APPLIED DATA SCIENCE 1

ASSIGNMENT 1: VISUALISATION

STUDENT NAME: GAYATRI BALIVADA

STUDENT ID: 22093609

GITHUB : [https://github.com/Gayatribalivada/ADS1-Assignment-1.git]

Table of Contents

1. Introduction

In order to better understand and communicate patterns and trends in the data, data visualisations play a key role in the area of data analysis. This study explores the Python Spyder environment for making line plots, scatter plots, and bar graphs, three of the most often-used forms of charts. Based on its use in visually representing patterns over time or over a succession of data points, line plots find widespread application in the analysis of time series and the display of continuous data (Su *et al*. 2020). These data visualisation plots are often used to depict interrelationships across data sets, reveal patterns, or assess the dynamics of a statistic over time. Scatter plots, on the other hand, are useful visualisations for analysing the connections between two variables since they reveal patterns and outliers. These graphs allow one to visually inspect the direction and magnitude of correlations between variables. Bar charts excel at comparing categories and displaying the distribution of data. Bar charts are a simple and effective method of visually depicting and comparing data from many different types of categories or sets.

2. Line plots

It has been seen that in the Python Spyder environment, line graphs are a crucial feature of data visualisation. A line plot is a popular choice for the analysis of time series data and the representation of continuous information because of its main aim of graphically illustrating the continual movement of a variable. Data analysts, scientists, as well as other investigators get a deeper understanding of the evolution of a variable during a certain time period by using this visualisation method.

2.1 Description

The line plot comes with graphing lines that are straightforward and explain their behaviour clearly. The independent variable is shown along the x-axis, while the dependent variable, which is anything quantitative, is shown along the y-axis. Each data point in the collection has a representation in which the x and y axes intersect (Su *et al* 2022). It has been seen that a line, following the sequence of the dataset, joins these points together. The line between data points displays data trends and oscillations. Data analysts develop relevant conclusions and well-informed assessments by taking the temporal features of data into account.

2.2 Methods

It has been noted that the line plots are developed by using the Spyder software application with the help of the def function. In this section, a methodology to illustrate the diagram has been performed. A dataset has been selected from a government data resource link and the name of the dataset is "***electronic-file-series-2004-2013.csv***". The dataset has been prepared by cleaning the null values and converting categorical values into numbers (Frandsen *et al*. 2022).

```
#using def function
def line_plot(dataframe):
    plt.figure(figsize=(10, 8))
    plt.plot(dataframe['UsagePeriod'], dataframe['CurrentOwner'], color='red')
    plt.xlabel('UsagePeriod')
    plt.ylabel('CurrentOwner')
    plt.title('Line Plot')
    plt.legend(title='relationship between variables')
    plt.show()

line_plot(df)


def create_line_plot(dataframe, x_col, y_col, title, x_lab, y_lab, legend):
    plt.figure(figsize=(10, 8))
    plt.plot(dataframe[x_col], dataframe[y_col], color='brown')
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    plt.title(title)
    plt.legend(title=legend)
    plt.show()

create_line_plot(df, 'Prefix', 'FileOffice', 'Line Plot', 'Prefix', 'FileOffice', 'relationship between variables')
```

Figure 1: Line plot using def function

(Source: Developed using Spyder)

The two line plots at the top of the page have been created employing the Spyder IDE along with the Matplotlib package of the programming language Python. In both cases, the passed-in data frame (df) is used to generate a line plot displaying the correlation between two specified variables (Lanari *et al*. 2020). The first method, line_plot, has a broader scope compared to the second option. The function accepts the whole data frame as input and generates a plot comparing the 'UsagePeriod' field with the 'CurrentOwner' variable. The association between the factors is designated, and the name and legend setting are established. The plot can be observed via the use of the Plt. show() function. The create_line_plot function offers more flexibility as it allows for the inclusion of parameters specifying both the y and x columns, the title of the plot, the labels for the axes, and the title of the legend. This approach has a propensity for accommodating a broader spectrum of data types and offers novel opportunities for customisation.

