pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

one of the best libraries for manipulation and visualization data

The topics we will cover in this presentation.

- Installation and importation.
- Why use the Pandas library?
- Creation and usage of a Pandas Series.
- Creation and usage of a Pandas DataFrame.
- Basic operations on DataFrame objects.
- Advanced operations on DataFrame objects.
- Using Pandas for Deep and Machine Learning.

installation and importion

- to had the pandas librarie to the vemv

```
#Put In the Terminal
pip install numpy
pip install pandas
```

- import the libraries in your python project

```
#Import Python Libraries
import numpy as np
import pandas as pd
```

- NB: only if you use a local interpreter, if you work with Collab jupyter, Anaconda you only have to import them

Why use the Pandas library?

- Pandas is a popular Python library for data manipulation and analysis.
- It offers efficient data structures, like DataFrames and Series, making it easy to handle and analyze large datasets.
- With Pandas, you can clean and preprocess data, explore and analyze it, and integrate with other libraries. Its versatility, time series support, and flexibility make it a valuable tool for data scientists and analysts.

What is Pandas Series

- Pandas Series is a labeled one-dimensional data structure offered by the Pandas library. Here's why to use Pandas Series

- Flexibility: Pandas Series can hold data of different types, including numbers, strings, and dates. This allows for efficient storage and manipulation of heterogeneous data

- Indexing: Each element in a Pandas Series is labeled with an index, enabling quick and easy access to data. The index can be used for selection, filtering, and grouping operations on the data

- Advanced Functionality: Pandas Series offers a wide range of advanced features for data manipulation. This includes mathematical operations, transformation functions, boolean operations, date manipulations, and more.
- Integration with other Pandas features: Pandas Series seamlessly integrates with other Pandas data structures, such as DataFrames. This allows for performing complex operations on complete datasets.

Exemple of a Pandas Series in Code

- Creation the Data Series Object

```
# Creating a series with student grades
grades = pd.Series([85, 90, 75, 92, 88])

# Printing the series
print(grades)
```

- Result Output

```
0    85
1    90
2    75
3    92
4    88
dtype: int64
```

- In this example, we create a series "grades" containing student grades. When we print the series, we get the values along with their respective indices. In this case, the indices range from 0 to 4, and the values are the corresponding grades.

Next Exemple of a Pandas Series in Code

- Initialize Dictionary of student names and their corresponding grades

```
# Dictionary of student names and their corresponding grades
grades_dict = {'Alice': 85, 'Bob': 90, 'Charlie': 75, 'David': 92, 'Emily': 88}

# Creating a series from the dictionary
grades_series = pd.Series(grades_dict)

# Printing the series
print(grades_series)
```

- Result Output

```
Alice      85
Bob        90
Charlie    75
David      92
Emily      88
dtype: int64
```

- In this example, we have a dictionary where the keys represent student names and the values represent their corresponding grades. We use the dictionary to create a Pandas Series called grades_series. When we print the series, we see the student names as the indices and their grades as the values.

Usage of panda series in code

- Accessing values: You can access the values of the series using positional indexing

```
print(grades_series[0])   # Access the first value of the series (85)
```

- Accessing indexes: You can access the indexes of the series using the index property.

```
print(grades_series.index)   # Print the indexes of the series
```

- Mathematical operations: You can make mathematical operations on the series, for example, adding a value to each grade

```
updated_grades = grades_series + 5 # Add 5 to each grade
```

- Filtering data: You can filter the series based on certain conditions to select specific values.

```
filtered_grades = grades_series[grades_series > 80]
# Filter grades greater than 80
```

- Checking for value presence: You can check if a specific value is present in the series using the in operator.

```
print('Alice' in grades_series) # Check if 'Alice' is present in the
series (True)
```

Usage of panda series statistics and plot

- Getting statistics: You can obtain summary statistics of the series using methods like mean, min, max, sum, standard deviation, variance...

```
print(grades_series.mean()) # Calculate the mean grade
print(grades_series.min())  # Find the minimum grade
print(grades_series.std())  # Calculating standard deviation
print(grades_series.var())  # Calculating variance
```

- Data Visualization: Pandas Series can be used to create plots and visualizations.

```
import matplotlib.pyplot as plt

grades_series.plot(kind='bar')# Plot a bar chart of the series values
plt.show()   # Show the Data

NB: There are numerous different ways to graphically represent data. To
gain a better understanding of graphic libraries, please be patient and
wait for the introduction to Matplotlib or Seaborn
```

