GOVERNMENT POLYTECHNIC NAGAMANGALA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**Artificial Intelligence and Machine Learning (20CS51)**

**Assignment NO.: 05**

**SUBMITTED BY :-**

**NAME: YASHWANTH GOWDA MS**

**ROLL NO: 158CS21025**

**Under the guidance of :-**

**Mr. JAYARAMU H K (BE, Meth, (PhD))**

**Lecture, Dept. of CSE**

**Govt polytechnic Nagamangala**

**AIML (20CS51)**

ASSIGNMENT – WEEK -06

1. Perform data exploration, Preprocessing and Splitting on datasets like:

(a). Boston housing price from sci-kit learn datasets.

(b). Cricket match result - past data

(c). Performance of a cricket player - past data

(d). Crop yield - past data

**Dataset Name : House price**

(a). Boston housing price from sci-kit learn datasets.

* Perform data exploration:

import pandas as pd

df = pd.read\_csv("/content/HousingData.csv")

df.describe()

output:

| **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** | **LSTAT** | **MEDV** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 486.000000 | 486.000000 | 486.000000 | 486.000000 | 506.000000 | 506.000000 | 486.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 486.000000 | 506.000000 |
| **mean** | 3.611874 | 11.211934 | 11.083992 | 0.069959 | 0.554695 | 6.284634 | 68.518519 | 3.795043 | 9.549407 | 408.237154 | 18.455534 | 356.674032 | 12.715432 | 22.532806 |
| **std** | 8.720192 | 23.388876 | 6.835896 | 0.255340 | 0.115878 | 0.702617 | 27.999513 | 2.105710 | 8.707259 | 168.537116 | 2.164946 | 91.294864 | 7.155871 | 9.197104 |
| **min** | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 187.000000 | 12.600000 | 0.320000 | 1.730000 | 5.000000 |
| **25%** | 0.081900 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.175000 | 2.100175 | 4.000000 | 279.000000 | 17.400000 | 375.377500 | 7.125000 | 17.025000 |
| **50%** | 0.253715 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 76.800000 | 3.207450 | 5.000000 | 330.000000 | 19.050000 | 391.440000 | 11.430000 | 21.200000 |
| **75%** | 3.560263 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 93.975000 | 5.188425 | 24.000000 | 666.000000 | 20.200000 | 396.225000 | 16.955000 | 25.000000 |
| **max** | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 | 37.970000 | 50.000000 |

df.head()

output:

| **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** | **LSTAT** | **MEDV** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 296 | 15.3 | 396.90 | 4.98 | 24.0 |
| **1** | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | 17.8 | 396.90 | 9.14 | 21.6 |
| **2** | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | 17.8 | 392.83 | 4.03 | 34.7 |
| **3** | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | 18.7 | 394.63 | 2.94 | 33.4 |
| **4** | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | 18.7 | 396.90 | NaN | 36.2 |

df.tail()

output:

| **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** | **LSTAT** | **MEDV** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **501** | 0.06263 | 0.0 | 11.93 | 0.0 | 0.573 | 6.593 | 69.1 | 2.4786 | 1 | 273 | 21.0 | 391.99 | NaN | 22.4 |
| **502** | 0.04527 | 0.0 | 11.93 | 0.0 | 0.573 | 6.120 | 76.7 | 2.2875 | 1 | 273 | 21.0 | 396.90 | 9.08 | 20.6 |
| **503** | 0.06076 | 0.0 | 11.93 | 0.0 | 0.573 | 6.976 | 91.0 | 2.1675 | 1 | 273 | 21.0 | 396.90 | 5.64 | 23.9 |
| **504** | 0.10959 | 0.0 | 11.93 | 0.0 | 0.573 | 6.794 | 89.3 | 2.3889 | 1 | 273 | 21.0 | 393.45 | 6.48 | 22.0 |
| **505** | 0.04741 | 0.0 | 11.93 | 0.0 | 0.573 | 6.030 | NaN | 2.5050 | 1 | 273 | 21.0 | 396.90 | 7.88 | 11.9 |

#replacement

import pandas as pd

data = pd.DataFrame({'name': ['0.06263', '0.04527']})

print("Original data:")

print(data)

data['name'].replace({'0.06263': '06263'}, inplace=True)

print("\nModified data:")

print(data)

output:

Original data:

name

0 0.06263

1 0.04527

Modified data:

name

0 06263

1. 0.04527

df.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 506 entries, 0 to 505

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 CRIM 486 non-null float64

1 ZN 486 non-null float64

2 INDUS 486 non-null float64

3 CHAS 486 non-null float64

4 NOX 506 non-null float64

5 RM 506 non-null float64

6 AGE 486 non-null float64

7 DIS 506 non-null float64

8 RAD 506 non-null int64

9 TAX 506 non-null int64

10 PTRATIO 506 non-null float64

11 B 506 non-null float64

12 LSTAT 486 non-null float64

13 MEDV 506 non-null float64

dtypes: float64(12), int64(2)

memory usage: 55.5 KB

#data duplication function

df.nunique()

output:

CRIM: 484

ZN: 26

INDUS :76

CHAS: 2

NOX: 81

RM: 446

AGE: 348

DIS: 412

RAD: 9

TAX :66

PTRATIO: 46

B :357

LSTAT: 438

MEDV: 229

dtype: int64

#missing value function

df.isnull().sum()

output:

CRIM :20

ZN :20

INDUS :20

CHAS: 20

NOX: 0

RM: 0

AGE: 20

DIS: 0

RAD: 0

TAX: 0

PTRATIO: 0

B: 0

LSTAT :20

MEDV: 0

dtype: int64

----------------------------------------------------------------------------------------------------

#is null().sum()

(df.isnull().sum()/(len(df)))\*100

output:

CRIM: 3.952569

ZN: 3.952569

INDUS: 3.952569

CHAS: 3.952569

NOX :0.000000

RM 0:.000000

AGE: 3.952569

DIS :0.000000

RAD: 0.000000

TAX :0.000000

PTRATIO: 0.000000

B: 0.000000

LSTAT :3.952569

MEDV: 0.000000

dtype: float64

---------------------------------------------------------------------------------------

df = df.drop(['CHAS'], axis=1)

df.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 506 entries, 0 to 505

Data columns (total 13 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 CRIM 486 non-null float64

1 ZN 486 non-null float64

2 INDUS 486 non-null float64

3 NOX 506 non-null float64

4 RM 506 non-null float64

5 AGE 486 non-null float64

6 DIS 506 non-null float64

7 RAD 506 non-null int64

8 TAX 506 non-null int64

9 PTRATIO 506 non-null float64

10 B 506 non-null float64

11 LSTAT 486 non-null float64

12 MEDV 506 non-null float64

dtypes: float64(11), int64(2)

memory usage: 51.5 KB

----------------------------------------------------------------------------------------------------

from datetime import date

df['TAX'] = date.today().year - df['PTRATIO']

df.head()

output:

| **CRIM** | **ZN** | **INDUS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** | **LSTAT** | **MEDV** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.00632 | 18.0 | 2.31 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 2007.7 | 15.3 | 396.90 | 4.98 | 24.0 |
| **1** | 0.02731 | 0.0 | 7.07 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 2005.2 | 17.8 | 396.90 | 9.14 | 21.6 |
| **2** | 0.02729 | 0.0 | 7.07 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 2005.2 | 17.8 | 392.83 | 4.03 | 34.7 |
| **3** | 0.03237 | 0.0 | 2.18 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 2004.3 | 18.7 | 394.63 | 2.94 | 33.4 |
| **4** | 0.06905 | 0.0 | 2.18 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 2004.3 | 18.7 | 396.90 | NaN | 36.2 |

------------------------------------------------------------------------------------------------------------------------------------------------------------

#unique function

print(df['ZN'].unique())

print(df['INDUS'].unique())

output:

[ 18. 0. 12.5 75. 21. 90. 85. 100. 25. 17.5 80. nan

28. 45. 60. 95. 82.5 30. 22. 20. 40. 55. 52.5 70.

