MPSSDEGB
कौशल से रोजगार, समृद्धि अपार

FITT
Foundation For Innovation And Technology Transfer

SANKALP
Ministry of Skill Development
& Entrepreneurship

Day 10 -Lab and Q & A

Introduction to Pandas and Data Processing

Agenda

- Lab - Data Analysis with Python
- Critiquing and improving model evaluations
- Complete a project on data analysis, featuring preprocessing and feature engineering

Data Processing

Unleashing the Power of Python for Data Manipulation

- **Pandas** is an open-source data manipulation and analysis library for Python.
- It provides a powerful and flexible framework for data manipulation, exploration, and analysis, making it a go-to tool for data scientists and analysts working with tabular data in Python.
- It is particularly well-suited for working with tabular and heterogeneous data, making it a fundamental tool for data scientists, analysts, and researchers.
- Pandas is is often used in conjunction with other libraries such as NumPy, Matplotlib, and Scikit-Learn.

```
import pandas as pd
```

Data Processing

Key features and components of Pandas:

- DataFrame
- Series
- Data Input/Output
- Data Cleaning and Transformation
- Indexing and Selection
- GroupBy
- Data Visualization
- Time Series Data, etc

Data Processing

DataFrame

A Pandas **DataFrame** is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns.

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Data Processing

Locate Row

loc attribute to return one or more specified row(s)

```
#refer to the row index:
print(df.loc[0])
```

```
calories    420
duration     50
Name: 0, dtype: int64
```

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

```
   calories  duration
0       420        50
1       380        40
```

Named Indexes

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Data Processing

```
#refer to the named index:
print(df.loc["day2"])
```

```
calories    380
duration     40
Name: day2, dtype: int64
```

Data Processing

Load Files Into a DataFrame

- Load a comma separated file (CSV file) into a DataFrame:

data.csv

```
Duration,Pulse,Maxpulse,Calories
60,110,130,409.1
60,117,145,479.0
60,103,135,340.0
45,109,175,282.4
45,117,148,406.0
60,102,127,300.5
60,110,136,374.0
45,104,134,253.3
30,109,133,195.1
```

```
import pandas as pd

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

print(df)
```

```
     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
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Data Processing

Series

- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

Data Processing

GroupBy in Pandas:

The groupby function in Pandas is a powerful tool for grouping data based on one or more criteria and then applying a function to each group independently.

```
import pandas as pd

# Creating a sample DataFrame
data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Value': [10, 20, 30, 40, 50, 60]}
df = pd.DataFrame(data)

# Grouping by 'Category'
grouped = df.groupby('Category')
```

```
import pandas as pd
data={'Category': ['A','B','A','B','A','B'],'Value': [10,20,30,40,50,60]}
df=pd.DataFrame(data)
```

```
print(df)
```

```
  Category  Value
0        A     10
1        B     20
2        A     30
3        B     40
4        A     50
5        B     60
```

```
#Grouping by category
grouped=df.groupby('Category')
```

```
# Displaying the effect of grouping
for name, group in grouped:
    print(f"Category: {name}")
    print(group)
    print("\n")
```

```
Category: A
  Category  Value
0        A     10
2        A     30
4        A     50


