Introduction to Pandas

Outline

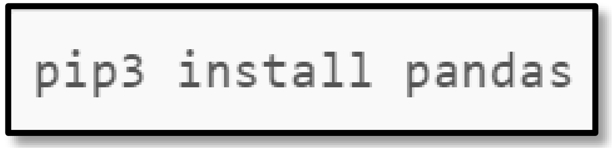
Pandas is a powerful and versatile open-source data analysis and manipulation library for Python. It provides data structures and functions that make it easy to work with structured data, enabling users to perform complex data transformations and analyses efficiently.

Pandas is widely used in data science, machine learning, and statistical analysis due to its ease of use, flexibility, and the ability to handle large datasets efficiently. Whether-we're analyzing financial data, conducting scientific research, or working on machine learning projects, Pandas is an essential tool in the Python ecosystem.

In this lecture, we will explore the different features of Pandas library, including data structures and data manipulation.

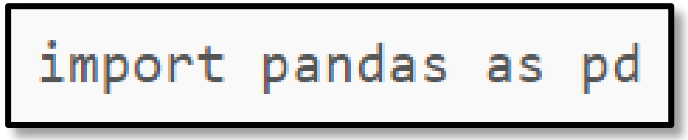
Getting started

Using the Pandas library requires the version of Python to be 3.5 and above. We can install this library using the following command:



```
pip3 install pandas
```

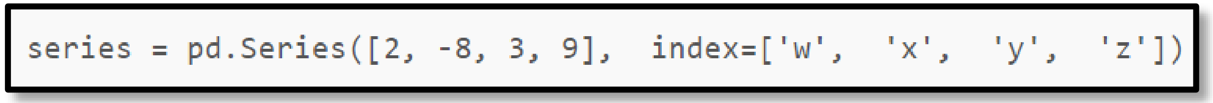
After installing the library, we must import Pandas, in order to use it in our program.



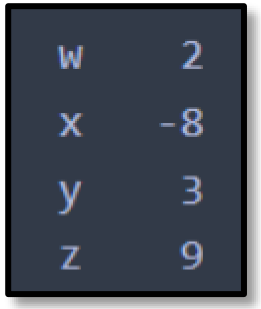
```
import pandas as pd
```

Series and Dataframes

A **Series** in Pandas is similar to any other series we come across. A series is a one-dimensional labeled array that can hold any data type.

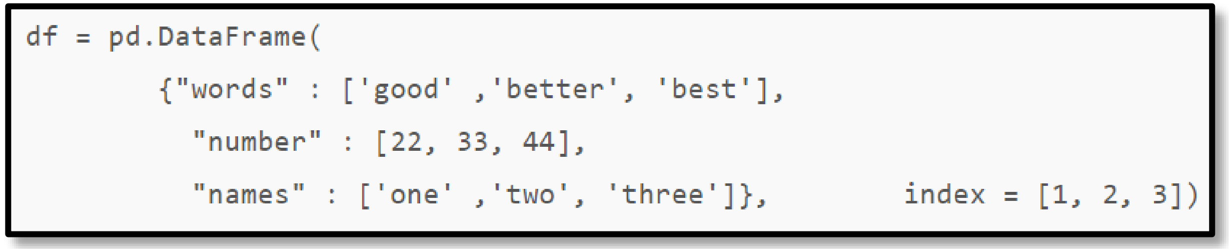


```
series = pd.Series([2, -8, 3, 9],  index=['w',  'x',  'y',  'z'])
```

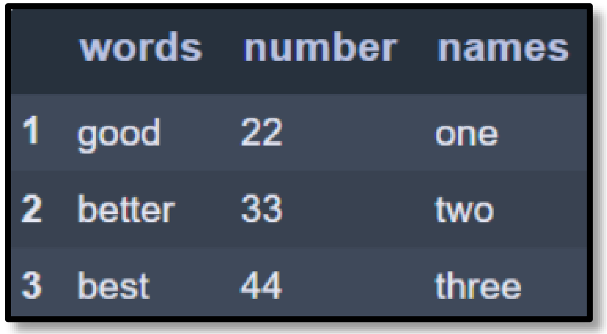


```
w    2
x   -8
y    3
z    9
```

A **Dataframe** in Pandas library is a two-dimensional labeled data structure where each row represents an observation.



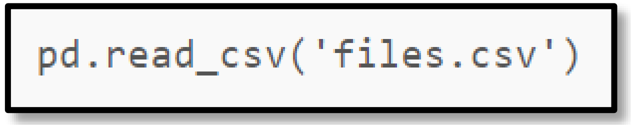
```
df = pd.DataFrame(
        {"words" : ['good' ,'better', 'best'],
          "number" : [22, 33, 44],
          "names" : ['one' ,'two', 'three']},        index = [1, 2, 3])
```



	words	number	names
1	good	22	one
2	better	33	two
3	best	44	three

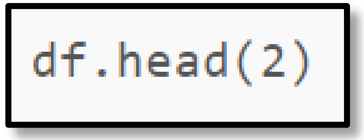
Working with Dataframes

In order to read a CSV file in Pandas, we use the ***pd.read_csv()*** function.

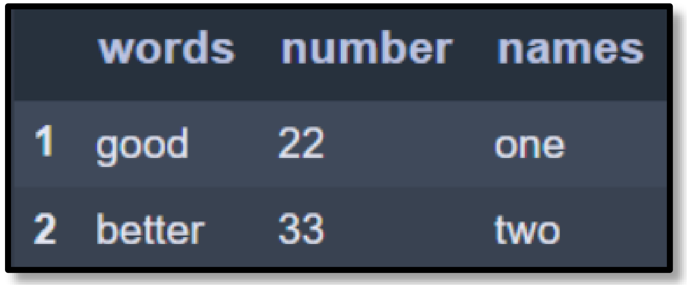


```
pd.read_csv('files.csv')
```

We can take a look at the first two lines of a dataframe.

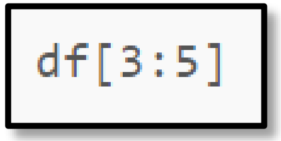


```
df.head(2)
```



	words	number	names
1	good	22	one
2	better	33	two

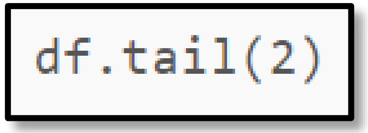
We can also take a look at the rows between certain indices.



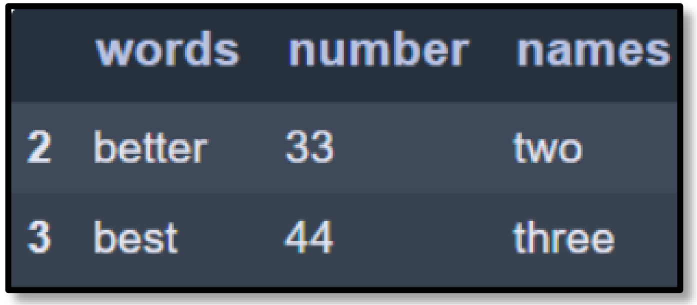
```
df[3:5]
```

Working with Dataframes

We can take a look at the last two lines in a dataframe using the ***df.tail()*** function.

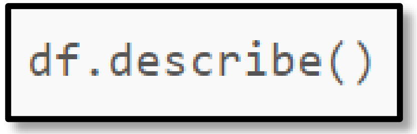


```
df.tail(2)
```

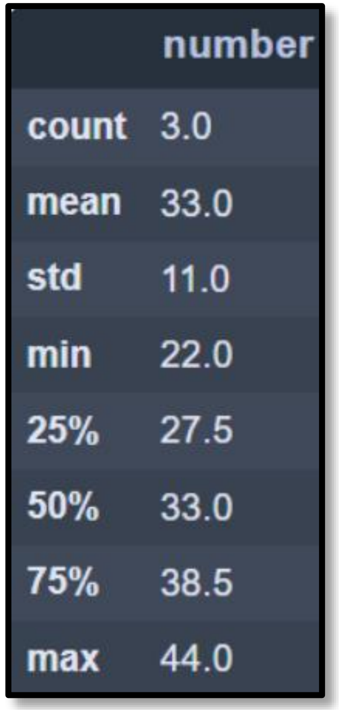


	words	number	names
2	better	33	two
3	best	44	three

We can see the analysis of numerical columns in a dataframe using the ***df.describe()*** function.