2.3 Implementation

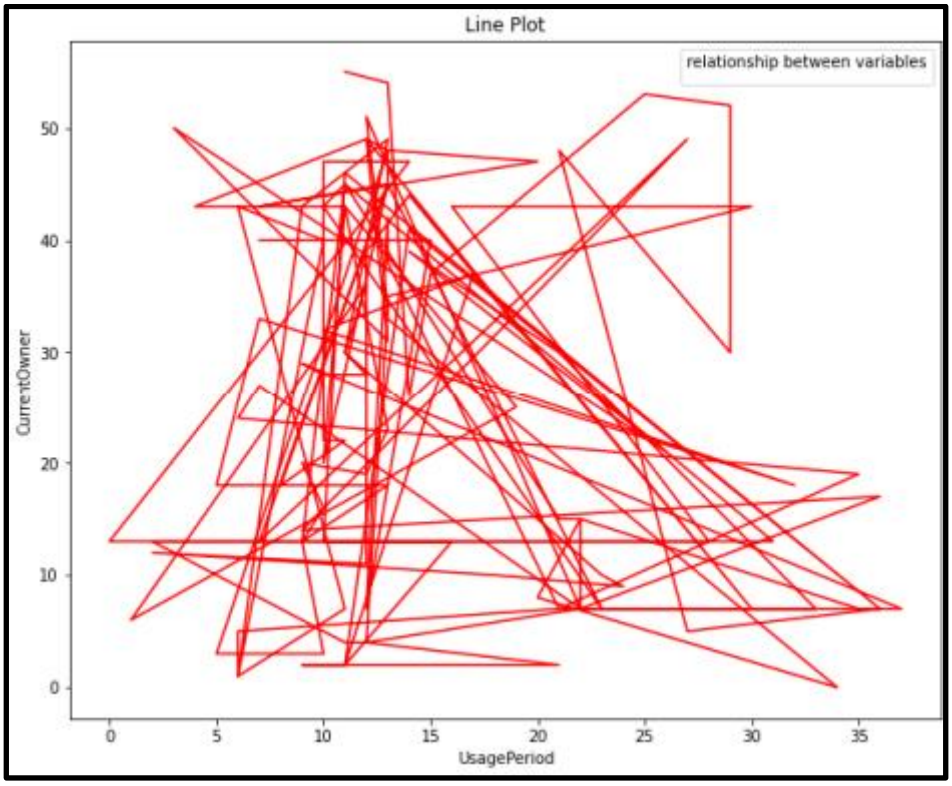

Figure 2: Evaluation of a line plot

(Source: Designed using Spyder)

It is observed that a line chart has been successfully created by applying the def function and required Python coding.