Category: B
  Category  Value
1        B     20
3        B     40
5        B     60
```

Data Input with Pandas

1. **Reading CSV Files:**

   Use pd.read_csv('file.csv') to read data from a CSV file into a Pandas DataFrame.

2. **Reading Excel Files:**

   Utilize pd.read_excel('file.xlsx') for reading data from an Excel file.

3. **Reading Other Formats:**

   Pandas supports reading from various formats like JSON, SQL, HDF5, and more.

Data Exploration with Pandas

1. Viewing Data:

df.head() and df.tail() display the first and last rows of the DataFrame.

2. Summary Statistics:

df.describe() provides summary statistics of numerical columns.

3. Data Types:

df.dtypes shows the data types of each column.

4. Handling Missing Values:

df.isnull() identifies missing values, and df.dropna() or df.fillna() addresses them.

Data Output with Pandas

1.Writing to CSV:

df.to_csv('output.csv', index=False) saves the DataFrame to a CSV file.

2.Writing to Excel:

df.to_excel('output.xlsx', index=False) writes data to an Excel file.

3.Writing to Other Formats:

Pandas supports writing to JSON, SQL databases, HDF5, and more.

Data Selection and Manipulation

1. **Selecting Columns:**
   df['column_name'] selects a specific column.
2. **Filtering Data:**
   Use boolean indexing for filtering rows based on conditions.
   # Filter rows where Age is greater than 25 and Salary is less than 60000
   combined_condition_df = df[(df['Age'] > 25) & (df['Salary'] < 60000)]
1. **Sorting Data:**
   df.sort_values('column_name') sorts the DataFrame based on a column.
2. **Grouping and Aggregating:**
   df.groupby('group_column').agg({'agg_column': 'aggregate_function'}) groups data and performs aggregation.
3. **Merging and Joining:**
   pd.merge(df1, df2, on='common_column') merges two DataFrames based on a common column.
4. **Creating New Columns:**
   Define new columns using arithmetic operations or functions applied to existing columns.

```
# Create a new column 'AgeGroup' based on age categories
df['AgeGroup'] = pd.cut(df['Age'], bins=[20, 30, 40, 50], labels=['20-30', '30-40', '40-50'])
```

Additional Pandas Functions

Additional Pandas Functions:

1.Handling DateTime:

Use pd.to_datetime() to convert a column to DateTime format.

2.Reshaping Data:

pd.pivot_table() and pd.melt() for reshaping data.

3.Applying Functions:

df.apply() and df.applymap() for applying functions to elements or entire columns.

4.Statistical Analysis:

df.corr() for correlation matrix and df.cov() for covariance matrix.

Data Correlations

The corr() method calculates the relationship between each column in your data set.

```
df.corr()
```

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922425
Pulse	-0.155408	1.000000	0.786535	0.029110
Maxpulse	0.009403	0.786535	1.000000	0.206020
Calories	0.922425	0.029110	0.206020	1.000000

Result Explained (Given an exercise tracking.dataset)

- The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.
- The number varies from -1 to 1.
- 1 means that there is a 1 to 1 relationship (a perfect correlation)
- 0.9 is also a good relationship
- -0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.
- 0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

Data Visualization

- The process of finding trends and correlations in our data by representing it pictorially is called Data Visualization.
- It graphically plots data and is an effective way to communicate inferences from data.
- We can use various python data visualization modules such as Matplotlib, Seaborn, Plotly, etc.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Visualization

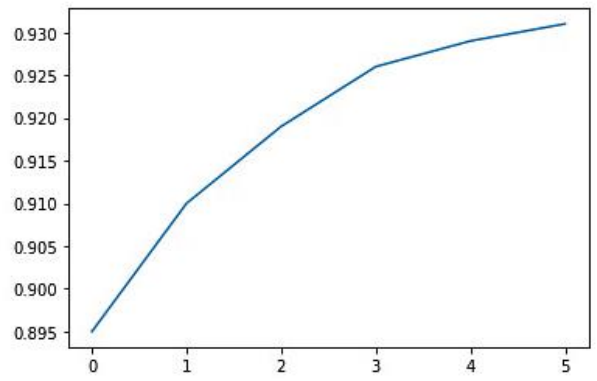
Line Charts

- A Line chart is a graph that represents information as a series of data points connected by a straight line.
- Each data point or marker is plotted and connected with a line or curve.

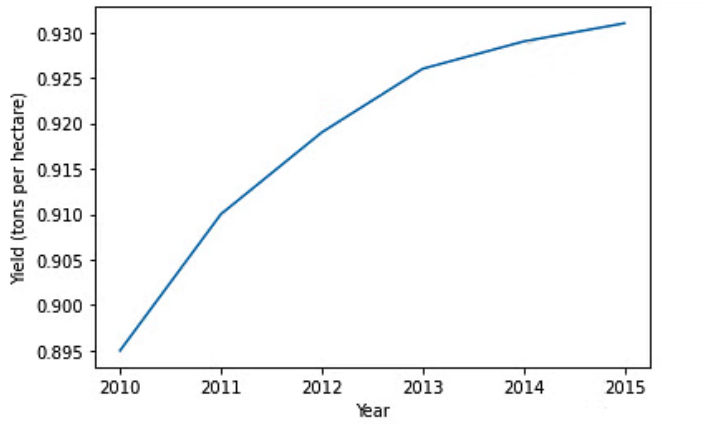
```
yield_apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
```

