UNIT 5

Software Tools and data for visualization

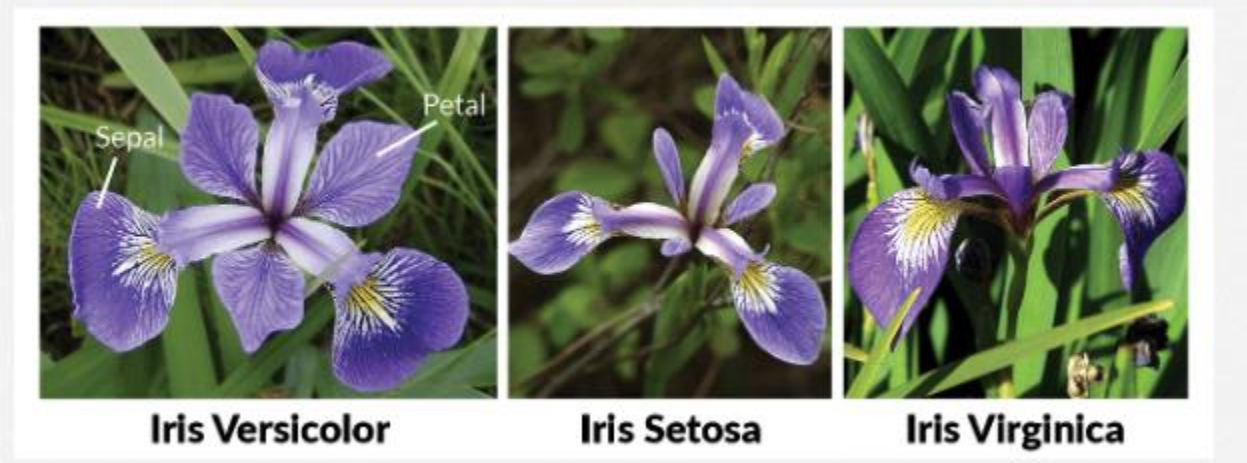
The Iris Datastet

This is an example of a notebook to demonstrate concepts of Data Science.

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification and clustering examples and to test algorithms.

Information about the original paper and usages of the dataset can be found in the UCI Machine Learning Repository -- Iris Data Set.

Just for reference, here are pictures of the three flowers species:



Iris Dataset is considered as the Hello World for data science. It contains five columns namely – Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally.

Note: This dataset can be downloaded from here.
You can download the Iris.csv file from the above link. Now we will use the Pandas library to load this CSV file, and we will convert it into the dataframe. read_csv() method is used to read CSV files.
Example:
Python3

```
import pandas as pd
```

```
# Reading the CSV file
df = pd.read_csv("Iris.csv")

# Printing top 5 rows
df.head()
```

Output:

	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

Getting Information about the Dataset

We will use the shape parameter to get the shape of the dataset.

Example:
Python3

```
df.shape
```

Output:
(150, 6)

We can see that the dataframe contains 6 columns and 150 rows.

Now, let's also the columns and their data types. For this, we will use the info() method.

Example:
Python3

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

We can see that only one column has categorical data and all the other columns are of the numeric type with non-Null entries.

Let's get a quick statistical summary of the dataset using the **describe()** method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc. Any missing value or NaN value is automatically skipped. describe() function gives a good picture of the distribution of data.

Example:

Python3

```
df.describe()
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

We can see the count of each column along with their mean value, standard deviation, minimum and maximum values.

Checking Missing Values

We will check if our data contains any missing values or not. Missing values can occur when no information is provided for one or more items or for a whole unit. We will use the isnull() method.

Example:

Python3

```
df.isnull().sum()
```

Output:

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

We can see that no column as any missing value.

Note: For more information, refer Working with Missing Data in Pandas.

Checking Duplicates

Let's see if our dataset contains any duplicates or not. Pandas drop_duplicates() method helps in removing duplicates from the data frame.

Example:

Python3

```
data = df.drop_duplicates(subset ="Species",)
data
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
100	101	6.3	3.3	6.0	2.5	Iris-virginica

We can see that there are only three unique species. Let's see if the dataset is balanced or not i.e. all the species contain equal amounts of rows or not. We will use the Series.value_counts() function. This function returns a Series containing counts of unique values.

Example:

Python3

```
df.value_counts("Species")
```

Output:

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

We can see that all the species contain an equal amount of rows, so we should not delete any entries.

Data Visualization

Visualizing the target column

Our target column will be the Species column because at the end we will need the result according to the species only. Let's see a countplot for species.

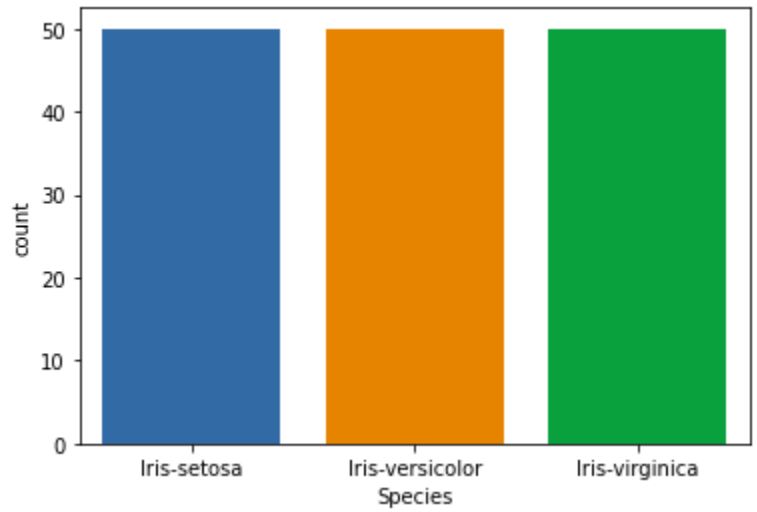
Note: *We will use Matplotlib and Seaborn library for the data visualization. If you want to know about these modules refer to the articles –*