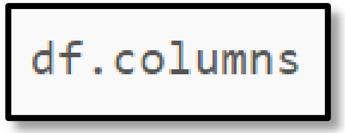
```
df.describe()
```



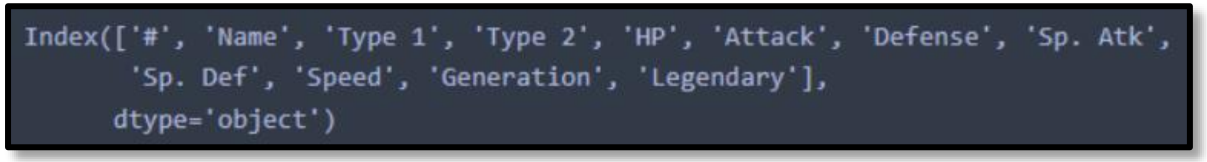
	number
count	3.0
mean	33.0
std	11.0
min	22.0
25%	27.5
50%	33.0
75%	38.5
max	44.0

Working with Dataframes

We can take a look at only the columns in a dataframe.

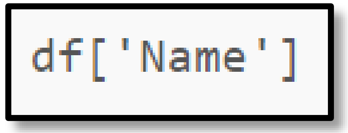


```
df.columns
```

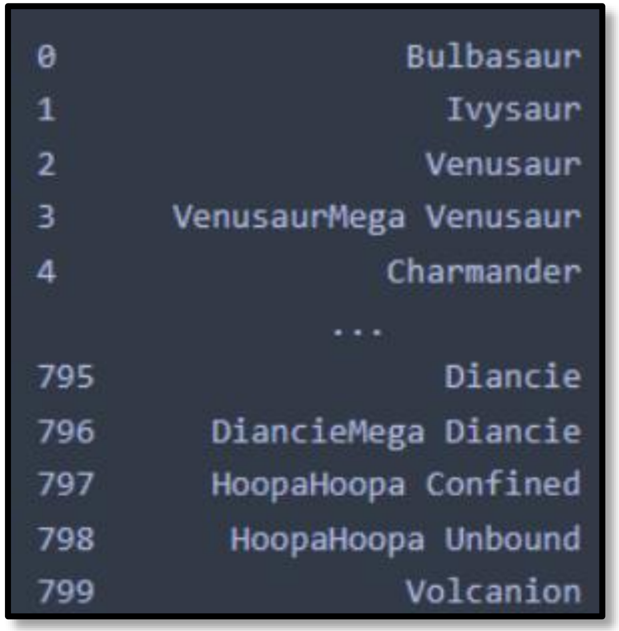


```
Index(['#', 'Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary'],
      dtype='object')
```

We can also see specific columns in a dataframe. Note that instead of the *Name* keyword, we can use the other names of the columns of the CSV file.



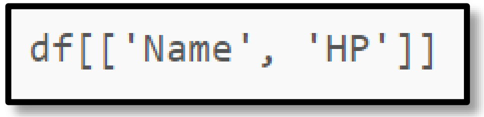
```
df['Name']
```



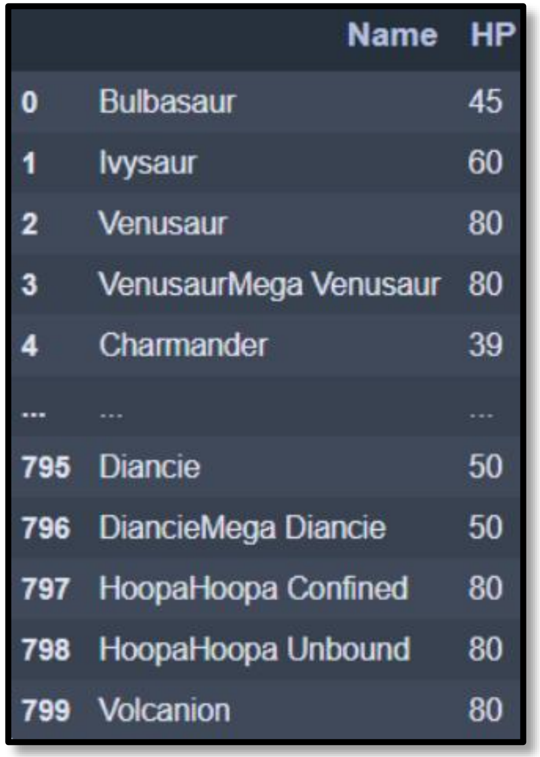
```
0                      Bulbasaur
1                        Ivysaur
2                       Venusaur
3          VenusaurMega Venusaur
4                     Charmander
                   ...
795                      Diancie
796          DiancieMega Diancie
797          HoopaHoopa Confined
798           HoopaHoopa Unbound
799                    Volcanion
```

Working with Dataframes

We can take a look at more than two columns at the same time.

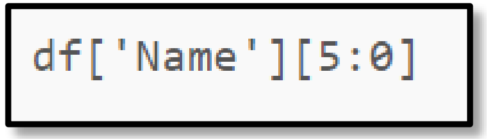


```
df[['Name', 'HP']]
```

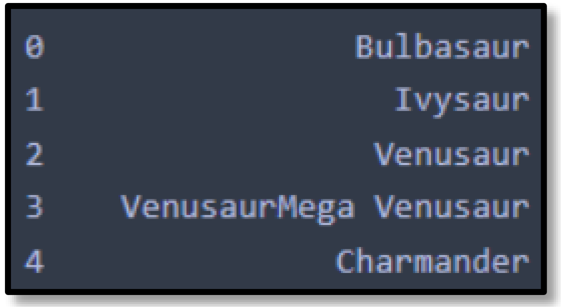


	Name	HP
0	Bulbasaur	45
1	Ivysaur	60
2	Venusaur	80
3	VenusaurMega Venusaur	80
4	Charmander	39
...	...	...
795	Diancie	50
796	DiancieMega Diancie	50
797	HoopaHoopa Confined	80
798	HoopaHoopa Unbound	80
799	Volcanion	80

We can also see a specific number of names. Note that we can put any other number in place of 5.



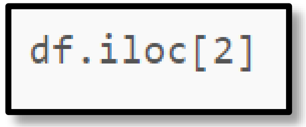
```
df['Name'][5:0]
```



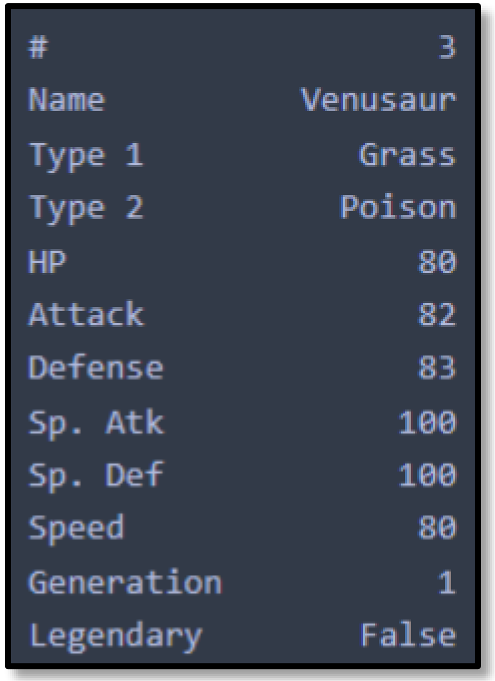
```
0                Bulbasaur
1                  Ivysaur
2                 Venusaur
3    VenusaurMega Venusaur
4               Charmander
```

Working with Dataframes

We can take a look at one specific row using the ***iloc*** keyword.