```
plt.plot(yield_apples)
```

```
[<matplotlib.lines.Line2D at 0x2054a700d30>]
```




```
plt.plot(years, yield_apples)
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)');
```



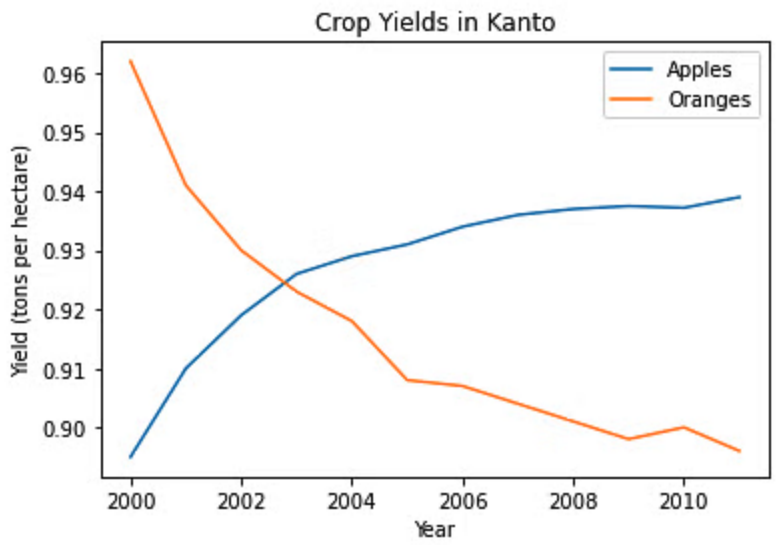
Data Visualization

Line Graph - Plotting multiple graphs

```
plt.plot(years, apples)
plt.plot(years, oranges)

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```




```
plt.plot(years, apples, marker='o')
plt.plot(years, oranges, marker='x')

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges'])
```

```
<matplotlib.legend.Legend at 0x2054a8940d0>
```