- *Matplotlib Tutorial*
- *Python Seaborn Tutorial*

Example:
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


sns.countplot(x='Species', data=df, )
plt.show()
```

Output:



Relation between variables

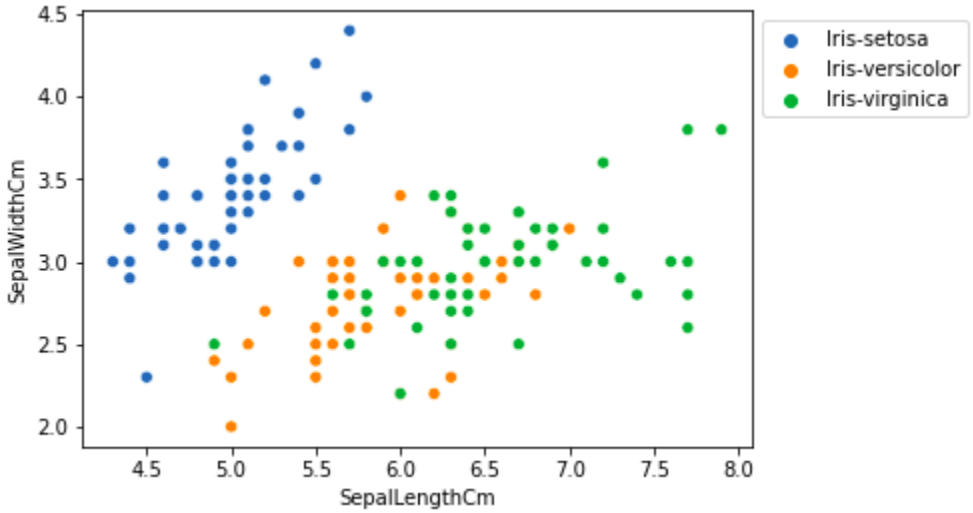
We will see the relationship between the sepal length and sepal width and also between petal length and petal width.

Example 1: Comparing Sepal Length and Sepal Width
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm',
                hue='Species', data=df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```

Output:



From the above plot, we can infer that –

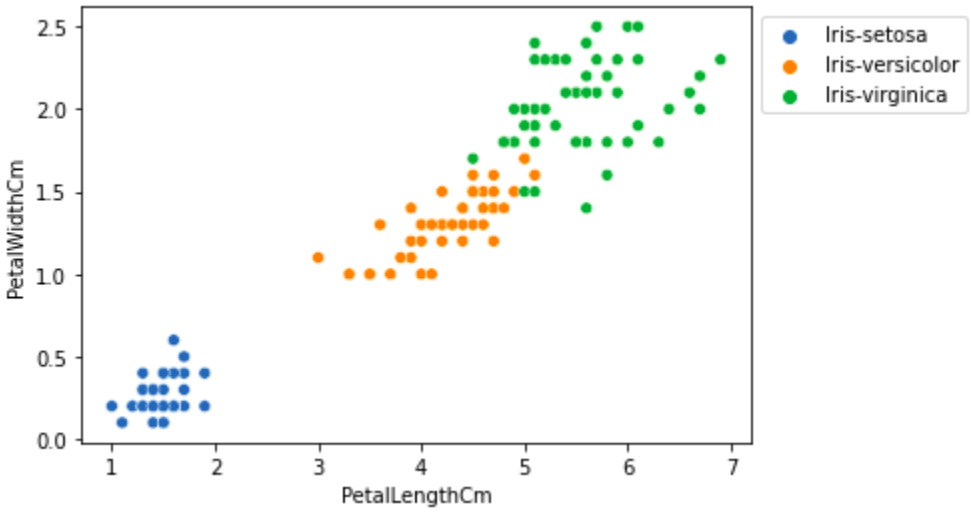
- Species Setosa has smaller sepal lengths but larger sepal widths.
- Versicolor Species lies in the middle of the other two species in terms of sepal length and width
- Species Virginica has larger sepal lengths but smaller sepal widths.

Example 2: Comparing Petal Length and Petal Width

Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


sns.scatterplot(x='PetalLengthCm', y='PetalWidthCm',
                hue='Species', data=df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```

Output:



From the above plot, we can infer that –

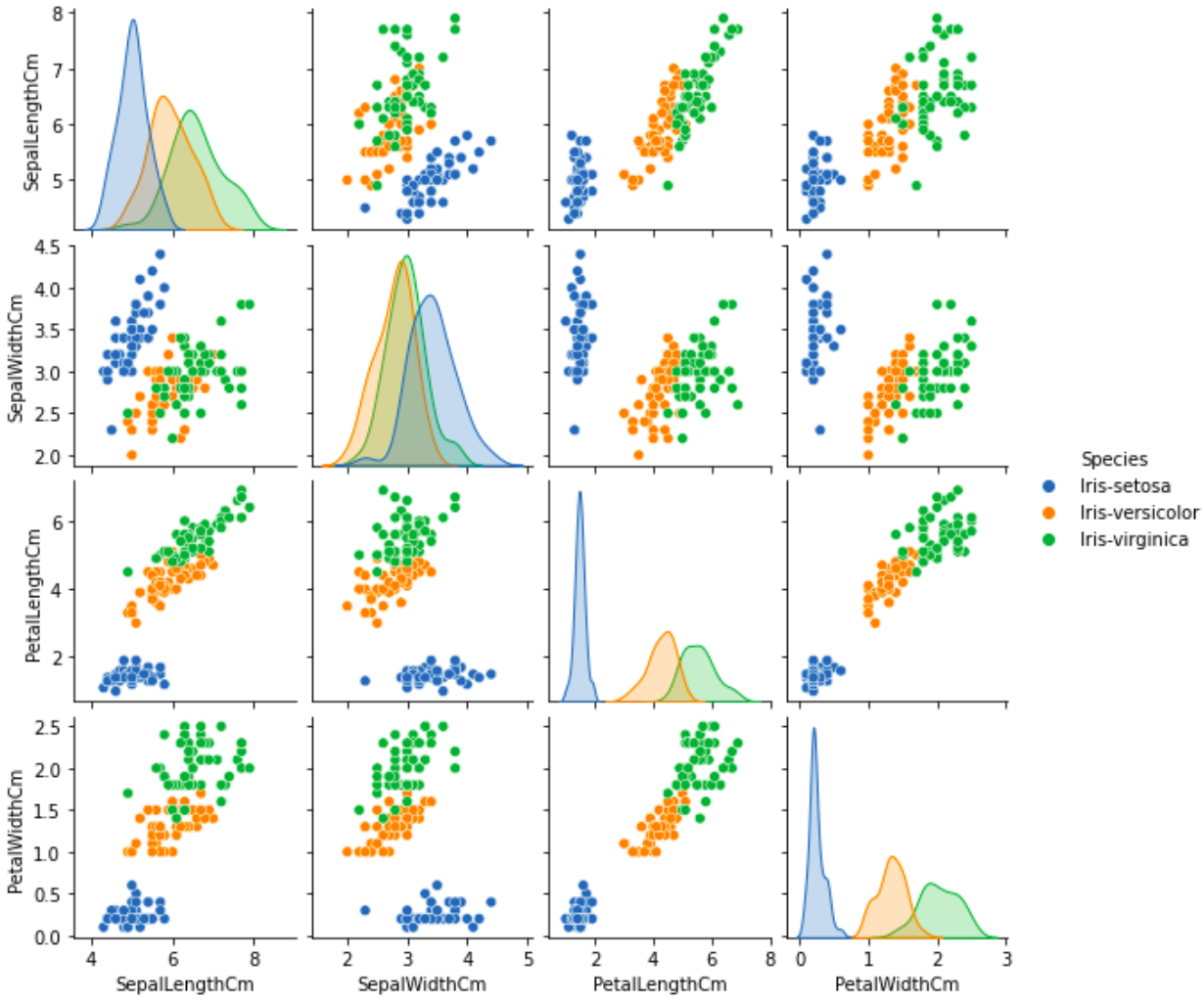
- Species Setosa has smaller petal lengths and widths.
- Versicolor Species lies in the middle of the other two species in terms of petal length and width
- Species Virginica has the largest of petal lengths and widths.

Let's plot all the column's relationships using a pairplot. It can be used for multivariate analysis.

Example:
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


sns.pairplot(df.drop(['Id'], axis = 1),
             hue='Species', height=2)
```

