What is linear regreesion and its uses

regression used find relationship b/w dependent variable and independent variables

Regression:

```
Regression is a statistical method used in machine learning and
statistics to examine the relationship between one dependent variable
and one or more independent variables  The goal of regression analysis
is to understand and quantify the relationship between these
variables.
```

Types

There are several types of linear regression models, each suited for different scenarios. Here are the main types:

1. **Simple Linear Regression:**
   - Involves only one independent variable.
   - The relationship between the dependent variable and the independent variable is assumed to be linear.
     - **Equation:** (Y = wx+b)
   - **Example:** Using the hours studied ((X)) and exam scores ((Y)) example from before.
2. **Multiple Linear Regression:**
   - Involves more than one independent variable.
   - The relationship is modeled as a linear combination of the independent variables.
   - **Equation:**= y=(w1x1+w2x2+w3x3+...wnxn)+c example:- **Example:** Predicting a person's salary ((Y)) based on multiple factors like years of experience ((X_1)), education level ((X_2)), and age ((X_3)).
3. **Polynomial Regression:**
   - Extends the linear regression model by considering polynomial relationships.
   - The relationship between the dependent variable and the independent variable is modeled as an nth degree polynomial.
   - **Example:** Predicting house prices ((Y)) based on the square footage ((X)) using a polynomial regression to capture non-linear relationships. Equation: y=w1x1+w2x2^2+w3x3^3+...wnx^n+c
4. **Ridge Regression (L2 Regularization):**
   - Adds a penalty term to the linear regression equation to prevent overfitting.
   - Helps when there is multicollinearity among the independent variables.

5. **Lasso Regression (L1 Regularization):**
   - Similar to Ridge Regression but uses the absolute values of the coefficients.
   - Can be useful for feature selection by driving some coefficients to exactly zero.
6. **Elastic Net Regression:**
   - Combines Ridge and Lasso regression.
   - It has both L1 and L2 regularization terms.
7. **Logistic Regression:**
   - Despite its name, logistic regression is used for classification, not regression.
   - It models the probability that an instance belongs to a particular category.

Diff b/w Ridge and Lasso

Ridge regression and Lasso regression are both variations of linear regression that include regularization terms to prevent overfitting and handle multicollinearity. The main difference between the two lies in the type of regularization they apply:

1. **Ridge Regression (L2 Regularization):**
   - **Regularization Term:** ($\lambda \sum_{i=1}^{n} b_i^2$)
   - **Objective Function:** Minimize the sum of squared differences between predicted and actual values plus the regularization term.
   - **Effect on Coefficients:** Ridge regression tends to shrink the coefficients toward zero, but it rarely sets them exactly to zero.
   - **Use Case:** Ridge regression is useful when there is a high correlation among the independent variables, and you want to prevent multicollinearity.
2. **Lasso Regression (L1 Regularization):**
   - **Regularization Term:** ($\lambda \sum_{i=1}^{n} |b_i|$)
   - **Objective Function:** Minimize the sum of squared differences between predicted and actual values plus the regularization term.
   - **Effect on Coefficients:** Lasso regression not only shrinks the coefficients but also tends to set some of them exactly to zero.
   - **Use Case:** Lasso regression is useful when you have a large number of features, and you want to perform feature selection by eliminating some of them.

In summary, while both Ridge and Lasso regression introduce a penalty term to the linear regression objective function to prevent overfitting, Ridge tends to shrink coefficients towards zero without eliminating them, while Lasso can eliminate some coefficients entirely, effectively performing feature selection. The choice between the two depends on the specific characteristics of the data and the goals of the analysis.

how ridge regrssion can handle multicolinearity?

Ridge regression is a regularization technique used in linear regression to handle multicollinearity, which occurs when two or more independent variables in a regression model are highly correlated. Multicollinearity can lead to unstable coefficient estimates and result in difficulties in interpreting the model.

Ridge regression introduces a regularization term, also known as a penalty term, to the linear regression objective function. The objective function for ridge regression is:

$$ \text{minimize } \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} $$

Here:

- y_i is the observed value for the i-th observation.
- β_0 is the intercept term.
- x_{ij} is the j-th predictor variable for the i-th observation.
- β_j is the coefficient for the j-th predictor variable.
- λ is the regularization parameter that controls the strength of the penalty term.

The additional term $\lambda \sum_{j=1}^{p} \beta_j^2$ penalizes large coefficient values. This penalty helps to shrink the coefficients toward zero, reducing their variance. As a result, ridge regression is particularly effective in mitigating multicollinearity by preventing the model from relying too heavily on any single variable.

By introducing this penalty term, ridge regression improves the conditioning of the problem and provides more stable estimates of the coefficients, even in the presence of multicollinearity. The regularization term helps balance the trade-off between fitting the training data well and keeping the coefficients within reasonable bounds.

what is multicolinearity and example

Multicollinearity is a phenomenon in statistics and regression analysis where two or more independent variables in a regression model are highly correlated. This high correlation can cause issues in estimating the individual coefficients of the variables, leading to unstable and unreliable results. Multicollinearity does not affect the predictive power of the model, but it makes it challenging to interpret the significance of each variable.

Here's a simple example to illustrate multicollinearity:

Suppose you want to predict a person's income (Y) based on two independent variables: years of education (X_1) and years of work experience (X_2). A multicollinearity issue might arise if these two variables are strongly correlated.

Example data:

```
| Person | Education (X1) | Experience (X2) | Income (Y) |
|--------|----------------|-----------------|------------|
|   A    |       16       |        4        |   80,000   |
|   B    |       18       |        6        |   90,000   |
|   C    |       14       |        3        |   75,000   |
|   D    |       16       |        5        |   85,000   |
```

In this example, if education and experience are highly correlated (for instance, people with more education tend to have more work experience), multicollinearity may occur. The issue arises when you try to estimate the coefficients of the regression equation:

[Y = \beta_0 + \beta_1X_1 + \beta_2X_2]

High multicollinearity could make it difficult to determine the true effect of each variable on the outcome (income). The coefficients ((\beta_1) and (\beta_2)) might be unstable, and it becomes challenging to attribute changes in income to changes in either education or experience individually.

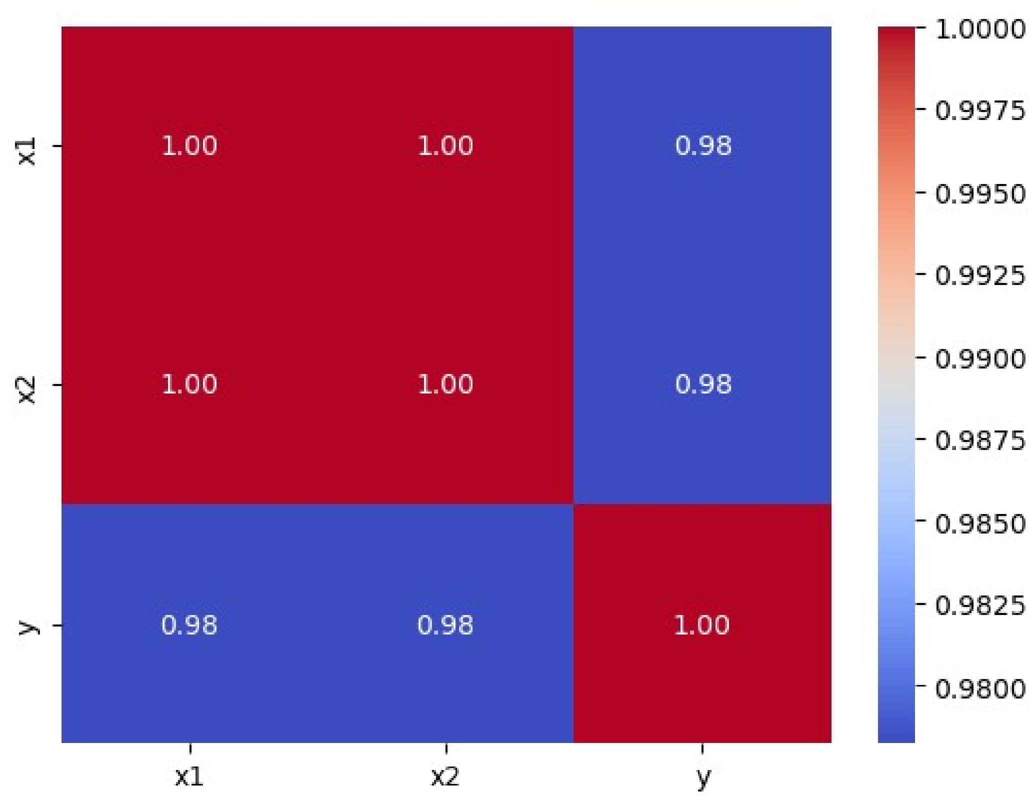
To address multicollinearity, techniques like ridge regression or principal component analysis (PCA) can be employed to stabilize the coefficient estimates and improve the interpretability of the model.

How to find there is multicolinearity in dataset

1.Co relationMatrix

```
!pip install pandas

Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\
site-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\
programdata\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas) (2022.7)
Requirement already satisfied: numpy>=1.21.0 in c:\programdata\
anaconda3\lib\site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\
lib\site-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)

import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt

data={
"x1":[1,2,3,4,5],
```

```
"x2":[2,4,6,8,10],
"y":[3,9,12,15,17]
}

# we have a methods called corr in dataframes,so we have to create or
convert our data into datafrane.
df=pd.DataFrame(data)
corr_matrix=df.corr()
sb.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")

<Axes: >
```



how to handle multicolinearity ?

#1.we can delete any one of the features. #2.we can combine the features and create a new one. #3.we can do nothing just keep all features based on the applicaiton scenario

Ref https://youtu.be/NAPhUDjgG_s?si=dGaWl-U8L57dmURs

PCA(principal component analysis)

PCA used for dimensionality reduction

why we want to do dimensionality reduction?

```
Because we want to protect the model from curse of dimensionality

curse of dimensionality :
Having lot of features will reduce the performance of the model.
because features may or may not be important .
so,we can do :
1.feature selection:
    based on covariance or co relaion matrix b/w the features.
2.feature extraction:
reduce high no features into low no of features.
This is called PCA
```

How PCA works (Pricipal component analysis)

PCA will choose best principal component which has highest variance or spread of the data using eigen values and vectors.

