Chap.-5 Regression-Model training & Evaluation

- ## Simple Linear Regression ¶
  - **Y = mX + b**
  - Where Y = Dependent variable
    - X = Independent variable
    - m = slop (the value represent change in Y for a unit change in X)
    - b = Intercept (value of Y for X = 0)
- ## to cal
  - **S1 : calculate mean of X and mean Y**
  - **S2 : cal deviation in X (X-mean(X))**
    - **cal deviation in Y (Y-mean(Y))**
  - **S3 : cal product of deviation in X and Y : (X-mean(X)) * (Y-mean(Y))**
  - **S4 : cal the square of deviation for X**
- ## m = sum of product of deviation / sum of square of deviation for X
- ## b = mean of Y - (m * mean of X)

In [ ]:
```
1 =A2-$A$17
```

In [9]:
```
import pandas as pd
df = pd.read_csv('reg.csv')
df
```

Out[9]:

	x	y	dx	dy	product	dx2	m	b
0	15.000000	49.0	-4.933333	-7.8	38.480000	24.337778	1.893	19.0662
1	23.000000	63.0	3.066667	6.2	19.013333	9.404444	NaN	NaN
2	18.000000	58.0	-1.933333	1.2	-2.320000	3.737778	NaN	NaN
3	23.000000	60.0	3.066667	3.2	9.813333	9.404444	NaN	NaN
4	24.000000	58.0	4.066667	1.2	4.880000	16.537778	NaN	NaN
5	22.000000	61.0	2.066667	4.2	8.680000	4.271111	NaN	NaN
6	22.000000	60.0	2.066667	3.2	6.613333	4.271111	NaN	NaN
7	19.000000	63.0	-0.933333	6.2	-5.786667	0.871111	NaN	NaN
8	19.000000	60.0	-0.933333	3.2	-2.986667	0.871111	NaN	NaN
9	16.000000	52.0	-3.933333	-4.8	18.880000	15.471111	NaN	NaN
10	24.000000	62.0	4.066667	5.2	21.146667	16.537778	NaN	NaN
11	11.000000	30.0	-8.933333	-26.8	239.413333	79.804444	NaN	NaN
12	24.000000	59.0	4.066667	2.2	8.946667	16.537778	NaN	NaN
13	16.000000	49.0	-3.933333	-7.8	30.680000	15.471111	NaN	NaN
14	23.000000	68.0	3.066667	11.2	34.346667	9.404444	NaN	NaN
15	19.933333	56.8	NaN	NaN	429.800000	226.933333	NaN	NaN

In [3]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv('reg.csv')

LR = LinearRegression()
LR.fit(df[['x']],df[['y']])

print(LR.coef_)
print(LR.intercept_)
```

```
[[1.8939483]]
[19.0472973]
```

In [10]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv('reg.csv')

LR = LinearRegression()
LR.fit(df[['x']],df[['y']])

ypred = LR.predict(df[['x']])
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)
```

```
[[47.45652174]
 [62.60810811]
 [53.13836663]
 [62.60810811]
 [64.5020564 ]
 [60.71415981]
 [60.71415981]
 [55.03231492]
 [55.03231492]
 [49.35047004]
 [64.5020564 ]
 [39.88072856]
 [64.5020564 ]
 [49.35047004]
 [62.60810811]
 [56.79999999]]
-----------------------
[[1.8939483]]
[19.0472973]
```

- **fit function - to learn model**
- **fit(x,y) -> x must be 2-D**

In [1]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.DataFrame({'y':[140,155,159,179,192,200,212,215],
                   'X1':[60,62,67,70,71,72,75,78]})
LR = LinearRegression()
LR.fit(df[['y']],df[['X1']])

ypred = LR.predict(df[['y']])
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)
```

