Start coding or generate with AI.

Start coding or generate with AI.

```
import pandas as pd

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

print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
17        45     90       112       NaN
18        60    103       123     323.0
19        45     97       125     243.0
20        60    108       131     364.2
21        45    100       119     282.0
22        60    130       101     300.0
23        45    105       132     246.0
24        60    102       126     334.5
25        60    100       120     250.0
26        60     92       118     241.0
27        60    103       132       NaN
28        60    100       132     280.0
29        60    102       129     380.3
30        60     92       115     243.0
31        45     90       112     180.1
32        60    101       124     299.0
33        60     93       113     223.0
34        60    107       136     361.0
35        60    114       140     415.0
36        60    102       127     300.0
37        60    100       120     300.0
38        60    100       120     300.0
39        45    104       129     266.0
40        45     90       112     180.1
```

```
41        60    98       126     286.0
42        60   100       122     329.4
43        60   111       138     400.0
44        60   111       131     397.0
45        60    99       119     273.0
46        60   109       153     387.6
47        45   111       136     300.0
48        45   108       129     298.0
49        60   111       139     397.6
50        60   107       136     380.2
51        80   123       146     643.1
52        60   106       130     263.0
53        60   118       151     486.0
54        30   136       175     238.0
55        60   121       146     450.7
56        60   118       121     413.0
```

Start coding or generate with AI.

```
import pandas as pd

print(pd.options.display.max_rows)
```

```
60
```

```
import pandas as pd

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

print(df.head(20))
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
17        45     90       112       NaN
18        60    103       123     323.0
19        45     97       125     243.0
```

```
import pandas as pd

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

print(df.head())
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
```

```
print(df.tail())
```

```
     Duration  Pulse  Maxpulse  Calories
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

```
import pandas as pd

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

new_df = df.dropna()

print(new_df.to_string())
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
```

```
5          60  '2020/12/06'    102        127      300.0
6          60  '2020/12/07'    110        136      374.0
7         450  '2020/12/08'    104        134      253.3
8          30  '2020/12/09'    109        133      195.1
9          60  '2020/12/10'     98        124      269.0
10         60  '2020/12/11'    103        147      329.3
11         60  '2020/12/12'    100        120      250.7
12         60  '2020/12/12'    100        120      250.7
13         60  '2020/12/13'    106        128      345.3
14         60  '2020/12/14'    104        132      379.3
15         60  '2020/12/15'     98        123      275.0
16         60  '2020/12/16'     98        120      215.2
17         60  '2020/12/17'    100        120      300.0
19         60  '2020/12/19'    103        123      323.0
20         45  '2020/12/20'     97        125      243.0
21         60  '2020/12/21'    108        131      364.2
23         60  '2020/12/23'    130        101      300.0
24         45  '2020/12/24'    105        132      246.0
25         60  '2020/12/25'    102        126      334.5
26         60      20201226    100        120      250.0
27         60  '2020/12/27'     92        118      241.0
29         60  '2020/12/29'    100        132      280.0
30         60  '2020/12/30'    102        129      380.3
31         60  '2020/12/31'     92        115      243.0
```

Start coding or generate with AI.

```
import pandas as pd

df = pd.read_csv('/dirtydata.csv')
new_df = df.dropna()

new_df.fillna(130, inplace = True)
print(df)
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0          60  '2020/12/01'    110        130      409.1
1          60  '2020/12/02'    117        145      479.0
2          60  '2020/12/03'    103        135      340.0
3          45  '2020/12/04'    109        175      282.4
4          45  '2020/12/05'    117        148      406.0
5          60  '2020/12/06'    102        127      300.0
6          60  '2020/12/07'    110        136      374.0
7         450  '2020/12/08'    104        134      253.3
8          30  '2020/12/09'    109        133      195.1
9          60  '2020/12/10'     98        124      269.0
10         60  '2020/12/11'    103        147      329.3
11         60  '2020/12/12'    100        120      250.7
12         60  '2020/12/12'    100        120      250.7
13         60  '2020/12/13'    106        128      345.3
14         60  '2020/12/14'    104        132      379.3
15         60  '2020/12/15'     98        123      275.0
16         60  '2020/12/16'     98        120      215.2
17         60  '2020/12/17'    100        120      300.0
```

```
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
<ipython-input-2-01a4964b0107>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  new_df.fillna(130, inplace = True)
```

```
import pandas as pd

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

x = df["Calories"].mean()

df["Calories"].fillna(x, inplace = True)
print(df)
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130    409.10
1         60  '2020/12/02'    117       145    479.00
2         60  '2020/12/03'    103       135    340.00
3         45  '2020/12/04'    109       175    282.40
4         45  '2020/12/05'    117       148    406.00
5         60  '2020/12/06'    102       127    300.00
6         60  '2020/12/07'    110       136    374.00
7        450  '2020/12/08'    104       134    253.30
8         30  '2020/12/09'    109       133    195.10
9         60  '2020/12/10'     98       124    269.00
10        60  '2020/12/11'    103       147    329.30
11        60  '2020/12/12'    100       120    250.70
12        60  '2020/12/12'    100       120    250.70
13        60  '2020/12/13'    106       128    345.30
14        60  '2020/12/14'    104       132    379.30
15        60  '2020/12/15'     98       123    275.00
16        60  '2020/12/16'     98       120    215.20
17        60  '2020/12/17'    100       120    300.00
18        45  '2020/12/18'     90       112    304.68
19        60  '2020/12/19'    103       123    323.00
20        45  '2020/12/20'     97       125    243.00
21        60  '2020/12/21'    108       131    364.20
```

```
22        45           NaN    100       119    282.00
23        60  '2020/12/23'    130       101    300.00
24        45  '2020/12/24'    105       132    246.00
25        60  '2020/12/25'    102       126    334.50
26        60      20201226    100       120    250.00
27        60  '2020/12/27'     92       118    241.00
28        60  '2020/12/28'    103       132    304.68
29        60  '2020/12/29'    100       132    280.00
30        60  '2020/12/30'    102       129    380.30
31        60  '2020/12/31'     92       115    243.00
```

```
 import pandas as pd

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

x = df["Calories"].mode()[0]

df["Calories"].fillna(x, inplace = True)
print(df)
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     300.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132     300.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Start coding or generate with AI.

Start coding or generate with AI.

Double-click (or enter) to edit

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
import pandas as pd

df = pd.read_csv('/dirtydata.csv')

#df.loc[7,'Duration'] = 450
df['Duration'][7]=30

print(df)
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7         30  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
```

```
26         60      20201226    100        120      250.0
27         60  '2020/12/27'     92        118      241.0
28         60  '2020/12/28'    103        132        NaN
29         60  '2020/12/29'    100        132      280.0
30         60  '2020/12/30'    102        129      380.3
31         60  '2020/12/31'     92        115      243.0
<ipython-input-5-b717fedb8738>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  df['Duration'][7]=30
```

Start coding or generate with AI.

```
import pandas as pd

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

x = df["Calories"].median()

df["Calories"].fillna(x, inplace = True)

print(df.to_string())
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     291.2
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
```

```
    27        60  '2020/12/27'     92       118     241.0
    28        60  '2020/12/28'    103       132     291.2
    29        60  '2020/12/29'    100       132     280.0
    30        60  '2020/12/30'    102       129     380.3
    31        60  '2020/12/31'     92       115     243.0
```

Start coding or generate with AI.

```
import pandas as pd

df = pd.read_csv('/data (2).csv')

print(df.to_string())
```

Show hidden output

```
import pandas as pd

df = pd.read_csv('/data (2).csv')

print(df.corr())
```

```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922717
Pulse    -0.155408  1.000000  0.786535  0.025121
Maxpulse  0.009403  0.786535  1.000000  0.203813
Calories  0.922717  0.025121  0.203813  1.000000
```

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('/data (2).csv')

df.plot()

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-cb0037b66813> in <cell line: 16>()
     14 
     15 #Two  lines to make our compiler able to draw:
---> 16 plt.savefig(sys.stdout.buffer)
     17 sys.stdout.flush()
     18 

AttributeError: 'OutStream' object has no attribute 'buffer'
```

```
from google.colab import drive
drive.mount('/content/drive')


#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('/dirtydata.csv')

df.plot()

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-584da5154f8d> in <cell line: 16>()
     14 
     15 #Two  lines to make our compiler able to draw:
---> 16 plt.savefig(sys.stdout.buffer)
     17 sys.stdout.flush()

AttributeError: 'OutStream' object has no attribute 'buffer'
```

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
```

```
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2],
  'Reating': [2, 3, 1]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings  Reating
0    BMW         3        2
1  Volvo         7        3
2   Ford         2        1
```

```
#/dirtydata.csv

import pandas as pd

df = pd.read_csv('/dirtydata.csv')

#new_df = df.dropna()

print(df)
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
```

```
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

```
import pandas as pd

df = pd.read_csv('/dirtydata.csv')

new_df = df.dropna()

print(new_df.to_string())
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
```

```
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

```
import pandas as pd

df = pd.read_csv('/dirtydata.csv')

df.fillna(9, inplace = True)

print(df.to_string())
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       9.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45             9    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       9.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Start coding or generate with AI.