34. 33. 35. ]

[ 2.31 7.07 2.18 7.87 8.14 5.96 nan 2.95 6.91 5.64 4. 1.22

0.74 1.32 5.13 1.38 3.37 6.07 10.81 12.83 4.86 4.49 3.41 15.04

2.89 8.56 10.01 25.65 21.89 19.58 4.05 2.46 3.44 2.93 0.46 1.52

1.47 2.03 2.68 10.59 13.89 6.2 4.93 5.86 3.64 3.75 3.97 6.96

6.41 3.33 1.21 2.97 2.25 1.76 5.32 4.95 13.92 2.24 6.09 9.9

7.38 3.24 6.06 5.19 1.89 3.78 4.39 4.15 2.01 1.25 1.69 2.02

1.91 18.1 27.74 9.69 11.93]

---------------------------------------------------------------------------------------

import numpy as np

catechol’s=df.select\_dtypes(include=['object']).columns

num\_cols=df.select\_dtypes(include=np.number).columns.tolist()

print("categorical variables:")

print(cat\_cols)

print("numerical variables:")

print(num\_cols)

output:

categorical variables:

Index([], dtype='object')

numerical variables:

['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

---------------------------------------------------------------------------------------------------------------------

* Preprocessing and Splitting:

df1 =df.dropna()

print(df1)

output:

CRIM ZN INDUS NOX RM AGE DIS RAD TAX PTRATIO \

0 0.00632 18.0 2.31 0.538 6.575 65.2 4.0900 1 2007.7 15.3

1 0.02731 0.0 7.07 0.469 6.421 78.9 4.9671 2 2005.2 17.8

2 0.02729 0.0 7.07 0.469 7.185 61.1 4.9671 2 2005.2 17.8

3 0.03237 0.0 2.18 0.458 6.998 45.8 6.0622 3 2004.3 18.7

5 0.02985 0.0 2.18 0.458 6.430 58.7 6.0622 3 2004.3 18.7

.. ... ... ... ... ... ... ... ... ... ...

499 0.17783 0.0 9.69 0.585 5.569 73.5 2.3999 6 2003.8 19.2

500 0.22438 0.0 9.69 0.585 6.027 79.7 2.4982 6 2003.8 19.2

502 0.04527 0.0 11.93 0.573 6.120 76.7 2.2875 1 2002.0 21.0

503 0.06076 0.0 11.93 0.573 6.976 91.0 2.1675 1 2002.0 21.0

504 0.10959 0.0 11.93 0.573 6.794 89.3 2.3889 1 2002.0 21.0

B LSTAT MEDV

0 396.90 4.98 24.0

1 396.90 9.14 21.6

2 392.83 4.03 34.7

3 394.63 2.94 33.4

5 394.12 5.21 28.7

.. ... ... ...

499 395.77 15.10 17.5

500 396.90 14.33 16.8

502 396.90 9.08 20.6

503 396.90 5.64 23.9

504 393.45 6.48 22.0

[411 rows x 13 columns]

--------------------------------------------------------------------------

df2=df.dropna(axis=1)

print(df2)

output:

NOX RM DIS RAD TAX PTRATIO B MEDV

0 0.538 6.575 4.0900 1 2007.7 15.3 396.90 24.0

1 0.469 6.421 4.9671 2 2005.2 17.8 396.90 21.6

2 0.469 7.185 4.9671 2 2005.2 17.8 392.83 34.7

3 0.458 6.998 6.0622 3 2004.3 18.7 394.63 33.4

4 0.458 7.147 6.0622 3 2004.3 18.7 396.90 36.2

.. ... ... ... ... ... ... ... ...

501 0.573 6.593 2.4786 1 2002.0 21.0 391.99 22.4

502 0.573 6.120 2.2875 1 2002.0 21.0 396.90 20.6

503 0.573 6.976 2.1675 1 2002.0 21.0 396.90 23.9

504 0.573 6.794 2.3889 1 2002.0 21.0 393.45 22.0

505 0.573 6.030 2.5050 1 2002.0 21.0 396.90 11.9

[506 rows x 8 columns]

---------------------------------------------------------------------------------------------------------------------

df3 = df.fillna(df.mean())

print(df3)

output:

CRIM ZN INDUS NOX RM AGE DIS RAD TAX \

0 0.00632 18.0 2.31 0.538 6.575 65.200000 4.0900 1 2007.7

1 0.02731 0.0 7.07 0.469 6.421 78.900000 4.9671 2 2005.2

2 0.02729 0.0 7.07 0.469 7.185 61.100000 4.9671 2 2005.2

3 0.03237 0.0 2.18 0.458 6.998 45.800000 6.0622 3 2004.3

4 0.06905 0.0 2.18 0.458 7.147 54.200000 6.0622 3 2004.3

.. ... ... ... ... ... ... ... ... ...

501 0.06263 0.0 11.93 0.573 6.593 69.100000 2.4786 1 2002.0

502 0.04527 0.0 11.93 0.573 6.120 76.700000 2.2875 1 2002.0

503 0.06076 0.0 11.93 0.573 6.976 91.000000 2.1675 1 2002.0

504 0.10959 0.0 11.93 0.573 6.794 89.300000 2.3889 1 2002.0

505 0.04741 0.0 11.93 0.573 6.030 68.518519 2.5050 1 2002.0

PTRATIO B LSTAT MEDV

0 15.3 396.90 4.980000 24.0

1 17.8 396.90 9.140000 21.6

2 17.8 392.83 4.030000 34.7

3 18.7 394.63 2.940000 33.4

4 18.7 396.90 12.715432 36.2

.. ... ... ... ...