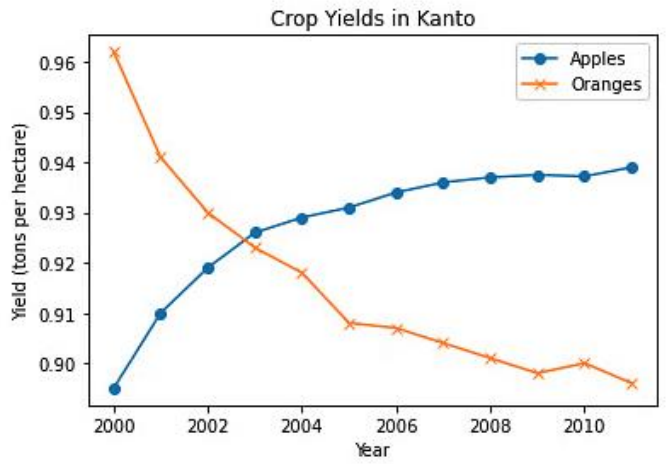


Figure 8: Using markers

Data Visualization

Plotting multiple graphs Using Seaborn

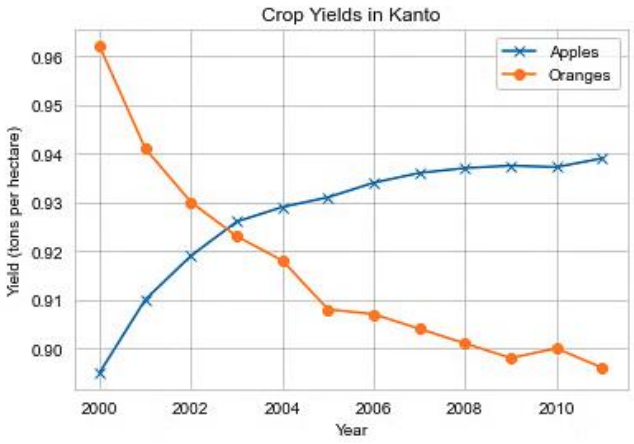
```
sns.set_style("whitegrid")
```

```
plt.plot(years, apples, marker = 'x')
plt.plot(years, oranges, marker = 'o')

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges'])
```

```
<matplotlib.legend.Legend at 0x2054aa1d1f0>
```



Data Visualization

Bar Graphs

- Bar graphs use bars with varying heights to show the data which belongs to a specific category.

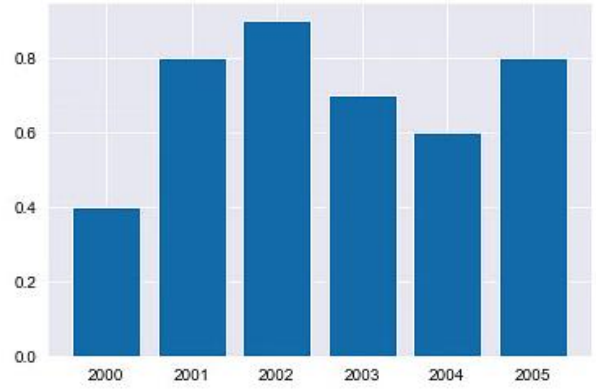
```
years = range(2000, 2006)
apples = [0.35, 0.6, 0.9, 0.8, 0.65, 0.8]
oranges = [0.4, 0.8, 0.9, 0.7, 0.6, 0.8]
```

```
plt.bar(years, oranges)

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
```

```
<BarContainer object of 6 artists>
```



Data Visualization

Bar Graphs

- The Seaborn library also provides a barplot function that can automatically compute averages.

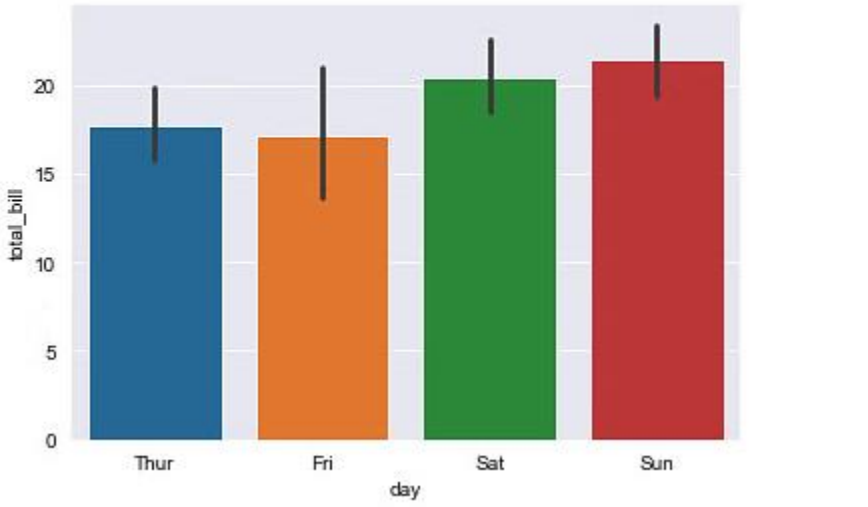
```
tips_df = sns.load_dataset("tips")
tips_df
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

