

IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Module Code: PUSL 3121	Module Name: Big Data Analysis
Coursework Title: PUSL 3121 Big Data Analysis – Group Work	
Deadline Date: 28/ 03/ 2025	
Programme: BSc.(Hons) in Data science	
Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook .	
<p>10899179 - Pathirathnage Attygala 10899185 - Halgaha Nimeka 10899187 - Kasthuri Prabodya 10899193 - K L Subawickrama 10899472 - Bandari Gnanodya 10900341 - Samarathunga Samarathunga</p> <p><i>We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.</i></p> <p>Signed on behalf of the group:</p> <p style="text-align: right;">Project Leader</p>	
<p>Overall mark%Assessors Initials Date</p>	

Content

1.Introduction to Big Data Processing.....	3
1.2 Advantages of Spark for large-scale data processing.....	5
1.3 The Role of Snowflake in Cloud-Based Data Warehousing and Analytics	6
2. Data Preprocessing.....	9
2.1 Dataset Overview.....	9
2.2 Demonstrate code and explanations of the preprocessing steps.	11
2.3 Comparing Spark’s Performance with Pandas	15
3.Demensionality Reduction	16
3.1 Principal Component Analysis	16
3.2 Explanation of Loadings in Principal Component Analysis (PCA).....	19
Cloud Analytics	22
Uploading a Dataset to AWS S3	22
Loading a Dataset from AWS S3 into Google Colab.....	23
4. Realtime Dashboard Insights.	24
4.1Dashboard insights through power BI	28
5. Predictive Model Analytics.....	33
5.1 Model Performance Analysis & Justification.....	37
6. Conclusion.....	39
References.....	40

Figure 1: Apache Spark Ecosystem - Complete Spark Components Guide - DataFlair ... 3

Figure 2:Snowflake Architecture 4

1.Introduction to Big Data Processing.

Apache Spark and Snowflake are two prominent platforms in the field of data processing and analytics, each offering unique capabilities tailored to modern data challenges.

Apache Spark: Unified Analytics Engine

Apache Spark is an open-source, unified analytics engine designed for large-scale data processing. It provides high-level APIs in Java, Scala, Python, and R, facilitating versatility for developers. Spark's core is built upon an optimized engine that supports general execution graphs, enabling efficient data processing across various workloads. Additionally, Spark includes a rich set of higher-level tools:

- **Spark SQL:** For SQL and structured data processing.
- **pandas API on Spark:** To handle pandas workloads.
- **MLlib:** For machine learning applications.
- **GraphX:** For graph processing.
- **Structured Streaming:** For incremental computation and stream processing.

These components make Spark a versatile platform capable of addressing a wide array of data processing needs.

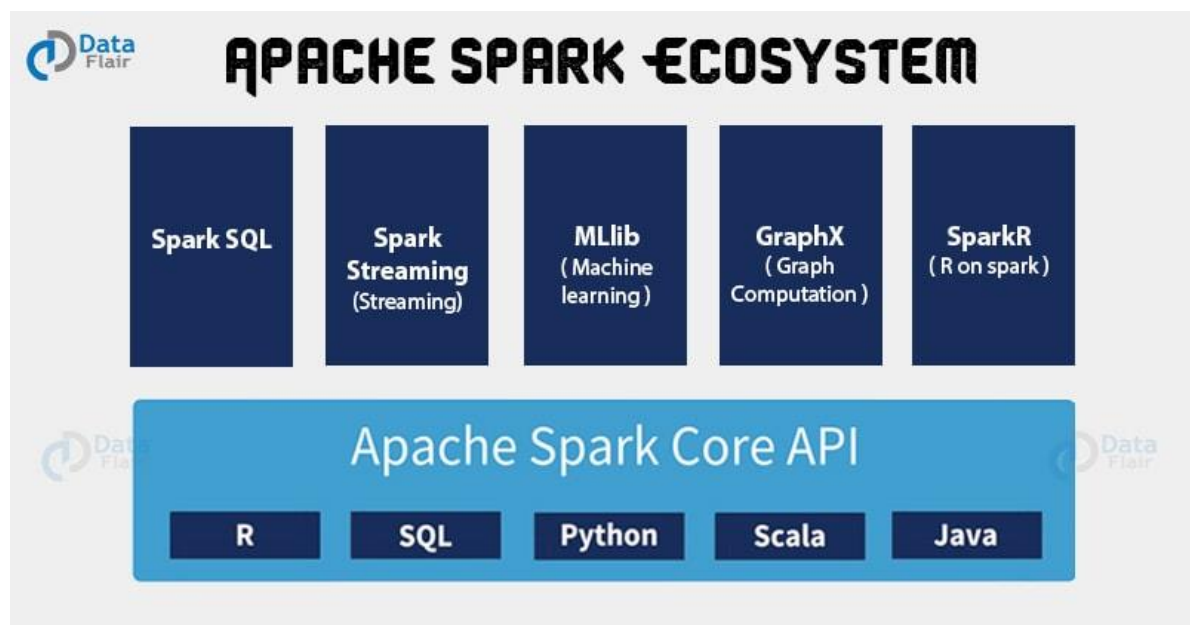


Figure 1: [Apache Spark Ecosystem - Complete Spark Components Guide - DataFlair](#)

Snowflake: Cloud-Native Data Platform

Snowflake is a cloud-native data platform that offers data storage, processing, and analytic solutions. It is designed to be faster, easier to use, and more flexible than traditional offerings. Snowflake's architecture is not built on existing database technology or big data software platforms like Hadoop. Instead, it features a unique design that includes:

- **Separation of Storage and Compute:** Allowing independent scaling and efficient resource management.
- **Support for Structured and Semi-Structured Data:** Enabling versatile data handling capabilities.
- **Data Sharing and Collaboration:** Facilitating secure data sharing across organizations and platforms.

These features position Snowflake as a robust solution for modern data warehousing and analytics needs.

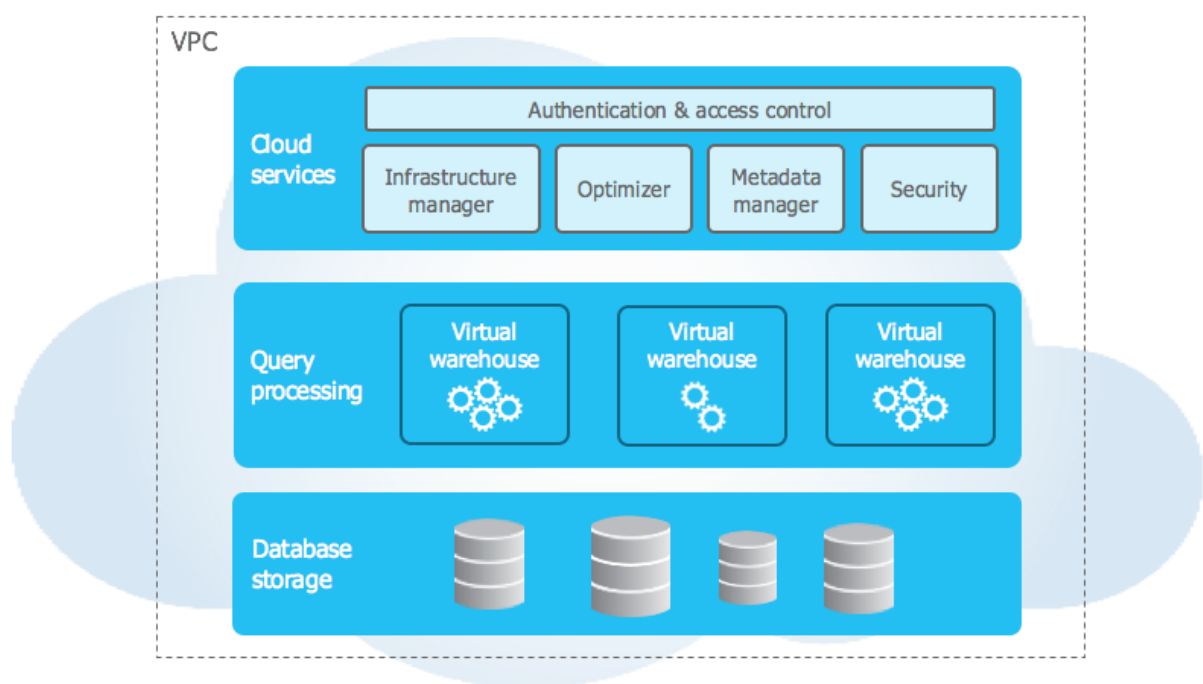


Figure 2: Snowflake Architecture

1.2 Advantages of Spark for large-scale data processing.

Apache Spark offers several advantages for large-scale data processing, making it a preferred choice for handling big data workloads. Below are some key benefits:

1. High-Speed Performance

- **In-Memory Computing:** Spark processes data in memory rather than writing intermediate results to disk, significantly improving speed (up to 100x faster than Hadoop MapReduce for in-memory operations).
- **Optimized Execution Engine:** Spark's Directed Acyclic Graph (DAG) scheduler and advanced query optimization enhance execution speed.