PCA will apply eigen decomposition matrix to the dataset and it will find the best features which has more variance

Mathmetical explanation

Dataset -> each value will apply it into linear equation for random initial slope value ->predicted value will come -> -> we will calculate error b/w actual and predicted its called cost function -> we want to reduce the cost function -> -> we will try gradient descent algorithm ->it will reduce the cost function by trying different value of slope by using convergence therom -> it will converge until derivative of slope will equal to zero.->then we will take that particular slope and intercept values-> and we can use it for new data

Progrmatical explanation for each and every type of regression

Ref: https://www.geeksforgeeks.org/python-linear-regression-using-sklearn/

Ref :https://www.kaggle.com/code/emrearslan123/house-price-prediction

```
import pandas as pd
import seaborn as sns

df=pd.read_csv("dataset/house.csv")

df.head()

   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave   NaN      Reg
```

```
1   2           20        RL           80.0     9600   Pave   NaN      Reg

2   3           60        RL           68.0    11250   Pave   NaN      IR1

3   4           70        RL           60.0     9550   Pave   NaN      IR1

4   5           60        RL           84.0    14260   Pave   NaN      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

```
df.shape
# 1460=row,81=columns
```

```
(1460, 81)
```

```
df.info()
# it gives datatypes and null or not null informations for each
feature
```

```
<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
```

```
df.describe()
# it will give mean,std,min,max,the data point which is 25%of the
whole dataset,50% datapoint,75% data point
```

```
                Id   MSSubClass  LotFrontage        LotArea
OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000
1460.000000
mean    730.500000    56.897260    70.049958   10516.828082
6.099315
std     421.610009    42.300571    24.284752    9981.264932
1.382997
min       1.000000    20.000000    21.000000    1300.000000
1.000000
25%     365.750000    20.000000    59.000000    7553.500000
5.000000
50%     730.500000    50.000000    69.000000    9478.500000
6.000000
75%    1095.250000    70.000000    80.000000   11601.500000
7.000000
max    1460.000000   190.000000   313.000000  215245.000000
```

```
10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea
BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000
1460.000000  ...
mean      5.575342  1971.267808   1984.865753   103.685262
443.639726  ...
std       1.112799    30.202904     20.645407   181.066207
456.098091  ...
min       1.000000  1872.000000   1950.000000     0.000000
0.000000  ...
25%       5.000000  1954.000000   1967.000000     0.000000
0.000000  ...
50%       5.000000  1973.000000   1994.000000     0.000000
383.500000  ...
75%       6.000000  2000.000000   2004.000000   166.000000
712.250000  ...
max       9.000000  2010.000000   2010.000000  1600.000000
5644.000000  ...

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch
ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000
1460.000000
mean     94.244521    46.660274      21.954110     3.409589
15.060959
std     125.338794    66.256028      61.119149    29.317331
55.757415
min       0.000000     0.000000       0.000000     0.000000
0.000000
25%       0.000000     0.000000       0.000000     0.000000
0.000000
50%       0.000000    25.000000       0.000000     0.000000
0.000000
75%     168.000000    68.000000       0.000000     0.000000
0.000000
max     857.000000   547.000000     552.000000   508.000000
480.000000

          PoolArea      MiscVal       MoSold       YrSold
SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000
1460.000000
mean      2.758904     43.489041     6.321918  2007.815753
180921.195890
std      40.177307    496.123024     2.703626     1.328095
79442.502883
min       0.000000      0.000000     1.000000  2006.000000
34900.000000
```

```
25%        0.000000      0.000000     5.000000  2007.000000
129975.000000
50%        0.000000      0.000000     6.000000  2008.000000
163000.000000
75%        0.000000      0.000000     8.000000  2009.000000
214000.000000
max      738.000000  15500.000000    12.000000  2010.000000
755000.000000

[8 rows x 38 columns]
```

```
corr=df.corr()
```

```
C:\Users\shan\AppData\Local\Temp\ipykernel_11712\953174619.py:1:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this
warning.
  corr=df.corr()
```

```
corr_matrix
```

```
         x1       x2        y
x1  1.00000  1.00000  0.97824
x2  1.00000  1.00000  0.97824
y   0.97824  0.97824  1.00000
```

Example for Simple linear regression

```
# i selected overallqual is good feature for simple linear regression
because based on sales price other features co relation ,
# overallqual has 0.75 value with output
df_simple=df[["OverallQual","SalePrice"]]
df_simple
```

```
      OverallQual  SalePrice
0               7     208500
1               6     181500
2               7     223500
3               7     140000
4               8     250000
...           ...        ...
1455            6     175000
1456            6     210000
```

```
1457            7     266500
1458            5     142125
1459            5     147500

[1460 rows x 2 columns]
```

```
df_simple.head()
```

```
   OverallQual  SalePrice
0            7     208500
1            6     181500
2            7     223500
3            7     140000
4            8     250000
```

```
df_simple.describe()
```

```
       OverallQual      SalePrice
count  1460.000000    1460.000000
mean      6.099315  180921.195890
std       1.382997   79442.502883
min       1.000000   34900.000000
25%       5.000000  129975.000000
50%       6.000000  163000.000000
75%       7.000000  214000.000000
max      10.000000  755000.000000
```

```
df_simple.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   OverallQual  1460 non-null   int64
 1   SalePrice    1460 non-null   int64
dtypes: int64(2)
memory usage: 22.9 KB
```

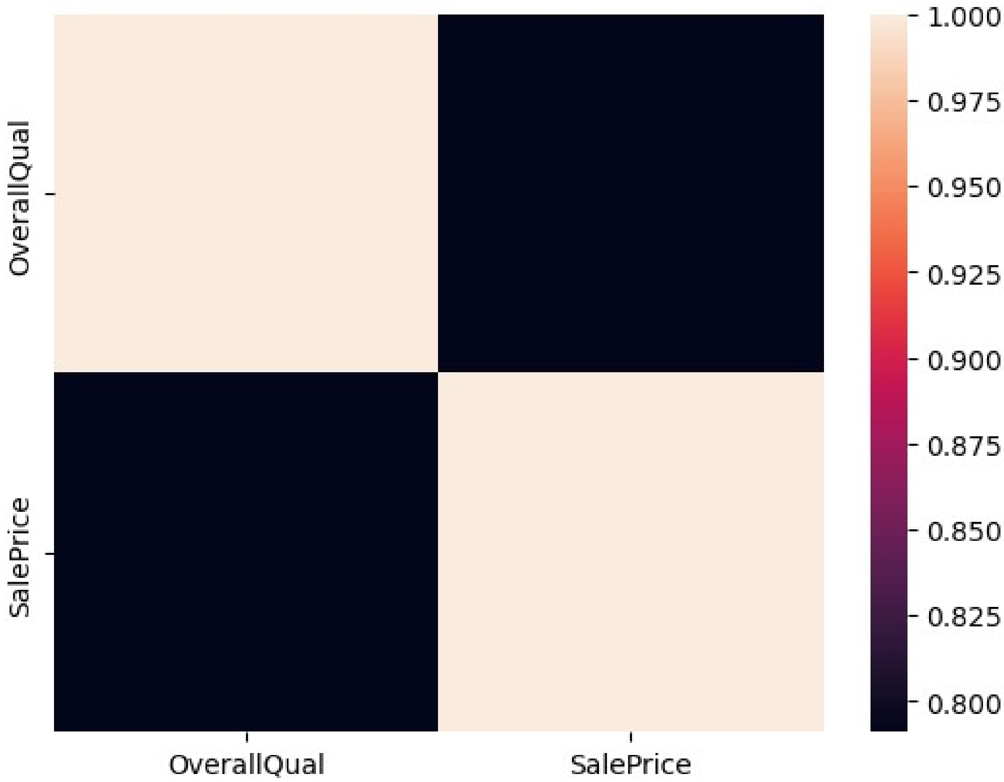
Find relationship b/w features by visulaization

```
df_simple.corr()
```

```
             OverallQual  SalePrice
OverallQual     1.000000   0.790982
SalePrice       0.790982   1.000000
```

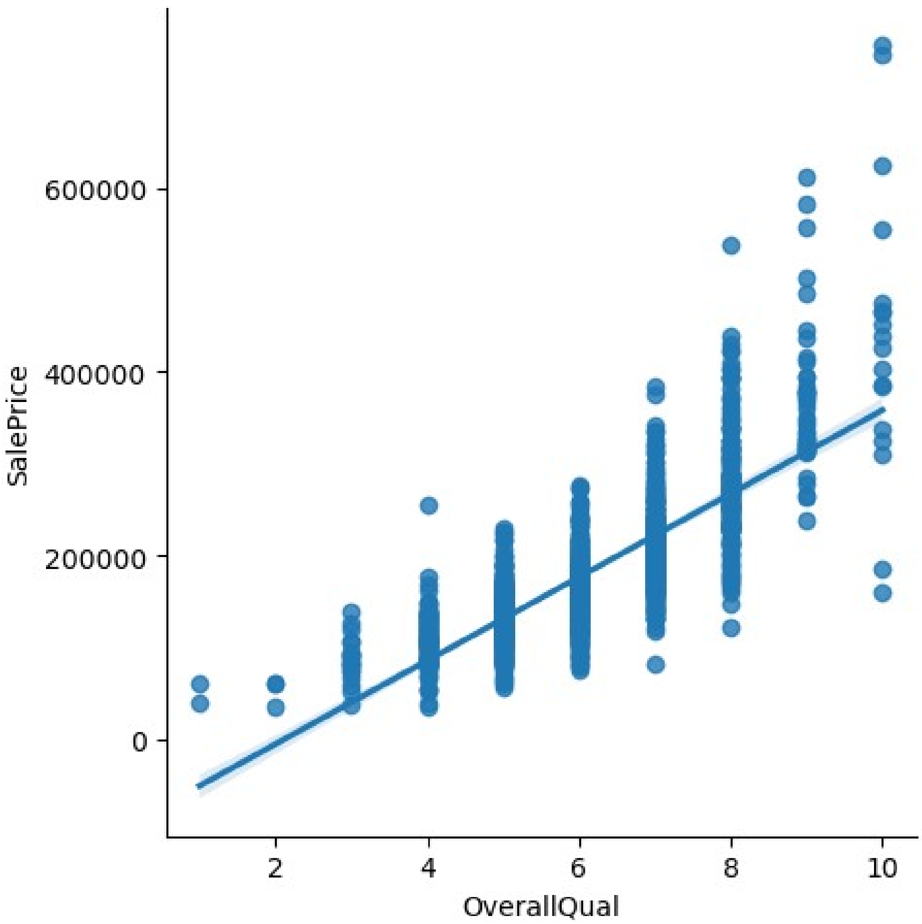
```
sb.heatmap(df_simple.corr())
```

```
<Axes: >
```




```
sns.lmplot(x="OverallQual",y="SalePrice",data=df_simple)
```

```
<seaborn.axisgrid.FacetGrid at 0x2431dfee710>
```




```
df_simple

      OverallQual  SalePrice
0               7     208500
1               6     181500
2               7     223500
3               7     140000
4               8     250000
...           ...        ...
1455            6     175000
1456            6     210000
1457            7     266500
1458            5     142125
1459            5     147500

[1460 rows x 2 columns]
```

Feature selection :

```
we have to choose the independent feature which has more corelation
with dependent feature ,we can check it by correlation matrix,but here
we do not have to do that because here we have only one feature.
```

we need to choose features based on two factors:

```
1.remove multicolinearity b/w independent features
2.features which has good relation(it may be +ve,-ve) with output.
```

Handling missing or null values

we can do it by 2 methods

There are 2 primary ways of handling missing values:

```
 1.Deleting the Missing values
2.Imputing the Missing Values
    1.Replacing with an arbitrary value(random)
    2.Replacing with the mean
    3.Replacing with mode
    4.Replacing with the median
    5.Replacing with the previous value – forward fill
    6.Replacing with the next value – backward fill
    7.Interpolation
        -Missing values can also be imputed using interpolation.
Pandas' interpolate method can be used to replace the missing values
with different interpolation methods like 'polynomial,' 'linear,' and
'quadratic.' The default method is 'linear.'
    Interpolation is a technique of constructing data points between
given data points.