```
sns.barplot(x='day', y='total_bill', data=tips_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054afa7370>
```



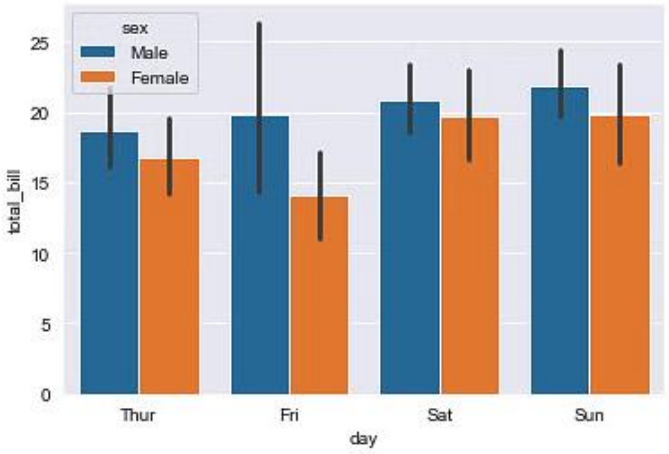
Data Visualization

Bar Graphs

- If you want to compare bar plots side-by-side, you can use the hue argument.

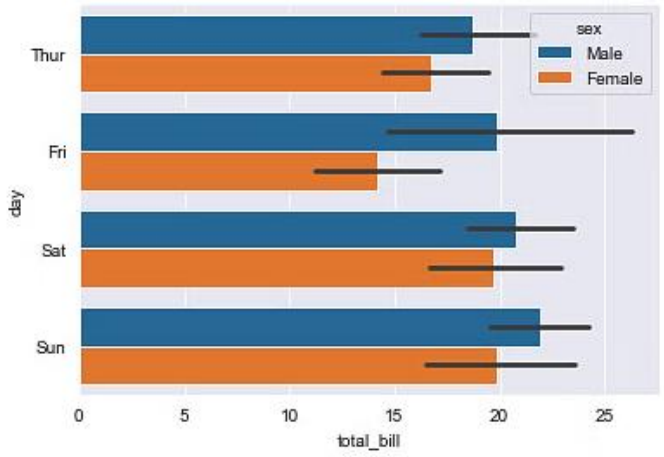
```
sns.barplot(x='day', y='total_bill', hue='sex', data=tips_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054b046700>
```




```
sns.barplot(x='total_bill', y='day', hue='sex', data=tips_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054af1b250>
```



Data Visualization

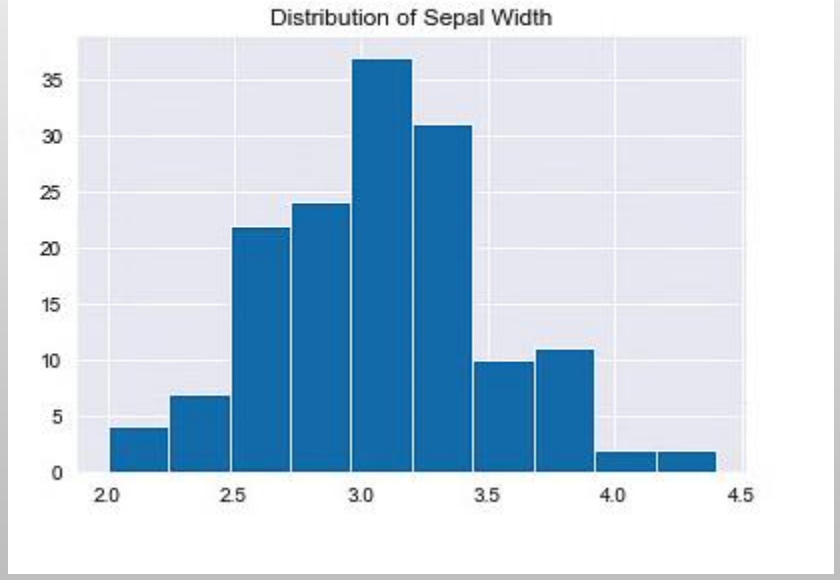
Histograms

- It is a bar representation of data that varies over a range.
- It plots the height of the data belonging to a range along the y-axis and the range along the x-axis.