```
[[60.50992282]
 [63.71416759]
 [64.56863286]
 [68.84095921]
 [71.61797133]
 [73.32690187]
 [75.89029768]
 [76.53114664]]
-----------------------
[[0.21361632]]
[30.60363837]
```

In [14]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[57.12068966]
 [63.47126437]
 [52.04022989]
 [46.95977011]]
-----------------------
[[1.27011494]]
[32.98850575]
-----------------------
(11, 1)
(4, 1)
-----------------------
(11, 1)
(4, 1)
```

- **train_test_split(x,y) first col must be 2-D**
- **x_train, x_test, y_train, y_test = train_test_split(x,y)**
- **sequence must be followed**

In [16]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, train_size=0.5)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[51.72988506]
 [60.71264368]
 [44.24425287]
 [63.70689655]
 [56.22126437]
 [63.70689655]
 [50.23275862]
 [54.72413793]]
-----------------------
[[1.49712644]]
[27.77586207]
-----------------------
(7, 1)
(8, 1)
-----------------------
(7, 1)
(8, 1)
```

In [17]:

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, train_size=0.9)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[47.05376344]
 [64.95698925]]
-----------------------
[[1.98924731]]
[17.21505376]
-----------------------
(13, 1)
(2, 1)
-----------------------
(13, 1)
(2, 1)
```

In [19]:

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, train_size=0.5)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)

print("x_train")
print(x_train)
print("x_test")
print(x_test)
```

```
[[47.3740458 ]
 [62.39694656]
 [53.63358779]
 [61.14503817]
 [61.14503817]
 [57.38931298]
 [53.63358779]
 [63.64885496]]
x_train
     x
4   24
2   18
10  24
8   19
1   23
14  23
0   15
x_test
     x
11  11
3   23
13  16
6   22
5   22
7   19
9   16
12  24
```

In [26]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=3)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print("-----------------------\nx_train")
print(x_train)
print("-----------------------\nx_test")
print(x_test)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[67.67663344]
 [67.67663344]
 [65.393134  ]
 [63.10963455]]
------------------------
[[2.28349945]]
[12.87264673]
------------------------
x_train
     x
2   18
11  11
7   19
5   22
0   15
14  23
13  16
3   23
9   16
8   19
10  24
------------------------
x_test
     x
12  24
4   24
1   23
6   22
------------------------
(11, 1)
(4, 1)
------------------------
(11, 1)
(4, 1)
```

In [24]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=4)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print("-----------------------\nx_train")
print(x_train)
print("-----------------------\nx_test")
print(x_test)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[66.26730564]
 [46.99254526]
 [61.98402556]
 [64.1256656 ]]
-----------------------
[[2.14164004]]
[14.86794462]
-----------------------
x_train
     x
4   24
9   16
11  11
2   18
14  23
13  16
8   19
1   23
5   22
7   19
10  24
-----------------------
x_test
     x
12  24
0   15
6   22
3   23
-----------------------
(11, 1)
(4, 1)
-----------------------
(11, 1)
(4, 1)
```

In [27]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=5)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print("-----------------------\nx_train")
print(x_train)
print("-----------------------\nx_test")
print(x_test)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[59.62008734]
 [61.57292576]
 [53.76157205]
 [51.80873362]]
-----------------------
[[1.95283843]]
[16.65764192]
-----------------------
x_train
     x
10  24
14  23
11  11
4   24
8   19
9   16
0   15
12  24
6   22
13  16
3   23
-----------------------
x_test
    x
5  22
1  23
7  19
2  18
-----------------------
(11, 1)
(4, 1)
-----------------------
(11, 1)
(4, 1)
```

In [28]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "x": [15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23],
    "y": [49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68]
})
x = df[['x']]
y = df[['y']]

x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=42)

LR = LinearRegression()
LR.fit(x_train,y_train)

ypred = LR.predict(x_test)
print(ypred)
print('-----------------------')
print(LR.coef_)
print(LR.intercept_)

print("-----------------------\nx_train")
print(x_train)
print("-----------------------\nx_test")
print(x_test)

print('-----------------------')
print(x_train.shape)
print(x_test.shape)
print('-----------------------')
print(y_train.shape)
print(y_test.shape)
```