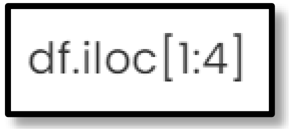


```
df.iloc[2]
```

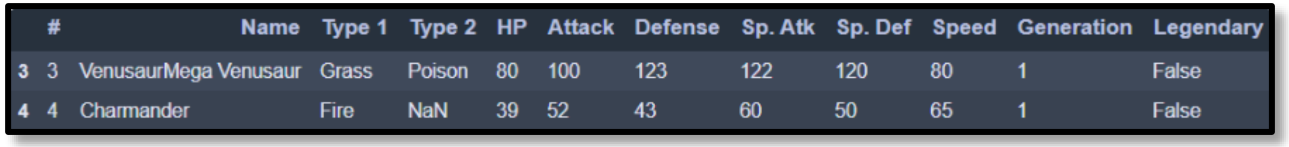


```
#                    3
Name          Venusaur
Type 1           Grass
Type 2          Poison
HP                  80
Attack              82
Defense             83
Sp. Atk            100
Sp. Def            100
Speed               80
Generation           1
Legendary        False
```

We can also see multiple rows of data at the same time.

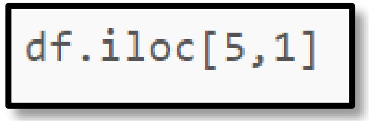


```
df.iloc[1:4]
```

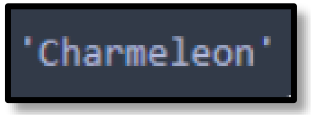


	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	False

It is also possible to see the value of a specific row and column.



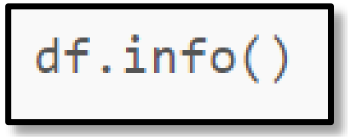
```
df.iloc[5,1]
```



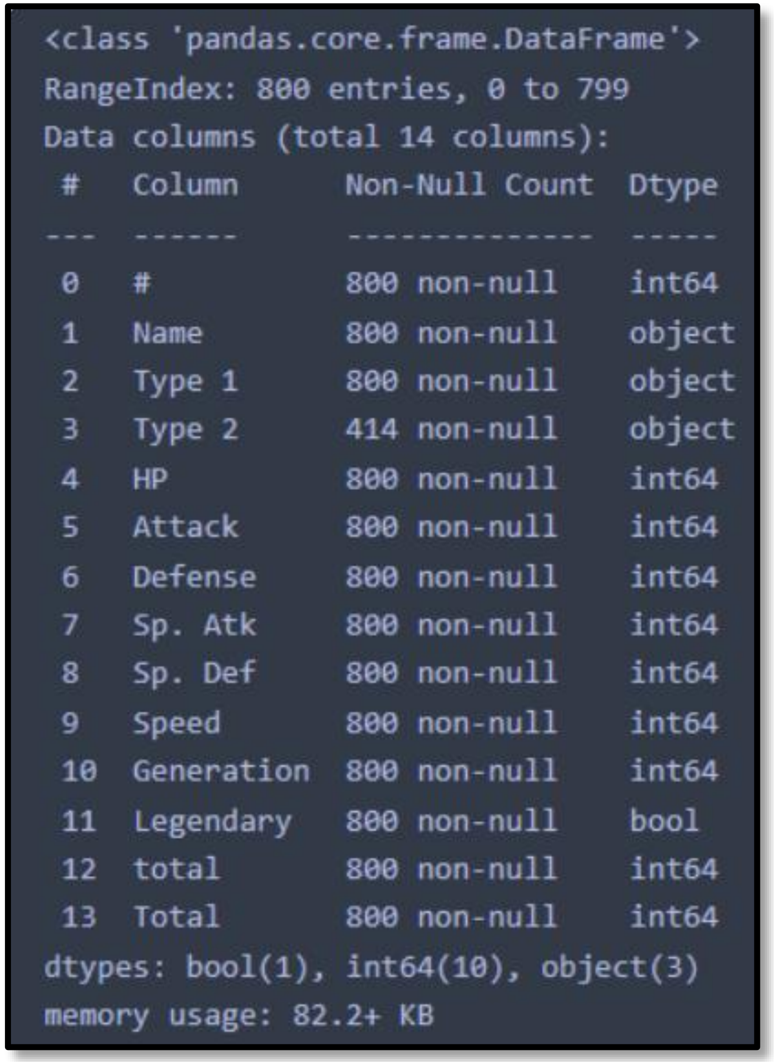
```
'Charmeleon'
```

Working with Dataframes

We can observe all the info of the dataframe using the ***info()*** function.



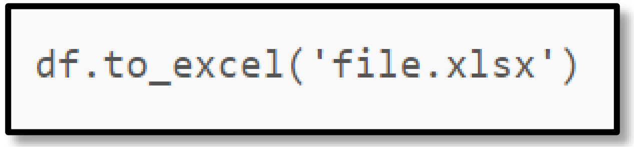
```
df.info()
```



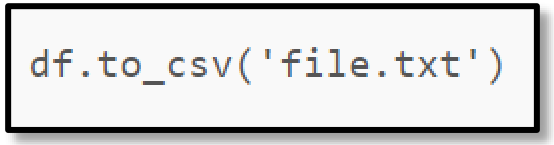
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   #           800 non-null    int64
 1   Name        800 non-null    object
 2   Type 1      800 non-null    object
 3   Type 2      414 non-null    object
 4   HP          800 non-null    int64
 5   Attack      800 non-null    int64
 6   Defense     800 non-null    int64
 7   Sp. Atk     800 non-null    int64
 8   Sp. Def     800 non-null    int64
 9   Speed       800 non-null    int64
 10  Generation  800 non-null    int64
 11  Legendary   800 non-null    bool
 12  total       800 non-null    int64
 13  Total       800 non-null    int64
dtypes: bool(1), int64(10), object(3)
memory usage: 82.2+ KB
```

Exporting Dataframes

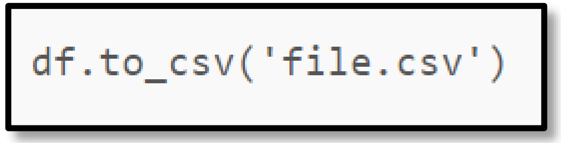
We can export a dataframe into an excel file using the ***to_excel()*** function. We can also export a dataframe into a text file using the ***to_csv()*** function. It is also possible to export a dataframe into a CSV file, using the ***to_csv()*** function.
It is important to note that when the script is executed, the new file will be located within the project’s root directory. If we do not want to include the index numbers, we can set the *index* keyword to False.



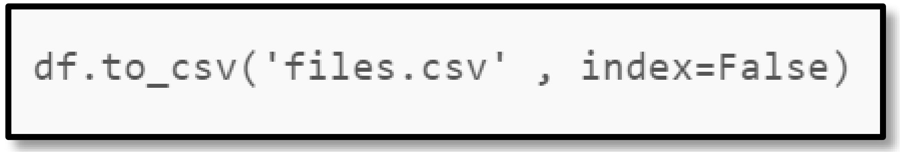
```
df.to_excel('file.xlsx')
```