```
flowers_df = sns.load_dataset("iris")
flowers_df.sepal_width
```

```
0      3.5
1      3.0
2      3.2
3      3.1
4      3.6
      ...
145    3.0
146    2.5
147    3.0
148    3.4
149    3.0
Name: sepal_width, Length: 150, dtype: float64
```

```
plt.title("Distribution of Sepal Width")
plt.hist(flowers_df.sepal_width)
```



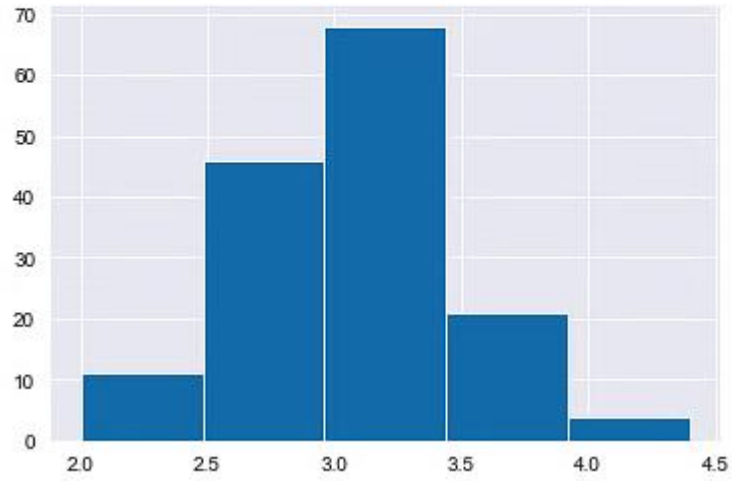
Data Visualization

Histograms

- We can control the number or size of bins too.

```
# Specifying the number of bins
plt.hist(flowers_df.sepal_width, bins=5)
```

```
(array([11., 46., 68., 21.,  4.]),
 array([2.  , 2.48, 2.96, 3.44, 3.92, 4.4 ]),
 <a list of 5 Patch objects>)
```



Data Visualization

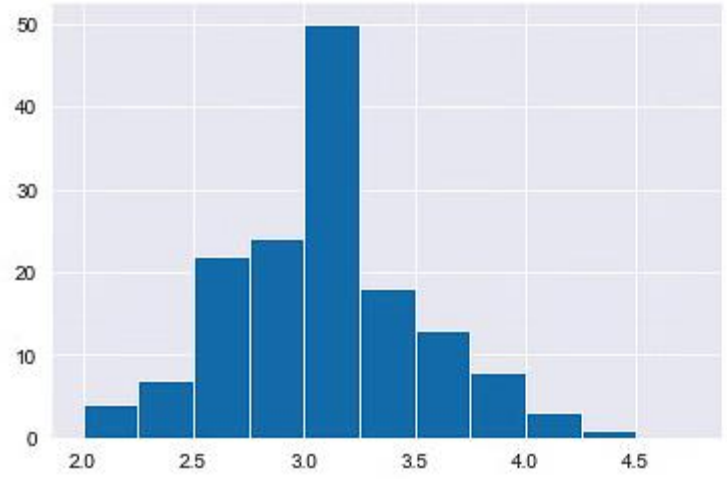
Histograms

- We can change the number and size of bins using numpy too.

```
import numpy as np

# Specifying the boundaries of each bin
plt.hist(flowers_df.sepal_width, bins=np.arange(2, 5, 0.25))
```

```
(array([ 4.,  7., 22., 24., 50., 18., 13.,  8.,  3.,  1.,  0.]),
 array([2.  , 2.25, 2.5 , 2.75, 3.  , 3.25, 3.5 , 3.75, 4.  , 4.25, 4.5 ,
        4.75]),
 <a list of 11 Patch objects>)
```



Data Visualization

Scatter Plots

- Two or more variables are plotted in a Scatter Plot, with each variable being represented by a different color.

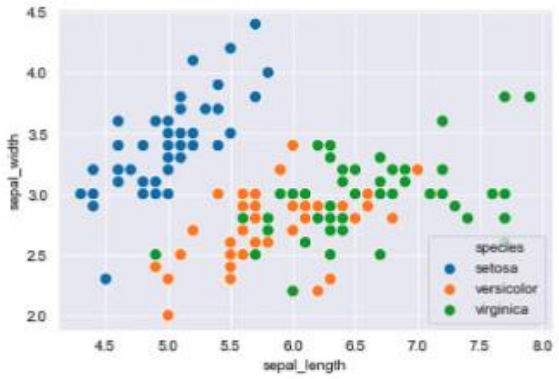
```
flowers_df = sns.load_dataset("iris")
```