```
[[59.99256506]
 [59.06319703]
 [59.80669145]
 [59.99256506]]
-----------------------
[[0.18587361]]
[57.01858736]
-----------------------
x_train
     x
5   22
8   19
2   18
1   23
14  23
4   24
7   19
10  24
12  24
3   23
6   22
-----------------------
x_test
     x
9   16
11  11
0   15
13  16
-----------------------
(11, 1)
(4, 1)
-----------------------
(11, 1)
(4, 1)
```

In [34]:
```
1 import sklearn
2 help(sklearn)
```

```
Help on package sklearn:

NAME
    sklearn

DESCRIPTION
    Machine learning module for Python
    ==================================

    sklearn is a Python module integrating classical machine
    learning algorithms in the tightly-knit world of scientific Python
    packages (numpy, scipy, matplotlib).

    It aims to provide simple and efficient solutions to learning problems
    that are accessible to everybody and reusable in various contexts:
    machine-learning as a versatile tool for science and engineering.

    See http://scikit-learn.org (http://scikit-learn.org) for complete docu
mentation.
```

In [36]:
```
1 import sklearn
2 help(sklearn.model_selection.train_test_split)
```

```
Help on function train_test_split in module sklearn.model_selection._split:

train_test_split(*arrays, **options)
    Split arrays or matrices into random train and test subsets

    Quick utility that wraps input validation and
    ``next(ShuffleSplit().split(X, y))`` and application to input data
    into a single call for splitting (and optionally subsampling) data in a
    oneliner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the proportion
        of the dataset to include in the test split. If int, represents the
        absolute number of test samples. If None, the value is set to the
        complement of the train size. If ``train_size`` is also None, it will
        be set to 0.25.

    train_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test size.

    random_state : int or RandomState instance, default=None
        Controls the shuffling applied to the data before applying the split.
        Pass an int for reproducible output across multiple function calls.
        See :term:`Glossary <random_state>`.


    shuffle : bool, default=True
        Whether or not to shuffle the data before splitting. If shuffle=False
        then stratify must be None.

    stratify : array-like, default=None
        If not None, data is split in a stratified fashion, using this as
        the class labels.

    Returns
    -------
    splitting : list, length=2 * len(arrays)
        List containing train-test split of inputs.

        .. versionadded:: 0.16
            If the input is sparse, the output will be a
            ``scipy.sparse.csr_matrix``. Else, output type is the same as the
            input type.

    Examples
    --------
```

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]

>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]

>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]
```

P.b. 195

In [ ]:

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

x = np.array([5, 15, 25, 35, 45, 55]).reshape(-1,1)
y = np.array([5, 20, 14, 32, 22, 38])

LR = LinearRegression()
LR.fit(x,y)

x_test = np.arange(5).reshape(-1,1)
print(x_test)

ypred = LR.predict(x_test)
print(ypred)
print(LR.coef_)
print(LR.intercept_)
```

Mean Square Error

- **In MCQ**
- **it's a matrix to evaluate the performance of predictive models**
- **it measure the avg. of the square diffreneces between predicted values & target values.**
- **lower mse indicates that the models prediction are closure to the true values reflecting better overall performance.**

MSE = sum (Observed value - predicted value)^2 / n

In [23]:
```
1 import pandas as pd
2
3 df = pd.read_csv('mse.csv')
4 df
```

Out[23]:

	actual	predicted	diff	diff^2
0	67.0	70.0	-3.0	9
1	50.0	49.0	1.0	1
2	36.0	38.0	-2.0	4
3	74.0	76.0	-2.0	4
4	84.0	83.0	1.0	1
5	84.0	80.0	4.0	16
6	64.0	67.0	-3.0	9
7	34.0	30.0	4.0	16
8	23.0	20.0	3.0	9
9	72.0	75.0	-3.0	9
10	62.0	60.0	2.0	4
11	42.0	38.0	4.0	16
12	NaN	NaN	NaN	98

R-squared

- **It is a comparisionof the residual sum of squares (SSres) withthe total sum of squares(SStot)**
- **r-squared is statical measure that represents goodness od fit in model**
- **the value of r^2 is lie bettwen 0 or 1**

In [27]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error

y = np.array([67,50,36,74,84,84,64,34,23,72,62,42])
ypred = np.array([70,49,38,76,83,80,67,30,20,75,60,38])

mse = mean_squared_error(y,ypred)
print("MSE : ", mse)
```

```
MSE :  8.166666666666666
```