What is pandas DataFrame

- A Pandas DataFrame is a two-dimensional labeled data structure with rows and columns.
- It can hold data of different types in each column and is similar to a table in a relational database.
- It is ideal for working with multiple sets of related data and performing operations that involve multiple columns.
- In summary, while Pandas Series is designed for one-dimensional data, Pandas DataFrame is designed for working with two-dimensional tabular data, providing more versatility and functionality for handling complex datasets with multiple variables.
- the Pandas DataFrame is a more powerful tool than a Pandas Series because it provides a tabular data structure, data flexibility, advanced manipulation capabilities, and seamless integration with other data analysis libraries. It is an ideal choice for analyzing, manipulating, and transforming complex data.

Creation and Usage on DataFrame

- the most common ways to create a DataFrame
- Hard coding the data:

```
in code

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['Paris', 'London', 'New York']}

df = pd.DataFrame(data)
```

```
output

      Name    Age    City
0    Alice    25     Paris
1      Bob    30     London
2  Charlie    35     New York
```

- Converting a Pandas Series into a DataFrame:

```
in code

series = pd.Series([85, 90, 75, 92, 88],name='Grades')
df = pd.DataFrame(data)
```

```
output

   Grades
0      85
1      90
2      75
3      92
4      88
```

- Reading a CSV file to create a DataFrame

```
in code
df = pd.read_csv('name_file.csv') # the output depend the csv files
```

Basics operation on DataFrame

- Once the DataFrame is created and contains data, there is a long list of functions that we can call to help us examine our dataset. Here are some commonly used functions:
- *head(n): Displays the first n rows of the DataFrame.*
- *tail(n): Displays the last n rows of the DataFrame.*
- *info(): Displays information about the DataFrame, including data types and the number of non-null values.*
- *describe(): Provides descriptive statistics for the numeric columns of the DataFrame, such as mean, standard deviation, etc.*
- *shape: Returns the number of rows and columns in the DataFrame.*
- *columns: Returns the list of column names in the DataFrame.*
- *index: Returns the index of the DataFrame*
- *unique(): Returns the unique values in a column of the DataFrame.*

Advance operation on DataFrame

- *loc[row_index, col_index]: Accesses a specific value using label-based indexing.*
- *iloc[row_index, col_index]: Accesses a specific value using positional indexing.*
- *drop(labels): Removes specified rows or columns from the DataFrame.*

 # The drop(labels) function is extremely important, and we will come back to this function later in the practical case.

- *fillna(value): Fills missing values with a specified value.*
- *groupby(column): Groups the data based on values in a specified column*
- *sort_values(column): Sorts the DataFrame based on values in a specified column*
- *value_counts(): Counts the occurrences of each value in a column of the DataFrame.*

 # This has the merit of existing, but it is not very conclusive in the case of real datasets.

- *apply(func): Applies a specified function to a column or DataFrame.*

Advance operation on DataFrame

- So far, we have seen the different functions that allow us to examine the dataset, such as displaying the first/last rows, obtaining statistical information about the data, and performing various operations like selecting a specific row, column, or cell

- Now we will present specific operations with one or multiple DataFrames.

- *merge(df1, df2): Merges two DataFrames based on common columns.*
- *concat([df1, df2]): Concatenates multiple DataFrames along a specified axis.*
- *pivot_table(): Creates a pivot table from the DataFrame data.*
- *plot(): Generates plots and visualizations from the DataFrame data.*

- Now that we have all the necessary tools to examine and work with datasets, we will explore concrete examples on a real dataset in the next slides.