501 21.0 391.99 12.715432 22.4

502 21.0 396.90 9.080000 20.6

503 21.0 396.90 5.640000 23.9

504 21.0 393.45 6.480000 22.0

505 21.0 396.90 7.880000 11.9

[506 rows x 13 columns]

--------------------------------------------------------------------------

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

df.dropna(inplace=True)

X = df[['CRIM', 'ZN']]

y = df['INDUS']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

output:

Mean Squared Error: 30.132823538717176

R-squared: 0.286409349436472

--------------------------------------------------------------------------

Simple linear regression and multi linear regression:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

df=pd.read\_csv("/content/HousingData.csv")

X = df["DIS"].values.reshape(-1, 1)

y = df["TAX"].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

y\_pred\_test = regressor.predict(x\_test)

y\_pred\_train = regressor.predict(x\_train)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_train, y\_pred\_train, color="red", label="Regression Line")

plt.title("Simple Linear Regression")

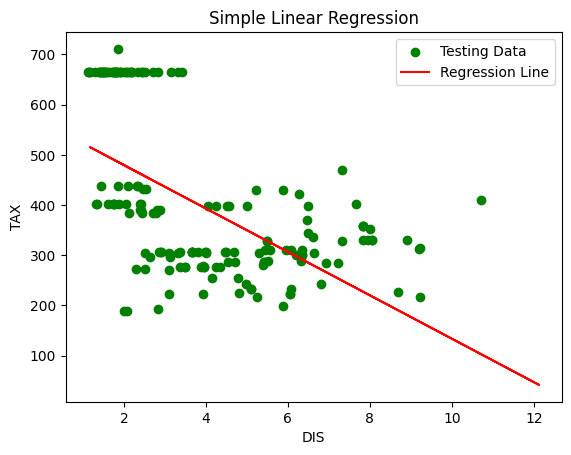
plt.xlabel("DIS")

plt.ylabel("TAX")

plt.legend()

plt.show()

output:



---------------------------------------------------------------------------------------

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

df = pd.read\_csv("/content/HousingData.csv")

X = df[["MEDV"]]

y = df["RAD"]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

model = LinearRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred, color="red", label="Regression Line")

plt.title("multiple Linear Regression")

plt.xlabel("MEDV")

plt.ylabel("RAD")

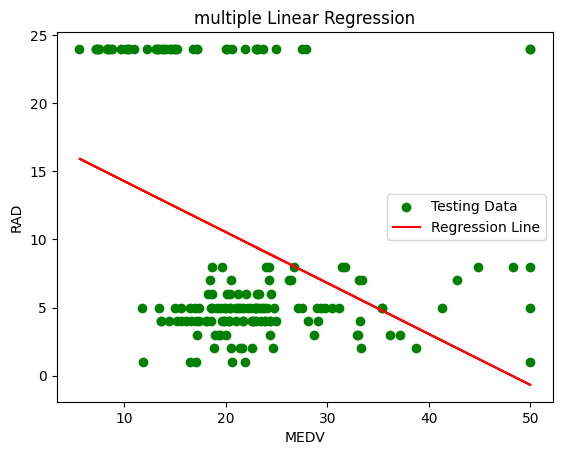
plt.legend()

plt.show()

output:

Mean Squared Error: 65.7315632589674

R-squared: 0.1173659103076038



---------------------------------------------------------------------------------------

(b). Cricket match result - past data:

**Dataset name: india t20 match**

Perform data exploration:

import pandas as pd

data = pd.read\_csv("/content/t20\_matches.csv")

data.describe()

output:

|  | **match\_id** | **series\_id** | **win\_by\_runs** | **win\_by\_wickets** | **balls\_remaining** | **innings1\_runs** | **innings1\_wickets** | **innings1\_overs\_batted** | **innings1\_overs** | **innings2\_runs** | **innings2\_wickets** | **innings2\_overs\_batted** | **innings2\_overs** | **D/L\_method** | **target** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 6.417000e+03 | 6.417000e+03 | 2931.000000 | 3004.000000 | 2969.000000 | 6151.000000 | 6151.000000 | 6151.000000 | 6151.000000 | 6070.000000 | 6070.000000 | 6070.000000 | 6070.000000 | 233.0 | 231.000000 |
| **mean** | 6.000441e+05 | 5.879816e+05 | 31.320710 | 6.089214 | 13.978107 | 151.409527 | 6.577630 | 19.523199 | 19.738612 | 136.125371 | 5.998847 | 17.862537 | 19.530659 | 1.0 | 102.528139 |
| **std** | 2.302316e+05 | 2.387841e+05 | 26.838237 | 1.996341 | 14.664918 | 32.986924 | 2.152355 | 1.924720 | 1.510632 | 31.880776 | 2.923842 | 3.075877 | 2.108147 | 0.0 | 40.406713 |
| **min** | 1.295970e+05 | 1.241210e+05 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.200000 | 5.000000 | 0.000000 | 0.000000 | 0.100000 | 5.000000 | 1.0 | 20.000000 |
| **25%** | 4.249410e+05 | 4.200050e+05 | 11.000000 | 5.000000 | 4.000000 | 131.000000 | 5.000000 | 20.000000 | 20.000000 | 117.000000 | 4.000000 | 17.200000 | 20.000000 | 1.0 | 72.000000 |
| **50%** | 5.673530e+05 | 5.642800e+05 | 24.000000 | 6.000000 | 9.000000 | 152.000000 | 7.000000 | 20.000000 | 20.000000 | 138.000000 | 6.000000 | 19.100000 | 20.000000 | 1.0 | 98.000000 |
| **75%** | 7.760790e+05 | 7.754450e+05 | 45.000000 | 8.000000 | 19.000000 | 173.000000 | 8.000000 | 20.000000 | 20.000000 | 157.000000 | 9.000000 | 20.000000 | 20.000000 | 1.0 | 129.000000 |
| **max** | 1.085496e+06 | 1.083439e+06 | 172.000000 | 10.000000 | 100.000000 | 263.000000 | 10.000000 | 20.000000 | 20.000000 | 248.000000 | 10.000000 | 20.000000 | 20.000000 | 1.0 | 233.000000 |

-----------------------------------------------------------------------------------------------------------------------------------------------------------

data.head()

output:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 6.417000e+03 | 6.417000e+03 | 2931.000000 | 3004.000000 | 2969.000000 | 6151.000000 | 6151.000000 | 6151.000000 | 6151.000000 | 6070.000000 | 6070.000000 | 6070.000000 | 6070.000000 | 233.0 | 231.000000 |
| **mean** | 6.000441e+05 | 5.879816e+05 | 31.320710 | 6.089214 | 13.978107 | 151.409527 | 6.577630 | 19.523199 | 19.738612 | 136.125371 | 5.998847 | 17.862537 | 19.530659 | 1.0 | 102.528139 |
| **std** | 2.302316e+05 | 2.387841e+05 | 26.838237 | 1.996341 | 14.664918 | 32.986924 | 2.152355 | 1.924720 | 1.510632 | 31.880776 | 2.923842 | 3.075877 | 2.108147 | 0.0 | 40.406713 |
| **min** | 1.295970e+05 | 1.241210e+05 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.200000 | 5.000000 | 0.000000 | 0.000000 | 0.100000 | 5.000000 | 1.0 | 20.000000 |
| **25%** | 4.249410e+05 | 4.200050e+05 | 11.000000 | 5.000000 | 4.000000 | 131.000000 | 5.000000 | 20.000000 | 20.000000 | 117.000000 | 4.000000 | 17.200000 | 20.000000 | 1.0 | 72.000000 |
| **50%** | 5.673530e+05 | 5.642800e+05 | 24.000000 | 6.000000 | 9.000000 | 152.000000 | 7.000000 | 20.000000 | 20.000000 | 138.000000 | 6.000000 | 19.100000 | 20.000000 | 1.0 | 98.000000 |

data.tail()

output:

|  | **match\_id** | **series\_id** | **win\_by\_runs** | **win\_by\_wickets** | **balls\_remaining** | **innings1\_runs** | **innings1\_wickets** | **innings1\_overs\_batted** | **innings1\_overs** | **innings2\_runs** | **innings2\_wickets** | **innings2\_overs\_batted** | **innings2\_overs** | **D/L\_method** | **target** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 6.417000e+03 | 6.417000e+03 | 2931.000000 | 3004.000000 | 2969.000000 | 6151.000000 | 6151.000000 | 6151.000000 | 6151.000000 | 6070.000000 | 6070.000000 | 6070.000000 | 6070.000000 | 233.0 | 231.000000 |
| **mean** | 6.000441e+05 | 5.879816e+05 | 31.320710 | 6.089214 | 13.978107 | 151.409527 | 6.577630 | 19.523199 | 19.738612 | 136.125371 | 5.998847 | 17.862537 | 19.530659 | 1.0 | 102.528139 |
| **std** | 2.302316e+05 | 2.387841e+05 | 26.838237 | 1.996341 | 14.664918 | 32.986924 | 2.152355 | 1.924720 | 1.510632 | 31.880776 | 2.923842 | 3.075877 | 2.108147 | 0.0 | 40.406713 |
| **min** | 1.295970e+05 | 1.241210e+05 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.200000 | 5.000000 | 0.000000 | 0.000000 | 0.100000 | 5.000000 | 1.0 | 20.000000 |
| **25%** | 4.249410e+05 | 4.200050e+05 | 11.000000 | 5.000000 | 4.000000 | 131.000000 | 5.000000 | 20.000000 | 20.000000 | 117.000000 | 4.000000 | 17.200000 | 20.000000 | 1.0 | 72.000000 |
| **50%** | 5.673530e+05 | 5.642800e+05 | 24.000000 | 6.000000 | 9.000000 | 152.000000 | 7.000000 | 20.000000 | 20.000000 | 138.000000 | 6.000000 | 19.100000 | 20.000000 | 1.0 | 98.000000 |
| **75%** | 7.760790e+05 | 7.754450e+05 | 45.000000 | 8.000000 | 19.000000 | 173.000000 | 8.000000 | 20.000000 | 20.000000 | 157.000000 | 9.000000 | 20.000000 | 20.000000 | 1.0 | 129.000000 |
| **max** | 1.085496e+06 | 1.083439e+06 | 172.000000 | 10.000000 | 100.000000 | 263.000000 | 10.000000 | 20.000000 | 20.000000 | 248.000000 | 10.000000 | 20.000000 | 20.000000 | 1.0 | 233.000000 |

----------------------------------------------------------------------------------------------------

#missing value function

data.isnull().sum()

output:

match\_id: 0

series\_id: 0

match details: 0

result :14

scores: 266

date: 0

venue: 0

round: 1274

home: 0

away: 0

winner: 14

win\_by\_runs :3486

win\_by\_wickets: 3413

balls\_remaining: 3448

innings1 :266

innings1\_runs :266

innings1\_wickets: 266

innings1\_overs\_batted :266

innings1\_overs: 266

innings2 :347

innings2\_runs: 347

innings2\_wickets :347

innings2\_overs\_batted :347

innings2\_overs :347

D/L\_method: 6184

target :6186

dtype: int64

#is null().sum()

(data.isnull().sum()/(len(data)))\*100

output:

match\_id :0.000000

series\_id :0.000000

match\_details: 0.000000

result :0.218170

scores: 4.145239

date :0.000000

venue: 0.000000

round :19.853514

home :0.000000

away: 0.000000

winner :0.218170

win\_by\_runs :54.324451

win\_by\_wickets :53.186847

balls\_remaining: 53.732274

innings1: 4.145239

innings1\_runs: 4.145239

innings1\_wickets :4.145239

innings1\_overs\_batted :4.145239

innings1\_overs :4.145239

innings2 :5.407511

innings2\_runs :5.407511

innings2\_wickets :5.407511

innings2\_overs\_batted :5.407511

innings2\_overs: 5.407511

D/L\_method :96.369020

target 96.400187

----------------------------------------------------------------------------------------------------

#replacement

import pandas as pd

data = pd.DataFrame({'innings1\_wickets': ['9.0  ', '4.0']})

print("Original data:")

print(data)

data['innings1\_wickets'].replace({'9.0': '10'}, inplace=True)

print("\nModified data:")

print(data)

output:

Original data:

innings1\_wickets

0 9.0

1 4.0

Modified data:

innings1\_wickets

0 9.0

1. 4.0

-----------------------------------------------------------------------------------------------------------

data.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2 entries, 0 to 1

Data columns (total 1 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 innings1\_wickets 2 non-null object

dtypes: object(1)

memory usage: 144.0+ bytes

---------------------------------------------------------------------------------------------------------------------

#data duplication function

data.nunique()

output:

innings1\_wickets: 2

dtype: int64

----------------------------------------------------------------------------------------------------

import numpy as np

cat\_cols=data.select\_dtypes(include=['object']).columns

num\_cols=data.select\_dtypes(include=np.number).columns.tolist()

print("categorical variables:")

print(cat\_cols)

print("numerical variables:")

print(num\_cols)

output:

categorical variables:

Index(['innings1\_wickets'], dtype='object')

numerical variables:

[]

----------------------------------------------------------------------------------------------------

Preprocessing and Splitting:

df1 =data.dropna()

print(df1)

output:

innings1\_wickets

1. 9.0
2. 4.0

---------------------------------------------------------------------------------------------------

data2=data.dropna(axis=1)

print(data2)

output:

innings1\_wickets

0 9.0

1. 1 4.0

----------------------------------------------------------------------------------------------------

data3= data.fillna(data.mean())

print(data3)

output:

innings1\_wickets

0 9.0

1 4.0

<ipython-input-40-94362da0678c>:1:

----------------------------------------------------------------------------------------------------

Simple linear regression and multi linear regression:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

df=pd.read\_csv("/content/t20\_matches.csv")

X = df["match\_id"].values.reshape(-1, 1)

y = df["series\_id"].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

y\_pred\_test = regressor.predict(x\_test)

y\_pred\_train = regressor.predict(x\_train)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred\_test, color="red", label="Regression Line")

plt.title("Linear Regression")

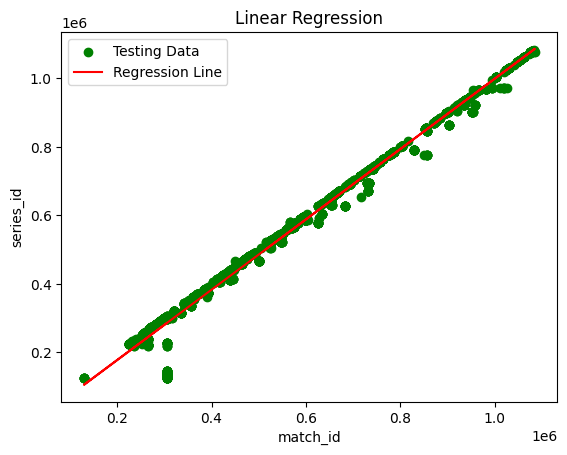
plt.xlabel("match\_id")

plt.ylabel("series\_id")

plt.legend()

plt.show()

output:



---------------------------------------------------------------------------------------------------

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

df=pd.read\_csv("/content/t20\_matches.csv")

X = df[["series\_id"]]

y = df["match\_id"]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

model = LinearRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred, color="red", label="Regression Line")

plt.title("multiple Linear Regression")

plt.xlabel("match\_id")

plt.ylabel("series\_id")

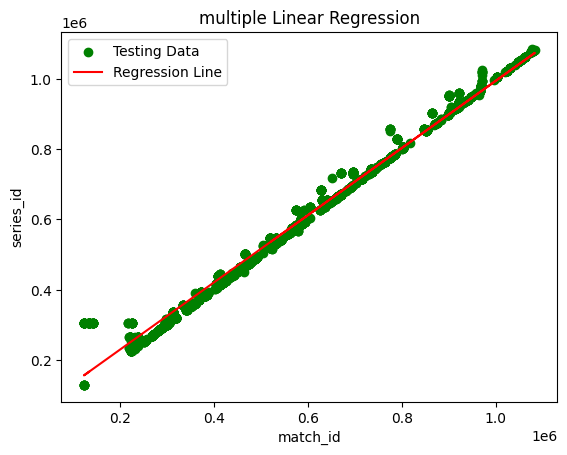
plt.legend()

plt.show()

output:

Mean Squared Error: 914354099.7849469

R-squared: 0.9823635982192883



----------------------------------------------------------------------------------------------------

(c). Performance of a cricket player - past data

**Dataset Name : virat kohil**

* Perform data exploration:

import pandas as pd

df = pd.read\_csv("/content/final (1).csv")

df.describe()

output:

|  | **index** | **runs** | **Match\_No** | **total** |
| --- | --- | --- | --- | --- |
| **count** | 516.000000 | 516.000000 | 516.000000 | 516.000000 |
| **mean** | 270.118217 | 45.947674 | 258.500000 | 11681.726744 |
| **std** | 155.219618 | 44.584372 | 149.100637 | 7301.114849 |
| **min** | 0.000000 | 0.000000 | 1.000000 | 12.000000 |
| **25%** | 134.750000 | 11.000000 | 129.750000 | 5328.250000 |
| **50%** | 270.500000 | 32.500000 | 258.500000 | 10886.500000 |
| **75%** | 403.250000 | 70.250000 | 387.250000 | 18535.750000 |
| **max** | 539.000000 | 254.000000 | 516.000000 | 23709.000000 |

**---------------------------------------------------------------------------------------**

df.head()

output:

|  | **index** | **runs** | **opponent** | **ground** | **date** | **match** | **Match\_No** | **total** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 12 | SriLanka | Dambulla | 18Aug2008 | ODI | 1 | 12 |
| **1** | 1 | 37 | SriLanka | Dambulla | 20Aug2008 | ODI | 2 | 49 |
| **2** | 2 | 25 | SriLanka | Colombo(RPS) | 24Aug2008 | ODI | 3 | 74 |
| **3** | 3 | 54 | SriLanka | Colombo(RPS) | 27Aug2008 | ODI | 4 | 128 |
| **4** | 4 | 31 | SriLanka | Colombo(RPS) | 29Aug2008 | ODI | 5 | 159 |

**---------------------------------------------------------------------------------------**

df.tail()

output:

|  | **index** | **runs** | **opponent** | **ground** | **date** | **match** | **Match\_No** | **total** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **511** | 535 | 11 | England | Birmingham | 1Jul2022 | Test | 512 | 23661 |
| **512** | 536 | 20 | England | Birmingham | 1Jul2022 | Test | 513 | 23681 |
| **513** | 537 | 1 | England | Birmingham | 9Jul2022 | T20 | 514 | 23682 |
| **514** | 538 | 11 | England | Nottingham | 10Jul2022 | T20 | 515 | 23693 |
| **515** | 539 | 16 | England | Lord's | 14Jul2022 | ODI | 516 | 23709 |

--------------------------------------------------------------------------------------------------------------------

#replacement

import pandas as pd

data = pd.DataFrame({'  index': ['535', '536']})

print("Original data:")

print(data)

data['  index'].replace({'535': '5353'}, inplace=True)

print("\nModified data:")

print(data)

output:

Original data:

\tindex

0 535

1 536

Modified data:

\tindex

0 5353

1. 536

---------------------------------------------------------------------------------------

df.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 516 entries, 0 to 515

Data columns (total 8 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 index 516 non-null int64

1 runs 516 non-null int64

2 opponent 516 non-null object

3 ground 516 non-null object

4 date 516 non-null object

5 match 516 non-null object

6 Match\_No 516 non-null int64

7 total 516 non-null int64

dtypes: int64(4), object(4)

memory usage: 32.4+ KB

---------------------------------------------------------------------------------------

#data duplication function

df.nunique()

output:

index :516

runs: 142

opponent :14

ground :73

date: 443

match :3

Match\_No :516

total :484

dtype: int64

--------------------------------------------------------------------------

#missing value function

df.isnull().sum()

output:

index :0

runs :0

opponent: 0

ground :0

date :0

match :0

Match\_No: 0

total :0

dtype: int64

--------------------------------------------------------------------------

#is null().sum()

(df.isnull().sum()/(len(df)))\*100

output:

index :0.0

runs: 0.0

opponent: 0.0

ground :0.0

date: 0.0

match :0.0

Match\_No :0.0

total :0.0

dtype: float64

-------------------------------------------------------------------------

df = df.drop(['index'], axis=1)

df.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 516 entries, 0 to 515

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 runs 516 non-null int64

1 opponent 516 non-null object

2 ground 516 non-null object

3 date 516 non-null object

4 match 516 non-null object

5 Match\_No 516 non-null int64

6 total 516 non-null int64

dtypes: int64(3), object(4)

memory usage: 28.3+ KB

--------------------------------------------------------------------------

#unique function

print(df['runs'].unique())

print(df['total'].unique())

output:

[ 12 37 25 54 31 2 16 79 30 10 27 107 9 91 71 102 57 0

82 18 68 26 11 28 8 118 105 64 63 22 87 100 34 1 59 24

35 14 81 94 4 15 55 7 112 86 52 3 117 20 23 80 44 75

116 77 66 21 133 108 183 106 38 128 58 103 51 70 50 40 78 19

6 67 41 43 115 29 61 99 119 96 46 123 136 48 5 36 72 39

13 62 127 49 53 139 141 169 147 33 138 88 90 56 89 200 45 211

17 85 154 65 167 235 122 204 76 32 111 42 131 110 92 121 113 104

213 243 153 160 129 47 149 97 140 157 60 120 114 254 74 73]