```
df.to_csv('file.txt')
```



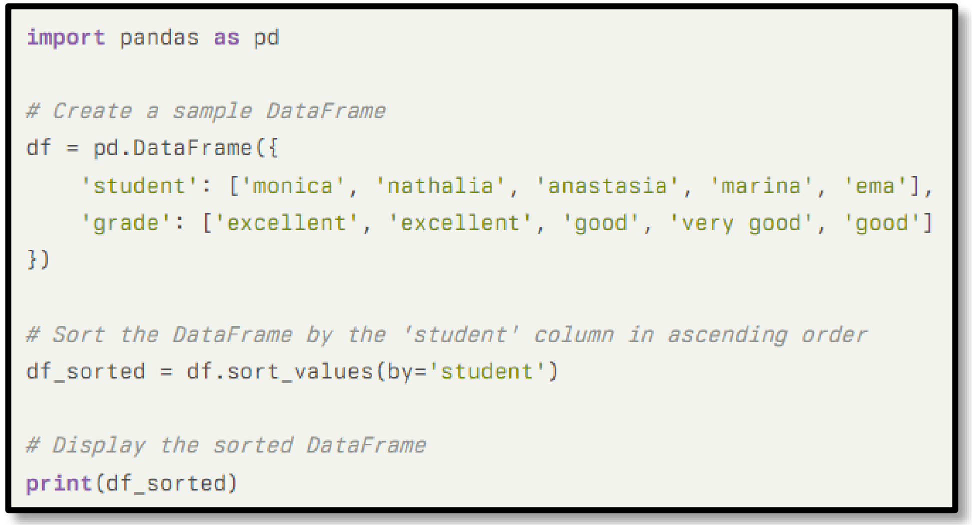
```
df.to_csv('file.csv')
```



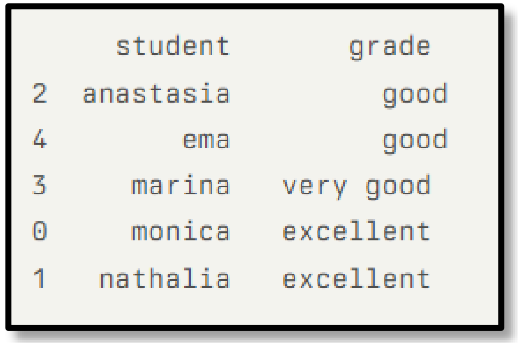
```
df.to_csv('files.csv' , index=False)
```

Sorting Dataframes

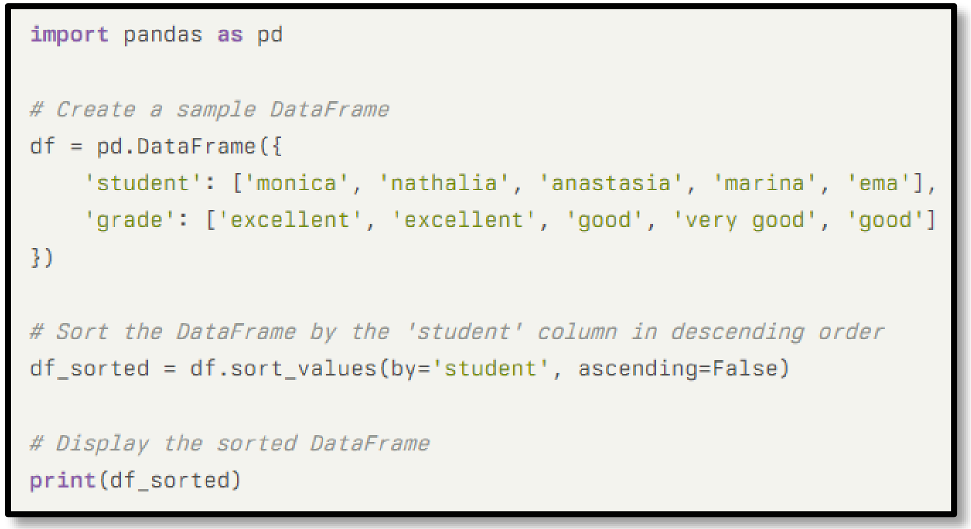
We can sort the data by alphabetical name in ascending or descending order, using the ***df.sort_values()*** function. The default order is ascending. However, If we want to sort in descending order, we set the *ascending* keyword to False.



```python
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema'],
    'grade': ['excellent', 'excellent', 'good', 'very good', 'good']
})

# Sort the DataFrame by the 'student' column in ascending order
df_sorted = df.sort_values(by='student')

# Display the sorted DataFrame
print(df_sorted)
```



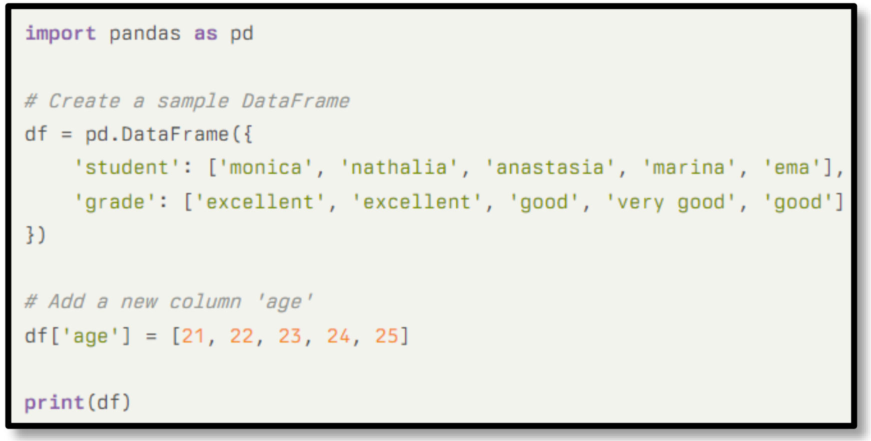
```
     student      grade
2  anastasia       good
4        ema       good
3     marina  very good
0     monica  excellent
1   nathalia  excellent
```



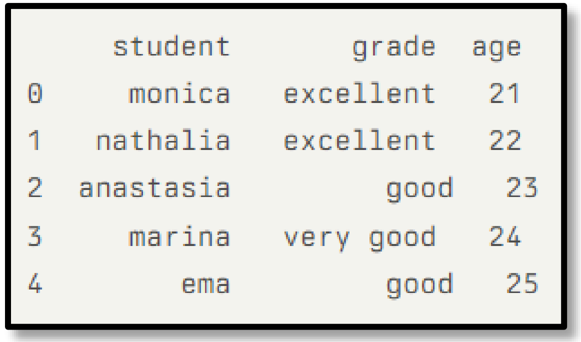
```python
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema'],
    'grade': ['excellent', 'excellent', 'good', 'very good', 'good']
})

# Sort the DataFrame by the 'student' column in descending order
df_sorted = df.sort_values(by='student', ascending=False)

# Display the sorted DataFrame
print(df_sorted)
```

Applying changes to a dataframe

We can add a new column to the dataframe.

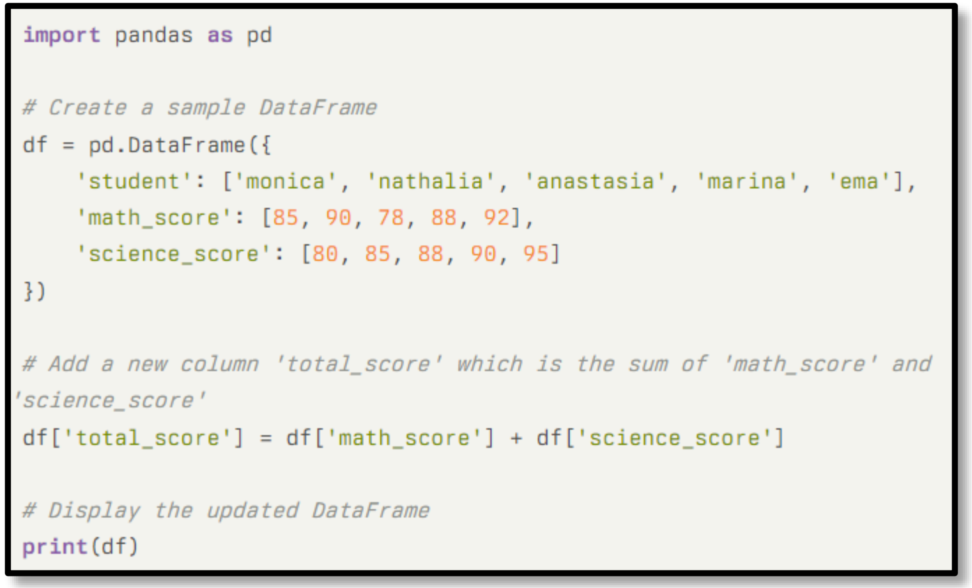


