Industry Chosen:

Finance and Banking/Credit card datasets

Unit – 4

Part 1: Pandas

Ans-1: Pandas is commonly used in the financial and banking industries for tasks such as financial data analysis, risk management, and fraud detection.

1. **Financial Data Analysis**: Stock prices, market indices, and other financial data, as well as large datasets containing such data, can all be analysed using pandas. Pandas can be used, for instance, to compute returns, carry out statistical analysis, and produce different financial metrics like VaR, beta, and Sharpe ratio.

2. **Credit Risk Analysis**: Pandas can be used to analyse credit risk by combining financial data, credit scores, and customer information. You can use Pandas to merge these datasets, filter customers based on specific criteria, and generate credit risk scores.

3. **Portfolio Optimization**: By examining correlations, historical returns, and other pertinent data, pandas can be used to optimize investments in a portfolio. Pandas can be used to optimize portfolio weights using a variety of algorithms, compute pairwise correlation matrices, and carry out factor analysis.

Ans-2: In the context of Pandas, a Series is a one-dimensional labeled array capable of holding any data type. The axis labels are collectively referred to as the index.

Here is an example of creating a Series:

```
In [1]: import pandas as pd
        import numpy as np
        s = pd.Series([1, 3,4,5, np.nan, 6], index=['p', 'q', 'r', 'd', 'e', 'f'])
        s

Out[1]: p    1.0
        q    3.0
        r    4.0
        d    5.0
        e    NaN
        f    6.0
        dtype: float64
```

A DataFrame, on the other hand, is a two-dimensional size-mutable, tabular data structure with columns of potentially different types. The axis labels for rows are called the index, and the axis labels for columns are called the columns.

Here is an example of creating a DataFrame:

```
In [6]: data = {'Category': ['cat', 'dog', 'bird'],
                'Price($)': [20, 30, 5]}

        df = pd.DataFrame(data)
        df
```

Out[6]:

	Category	Price($)
0	cat	20
1	dog	30
2	bird	5

Many operations that are common to both Series and DataFrame include indexing, selection, filtering, aggregation, and merging.

The application of DataFrame and Series in the context of financial data analysis differs according to the data under study.

A Series would be a good way to display the data, for instance, when examining the past performance of a single asset. The values would be the asset's returns, and the index would be the dates.

However, a DataFrame would be more appropriate when analysing multiple assets, like a portfolio of stocks, with the columns representing different assets and the rows representing different time periods.

In conclusion, the specifics of the data being analysed and the kinds of operations that must be carried out determine whether to use a Series or a DataFrame for data analysis.

Part 2: NumPy

Ans-1: **NumPy**, which stands for Numerical Python, is an open-source Python library that supports enormous, multi-dimensional matrices and arrays. It also offers a sizable number of sophisticated mathematical operations that can be performed on these arrays.

Because NumPy enables developers to efficiently manipulate and perform operations on large data sets—a common requirement in fields like finance and banking—it is indispensable for **scientific computing** and data analysis in Python.

Apart from its fundamental features, NumPy can be effortlessly incorporated with other well-known Python libraries, like Pandas and SciPy, that offer extra instruments for manipulating, analysing, and visualizing data.

Because NumPy in Python provides a consistent and effective interface for working with numerical data, regardless of its size or complexity, its use in scientific computing and **data analysis** has grown in popularity.

For example, NumPy's arrays and matrices provide a powerful and flexible way to represent and manipulate complex data structures, such as **stock price data, market trends, or financial instruments**. Additionally, NumPy's extensive mathematical functions make it easy to perform complex operations on this data, such as linear algebra computations, statistical analysis, or numerical integration.

In conclusion, NumPy plays a crucial role in scientific computing and data analysis in Python, as it provides a fundamental building block for working with numerical data in a high-performance and efficient manner.

Ans-2: NumPy's effectiveness and performance are crucial when managing big datasets and performing numerical operations in Python. As a C-extension library, NumPy is written in C and uses calls to Python.

NumPy's high performance can be attributed in large part to its parallel array operations, which make use of all available CPU cores. Because of this, NumPy can handle massive volumes of data effectively, even on multi-core computers.

NumPy's performance is also influenced by its effective C loops and internal memory management in addition to parallelism. NumPy reduces the overhead of using Python's built-in data structures, making computation more effective and speedier.