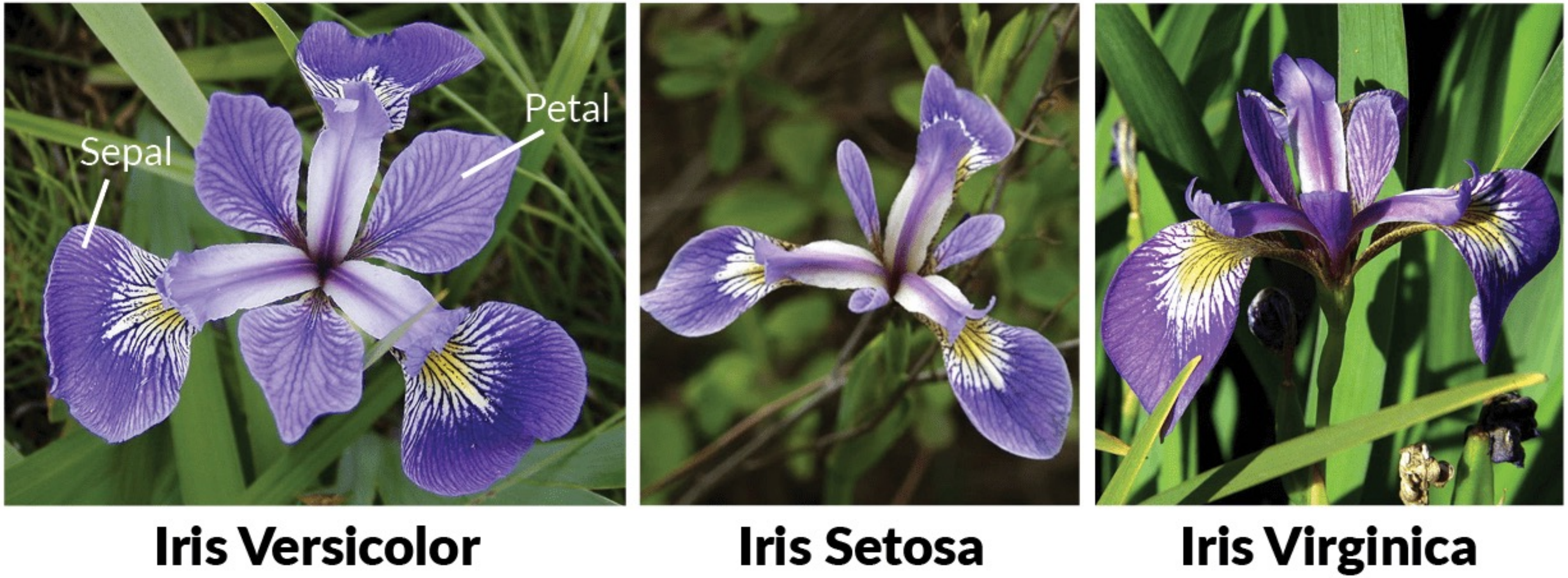
We have discussed enough, it's time to see the concrete implementations on a real dataset.

- I have chosen to do this using the Iris CSV dataset
- what is the Iris Data Set
- The Iris dataset is a popular and well-known dataset in the field of machine learning and statistics. It was introduced by the British statistician and biologist Ronald Fisher in 1936. The dataset contains measurements of various attributes of three different species of Iris flowers: Setosa, Versicolor, and Virginica. The attributes include sepal length, sepal width, petal length, and petal width.

- The goal of analyzing the Iris dataset is typically to explore the relationship between these attributes and the different Iris species. By examining the measurements, researchers aim to understand the patterns and differences among the species and potentially develop classification models to distinguish between them based on the attribute values.

- The Iris dataset is often used as a beginner's dataset in machine learning and data analysis due to its simplicity and well-defined classes. It serves as a great example for learning various data analysis techniques, data visualization, and classification algorithms.

Image of the different flower varieties in the Iris dataset



IrisPresentation.py

IrisPresentation
venv library ro
iris.csv
IrisPresentatio
External Libraries
Scratches and Co

```python

def IrisDataSet():
    # Load Iris dataset
    df = pd.read_csv('iris.csv')
    print()

    # 1. head(n)
    print("1. head(n)")
    print(df.head(5))
    print()

    # 2. tail(n)
    print("2. tail(n)")
    print(df.tail(5))
    print()

    # 3. info()
    print("3. info()")
    print(df.info())
    print()

    # 4. describe()
    print("4. describe()")
    print(df.describe())
    print()

```

IrisDataSet()

Run: IrisPresentation

iris.csv
IrisPresentatio
External Libraries
Scratches and Co

```
# 5. shape
print("5. shape")
print(df.shape)
print()

# 6. columns
print("6. columns")
print(df.columns)
print()

# 7. index
print("7. index")
print(df.index)
print()

# 8. loc[row_index, col_index]
print("8. loc[row_index, col_index]")
print(df.loc[0, 'variety'])
print()

# 9. iloc[row_index, col_index]
print("9. iloc[row_index, col_index]")
print(df.iloc[0, 4])
print()
```

IrisPresentation › IrisPresentation.py

```python
    # 10. drop(labels)
    print("10. drop(labels)")
    new_df = df.drop('variety', axis=1)
    print(new_df.head())
    print()

    # 11. fillna(value)
    print("11. fillna(value)")
    filled_df = df.fillna(0)
    print(filled_df.head())
    print()

    # 12. groupby(column)
    print("12. groupby(column)")
    grouped_df = df.groupby('variety').mean()
    print(grouped_df)
    print()

    # 13. sort_values(column)
    print("13. sort_values(column)")
    sorted_df = df.sort_values('sepal.length', ascending=False)
    print(sorted_df.head())
    print()