Data Science Class no-17

```
import pandas as pd
import numpy as np
bigmart_train=pd.read_csv('/content/Test.csv')
bigmart_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5681 entries, 0 to 5680
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            5681 non-null   object
 1   Item_Weight                4705 non-null   float64
 2   Item_Fat_Content           5681 non-null   object
 3   Item_Visibility            5681 non-null   float64
 4   Item_Type                  5681 non-null   object
 5   Item_MRP                   5681 non-null   float64
 6   Outlet_Identifier          5681 non-null   object
 7   Outlet_Establishment_Year  5681 non-null   int64
 8   Outlet_Size                4075 non-null   object
 9   Outlet_Location_Type       5681 non-null   object
 10  Outlet_Type                5681 non-null   object
dtypes: float64(3), int64(1), object(7)
memory usage: 488.3+ KB
```

```
print(bigmart_train['Item_Identifier'].unique(),bigmart_train['Item_Fat_Content'].unique(),b
```

```
['FDW58' 'FDW14' 'NCN55' ... 'NCI29' 'FDP28' 'FDF04'] ['Low Fat' 'reg' 'Regular' 'LF' ']
 'Health and Hygiene' 'Breads' 'Hard Drinks' 'Seafood' 'Soft Drinks'
 'Household' 'Frozen Foods' 'Meat' 'Canned' 'Starchy Foods' 'Breakfast'] ['OUT049' 'OUT0
 'OUT013' 'OUT035']
```

```
import pandas as pd
import numpy as np
bigmart_train=pd.read_csv('/content/Test.csv')
#bigmart_train.info()
print(bigmart_train)
```

```
     Item_Identifier  Item_Weight Item_Fat_Content  Item_Visibility  \
0              FDW58       20.750          Low Fat         0.007565
1              FDW14        8.300              reg         0.038428
2              NCN55       14.600          Low Fat         0.099575
3              FDQ58        7.315          Low Fat         0.015388
4              FDY38          NaN          Regular         0.118599
...              ...          ...              ...              ...
5676           FDB58       10.500          Regular         0.013496
5677           FDD47        7.600          Regular         0.142991
5678           NCO17       10.000          Low Fat         0.073529
5679           FDJ26       15.300          Regular         0.000000
5680           FDU37        9.500          Regular         0.104720

               Item_Type  Item_MRP Outlet_Identifier  \
```

```
0           Snack Foods  107.8622           OUT049
1                 Dairy   87.3198           OUT017
2                Others  241.7538           OUT010
3           Snack Foods  155.0340           OUT017
4                 Dairy  234.2300           OUT027
...                 ...       ...              ...
5676        Snack Foods  141.3154           OUT046
5677      Starchy Foods  169.1448           OUT018
5678  Health and Hygiene  118.7440           OUT045
5679             Canned  214.6218           OUT017
5680             Canned   79.7960           OUT045

      Outlet_Establishment_Year Outlet_Size Outlet_Location_Type  \
0                          1999      Medium               Tier 1
1                          2007         NaN               Tier 2
2                          1998         NaN               Tier 3
3                          2007         NaN               Tier 2
4                          1985      Medium               Tier 3
...                         ...         ...                  ...
5676                       1997       Small               Tier 1
5677                       2009      Medium               Tier 3
5678                       2002         NaN               Tier 2
5679                       2007         NaN               Tier 2
5680                       2002         NaN               Tier 2

            Outlet_Type
0     Supermarket Type1
1     Supermarket Type1
2         Grocery Store
3     Supermarket Type1
4     Supermarket Type3
...                 ...
5676  Supermarket Type1
5677  Supermarket Type2
5678  Supermarket Type1
5679  Supermarket Type1
5680  Supermarket Type1

[5681 rows x 11 columns]
```

```
print(bigmart_train['Item_Identifier'].unique(),bigmart_train['Item_Fat_Content'].unique(),b
```

```
['FDW58' 'FDW14' 'NCN55' ... 'NCI29' 'FDP28' 'FDF04'] ['Low Fat' 'reg' 'Regular' 'LF' ']
 'Health and Hygiene' 'Breads' 'Hard Drinks' 'Seafood' 'Soft Drinks'
 'Household' 'Frozen Foods' 'Meat' 'Canned' 'Starchy Foods' 'Breakfast'] ['OUT049' 'OUT0
 'OUT013' 'OUT035'] ['Supermarket Type1' 'Grocery Store' 'Supermarket Type3'
 'Supermarket Type2']
```

```
# Identify columns with object (string) or category data types
categorical_columns = bigmart_train.select_dtypes(include=['object', 'category', 'bool']).cc
print("Categorical Columns Based on Data Type:")
print(categorical_columns)
```

```
Categorical Columns Based on Data Type:
['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier', 'Outlet_Size',
```

One hot Encoding

```
from sklearn.preprocessing import OneHotEncoder

def one_hot_encode(bigmart_train, categorical_columns):

    # Initialize the OneHotEncoder
    encoder_bigmart = OneHotEncoder(sparse_output=False, drop='first')  # drop='first' to av

    # Fit and transform the categorical columns
    encoded_data_mart = encoder_bigmart.fit_transform(bigmart_train[categorical_columns])

    # Get the new column names
    encoded_columns_new = encoder_bigmart.get_feature_names_out(categorical_columns)

    # Create a DataFrame with the encoded data
    encoded_df_new_data = pd.DataFrame(encoded_data_mart, columns=encoded_columns_new)

    # Drop the original categorical columns and concatenate the new one-hot encoded columns
    bigmart_train = bigmart_train.drop(categorical_columns, axis=1)
    bigmart_train = pd.concat([bigmart_train, encoded_df_new_data], axis=1)

    return bigmart_train




    # Identify categorical columns
categorical_columns = bigmart_train.select_dtypes(include=['object', 'category']).columns.tc

# Call the function to encode the DataFrame
bigmart_train_encoded = one_hot_encode(bigmart_train.copy(), categorical_columns)

# Print the encoded DataFrame
#print(bigmart_train_encoded)
print(bigmart_train_encoded.head())
```

```
   Item_Weight  Item_Visibility  Item_MRP  Outlet_Establishment_Year  \
0       20.750         0.007565  107.8622                       1999
1        8.300         0.038428   87.3198                       2007
2       14.600         0.099575  241.7538                       1998
3        7.315         0.015388  155.0340                       2007
4          NaN         0.118599  234.2300                       1985

   Item_Identifier_DRA24  Item_Identifier_DRA59  Item_Identifier_DRB01  \
```

```
0                  0.0                    0.0                    0.0
1                  0.0                    0.0                    0.0
2                  0.0                    0.0                    0.0
3                  0.0                    0.0                    0.0
4                  0.0                    0.0                    0.0

   Item_Identifier_DRB13  Item_Identifier_DRB24  Item_Identifier_DRB25  ...  \
0                  0.0                    0.0                    0.0  ...
1                  0.0                    0.0                    0.0  ...
2                  0.0                    0.0                    0.0  ...
3                  0.0                    0.0                    0.0  ...
4                  0.0                    0.0                    0.0  ...

   Outlet_Identifier_OUT046  Outlet_Identifier_OUT049  Outlet_Size_Medium  \
0                     0.0                       1.0                 1.0
1                     0.0                       0.0                 0.0
2                     0.0                       0.0                 0.0
3                     0.0                       0.0                 0.0
4                     0.0                       0.0                 1.0

   Outlet_Size_Small  Outlet_Size_nan  Outlet_Location_Type_Tier 2  \
0                0.0              0.0                          0.0
1                0.0              1.0                          1.0
2                0.0              1.0                          0.0
3                0.0              1.0                          1.0
4                0.0              0.0                          0.0

   Outlet_Location_Type_Tier 3  Outlet_Type_Supermarket Type1  \
0                        0.0                            1.0
1                        0.0                            1.0
2                        1.0                            0.0
3                        0.0                            1.0
4                        1.0                            0.0

   Outlet_Type_Supermarket Type2  Outlet_Type_Supermarket Type3
0                          0.0                            0.0
1                          0.0                            0.0
2                          0.0                            0.0
3                          0.0                            0.0
4                          0.0                            1.0

[5 rows x 1582 columns]
```

Double-click (or enter) to edit

```
    # Identify categorical columns
categorical_columns = bigmart_train.select_dtypes(include=['object', 'category']).columns.tc

# Call the function to encode the DataFrame
bigmart_train_encoded = one_hot_encode(bigmart_train.copy(), categorical_columns)

# Print the encoded DataFrame
```

```
#print(bigmart_train_encoded)
print(bigmart_train_encoded.head())
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming your dataset is already loaded as 'df'

# Identify categorical columns (non-numerical columns)
categorical_columns = bigmart_train.select_dtypes(include=['object', 'category']).columns.tc

# Initialize LabelEncoder
label_encoders = {}

# Apply label encoding to each categorical column
for col in categorical_columns:
    label_encoders[col] = LabelEncoder()
    bigmart_train[col] = label_encoders[col].fit_transform(bigmart_train[col])

#print("Dataset after Label Encoding:")
#print(bigmart_train.head())
bigmart_train.info()
```

Data Info Finding

```
import pandas as pd
import numpy as np
bigmart_train=pd.read_csv('/content/customer.csv')
bigmart_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        50 non-null     int64 
 1   gender     50 non-null     object
 2   review     50 non-null     object
 3   education  50 non-null     object
 4   purchased  50 non-null     object
dtypes: int64(1), object(4)
memory usage: 2.1+ KB
```

```
print(bigmart_train['age'].unique(),bigmart_train['gender'].unique(),bigmart_train['rev
```

```
[30 68 70 72 16 31 18 60 65 74 98 51 57 15 75 59 22 19 97 32 96 53 69 48
 83 73 92 89 86 34 94 45 76 39 23 27 77 61 64 38 25] ['Female' 'Male'] ['Average' 'Poor'
```

One Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder

def one_hot_encode(bigmart_train, categorical_columns):

    # Initialize the OneHotEncoder
    encoder_bigmart = OneHotEncoder(sparse_output=False, drop='first')  # drop='first' 

    # Fit and transform the categorical columns
    encoded_data_mart = encoder_bigmart.fit_transform(bigmart_train[categorical_columns

    # Get the new column names
    encoded_columns_new = encoder_bigmart.get_feature_names_out(categorical_columns)

    # Create a DataFrame with the encoded data
    encoded_df_new_data = pd.DataFrame(encoded_data_mart, columns=encoded_columns_new)

    # Drop the original categorical columns and concatenate the new one-hot encoded col
    bigmart_train = bigmart_train.drop(categorical_columns, axis=1)
    bigmart_train = pd.concat([bigmart_train, encoded_df_new_data], axis=1)

    return bigmart_train

 # Identify categorical columns
categorical_columns = bigmart_train.select_dtypes(include=['object', 'category']).colum

# Call the function to encode the DataFrame
bigmart_train_encoded = one_hot_encode(bigmart_train.copy(), categorical_columns)

# Print the encoded DataFrame
#print(bigmart_train_encoded)
print(bigmart_train_encoded.head())
print(bigmart_train_encoded.head())
```

```
   age  gender_Male  review_Good  review_Poor  education_School  education_UG  \
0   30          0.0          0.0          0.0               1.0           0.0
1   68          0.0          0.0          1.0               0.0           1.0
2   70          0.0          1.0          0.0               0.0           0.0
3   72          0.0          1.0          0.0               0.0           0.0
4   16          0.0          0.0          0.0               0.0           1.0

   purchased_Yes
0            0.0
1            0.0
2            0.0
3            0.0
4            0.0
```

```
from sklearn.preprocessing import OneHotEncoder

def one_hot_encode(bigmart_train, categorical_columns):

    # Initialize the OneHotEncoder
    encoder_bigmart = OneHotEncoder(sparse_output=False, drop='first')  # drop='first' to av

    # Fit and transform the categorical columns
    encoded_data_mart = encoder_bigmart.fit_transform(bigmart_train[categorical_columns])

    # Get the new column names
    encoded_columns_new = encoder_bigmart.get_feature_names_out(categorical_columns)

    # Create a DataFrame with the encoded data
    encoded_df_new_data = pd.DataFrame(encoded_data_mart, columns=encoded_columns_new)

    # Drop the original categorical columns and concatenate the new one-hot encoded columns
    bigmart_train = bigmart_train.drop(categorical_columns, axis=1)
    bigmart_train = pd.concat([bigmart_train, encoded_df_new_data], axis=1)

    return bigmart_train

 # Identify categorical columns
categorical_columns = bigmart_train.select_dtypes(include=['object', 'category']).columns.tc

# Call the function to encode the DataFrame
bigmart_train_encoded = one_hot_encode(bigmart_train.copy(), categorical_columns)

# Print the encoded DataFrame
#print(bigmart_train_encoded)
print(bigmart_train_encoded.head())
print(bigmart_train_encoded.head())
```

Start coding or generate with AI.

Class no 18

Start coding or generate with AI.