Output:



We can see many types of relationships from this plot such as the species Setosa has the smallest of petals widths and lengths. It also has the smallest sepal length but larger sepal widths. Such information can be gathered about any other species.

Histograms

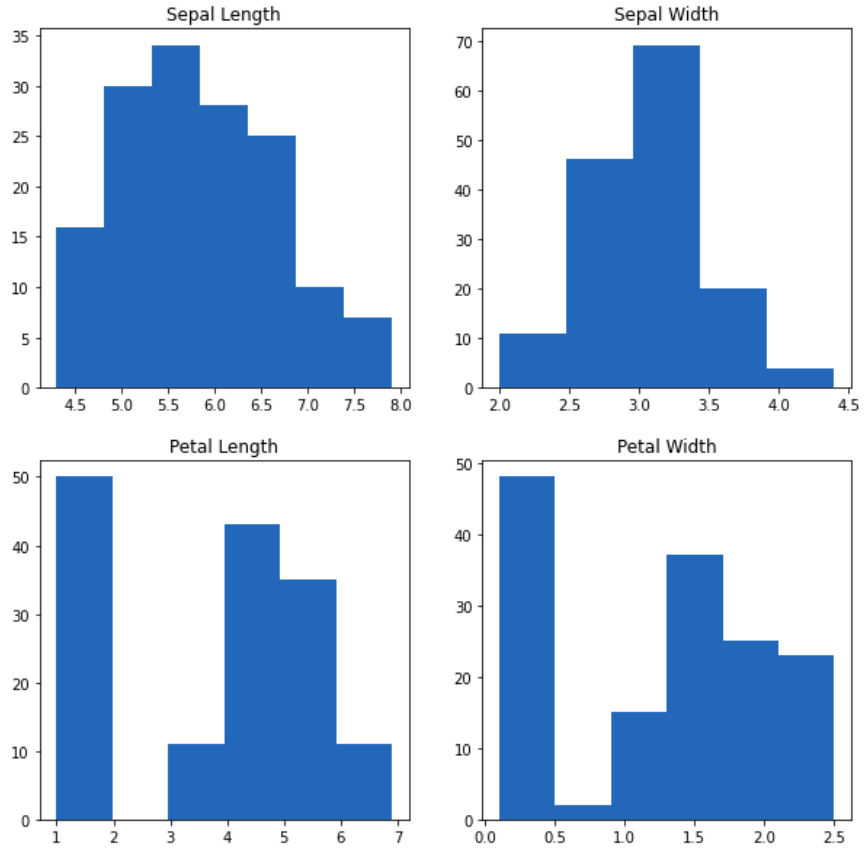
Histograms allow seeing the distribution of data for various columns. It can be used for uni as well as bi-variate analysis.