[ 12 49 74 128 159 161 177 256 286 296 323 377

484 493 584 655 757 759 790 847 929 947 1015 1041

1052 1070 1080 1108 1116 1153 1271 1376 1440 1503 1505 1533

1587 1609 1637 1724 1726 1826 1834 1868 1880 1881 1940 1964

1973 2008 2022 2024 2105 2127 2221 2225 2240 2267 2297 2301

2356 2365 2372 2388 2495 2532 2644 2679 2765 2780 2832 2895

2898 3015 3035 3058 3138 3149 3172 3181 3225 3300 3416 3438

3460 3491 3522 3599 3617 3632 3644 3710 3731 3864 3972 4038

4221 4327 4328 4366 4494 4517 4585 4643 4746 4797 4867 4917

4957 4972 5050 5052 5071 5085 5104 5111 5117 5137 5240 5261

5299 5308 5335 5341 5348 5363 5400 5477 5503 5610 5644 5711

5745 5746 5787 5818 5840 5862 5920 5963 5974 5976 6078 6109

6111 6226 6240 6308 6337 6398 6498 6566 6681 6684 6741 6827

6926 6945 6976 7095 7191 7237 7248 7371 7449 7455 7457 7539

7543 7610 7648 7753 7889 7937 7942 7978 8032 8089 8112 8184

8261 8262 8270 8295 8334 8362 8369 8375 8395 8435 8436 8449

8515 8517 8579 8706 8728 8777 8830 8896 9035 9150 9291 9310

9311 9480 9534 9681 9727 9736 9740 9743 9751 9858 9904 9937

9970 10014 10052 10055 10056 10070 10071 10094 10119 10222 10225 10303

10313 10331 10352 10395 10396 10407 10419 10496 10634 10641 10642 10671

10693 10709 10753 10841 10932 10991 11108 11214 11222 11312 11371 11421

11428 11477 11533 11574 11597 11652 11676 11758 11847 12047 12091 12094

12098 12114 12123 12141 12150 12195 12406 12423 12508 12517 12671 12716

12781 12821 12870 13037 13118 13180 13186 13421 13436 13558 13566 13621

13650 13671 13673 13877 13915 13928 13940 13955 13961 14042 14118 14214

14219 14251 14338 14349 14352 14463 14502 14505 14608 14621 14663 14745

14749 14752 14883 14993 15075 15167 15195 15216 15255 15277 15398 15427

15540 15566 15631 15644 15748 15961 16204 16254 16259 16287 16440 16445

16499 16540 16652 16698 16858 16933 16969 17098 17124 17125 17134 17154

17201 17244 17319 17364 17435 17584 17635 17658 17675 17772 17875 17921

17979 18028 18167 18212 18352 18509 18616 18632 18665 18669 18730 18733

18767 18890 18907 18989 19012 19015 19119 19165 19210 19253 19313 19337

19409 19453 19569 19692 19699 19719 19737 19819 19896 19963 20035 20101

20127 20161 20162 20181 20209 20268 20388 20502 20511 20562 20638 20710

20719 20739 20770 21024 21036 21172 21266 21285 21355 21359 21444 21474

21500 21516 21594 21683 21728 21739 21777 21788 21839 21854 21863 21865

21884 21887 21901 21922 22011 22074 22083 22123 22208 22282 22286 22297

22369 22431 22458 22531 22608 22609 22689 22745 22811 22818 22862 22875

22917 22937 22944 22999 23049 23093 23150 23159 23161 23197 23232 23250

23329 23358 23409 23474 23482 23500 23517 23569 23614 23637 23650 23661

23681 23682 23693 23709]

--------------------------------------------------------------------------

import numpy as np

cat\_cols=df.select\_dtypes(include=['object']).columns

num\_cols=df.select\_dtypes(include=np.number).columns.tolist()

print("categorical variables:")

print(cat\_cols)

print("numerical variables:")

print(num\_cols)

output:

categorical variables:

Index(['opponent', 'ground', 'date', 'match'], dtype='object')

numerical variables:

['runs', 'Match\_No', 'total']

--------------------------------------------------------------------------

Preprocessing and Splitting:

df1 =df.dropna()

print(df1)

output:

runs opponent ground date match Match\_No total

0 12 SriLanka Dambulla 18Aug2008 ODI 1 12

1 37 SriLanka Dambulla 20Aug2008 ODI 2 49

2 25 SriLanka Colombo(RPS) 24Aug2008 ODI 3 74

3 54 SriLanka Colombo(RPS) 27Aug2008 ODI 4 128

4 31 SriLanka Colombo(RPS) 29Aug2008 ODI 5 159

.. ... ... ... ... ... ... ...

511 11 England Birmingham 1Jul2022 Test 512 23661

512 20 England Birmingham 1Jul2022 Test 513 23681

513 1 England Birmingham 9Jul2022 T20 514 23682

514 11 England Nottingham 10Jul2022 T20 515 23693

515 16 England Lord's 14Jul2022 ODI 516 23709

[516 rows x 7 columns]

-----------------------------------------------------------------------------------

df2=df.dropna(axis=1)

print(df2)

output:

runs opponent ground date match Match\_No total

0 12 SriLanka Dambulla 18Aug2008 ODI 1 12

1 37 SriLanka Dambulla 20Aug2008 ODI 2 49

2 25 SriLanka Colombo(RPS) 24Aug2008 ODI 3 74

3 54 SriLanka Colombo(RPS) 27Aug2008 ODI 4 128

4 31 SriLanka Colombo(RPS) 29Aug2008 ODI 5 159

.. ... ... ... ... ... ... ...

511 11 England Birmingham 1Jul2022 Test 512 23661

512 20 England Birmingham 1Jul2022 Test 513 23681

513 1 England Birmingham 9Jul2022 T20 514 23682

514 11 England Nottingham 10Jul2022 T20 515 23693

515 16 England Lord's 14Jul2022 ODI 516 23709

[516 rows x 7 columns]

---------------------------------------------------------------------------------------

df3 = df.fillna(df.mean())

print(df3)

output:

runs opponent ground date match Match\_No total

0 12 SriLanka Dambulla 18Aug2008 ODI 1 12

1 37 SriLanka Dambulla 20Aug2008 ODI 2 49

2 25 SriLanka Colombo(RPS) 24Aug2008 ODI 3 74

3 54 SriLanka Colombo(RPS) 27Aug2008 ODI 4 128

4 31 SriLanka Colombo(RPS) 29Aug2008 ODI 5 159

.. ... ... ... ... ... ... ...

511 11 England Birmingham 1Jul2022 Test 512 23661

512 20 England Birmingham 1Jul2022 Test 513 23681

513 1 England Birmingham 9Jul2022 T20 514 23682

514 11 England Nottingham 10Jul2022 T20 515 23693

515 16 England Lord's 14Jul2022 ODI 516 23709

[516 rows x 7 columns]

<ipython-input-82-058b506fa629>:1: FutureWarning: The default value of numeric\_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric\_only=None' is deprecated. Select only valid columns or specify the value of numeric\_only to silence this warning.

df3 = df.fillna(df.mean())

---------------------------------------------------------------------------------------

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

df.dropna(inplace=True)

X = df[['total', 'runs']]

y = df['Match\_No']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

output:

Mean Squared Error: 85.10542195383287

R-squared: 0.9964209723458826

---------------------------------------------------------------------------------------

Simple linear regression and multi linear regression:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

df = pd.read\_csv("/content/final (1).csv")

X = df["index"].values.reshape(-1, 1)

y = df["total"].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

y\_pred\_test = regressor.predict(x\_test)

y\_pred\_train = regressor.predict(x\_train)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred\_test, color="red", label="Regression Line")

plt.title("Linear Regression")