3.use missingness as a feature
```

Ref: https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/

Deleting the Missing value Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values. If the missing value is of the type Missing Not At Random (MNAR), then it should not be deleted.

If the missing value is of type Missing At Random (MAR) or Missing Completely At Random (MCAR) then it can be deleted (In the analysis, all cases with available data are utilized, while missing observations are assumed to be completely random (MCAR) and addressed through pairwise deletion.)

```

```

Train /test data split

```
from sklearn.model_selection import train_test_split
```

Normalization or feature scaling

ref-https://www.javatpoint.com/normalization-in-machine-learning

While many machine learning algorithms benefit from feature scaling, some algorithms are inherently less sensitive to the scale of input features. Here are a few machine learning algorithms that generally don't require feature scaling:

1. **Decision Trees and Random Forests:**
   - Decision trees and random forests make decisions based on feature thresholds, and the scale of features does not affect their performance significantly.
2. **Naive Bayes:**
   - Naive Bayes algorithms, such as Gaussian Naive Bayes, are probabilistic models that assume independence between features. The scale of individual features doesn't impact the classification decisions.
3. **Gradient Boosting Machines (e.g., XGBoost, LightGBM):**
   - Gradient boosting algorithms are robust to the scale of features due to their ensemble nature. They build trees sequentially, and the scale of features doesn't affect the model's performance.
4. **Support Vector Machines (SVM) with Linear Kernel:**
   - Support Vector Machines can be sensitive to feature scaling, but if a linear kernel is used, the decision boundaries depend on the dot product of feature vectors, making them less sensitive to scaling.
5. **K-Nearest Neighbors (KNN):**
   - KNN considers distances between data points, and while scaling can affect the distance metrics, it's not strictly required for KNN to work. However, it might still be beneficial in certain cases.

It's important to note that while these algorithms may be less sensitive to feature scaling, scaling can still improve their performance or convergence speed in some situations. Additionally, if you are using distance-based metrics or regularization in your model, scaling may still be beneficial.

For most other algorithms, especially those that involve distance calculations or optimization processes (like gradient descent), it's generally a good practice to scale your features. Always consider the specific requirements and characteristics of your chosen algorithm and dataset.

```
import numpy as np

# so now i need to choose which one is better
#here mean !=0,std approximatly =1 for overall qual but not in sale price
# i can see there is no much outlier
#so i am going to implement min max scaler
# df_simple.describe()

#split data into dependent and independent:
X=df_simple["OverallQual"]
Y=df_simple["SalePrice"]
# we need to change 1d to 2d because sklearn will expect 2d
X=np.array(X).reshape(-1,1)
Y=np.array(Y).reshape(-1,1)

from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25)
```

Min Max scaler implementation

```
from sklearn.preprocessing import MinMaxScaler
```

what is diff b/w fit and fit_transform

fit will calculate the parameters like min,max,mean,std ex:In min max scaler fit will calculate min,max values of dataset.

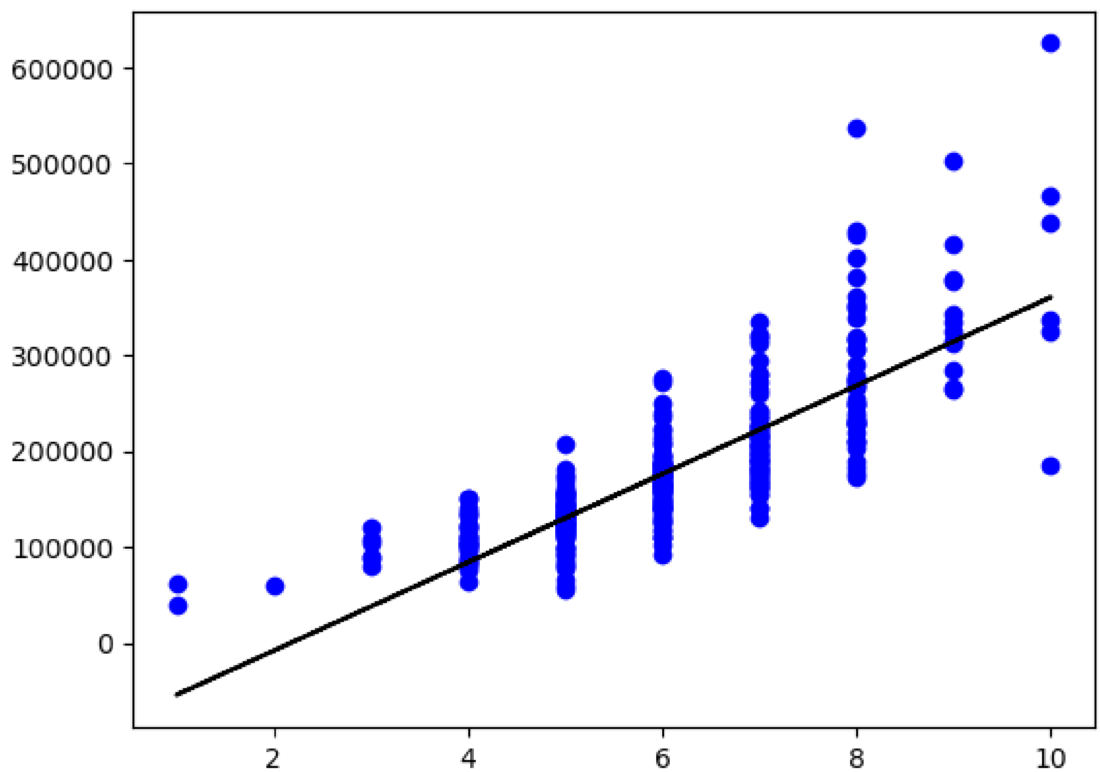
fit_transform will compute necessary parameters and apply it into the dataset

```
min_max_object=MinMaxScaler()
min_max_object

MinMaxScaler()

min_max_object.fit(X_train)

MinMaxScaler()

X_train_normalized=min_max_object.transform(X_train)

X_train_normalized.size

1095

X_test_normalized=min_max_object.transform(X_test)
X_test_normalized.size

365
```

Create Linear regression model

```
from sklearn.linear_model import LinearRegression

lm=LinearRegression()

lm.fit(X_train,Y_train)

LinearRegression()

lm.score(X_test,Y_test)
```

```
0.636088834253365

Y_pred=lm.predict(X_test)

lm.intercept_

array([-100453.29432642])




y_pred = lm.predict(X_test)
plt.scatter(X_test,Y_test, color ='b')
plt.plot(X_test, Y_pred, color ='k')

plt.show()
# Data scatter of predicted values
```



Evaluation of the model

```
from sklearn.metrics import mean_absolute_error,mean_squared_error

mae = mean_absolute_error(y_true=Y_test,y_pred=Y_pred)
#squared True returns MSE value, False returns RMSE value.
```

```
mse = mean_squared_error(y_true=Y_test,y_pred=Y_pred) #default=True
rmse = mean_squared_error(y_true=Y_test,y_pred=Y_pred,squared=False)

print("MAE:",mae)
print("MSE:",mse)
print("RMSE:",rmse)
```

```
MAE: 33838.261473512866
MSE: 2350065159.6346955
RMSE: 48477.47063982088
```

Reducing the error

1.Cross validatation

Ref: https://www.youtube.com/watch?v=3fzYdnuvEfk

Ref :https://www.javatpoint.com/cross-validation-in-machine-learning

cross validation method used to find best parameters for the different type of train,validation dataset using train dataset.

Types:

```
1.Leave one out cross validation
2.P one out cross validation
3.K fold- cross validation
4.startified cross validation
```

Leave one out cross-validation:

```
 we need to take 1 dataset out of training. It means, in this
approach, for each learning set, only one datapoint is reserved, and
the remaining dataset is used to train the model. This process repeats
for each datapoint. Hence for n samples, we get n different training
set and n test set. It has the following features:
```

In this approach, the bias is minimum as all the data points are used. The process is executed for n times; hence execution time is high. This approach leads to high variation in testing the effectiveness of the model as we iteratively check against one data point.

Leave-P-out cross-validation

In this approach, the p datasets are left out of the training data. It means, if there are total n datapoints in the original input dataset, then n-p data points will be used as the training dataset and the p data points as the validation set. This complete process is repeated for all the samples, and the average error is calculated to know the effectiveness of the model.

There is a disadvantage of this technique; that is, it can be computationally difficult for the large p.

The steps for k-fold cross-validation are:

Split the input dataset into K groups For each group: Take one group as the reserve or test data set. Use remaining groups as the training dataset Fit the model on the training set and evaluate the performance of the model using the test set. Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1st iteration, the first fold is reserved for test the model, and rest are used to train the model. On 2nd iteration, the second fold is used to test the model, and rest are used to train the model. This process will continue until each fold is not used for the test fold.

Consider the below diagram:

Cross-Validation in Machine Learning

Stratified k-fold cross-validation

This technique is similar to k-fold cross-validation with some little changes. This approach works on stratification concept, it is a process of rearranging the data to ensure that each fold or group is a good representative of the complete dataset. To deal with the bias and variance, it is one of the best approaches.

It can be understood with an example of housing prices, such that the price of some houses can be much high than other houses. To tackle such situations, a stratified k-fold cross-validation technique is useful.

2.Hyperparameter tuning

Ref :https://www.youtube.com/watch?v=355u2bDqB7c

Find the best fit parameters of the model by using Cross validation

1.Grid serach cv -for all combinations of parameters ex.penalty,solver in logistic regression

2.Randomized search cv-for selective combinations we can try build the model

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression

# Assuming X_train, y_train are your training data and labels

# Create a simple linear regression model
model = LinearRegression()

# Define the hyperparameter grid (empty for simple linear regression)
param_grid = {}

# Use GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_absolute_error')
grid_search.fit(X_train, Y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Train the final model with the best hyperparameters on the entire
training set
final_model = grid_search.best_estimator_
final_model.fit(X_train, Y_train)
```

```
Best Hyperparameters: {}
```

```
LinearRegression()
```

For simple linear regression we dont have much parameters to fine tune ,,so i am going to try to cut half of the data

```
from sklearn.preprocessing import MinMaxScaler

X_train_normalized=X_train_normalized[0:500]
X_train_normalized.shape

(500, 1)

Y_train=Y_train[0:500]
Y_train.shape

(500, 1)

from sklearn.linear_model import LinearRegression

lm_2=LinearRegression()

lm_2.fit(X_train_normalized,Y_train)

LinearRegression()

Y_predict=lm_2.predict(X_test_normalized)

lm_2.score(X_train_normalized,Y_train)

0.630604334696119

from sklearn.metrics import mean_squared_error

mean_squared_error(Y_test,Y_predict)

2348897526.0790462
```

trying normalize result value will give better results?

lets try ? even if its give better results.after we trying to predict new input will it give normalized value ,in real time it will not look like right,so ,we need to revise the normalized operation.so,it wont work.

Yes, denormalization is a concept that can be applied in machine learning, particularly when working with normalized data. Normalization is the process of scaling and transforming numerical features to a standard range, often between 0 and 1. Denormalization, on the other hand, involves reversing this process, restoring the original scale of the data.