2. Ease of Use

- **Multiple Language Support:** Provides APIs for Python (PySpark), Scala, Java, and R, making it accessible to a broad range of developers.
- **Simple and Expressive APIs:** Features high-level APIs for SQL queries (Spark SQL), streaming data processing (Structured Streaming), machine learning (MLlib), and graph processing (GraphX).

3. Scalability and Distributed Computing

- **Cluster Computing:** Spark runs on large-scale distributed systems, allowing it to process petabytes of data efficiently.
- **Cloud and On-Premise Deployment:** Compatible with cloud providers like AWS, Azure, and Google Cloud, as well as on-premise clusters.

4. Versatility and Unified Data Processing

- **Batch and Real-Time Processing:** Unlike Hadoop, Spark supports both batch and real-time data processing via Structured Streaming.
- **Machine Learning & AI Integration:** Includes MLlib for scalable machine learning algorithms, making it ideal for AI-driven applications.
- **Graph Processing:** GraphX supports graph-based analytics, expanding Spark's utility.

5. Fault Tolerance and Reliability

- **Resilient Distributed Dataset (RDD):** Ensures fault tolerance by recomputing lost data partitions automatically.
- **Checkpointing & Data Lineage:** Maintains lineage information, allowing the system to recover from failures efficiently.

6. Cost Efficiency

- **Optimized Resource Utilization:** Dynamically allocates resources to tasks, reducing infrastructure costs.
- **Integration with Cost-Effective Storage Solutions:** Works with Hadoop Distributed File System (HDFS), Amazon S3, and other cloud storage options, minimizing storage costs.

7. Open Source and Strong Community Support

- **Continuous Improvements:** As an open-source project under Apache, Spark benefits from ongoing enhancements.
- **Large Developer Community:** Extensive documentation, tutorials, and forums make problem-solving easier.

1.3 The Role of Snowflake in Cloud-Based Data Warehousing and Analytics

Snowflake is a **fully managed, cloud-native data platform** designed to handle large-scale data warehousing and analytics. Its **unique architecture** and **scalability** make it a top choice for organizations looking for an efficient, cost-effective, and high-performance solution for storing, processing, and analyzing data.

1. Cloud-Native Architecture

Unlike traditional on-premise databases or Hadoop-based systems, Snowflake was **built exclusively for the cloud** and is available on AWS, Azure, and Google Cloud.

- **No Infrastructure Management** – Snowflake eliminates the need for organizations to manage hardware, storage, or complex configurations.
- **Elastic Scalability** – Users can dynamically scale compute and storage resources up or down based on demand, optimizing performance and cost.

2. Separation of Compute and Storage

One of Snowflake's biggest innovations is its architecture, which separates **compute, storage, and cloud services**:

- **Storage Layer:** Stores structured and semi-structured data efficiently.
- **Compute Layer:** Uses virtual warehouses to run queries independently, ensuring workload isolation and scalability.

- **Cloud Services Layer:** Manages authentication, metadata, query optimization, and access control.

This separation allows **multiple workloads to run concurrently** without competing for resources, improving **performance and efficiency**.

3. Performance and Speed

- **Automatic Query Optimization** – Snowflake optimizes queries dynamically without requiring manual tuning.
- **Massively Parallel Processing (MPP)** – Uses distributed computing to process queries faster than traditional databases.
- **Instantaneous Elastic Scaling** – Compute clusters can resize instantly to handle peak workloads and scale down when demand decreases.

4. Multi-Cloud and Cross-Region Capabilities

- Snowflake operates **natively on multiple cloud providers**, allowing businesses to avoid cloud vendor lock-in.
- **Cross-region and cross-cloud data sharing** enable businesses to collaborate seamlessly.

5. Advanced Data Sharing and Collaboration

- **Secure Data Sharing** – Organizations can share live data with partners and customers without copying or moving it.
- **Marketplace and Exchange** – Snowflake's Data Marketplace allows access to third-party datasets for enhanced analytics.

6. Support for Structured and Semi-Structured Data

- Snowflake **natively supports semi-structured data formats** like JSON, Avro, ORC, Parquet, and XML.
- **Automatic schema detection** and **flattening** of semi-structured data make it easier to query and analyze.

7. Built-in Security and Compliance

Snowflake meets enterprise-grade security standards, including:

- **End-to-End Encryption** (both in transit and at rest)
- **Role-Based Access Control (RBAC)**

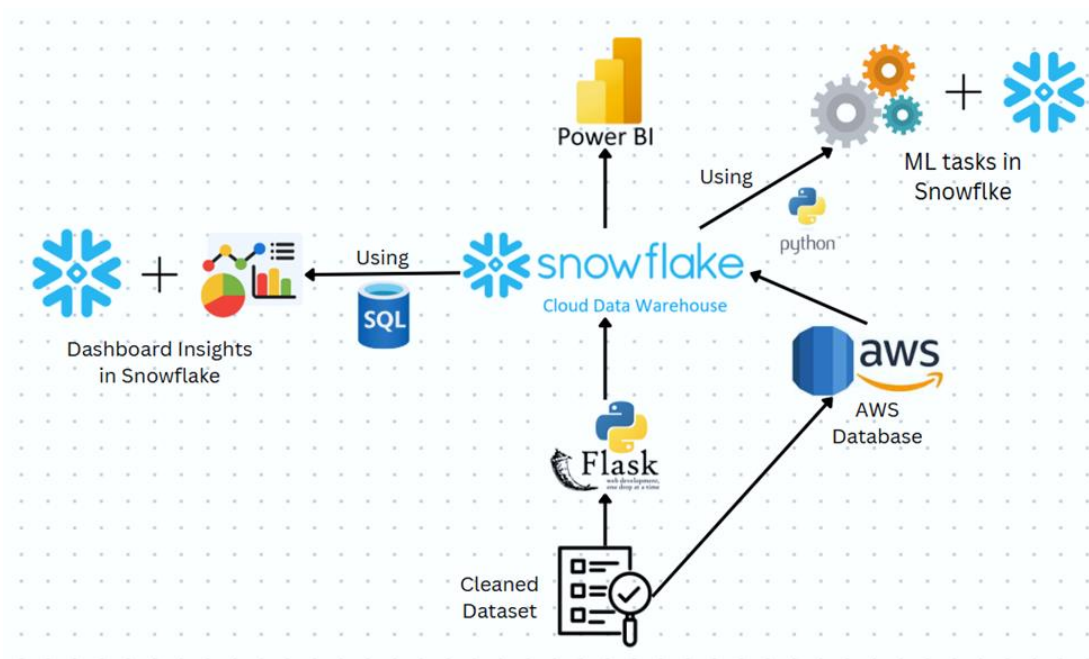
- **GDPR, HIPAA, SOC 2, and PCI Compliance**

8. Cost Efficiency

- **Pay-as-You-Go Pricing Model** – Users pay only for the compute and storage they use, reducing infrastructure costs.
- **Auto-Suspend and Auto-Resume** – Virtual warehouses automatically pause when idle, preventing unnecessary charges.

9. AI and Machine Learning Integration

- Snowflake integrates with **Python, TensorFlow, Spark, and other ML frameworks** for advanced analytics.
- **Snowpark** allows users to build AI/ML models directly within Snowflake using Python.