```
flowers_df
```

	sepal_length	sepal_width	petal_length	petal_width	species
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
...	...	...	...	...	
145	6.7	3.0	5.2	2.3	virgin
146	6.3	2.5	5.0	1.9	virgin
147	6.5	3.0	5.2	2.0	virgin
148	6.2	3.4	5.4	2.3	virgin
149	5.9	3.0	5.1	1.8	virgin

150 rows × 5 columns

```
sns.scatterplot(x=flowers_df.sepal_length, y=flowers_df.sepal_width, hue=flowers_df.species, s=70);
```



Data Visualization

Heat Maps

- Heatmaps are graphical representations of data where values in a matrix are represented as colors.
- They are widely used for visualizing relationships, patterns, or distributions in a dataset.

```
flights_df = sns.load_dataset("flights").pivot("month", "year", "passengers")
flights_df
```

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
June	135	149	178	218	243	264	315	374	422	435	472	535
July	148	170	199	230	264	302	364	413	465	491	548	622
August	148	170	199	242	272	293	347	405	467	505	559	606
September	136	158	184	209	237	259	312	355	404	404	463	508
October	119	133	162	191	211	229	274	306	347	359	407	461
November	104	114	146	172	180	203	237	271	305	310	362	390
December	118	140	166	194	201	229	278	306	336	337	405	432

Figure 31: Flights dataset

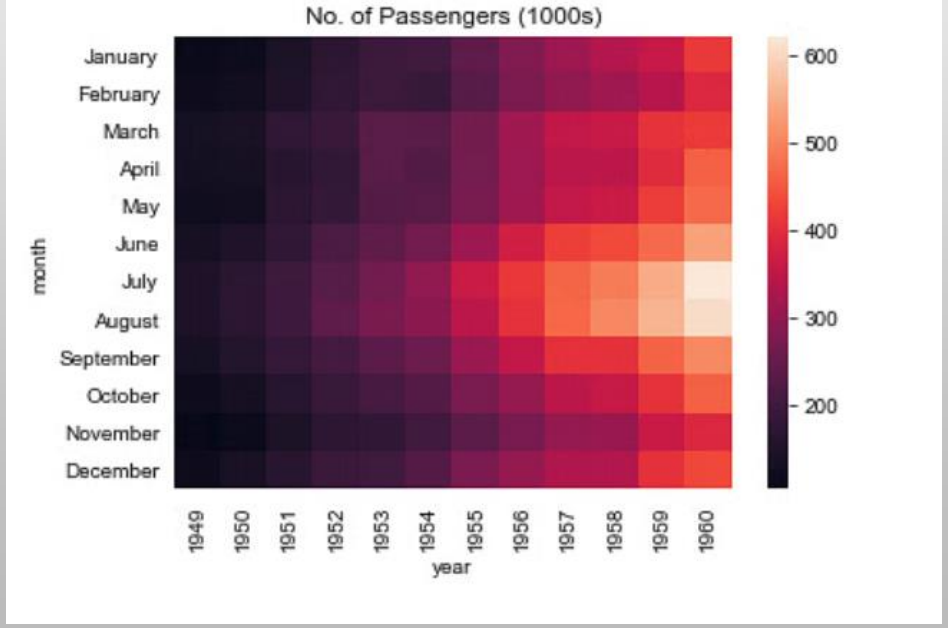
Data Visualization

Heat Maps

The above dataset, flights_df shows us the monthly footfall in an airport for each year, from 1949 to 1960. The values represent the number of passengers (in thousands) that passed through the airport. Let's use a heatmap to visualize the above data.