```

IrisPresentation › IrisPresentation.py

IrisPresentation.py

IrisPresentation
venv library root
iris.csv
IrisPresentation.py
External Libraries
Scratches and Consoles

```python
    # 14. unique()
    print("14. unique()")
    unique_species = df['variety'].unique()
    print(unique_species)
    print()

    # 15. value_counts()
    print("15. value_counts()")
    value_counts = df['variety'].value_counts()
    print(value_counts)
    print()

    # 16. apply(func)
    print("16. apply(func)")
    squared_sepal_length = df['sepal.length'].apply(lambda x: x ** 2)
    print(squared_sepal_length.head())
    print()

    # 17. merge(df1, df2)
    print("17. merge(df1, df2)")
    df1 = pd.DataFrame({'id': [1, 2, 3], 'name': ['John', 'Emma', 'Mark']})
    df2 = pd.DataFrame({'id': [2, 3, 4], 'age': [25, 30, 35]})
    merged_df = pd.merge(df1, df2, on='id')
    print(merged_df)
    print()

```

IrisDataSet()

iris.csv
IrisPresentatio
External Libraries
Scratches and Co

```python

    # 18. concat([df1, df2])
    print("18. concat([df1, df2])")
    df3 = pd.DataFrame({'id': [4, 5], 'name': ['Anna', 'David']})
    concatenated_df = pd.concat([df1, df2, df3], axis=0)
    print(concatenated_df)
    print()

    # 19. pivot_table()
    print("19. pivot_table()")
    pivot_table = df.pivot_table(index='variety', values=['sepal.length', 'petal.length'], aggfunc='mean')
    print(pivot_table)
    print()

    # 20. plot()
    print("20. plot()")
    df['sepal.length'].plot(kind='hist')
    plt.xlabel('Sepal Length')
    plt.ylabel('Frequency')
    plt.title('Histogram of Sepal Length')
    plt.show()

if __name__ == '__main__':
    IrisDataSet()

```

Output Code Results

Run: IrisPresentation

```
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa

2. tail(n)
     sepal.length  sepal.width  petal.length  petal.width    variety
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

3. info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

```
4. describe()
       sepal.length  sepal.width  petal.length  petal.width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
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

5. shape
(150, 5)

6. columns
Index(['sepal.length', 'sepal.width', 'petal.length', 'petal.width',
       'variety'],
      dtype='object')

7. index
RangeIndex(start=0, stop=150, step=1)

8. loc[row_index, col_index]
Setosa

9. iloc[row_index, col_index]
Setosa
```

```
10. drop(labels)
   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

11. fillna(value)
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa

12. groupby(column)
            sepal.length  sepal.width  petal.length  petal.width
variety
Setosa             5.006        3.428         1.462        0.246
Versicolor         5.936        2.770         4.260        1.326
Virginica          6.588        2.974         5.552        2.026

13. sort_values(column)
     sepal.length  sepal.width  petal.length  petal.width   variety
131           7.9          3.8           6.4          2.0  Virginica
135           7.7          3.0           6.1          2.3  Virginica
122           7.7          2.8           6.7          2.0  Virginica
117           7.7          3.8           6.7          2.2  Virginica
118           7.7          2.6           6.9          2.3  Virginica
```

```
14. unique()
['Setosa' 'Versicolor' 'Virginica']

15. value_counts()
variety
Setosa        50
Versicolor    50
Virginica     50
Name: count, dtype: int64

16. apply(func)
0    26.01
1    24.01
2    22.09
3    21.16
4    25.00
Name: sepal.length, dtype: float64

17. merge(df1, df2)
   id  name  age
0   2  Emma   25
1   3  Mark   30

18. concat([df1, df2])
   id   name   age
0   1   John   NaN
1   2   Emma   NaN
2   3   Mark   NaN
0   2    NaN  25.0
1   3    NaN  30.0
2   4    NaN  35.0
0   4   Anna   NaN
1   5  David   NaN
```

Plot the data

```
19. pivot_table()
            petal.length  sepal.length
variety
Setosa             1.462         5.006
Versicolor         4.260         5.936
Virginica          5.552         6.588

20. plot()

Process finished with exit code 0
```