```
import pandas
filename = '/content/pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(filename, names=names)
print(data.shape)
```

```
(768, 9)
```

```
# Load CSV using Pandas
import pandas
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(filename, names=names)
print(data.shape)
```

```
(768, 9)
```

Start coding or generate with AI.

```
# Load CSV using Pandas from URL
import pandas
url = "/content/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(url, names=names)
print (data.to_string())
```

```
    preg  plas  pres  skin  test  mass   pedi  age  class
0      6   148    72    35     0  33.6  0.627   50      1
1      1    85    66    29     0  26.6  0.351   31      0
2      8   183    64     0     0  23.3  0.672   32      1
3      1    89    66    23    94  28.1  0.167   21      0
4      0   137    40    35   168  43.1  2.288   33      1
5      5   116    74     0     0  25.6  0.201   30      0
6      3    78    50    32    88  31.0  0.248   26      1
7     10   115     0     0     0  35.3  0.134   29      0
8      2   197    70    45   543  30.5  0.158   53      1
9      8   125    96     0     0   0.0  0.232   54      1
10     4   110    92     0     0  37.6  0.191   30      0
11    10   168    74     0     0  38.0  0.537   34      1
12    10   139    80     0     0  27.1  1.441   57      0
13     1   189    60    23   846  30.1  0.398   59      1
14     5   166    72    19   175  25.8  0.587   51      1
15     7   100     0     0     0  30.0  0.484   32      1
16     0   118    84    47   230  45.8  0.551   31      1
17     7   107    74     0     0  29.6  0.254   31      1
18     1   103    30    38    83  43.3  0.183   33      0
19     1   115    70    30    96  34.6  0.529   32      1
20     3   126    88    41   235  39.3  0.704   27      0
21     8    99    84     0     0  35.4  0.388   50      0
22     7   196    90     0     0  39.8  0.451   41      1
23     9   119    80    35     0  29.0  0.263   29      1
24    11   143    94    33   146  36.6  0.254   51      1
25    10   125    70    26   115  31.1  0.205   41      1
26     7   147    76     0     0  39.4  0.257   43      1
27     1    97    66    15   140  23.2  0.487   22      0
28    13   145    82    19   110  22.2  0.245   57      0
29     5   117    92     0     0  34.1  0.337   38      0
30     5   109    75    26     0  36.0  0.546   60      0
```

```
31     3  158   76   36  245  31.6  0.851  28     1
32     3   88   58   11   54  24.8  0.267  22     0
33     6   92   92    0    0  19.9  0.188  28     0
34    10  122   78   31    0  27.6  0.512  45     0
35     4  103   60   33  192  24.0  0.966  33     0
36    11  138   76    0    0  33.2  0.420  35     0
37     9  102   76   37    0  32.9  0.665  46     1
38     2   90   68   42    0  38.2  0.503  27     1
39     4  111   72   47  207  37.1  1.390  56     1
40     3  180   64   25   70  34.0  0.271  26     0
41     7  133   84    0    0  40.2  0.696  37     0
42     7  106   92   18    0  22.7  0.235  48     0
43     9  171  110   24  240  45.4  0.721  54     1
44     7  159   64    0    0  27.4  0.294  40     0
45     0  180   66   39    0  42.0  1.893  25     1
46     1  146   56    0    0  29.7  0.564  29     0
47     2   71   70   27    0  28.0  0.586  22     0
48     7  103   66   32    0  39.1  0.344  31     1
49     7  105    0    0    0   0.0  0.305  24     0
50     1  103   80   11   82  19.4  0.491  22     0
51     1  101   50   15   36  24.2  0.526  26     0
52     5   88   66   21   23  24.4  0.342  30     0
53     8  176   90   34  300  33.7  0.467  58     1
54     7  150   66   42  342  34.7  0.718  42     0
55     1   73   50   10    0  23.0  0.248  21     0
56     7  187   68   39  304  37.7  0.254  41     1
```

```
# Load CSV using Pandas from URL
import pandas
url = "/content/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(url, names=names)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   preg    768 non-null    int64
 1   plas    768 non-null    int64
 2   pres    768 non-null    int64
 3   skin    768 non-null    int64
 4   test    768 non-null    int64
 5   mass    768 non-null    float64
 6   pedi    768 non-null    float64
 7   age     768 non-null    int64
 8   class   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
print(data.head(20))
```

```
    preg  plas  pres  skin  test  mass   pedi  age  class
0      6   148    72    35     0  33.6  0.627   50      1
1      1    85    66    29     0  26.6  0.351   31      0
2      8   183    64     0     0  23.3  0.672   32      1
3      1    89    66    23    94  28.1  0.167   21      0
4      0   137    40    35   168  43.1  2.288   33      1
5      5   116    74     0     0  25.6  0.201   30      0
6      3    78    50    32    88  31.0  0.248   26      1
7     10   115     0     0     0  35.3  0.134   29      0
8      2   197    70    45   543  30.5  0.158   53      1
9      8   125    96     0     0   0.0  0.232   54      1
10     4   110    92     0     0  37.6  0.191   30      0
11    10   168    74     0     0  38.0  0.537   34      1
12    10   139    80     0     0  27.1  1.441   57      0
13     1   189    60    23   846  30.1  0.398   59      1
14     5   166    72    19   175  25.8  0.587   51      1
15     7   100     0     0     0  30.0  0.484   32      1
16     0   118    84    47   230  45.8  0.551   31      1
17     7   107    74     0     0  29.6  0.254   31      1
18     1   103    30    38    83  43.3  0.183   33      0
19     1   115    70    30    96  34.6  0.529   32      1
```

```
print(data.shape)
```

```
(768, 9)
```

Start coding or generate with AI.

Start coding or generate with AI.

```
print (data.head())
```

```
   preg  plas  pres  skin  test  mass   pedi  age  class
0     6   148    72    35     0  33.6  0.627   50      1
1     1    85    66    29     0  26.6  0.351   31      0
2     8   183    64     0     0  23.3  0.672   32      1
3     1    89    66    23    94  28.1  0.167   21      0
4     0   137    40    35   168  43.1  2.288   33      1
```

```
print(data.head(999))
```

```
     preg  plas  pres  skin  test  mass   pedi  age  class
0       6   148    72    35     0  33.6  0.627   50      1
1       1    85    66    29     0  26.6  0.351   31      0
2       8   183    64     0     0  23.3  0.672   32      1
3       1    89    66    23    94  28.1  0.167   21      0
4       0   137    40    35   168  43.1  2.288   33      1
..    ...   ...   ...   ...   ...   ...    ...  ...    ...
763    10   101    76    48   180  32.9  0.171   63      0
764     2   122    70    27     0  36.8  0.340   27      0
765     5   121    72    23   112  26.2  0.245   30      0
```

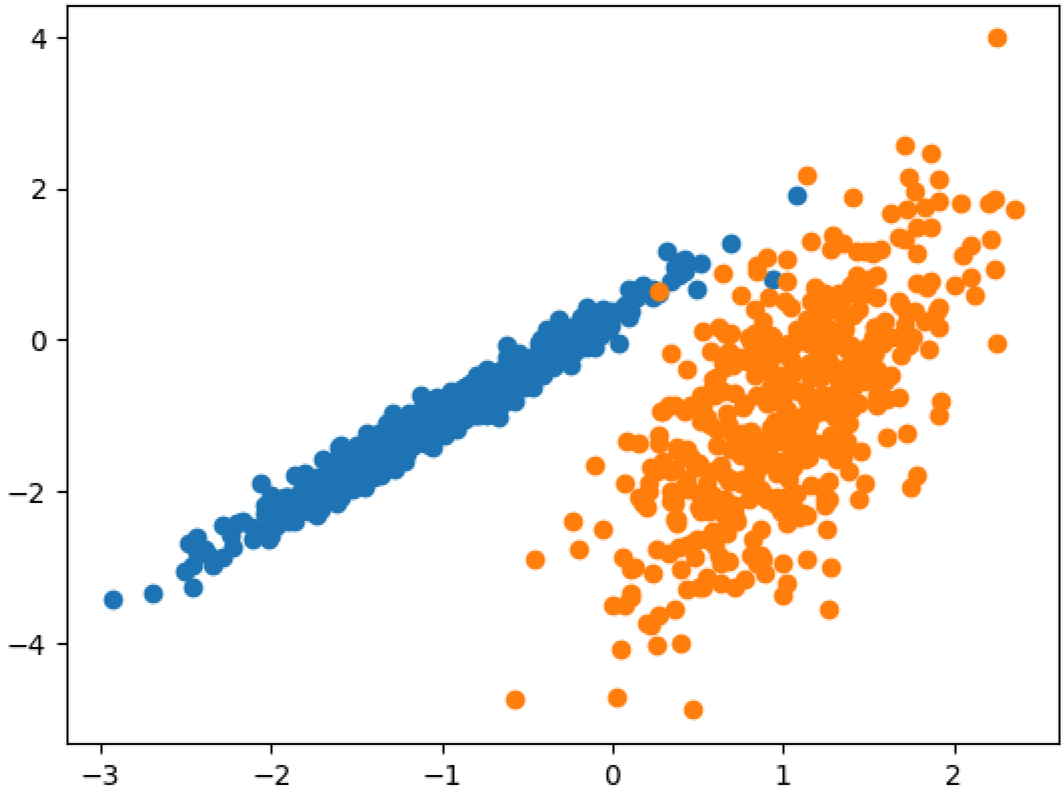
```
766     1   126    60     0     0  30.1  0.349   47      1
767     1    93    70    31     0  30.4  0.315   23      0