Here are some key performance features of NumPy:

- Optimized C loops for basic operations, resulting in significant speed-ups.
- Utilization of vectorized operations, which leverage the full computational power of CPUs.

- Compatibility with hardware accelerators, such as GPUs, allowing for further speed-ups.
- Integration with libraries like BLAS and LAPACK, which provide high-performance linear algebra routines.

By offering these advanced features, NumPy ensures optimal performance and efficiency when dealing with large datasets and complex numerical computations in Python, making it an indispensable tool for professionals in the finance and banking industry.

Unit – 5

Data Visualization:

Ans-1: Matplotlib bar plot

```
In [8]: import matplotlib.pyplot as plt
        import numpy as np
        product_names = ['Credit Cards', 'Debit Cards', 'Loans', 'Investments']
        sales_figures = [2000, 1500, 1000, 500]
        bar_plot = plt.bar(product_names, sales_figures, color='c', width=0.4, align='center')
        plt.xlabel('Products')
        plt.ylabel('Sales (units)')
        plt.title('Monthly Sales of Banking Products')
        plt.show()
```

Ans-2: Here are three examples in banking and finance where Seaborn excels over Matplotlib:

1. Visualizing Loan Default Correlations with Pair Plots:

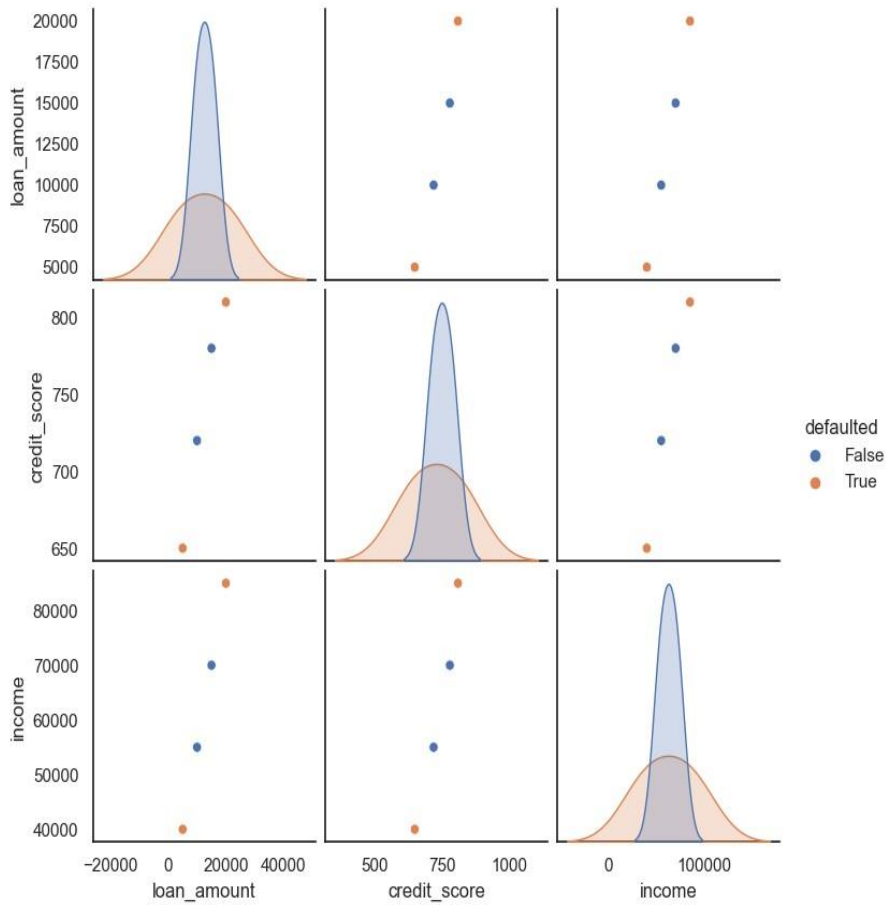
- Plot Type: **Pair plot**
- Why Seaborn:

- Simplifies creation of pairwise relationships for multiple variables.
- Automatically calculates summary statistics and applies informative color palettes.
- Seamlessly handles categorical variables with hue argument for easy group comparisons.

```
In [13]: import seaborn as sns
         import pandas as pd
         loan_data = pd.DataFrame({
             'loan_amount': [5000, 10000, 15000, 20000],
             'credit_score': [650, 720, 780, 810],
             'income': [40000, 55000, 70000, 85000],
             'defaulted': [True, False, False, True]
         })
         sns.pairplot(loan_data, hue='defaulted')
```

```
C:\Users\Pranav Malik\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

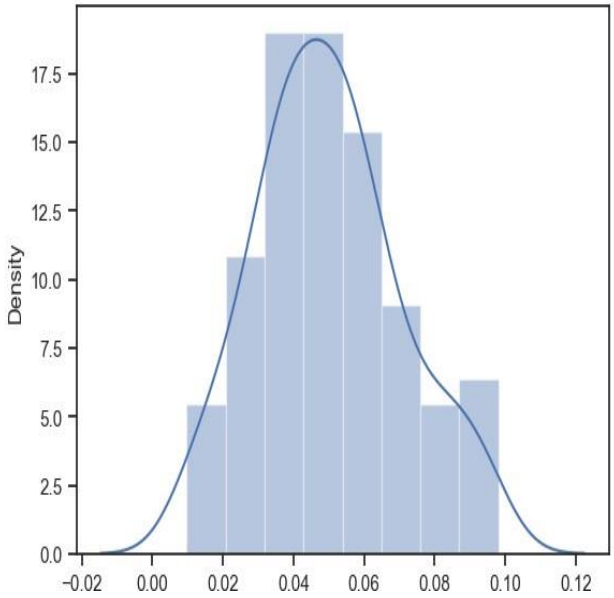
```
Out[13]: <seaborn.axisgrid.PairGrid at 0x208de135490>
```



2. Analysing Portfolio Distributions with Histograms and KDE Plots:

- Plot Types: **Histograms and KDE plots**
- Why Seaborn:
    - Produces visually appealing and informative distributions with default aesthetics.
    - Facilitates comparison of multiple distributions with **distplot** for overlapping histograms and KDE plots.

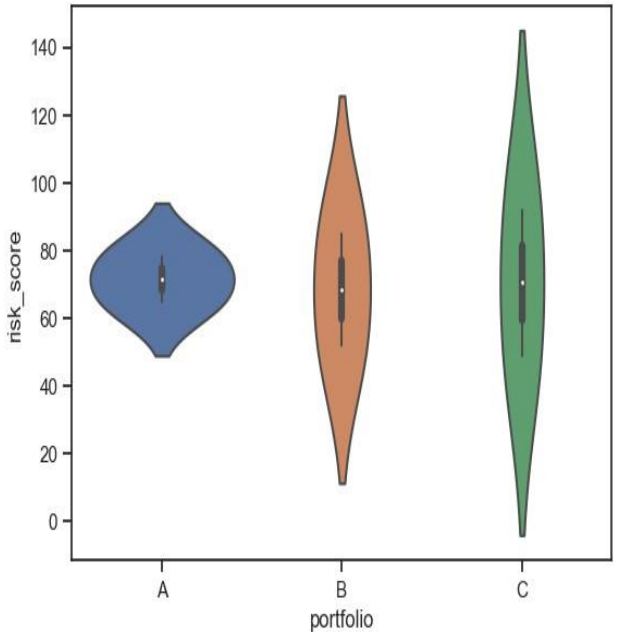
```
In [15]: import numpy as np
         returns = np.random.normal(loc=0.05, scale=0.02, size=100)
         sns.distplot(returns)

C:\Users\Pranav Malik\AppData\Local\Temp\ipykernel_15204\3676426025.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(returns)

Out[15]: <Axes: ylabel='Density'>
```



3. Comparing Risk Profiles of Different Loan Portfolios:

- Plot Type: **Violin plot**
- Why Seaborn:
    - Effectively visualizes distributions of continuous variables across multiple groups.
    - Clearly shows both central tendencies and shape of distributions for informative comparisons.
    - Easily handles categorical variables for grouping.