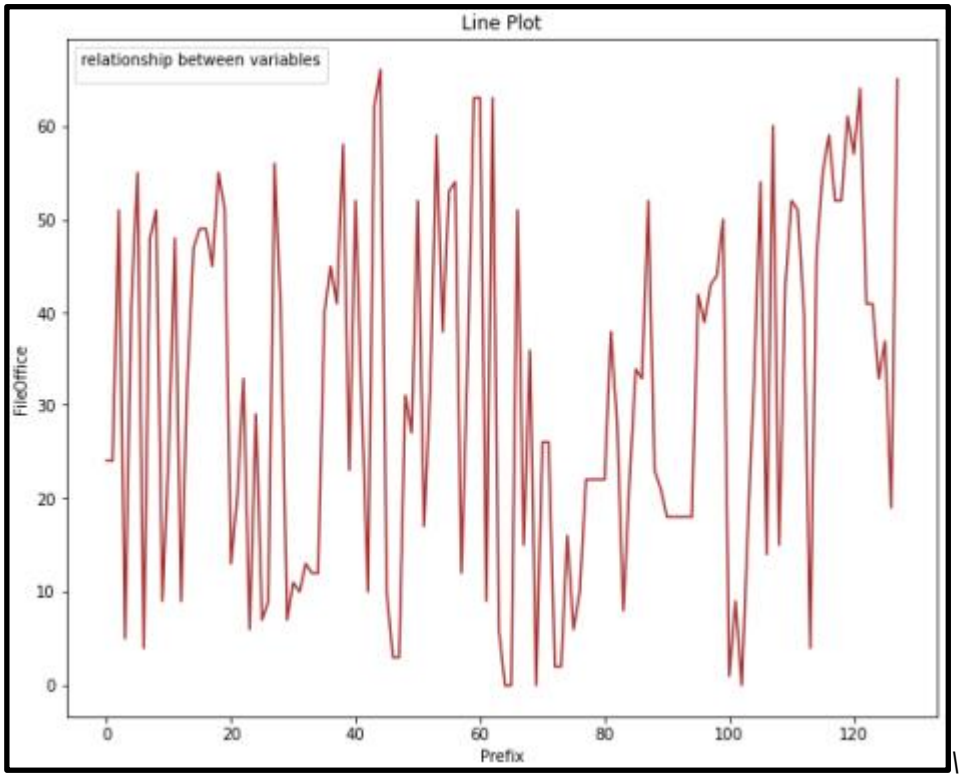

Figure 3: Design a line chart

(Source: Designed using Spyder)

Another line plot has been plotted to execute the relationship between two particular data attributes from the dataset.

3. Scatter plots

As a flexible and easy-to-use data visualisation tool, scatter plots in Python have been used to reveal and make sense of the relationships between several variables. Scatter plots are widely recognised as a standard tool for revealing trends, identifying groups of data points, pinpointing outliers, and gauging the strength and direction of associations between points in a dataset. In order to graphically display the dispersion and association between two numerical variables is the primary goal of a scatter plot. It is common practice to plot one variable along an x-axis and another along a y-axis. As a result, scientists, researchers, and data analysts can quickly assess how shifting values in one variable affect other variables.

3.1 Explanation

Scatter plots have intuitive mechanisms that are easy to understand. Every data point is represented as a dot on the graph, with the centre of the dot located at the intersection of the x and y axes. When plotted on a graph, these data points reveal hidden relationships and correlations within the data set. Clusters of data points create a noticeable aggregation of data values, whereas outliers stand out since they don't fit in with the main grouping (Song *et al* 2023). It has been seen that the degree and direction of the connection can be inferred from the data points' placement in a visual depiction. In a similar way, a positive correlation is commonly shown by a diagonal pattern running from the bottom left to the upper right, whereas a negative correlation is often depicted in the opposite manner. For the sake of data analysis and visualisation, scatter plots have been easily created and modified in Python with the help of tools such as Matplotlib (Sivapriya *et al*. 2019). In order to better understand the connection between two variables and to draw meaningful conclusions from the data, scatter plots are often regarded as vital data analysis tools.

3.2 Analysis

In this part, a procedure to design the scatter plot has been described with the help of Python programming language.

```
#using def function
def scatter_plot_(data, x_lab, y_lab, title, color='purple'):
    plt.figure(figsize=(10, 8))
    plt.scatter(data[x_lab], data[y_lab], color=color)
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    plt.title(title)
    plt.show()

scatter_plot_(df, 'NumberOfDocumentsHeld', 'BusinessFunctionCovered', 'Scatter Plot')

def scatter_plot(x, y, x_lab, y_lab, title):
    plt.figure(figsize=(10, 8))
    plt.scatter(x, y, color='blue')
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    plt.title(title)
    plt.show()

x_data = df['CurrentOwner']
y_data = df['NumberOfFilesHeld']
x_lab = 'CurrentOwner'
y_lab = 'NumberOfFilesHeld'
plot = 'Scatter Plot'

scatter_plot(x_data, y_data, x_lab, y_lab, plot)
```

Figure 4: Scatter plot using def function

(Source: Developed using Spyder)

The def function takes an input Data and returns the axis labels (x, y) with the heading as arguments. The default hue of the points that scatter is purple, however, this may be changed if desired. Through the use of the plt.scatter operation, a scatter plot can be created with further modification for the labelling and designation of the plot. In addition to the mandatory pair of labels and the title, the function also takes a DataFrame, represented by df, as an argument (Chrysantina and Sæbø, 2019). The current operation might be thought of as a refined and modernised version of the one that came before it. The function expects two lists as inputs, one containing the x values and the other containing the y data. A title and axis labels are also required. In addition, the scattered data points are shaded with a uniform blue hue. The x and y organise, together with labels and a title, are required as input parameters for producing a scatter plot.