[768 rows x 9 columns]
```

```
data.describe()
```

	preg	plas	pres	skin	test	mass	pedi
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000

```
# synthetic classification dataset
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_c
# create scatter plot for samples from each class
for class_value in range(2):
    # get row indexes for samples with this class
    row_ix = where(y == class_value)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```




```
import numpy as np
np.random.seed(10)
# generating 10 random values for each of the two variables
X = np.random.randn(10)
Y = np.random.randn(10)
# computing the correlation matrix
C = np.corrcoef(X,Y)
print(C)
```

```
[[1.         0.37258014]
 [0.37258014 1.        ]]
```

```
# Exploring Iris Dataset
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy import stats
from pandas import read_csv
```

```
# define the location of the dataset
path = r"https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
```

```
# load the dataset and use df as a data frame
iris_df = read_csv(path, header=None)
# show the first few rows of the data
iris_df.head()
```

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
print(iris_df.head(20))
```

```
      0    1    2    3            4
0   5.1  3.5  1.4  0.2  Iris-setosa
1   4.9  3.0  1.4  0.2  Iris-setosa
2   4.7  3.2  1.3  0.2  Iris-setosa
3   4.6  3.1  1.5  0.2  Iris-setosa
4   5.0  3.6  1.4  0.2  Iris-setosa
5   5.4  3.9  1.7  0.4  Iris-setosa
6   4.6  3.4  1.4  0.3  Iris-setosa
7   5.0  3.4  1.5  0.2  Iris-setosa
8   4.4  2.9  1.4  0.2  Iris-setosa
9   4.9  3.1  1.5  0.1  Iris-setosa
10  5.4  3.7  1.5  0.2  Iris-setosa
11  4.8  3.4  1.6  0.2  Iris-setosa
12  4.8  3.0  1.4  0.1  Iris-setosa
13  4.3  3.0  1.1  0.1  Iris-setosa
14  5.8  4.0  1.2  0.2  Iris-setosa
15  5.7  4.4  1.5  0.4  Iris-setosa
16  5.4  3.9  1.3  0.4  Iris-setosa
17  5.1  3.5  1.4  0.3  Iris-setosa
18  5.7  3.8  1.7  0.3  Iris-setosa
19  5.1  3.8  1.5  0.3  Iris-setosa
```

```
print(iris_df.to_string())
```

```
       0    1    2    3               4
0    5.1  3.5  1.4  0.2     Iris-setosa
1    4.9  3.0  1.4  0.2     Iris-setosa
2    4.7  3.2  1.3  0.2     Iris-setosa
3    4.6  3.1  1.5  0.2     Iris-setosa
4    5.0  3.6  1.4  0.2     Iris-setosa
5    5.4  3.9  1.7  0.4     Iris-setosa
6    4.6  3.4  1.4  0.3     Iris-setosa
7    5.0  3.4  1.5  0.2     Iris-setosa
8    4.4  2.9  1.4  0.2     Iris-setosa
```

```
9    4.9  3.1  1.5  0.1      Iris-setosa
10   5.4  3.7  1.5  0.2      Iris-setosa
11   4.8  3.4  1.6  0.2      Iris-setosa
12   4.8  3.0  1.4  0.1      Iris-setosa
13   4.3  3.0  1.1  0.1      Iris-setosa
14   5.8  4.0  1.2  0.2      Iris-setosa
15   5.7  4.4  1.5  0.4      Iris-setosa
16   5.4  3.9  1.3  0.4      Iris-setosa
17   5.1  3.5  1.4  0.3      Iris-setosa
18   5.7  3.8  1.7  0.3      Iris-setosa
19   5.1  3.8  1.5  0.3      Iris-setosa
20   5.4  3.4  1.7  0.2      Iris-setosa
21   5.1  3.7  1.5  0.4      Iris-setosa
22   4.6  3.6  1.0  0.2      Iris-setosa
23   5.1  3.3  1.7  0.5      Iris-setosa
24   4.8  3.4  1.9  0.2      Iris-setosa
25   5.0  3.0  1.6  0.2      Iris-setosa
26   5.0  3.4  1.6  0.4      Iris-setosa
27   5.2  3.5  1.5  0.2      Iris-setosa
28   5.2  3.4  1.4  0.2      Iris-setosa
29   4.7  3.2  1.6  0.2      Iris-setosa
30   4.8  3.1  1.6  0.2      Iris-setosa
31   5.4  3.4  1.5  0.4      Iris-setosa
32   5.2  4.1  1.5  0.1      Iris-setosa
33   5.5  4.2  1.4  0.2      Iris-setosa
34   4.9  3.1  1.5  0.1      Iris-setosa
35   5.0  3.2  1.2  0.2      Iris-setosa
36   5.5  3.5  1.3  0.2      Iris-setosa
37   4.9  3.1  1.5  0.1      Iris-setosa
38   4.4  3.0  1.3  0.2      Iris-setosa
39   5.1  3.4  1.5  0.2      Iris-setosa
40   5.0  3.5  1.3  0.3      Iris-setosa
41   4.5  2.3  1.3  0.3      Iris-setosa
42   4.4  3.2  1.3  0.2      Iris-setosa
43   5.0  3.5  1.6  0.6      Iris-setosa
44   5.1  3.8  1.9  0.4      Iris-setosa
45   4.8  3.0  1.4  0.3      Iris-setosa
46   5.1  3.8  1.6  0.2      Iris-setosa
47   4.6  3.2  1.4  0.2      Iris-setosa
48   5.3  3.7  1.5  0.2      Iris-setosa
49   5.0  3.3  1.4  0.2      Iris-setosa
50   7.0  3.2  4.7  1.4  Iris-versicolor
51   6.4  3.2  4.5  1.5  Iris-versicolor
52   6.9  3.1  4.9  1.5  Iris-versicolor
53   5.5  2.3  4.0  1.3  Iris-versicolor
54   6.5  2.8  4.6  1.5  Iris-versicolor
55   5.7  2.8  4.5  1.3  Iris-versicolor
56   6.3  3.3  4.7  1.6  Iris-versicolor
```

```
# show the names of the columns or features
iris_df.columns
```

```
Index([0, 1, 2, 3, 4], dtype='int64')
```

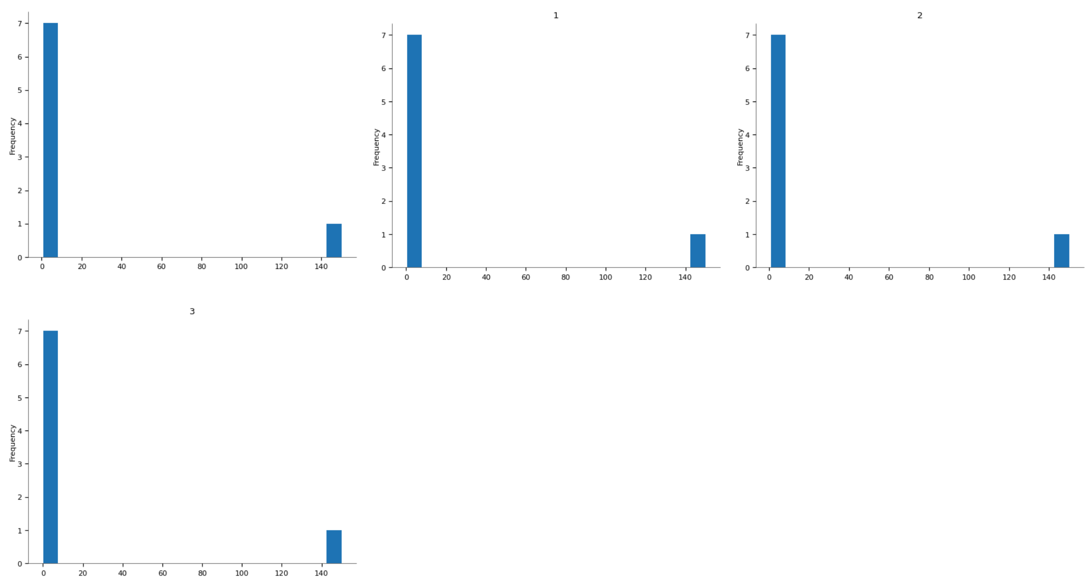
```
# show the number of rows and columns
iris_df.shape
```

```
(150, 5)
```

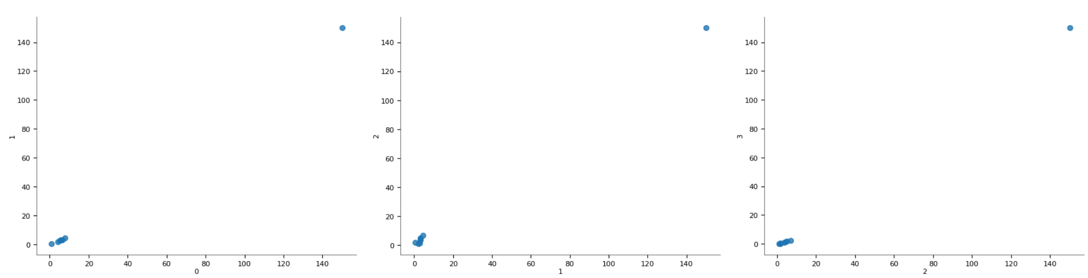
```
# get various summary stats of the data
iris_df.describe()
```

	0	1	2	3
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

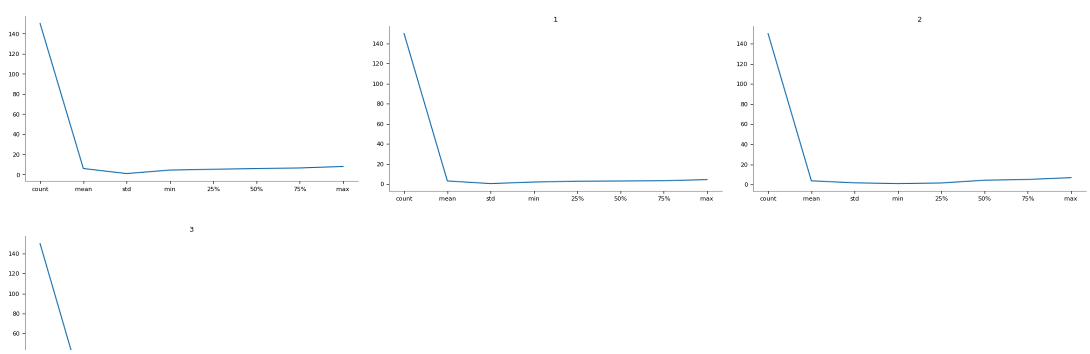
Distributions

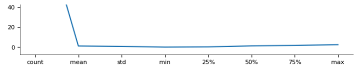


2-d distributions



Values