Example:
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
```

```
fig, axes = plt.subplots(2, 2, figsize=(10,10))

axes[0,0].set_title("Sepal Length")
axes[0,0].hist(df['SepalLengthCm'], bins=7)

axes[0,1].set_title("Sepal Width")
axes[0,1].hist(df['SepalWidthCm'], bins=5);

axes[1,0].set_title("Petal Length")
axes[1,0].hist(df['PetalLengthCm'], bins=6);

axes[1,1].set_title("Petal Width")
axes[1,1].hist(df['PetalWidthCm'], bins=6);
```

Output:



From the above plot, we can see that –

- The highest frequency of the sepal length is between 30 and 35 which is between 5.5 and 6
- The highest frequency of the sepal Width is around 70 which is between 3.0 and 3.5

- The highest frequency of the petal length is around 50 which is between 1 and 2
- The highest frequency of the petal width is between 40 and 50 which is between 0.0 and 0.5

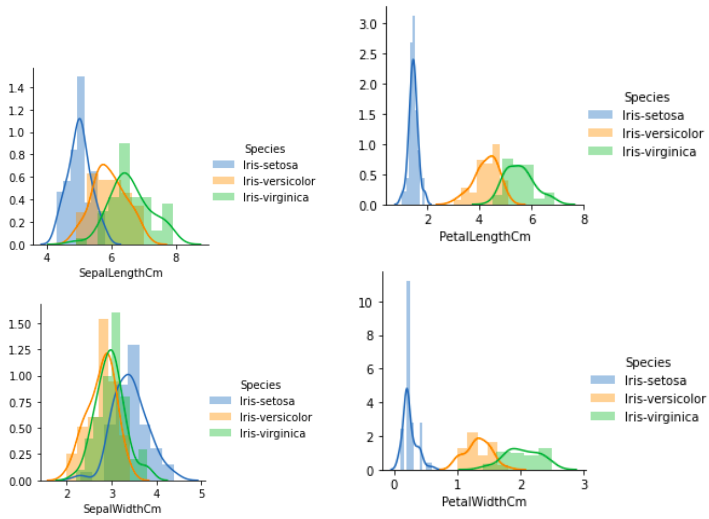
Histograms with Distplot Plot

Distplot is used basically for the univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.

Example:
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "SepalLengthCm").add_legend()

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "SepalWidthCm").add_legend()

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "PetalLengthCm").add_legend()

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "PetalWidthCm").add_legend()

plt.show()
```

Output:



From the above plots, we can see that –

- In the case of Sepal Length, there is a huge amount of overlapping.
- In the case of Sepal Width also, there is a huge amount of overlapping.
- In the case of Petal Length, there is a very little amount of overlapping.
- In the case of Petal Width also, there is a very little amount of overlapping.

So we can use Petal Length and Petal Width as the classification feature.

Handling Correlation

Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

Example:

Python3

```
data.corr(method='pearson')
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.624413	-0.654654	0.969909	0.999685
SepalLengthCm	0.624413	1.000000	-0.999226	0.795795	0.643817
SepalWidthCm	-0.654654	-0.999226	1.000000	-0.818999	-0.673417
PetalLengthCm	0.969909	0.795795	-0.818999	1.000000	0.975713
PetalWidthCm	0.999685	0.643817	-0.673417	0.975713	1.000000

Heatmaps

The heatmap is a data visualization technique that is used to analyze the dataset as colors in two dimensions. Basically, it shows a correlation between all numerical variables in the dataset. In simpler terms, we can plot the above-found correlation using the heatmaps.

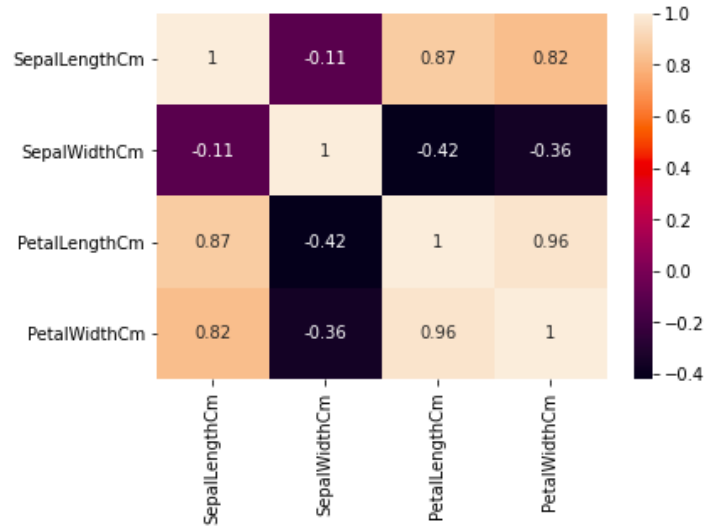
Example:

Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


sns.heatmap(df.corr(method='pearson').drop(
  ['Id'], axis=1).drop(['Id'], axis=0),
```

```
        annot = True);

plt.show()
```

Output:



From the above graph, we can see that –

- Petal width and petal length have high correlations.
- Petal length and sepal width have good correlations.
- Petal Width and Sepal length have good correlations.