3.3 Implementation

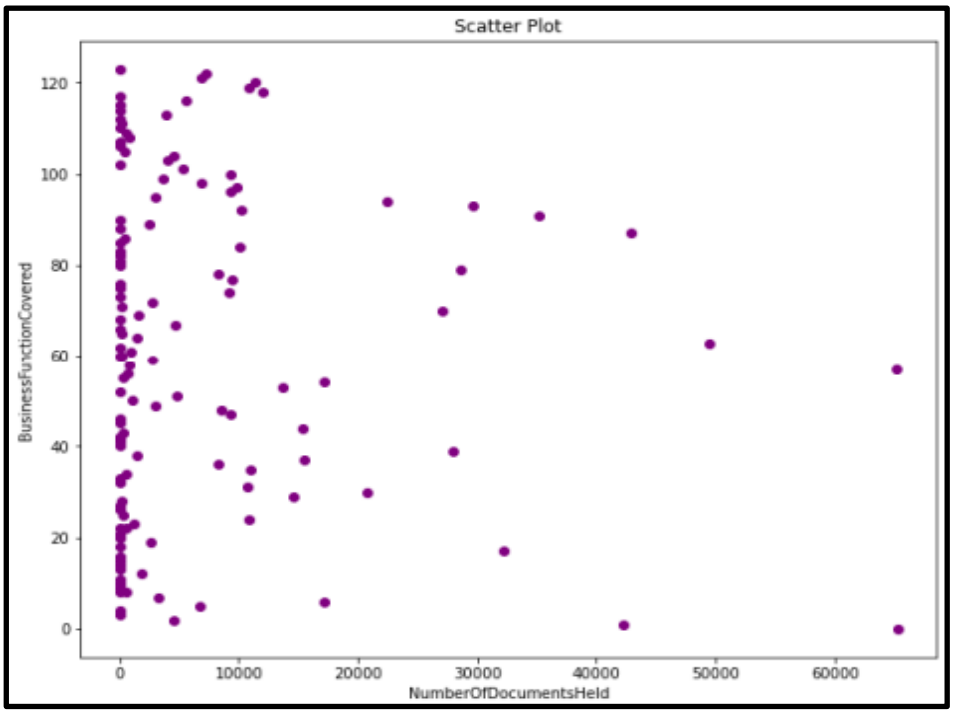

Figure 5: A scatter diagram to correlate variables

(Source: Designed using Spyder)

A scatter diagram has been illustrated by employing the data visualisation methods in Spyder.

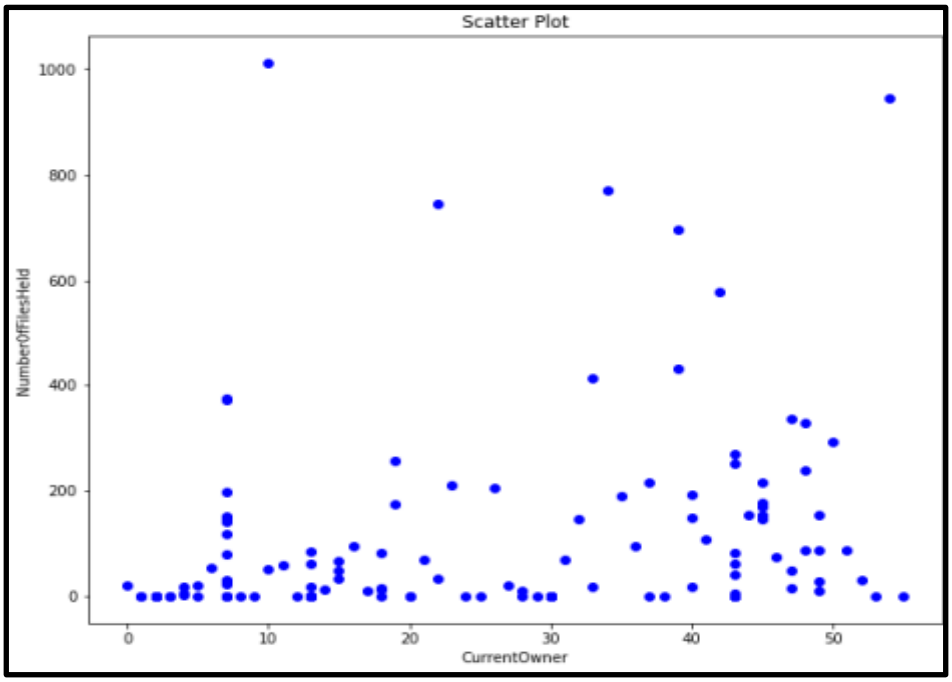

Figure 6: A scatter diagram to observe a correlation between data points

(Source: Designed using Spyder)

Another similar scatter diagram is made using the def function to identify a link between data points.