```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       150 non-null    float64
 1   1       150 non-null    float64
 2   2       150 non-null    float64
 3   3       150 non-null    float64
 4   4       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Start coding or generate with AI.

```
# Identify rows that contain missing values
iris_df.isnull()
```

	0	1	2	3	4
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
145	False	False	False	False	False
146	False	False	False	False	False
147	False	False	False	False	False
148	False	False	False	False	False
149	False	False	False	False	False

150 rows × 5 columns

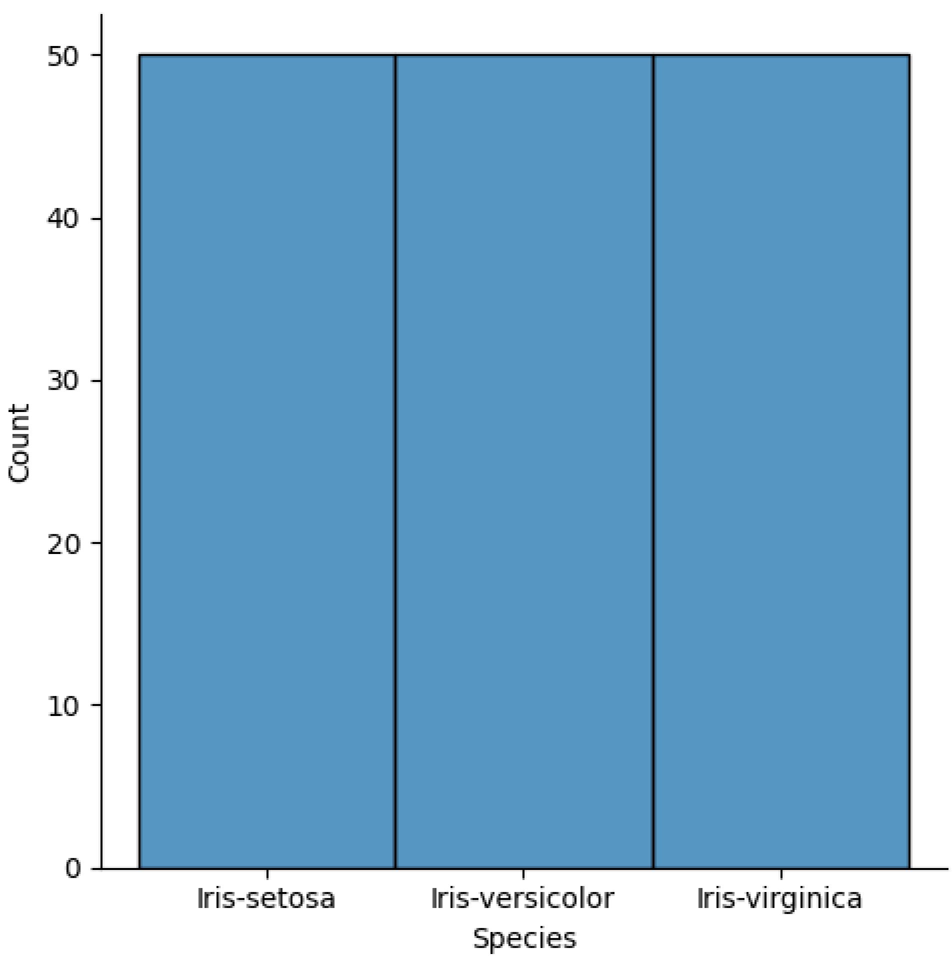
```
# Assign Names to the Features
names = ['Sepal length', 'Sepal width', 'Petal length', 'Petal width','Species']
```

```
iris_df = read_csv(path, names=names, header=None)
# Show the first few rows of the data again
iris_df.head()
```

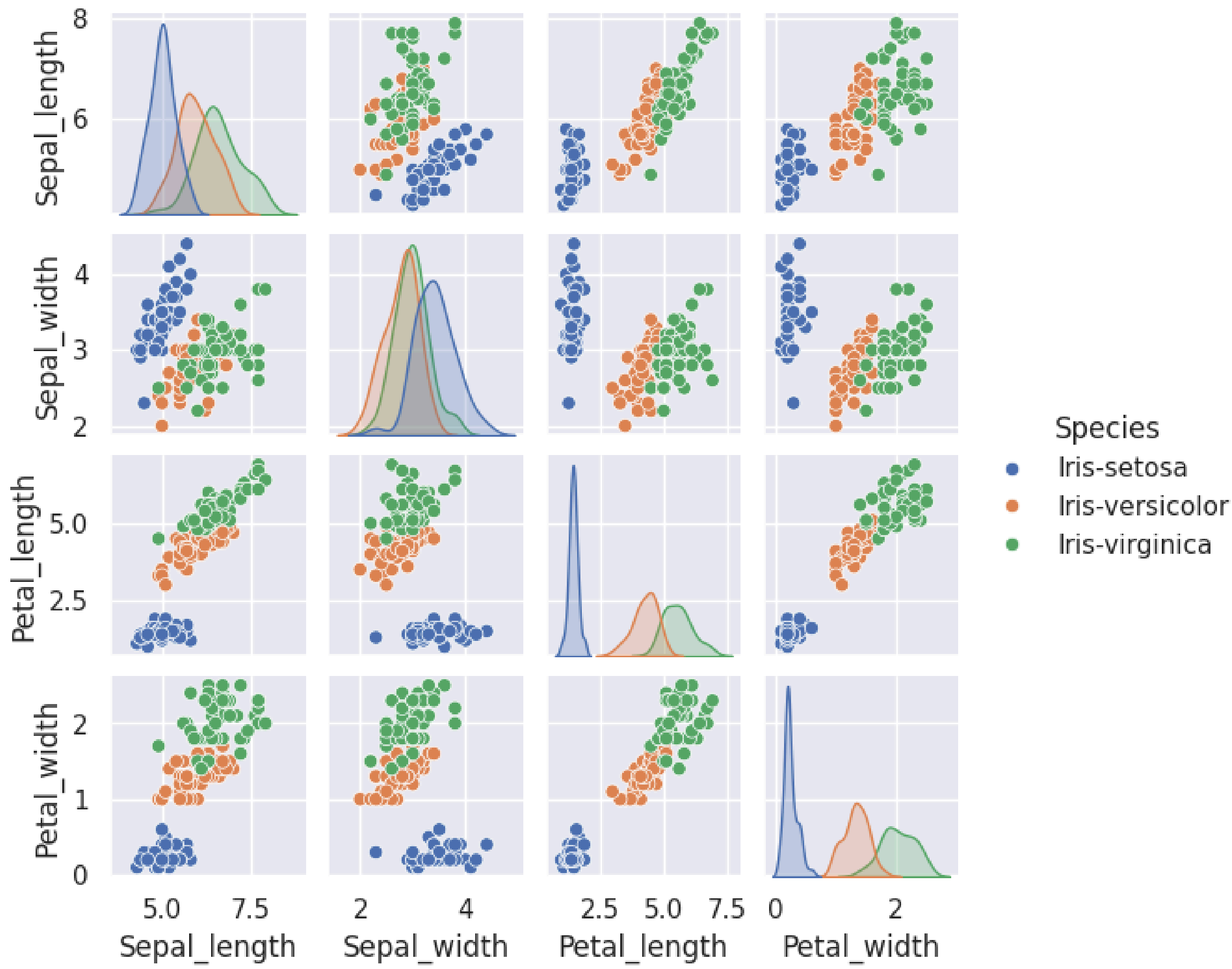
	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
#Species Distribution
sns.displot(iris_df['Species'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7f3e30076530>
```

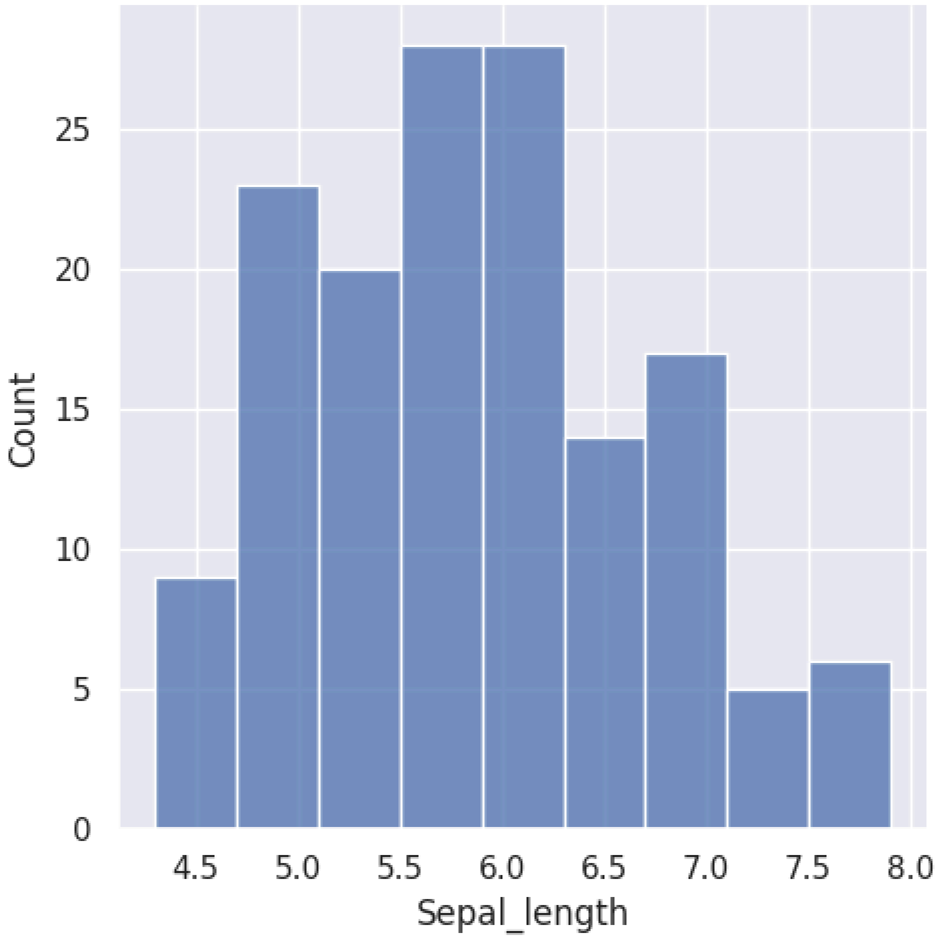