Here are some scenarios where denormalization might be relevant in machine learning:

1. **Interpretability:**
   - If you've normalized your features for model training to aid convergence or improve performance, you might want to denormalize them for better interpretability of the model results. This is especially important when presenting the results to stakeholders who are more familiar with the original scale of the data.
2. **Prediction Outputs:**
   - When making predictions using a model trained on normalized data, you'll need to denormalize the predicted outputs to obtain predictions in the original scale. This is crucial for making practical use of your model's predictions.
3. **Visualization:**
   - If you want to visualize the relationships between features or explore the data, denormalization can be helpful. Plots and visualizations are more intuitive when the data is in its original scale.
4. **External Integration:**
   - If your model is part of a larger system that interacts with external components or databases where data is stored in its original scale, denormalization is necessary for seamless integration.

Here's a simplified example in Python to illustrate denormalization:

```python
# Assuming 'normalized_data' is a DataFrame with normalized features
# 'min_values' and 'max_values' are the minimum and maximum values of
each feature before normalization

# Denormalization function
def denormalize(data, min_values, max_values):
    denormalized_data = data * (max_values - min_values) + min_values
    return denormalized_data

# Apply denormalization to a DataFrame of normalized features
denormalized_features = denormalize(normalized_data, min_values,
max_values)
```

Keep in mind that denormalization is not always necessary, and it depends on the context of your specific machine learning problem and how you intend to use the results.

Multiple linear Regression

```
# First u have to choose 3 features which is more co related with
output feature
```

```
df=pd.read_csv("dataset/house.csv")
```

```
df.corr()
# OverallQual-0.79 corelation with output feature
# GrLivArea -0.70
# GarageCars-0.68
data=df[["OverallQual","GrLivArea","GarageCars","SalePrice"]]
X=df[["OverallQual","GrLivArea","GarageCars"]]
Y=df[["SalePrice"]]
```

```
C:\Users\shan\AppData\Local\Temp\ipykernel_11712\2493397992.py:1:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this
warning.
  df.corr()
```

```
X.head()
```

	OverallQual	GrLivArea	GarageCars
0	7	1710	2
1	6	1262	2
2	7	1786	2
3	7	1717	3
4	8	2198	3

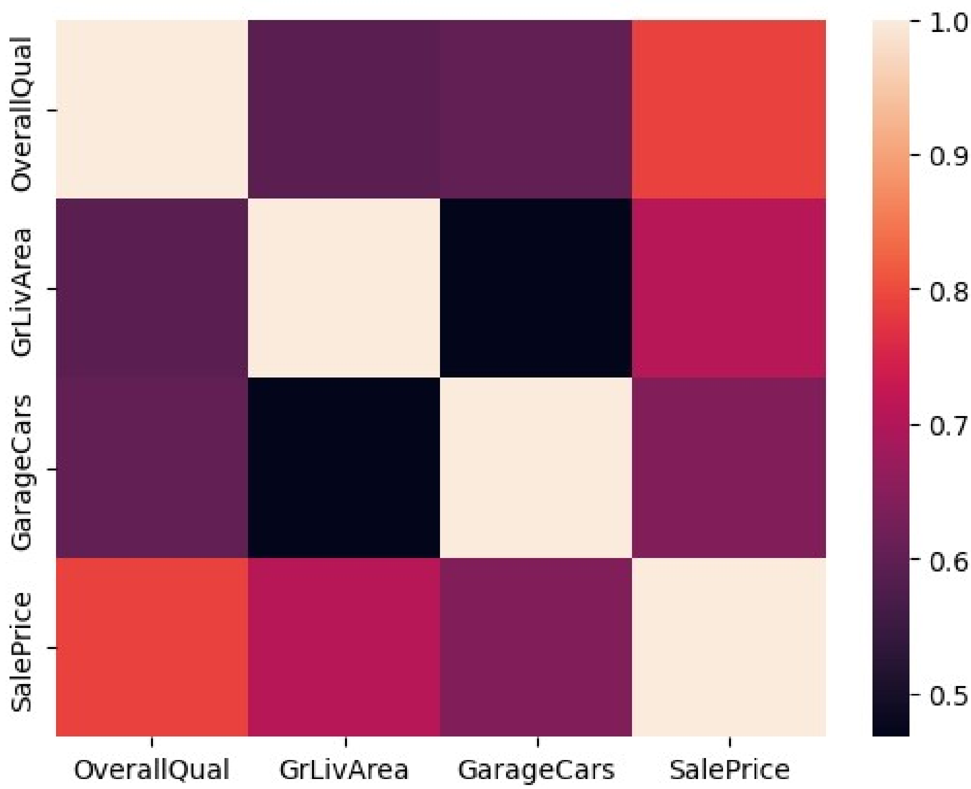
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   OverallQual  1460 non-null   int64
 1   GrLivArea    1460 non-null   int64
 2   GarageCars   1460 non-null   int64
dtypes: int64(3)
memory usage: 34.3 KB
```

```
X.describe()
```

	OverallQual	GrLivArea	GarageCars
count	1460.000000	1460.000000	1460.000000
mean	6.099315	1515.463699	1.767123
std	1.382997	525.480383	0.747315
min	1.000000	334.000000	0.000000

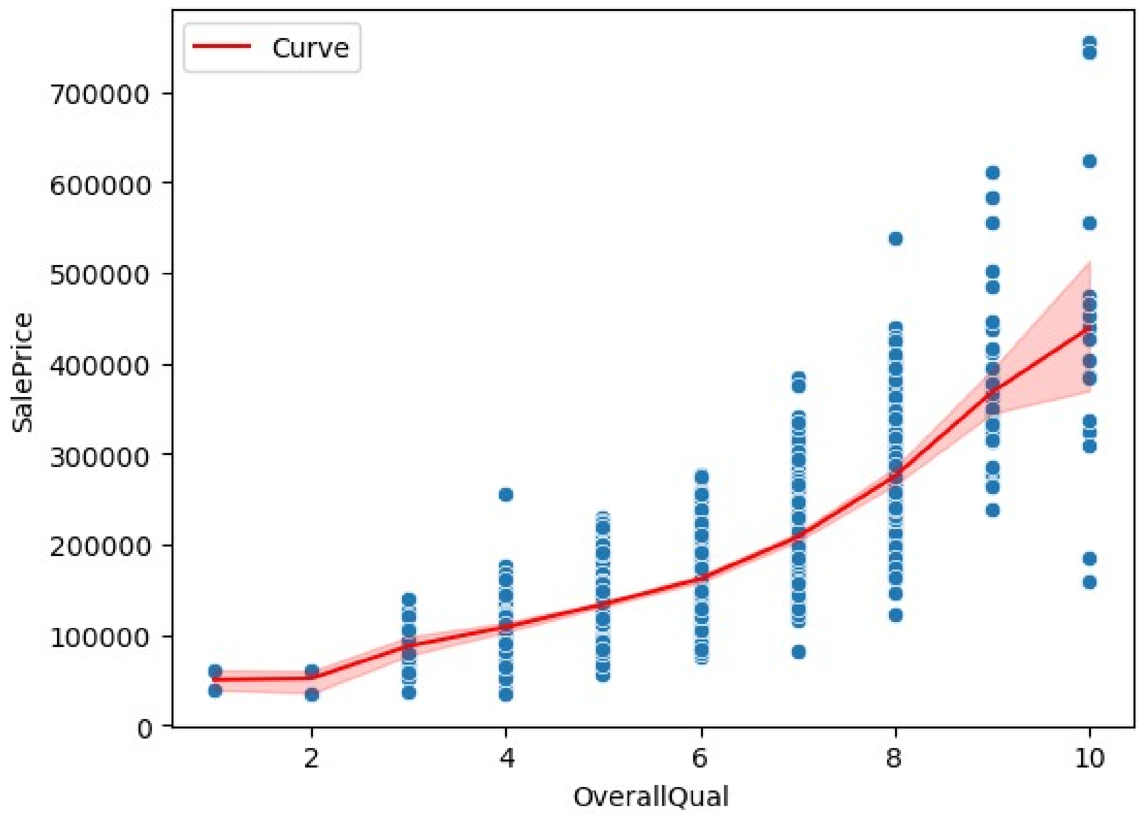
```
25%       5.000000  1129.500000     1.000000
50%       6.000000  1464.000000     2.000000
75%       7.000000  1776.750000     2.000000
max      10.000000  5642.000000     4.000000



import seaborn as sb

# we need to understand relationship b/w both dependent and independet
variables

corr_poly=data.corr()
sb.heatmap(corr_poly)
print(corr_poly)

             OverallQual  GrLivArea  GarageCars  SalePrice
OverallQual     1.000000   0.593007    0.600671   0.790982
GrLivArea       0.593007   1.000000    0.467247   0.708624
GarageCars      0.600671   0.467247    1.000000   0.640409
SalePrice       0.790982   0.708624    0.640409   1.000000
```



we have to find the distribution of the dataset

```
sns.scatterplot(data=data,x="OverallQual",y="SalePrice")
sns.lineplot(x='OverallQual', y='SalePrice', data=data, color='red',
label='Curve')
```

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




```
# GrLivArea -0.70
# GarageCars-0.68
df
```

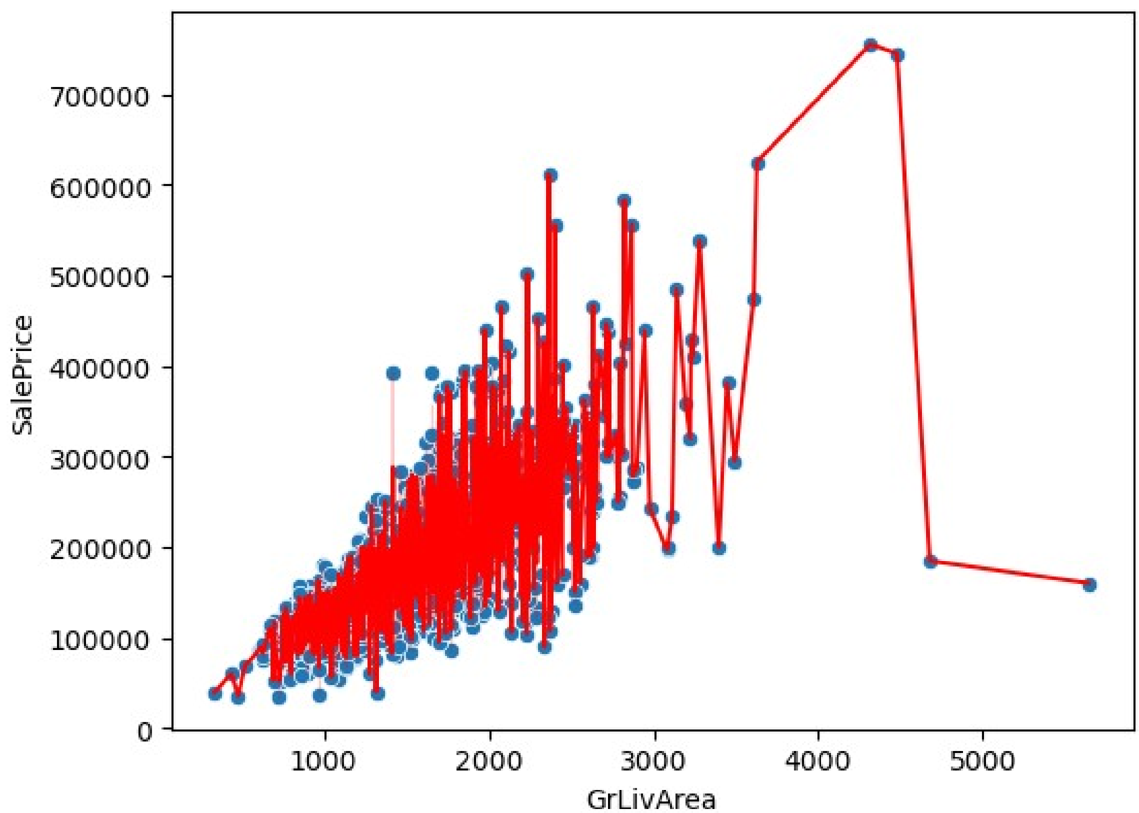
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley
LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN
Reg
1   2          20       RL         80.0     9600   Pave   NaN
Reg
2   3          60       RL         68.0    11250   Pave   NaN
IR1
3   4          70       RL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