2. Data Preprocessing

2.1 Dataset Overview

Direct marketing campaigns are a widely used strategy in the banking industry to attract potential clients for financial products. The dataset analyzed in this report pertains to the direct marketing campaigns of a Portuguese banking institution. These campaigns involved phone calls to clients, with some clients being contacted multiple times to assess their likelihood of subscribing to a term deposit.

The primary objective of this dataset is to predict whether a client will subscribe to a term deposit based on various features. This is a binary classification problem where the target variable (y) can take two values: '**yes**' (the client subscribes) or '**no**' (the client does not subscribe).

This dataset is based on "Bank Marketing" UCI dataset (please check the description at: <http://archive.ics.uci.edu/ml/datasets/Bank+Marketing>)

Features in the Dataset

The " bank-additional-full.csv " file contains 41,188 rows and 21 columns. The dataset includes various client attributes, campaign details, and economic indicators to predict whether a client will subscribe to a term deposit.

- **Number of rows:** 41,188
- **Number of columns:** 21
- **Target variable:** y (client subscribed to term deposit? 'yes' or 'no')
- **No missing values** detected.

Feature Description

1. Client Information:

- a. age (numeric): Client's age.
- b. job (categorical): Type of job (e.g., admin, services).
- c. marital (categorical): Marital status (e.g., married, single).
- d. education (categorical): Level of education.
- e. default (categorical): Has credit in default? ('yes', 'no', 'unknown').
- f. housing (categorical): Has a housing loan? ('yes', 'no', 'unknown').
- g. loan (categorical): Has a personal loan? ('yes', 'no', 'unknown').

2. Last Contact Details:

- a. contact (categorical): Contact communication type (e.g., telephone, cellular).
- b. month (categorical): Last contact month.
- c. day_of_week (categorical): Day of the last contact.

- d. duration (numeric): Last contact duration (in seconds).

3. Campaign Details:

- a. campaign (numeric): Number of contacts performed during this campaign.
- b. pdays (numeric): Days since the client was last contacted (999 means never contacted before).
- c. previous (numeric): Number of times contacted before this campaign.
- d. poutcome (categorical): Outcome of the previous marketing campaign ('success', 'failure', 'nonexistent').

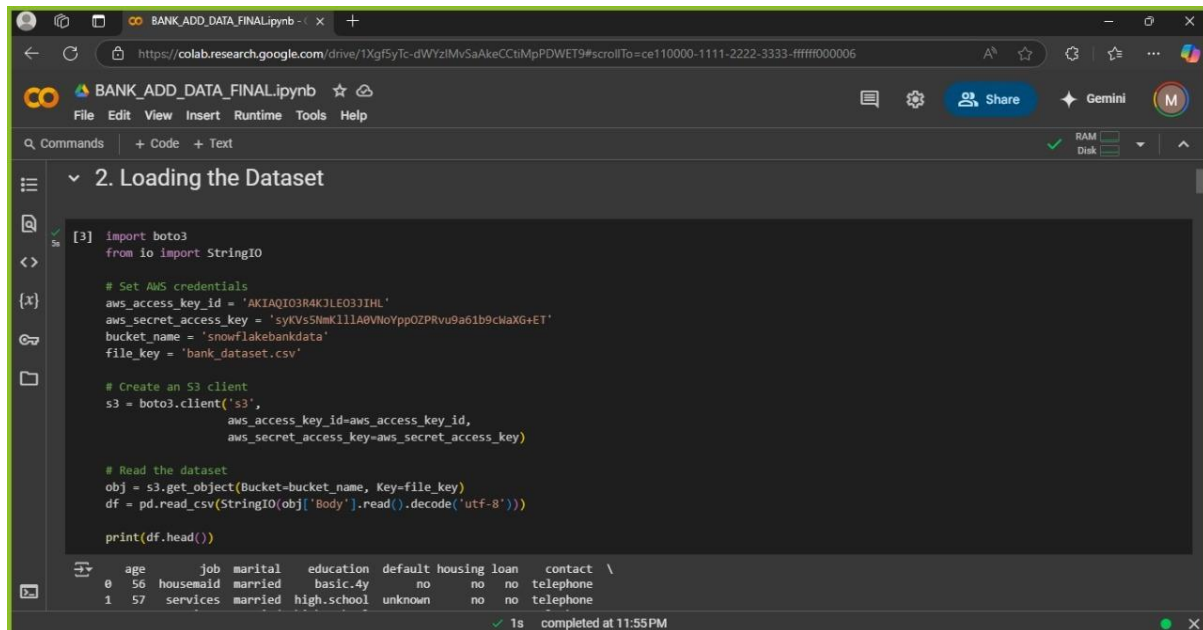
4. Economic Indicators:

- a. emp_var_rate (numeric): Employment variation rate.
- b. cons_price_idx (numeric): Consumer price index.
- c. cons_conf_idx (numeric): Consumer confidence index.
- d. euribor3m (numeric): Euribor 3-month rate.
- e. nr_employed (numeric): Number of employees.

Missing Attribute Values: There are several missing values in some categorical attributes, all coded with the "unknown" label. These missing values can be treated as a possible class label or using deletion or imputation techniques.

2.2 Demonstrate code and explanations of the preprocessing steps.

Data loading



```
[3] import boto3
from io import StringIO

# Set AWS credentials
aws_access_key_id = 'AKIAQIO3R4KJLEO3JIHL'
aws_secret_access_key = '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'
bucket_name = 'snowflakebankdata'
file_key = 'bank_dataset.csv'

# Create an S3 client
s3 = boto3.client('s3',
                  aws_access_key_id=aws_access_key_id,
                  aws_secret_access_key=aws_secret_access_key)

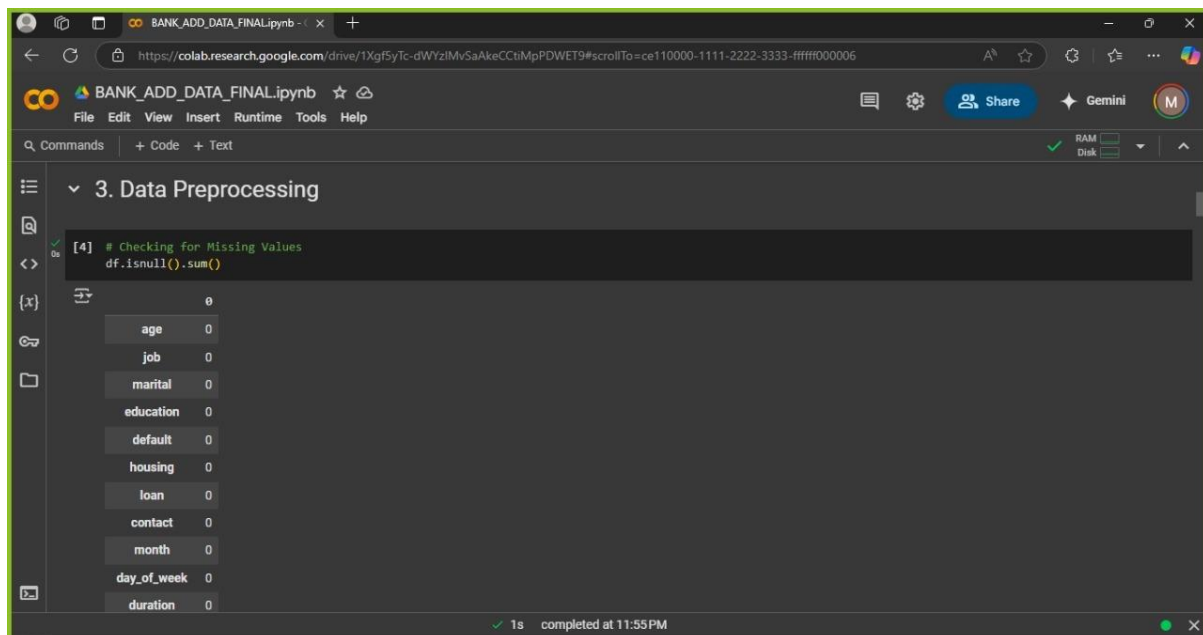
# Read the dataset
obj = s3.get_object(Bucket=bucket_name, Key=file_key)
df = pd.read_csv(StringIO(obj['Body'].read().decode('utf-8')))

print(df.head())
```