```
%matplotlib inline
import seaborn as sns; sns.set()
sns.pairplot(iris_df, hue='Species', height=1.5);
```




```
# histogram and Sepal_length Distribution
sns.displot(iris_df['Sepal_length'])
```

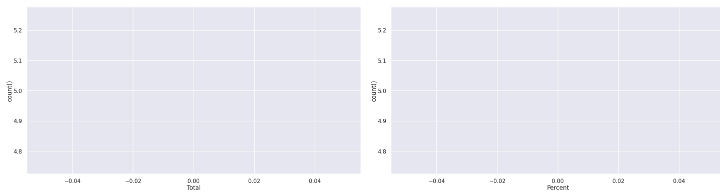
<seaborn.axisgrid.FacetGrid at 0x7f3e2f0c5d20>




```
# Missing Data
# If more than 15% of the data is missing then we might want to delete the feature (variable
total = iris_df.isnull().sum().sort_values(ascending=False)
percent = (iris_df.isnull().sum()/iris_df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

	Total	Percent
Sepal_length	0	0.0
Sepal_width	0	0.0
Petal_length	0	0.0
Petal_width	0	0.0
Species	0	0.0

Time series



```
# Dealing with Missing Data
# First, ensure 'missing_data' exists:
missing_data = iris_df.isnull().sum(axis=1)  # Example to calculate missing data row-wise if
# Drop rows with missing data count greater than 1
iris_df = iris_df.drop(missing_data[missing_data > 1].index)
# Check if there's any missing data left
missing_data_check = iris_df.isnull().sum().max()  # This will give the maximum number of mi
print(missing_data_check)  # Will return 0 if no missing data exists
```

```
0
```

```
# We can extract the feature matrix and target array from the iris data_frame
# This would be useful later on when we use Scikit-Learn to perform classification
X_iris = iris_df.drop('Species', axis=1)
X_iris.shape
```

```
(150, 4)
```

```
# We can extract the feature matrix and target array from the iris data_frame
# This would be useful later on when we use Scikit-Learn to perform classification
y_iris = iris_df['Species’]
y_iris.shape
```

```
  File "<ipython-input-56-e68ab886a21c>", line 3
    y_iris = iris_df['Species’]
                     ^
SyntaxError: unterminated string literal (detected at line 3)
```

```
from sklearn.model_selection import train_test_split

# Assuming 'target' is the name of your target variable column
X = iris_df.drop(columns=['Species'])
y = iris_df['Species']
print(f"x data head",X.head())
print(f"y data head",y.head())
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the sizes of the splits
print(f"Training set size: {len(X_train)}")
print(f"Test set size: {len(X_test)}")
```

```
x data head    Sepal_length  Sepal_width  Petal_length  Petal_width
0            5.1          3.5           1.4          0.2
1            4.9          3.0           1.4          0.2
2            4.7          3.2           1.3          0.2
3            4.6          3.1           1.5          0.2
4            5.0          3.6           1.4          0.2
y data head 0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: Species, dtype: object
Training set size: 120
Test set size: 30
```

```
# Check the first few rows of training and testing sets
print("Training data (features):")
print(X_train.head())

print("\nTraining data (target):")
print(y_train.head())

print("\nTesting data (features):")
print(X_test.head())

print("\nTesting data (target):")
print(y_test.head())
```

```
Training data (features):
    Sepal_length  Sepal_width  Petal_length  Petal_width
```

```
22           4.6          3.6           1.0          0.2
15           5.7          4.4           1.5          0.4
65           6.7          3.1           4.4          1.4
11           4.8          3.4           1.6          0.2
42           4.4          3.2           1.3          0.2

Training data (target):
22        Iris-setosa
15        Iris-setosa
65    Iris-versicolor
11        Iris-setosa
42        Iris-setosa
Name: Species, dtype: object

Testing data (features):
     Sepal_length  Sepal_width  Petal_length  Petal_width
73            6.1          2.8           4.7          1.2
18            5.7          3.8           1.7          0.3
118           7.7          2.6           6.9          2.3
78            6.0          2.9           4.5          1.5
76            6.8          2.8           4.8          1.4

Testing data (target):
73     Iris-versicolor
18         Iris-setosa
118     Iris-virginica
78     Iris-versicolor
76     Iris-versicolor
Name: Species, dtype: object
```

```
# Save the training features and target to CSV files
X_train.to_csv('X_train.csv', index=False)
y_train.to_csv('y_train.csv', index=False)

# Save the testing features and target to CSV files
X_test.to_csv('X_test.csv', index=False)
y_test.to_csv('y_test.csv', index=False)
import os
os.listdir('/content/')

from google.colab import files

# Download files
files.download('/content/X_train.csv')
files.download('/content/y_train.csv')
files.download('/content/X_test.csv')
files.download('/content/y_test.csv')
```

```
# Save the training features and target to CSV files
X_train.to_csv('X_train.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
```

```
# Save the testing features and target to CSV files
X_test.to_csv('X_test.csv', index=False)
y_test.to_csv('y_test.csv', index=False)
import os
os.listdir('/content/')

from google.colab import files

# Download files
files.download('/content/X_train.csv')
files.download('/content/y_train.csv')
files.download('/content/X_test.csv')
files.download('/content/y_test.csv')
```

Class No-20(New Teacher)

```
import pandas
filename = '/content/Tennis_dataset.csv'
print(filename)
```

```
/content/Tennis_dataset.csv
```

```
import pandas as pd

df = pd.read_csv('/content/Tennis_dataset.csv')

print(df.to_string())
```

```
     Outlook Temperature Humidity  Windy Play Tennis
0      Sunny         Hot     High  False          No
1      Sunny         Hot     High   True          No
2   Overcast         Hot     High  False         Yes
3       Rain        Mild     High  False         Yes
4       Rain        Cool   Normal  False         Yes
5       Rain        Cool   Normal   True          No
6   Overcast        Cool   Normal   True         Yes
7      Sunny        Mild     High  False          No
8      Sunny        Cool   Normal  False         Yes
9       Rain        Mild   Normal  False         Yes
10     Sunny        Mild   Normal   True         Yes
11  Overcast        Mild     High   True         Yes
12  Overcast         Hot   Normal  False         Yes
13      Rain        Mild     High   True          No
```

Start coding or generate with AI.

```
import pandas as pd
df = pd.read_csv('/content/Tennis_dataset.csv')
```

```
df.head()
```

	Outlook	Temperature	Humidity	Windy	Play Tennis
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rain	Mild	High	False	Yes
4	Rain	Cool	Normal	False	Yes

```
df.to_string()
```

```
'      Outlook Temperature Humidity  Windy Play Tennis\n0      Sunny         Hot     Hig
h  False          No\n1      Sunny         Hot     High   True          No\n2   Overcas
t        Hot     High  False         Yes\n3       Rain        Mild     High  False
Yes\n4       Rain        Cool   Normal  False         Yes\n5       Rain        Cool   N
ormal   True          No\n6   Overcast        Cool   Normal   True         Yes\n7
Sunny        Mild     High  False          No\n8      Sunny        Cool   Normal  False
Yes\n9       Rain        Mild   Normal  False         Yes\n10     Sunny        Mild   N
```

```
import pandas as pd
df = pd.read_csv('/content/Iris.csv')
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Start coding or generate with AI.

Project Python

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

```
import matplotlib.pyplot as plt
```

Load your dataset

```
import pandas as pd

# Try specifying a different delimiter if it's not a comma
data = pd.read_csv('/content/Daily Avg. Humidity.csv', sep=';')  # Try with semicolon
```

```
print(data.head())
```

```
                                  Station :Khulna
0                              Daily & Monthly Avera...
1                              ---------------------...
2  Year,Month,Dt(01,02,03,04,05,06,07,08,09,10,11...
3  1993, 1, 82, 86, 82, 84, 81, 79, 83, 85, 86, 8...
4  1993, 2, 76, 76, 72, 68, 66, 74, 70, 75, 80, 7...
```

Select features (assuming 'Temperature' is the target and others are features)

```
# Print the actual column names to check for discrepancies
print(data.columns)

# Modify the 'features' list to match the actual column names
features = ['Year', 'Month', 'avg']  # Example: Corrected column names
# If your column names have spaces, try enclosing them in backticks: `Daily Avg`
# Adjust based on the output of data.columns

# Select the desired columns
data = data[features]
```

```
Index(['                                         Station :Khulna       '], dtype='object')
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-29-513968a9f5b9> in <cell line: 10>()
      8
      9 # Select the desired columns
---> 10 data = data[features]

2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
_raise_if_missing(self, key, indexer, axis_name)
   6247         if nmissing:
   6248             if nmissing == len(indexer):
-> 6249                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6250
   6251             not_found = list(ensure_index(key)[missing_mask.nonzero()
[0]].unique())

KeyError: "None of [Index(['Year', 'Month', 'avg'], dtype='object')] are in the
[columns]"
```

Start coding or generate with AI.

Temperature Prediction with Python and Machine Learning FOR Dhaka City Corporation.