```
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema'],
    'grade': ['excellent', 'excellent', 'good', 'very good', 'good']
})

# Add a new column 'age'
df['age'] = [21, 22, 23, 24, 25]

print(df)
```

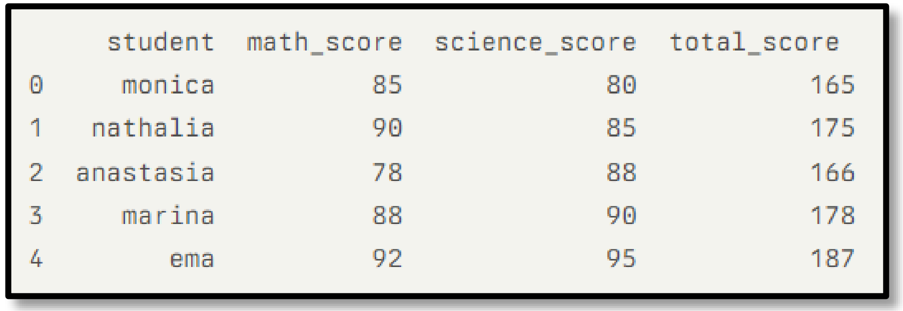


```
     student      grade  age
0     monica  excellent   21
1   nathalia  excellent   22
2  anastasia       good   23
3     marina  very good   24
4        ema       good   25
```

We can also add a new column to the dataframe which is a combination of other columns.



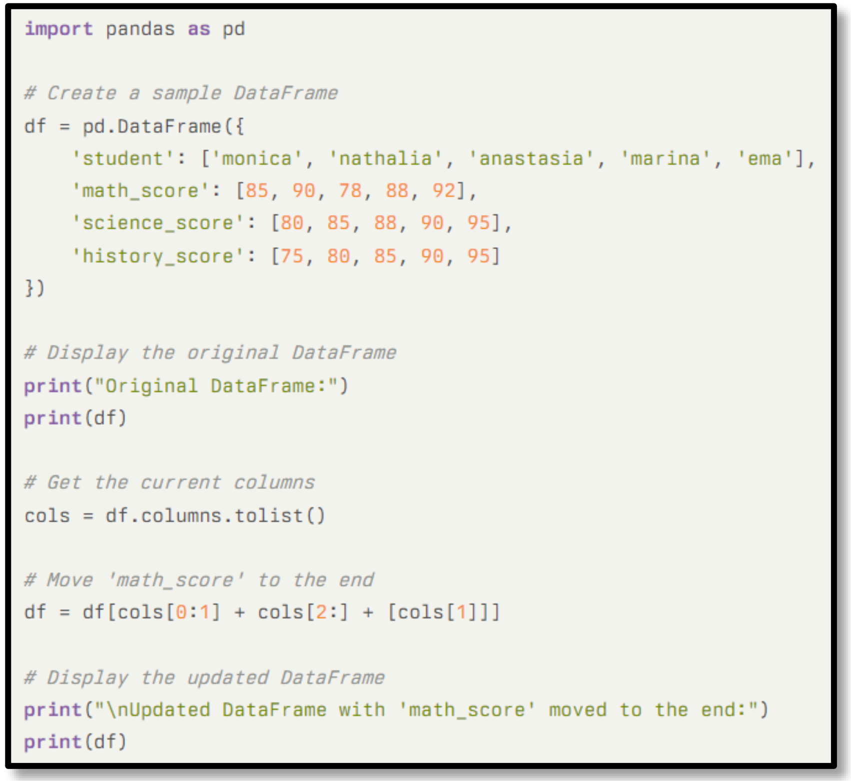
```
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema'],
    'math_score': [85, 90, 78, 88, 92],
    'science_score': [80, 85, 88, 90, 95]
})

# Add a new column 'total_score' which is the sum of 'math_score' and 'science_score'
df['total_score'] = df['math_score'] + df['science_score']

# Display the updated DataFrame
print(df)
```



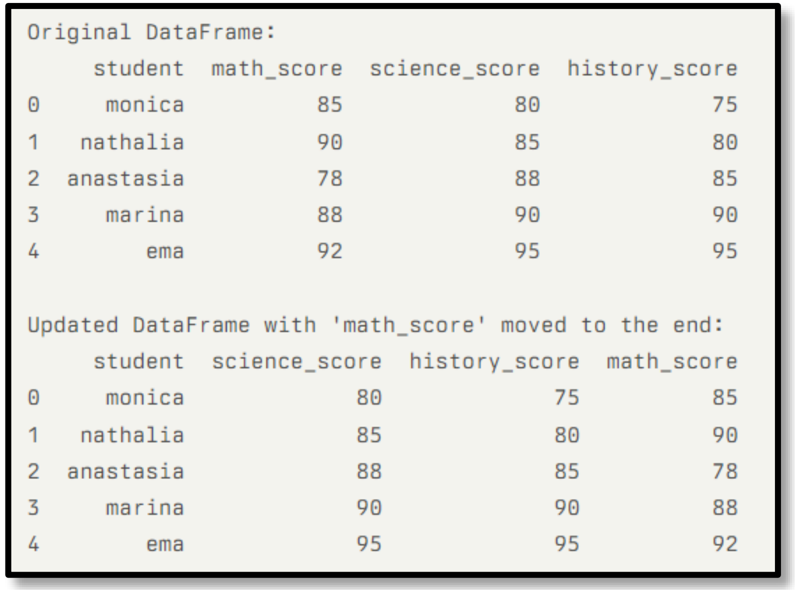
```
     student  math_score  science_score  total_score
0     monica          85             80          165
1   nathalia          90             85          175
2  anastasia          78             88          166
3     marina          88             90          178
4        ema          92             95          187
```

Applying changes to a dataframe

We can move the place of a column in a dataframe.

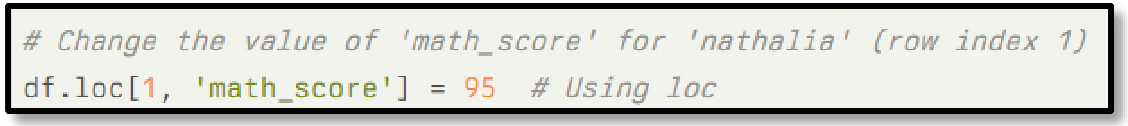


```
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema'],
    'math_score': [85, 90, 78, 88, 92],
    'science_score': [80, 85, 88, 90, 95],
    'history_score': [75, 80, 85, 90, 95]
})

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Get the current columns
cols = df.columns.tolist()

# Move 'math_score' to the end
df = df[cols[0:1] + cols[2:] + [cols[1]]]

# Display the updated DataFrame
print("\nUpdated DataFrame with 'math_score' moved to the end:")
print(df)
```



```
Original DataFrame:
     student  math_score  science_score  history_score
0     monica          85             80             75
1   nathalia          90             85             80
2  anastasia          78             88             85
3     marina          88             90             90
4        ema          92             95             95