```
...    ...          ...      ...           ...      ...    ...   ...
...
1455  1456           60       RL          62.0     7917   Pave   NaN
Reg
1456  1457           20       RL          85.0    13175   Pave   NaN
Reg
1457  1458           70       RL          66.0     9042   Pave   NaN
Reg
1458  1459           20       RL          68.0     9717   Pave   NaN
Reg
1459  1460           20       RL          75.0     9937   Pave   NaN
Reg

     LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature
MiscVal  \
0            Lvl    AllPub  ...        0    NaN    NaN         NaN
0
1            Lvl    AllPub  ...        0    NaN    NaN         NaN
0
2            Lvl    AllPub  ...        0    NaN    NaN         NaN
0
3            Lvl    AllPub  ...        0    NaN    NaN         NaN
0
4            Lvl    AllPub  ...        0    NaN    NaN         NaN
0
...          ...       ...  ...      ...    ...    ...         ...
...
1455         Lvl    AllPub  ...        0    NaN    NaN         NaN
0
1456         Lvl    AllPub  ...        0    NaN  MnPrv         NaN
0
1457         Lvl    AllPub  ...        0    NaN  GdPrv        Shed
2500
1458         Lvl    AllPub  ...        0    NaN    NaN         NaN
0
1459         Lvl    AllPub  ...        0    NaN    NaN         NaN
0

     MoSold YrSold  SaleType  SaleCondition  SalePrice
0         2   2008        WD         Normal     208500
1         5   2007        WD         Normal     181500
2         9   2008        WD         Normal     223500
3         2   2006        WD        Abnorml     140000
4        12   2008        WD         Normal     250000
...     ...    ...       ...            ...        ...
1455      8   2007        WD         Normal     175000
1456      2   2010        WD         Normal     210000
1457      5   2010        WD         Normal     266500
1458      4   2010        WD         Normal     142125
```

```
1459       6   2008         WD          Normal     147500

[1460 rows x 81 columns]
```

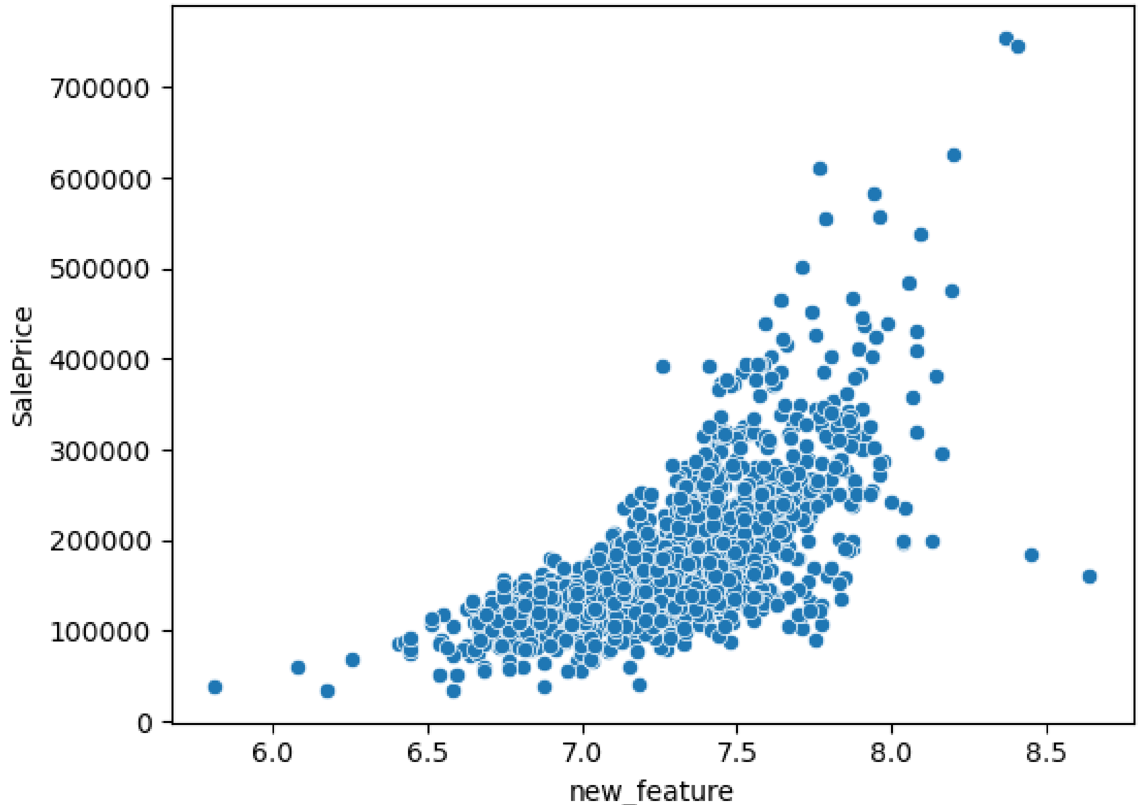
```python
sns.scatterplot(data=df,x="GrLivArea",y="SalePrice")
sns.lineplot(x='GrLivArea', y='SalePrice', data=df, color='red')
# sns.lmplot(x='GrLivArea', y='SalePrice', data=data)
```

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

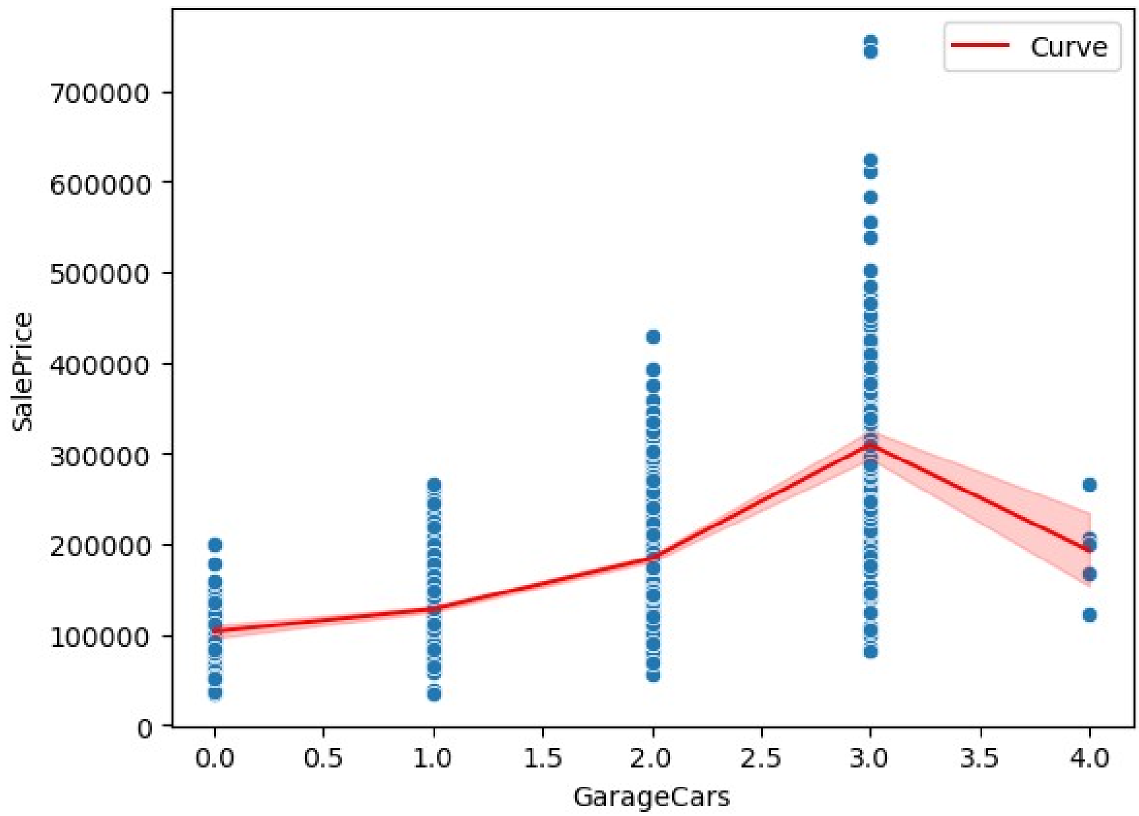



```python
#Above GR liv area not distributed linearly,so we have to do some transformation

df["new_feature"]=np.log(df["GrLivArea"])

sns.scatterplot(data=df,x="new_feature",y="SalePrice")
# sns.lineplot(x='new_feature', y='SalePrice', data=df, color='red')
```

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




```
sns.scatterplot(data=df,x="GarageCars",y="SalePrice")
sns.lineplot(x='GarageCars', y='SalePrice', data=df, color='red',
label='Curve')

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



If the distribution of your dataset doesn't appear linear, and you still want to perform linear regression, you might consider the following strategies:

If the distribution of your dataset doesn't appear linear, and you still want to perform linear regression, you might consider the following strategies:

1. **Transformations:**

- Apply transformations to the variables to make the relationship more linear. Common transformations include logarithmic, square root, or reciprocal transformations. This can help handle situations where the relationship is non-linear in the original scale.

```
import numpy as np

# Example: Logarithmic transformation
df['transformed_feature'] = np.log(df['original_feature'])
```

2. **Polynomial Regression:**

- Instead of fitting a straight line, you can fit a polynomial function to your data. This involves introducing higher-order terms (e.g., quadratic or cubic terms) into the regression model.

```
import numpy as np
import statsmodels.api as sm
```

```
# Example: Quadratic regression
df['feature_squared'] = df['feature'] ** 2
X = sm.add_constant(df[['feature', 'feature_squared']])
model = sm.OLS(df['target'], X).fit()
```

3. **Feature Engineering:**

- Create new features based on domain knowledge or experimentation to capture non-linear relationships. This might involve combining existing features or creating interaction terms.

```
# Example: Interaction term
df['interaction_term'] = df['feature1'] * df['feature2']
```

4. **Non-Linear Models:**

- If linear regression is not suitable for your data, consider using non-linear regression models such as decision trees, random forests, or support vector machines.

5. **Check Assumptions:**

- Ensure that the assumptions of linear regression are met. If the residuals (the differences between predicted and observed values) show a pattern or non-constant variance, it might indicate a violation of assumptions.

```
# Example: Residual plot
residuals = model.resid
plt.scatter(df['feature'], residuals)
```

6. **Data Exploration:**

- Explore the data visually using scatter plots, histograms, or other visualizations to understand the underlying patterns. This might guide you in choosing the right approach.