	age	job	marital	education	default	housing	loan	contact	
0	56	housemaid	married	basic.4y	no	no	no	telephone	
1	57	services	married	high.school	unknown	no	no	telephone	

completed at 11:55PM

Cleaning transformation

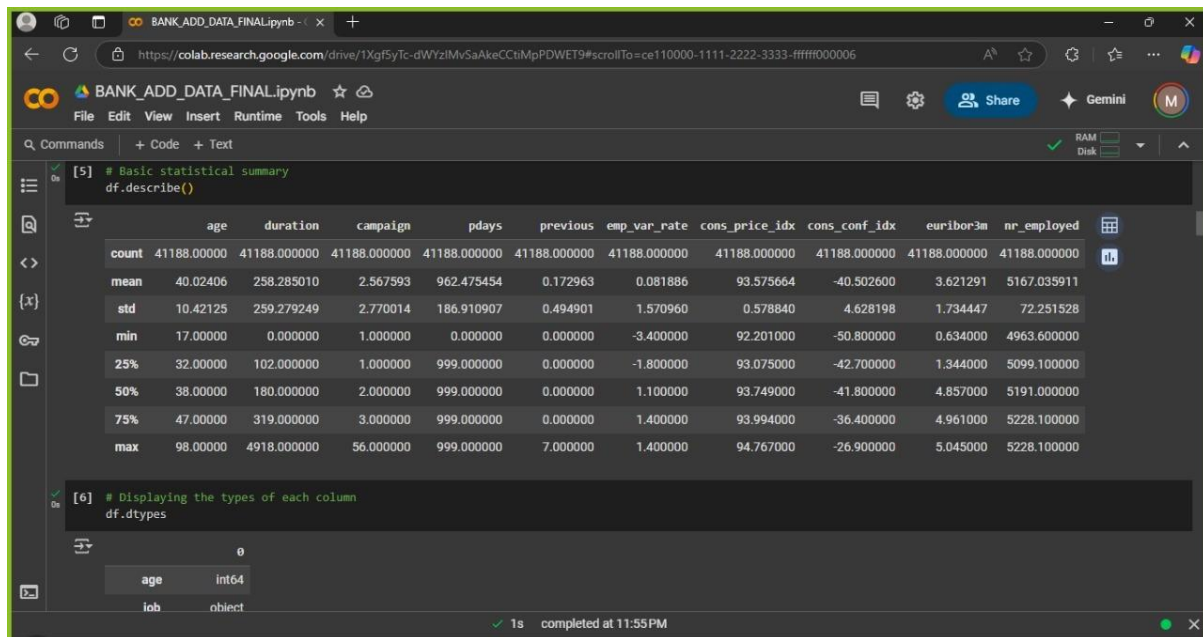


```
[4] # Checking for Missing Values
df.isnull().sum()
```

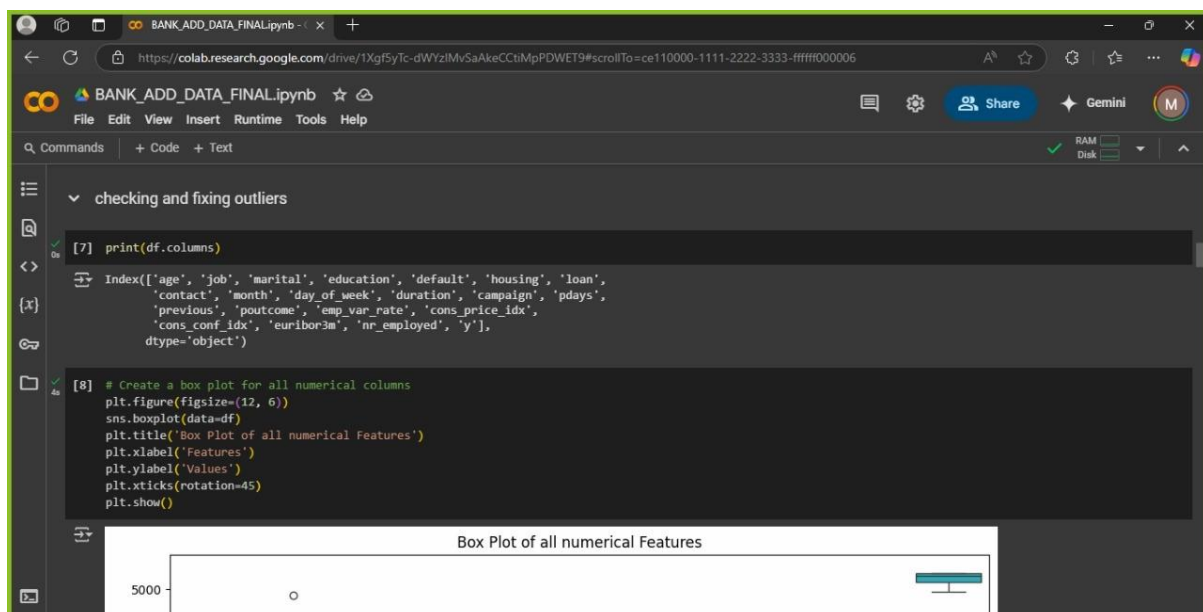
	0
age	0
job	0
marital	0
education	0
default	0
housing	0
loan	0
contact	0
month	0
day_of_week	0
duration	0

completed at 11:55PM

Identifying missing values is crucial as they can negatively impact the performance of machine learning models. It helps decide on an appropriate strategy to handle them (e.g., imputation or removal)



Understanding the statistical distribution of data helps identify potential issues like outliers or skewed data, which might need further preprocessing and Knowing the data types is essential for selecting appropriate preprocessing techniques. For example, categorical features may need encoding before being used in machine learning models.



- This line is likely used during the data exploration or preprocessing phase to understand the structure of dataset.
- It provides a quick way to see the names of all the variables (columns) available in the DataFrame df.