4. Bar plot

In the environment of Python, bar charts are an integral part of data visualisation because of the crucial role they play in depicting comparisons and distributions when dealing with categorical or discrete data. These plots are crucial for graphically showing the relative sizes of individual data points, and their usage has been shown to be very useful in efficiently conveying data relevant to distinct categories or groups. A bar plot's primary use is to provide an easily digestible visual representation of differences and relationships between distinct subsets of a dataset. It has been seen that data analysts, researchers, and decision-makers use this tool to easily compare, contrast, as well as rank different groups of data. As a consequence, it is ideal for the display of data relevant to market share, survey findings, or performance indicators throughout a broad variety of industries.

4.1 Description

The structure of the bar plot comes with the x-axis often symbolises the groups or categories being discussed, while the y-axis displays the values or frequencies associated with each group or category. Each category is represented by a bar, the length or height of which corresponds to the appropriate data value for that category (Kůrka *et al*. 2020). The bars' uniform spacing and parallel arrangement make it easy to visually compare many categories at once. Numerous programs in Python, such as Matplotlib, make it simple to create and modify bar graphs in a variety of formats. In order to enhance readability and effectiveness, users change the background colour of the plot, as well as the hue of the bars, labels, and titles. Bar graphs are an essential tool for summarising and showing categorised data, as they allow for the quick identification of trends, patterns, and discrepancies across various subsets of a particular dataset.

4.2 Analysis

```
#Bar plot

#using def function
def plot_bar_chart(data_frame, x_col, y_col, title, color='orange'):
    plt.figure(figsize=(10, 8))
    plt.bar(data_frame[x_col], data_frame[y_col], color=color)
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    plt.title(title)
    plt.show()

plot_bar_chart(df, 'Prefix', 'BusinessFunctionCovered', 'Bar Plot')


def bar_plot(data_frame, x_colm, y_colm, x_lab, y_lab, title, figsize=(10, 8), color='green'):
    plt.figure(figsize=figsize)
    plt.bar(data_frame[x_colm], data_frame[y_colm], color=color)
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    plt.title(title)
    plt.show()

bar_plot(df, 'Prefix', 'NumberOfFilesHeld', 'Prefix', 'NumberOfFilesHeld', 'Bar Plot')
```

Figure 7: Bar plot using def function

(Source: Developed using Spyder)

The pyplot module from the Matplotlib package can be used to create bar graphs in the Spyder environment. One option for encapsulating this work is to write a Python function that, given some data and some other arguments, outputs a bar chart. The title of the plot, axis labels, bar colour, and any other decorative elements are all part of this process (Kanungo and Tulasi, 2019). The opportunity for extending the versatility of these customisation choices resides in the ability to submit them as parameters to the function. Bar plot construction is abstracted into a separate function, allowing it to be used with other data sets and tweaked as needed. As a result, the code becomes more modular and easier to maintain.

The initial function, named plot_bar_chart, accepts a data_frame, both the x and y values for the columns, a description, and a colour parameter to choose the colour of the bars. The first step is to make a graph, name the axes, and show it to the audience. The next method, named bar_plot, is quite similar but provides greater room for customization (Serrano *et al*. 2023). The function allows for the x and y designations, a title, the size of the figure, and the colour of the bars to be set. The data in the data_frame is used to generate a bar chart using either approach.