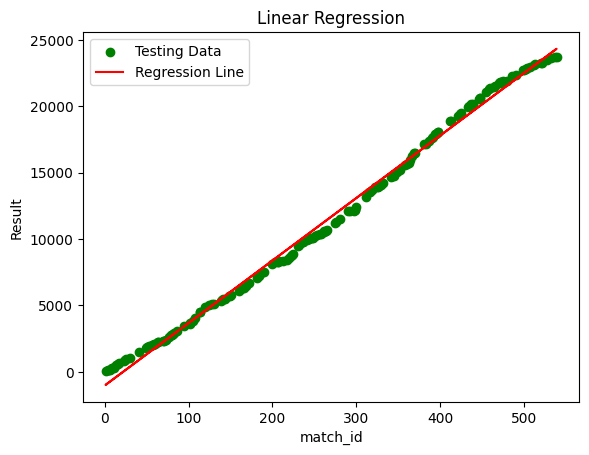
plt.xlabel("match\_id")

plt.ylabel("Result")

plt.legend()

plt.show()

output:



import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

df = pd.read\_csv("/content/final (1).csv")

X = df[["total"]]

y = df["index"]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

model = LinearRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred, color="red", label="Regression Line")

plt.title("multiple Linear Regression")

plt.xlabel("Item Code")

plt.ylabel("Year Code")

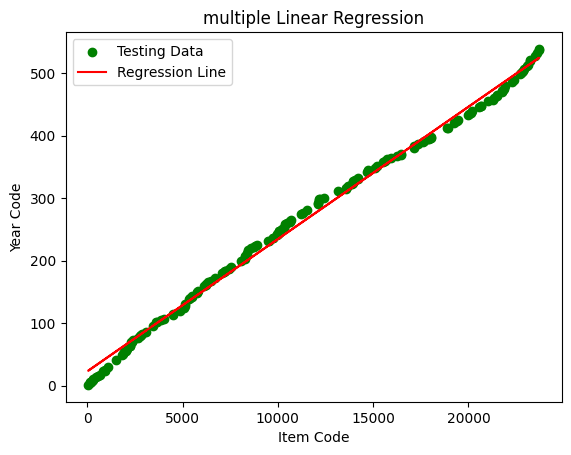
plt.legend()

plt.show()

output:

Mean Squared Error: 109.0633884467426

R-squared: 0.9954820535804508



(d). Crop yield - past data

**Dataset name: crop yield recommandmention**

* Perform data exploration:

import pandas as pd

df = pd.read\_csv("/content/Crop\_recommendation.csv")

df.describe()

output:

|  | **N** | **P** | **K** | **temperature** | **humidity** | **ph** | **rainfall** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 2200.000000 | 2200.000000 | 2200.000000 | 2200.000000 | 2200.000000 | 2200.000000 | 2200.000000 |
| **mean** | 50.551818 | 53.362727 | 48.149091 | 25.616244 | 71.481779 | 6.469480 | 103.463655 |
| **std** | 36.917334 | 32.985883 | 50.647931 | 5.063749 | 22.263812 | 0.773938 | 54.958389 |
| **min** | 0.000000 | 5.000000 | 5.000000 | 8.825675 | 14.258040 | 3.504752 | 20.211267 |
| **25%** | 21.000000 | 28.000000 | 20.000000 | 22.769375 | 60.261953 | 5.971693 | 64.551686 |
| **50%** | 37.000000 | 51.000000 | 32.000000 | 25.598693 | 80.473146 | 6.425045 | 94.867624 |
| **75%** | 84.250000 | 68.000000 | 49.000000 | 28.561654 | 89.948771 | 6.923643 | 124.267508 |
| **max** | 140.000000 | 145.000000 | 205.000000 | 43.675493 | 99.981876 | 9.935091 | 298.560117 |

---------------------------------------------------------------------------------------

df.head()

output:

|  | **N** | **P** | **K** | **temperature** | **humidity** | **ph** | **rainfall** | **label** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 90 | 42 | 43 | 20.879744 | 82.002744 | 6.502985 | 202.935536 | rice |
| **1** | 85 | 58 | 41 | 21.770462 | 80.319644 | 7.038096 | 226.655537 | rice |
| **2** | 60 | 55 | 44 | 23.004459 | 82.320763 | 7.840207 | 263.964248 | rice |
| **3** | 74 | 35 | 40 | 26.491096 | 80.158363 | 6.980401 | 242.864034 | rice |
| **4** | 78 | 42 | 42 | 20.130175 | 81.604873 | 7.628473 | 262.717340 | rice |

---------------------------------------------------------------------------------------

df.tail()

output:

|  | **N** | **P** | **K** | **temperature** | **humidity** | **ph** | **rainfall** | **label** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2195** | 107 | 34 | 32 | 26.774637 | 66.413269 | 6.780064 | 177.774507 | coffee |
| **2196** | 99 | 15 | 27 | 27.417112 | 56.636362 | 6.086922 | 127.924610 | coffee |
| **2197** | 118 | 33 | 30 | 24.131797 | 67.225123 | 6.362608 | 173.322839 | coffee |
| **2198** | 117 | 32 | 34 | 26.272418 | 52.127394 | 6.758793 | 127.175293 | coffee |
| **2199** | 104 | 18 | 30 | 23.603016 | 60.396475 | 6.779833 | 140.937041 | coffee |

-------------------------------------------------------------------------------------

#replacement

import pandas as pd

data = pd.DataFrame({'N': ['107', '99']})

print("Original data:")

print(data)

data['N'].replace({'107': '100'}, inplace=True)

print("\nModified data:")

print(data)

output:

Original data:

N

0 107

1 99

Modified data:

N

0 100

1. 1 99

----------------------------------------------------------------------------------------------------

df.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2200 entries, 0 to 2199

Data columns (total 8 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 N 2200 non-null int64

1 P 2200 non-null int64

2 K 2200 non-null int64

3 temperature 2200 non-null float64

4 humidity 2200 non-null float64

5 ph 2200 non-null float64

6 rainfall 2200 non-null float64

7 label 2200 non-null object

dtypes: float64(4), int64(3), object(1)

memory usage: 137.6+ KB

----------------------------------------------------------------------------------------------------

#data duplication function

df.nunique()

output:

N: 137

P: 117

K :73

Temperature: 2200

humidity :2200

ph :2200

rainfall: 2200

label :22

dtype: int64

----------------------------------------------------------------------------------------------------

#missing value function

df.isnull().sum()

output:

N :0

P: 0

K: 0

Temperature: 0

Humidity: 0

ph :0

rainfall :0

label :0

dtype: int64

----------------------------------------------------------------------------------------------------

#is null().sum()

(df.isnull().sum()/(len(df)))\*100

output:

N :0.0

P: 0.0

K 0.0

temperature :0.0

humidity: 0.0

ph :0.0

rainfall :0.0

label: 0.0

dtype: float64

---------------------------------------------------------------------------------------------------------------------

df = df.drop(['N'], axis=1)

df.info()

output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2200 entries, 0 to 2199

