Experiment No 7

* 1. **Aim/Purpose of the Experiment**

To familiarize the students with data visualization using one feature variables.

* 1. **Learning Outcomes**

Knowledge of the Data cleaning, Data preparation and data visualization using univariate analysis in python.

* 1. **Prerequisites**

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

* 1. **Materials/Equipment/Apparatus / Devices/Software required**

Jupyter Notebook.

* 1. **Introduction and Theory**

Univariate analysis is a statistical method used to describe and understand the distribution, central tendency, and variability of a single variable. In Python, you can perform univariate analysis using libraries such as NumPy, Pandas, and Matplotlib/Seaborn for data manipulation, analysis, and visualization. Here's a brief outline of the process:

* Data Preparation: Load your dataset into a Pandas DataFrame and clean/preprocess the data if necessary. Ensure that the variable of interest is properly formatted and ready for analysis.
* Descriptive Statistics: Compute descriptive statistics for the variable of interest. This includes measures such as mean, median, mode, standard deviation, variance, minimum, maximum, and quartiles. These statistics provide an initial understanding of the distribution and characteristics of the variable.
* Visualization: Create visualizations to explore the distribution of the variable. Common plots for univariate analysis include histograms, box plots, density plots, and bar plots. Matplotlib and Seaborn are popular libraries for creating these visualizations.
* Central Tendency: Analyze the central tendency of the variable by examining the mean, median, and mode. This helps to understand the typical or central value around which the data is distributed.
* Variability: Explore the variability of the variable using measures such as standard deviation and variance. Variability indicates how spread out the data points are from the central tendency and provides insights into the dispersion of the data.
* Skewness and Kurtosis: Examine skewness and kurtosis to understand the shape of the distribution. Skewness measures the asymmetry of the distribution, while kurtosis measures the tail heaviness or peakedness of the distribution.
* Outlier Detection: Identify outliers in the data, which are data points that significantly deviate from the rest of the observations. Outliers can be detected visually using box plots or statistically using methods such as z-scores or interquartile range (IQR).
* Interpretation: Interpret the results of the analysis in the context of your research or problem domain. Draw conclusions about the distribution, central tendency, and variability of the variable, and consider any implications for further analysis or decision-making.

**Univariate Analysis**

import pandas as pd

import numpy as np

df=pd.read\_csv('iris.csv')

df

#Finding the data types of variables in the DataFrame

df.dtypes

#Importing libraries essential for data visualization

#MATPLOTLIB

import matplotlib.pyplot as plt

%matplotlib inline

#SEABORN

import seaborn as sns

#Plots for continuous variables' analysis

#ENUMERATIVE PLOTS

#UNIVARIATE SCATTER PLOT

plt.scatter(df.index,df['sepal.width'])

plt.show()

sns.scatterplot(x=df.index,y=df['sepal.width'],hue=df['variety'])

#LINE PLOT WITH MARKERS

#Setting title, figure size, labels and font size in matplotlib

plt.figure(figsize=(6,6))

plt.title('Line plot of petal length')

plt.xlabel('index',fontsize=20)

plt.ylabel('petal length',fontsize=20)

plt.plot(df.index,df['petal.length'],markevery=1,marker='d')

for name, group in df.groupby('variety'):

plt.plot(group.index, group['petal.length'], label=name,markevery=1,marker='d')

plt.legend()

plt.show()

#Setting title, figure size,labels and font size in seaborn

sns.set(rc={'figure.figsize':(7,7)})

sns.set(font\_scale=1.5)

fig=sns.lineplot(x=df.index,y=df['petal.length'],markevery=1,marker='d',data=df,hue=df['variety'])

fig.set(xlabel='index')

#STRIP PLOT

sns.stripplot(y=df['sepal.width'])

# Strip-plot(category wise)

sns.stripplot(x=df['variety'],y=df['sepal.width'])

#SWARM PLOT

#Setting figure size

sns.set(rc={'figure.figsize':(5,5)})