4.3 Implementation

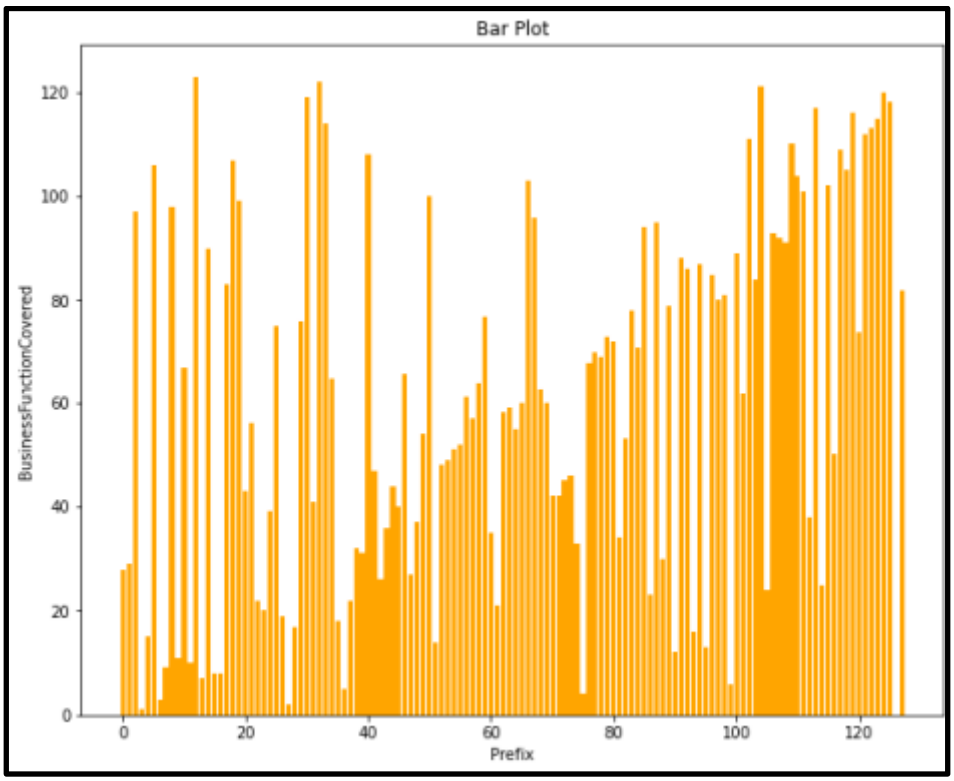

Figure 8: Data representation using a bar chart

(Source: Designed using Spyder)

A bar chart has been illustrated to compare the two particular data entities for the dataset.

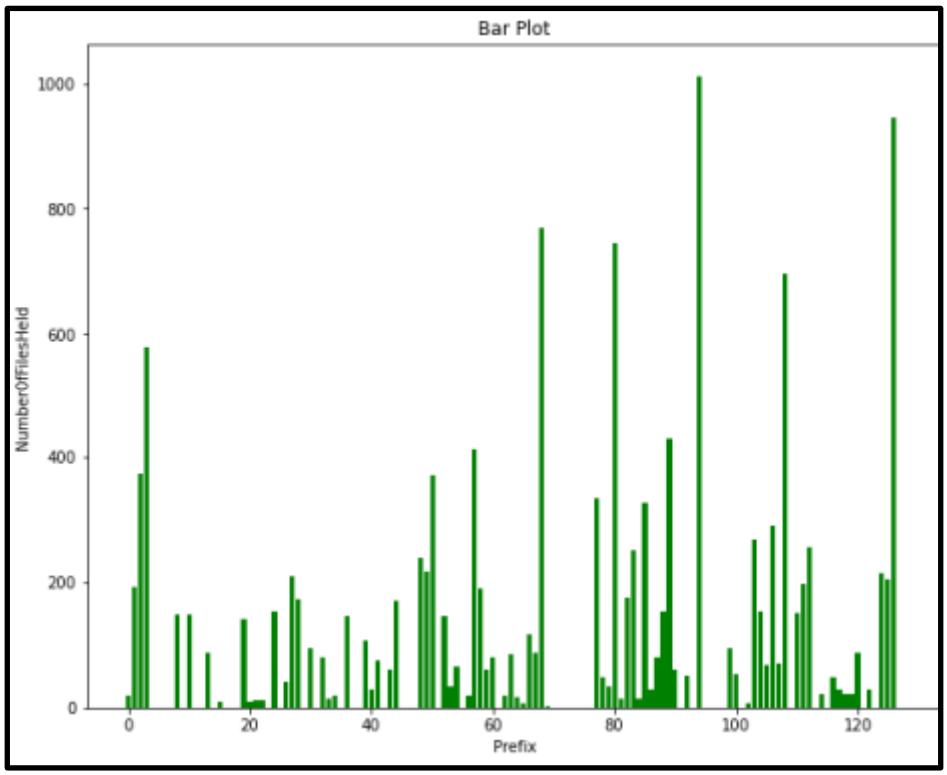

Figure 9: Data visualisation by designing a bar chart

(Source: Designed using Spyder)

Another bar chart is developed in the software application to comprehend the correlation between two numeric attributes in the dataset.