```
import pandas as pd

weather = pd.read_csv("/content/weather.csv", index_col="DATE")


weather
```

DATE	STATION	NAME	PRCP	TAVG	TMAX	TMIN
1990-01-01	BGM00041923	TEJGAON, BG	0.00	63	74.0	53.0
1990-01-03	BGM00041923	TEJGAON, BG	0.00	61	75.0	52.0
1990-01-04	BGM00041923	TEJGAON, BG	NaN	64	NaN	53.0
1990-01-06	BGM00041923	TEJGAON, BG	0.00	63	74.0	53.0
1990-01-07	BGM00041923	TEJGAON, BG	0.00	64	77.0	55.0
...	...	...	...	...	...	...
2024-10-21	BGM00041923	TEJGAON, BG	0.00	83	NaN	76.0
2024-10-22	BGM00041923	TEJGAON, BG	0.00	86	NaN	77.0
2024-10-23	BGM00041923	TEJGAON, BG	0.10	83	NaN	NaN
2024-10-24	BGM00041923	TEJGAON, BG	0.61	76	82.0	NaN
2024-10-25	BGM00041923	TEJGAON, BG	0.01	83	90.0	72.0

8403 rows × 6 columns

```
null_pct = weather.apply(pd.isnull).sum()/weather.shape[0]
null_pct
```

	0
STATION	0.000000
NAME	0.000000
PRCP	0.114007
TAVG	0.000000
TMAX	0.124360
TMIN	0.669166

dtype: float64

```
weather.apply(pd.isnull).sum()
```

	0
STATION	0
NAME	0
PRCP	958
TAVG	0
TMAX	1045
TMIN	5623

dtype: int64

```
valid_columns = weather.columns[null_pct < .05]
```

```
valid_columns
```

```
Index(['STATION', 'NAME', 'TAVG'], dtype='object')
```

```
weather = weather[valid_columns].copy()
```

```
weather.columns = weather.columns.str.lower()
```

```
weather
```

DATE	station	name	tavg
1990-01-01	BGM00041923	TEJGAON, BG	63
1990-01-03	BGM00041923	TEJGAON, BG	61
1990-01-04	BGM00041923	TEJGAON, BG	64
1990-01-06	BGM00041923	TEJGAON, BG	63
1990-01-07	BGM00041923	TEJGAON, BG	64
...	...	...	...
2024-10-21	BGM00041923	TEJGAON, BG	83
2024-10-22	BGM00041923	TEJGAON, BG	86
2024-10-23	BGM00041923	TEJGAON, BG	83
2024-10-24	BGM00041923	TEJGAON, BG	76
2024-10-25	BGM00041923	TEJGAON, BG	83

8403 rows × 3 columns

```
weather['tavg'] = weather['tavg'].fillna(weather['tavg'].mean())
# Display the modified dataset
print(weather)
print(weather.info())
```

```
                station         name  tavg
DATE
1990-01-01  BGM00041923  TEJGAON, BG    63
1990-01-03  BGM00041923  TEJGAON, BG    61
1990-01-04  BGM00041923  TEJGAON, BG    64
1990-01-06  BGM00041923  TEJGAON, BG    63
1990-01-07  BGM00041923  TEJGAON, BG    64
...                 ...          ...   ...
2024-10-21  BGM00041923  TEJGAON, BG    83
2024-10-22  BGM00041923  TEJGAON, BG    86
2024-10-23  BGM00041923  TEJGAON, BG    83
2024-10-24  BGM00041923  TEJGAON, BG    76
2024-10-25  BGM00041923  TEJGAON, BG    83

[8403 rows x 3 columns]
<class 'pandas.core.frame.DataFrame'>
Index: 8403 entries, 1990-01-01 to 2024-10-25
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   station  8403 non-null   object
 1   name     8403 non-null   object
 2   tavg     8403 non-null   int64
```

```
dtypes: int64(1), object(2)
memory usage: 262.6+ KB
None
```

```
weather = weather.ffill()
```

```
weather.apply(pd.isnull).sum()
```

	0
station	0
name	0
tavg	0

dtype: int64

```
weather.apply(lambda x: (x == 9999).sum())
```

	0
station	0
name	0
tavg	0

dtype: int64

```
weather.dtypes
```

	0
station	object
name	object
tavg	int64

dtype: object

```
weather.index
```

```
Index(['1990-01-01', '1990-01-03', '1990-01-04', '1990-01-06', '1990-01-07',
       '1990-01-08', '1990-01-09', '1990-01-10', '1990-01-12', '1990-01-13',
       ...
       '2024-10-16', '2024-10-17', '2024-10-18', '2024-10-19', '2024-10-20',
       '2024-10-21', '2024-10-22', '2024-10-23', '2024-10-24', '2024-10-25'],
      dtype='object', name='DATE', length=8403)
```

```
weather.index = pd.to_datetime(weather.index)
```

```
weather.index.year.value_counts().sort_index()
```

	count
DATE	
1990	257
1991	290
1992	343
1993	321
1994	250
1995	289
1996	284
1997	166
1998	172
1999	132
2000	209
2001	278
2002	225
2003	201
2004	193
2005	230
2006	291
2007	214
2008	170
2009	290
2010	269
2011	99
2012	148
2013	185
2014	318
2015	329
2016	312
2017	296
2018	311

2019	221
2020	320
2021	25
2022	131
2023	355
2024	279

dtype: int64

Start coding or generate with AI.

Filling with Mean, Median, or Mode

```
weather['tavg'] = weather['tavg'].fillna(weather['tavg'].mean())
# Display the modified dataset
print(weather)
print(weather.info())
```

```
                station         name  tavg
DATE
1990-01-01  BGM00041923  TEJGAON, BG    63
1990-01-03  BGM00041923  TEJGAON, BG    61
1990-01-04  BGM00041923  TEJGAON, BG    64
1990-01-06  BGM00041923  TEJGAON, BG    63
1990-01-07  BGM00041923  TEJGAON, BG    64
...                 ...          ...   ...
2024-10-21  BGM00041923  TEJGAON, BG    83
2024-10-22  BGM00041923  TEJGAON, BG    86
2024-10-23  BGM00041923  TEJGAON, BG    83
2024-10-24  BGM00041923  TEJGAON, BG    76
2024-10-25  BGM00041923  TEJGAON, BG    83

[8403 rows x 3 columns]
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8403 entries, 1990-01-01 to 2024-10-25
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   station  8403 non-null   object
 1   name     8403 non-null   object
 2   tavg     8403 non-null   int64
dtypes: int64(1), object(2)
memory usage: 262.6+ KB
None
```

```
weather.index.year.value_counts().sort_index()
```

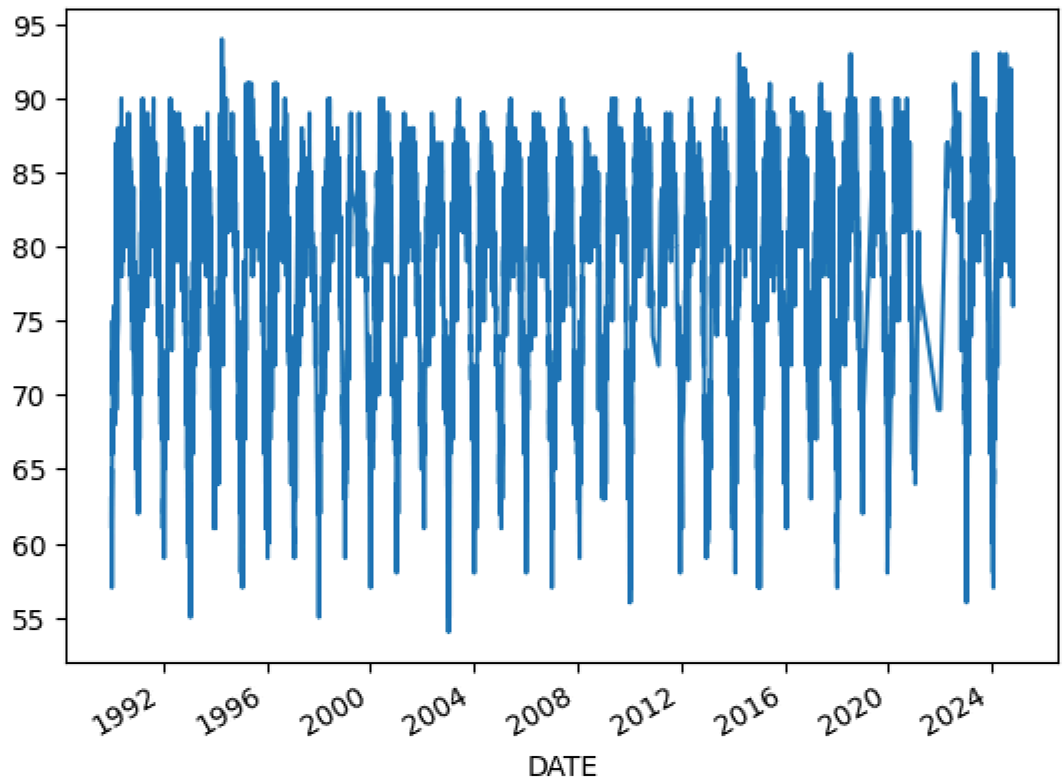
	count
DATE	
1990	257
1991	290
1992	343
1993	321
1994	250
1995	289
1996	284
1997	166
1998	172
1999	132
2000	209
2001	278
2002	225
2003	201
2004	193
2005	230
2006	291
2007	214
2008	170
2009	290
2010	269
2011	99
2012	148
2013	185
2014	318
2015	329
2016	312
2017	296
2018	311

2019	221
2020	320
2021	25
2022	131
2023	355
2024	279

dtype: int64

```
weather["tavg"].plot()
```

```
<Axes: xlabel='DATE'>
```



Start coding or generate with AI.

Tomorrow Weather Pattern

```
weather["target"] = weather.shift(-1)["tavg"]
```

```
weather
```

	station	name	tavg	target
DATE				
1990-01-01	BGM00041923	TEJGAON, BG	63	61.0
1990-01-03	BGM00041923	TEJGAON, BG	61	64.0
1990-01-04	BGM00041923	TEJGAON, BG	64	63.0
1990-01-06	BGM00041923	TEJGAON, BG	63	64.0
1990-01-07	BGM00041923	TEJGAON, BG	64	65.0
...	...	...	...	...
2024-10-21	BGM00041923	TEJGAON, BG	83	86.0
2024-10-22	BGM00041923	TEJGAON, BG	86	83.0
2024-10-23	BGM00041923	TEJGAON, BG	83	76.0
2024-10-24	BGM00041923	TEJGAON, BG	76	83.0
2024-10-25	BGM00041923	TEJGAON, BG	83	NaN

8403 rows × 4 columns

```
weather = weather.ffill()
```

```
weather
```

	station	name	tavg	target
DATE				
1990-01-01	BGM00041923	TEJGAON, BG	63	61.0
1990-01-03	BGM00041923	TEJGAON, BG	61	64.0
1990-01-04	BGM00041923	TEJGAON, BG	64	63.0
1990-01-06	BGM00041923	TEJGAON, BG	63	64.0
1990-01-07	BGM00041923	TEJGAON, BG	64	65.0
...	...	...	...	...
2024-10-21	BGM00041923	TEJGAON, BG	83	86.0
2024-10-22	BGM00041923	TEJGAON, BG	86	83.0
2024-10-23	BGM00041923	TEJGAON, BG	83	76.0
2024-10-24	BGM00041923	TEJGAON, BG	76	83.0
2024-10-25	BGM00041923	TEJGAON, BG	83	83.0

8403 rows × 4 columns

```
from sklearn.linear_model import Ridge

rr = Ridge(alpha=.1)



predictors = weather.columns[~weather.columns.isin(["target", "name", "station"])]


predictors
```

```
Index(['tavg'], dtype='object')
```

```
def backtest(weather, model, predictors, start=3650, step=90):
    all_predictions = []

    for i in range(start, weather.shape[0], step):
        train = weather.iloc[:i,:]
        test = weather.iloc[i:(i+step),:]

        model.fit(train[predictors], train["target"])

        preds = model.predict(test[predictors])
        preds = pd.Series(preds, index=test.index)
        combined = pd.concat([test["target"], preds], axis=1)
        combined.columns = ["actual", "prediction"]
```