Encoding Catogorical

```
BANK_ADD_DATA_FINAL.ipynb
https://colab.research.google.com/drive/1XgF5yTc-dWYzIMvSaAkeCCtMpPDWET9#scrollTo=ce110000-1111-2222-3333-fffff000006

BANK_ADD_DATA_FINAL.ipynb
File Edit View Insert Runtime Tools Help

Commands + Code + Text

Random Forest

[20] from sklearn.ensemble import RandomForestClassifier

[21] #selecting feature/ variables from the dataset for the models

#encoding categorical variables
df['job'] = df['job'].astype('category').cat.codes
df['marital'] = df['marital'].astype('category').cat.codes
df['education'] = df['education'].astype('category').cat.codes
df['housing'] = df['housing'].astype('category').cat.codes
df['loan'] = df['loan'].astype('category').cat.codes
df['poutcome'] = df['poutcome'].astype('category').cat.codes

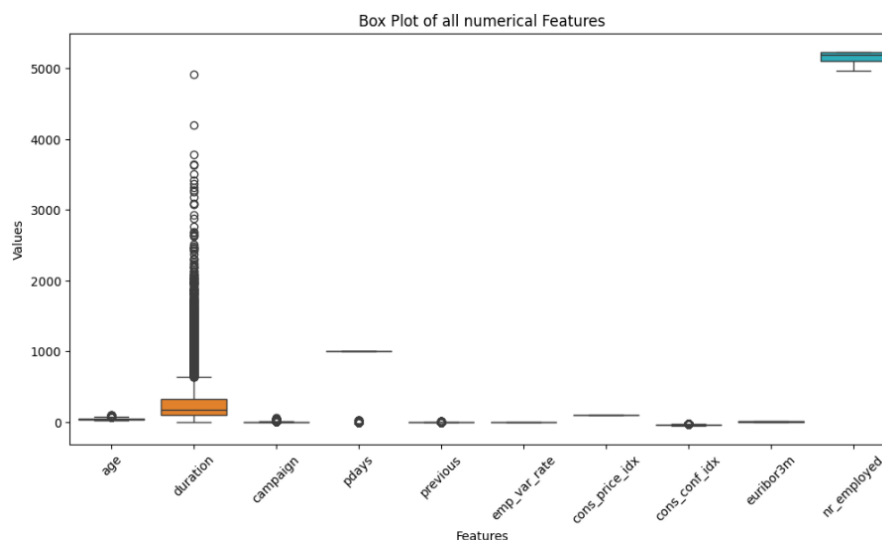
#encoding Target variable
df['y'] = df['y'].astype('category').cat.codes

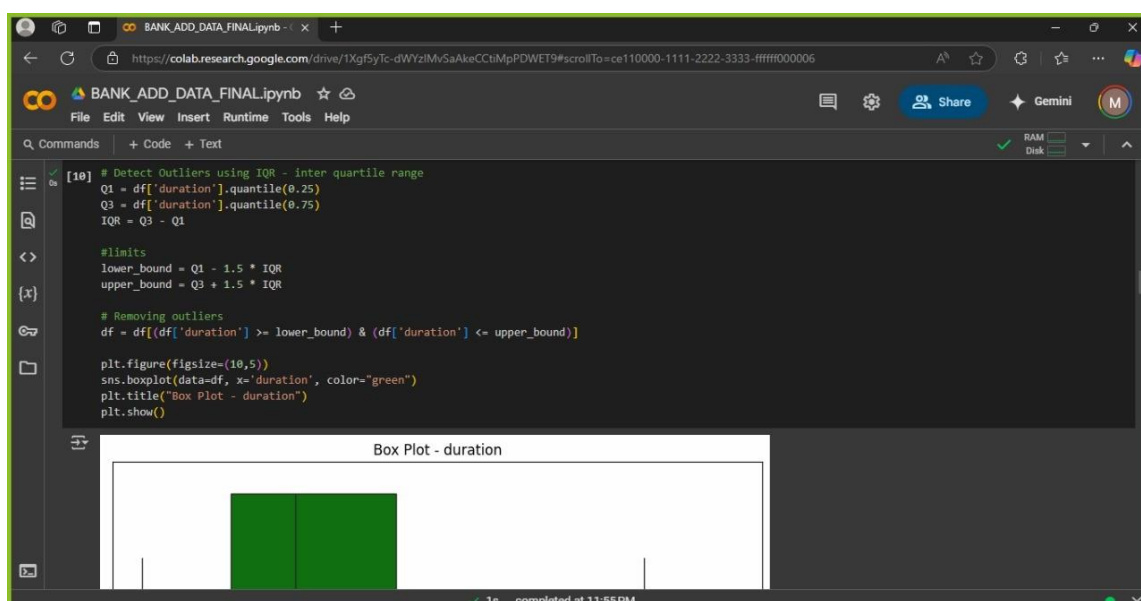
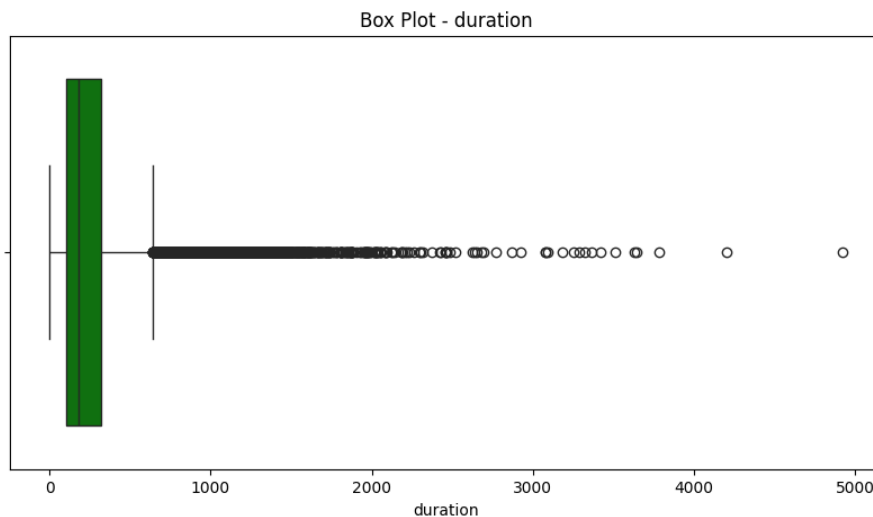
#features and target variable
features = ['age', 'pdays', 'previous', 'cons_conf_idx', 'job', 'marital', 'education', 'housing', 'loan', 'poutcome', 'duration', 'campaign', 'emp_var_rate',
target = ['y']
x = df[features]
y = df[target]

[22] #split the dataset
```

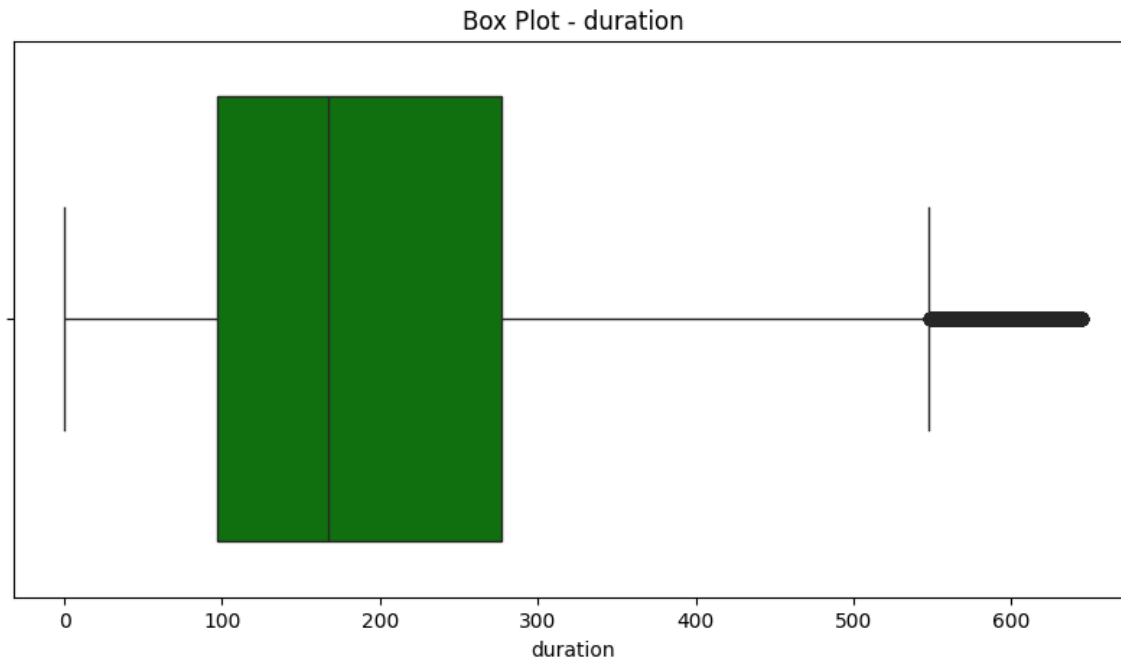
This code converts categorical variables in a dataset into numerical values using `.astype('category').cat.codes`, ensuring they can be used in machine learning models. It encodes features like 'JOB', 'MARITAL', 'EDUCATION', 'HOUSING', 'LOAN', and 'POUTCOME', as well as the target variable 'Y', by assigning unique numeric codes to each category for further analysis.

Checking and fixing outliers





This code is **detecting and removing outliers** from the "duration" column using the **Interquartile Range (IQR) method** and then plotting a **box plot** to visualize the cleaned data.



