation Data Mean Absolute Error:",mae_valid)
```

```
    # print("Validation Data Mean Absolute Error:",mae_valid)
    print("")

    # print("Training Data r2_score:",r2_training)
    print("Testing Data r2_score:",r2_testing)
    # print("Validation Data r2_score:",r2_valid)
    print("")

    print("Residual Analysis:")
    # plt.figure(figsize = (20,5))
    # # plt.scatter(y_train,(y_train-y_train_pred),color = "red",label = 'Training Predictions')
    # plt.scatter(y_test,(y_test-y_test_pred),color = "green",label = 'Testing Predictions')
    # # plt.scatter(y_valid,(y_valid-y_val_pred),color = 'blue',label = "Validation Predictions")
    # plt.legend()
    # plt.show()

    return mse_testing,mae_testing,r2_testing
```

```
model1 = LinearRegression()
model1.fit(x_train,y_train)
```

Comparisation

```
lr_perf = model_performance(model1,model_name = model1)
```

```
Model Performance for: <__main__.LinearRegression object at 0x7c1df46b4850>

Testing Data Mean Squared Error: 0.186

Testing Data Mean Absolute Error: 0.248

Testing Data r2_score: 0.847

Residual Analysis:
```

Linear Reggression written in scrach (For testing set)

```
Mean Squared Error: 0.2155905940198445
Root Mean Squared Error: 0.4643173419331271
R-squared: 0.822679612350427
```