```
In [16]: import seaborn as sns
         import pandas as pd
         loan_portfolios = pd.DataFrame({
             'portfolio': ['A', 'A', 'B', 'B', 'C', 'C'],
             'risk_score': [65, 78, 52, 85, 49, 92]
         })
         sns.violinplot(x='portfolio', y='risk_score', data=loan_portfolios)

Out[16]: <Axes: xlabel='portfolio', ylabel='risk_score'>
```



Unit – 6

Describe the three key structures in Plotly:

Ans-1: Figure:

A figure in a visualization represents the entire graphic. It contains all the elements that make up the visualization, such as the axes, title, and legend. In the context of Seaborn, the figure serves as the canvas for displaying the data.

Data:

The data in a visualization represents the numerical values that are being plotted on the axes. In the case of Seaborn, this could be a Pandas dataframe or a NumPy array.

For example, in the context of analysing credit card usage and balances, the data could be the total credit card balance and the amount spent on the credit card.

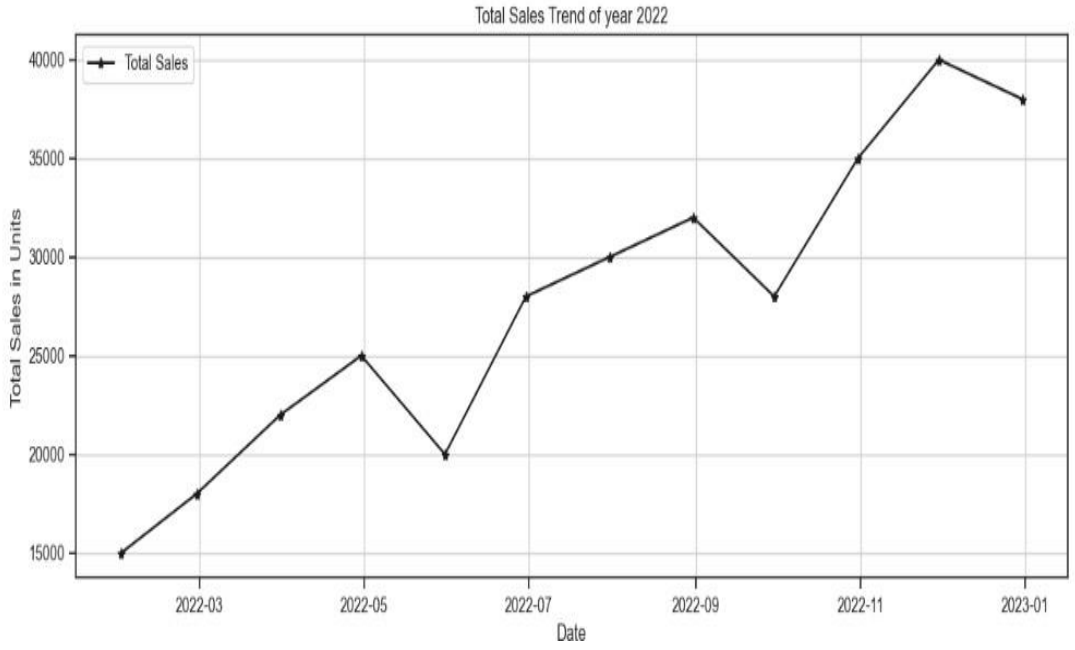
Layout:

The layout of a visualization refers to the organization and arrangement of elements within the figure. In Seaborn, this includes aspects such as the color palette, marker style, and the position of labels.

By understanding and considering these structures when creating visualizations, we can ensure that our data is effectively represented and interpreted by the intended audience.

Ans-2:

```
In [36]: import pandas as pd
         import matplotlib.pyplot as plt
         data = {'Date': pd.date_range(start='2022-01-01', end='2022-12-31', freq='M'),
                 'Sales': [15000, 18000, 22000, 25000, 20000, 28000, 30000, 32000, 28000, 35000, 40000, 38000]}
         salesdf = pd.DataFrame(data)
         plt.figure(figsize=(15, 5))
         plt.plot(salesdf['Date'], salesdf['Sales'], marker='*', linestyle='-', color='k', label='Total Sales')
         plt.title('Total Sales Trend of year 2022')
         plt.xlabel('Date')
         plt.ylabel('Total Sales in Units')
         plt.legend()
         plt.grid(True)
         plt.show()
```



Interpretation:

1. Sales are increasing steadily: The graph shows a consistent upward trend from March 2022 to January 2023. This suggests that sales are increasing over time, with no significant dips or plateaus.

2. Growth may be accelerating: The slope of the line appears to be slightly steeper in the later months of 2022 compared to the earlier months.