Remember, the choice between these strategies depends on the specific characteristics of your data and the goals of your analysis. It's crucial to evaluate the performance of different approaches and choose the one that best captures the underlying relationship in your dataset.

```


#Data Cleaning
# data visulazation
#Feature selection (multicoliearity) ,Feature extraction/reduction
# handling missing values
# normalization
# train
# predict
# evluate
# cross validate
# hyper parameter tunning
# evaluate
```

```
X.head()
```

```
   OverallQual  GrLivArea  GarageCars
0            7       1710           2
1            6       1262           2
2            7       1786           2
3            7       1717           3
4            8       2198           3
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   OverallQual  1460 non-null   int64
 1   GrLivArea    1460 non-null   int64
 2   GarageCars   1460 non-null   int64
dtypes: int64(3)
memory usage: 34.3 KB
```

```
X.describe()
```

```
       OverallQual    GrLivArea   GarageCars
count  1460.000000  1460.000000  1460.000000
mean      6.099315  1515.463699     1.767123
std       1.382997   525.480383     0.747315
min       1.000000   334.000000     0.000000
25%       5.000000  1129.500000     1.000000
50%       6.000000  1464.000000     2.000000
75%       7.000000  1776.750000     2.000000
max      10.000000  5642.000000     4.000000
```

Normalization

```
1.MIn max scaler - based on min,max values,range 0,1 or -1,1
2.Z score standardization -mean,std. helpful when we have
mean=0,variance=1 ,no range
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

```
# train /test split
# test should not know abour train data information ,so  i have to
split train,test first

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25)

norm=MinMaxScaler()

norm.fit(X_train)

MinMaxScaler()

X_train=norm.transform(X_train)

X_test=norm.transform(X_test)
```

training

```
from sklearn.linear_model import LinearRegression

LM=LinearRegression()

LM.fit(X_train,Y_train)

LinearRegression()

Y_pred=LM.predict(X_test)

LM.score(X_train,Y_train)

0.7464502716628104
```

Evaluate

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

mean_absolute_error(Y_test,Y_pred)

27209.76315085111

mean_squared_error(Y_test,Y_pred,squared=False)
# root of mean squared

40289.83466000029

mean_squared_error(Y_test,Y_pred)
```

```
1623270776.930161
```

Reducing Mean Squared Error (MSE) in multiple linear regression involves improving the model's predictive accuracy. Here are some strategies to achieve that:

1. **Feature Selection:**
   - Identify and select relevant features. Remove any irrelevant or highly correlated features that do not contribute significantly to the model.
2. **Data Scaling:**
   - Standardize or normalize the numerical features to ensure that all variables are on a similar scale. This helps prevent certain features from dominating the others.
3. **Outlier Removal:**
   - Identify and handle outliers in the dataset, as they can have a significant impact on the regression model and contribute to higher MSE.
4. **Polynomial Features:**
   - Consider adding polynomial features for nonlinear relationships between predictors and the target variable. This can capture more complex patterns in the data.
5. **Regularization:**
   - Apply regularization techniques like L1 (Lasso) or L2 (Ridge) regularization to prevent overfitting and improve the model's generalization.
6. **Cross-Validation:**
   - Use cross-validation techniques, such as k-fold cross-validation, to assess the model's performance on different subsets of the data. This helps ensure the model's robustness and generalizability.
7. **Increase Sample Size:**
   - A larger dataset can often lead to a more accurate model. If possible, collect more data to improve the model's training.
8. **Check Assumptions:**
   - Ensure that the assumptions of multiple linear regression are met. This includes checking for linearity, independence of errors, homoscedasticity, and normality of residuals.
9. **Model Complexity:**
   - Avoid overly complex models. Balance the trade-off between bias and variance, and choose a model that fits the data well without overfitting.
10. **Feature Engineering:**
    - Create new features based on domain knowledge that might enhance the model's performance.
11. **Hyperparameter Tuning:**
    - Fine-tune the hyperparameters of the regression algorithm. Grid search or random search can be used to find optimal parameter values.

Implementing these strategies should help you reduce MSE in multiple linear regression. It's often a combination of these techniques that leads to the most effective model improvement.

```
Y.describe()
```

```
            SalePrice
count    1460.000000
mean   180921.195890
std     79442.502883
min     34900.000000
25%    129975.000000
50%    163000.000000
75%    214000.000000
max    755000.000000
```

```
Y_pred.shape
```

```
(365, 1)
```

```
Y_pred=Y_pred.flatten()
Y_pred.shape
```

```
(365,)
```

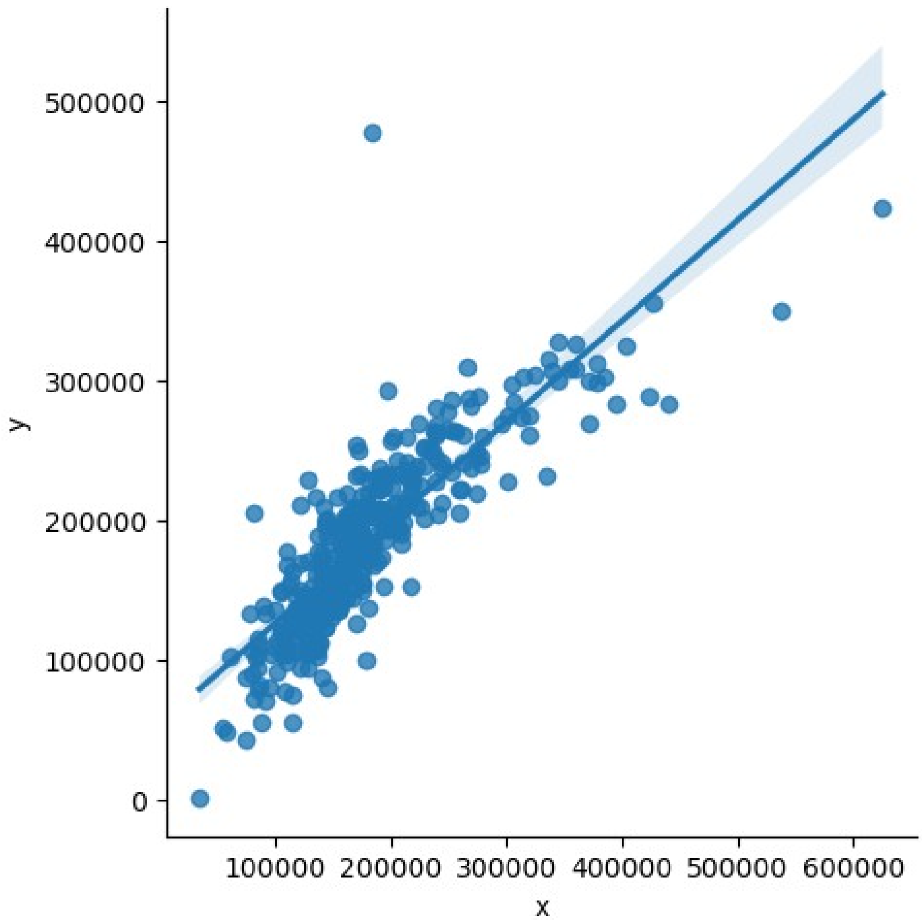
```
Y_test=np.array(Y_test)
```

```
Y_test=Y_test.flatten()
```

```
df3=pd.DataFrame({'x':Y_test,'y':Y_pred})
```

```
sns.lmplot(x='x',y='y',data=df3[0:500])
```

```
<seaborn.axisgrid.FacetGrid at 0x2431fc39e50>
```



HYper parameter tuning -not works

So,i am going to try Polynomial regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly=PolynomialFeatures()

X_poly=poly.fit_transform(X_train)
X_test_poly=poly.fit_transform(X_test)

lm2=LinearRegression()

lm2.fit(X_poly,Y_train)

LinearRegression()

lm2.score(X_poly,Y_train)

0.8138478298518979
```

```
Y_pred=lm2.predict(X_poly)

from sklearn.metrics import r2_score


r2_score(Y_train,Y_pred)

0.8138478298518979
```

Ridge regression

```
from sklearn.linear_model import Ridge

model=Ridge()

model.fit(X_train,Y_train)

Ridge()

model.score(X_train,Y_train)

0.7455314602763969

Y_pred2=model.predict(X_train)

mean_absolute_error(Y_train,Y_pred2)

27477.829890240388
```

Hyper parametr tunning for Ridge

```
from sklearn.model_selection import GridSearchCV

model3=Ridge()

parameters = {'alpha':[1, 10]}
grid=GridSearchCV(model3,parameters,scoring='neg_mean_squared_error',c
v=5)

grid.fit(X_train,Y_train)

GridSearchCV(cv=5, estimator=Ridge(), param_grid={'alpha': [1, 10]},
             scoring='neg_mean_squared_error')

grid.best_estimator_
```

```
Ridge(alpha=1)
```

```
best_model = grid.best_estimator_
best_model.fit(X_train,Y_train)
```

```
Ridge(alpha=1)
```

```
best_model.score(X_train,Y_train)
```

```
0.7455314602763969
```

Tuning Lasso Hyperparameters

```
# grid search hyperparameters for lasso regression
from numpy import arange
from pandas import read_csv
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Lasso

model = Lasso()
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
print(cv)
# define grid
grid = dict()
grid['alpha'] = arange(0, 1, 0.01)
# define search
search = GridSearchCV(model, grid, scoring='neg_mean_absolute_error',
cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X_train,Y_train)
# summarize
print('MAE: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
RepeatedKFold(n_repeats=3, n_splits=10, random_state=1)
MAE: -27738.411
Config: {'alpha': 0.99}
```

```
print('MAE: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
MAE: -27738.411
Config: {'alpha': 0.99}
```

```
model=Lasso(alpha=0.99)
```

```
model.fit(X_train,Y_train)
```

```
Lasso(alpha=0.99)
```

```
model.score(X_train,Y_train)
```

```
0.7464502560405946
```

Logisitic regression

LOgisitic regression used to detect categorical values. ex.Based on FACIAL features and body structure we can decide it belongs to whether it'a male ir female

It is used for predicting the categorical dependent variable using a given set of independent variables.

In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function(Sigmoid Function), which predicts two maximum values (0 or 1).

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as “cat”, “dogs”, or “sheep”

Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as “low”, “Medium”, or “High”.

Sr.No

Linear Regresssion

Logistic Regression

1

Linear regression is used to predict the continuous dependent variable using a given set of independent variables.

Logistic regression is used to predict the categorical dependent variable using a given set of independent variables.

2

Linear regression is used for solving Regression problem.

It is used for solving classification problems.

3

In this we predict the value of continuous variables

In this we predict values of categorical varibles

4

In this we find best fit line.

In this we find S-Curve .

5

Least square estimation method is used for estimation of accuracy.

Maximum likelihood estimation method is used for Estimation of accuracy.

6

The output must be continuous value,such as price,age,etc.

Output is must be categorical value such as 0 or 1, Yes or no, etc.

7

It required linear relationship between dependent and independent variables.

It not required linear relationship.

8

There may be collinearity between the independent variables.

There should not be collinearity between independent varible.

SIGMoid=1/1+e^-z

The sigmoid activation function, also known as the logistic function, squashes input values to a range between 0 and 1. It's commonly used in binary classification problems where the goal is to predict probabilities. The mathematical expression for the sigmoid function is ($\sigma(x) = \frac{1}{1 + e^{-x}}$).

In the context of the sigmoid activation function, "e" refers to Euler's number, a mathematical constant approximately equal to 2.71828. In the sigmoid function, (e) is raised to the power of the negative input ((-x)), leading to the exponential growth or decay component. This exponential operation helps map any real-valued number to a value between 0 and 1, making it suitable for modeling probabilities in machine learning.

euler testing

