Experiment No 8

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

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

* 1. **Learning Outcomes**

Knowledge of the Data cleaning, Data preparation and data visualization using bivariate 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**

Bivariate analysis is a statistical method used to examine the relationship between two variables. In Python, you can perform bivariate 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 two variables of interest are numeric or can be appropriately converted into numeric format.
* Descriptive Analysis: Compute descriptive statistics for each variable separately using methods like mean, median, standard deviation, etc. This provides initial insights into the characteristics of the variables.
* Visualization: Create visualizations to explore the relationship between the two variables. Common plots for bivariate analysis include scatter plots, line plots, box plots, and correlation matrices. Seaborn is particularly useful for creating attractive statistical visualizations.
* Correlation Analysis: Calculate the correlation coefficient between the two variables to measure the strength and direction of the linear relationship. Pearson correlation coefficient is commonly used for this purpose.

**Case Study:**

Term deposits also called fixed deposits, are the cash investments made for a specific time

period ranging from 1 month to 5 years for predetermined fixed interest rates. The fixed interest

rates offered for term deposits are higher than the regular interest rates for savings accounts.

The customers receive the total amount (investment plus the interest) at the end of the maturity

period. Also, the money can only be withdrawn at the end of the maturity period. Withdrawing

money before that will result in an added penalty associated, and the customer will not receive

any interest returns.

Your target is to do end to end EDA on this bank telemarketing campaign data set to infer

knowledge that where bank has to put more effort to improve it's positive response rate.

**Bivariate Analysis**

#import the warnings.

import warnings

warnings.filterwarnings("ignore")

#import the useful libraries.

import pandas as pd, numpy as np

import matplotlib.pyplot as plt, seaborn as sns

%matplotlib inline

Session- 2, Data Cleaning

Segment- 2, Data Types

There are multiple types of data types available in the data set. some of them are numerical type

and some of categorical type. You are required to get the idea about the data types after reading

the data frame.

Following are the some of the types of variables:

• Numeric data type: banking dataset: salary, balance, duration and age.

• Categorical data type: banking dataset: education, job, marital, poutcome and month

etc.

• Ordinal data type: banking dataset: Age group.

• Time and date type

• Coordinates type of data: latitude and longitude type.

#read the data set of "bank telemarketing campaign" in inp0.

inp0= pd.read\_csv("bank\_marketing\_updated\_v1.csv")

#Print the head of the data frame.

inp0.head()

Segment- 3, Fixing the Rows and Columns

Checklist for fixing rows:

• Delete summary rows: Total and Subtotal rows

• Delete incorrect rows: Header row and footer row

• Delete extra rows: Column number, indicators, Blank rows, Page No.

Checklist for fixing columns:

• Merge columns for creating unique identifiers, if needed, for example, merge the

columns State and City into the column Full address.

• Split columns to get more data: Split the Address column to get State and City columns

to analyse each separately.

• Add column names: Add column names if missing.

• Rename columns consistently: Abbreviations, encoded columns.

• Delete columns: Delete unnecessary columns.

• Align misaligned columns: The data set may have shifted columns, which you need to

align correctly

#read the file in inp0 without first two rows as it is of no use.

inp0=pd.read\_csv("bank\_marketing\_updated\_v1.csv", skiprows= 2)

#print the head of the data frame.

inp0.head()

#drop the customer id as it is of no use.

inp0.drop("customerid", axis=1, inplace=True)

inp0.head()

#Extract job in newly created 'job' column from "jobedu" column.

inp0['job']=inp0.jobedu.apply(lambda x: x.split(",")[0])

inp0.head()

#Extract education in newly created 'education' column from "jobedu"

column.

inp0['education']=inp0.jobedu.apply(lambda x: x.split(",")[1])

inp0.head()

#drop the "jobedu" column from the dataframe.

inp0.drop('jobedu',axis= 1, inplace= True)

inp0.head()

inp0[inp0.month.apply(lambda x: isinstance(x,float))== True]

inp0.isnull().sum()

Segment- 4, Impute/Remove missing values

Take aways from the lecture on missing values:

• Set values as missing values: Identify values that indicate missing data, for example,

treat blank strings, "NA", "XX", "999", etc., as missing.

• Adding is good, exaggerating is bad: You should try to get information from reliable

external sources as much as possible, but if you can’t, then it is better to retain missing

values rather than exaggerating the existing rows/columns.

• Delete rows and columns: Rows can be deleted if the number of missing values is

insignificant, as this would not impact the overall analysis results. Columns can be

removed if the missing values are quite significant in number.

• Fill partial missing values using business judgement: Such values include missing time

zone, century, etc. These values can be identified easily.

Types of missing values:

• MCAR: It stands for Missing completely at random (the reason behind the missing value

is not dependent on any other feature).

• MAR: It stands for Missing at random (the reason behind the missing value may be

associated with some other features).

• MNAR: It stands for Missing not at random (there is a specific reason behind the missing

value).