```
plt.title("No. of Passengers (1000s)")
sns.heatmap(flights_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054c1feaf0>
```



The brighter the color, the higher the footfall at the airport. By looking at the graph, we can infer that :

- The annual footfall for any given year is highest around July and August.
- The footfall grows annually. Any month in a year will have a higher footfall when compared to the previous years.

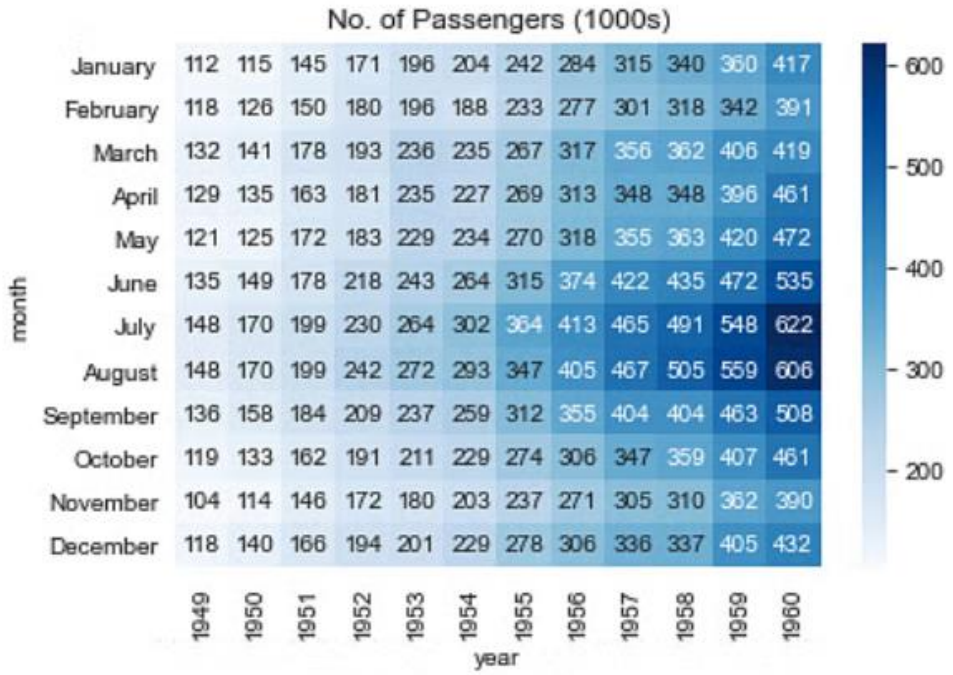
Data Visualization

Heat Maps

- To display the actual values in our heatmap and change the hue to blue.

```
plt.title("No. of Passengers (1000s)")
sns.heatmap(flights_df, fmt="d", annot=True, cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054c2dc850>
```



Quiz

Quiz - 12 MCQs

https://forms.office.com/r/DCycXtTBQW

Colab Notebooks to familiarize with data analysis, featuring preprocessing and feature engineering**:**

1. **House Price Prediction using Linear Regression**

   House Price Prediction Dataset

   (Understand the dataset)

1. **Cancer Prediction - Classification.**

https://drive.google.com/file/d/1U1TuZN_VGJjcAF4qTAZAmVdV5n_ZQ5D6/view

References

1. https://www.analyticsvidhya.com/blog/2021/05/a-comprehensive-guide-to-data-analysis-using-pandas-hands-on-data-analysis-on-imdb-movies-data/
2. https://www.w3schools.com/python/pandas/pandas_dataframes.asp
3. https://pandas.pydata.org/docs/getting_started/index.html#getting-started
4. https://web.cse.ohio-state.edu/~shen.94/5544/pandas.pdf
5. https://www.simplilearn.com/tutorials/python-tutorial/data-visualization-in-python
6. https://colab.research.google.com/drive/1nHQA8OkzUUTy25sNfkTu64GyvVd7Gobi?usp=sharing

THANKS