R^2=1-SSR/SST

R=sum(observed value-predicted)^2

SST=sum(observed value-mean of (y))^2

In [31]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

y = np.array([67,50,36,74,84,84,64,34,23,72,62,42])
ypred = np.array([70,49,38,76,83,80,67,30,20,75,60,38])

R2 = r2_score(y,ypred)
print("R2 :", R2)
```

```
R2 : 0.9784172661870504
```

In [48]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

dataset=pd.read_csv("Book1.csv")
x=dataset[["cgpa"]]
y=dataset["package"]

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, ra
print(x_train.shape)

LR = LinearRegression()
LR.fit(x_train,y_train)
ypred = LR.predict(x_test)
print("-----------Predicted value------------")
print(ypred)

mse = mean_squared_error(y_test, ypred)
r2 = r2_score(y_test, ypred)

print("MSE :", mse)
print("R2 :", r2)
```

```
(160, 1)
-----------Predicted value------------
[2.9383335  4.36894346 3.18258398 1.89736121 3.49662031 3.35123312
 2.76968435 2.94996447 3.07208971 3.94441286 3.57222165 2.94996447
 2.75805338 2.64755911 3.67108494 3.2174769  3.97930579 2.90925606
 2.19395108 3.31052471 4.29915761 2.8918096  1.87409926 2.30444534
 3.62456104 2.12998071 3.9269664  2.36841571 1.5716939  2.06601035
 2.31026083 3.6885314  3.5024358  3.03719679 2.57195777 2.39167766
 3.170953   3.82228762 3.15932203 2.94414898]
MSE : 0.1370062519255722
R2 : 0.7283345498058083
```

In [50]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

dataset=pd.read_csv("Book1.csv")
x=dataset[["cgpa"]]
y=dataset["package"]

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, ra
print(x_train.shape)

LR = LinearRegression()
LR.fit(x_train,y_train)
ypred = LR.predict(x_test)
print("-----------Predicted value------------")
print(ypred)

mse = mean_squared_error(y_test, ypred)
r2 = r2_score(y_test, ypred)

print("MSE :", mse)
print("R2 :", r2)
```

```
(160, 1)
-----------Predicted value------------
[4.39751239 4.32680345 3.10707437 3.11885919 2.5355105  2.1583962
 3.16010607 2.44712434 2.28213683 3.55489761 3.07761232 3.84362575
 3.16010607 2.51783327 4.00272085 2.84780829 2.77709936 3.24259983
 2.93030204 2.94797928 3.42526457 1.7282502  4.1382463  2.83602347
 2.33516853 3.17189089 3.54311279 1.88734529 2.82423864 3.35455564
 3.20724536 2.31159889 2.37641541 3.19546054 3.94968915 2.34106094
 2.9597641  3.45472662 2.70639042 2.31159889]
MSE : 0.14551953158290792
R2 : 0.6516754875100943
```

In [52]:
```
1 import pandas as pd
2
3 df = pd.read_csv('csv/winequalityN.csv')
4 df
```

Out[52]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	white	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45
1	white	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49
2	white	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44
3	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40
4	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40
...	...	...	...	...	...	...	...	...	...	...	...
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	NaN
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

6497 rows × 13 columns

In [74]:

```
import pandas as pd

df = pd.read_csv('csv/winequalityN.csv')
df.drop(columns = ['type'], inplace=True)
print(df)