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 P 2200 non-null int64

1 K 2200 non-null int64

2 temperature 2200 non-null float64

3 humidity 2200 non-null float64

4 ph 2200 non-null float64

5 rainfall 2200 non-null float64

6 label 2200 non-null object

dtypes: float64(4), int64(2), object(1)

memory usage: 120.4+ KB

--------------------------------------------------------------------------

from datetime import date

df['P'] = date.today().year - df['K']

df.head()

output:

|  | **P** | **K** | **temperature** | **humidity** | **ph** | **rainfall** | **label** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1980 | 43 | 20.879744 | 82.002744 | 6.502985 | 202.935536 | rice |
| **1** | 1982 | 41 | 21.770462 | 80.319644 | 7.038096 | 226.655537 | rice |
| **2** | 1979 | 44 | 23.004459 | 82.320763 | 7.840207 | 263.964248 | rice |
| **3** | 1983 | 40 | 26.491096 | 80.158363 | 6.980401 | 242.864034 | rice |
| **4** | 1981 | 42 | 20.130175 | 81.604873 | 7.628473 | 262.717340 | rice |

--------------------------------------------------------------------------

#unique function

print(df['temperature'].unique())

print(df['humidity'].unique())

output:

[20.87974371 21.77046169 23.00445915 ... 24.13179691 26.2724184

23.60301571]

[82.00274423 80.31964408 82.3207629 ... 67.22512329 52.12739421

60.39647474]

----------------------------------------------------------------------------------------------------

import numpy as np

cat\_cols=df.select\_dtypes(include=['object']).columns

num\_cols=df.select\_dtypes(include=np.number).columns.tolist()

print("categorical variables:")

print(cat\_cols)

print("numerical variables:")

print(num\_cols)

output:

categorical variables:

Index(['label'], dtype='object')

numerical variables:

['P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']

----------------------------------------------------------------------------------------------------

Preprocessing and Splitting:

df1 =df.dropna()

print(df1)

output:

P K temperature humidity ph rainfall label

0 1980 43 20.879744 82.002744 6.502985 202.935536 rice

1 1982 41 21.770462 80.319644 7.038096 226.655537 rice

2 1979 44 23.004459 82.320763 7.840207 263.964248 rice

3 1983 40 26.491096 80.158363 6.980401 242.864034 rice

4 1981 42 20.130175 81.604873 7.628473 262.717340 rice

... ... .. ... ... ... ... ...

2195 1991 32 26.774637 66.413269 6.780064 177.774507 coffee

2196 1996 27 27.417112 56.636362 6.086922 127.924610 coffee

2197 1993 30 24.131797 67.225123 6.362608 173.322839 coffee

2198 1989 34 26.272418 52.127394 6.758793 127.175293 coffee

2199 1993 30 23.603016 60.396475 6.779833 140.937041 coffee

[2200 rows x 7 columns]

------------------------------------------------------------------------------------------------------------

df2=df.dropna(axis=1)

print(df2)

output:

P K temperature humidity ph rainfall label

0 1980 43 20.879744 82.002744 6.502985 202.935536 rice

1 1982 41 21.770462 80.319644 7.038096 226.655537 rice

2 1979 44 23.004459 82.320763 7.840207 263.964248 rice

3 1983 40 26.491096 80.158363 6.980401 242.864034 rice

4 1981 42 20.130175 81.604873 7.628473 262.717340 rice

... ... .. ... ... ... ... ...

2195 1991 32 26.774637 66.413269 6.780064 177.774507 coffee

2196 1996 27 27.417112 56.636362 6.086922 127.924610 coffee

2197 1993 30 24.131797 67.225123 6.362608 173.322839 coffee

2198 1989 34 26.272418 52.127394 6.758793 127.175293 coffee

2199 1993 30 23.603016 60.396475 6.779833 140.937041 coffee

[2200 rows x 7 columns]

----------------------------------------------------------------------------------------------------

df3 = df.fillna(df.mean())

print(df3)

output:

P K temperature humidity ph rainfall label

0 1980 43 20.879744 82.002744 6.502985 202.935536 rice

1 1982 41 21.770462 80.319644 7.038096 226.655537 rice

2 1979 44 23.004459 82.320763 7.840207 263.964248 rice

3 1983 40 26.491096 80.158363 6.980401 242.864034 rice

4 1981 42 20.130175 81.604873 7.628473 262.717340 rice

... ... .. ... ... ... ... ...

2195 1991 32 26.774637 66.413269 6.780064 177.774507 coffee

2196 1996 27 27.417112 56.636362 6.086922 127.924610 coffee

2197 1993 30 24.131797 67.225123 6.362608 173.322839 coffee

2198 1989 34 26.272418 52.127394 6.758793 127.175293 coffee

2199 1993 30 23.603016 60.396475 6.779833 140.937041 coffee

[2200 rows x 7 columns]

<ipython-input-109-058b506fa629>:1: FutureWarning: The default value of numeric\_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric\_only=None' is deprecated. Select only valid columns or specify the value of numeric\_only to silence this warning.

df3 = df.fillna(df.mean())

-------------------------------------------------------------------------------------------------------------------------------------------

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

df.dropna(inplace=True)

X = df[['P', 'K']]

y = df['temperature']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

output:

Mean Squared Error: 24.09338287615048

R-squared: 0.018452706532588237

----------------------------------------------------------------------------------------------------

Simple linear regression and multi linear regression:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

df=pd.read\_csv("/content/Crop\_recommendation.csv")

X = df["P"].values.reshape(-1, 1)

y = df["N"].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

y\_pred\_test = regressor.predict(x\_test)

y\_pred\_train = regressor.predict(x\_train)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred\_test, color="red", label="Regression Line")

plt.title("Simple Linear Regression")

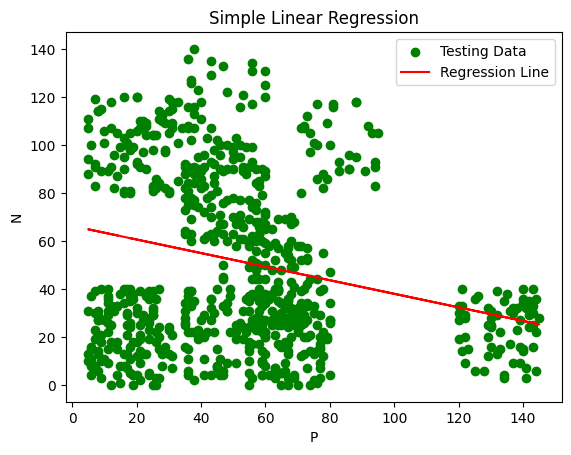
plt.xlabel("P")

plt.ylabel("N")

plt.legend()

plt.show()

output:



---------------------------------------------------------------------------------------

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

df=pd.read\_csv("/content/Crop\_recommendation.csv")

X = df[["rainfall"]]

y = df["K"]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

model = LinearRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

plt.scatter(x\_test, y\_test, color="green", label="Testing Data")

plt.plot(x\_test, y\_pred, color="red", label="Regression Line")

plt.title("multiple Linear Regression")

plt.xlabel("rainfall")

plt.ylabel("K")

plt.legend()

plt.show()

output:

Mean Squared Error: 2491.7091792359383

R-squared: 0.003102227209811792