Updated DataFrame with 'math_score' moved to the end:
     student  science_score  history_score  math_score
0     monica             80             75          85
1   nathalia             85             80          90
2  anastasia             88             85          78
3     marina             90             90          88
4        ema             95             95          92
```

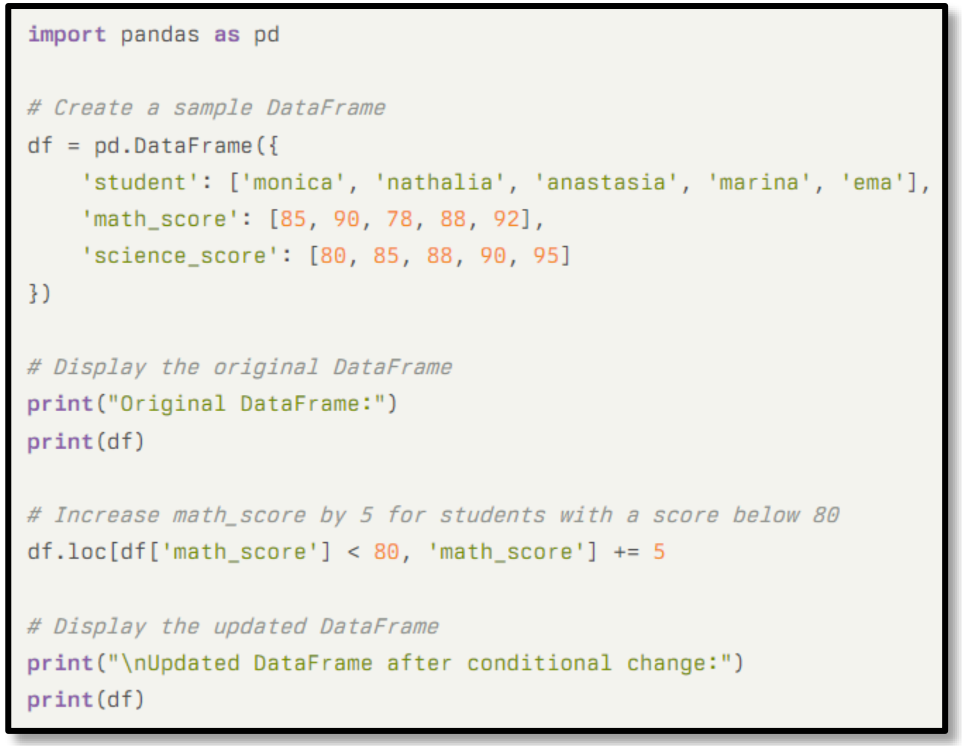
We can also change the value at a specified row and column.



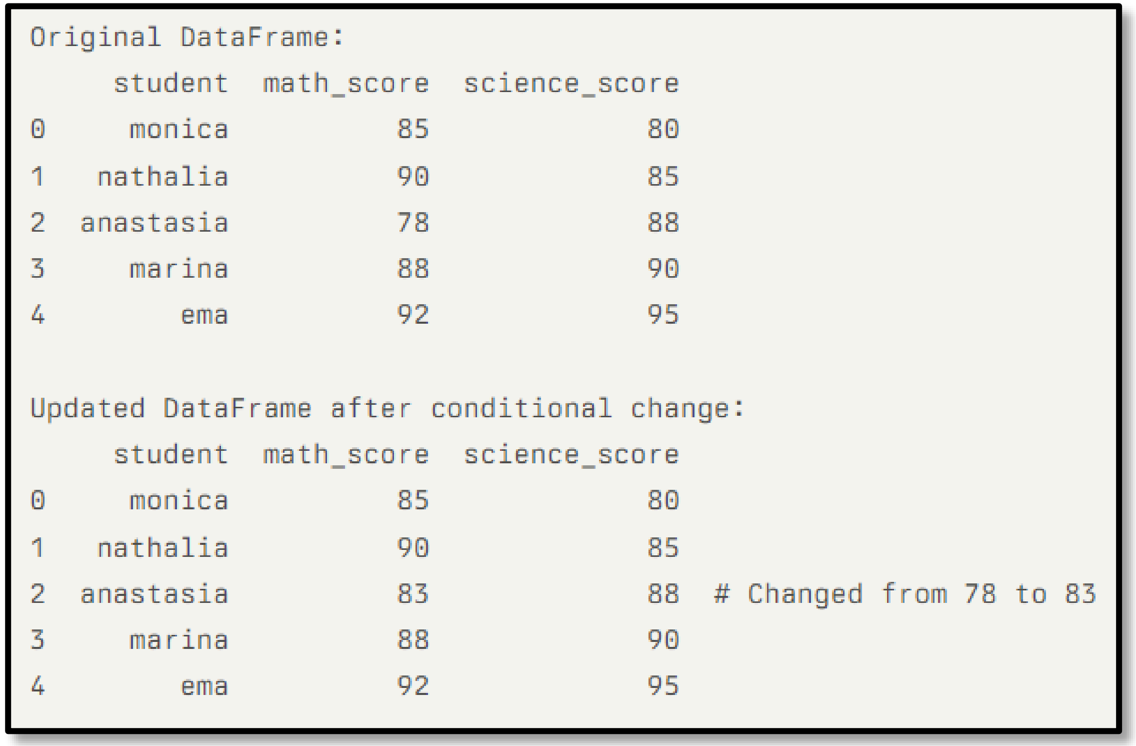
```
# Change the value of 'math_score' for 'nathalia' (row index 1)
df.loc[1, 'math_score'] = 95  # Using loc
```

Applying changes to a dataframe

We can apply conditional change to a dataframe.



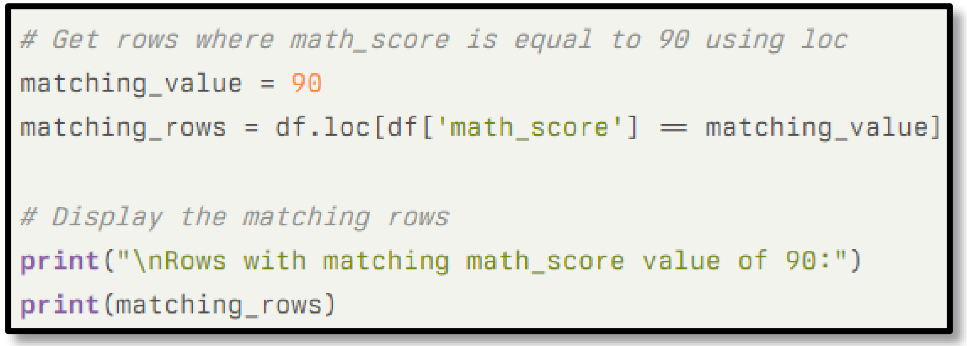
```python
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema'],
    'math_score': [85, 90, 78, 88, 92],
    'science_score': [80, 85, 88, 90, 95]
})

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Increase math_score by 5 for students with a score below 80
df.loc[df['math_score'] < 80, 'math_score'] += 5

# Display the updated DataFrame
print("\nUpdated DataFrame after conditional change:")
print(df)
```



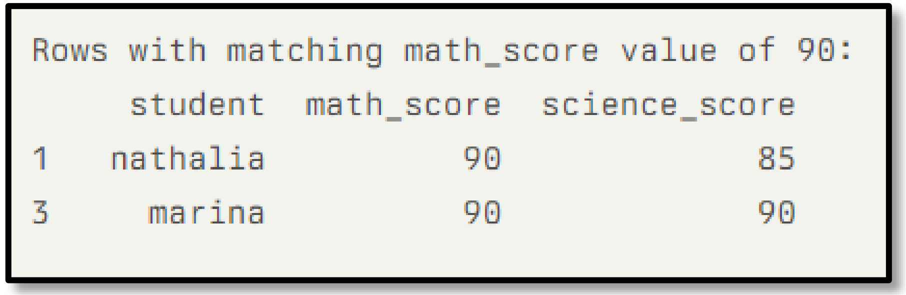
```
Original DataFrame:
     student  math_score  science_score
0     monica          85             80
1   nathalia          90             85
2  anastasia          78             88
3     marina          88             90
4        ema          92             95