```python
euler=2.71828
z=-100000

1/(euler**z+1)
```

```
1.0
```

Assumptions for Logistic Regression

The assumptions for Logistic regression are as follows:

Independent observations: Each observation is independent of the other. meaning there is no correlation between any input variables.

Binary dependent variables: It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories softmax functions are used.

Linearity relationship between independent variables and log odds: The relationship between the independent variables and the log odds of the dependent variable should be linear.

No outliers: There should be no outliers in the dataset. Large sample size: The sample size is sufficiently large

Ref:https://www.geeksforgeeks.org/understanding-logistic-regression/

Disadvantage: Vanishing Gradient

Binomial Logistic regression:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X,y=load_breast_cancer(return_X_y =True)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)

model1=LogisticRegression()

model1.fit(X_train,y_train)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:460: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
model1.score(X_train,y_train) *100
95.77464788732394
```

The dataset already preprocessed so,we have to check any other dataset and we need to do one hot encoding

one hot encoding is 1 techinque used to change categorical values into numerical values ,so it will create extra columns in dataset and the column will have a value based on category

Types of encoding methods

In machine learning, encoding methods are used to represent categorical data in a format that can be easily processed by algorithms. Here are some common types of encoding methods:

1. **One-Hot Encoding:**
   - This method represents each category as a binary vector.
   - Each category is converted into a binary vector where all elements are zero except for the index that corresponds to the category, which is marked with a one.
2. **Label Encoding:**
   - In label encoding, each category is assigned a unique numerical label.
   - It's suitable for ordinal data where there is a meaningful order among categories.
3. **Ordinal Encoding:**
   - Similar to label encoding, but it considers the ordinal relationship between categories.
   - It assigns numerical labels based on the order of categories.
4. **Binary Encoding:**
   - This method converts each category into binary code and represents it as a sequence of 0s and 1s.

- It's more space-efficient compared to one-hot encoding.

5. **Frequency Encoding:**
   - Categories are encoded based on their frequency or occurrence in the dataset.
   - This can be useful when the frequency of categories is informative for the model.
6. **Target Encoding (Mean Encoding):**
   - Categories are encoded based on the mean of the target variable for each category.
   - It's useful when the target variable is continuous.
7. **Hashing Encoding:**
   - This method uses hashing functions to map categories to a fixed-size space.
   - It's particularly useful when dealing with high cardinality categorical features.
8. **Entity Embeddings of Categorical Variables:**
   - This technique involves representing categories as vectors of continuous values through the use of embeddings.
   - It's commonly used in deep learning models.

The choice of encoding method depends on the nature of the data, the algorithm being used, and the specific requirements of the machine learning task.

REF: https://www.geeksforgeeks.org/feature-encoding-techniques-machine-learning/

Label encoding

```
import pandas as pd

data=pd.read_csv("dataset/Encoding Data.csv")

# here we have to convert all bin_1,bin_2 because both has binary
catagories

data
```

	id	bin_1	bin_2	nom_0	ord_2
0	0	F	N	Red	Hot
1	1	F	Y	Blue	Warm
2	2	F	N	Blue	Cold
3	3	F	N	Green	Warm
4	4	T	N	Red	Cold
5	5	T	N	Green	Hot
6	6	F	N	Red	Cold
7	7	T	N	Red	Cold
8	8	F	N	Blue	Warm
9	9	F	Y	Red	Hot

```
data["bin_1"]=data["bin_1"].apply(lambda x:1 if x=="T" else (0 if
x=="F" else None))
```

```python
data["bin_2"]=data["bin_2"].apply(lambda x:1 if x=="Y" else (0 if
x=="N" else None))

data
```

	id	bin_1	bin_2	nom_0	ord_2
0	0	0	0	Red	Hot
1	1	0	1	Blue	Warm
2	2	0	0	Blue	Cold
3	3	0	0	Green	Warm
4	4	1	0	Red	Cold
5	5	1	0	Green	Hot
6	6	0	0	Red	Cold
7	7	1	0	Red	Cold
8	8	0	0	Blue	Warm
9	9	0	1	Red	Hot

Label Encoding: Label encoding algorithm is quite simple and it considers an order for encoding, Hence can be used for encoding ordinal data.

```python
from sklearn.preprocessing import LabelEncoder

trans=LabelEncoder()

data["nom_0"]=trans.fit_transform(data[["nom_0"]])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\preprocessing\
_label.py:114: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```python
data["nom_0"]
```

```
0    2
1    0
2    0
3    1
4    2
5    1
6    2
7    2
```

```
8    0
9    2
Name: nom_0, dtype: int32
```

one hot encoding

```python
from sklearn.preprocessing import OneHotEncoder

encoder2=OneHotEncoder()

data["ord_2"]=np.array(data["ord_2"]).reshape(-1,1)

data_2=np.array(data["ord_2"])
data_2=data_2.reshape(-1,1)

data_2=encoder2.fit_transform(data[["ord_2"]]).toarray()

data_2=pd.DataFrame(data_2)

data
```

	id	bin_1	bin_2	nom_0	ord_2
0	0	0	0	2	Hot
1	1	0	1	0	Warm
2	2	0	0	0	Cold
3	3	0	0	1	Warm
4	4	1	0	2	Cold
5	5	1	0	1	Hot
6	6	0	0	2	Cold
7	7	1	0	2	Cold
8	8	0	0	0	Warm
9	9	0	1	2	Hot

```python
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
# transforming the column after fitting
enc = enc.fit_transform(data[['ord_2']]).toarray()
# converting arrays to a dataframe
encoded_colm = pd.DataFrame(enc)
# concatenating dataframes
# df = pd.concat([df, encoded_colm], axis=1)
# # removing the encoded column.
# df = df.drop(['nom_0'], axis=1)
# df.head(10)

encoded_colm
```

	0	1	2
0	0.0	1.0	0.0
1	0.0	0.0	1.0

```
2  1.0  0.0  0.0
3  0.0  0.0  1.0
4  1.0  0.0  0.0
5  0.0  1.0  0.0
6  1.0  0.0  0.0
7  1.0  0.0  0.0
8  0.0  0.0  1.0
9  0.0  1.0  0.0
```

```
data_2
```

```
     0    1    2
0  0.0  1.0  0.0
1  0.0  0.0  1.0
2  1.0  0.0  0.0
3  0.0  0.0  1.0
4  1.0  0.0  0.0
5  0.0  1.0  0.0
6  1.0  0.0  0.0
7  1.0  0.0  0.0
8  0.0  0.0  1.0
9  0.0  1.0  0.0
```

```
data
```

```
   id  bin_1  bin_2  nom_0 ord_2
0   0      0      0      2   Hot
1   1      0      1      0  Warm
2   2      0      0      0  Cold
3   3      0      0      1  Warm
4   4      1      0      2  Cold
5   5      1      0      1   Hot
6   6      0      0      2  Cold
7   7      1      0      2  Cold
8   8      0      0      0  Warm
9   9      0      1      2   Hot
```

```
#concat new columns and drop ord_2

df=pd.concat([data,data_2],axis=1)

df
```

```
   id  bin_1  bin_2  nom_0 ord_2    0    1    2
0   0      0      0      2   Hot  0.0  1.0  0.0
1   1      0      1      0  Warm  0.0  0.0  1.0
2   2      0      0      0  Cold  1.0  0.0  0.0
3   3      0      0      1  Warm  0.0  0.0  1.0
4   4      1      0      2  Cold  1.0  0.0  0.0
5   5      1      0      1   Hot  0.0  1.0  0.0
6   6      0      0      2  Cold  1.0  0.0  0.0
```

7	7	1	0	2	Cold	1.0	0.0	0.0
8	8	0	0	0	Warm	0.0	0.0	1.0
9	9	0	1	2	Hot	0.0	1.0	0.0

```
df.drop(["ord_2"],axis=1)
```

	id	bin_1	bin_2	nom_0	0	1	2
0	0	0	0	2	0.0	1.0	0.0
1	1	0	1	0	0.0	0.0	1.0
2	2	0	0	0	1.0	0.0	0.0
3	3	0	0	1	0.0	0.0	1.0
4	4	1	0	2	1.0	0.0	0.0
5	5	1	0	1	0.0	1.0	0.0
6	6	0	0	2	1.0	0.0	0.0
7	7	1	0	2	1.0	0.0	0.0
8	8	0	0	0	0.0	0.0	1.0
9	9	0	1	2	0.0	1.0	0.0

```
df
```

	id	bin_1	bin_2	nom_0	ord_2	0	1	2
0	0	0	0	2	Hot	0.0	1.0	0.0
1	1	0	1	0	Warm	0.0	0.0	1.0
2	2	0	0	0	Cold	1.0	0.0	0.0
3	3	0	0	1	Warm	0.0	0.0	1.0
4	4	1	0	2	Cold	1.0	0.0	0.0
5	5	1	0	1	Hot	0.0	1.0	0.0
6	6	0	0	2	Cold	1.0	0.0	0.0
7	7	1	0	2	Cold	1.0	0.0	0.0
8	8	0	0	0	Warm	0.0	0.0	1.0
9	9	0	1	2	Hot	0.0	1.0	0.0

Logistic Regression For Multiclass Classification

We can done it by 2 techniques:

```
1.OVR (one vs rest)
2.multinomial
```

OVR

1.In ovr we will do one hot encode the output category 2.then we wil create diff model for each output category. ex.if we have 3 output cateogory then we will have 3 models,because we will take each category as output in each model. finally when we test for new data,it will give 3 outputs ex.(0.25,0.25,0.5) so,3 rd model gives highest result,so 3 rd category will be final output

Real time code example

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import
confusion_matrix,accuracy_score,classification_report

X,y=make_classification(n_classes=3,n_informative=3)
#adjusting the n_informative parameter, you can control the complexity
of the generated dataset and observe how different numbers of
informative features impact the performance of your classification
model.