Box Plots

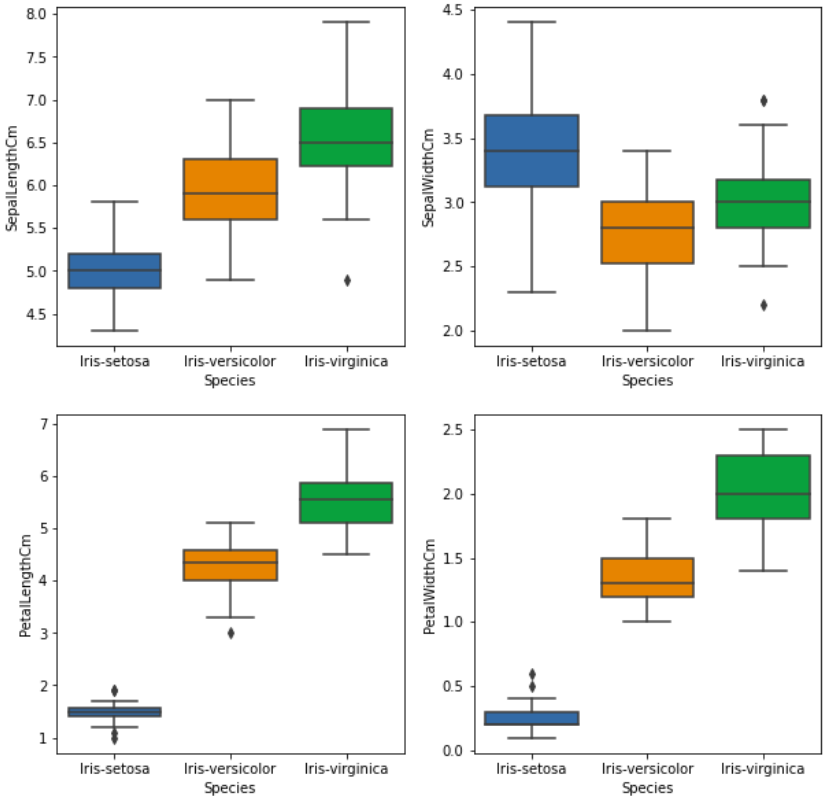
We can use boxplots to see how the categorical value os distributed with other numerical values.

Example:
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

def graph(y):
    sns.boxplot(x="Species", y=y, data=df)

plt.figure(figsize=(10,10))

# Adding the subplot at the specified
```

```
# grid position
plt.subplot(221)
graph('SepalLengthCm')

plt.subplot(222)
graph('SepalWidthCm')

plt.subplot(223)
graph('PetalLengthCm')

plt.subplot(224)
graph('PetalWidthCm')

plt.show()
```

Output:



From the above graph, we can see that –

- Species Setosa has the smallest features and less distributed with some outliers.
- Species Versicolor has the average features.
- Species Virginica has the highest features

Handling Outliers

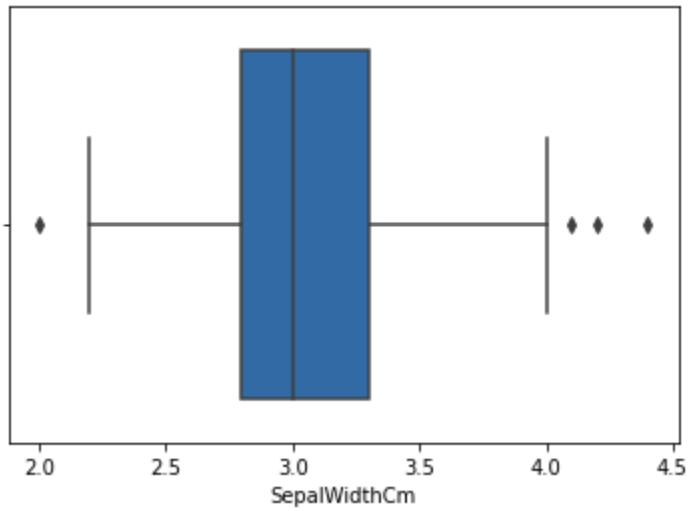
An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal)objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda's dataframe.

Let's consider the iris dataset and let's plot the boxplot for the SepalWidthCm column.

Example:
Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Iris.csv')

sns.boxplot(x='SepalWidthCm', data=df)
```

Output:



In the above graph, the values above 4 and below 2 are acting as outliers.

Removing Outliers

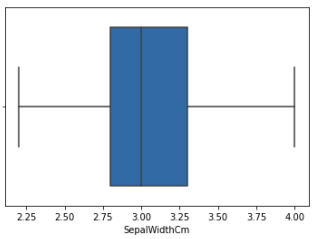
For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

Example: We will detect the outliers using IQR and then we will remove them. We will also draw the boxplot to see if the outliers are removed or not.
Python3

```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns

# Load the dataset
df = pd.read_csv('Iris.csv')

# IQR
Q1 = np.percentile(df['SepalWidthCm'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(df['SepalWidthCm'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", df.shape)

# Upper bound
upper = np.where(df['SepalWidthCm'] >= (Q3+1.5*IQR))

# Lower bound
lower = np.where(df['SepalWidthCm'] <= (Q1-1.5*IQR))

# Removing the Outliers
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)

print("New Shape: ", df.shape)

sns.boxplot(x='SepalWidthCm', data=df)
```

Output:

```
Old Shape:  (150, 6)
New Shape:  (146, 6)

<AxesSubplot:xlabel='SepalWidthCm'>
```



The Cereal Data Set

This dataset contains nutrition information about 77 cereals from 7 different manufacturers. The data displays the measurements of various items you would read on a nutrition label as well as information about serving sizes and the cereal's overall rating. This dataset might specifically appeal to people who eat cereal regularly and want to compare different types of cereal to find the healthiest option.