2.3 Comparing Spark's Performance with Pandas

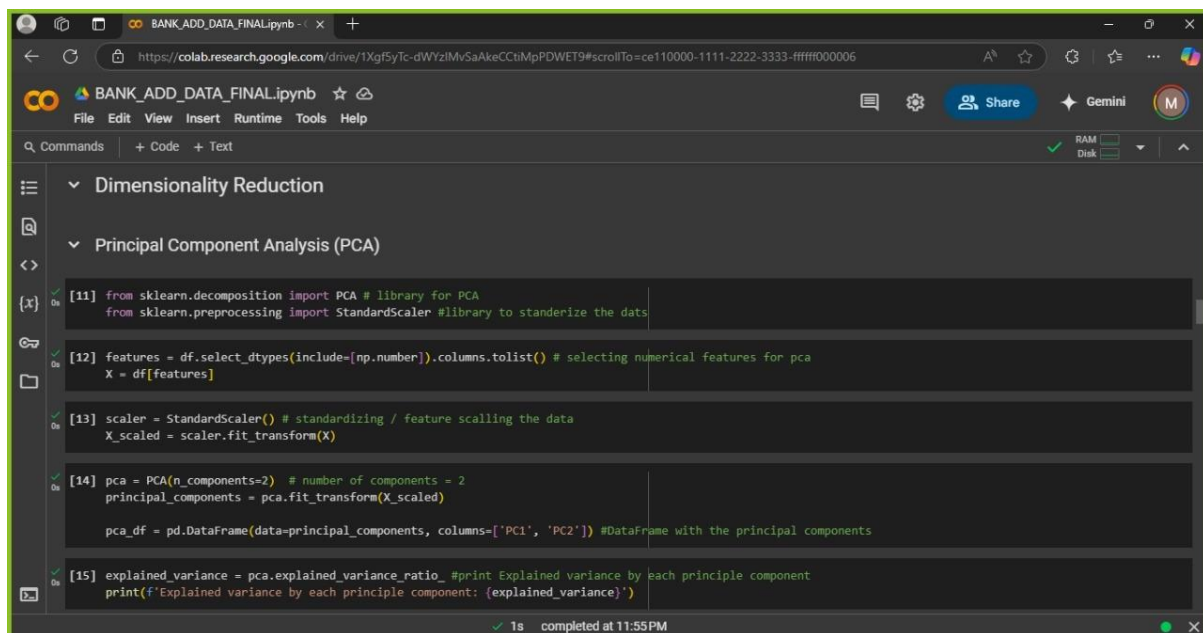
Apache Spark and Python's Pandas are both widely used for data processing, but they differ significantly in performance, scalability, and use cases.

Feature	Apache Spark	Pandas
Data Handling	Distributed computing for large datasets (terabytes)	Single-node processing (limited by RAM)
Speed & Scalability	Faster for big data due to parallel execution across clusters	Efficient for small-to-medium datasets but slows down with large data
Memory Management	Uses lazy evaluation and optimized execution plans to manage memory efficiently	Loads entire dataset into memory, leading to high RAM usage
Use Case	Best for big data, distributed analytics, real-time processing	Suitable for small-to-medium structured datasets
Parallelism	Runs on multiple nodes in a cluster (distributed computing)	Runs on a single machine (multi-threading but no true parallelism)
Ease of Use	Requires setup (Spark cluster, PySpark)	Easy to use with simple Python syntax
Integration	Works well with Hadoop, HDFS, and cloud platforms	Works well with in-memory analytics and ML libraries like Scikit-learn

3. Dimensionality Reduction

3.1 Principal Component Analysis

PCA (Principal Component Analysis) was applied only to the numerical features in the dataset, as it works with continuous data. Categorical variables were excluded from PCA and retained in their original form during this step to prevent misleading transformations. After performing PCA on numerical data, categorical variables were encoded using One-Hot Encoding or Label Encoding. (Gewers *et al.*, 2021) Finally, the encoded categorical features were combined with the PCA-transformed numerical features to create the final dataset for model training.

A screenshot of a Google Colab notebook titled 'BANK_ADD_DATA_FINAL.ipynb'. The notebook is open to a cell containing Python code for Principal Component Analysis (PCA). The code is organized into five numbered steps: [11] imports PCA and StandardScaler from sklearn; [12] selects numerical features from a DataFrame; [13] creates a StandardScaler and fits it to the data; [14] creates a PCA object with 2 components and fits it to the scaled data; [15] prints the explained variance ratio for each principal component. The notebook interface shows the file explorer on the left with folders for 'Dimensionality Reduction' and 'Principal Component Analysis (PCA)'. The bottom status bar indicates the code was completed at 11:55 PM.

```
[11] from sklearn.decomposition import PCA # library for PCA
from sklearn.preprocessing import StandardScaler #library to standerize the dats

[12] features = df.select_dtypes(include=[np.number]).columns.tolist() # selecting numerical features for pca
X = df[features]

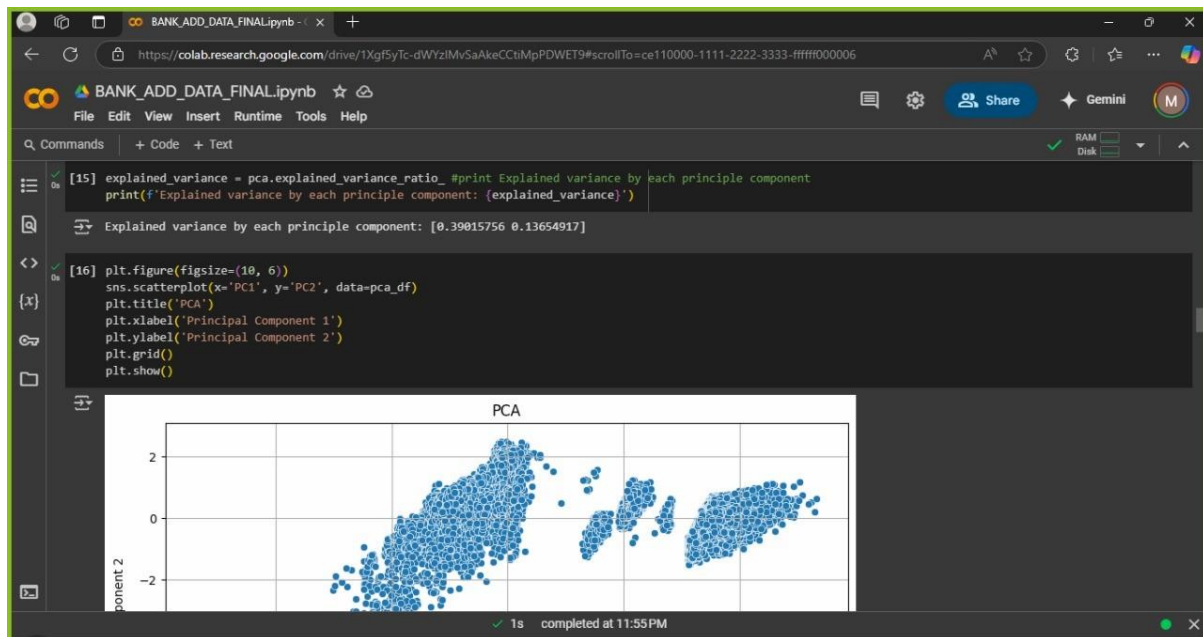
[13] scaler = StandardScaler() # standardizing / feature scalling the data
X_scaled = scaler.fit_transform(X)

[14] pca = PCA(n_components=2) # number of components = 2
principal_components = pca.fit_transform(X_scaled)

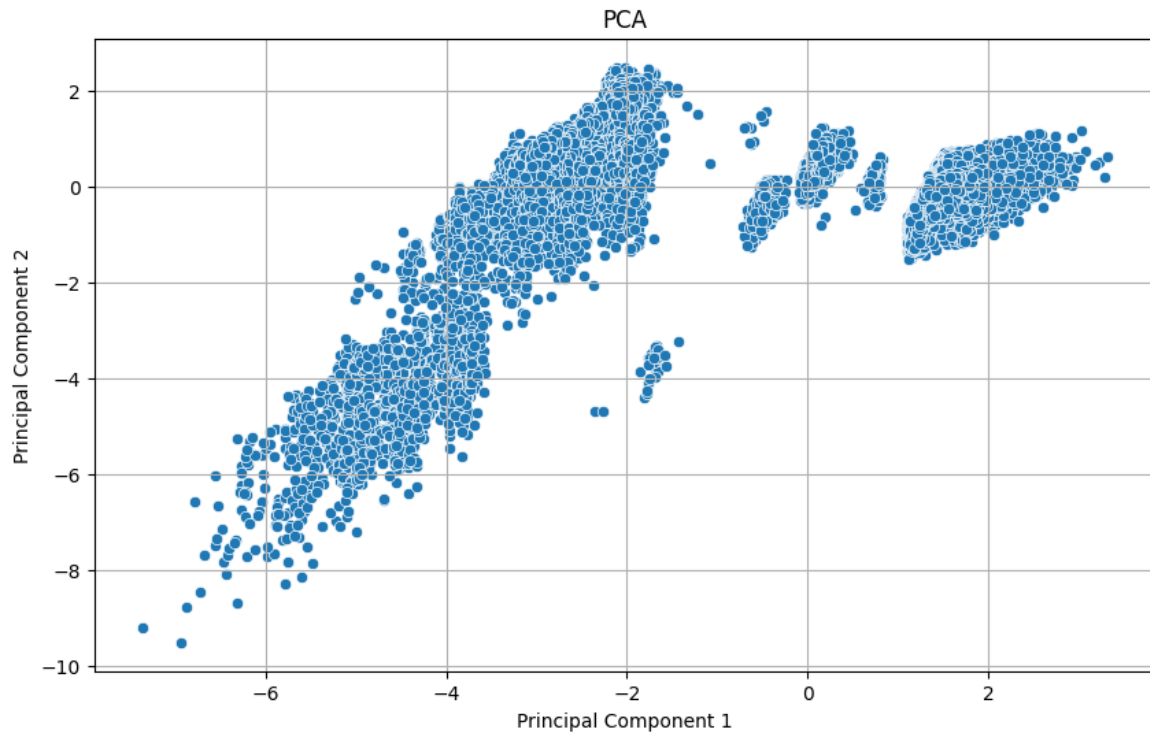
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2']) #DataFrame with the principal components

[15] explained_variance = pca.explained_variance_ratio_ #print Explained variance by each principle component
print(f'Explained variance by each principle component: {explained_variance}')
```