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)

model=LogisticRegression(multi_class="ovr",solver="lbfgs")

model.fit(X_train,y_train)

LogisticRegression(multi_class='ovr')

y_pred=model.predict(X_test)
y_pred

array([2, 1, 2, 0, 1, 0, 1, 1, 2, 0, 2, 2, 0, 0, 0, 1, 0, 2, 2, 2, 2,
0,
       2, 1, 1])

model.score(X_test,y_pred)

1.0

model.predict_proba(X_test)

array([[1.02496874e-01, 4.15228861e-03, 8.93350837e-01],
       [2.79549548e-01, 3.86801075e-01, 3.33649377e-01],
       [4.28612599e-01, 3.63574257e-03, 5.67751658e-01],
       [5.41437876e-01, 1.70583118e-03, 4.56856293e-01],
       [4.59745745e-01, 5.40143458e-01, 1.10796742e-04],
       [7.23819986e-01, 2.50958884e-01, 2.52211304e-02],
       [4.39772431e-01, 5.59607918e-01, 6.19650254e-04],
       [4.72167563e-04, 9.77370149e-01, 2.21576836e-02],
```

```
       [2.79620778e-01, 3.73546513e-04, 7.20005675e-01],
       [6.54303485e-01, 2.29464920e-01, 1.16231595e-01],
       [3.24163205e-01, 1.65784454e-02, 6.59258349e-01],
       [1.21975946e-01, 1.22935013e-03, 8.76794704e-01],
       [8.87135300e-01, 5.59156511e-02, 5.69490494e-02],
       [5.57674761e-01, 4.42322626e-01, 2.61290261e-06],
       [8.74542278e-01, 4.19868620e-02, 8.34708601e-02],
       [1.31344811e-02, 9.32152649e-01, 5.47128703e-02],
       [7.95860260e-01, 9.09347297e-02, 1.13205010e-01],
       [4.78398339e-01, 2.70822752e-03, 5.18893434e-01],
       [7.49126876e-03, 2.40482216e-02, 9.68460510e-01],
       [1.14657365e-02, 3.22653883e-01, 6.65880380e-01],
       [9.00409385e-02, 1.99730011e-02, 8.89986060e-01],
       [7.53015293e-01, 2.39624876e-01, 7.35983109e-03],
       [4.46000407e-01, 5.80219933e-03, 5.48197393e-01],
       [4.91933049e-01, 5.07618988e-01, 4.47963358e-04],
       [1.58763525e-01, 8.21310796e-01, 1.99256787e-02]])
```

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

```
1.0
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[ 8,  0,  0],
       [ 0,  7,  0],
       [ 0,  0, 10]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00        10

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25
```

In the case of logistic regression, including multinomial logistic regression, the activation function used is the softmax function. The softmax function is applied to the output layer of the neural network to convert the raw output scores into probabilities.

The softmax function takes a vector of raw scores (also known as logits) and transforms them into a probability distribution over multiple classes. It does this by exponentiating each score and normalizing the results. The formula for the softmax function for a class (i) is given by:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where:

- $P(y_i)$ is the probability of class (i),
- z_i is the raw score (logit) for class (i),
- K is the total number of classes.

In scikit-learn's logistic regression implementation with `multi_class='multinomial'`, the softmax function is used by default. The parameter `solver='lbfgs'` is commonly used for multinomial logistic regression, and it's a solver that supports softmax activation.

In neural networks, when using frameworks like TensorFlow or PyTorch, you explicitly specify the softmax activation function in the output layer to achieve the same purpose.

```
# Example using TensorFlow in a neural network
import tensorflow as tf

# Assuming you have defined your neural network architecture
model = tf.keras.Sequential([
    # ... other layers ...
    tf.keras.layers.Dense(units=num_classes, activation='softmax')
    # ...
])

# Compile the model with an appropriate optimizer, loss, and metrics
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

In this example, `activation='softmax'` in the output layer ensures that the softmax activation function is applied to the output layer of the neural network.

```
model2=LogisticRegression(multi_class="multinomial")
```

```
model2.fit(X_train,y_train)
```

```
LogisticRegression(multi_class='multinomial')
```

```
model2.score(X_train,y_train)
```

```
0.9733333333333334
```

The softmax function outputs a probability distribution over multiple classes, ensuring that the probabilities are non-negative and sum to 1. The range of each element in the output vector produced by the softmax function is between 0 and 1.

Mathematically, for a softmax output vector (p = [p_1, p_2, \ldots, p_K]) corresponding to (K) classes, the following properties hold:

1. **Non-Negativity:** Each (p_i) is greater than or equal to 0. [p_i \geq 0 \text{ for } i = 1, 2, \ldots, K]
2. **Sum to 1:** The probabilities sum to 1. [\sum_{i=1}^{K} p_i = 1]

These properties make the softmax function suitable for representing a probability distribution, where each (p_i) can be interpreted as the probability of the input belonging to class (i).

If you're working with the output of a softmax function in a programming context (e.g., using Python and NumPy), you'll find that the values of the softmax output for a given input vector are indeed within the [0, 1] range.

The Rectified Linear Unit (ReLU) is an activation function commonly used in neural networks. The mathematical expression for the ReLU activation function is:

[\text{ReLU}(x) = \max(0, x)]

In other words, for any input (x), the ReLU function outputs the input itself if it's positive, and zero otherwise.

The range of the ReLU function is [0, +∞). If the input is positive, the output is the input value. If the input is zero or negative, the output is zero. Therefore, the function is always non-negative and unbounded for positive inputs.

Mathematically, for any (x \geq 0), (\text{ReLU}(x) = x), and for any (x < 0), (\text{ReLU}(x) = 0).

In a programming context, if you're using a library like TensorFlow, PyTorch, or Keras, you can apply ReLU as an activation function to a layer in a neural network. Here's an example using Python with NumPy:

```
import numpy as np

def relu(x):
    return np.maximum(0, x)

# Example usage
input_data = np.array([-2, -1, 0, 1, 2])
output = relu(input_data)
print(output)
```

In this example, the output will be `[0, 0, 0, 1, 2]`, demonstrating the ReLU activation function's behavior of replacing negative values with zero and leaving positive values unchanged.

```
```

The hyperbolic tangent function, commonly denoted as (\tanh(x)), is an activation function used in neural networks. It is defined mathematically as follows:

[\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}]

The range of the hyperbolic tangent function is between -1 and 1. Mathematically, for any real number (x), (\tanh(x)) will always be in the range ((-1, 1)).

Here are some key properties of the hyperbolic tangent function:

1. **Symmetry:** (\tanh(-x) = -\tanh(x))
2. **Sigmoid-like Behavior:** The shape of the (\tanh) function is similar to that of the sigmoid function, but it ranges from -1 to 1 instead of 0 to 1.

In neural networks, the (\tanh) activation function is often used in the hidden layers to introduce non-linearity. It has the advantage of being zero-centered, which can help with the convergence of the optimization algorithm.

In a programming context, you can apply the (\tanh) function using libraries like NumPy, TensorFlow, PyTorch, or Keras. Here's an example using NumPy:

```
import numpy as np

def tanh(x):
    return np.tanh(x)

# Example usage
input_data = np.array([-2, -1, 0, 1, 2])
output = tanh(input_data)
print(output)
```

The output will be an array of values in the range ((-1, 1)), representing the result of applying the (\tanh) function to the input array.

```

```

complete 153 videos in krish naik ml play list https://youtube.com/playlist?list=PLZoTAELRMXVPBTrWtJkn3wWQxZkmTXGwe&si=776T3xTqDTv3vYld

Multinomial methods typically refer to techniques and models that are designed to handle multinomial or multiclass classification problems. In the context of machine learning, a multinomial classification problem involves predicting the category or class of an observation among three or more possible classes. Here are a few common methods used for multinomial classification:

1. **Multinomial Logistic Regression:**
   - This is an extension of binary logistic regression to handle multiple classes.

   - The model estimates probabilities for each class and assigns the class with the highest probability as the predicted class.
2. **Multinomial Naive Bayes:**
   - An extension of the Naive Bayes algorithm for multiple classes.
   - It assumes that the features are conditionally independent given the class.
3. **Decision Trees and Random Forests:**
   - Decision trees and ensemble methods like random forests can be used for multinomial classification.
   - They recursively split the data based on features to create a tree structure, and each leaf node represents a class.
4. **Support Vector Machines (SVM):**
   - SVMs can be extended for multinomial classification using methods like one-vs-one or one-vs-all.
   - These strategies involve training multiple binary classifiers and combining their outputs.
5. **Neural Networks:**
   - Deep learning models, especially neural networks with softmax activation in the output layer, are commonly used for multinomial classification.
   - The softmax function assigns probabilities to each class, and the class with the highest probability is selected as the predicted class.
6. **K-Nearest Neighbors (KNN):**
   - KNN can also be used for multinomial classification.
   - It classifies a data point based on the majority class among its k-nearest neighbors.

When dealing with multinomial classification problems, the choice of method depends on factors such as the size of the dataset, the nature of the features, and the desired interpretability of the model. Each method has its strengths and weaknesses, and the best choice often involves experimentation and validation on specific datasets.

Optimizers are algorithms or methods used to adjust the parameters of a neural network during the training process. They play a crucial role in minimizing the loss function and helping the model converge to a solution. Here are some common types of optimizers used in deep learning:

1. **Gradient Descent:**
   - **Stochastic Gradient Descent (SGD):** Updates the weights after processing each training sample. It introduces randomness into the optimization process, which can help escape local minima.
   - **Batch Gradient Descent:** Updates the weights based on the average gradient of the entire training dataset. It can be computationally expensive for large datasets.
2. **Adaptive Learning Rate Optimizers:**
   - **Adagrad:** Adapts the learning rates for each parameter based on historical gradients. It performs larger updates for infrequent parameters and smaller updates for frequent ones.
   - **RMSprop (Root Mean Square Propagation):** Addresses the diminishing learning rate problem in Adagrad by using a moving average of squared gradients.
   - **Adam (Adaptive Moment Estimation):** Combines ideas from RMSprop and momentum. It uses both the average of past gradients and the average of past squared gradients to adaptively adjust the learning rates.
3. **Momentum-based Optimizers:**
   - **Momentum:** Adds a fraction of the previous update to the current update. It helps the optimizer to accelerate in the correct direction and dampens oscillations.
4. **Nesterov Accelerated Gradient (NAG):**
   - An improvement over standard momentum that looks ahead in the direction of the momentum term before making an update.
5. **Second-Order Optimizers:**
   - **AdaDelta:** An extension of Adagrad that aims to address its monotonically decreasing learning rates by using a running average of the second moments of the gradients.
   - **L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno):** A quasi-Newton method that uses an approximation of the Hessian matrix.
6. **Others:**
   - **Adaptive Moment Estimation (AdamW):** A variant of Adam that includes weight decay to prevent overfitting.
   - **Nadam:** Combines Nesterov momentum with the benefits of the Adam optimizer.

The choice of optimizer can have a significant impact on the training process and the performance of a neural network. It often involves experimentation to find the optimizer that works well for a specific task and dataset.

Diff b/w sigmoid,soft max,relu,tanh acivation functions

Real time example

Evaluation methods

Basic Assumptions

1.Independence in Observations

2.linearity relationship of X,y

3.Homoscedasticity refers to a statistical property in which the variability of the residuals (the differences between observed and predicted values) is constant across all levels of the independent variable(s). In simpler terms, homoscedasticity indicates that the spread of the residuals remains roughly the same throughout the entire range of the predictor variable(s).

4.normal distribution of X,y

Advantages

1.Easy to implement

2.we can increase it performance by ,hyper parameter tunning,cv

3.we can reduce overfitting regularization

```

```

Disadvantages

1.Its need to do feature scaling

2.It will affect by outliers

3.its sensitive to missing values.

Is sensitive to outliers and how to handle? and coding part ?

what will happen when it contains missing values and how to handle? coding part?

Feature scaling required? and what is feature scaling and types and implematation of feature scaling