Variables

The fields in this dataset include:

- name: the name of the cereal
- mfr: the manufacturer of the cereal
    - A = American Home Food Products
    - G = General Mills
    - K = Kelloggs
    - N = Nabisco
    - P = Post
    - Q = Quaker Oats
    - R = Ralston Purina)
- type: the type of cereal
    - C = cold cereal
    - H = hot cereal
- calories: the number of calories per serving
- protein: the number of grams of protein per serving
- fat: the number of grams of fat per serving
- sodium: the number of milligrams of sodium per serving
- fiber: the number of grams of dietary fiber per serving
- carbo: the number of grams of complex carbohydrates per serving
- sugars: the number of grams of sugars per serving
- potass: the number of milligrams of potassium
- vitamins: the amount of vitamins and minerals
    - values = 0, 25, or 100 -> indicates the typical percentage of FDA recommended
- shelf: display shelf
    - values = 1, 2, or 3 - counting from the floor
- weight: the weight in ounces of one serving
- cups: the number of cups in one serving
- rating: a rating of the cereals - from consumer reports

The names of the cereals comprise the rows of the dataset whereas all of the other variables comprise the columns.

The Dow Jones Industrial Average Data

It measures the daily price movements of 30 large American companies on the Nasdaq and the New York Stock Exchange. The Dow Jones Industrial Average is widely viewed as a proxy for general market conditions and even the economy of the United States.

This Dow Jones Industrial Average dataset is downloaded from https://www.investing.com/indices/us-30-historical-data, including 2767 closing records from January 4th 2009 to December 31st 2019.
Today, the stock market is an important part of the lives of many people. People are looking to earn extra money in the stock market to supplement their income. However, there is a large risk associated with investing in the stock market. Many people think why they should risk there hard-earned money in the stock market. Therefore, there is a huge need to predict the Stock Market.

And predicting the Dow Jones is a perfect way to start predicting the Stock Market.

MS Spread Sheet:

Spreadsheets are powerful tools for data analysis, but they can also be overwhelming and confusing if you don't know how to use them effectively. In this article, you will learn some of the best ways to use spreadsheets for data analysis, whether you are an office administrator, a student, or a hobbyist. You will discover how to organize, filter, sort, format, and visualize your data, as well as how to perform some common calculations and functions. By the end of this article, you will have a better understanding of how to leverage spreadsheets for your data analysis needs.

Organize your data

The first step to using spreadsheets for data analysis is to organize your data in a clear and consistent way. This means using descriptive and unique names for your columns and rows, avoiding blank cells or rows, and ensuring that your data types are consistent. For example, if you have a column of dates, make sure they are all formatted the same way, such as YYYY-MM-DD. Organizing your data will make it easier to manipulate, analyze, and interpret later.

Filter and sort your data

One of the most useful features of spreadsheets is the ability to filter and sort your data based on various criteria. Filtering allows you to hide or show certain rows or columns based on their values, while sorting allows you to arrange your data in ascending or descending order. For example, if you have a spreadsheet of sales data, you can filter by region, product, or date, and sort by revenue, profit, or quantity. Filtering and sorting your data will help you to identify patterns, trends, and outliers in your data.

Format your data

Another way to use spreadsheets for data analysis is to format your data to make it more readable and attractive. Formatting includes changing the font size, color, style, and alignment of your cells, as well as adding borders, shading, and conditional formatting. Conditional formatting allows you to apply different formats to your cells based on their values, such as highlighting the highest or lowest values, or using a color scale or data bars. Formatting your data will help you to emphasize the most important information and create a visual contrast in your data.

Visualize your data

One of the best ways to use spreadsheets for data analysis is to visualize your data using charts, graphs, and tables. Visualizing your data allows you to present your data in a more engaging and understandable way, as well as to reveal insights that might not be obvious from the numbers alone. For example, if you have a spreadsheet of customer feedback, you can use a pie chart to show the distribution of ratings, a bar chart to show the frequency of comments, or a table to show the summary statistics. Visualizing your data will help you to communicate your findings and recommendations more effectively.

Perform calculations and functions

Another powerful feature of spreadsheets is the ability to perform calculations and functions on your data. Calculations are simple arithmetic operations that you can perform on your cells, such as adding, subtracting, multiplying, or dividing. Functions are predefined formulas that you can use to perform more complex calculations, such as finding the average, median, or standard deviation of a range of cells, or applying logical, text, or date functions. For example, if you have a spreadsheet of employee data, you can use the SUM function to calculate the total salary, the AVERAGE function to calculate the average age, or the IF function to assign a bonus based on performance. Performing calculations and functions on your data will help you to analyze and interpret your data more accurately and efficiently.

Excel data visualization steps:

Step 1: Enter the data into the Excel Spreadsheet or choose the data you want to visualize.

Step 2: The shortcut for making a chart is simply selecting a cell in the Excel data and clicking the F11 function key. To create a chart, click on the Insert tab and choose the desired chart from the list of available charts.

Step 3: The data entered in the excel sheet is used to create a chart.

Step 4: By choosing the design option, you may customize and decorate your chart with various colors and designs.

The Types of Data Visualization in Excel With Examples

We will illustrate the uses of various charts to grasp Excel's data visualization functionality better. It will acquaint you with the process of creating these Excel representations and using them to extract insights from data. The different **types of data visualization** examples using excel are:

1. Column Diagram

It is a basic sort of chart that displays data as vertical bars. To create a column chart, choose the data and the necessary choice from the Column chart menu. As we can see, the Column chart has a number of possibilities, and the appropriate one needs to be chosen. You can format the chart as necessary.

2. Pie Diagram

Pie diagrams or charts show how much percentage each category of data contributes. We can quickly understand the percentage contribution thanks to the pie chart. Select the necessary columns, then click the appropriate pie chart choice from the Pie menu to build a pie chart.

3. Bar Graph

The only distinction between this chart type and a column chart is the presence of horizontal bars. Choose the appropriate bar chart from the Bar option to create a horizontal bar.