- **Select Numerical Features** – Only numerical features are chosen for PCA, as it requires continuous data.
- **Apply StandardScaler** – The numerical data is standardized to have a mean of 0 and a standard deviation of 1, ensuring that features with different scales do not dominate the analysis.
- **Perform PCA** – The PCA class from scikit-learn is used to reduce dimensionality while preserving important variance in the data



1. **Initialize PCA** – PCA is set with `n_components=2`, meaning the data will be reduced to two principal components while preserving as much variance as possible.
2. **Apply fit_transform** – The standardized numerical data is passed through `fit_transform()`, which computes the principal components and projects the data into a new lower-dimensional space.
3. **Create a DataFrame** – A new DataFrame (`pca_df`) is created to store the transformed data, representing the dataset in terms of the two principal components.
4. **Analyze Explained Variance** – The `explained_variance_ratio_` attribute provides the proportion of total variance captured by each principal component. This helps evaluate how much information is retained after dimensionality reduction.



```

[17] loadings = pca.components_.T * np.sqrt(pca.explained_variance_) #the relationships between the original features and the principal components
      loading_df = pd.DataFrame(loadings, index=features, columns=['PC1', 'PC2'])
      print(loading_df)

```

	PC1	PC2
age	0.000422	-0.298763
duration	-0.104107	-0.198993
campaign	0.200657	0.054612
pdays	0.445520	0.722470
previous	-0.602773	-0.535341
emp_var_rate	0.962833	-0.191209
cons_price_idx	0.723360	-0.322120
cons_conf_idx	0.200784	-0.503800
euribor3m	0.967618	-0.175582
nr_employed	0.928362	0.012979

- Loading
 - Loadings show how much each original variable provides to each primary component.
 - A higher absolute value for a loading indicates that the variable has a greater influence on the major component.

PC1	PC2
emp.var.rate : 0.963183	pdays : 0.732906
cons.price.idx : 0.722576	previous : -0.553083
euribor3m : 0.967870	cons.conf.idx : -0.498544

1s completed at 11:55PM

3.2 Explanation of Loadings in Principal Component Analysis (PCA)

Loadings represent how much each original variable contributes to the principal components (PCs). A higher absolute value of a loading indicates that a variable has a stronger influence on the respective principal component. (Jolliffe and Cadima, 2016)

Principal Component 1 (PC1):

- Variables with high positive loadings:
 - **emp.var.rate (0.963)**
 - **cons.price.idx (0.723)**
 - **euribor3m (0.968)**
 - **nr.employed (0.928)**

These variables strongly contribute to PC1. This suggests that PC1 primarily captures variations related to economic indicators (employment rate, consumer price index, interest rates, and number of employed individuals).

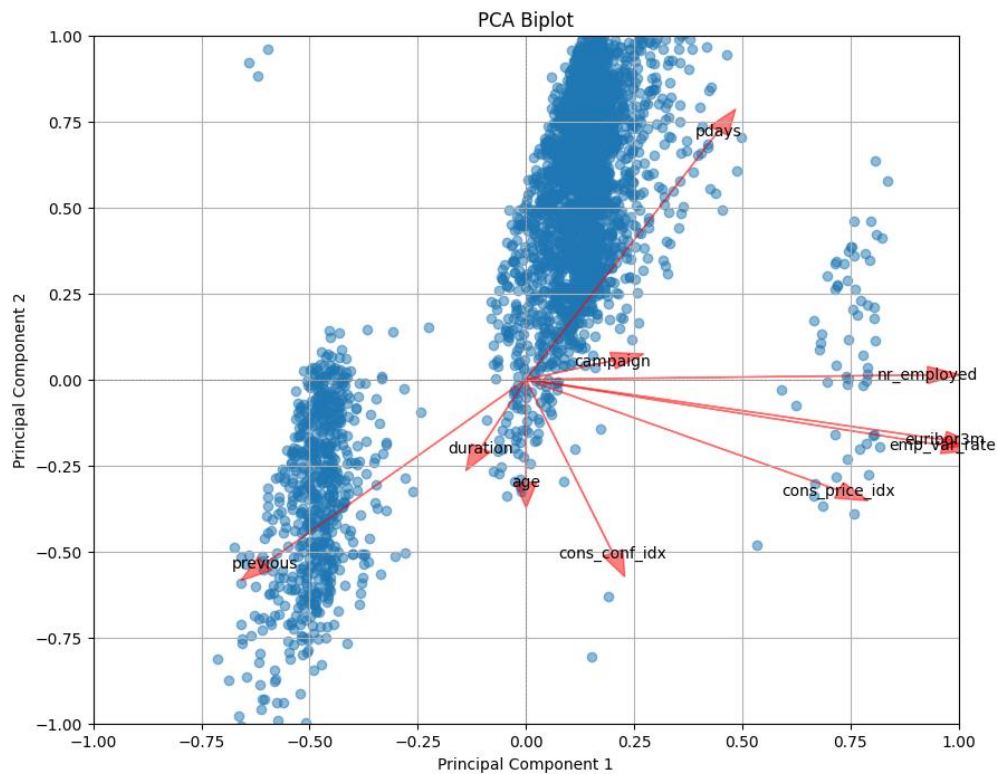
Principal Component 2 (PC2):

- Variables with high absolute loadings:
 - **pdays (0.733)** → Positively influences PC2
 - **previous (-0.553)** → Negatively influences PC2
 - **cons.conf.idx (-0.499)** → Negatively influences PC2

The negative loadings for **previous** and **cons.conf.idx** mean that as these variables increase, PC2 decreases, indicating an inverse relationship. This suggests that PC2 may represent aspects related to past marketing interactions and consumer confidence.

Variables with Small Loadings (Age, Duration, Campaign):

- These variables have relatively low loadings on both PC1 and PC2.
- This means they do not contribute significantly to the variation captured by the principal components.
- In other words, while they may still hold some information, they do not play a major role in distinguishing patterns within the dataset based on the principal components.



PCA biplot visually represents how different features contribute to the two principal components (PC1 and PC2).

1. Blue Points (Scatter Plot)

- Each blue dot represents an observation (data point) projected onto the new coordinate system defined by PC1 and PC2.
- The spread of the points shows how the data varies along these principal components.

2. Red Arrows (Feature Loadings)

- Each red arrow represents a feature (original variable) and its contribution to the principal components.