Updated DataFrame after conditional change:
     student  math_score  science_score
0     monica          85             80
1   nathalia          90             85
2  anastasia          83             88  # Changed from 78 to 83
3     marina          88             90
4        ema          92             95
```

Filtering data in a dataframe

We can get the rows with a matching value at a specified column.

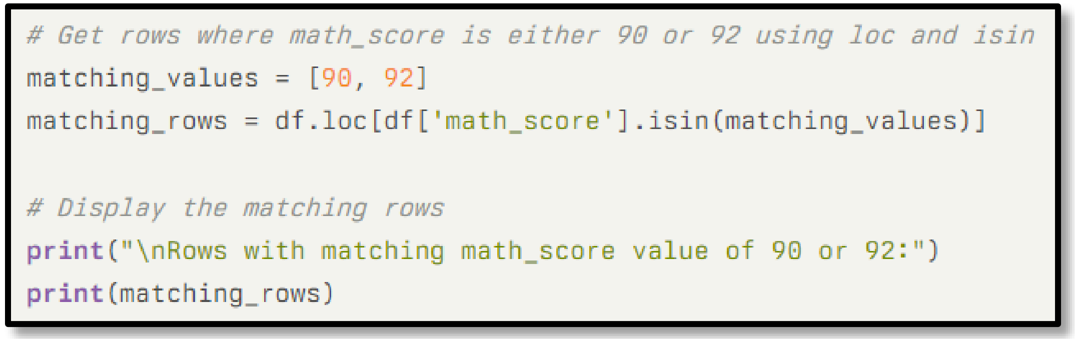


```
# Get rows where math_score is equal to 90 using loc
matching_value = 90
matching_rows = df.loc[df['math_score'] == matching_value]

# Display the matching rows
print("\nRows with matching math_score value of 90:")
print(matching_rows)
```

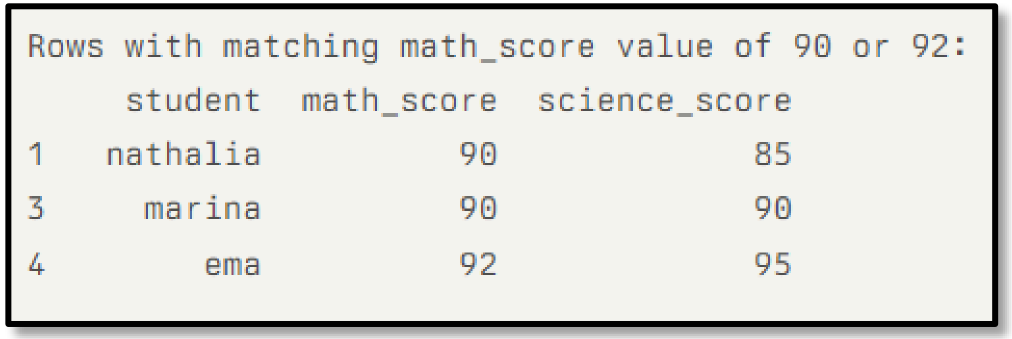


```
Rows with matching math_score value of 90:
    student  math_score  science_score
1  nathalia          90             85
3    marina          90             90
```

We can get the rows with multiple matching value at a specified column.



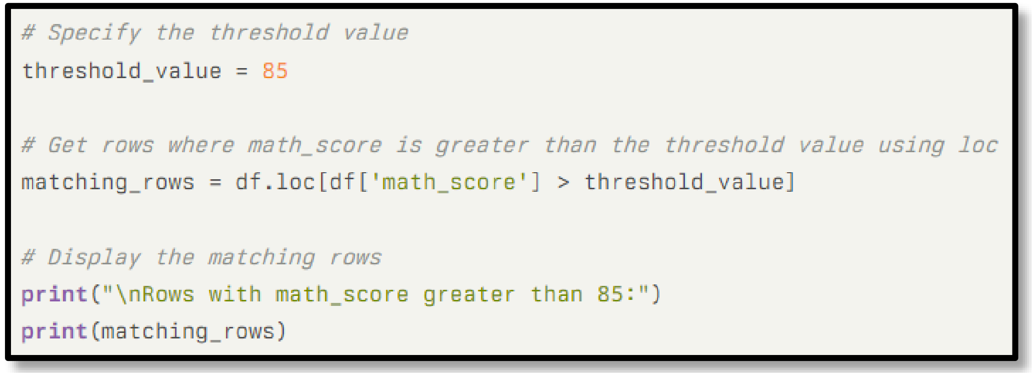
```
# Get rows where math_score is either 90 or 92 using loc and isin
matching_values = [90, 92]
matching_rows = df.loc[df['math_score'].isin(matching_values)]

# Display the matching rows
print("\nRows with matching math_score value of 90 or 92:")
print(matching_rows)
```



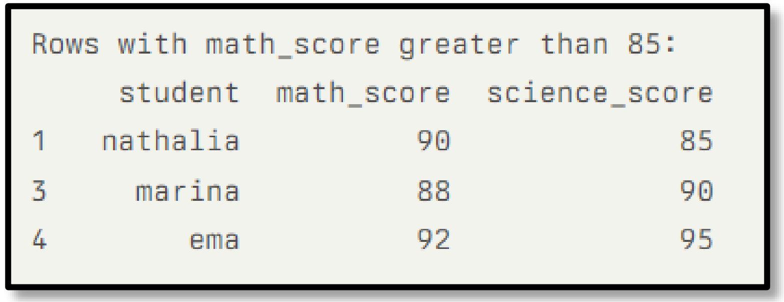
```
Rows with matching math_score value of 90 or 92:
    student  math_score  science_score
1  nathalia          90             85
3    marina          90             90
4       ema          92             95
```

Filtering data in a dataframe

We can retrieve rows from a dataframe where the values in a specified column exceed a certain threshold.

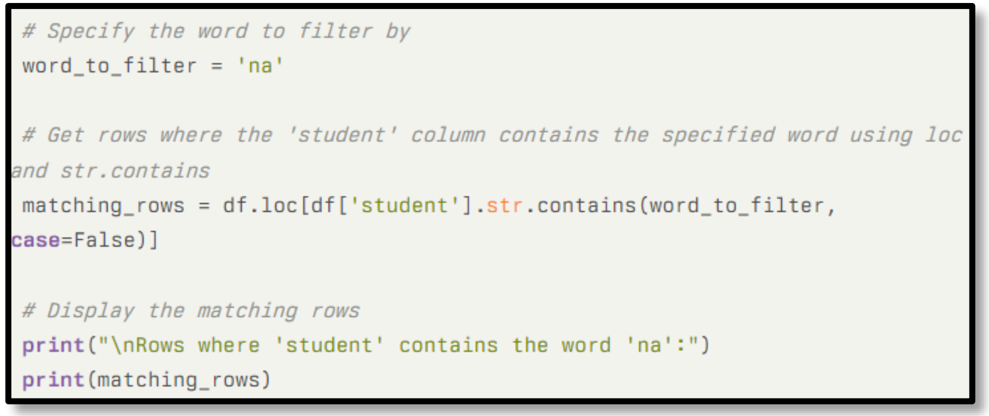


```
# Specify the threshold value
threshold_value = 85

# Get rows where math_score is greater than the threshold value using loc
matching_rows = df.loc[df['math_score'] > threshold_value]

# Display the matching rows
print("\nRows with math_score greater than 85:")
print(matching_rows)
```

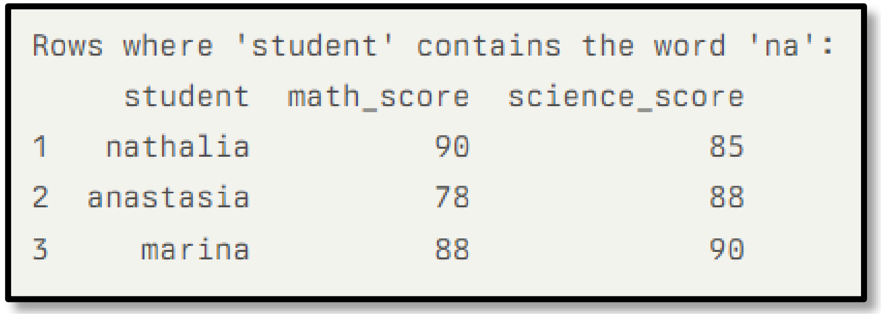