4. Line Diagram

This graph helps identify trends. To create a line chart, choose the data and the necessary line chart choice.

5. Pivot Table

Pivot Table in data visualization is a tabular view of data that can summarize, sort, and group large amounts of data.

Python

Python offers several plotting libraries, namely Matplotlib, Seaborn and many other such data visualization packages with different features for creating informative, customized, and appealing plots to present data in the most simple and effective way.

Why Do Data Analysts Prefer Using Python?

The highly cross-functional language offers several perks to its users. It helps data analysts to make sense of complicated data sets and make them easier to understand.

Another pro of using Python is its high readability. Python code is easier for collaborating with other analysts, for communicating with other technical stakeholders, and it makes it more maintainable when it comes time to adapt it for new data sources and needs.

The language fits well for data analyst professionals as it provides heavy support and offers an extensive range of libraries for several tasks.

An Easy Learning Curve

Python is known for its simple syntax and readability, which is a major benefit. It cuts down the time data analysts otherwise spend familiarising themselves with a programming language.

The gentle learning curve makes it stand out among old programming languages with complicated syntax.

For example, languages like Java, C+, and Ruby require a steep learning curve, especially for beginner data analysts. On the other hand, the simplicity of Python helps data analysts perform various data-related tasks simultaneously.

Therefore, if you're dipping your toes in the data analytics field, you'll enjoy working with a simple yet effective programming language. Luckily, Python offers solutions for most complexities encountered when handling data.

Vast Collection of Libraries

Python offers an extensive list of free libraries to its users. What's more enticing about the libraries is that they grow consistently, offering powerful solutions.

Numpy, Scikit-learn, Pandas, and Matplotlib are a few popular libraries which help expedite data analytics tasks.

Well-Supported

Despite Python's simplicity, you will be in situations where you'll need help with the programming language. Fortunately, it offers an array of useful libraries with helpful support material. What's more, you can use them free of charge.

Besides, you can access user-contributed codes, from mailing lists to documentation and more. In addition, users globally can reach out to skilled programmers to ask for help and advice when needed.

Epic Visualisation Tools

Our brains process visuals better than text. Therefore, data analysts present meaningful insights into graphs, charts, or graphics to make them understandable. Fortunately, you do not need to be an expert at data visualisation as a beginner.

Python has you covered with its diverse data visualisation tools. You can convert your complicated datasets into graphics, charts, or interactive plots.

The better you present the data to the company's decision-makers, the better they'll understand what you managed to extract from the web. This will help them identify the trends and see what they can do to change the existing business strategies.

Extended Data Analytics Tools

Although data collection is essential to data analysis, you must also handle the extracted data effectively. Python has several built-in analytics tools to help you identify patterns and spot helpful information for better insights.

Python libraries are known to simplify the work of data analysts. Therefore, it is crucial to understand their primary features.

Some libraries that can help you with data analysis include the following.

- **Numpy.** This Python library offers computational tools to help you optimise your statistical and mathematical functions. Numpy Arrays (whether multidimensional or one-dimensional) allow you to effectively apply an operation to the entire array at once, making the code both shorter and more efficient. In addition, Numpy makes it easy to perform matrix operations, which are central to many machine learning algorithms.
- **Matplotlib.** Matplotlib allows you to make data visualisations by offering graph-based representations. It also lets you control the graphs. Hence, you can alter the shape, thickness, colours, and style per your preferences.
- **Pandas** is one of the most significant Python libraries concerning data analytics. It entails high-level data manipulation tools and structures that help during data cleaning and manipulation processes. You can clean your error-filled data and remove what's not needed. The library allows storing data in tabular format using Data Frame.
- **Scikit-learn.** This Python library helps you execute ML models. It automates the process of generating helpful insights from massive volumes of data.

Data Visualization with MATLAB

MATLAB is a programming and numeric computing platform used to analyze data, develop algorithms, and create models. It supports the entire data analysis workflow including acquiring the data directly into MATLAB; analyzing and visualizing that data; and exporting results. You can use interactive apps to visualize your data without writing any code; the apps will automatically generate the appropriate MATLAB code for you, so you can automate and reuse your work.

Creating Data Visualizations

MATLAB offers a wide range of built-in chart types like line plots, scatter charts, distribution plots, and geographic plots to visualize datasets from a diverse set of applications. You can create visualizations either interactively or programmatically using the MATLAB language.

Exploring Data Visualizations

You can interactively explore your visualization including:

- Zooming in and out on a specific section of the data set
- Interactively panning and rotating visualizations

- Displaying trend lines or data values directly on the visualization
- Shading and highlighting data points
- Switching between domains (e.g., time, frequency, S, Z domain)

Annotating and Customizing Data Visualizations

You can interactively annotate your visualizations by highlighting essential information that you wish to convey such as:

- Annotating key data points
- Adding data tips
- Adding axis labels
- Grouping by different colors and patterns
- Adding data markers, line styles, and colors

MATLAB automatically generates code from your interactive chart modifications. You can reuse that code by adding it to your script.

Complex data sets can be difficult to visualize using simple charts. MATLAB enables you to create custom charts to meet your visualization needs and add custom interactions to them.

Examples include:

- Sparklines Component—Create small line graphs that show the general trend of each vector in a multi-vector data set such as a table. Observe and compare data trends for each row/column.
- Density Scatter Chart—Use color (or transparency) to identify the density of points.
- Data visualization is often combined with data analysis and preprocessing. MATLAB apps, such as the Data Cleaner and the Signal Analyzer, combine these steps.