#Swarm-plot

sns.swarmplot(x=df['sepal.width'])

#Swarm-plot category wise

sns.swarmplot(x=df['variety'],y=df['sepal.width'])

#SUMMARY PLOTS

#HISTOGRAM

plt.hist(df['petal.width'])

sns.distplot(df['petal.width'],kde=False,color='black',bins=10)

#DENSITY PLOT

plt.figure(figsize=(5,5))

df['petal.length'].plot(kind='density')

sns.set(rc={'figure.figsize':(5,5)})

sns.kdeplot(df['petal.length'],shade=True)

#RUG PLOT

fig, ax = plt.subplots()

sns.rugplot(df['sepal.length'])

ax.set\_xlim(3,9)

plt.show()

from scipy import stats

import numpy as np

kdf=df['sepal.length'].to\_numpy()

rdf=np.hstack(kdf)

density = stats.kde.gaussian\_kde(rdf)

x = np.arange(3,9,0.1)

plt.plot(x, density(x))

plt.plot(rdf,[0.01]\*len(rdf), '|')

sns.distplot(df['sepal.length'],rug=True,hist=False)

#BOX PLOT

plt.boxplot(df['sepal.width'])

#Removing the column with categorical variables

dfM=df.drop('variety',axis=1)

plt.figure(figsize=(9,9))

#Set Title

plt.title('Box plots of the 4 variables')

plt.boxplot(dfM.values,labels=['SepalLength','SepalWidth','PetalLength','PetalWidth'])

sns.boxplot(df['sepal.width'])

sns.set(rc={'figure.figsize':(9,9)})

sns.boxplot(x="variable", y="value", data=pd.melt(dfM))

#distplot()

sns.set(rc={'figure.figsize':(6,6)})

sns.distplot(df['petal.length'],color='black',rug=True)

#VIOLIN PLOT

plt.figure(figsize=(7,7))

plt.violinplot(dfM.values,showmedians=True)

sns.set(rc={'figure.figsize':(5,5)})

sns.violinplot(df['sepal.width'],orient='vertical')

sns.set(rc={'figure.figsize':(9,9)})

sns.violinplot(x=df['variety'], y=df['petal.width'],data=df)

#Plots for categorical variables' analysis

#BAR PLOT

df['variety'].value\_counts().plot.bar()

sns.countplot(df['variety'])

#PIE CHART

plt.pie(df['variety'].value\_counts(),labels=['SETOSA','VERSICOLOR','VIRGINICA'],shadow=True)

df1=df.sample(frac=0.35)

plt.figure(figsize=(5,5))

plt.pie(df1['variety'].value\_counts(),startangle=90,autopct='%.3f',labels=['SETOSA','VERSICOLOR','VIRGINICA'],shadow=True)

* 1. **Operating Procedure**
* Open Jupyter note book
* Take a new python file
* Type the code
* Run it
* Take inputs from user
* Observe the results
* Verify the results manually
* Store the note book file
  1. **Precautions and/or Troubleshooting**

**Precautions:**

* Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).
* Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."
* Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.
* Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.
* Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.
* Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.
* Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

**Troubleshooting:**

* Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
* Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
* Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
* Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
* Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
* Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.
  1. **Observations**

Observe the results obtained in each operation.

* 1. **Calculations & Analysis**

Calculations should be given for each operation.

* 1. **Result & Interpretation**

Result should be printed and pasted in laboratory copy found from Jupyter note book.

* 1. **Follow-up Questions**
  + You need to check the relationship between the two variables. Which graph would you use?
  + You need to check if a variable has outliers. Which graph would you use?
  + You need to perform a univariate analysis. Which graph will you use?
  + What is a data cleaning step?
  + What are the ways to handle missing data?
  + What are some of the methods for univariate analysis?
  + What problems can outliers cause?
  1. **Extension and Follow-up Activities (if applicable)**

NA

* 1. **Assessments**
  2. **Suggested reading**