```
        combined["diff"] = (combined["prediction"] - combined["actual"]).abs()

        all_predictions.append(combined)
    return pd.concat(all_predictions)
```

```
predictions = backtest(weather, rr, predictors)
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

mean_absolute_error(predictions["actual"], predictions["prediction"])
```

```
2.065274446896808
```

```
predictions.sort_values("diff", ascending=False)
```

DATE	actual	prediction	diff
2014-02-03	79.0	59.625368	19.374632
2022-01-01	87.0	69.814585	17.185415
2019-01-13	85.0	69.813581	15.186419
2014-02-02	58.0	71.610908	13.610908
2014-12-25	57.0	69.771097	12.771097
...	...	...	...
2007-12-26	67.0	67.003209	0.003209
2015-11-05	79.0	78.997211	0.002789
2017-07-25	79.0	78.997617	0.002383
2017-04-19	79.0	78.997617	0.002383
2018-05-18	79.0	78.999381	0.000619

4753 rows × 3 columns

```
pd.Series(rr.coef_, index=predictors)
```

	0
tavg	0.922487

dtype: float64

```
def pct_diff(old, new):
    return (new - old) / old

def compute_rolling(weather, horizon, col):
    label = f"rolling_{horizon}_{col}"
    weather[label] = weather[col].rolling(horizon).mean()
    weather[f"{label}_pct"] = pct_diff(weather[label], weather[col])
    return weather

rolling_horizons = [3, 14]
for horizon in rolling_horizons:
    for col in ["tavg"]:
        weather = compute_rolling(weather, horizon, col)


def expand_mean(df):
    return df.expanding(1).mean()

for col in ["tavg"]:
    weather[f"month_avg_{col}"] = weather[col].groupby(weather.index.month, group_keys=False
    weather[f"day_avg_{col}"] = weather[col].groupby(weather.index.day_of_year, group_keys=F


weather
```

	station	name	tavg	target	rolling_3_tavg	rolling_3_tavg_pct	rollin
DATE							
1990-01-01	BGM00041923	TEJGAON, BG	63	61.0	NaN	NaN	
1990-01-03	BGM00041923	TEJGAON, BG	61	64.0	NaN	NaN	
1990-01-04	BGM00041923	TEJGAON, BG	64	63.0	62.666667	0.021277	
1990-01-06	BGM00041923	TEJGAON, BG	63	64.0	62.666667	0.005319	
1990-01-07	BGM00041923	TEJGAON, BG	64	65.0	63.666667	0.005236	
...	...	...	...	...	...	...	
2024-10-21	BGM00041923	TEJGAON, BG	83	86.0	83.000000	0.000000	
2024-10-22	BGM00041923	TEJGAON, BG	86	83.0	84.000000	0.023810	
2024-10-23	BGM00041923	TEJGAON, BG	83	76.0	84.000000	-0.011905	
2024-10-24	BGM00041923	TEJGAON, BG	76	83.0	81.666667	-0.069388	
2024-10-25	BGM00041923	TEJGAON, BG	83	83.0	80.666667	0.028926	

8403 rows × 10 columns

```
weather = weather.iloc[14:,:]
weather = weather.fillna(0)
```

```
predictors = weather.columns[~weather.columns.isin(["target", "name", "station"])]
```

```
predictors
```

```
Index(['tavg', 'rolling_3_tavg', 'rolling_3_tavg_pct', 'rolling_14_tavg',
       'rolling_14_tavg_pct', 'month_avg_tavg', 'day_avg_tavg'],
      dtype='object')
```

```
predictions = backtest(weather, rr, predictors)
mean_absolute_error(predictions["actual"], predictions["prediction"])
```

```
1.9673166596363283
```

```
predictors.sort_values("diff", ascending=False)
```

```
<ipython-input-41-5bad40bdd310>:1: FutureWarning: Starting with pandas version 3.0 all a
  predictors.sort_values("diff", ascending=False)
(Index(['tavg', 'rolling_3_tavg_pct', 'rolling_3_tavg', 'rolling_14_tavg_pct',
        'rolling_14_tavg', 'month_avg_tavg', 'day_avg_tavg'],
       dtype='object'),
 array([0, 2, 1, 4, 3, 5, 6]))
```

```
weather.loc["1990-03-07": "1990-03-17"]
```

DATE	station	name	tavg	target	rolling_3_tavg	rolling_3_tavg_pct	rollin
1990-03-07	BGM00041923	TEJGAON, BG	71	72.0	73.000000	-0.027397	
1990-03-08	BGM00041923	TEJGAON, BG	72	81.0	73.000000	-0.013699	
1990-03-11	BGM00041923	TEJGAON, BG	81	69.0	74.666667	0.084821	
1990-03-12	BGM00041923	TEJGAON, BG	69	70.0	74.000000	-0.067568	
1990-03-13	BGM00041923	TEJGAON, BG	70	79.0	73.333333	-0.045455	
1990-03-16	BGM00041923	TEJGAON, BG	79	87.0	72.666667	0.087156	
1990-03-17	BGM00041923	TEJGAON, BG	87	83.0	78.666667	0.105932	

```
predictions["diff"].round().value_counts().sort_index() / predictions.shape[0]
```

diff	count
0.0	0.166702
1.0	0.303018
2.0	0.239291
3.0	0.144334
4.0	0.074066
5.0	0.035451
6.0	0.018569
7.0	0.008019
8.0	0.004220
9.0	0.002532
10.0	0.001688
11.0	0.000422
12.0	0.000633
13.0	0.000422
16.0	0.000211
17.0	0.000211
19.0	0.000211

dtype: float64

```
mean_squared_error(predictions["actual"], predictions["prediction"])
```

```
6.638035649016272
```

```
predictions.sort_values("diff", ascending=False)
```

DATE	actual	prediction	diff
2022-01-01	87.0	68.479289	18.520711
2019-01-13	85.0	67.595650	17.404350
2014-02-03	79.0	62.828462	16.171538
2014-02-02	58.0	70.868049	12.868049
2021-01-18	78.0	65.223965	12.776035
...	...	...	...
2024-08-07	85.0	85.002833	0.002833
2023-01-12	63.0	63.002649	0.002649
2007-12-07	69.0	69.001455	0.001455
2009-06-05	85.0	85.001127	0.001127
2023-11-23	75.0	75.000713	0.000713

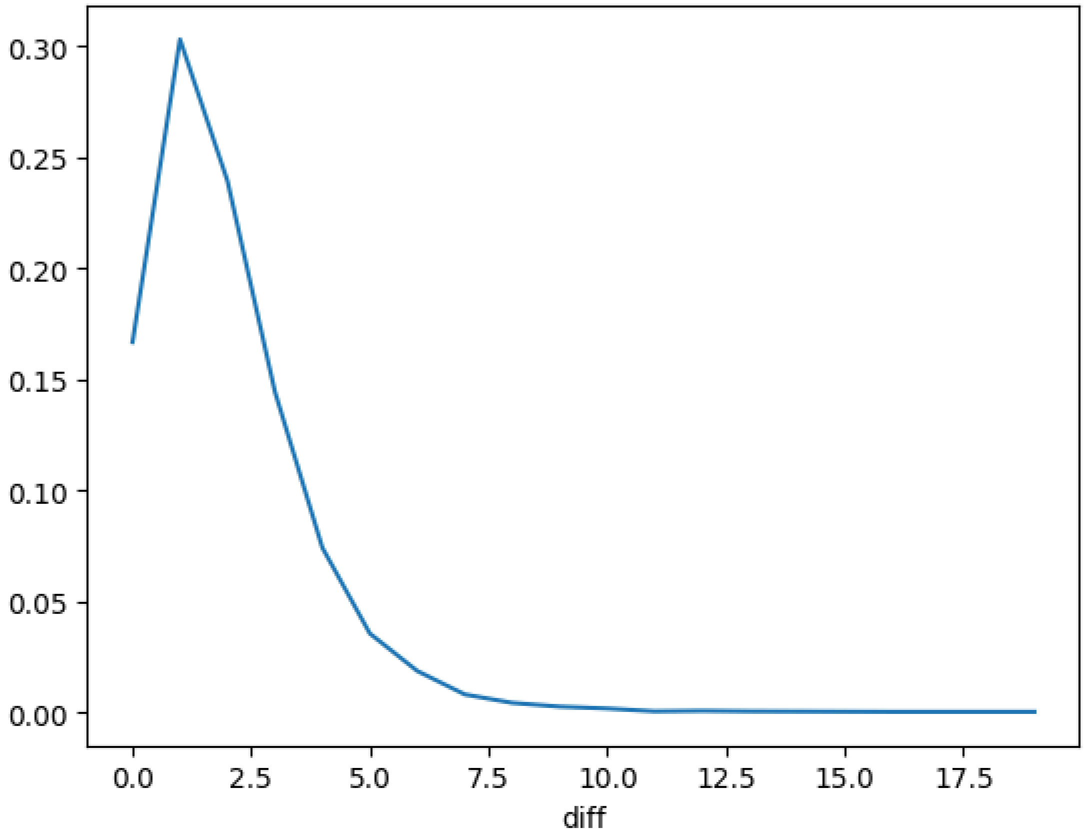
4739 rows × 3 columns

```
weather.loc["1990-03-07": "1990-03-17"]
```

DATE	station	name	tavg	target	rolling_3_tavg	rolling_3_tavg_pct	rollin
1990-03-07	BGM00041923	TEJGAON, BG	71	72.0	73.000000	-0.027397	
1990-03-08	BGM00041923	TEJGAON, BG	72	81.0	73.000000	-0.013699	
1990-03-11	BGM00041923	TEJGAON, BG	81	69.0	74.666667	0.084821	
1990-03-12	BGM00041923	TEJGAON, BG	69	70.0	74.000000	-0.067568	
1990-03-13	BGM00041923	TEJGAON, BG	70	79.0	73.333333	-0.045455	
1990-03-16	BGM00041923	TEJGAON, BG	79	87.0	72.666667	0.087156	
1990-03-17	BGM00041923	TEJGAON, BG	87	83.0	78.666667	0.105932	

```
(predictions["diff"].round().value_counts().sort_index() / predictions.shape[0]).plot()
```

<Axes: xlabel='diff'>




```
predictions
```