- Interactive controls allow you to specify operations without needing to write any code, and the corresponding data visualizations are integrated directly in the app. This allows you to immediately see the results of a given task. Once your analysis and preprocessing are complete, the apps can automatically generate the corresponding MATLAB code to allow you to automate the steps, even on different data.

Java

When it comes to data science, Java delivers a host of data science methods such as data processing, data analysis, data visualization statistical analysis, and NLP. Java and data science allow applying machine learning algorithms to real-world business products and applications. **Data Science**, Artificial Intelligence, and Machine Learning are tempting big money today. So if you can program in Java, you know you have an important skill. Hone your Java skills and use them for data science with our **data science training** that includes dedicated tutorials on data science for Java developers as well.

Java is suited for data science due to the following features -

Data Science Frameworks Using Java:

In order to stay relevant in the space of digital transformation, we suggest "selecting the right machine learning tool". Some of the **data processing frameworks** that use Java do the same for you. These frameworks help you come up with precise predictive models while your infrastructure can continue having the existing technology stack.

We are listing some of the Java and data science tools that would help you to keep a suitable interface to the production stack.

- DL4J for Deep Learning
- ADAMS for Advanced Data Mining
- Java for Machine Learning Library
- Neuroph for Object-Oriented Artificial Neural Network (ANN)
- RapidMiner for machine learning workflow
- Weka for Waikato Environment for Knowledge Analysis

Java is Easy To Understand :

Java is based on object-oriented programming, as a result it stays popular among programmers. While Java cannot be as easy as Python, it is fairly beginner-friendly and easy to understand.

Java Offers Scalability to Your Data Science Applications :

Java for data science is perfect when it comes to scaling your products and applications. This makes it a wonderful choice when you're considering building extensive and more complex ML/AI applications. If you are just starting out to create your products from the scratch, it is a good idea to choose Java as your programming language.

Unique Syntax in Data Science Using Java:

Java programmers are usually clear about the **data types**, variables, and data sources they deal with. It makes it easy for them to retain the code base and skip **documenting trivial unit tests cases** for products and applications. Java 8 included Lambda expressions, which corrected most of Java's rambling, thus making it less distressing to develop large business/data science tasks. Java 9 gets in the much-missed REPL, which enables iterative development.

Processing Speed and Compatibility:

Java is highly functional in several data science processes like data analysis, including data import, cleaning data, deep learning, statistical analysis, Natural Language Processing (NLP), and data visualization. The majority of code in Java is experimental. Java is a language that is statically typed and compiled, whereas Python is a dynamically organized and analyzed language. This single difference gives Java a faster runtime and more comfortable debugging.

Why Java for Data Science?

Java uses **Java Virtual Machine** (JVM) extensively for derivatives and frameworks that affect machine learning data analysis distributed systems in enterprise settings. Not just that, there are many reasons why Java if suitable for Data Science.

- Java is a language with a huge community in words
- Java is easy for almost all programmers to split functionality
- Java is strongly typed
- Java programming permits programmers to be explicit about data and types of variables data they deal with. This is helpful in the era of data science and data management machine learning.
- The suite of mechanisms for Java is rather well developed. A range of mature elements and IDEs allow developers to be well productive.

- The Java Virtual Machine (JVM) is especially good for documenting code that looks matching on multiple platforms and it works well the big data space.
- Scala is being used in machine learning technologies and **big data processing tools** like Apache Spark. Scala is built on the JVM and performs rather well with Java as compared to Scala.
- Decisions connected to programming language preference are made to reduce programming, features, code training maintainability and tooling.

What Does the Future Hold?

Data Science is disrupting businesses along with other latest technologies. The challenges that businesses dealing with data science face are selecting the right stack of technologies, onboarding the right set of developers with the right set of **data science skills**. Java developers can make use of data science to produce virtually any product and it's particularly well-suited for building scalable platforms.

In case you realize that the tech stack you are using has restrictions, you can expand it by making something in Java. It's more comfortable for Java developers to use technologies that need grid computing. Java for data science is becoming common not only because Java is the "best" programming language for Data Science but java developers are known to come up with visions that many data science products and applications are built upon.

Tableu

Tableau is a leading data visualization tool used for data analysis and business intelligence. Gartner's Magic Quadrant classified Tableau as a leader for analytics and business intelligence. This Tableau tutorial will cover a wide range of how-to topics, including how to create different charts and graphs, in addition to visualizing data through reports and dashboards in Tableau to derive meaningful insights.

Tableau is an excellent data visualization and business intelligence tool used for reporting and analyzing vast volumes of data. It is an American company that started in 2003—in June 2019,

Salesforce acquired Tableau. It helps users create different charts, graphs, maps, dashboards, and stories for visualizing and analyzing data, to help in making business decisions.

ableau has a lot of unique, exciting features that make it one of the most popular tools in business intelligence (BI). Let's learn more about some of the essential Tableau Desktop features. Now that we know what is tableau exactly, let us understand some of its salient features.

Tableau Features

- Tableau supports powerful data discovery and exploration that enables users to answer important questions in seconds
- No prior programming knowledge is needed; users without relevant experience can start immediately with creating visualizations using Tableau
- It can connect to several data sources that other BI tools do not support. Tableau enables users to create reports by joining and blending different datasets
- Tableau Server supports a centralized location to manage all published data sources within an organization

Tableau is capable of connecting with a wide range of data sources. It can connect to files present in your system, such as Microsoft Excel, text files, JSON, PDF, etc. It can also work on data present on a database server, such as Microsoft SQL Server, MySQL, Oracle, Teradata, etc. There are other saved data sources that Tableau can connect with. It also can connect and fetch data from cloud sources, like AWS, Azure SQL Data Warehouse, and Google Cloud SQL.