#count the missing values in age column.

inp0.age.isnull().sum()

#pring the shape of dataframe inp0

inp0.shape

#calculate the percentage of missing values in age column.

float(100.0\*20/45211)

#drop the records with age missing in inp0 and copy in inp1 dataframe.

inp1=inp0[-inp0.age.isnull()].copy()

inp1.shape

#count the missing values in month column in inp1.

inp1.month.isnull().sum()

#print the percentage of each month in the data frame inp1.

float(100.0\*50/45191)

#find the mode of month in inp1

month\_mode=inp1.month.mode()[0]

month\_mode

# fill the missing values with mode value of month in inp1.

inp1.month.fillna(month\_mode, inplace= True)

inp1.month.value\_counts(normalize= True)

#let's see the null values in the month column.

inp1.month.isnull().sum()

0

#count the missing values in response column in inp1.

inp1.response.isnull().sum()

30

#calculate the percentage of missing values in response column.

float(100.0\*30/45191)

0.06638489964816004

#drop the records with response missings in inp1.

inp1= inp1[~inp1.response.isnull()]

#calculate the missing values in each column of data frame: inp1.

inp1.isnull().sum()

#describe the pdays column of inp1.

inp1.pdays.describe()

-1 indicates the missing values. Missing value does not always be present as null. How to handle

it:

Objective is:

• you should ignore the missing values in the calculations

• simply make it missing - replace -1 with NaN.

• all summary statistics- mean, median etc. we will ignore the missing values of pdays.

#describe the pdays column with considering the -1 values.

inp1.loc[inp1.pdays<0,"pdays"]=np.NaN

inp1.pdays.describe()

Session- 4, Bivariate and Multivariate Analysis

Segment-2, Numeric- numeric analysis

There are three ways to analyse the numeric- numeric data types simultaneously.

• Scatter plot: describes the pattern that how one variable is varying with other variable.

• Correlation matrix: to describe the linearity of two numeric variables.

• Pair plot: group of scatter plots of all numeric variables in the data frame.

#plot the scatter plot of balance and salary variable in inp1

plt.scatter(inp1.salary, inp1.balance)

plt.show()

#plot the scatter plot of balance and age variable in inp1

inp1.plot.scatter(x='age', y='balance')

plt.show()

#plot the pair plot of salary, balance and age in inp1 dataframe.

sns.pairplot(data=inp1, vars=["salary","balance", "age"])

plt.show()

#plot the correlation matrix of salary, balance and age in inp1

dataframe.

sns.heatmap( inp1[["salary","balance", "age"]].corr(), annot= True,

cmap= "Reds")

plt.show()

Segment- 4, Numerical categorical variable

Salary vs response

#groupby the response to find the mean of the salary with response no

& yes seperatly.

inp1.groupby("response")["salary"].mean()

#groupby the response to find the median of the salary with response

no & yes seperatly.

inp1.groupby("response")["salary"].median()

#plot the box plot of salary for yes & no responses.

sns.boxplot(data=inp1,x="response", y="salary")

plt.show()

#plot the box plot of balance for yes & no responses.

sns.boxplot(data=inp1,x="response", y="balance")

plt.show()

#groupby the response to find the mean of the balance with response no

& yes seperatly.

inp1.groupby("response")["balance"].mean()

#groupby the response to find the median of the balance with response

no & yes seperatly.

inp1.groupby("response")["balance"].median()

#function to find the 75th percentile.

def p75(x):

return np.quantile(x, 0.75)

#calculate the mean, median and 75th percentile of balance with

response

inp1.groupby("response")["balance"].aggregate(["mean","median",p75])

#plot the bar graph of balance's mean an median with response.

inp1.groupby("response")

["balance"].aggregate(["mean","median"]).plot.bar()

plt.show()

Education vs salary

#groupby the education to find the mean of the salary education

category.

inp1.groupby("education")["salary"].mean()

#groupby the education to find the median of the salary for each

education category.

inp1.groupby("education")["salary"].median()

Job vs salary

#groupby the job to find the mean of the salary for each job category.

inp1.groupby('job')['salary'].mean()

inp1.groupby('job')['salary'].median()

Segment- 5, Categorical categorical variable

#create response\_flag of numerical data type where response "yes"= 1,

"no"= 0

inp1["response\_flag"]=np.where(inp1.response=="yes", 1, 0)

inp1.response.value\_counts()

inp1.response.value\_counts(normalize= True)

inp1.response\_flag.mean()

Education vs response rate

#calculate the mean of response\_flag with different education

categories.

inp1.groupby("education")["response\_flag"].mean()

Marital vs response rate

#calculate the mean of response\_flag with different marital status

categories.

inp1.groupby(["marital"])["response\_flag"].mean()

#plot the bar graph of marital status with average value of

response\_flag

inp1.groupby(["marital"])["response\_flag"].mean().plot.barh()

plt.show()

Loans vs response rate

#plot the bar graph of personal loan status with average value of

response\_flag

inp1.groupby(["loan"])["response\_flag"].mean().plot.bar()

plt.show()

Housing loans vs response rate

#plot the bar graph of housing loan status with average value of

response\_flag

inp1.groupby(["housing"])["response\_flag"].mean().plot.bar()

plt.show()

Age vs response

#plot the boxplot of age with response\_flag

sns.boxplot(data=inp1, x="response",y="age")

plt.show()

#plot the bar graph of job categories with response\_flag mean value.

inp1.groupby(['job'])['response\_flag'].mean().plot.barh()

plt.show()

* 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**

NA