# df.fillna(df.mean(), inplace=True)
df['fixed acidity'].fillna(df['fixed acidity'].mean(), inplace=True)
df['volatile acidity'].fillna(df['volatile acidity'].mean(), inplace=True)
df['citric acid'].fillna(df['citric acid'].mean(), inplace=True)
df['residual sugar'].fillna(df['residual sugar'].mean(), inplace=True)
df['chlorides'].fillna(df['chlorides'].mean(), inplace=True)
df['free sulfur dioxide'].fillna(df['free sulfur dioxide'].mean(), inplace
df['total sulfur dioxide'].fillna(df['total sulfur dioxide'].mean(), inpla
df['density'].fillna(df['density'].mean(), inplace=True)
df['pH'].fillna(df['pH'].mean(), inplace=True)
df['sulphates'].fillna(df['sulphates'].mean(), inplace=True)
df['alcohol'].fillna(df['alcohol'].mean(), inplace=True)

print("-----------------------------")
print(df.mean())
print("-----------------------------")

X = df.drop(columns=["quality"])  # Assuming 'quality' is the target colum
y = df["quality"]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Get model parameters
coefficients = model.coef_
intercept = model.intercept_

# Make predictions on test data
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Print model parameters and MSE
print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept}")
print(f"Mean Squared Error: {mse}")

# Predict wine quality for given input data
input_data = np.array([[8, 0.4, 0.40, 15, 0.048, 40, 150, 0.99, 3, 0.45, 1
predicted_quality = model.predict(input_data)

print(f"Predicted Wine Quality: {predicted_quality[0]}")
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides
\
0               7.0             0.270         0.36            20.7      0.045
1               6.3             0.300         0.34             1.6      0.049
2               8.1             0.280         0.40             6.9      0.050
3               7.2             0.230         0.32             8.5      0.058
4               7.2             0.230         0.32             8.5      0.058
...             ...               ...          ...             ...        ...
6492            6.2             0.600         0.08             2.0      0.090
6493            5.9             0.550         0.10             2.2      0.062
6494            6.3             0.510         0.13             2.3      0.076
6495            5.9             0.645         0.12             2.0      0.075
6496            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    45.0                 170.0  1.00100  3.00       0.45
1                    14.0                 132.0  0.99400  3.30       0.49
2                    30.0                  97.0  0.99510  3.26       0.44
3                    47.0                 186.0  0.99560  3.19       0.40
4                    47.0                 186.0  0.99560  3.19       0.40
...                   ...                   ...      ...   ...        ...
6492                 32.0                  44.0  0.99490  3.45       0.58
6493                 39.0                  51.0  0.99512  3.52        NaN
6494                 29.0                  40.0  0.99574  3.42       0.75
6495                 32.0                  44.0  0.99547  3.57       0.71
6496                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         8.8        6
1         9.5        6
2        10.1        6
3         9.9        6
4         9.9        6
...       ...      ...
6492     10.5        5
6493     11.2        6
6494     11.0        6
6495     10.2        5
6496     11.0        6

[6497 rows x 12 columns]
-----------------------------
fixed acidity             7.216579
volatile acidity          0.339691
citric acid               0.318722
residual sugar            5.444326
chlorides                 0.056042
free sulfur dioxide      30.525319
total sulfur dioxide    115.744574
density                   0.994697
pH                        3.218395
sulphates                 0.531215
alcohol                  10.491801
quality                   5.818378
dtype: float64
-----------------------------
Coefficients: [ 6.42757462e-02 -1.36491029e+00 -1.29485941e-01  4.40012463e-0
```

```
2
 -4.94887501e-01  5.47543324e-03 -2.54159642e-03 -4.94329238e+01
  4.45051582e-01  7.20715979e-01  2.78031584e-01]
Intercept: 50.21778442342276
Mean Squared Error: 0.586862752307335
Predicted Wine Quality: 6.248487834652494
```

In [ ]:
```
1
```