```
Rows with math_score greater than 85:
    student  math_score  science_score
1  nathalia          90             85
3    marina          88             90
4       ema          92             95
```

We can also filter rows in a dataframe based on whether a specific column contains a certain substring or word.



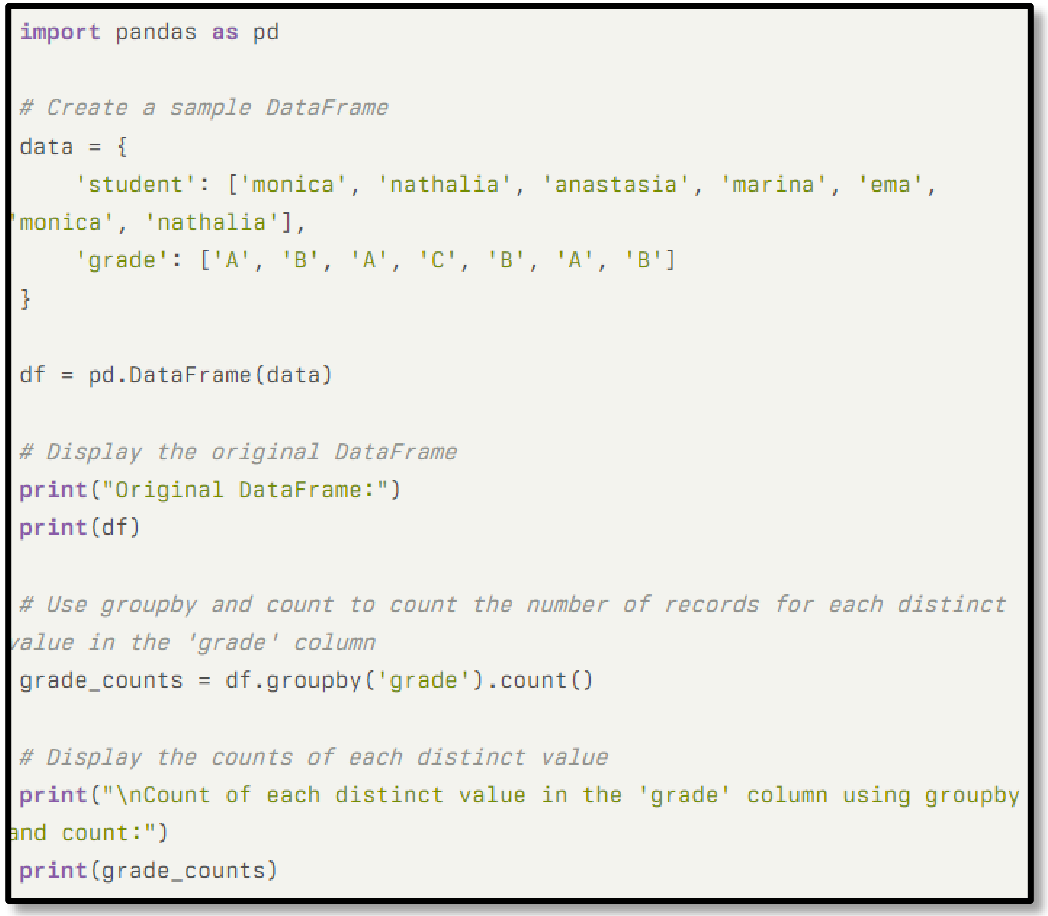
```
# Specify the word to filter by
word_to_filter = 'na'

# Get rows where the 'student' column contains the specified word using loc
and str.contains
matching_rows = df.loc[df['student'].str.contains(word_to_filter,
case=False)]

# Display the matching rows
print("\nRows where 'student' contains the word 'na':")
print(matching_rows)
```



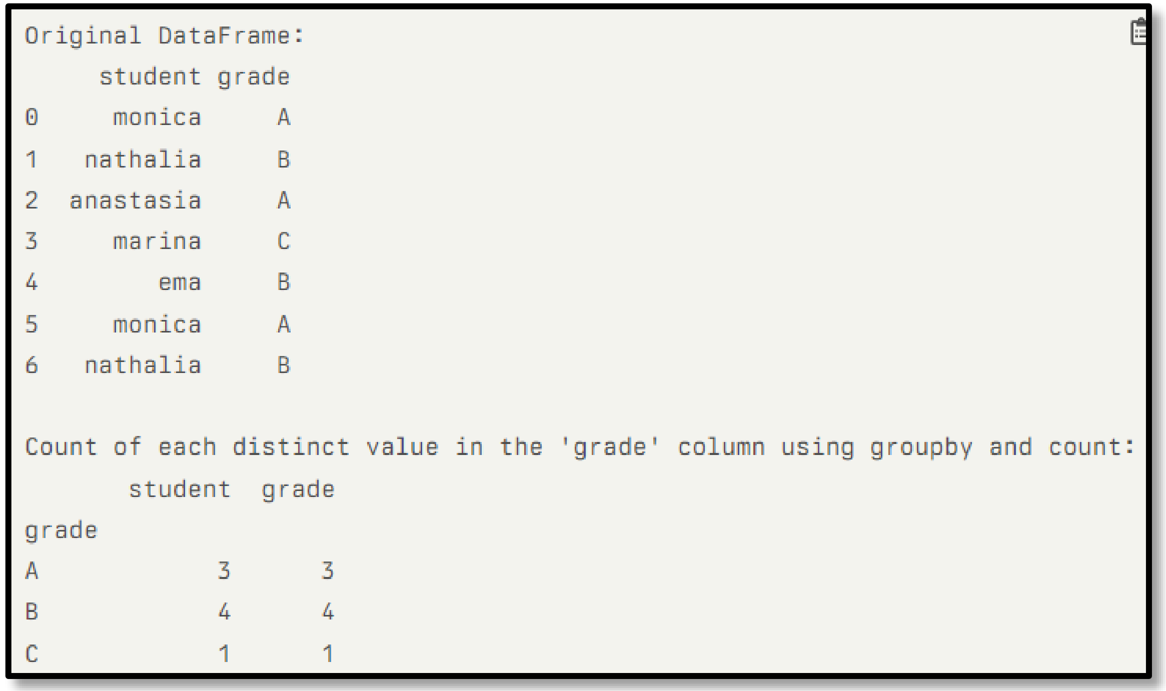
```
Rows where 'student' contains the word 'na':
     student  math_score  science_score
1   nathalia          90             85
2  anastasia          78             88
3     marina          88             90
```

Counting records in a dataframe

We can count the number of records for each distinct value in a specific column of a Pandas dataframe.



```
import pandas as pd

# Create a sample DataFrame
data = {
    'student': ['monica', 'nathalia', 'anastasia', 'marina', 'ema',
'monica', 'nathalia'],
    'grade': ['A', 'B', 'A', 'C', 'B', 'A', 'B']
}

df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Use groupby and count to count the number of records for each distinct
value in the 'grade' column
grade_counts = df.groupby('grade').count()

# Display the counts of each distinct value
print("\nCount of each distinct value in the 'grade' column using groupby
and count:")
print(grade_counts)
```



```
Original DataFrame:
     student grade
0     monica     A
1   nathalia     B
2  anastasia     A
3     marina     C
4        ema     B
5     monica     A
6   nathalia     B

Count of each distinct value in the 'grade' column using groupby and count:
       student  grade
grade
A            3      3
B            4      4
C            1      1
```