5. Conclusion

Finally, different graphical representations are key tools for comprehending and effectively disseminating insights gained from datasets, making data visualisation an essential aspect of data analysis. In this particular setting, line plots, scatter plots, as as bar graphs are the three most often investigated forms of visual representations. These representations are helpful for showing patterns and fluctuations in data. The Spyder platform for Python and Matplotlib's adaptable methods for generating line graphs allow data analysts to make educated decisions regarding temporal data. There are a number of applications for scatter plots, which help visualise the connections between numerical variables. Data points, represented by dots on a graph, may be visually inspected for trends and outliers and quantitatively analysed for correlation intensity and direction. Tools including Matplotlib make it simple to generate and modify scatter plots in Python for use in statistical analysis. Bar charts are good for presenting the relative importance of different categories owing to their utility in comparing and illustrating category or independent information. Creating and personalising bar graphs for comparing data across categories is simple using Python's Matplotlib package.

References

Chrysantina, A. and Sæbø, J.I., 2019. Assessing user-designed dashboards: A case for developing data visualization competency. In Information and Communication Technologies for Development. Strengthening Southern-Driven Cooperation as a Catalyst for ICT4D: 15th IFIP WG 9.4 International Conference on Social Implications of Computers in Developing Countries, ICT4D 2019, Dar es Salaam, Tanzania, May 1–3, 2019, Proceedings, Part I 15 (pp. 448-459). Springer International Publishing.

Frandsen, P.B., Hotaling, S., Powell, A., Heckenhauer, J.B., Kawahara, A.Y., Baker, R.H., Hayashi, C.Y., Rios-Touma, B., Holzenthal, R., Pauls, S.U. and Stewart, R.J., 2022. Uncovering hidden genetic diversity: allelic resolution of insect and spider silk genes. bioRxiv, pp.2022-12.

Kanungo, V. and Tulasi, B., 2019. Data visualization and toss related analysis of IPL teams and batsmen performances. International Journal of Electrical and Computer Engineering, 9(5), p.4423.

Kůrka, A., Naumova, M., Indzhov, S. and Deltshev, C., 2020. New faunistic and taxonomic data on the spider fauna of Albania (Arachnida: Araneae). Arachnologische Mitteilungen: Arachnology Letters, 59(1), pp.8-21.

Lanari, P. and Piccoli, F., 2020, July. New horizons in quantitative compositional mapping–Analytical conditions and data reduction using XMapTools. In IOP Conference Series: Materials Science and Engineering (Vol. 891, No. 1, p. 012016). IOP Publishing.

Serrano, A., Basante-Bedoya, M.A., Bassilana, M. and Arkowitz, R.A., 2023. A live-cell ergosterol reporter for visualization of the effects of fluconazole on a human fungal pathogen. bioRxiv, pp.2023-09.

Sivapriya, J., Kumar, A., Sai, S.S. and Sriram, S., 2019. Breast cancer prediction using machine learning. International Journal of Recent Technology and Engineering (IJRTE), 8(4), pp.4879-4881.

Song, Y., Zhao, X. and Wong, R.C.W., 2023. Marrying Dialogue Systems with Data Visualization: Interactive Data Visualization Generation from Natural Language Conversations. arXiv preprint arXiv:2307.16013.

Su, I., Hattwick, I., Southworth, C., Ziporyn, E., Bisshop, A., Mühlethaler, R., Saraceno, T. and Buehler, M.J., 2022. Interactive exploration of a hierarchical spider web structure with sound. Journal on Multimodal User Interfaces, pp.1-15.

Su, I., Qin, Z., Saraceno, T., Bisshop, A., Mühlethaler, R., Ziporyn, E. and Buehler, M.J., 2020. Sonification of a 3-D spider web and reconstitution for musical composition using granular synthesis. Computer Music Journal, 44(4), pp.43-59.