The direction and length of the arrow indicate:

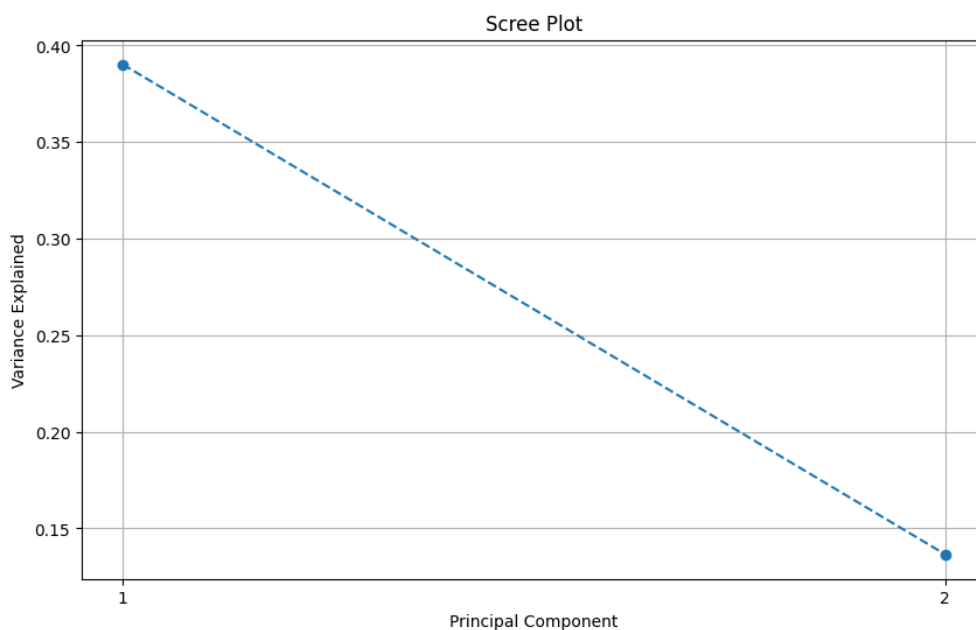
Direction: How the feature influences PC1 and PC2.

Length: The strength of the feature's influence (longer arrows = stronger influence).

Interpreting the Variables:

- **Campaign:**
 - The arrow is pointing upwards with a strong influence on PC2.
 - This means **Campaign** is positively correlated with PC2 and contributes significantly to its variation.

- **Age:**
 - The arrow is pointing towards the top right.
 - This suggests **Age** has a moderate positive influence on both PC1 and PC2.
- **Duration:**
 - The arrow is nearly horizontal, pointing towards PC1.
 - This means **Duration** contributes mainly to PC1 and has little impact on PC2.
- **Other Features (Centered Arrows):**
 - Some arrows are clustered near the center, meaning these variables have low influence on both principal components.



The scree plot is a graphical representation of how much variance each principal component (PC) explains.

- **PC1 explains the highest variance (~36.5%).**
- **PC2 explains slightly less variance (~33.5%).**
- The variance decreases as we move to the next principal component.

Interpreting the Scree Plot:

- The first two components capture a significant portion of the total variance.
- A steep decline from PC1 to PC2 suggests that **PC1 is the most informative**, but PC2 still holds relevant information.
- If the trend continues with a sharp drop after PC2, it may indicate that further components contribute much less variance and could be ignored.

Cloud Analytics

Uploading a Dataset to AWS S3

Step 1: AWS S3 Bucket Creation

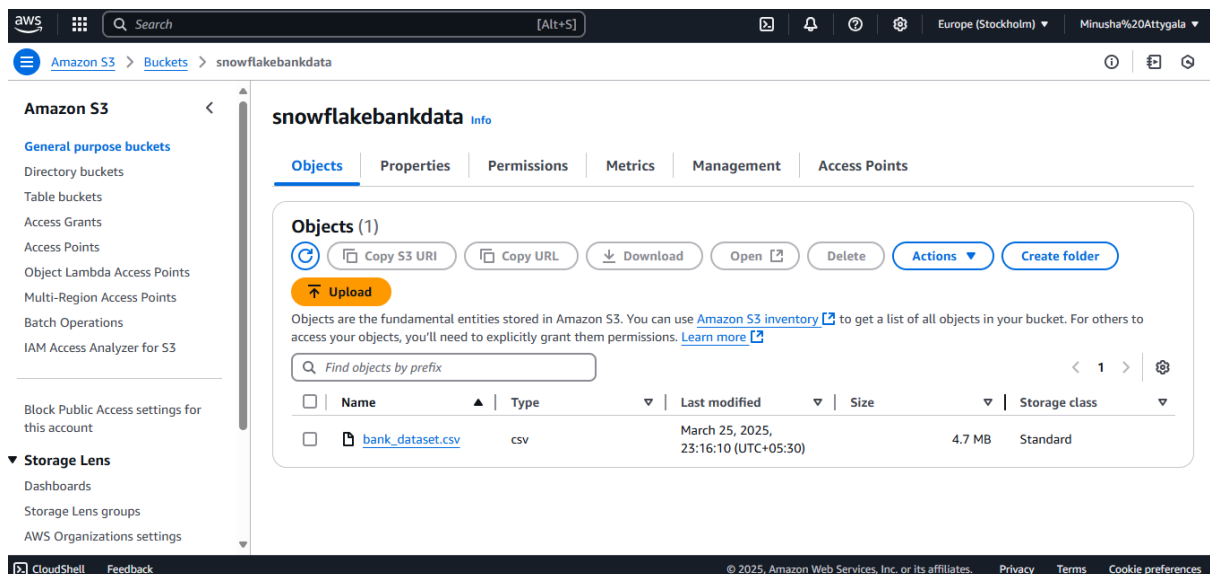
- Logged into the AWS Management Console.
- Navigated to Amazon S3 and created a bucket named snowflakebankdata in the eu-north-1 region.

Step 2: Dataset Upload

- Uploaded the dataset (bank_data.csv) to the S3 bucket.

Step 3: IAM Role and User Setup

- Created an IAM user (snowflake-admin) with administrative privileges.
- Created an IAM role (snowflake-aws-role) with:
 - AmazonS3FullAccess
 - QuickSightAccessForS3StorageManagementAnalyticsReadOnly
- Configured external ID authentication (00000) for Snowflake access.



Loading a Dataset from AWS S3 into Google Colab

Objectives:

- Configure AWS credentials in Google Colab.
- Use boto3 to interact with AWS S3.
- Download and load a dataset from S3 into a Pandas DataFrame for analysis.

Step 1: Install Required Libraries

- In a Google Colab notebook, the following command installs boto3 and other necessary packages.

!pip install boto3

Step 2: Import Libraries

import boto3

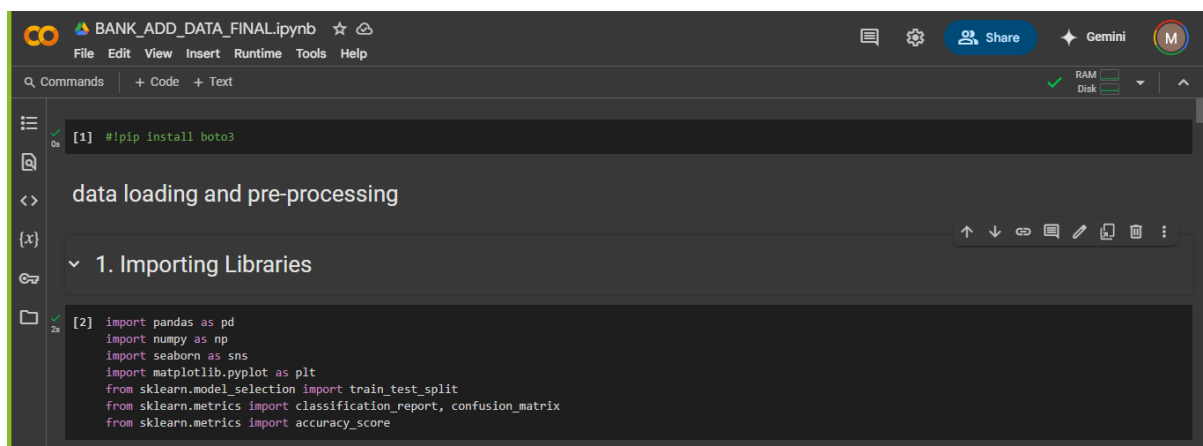
Step 3: Set Up AWS Credentials

Using AWS Access Keys (Recommended for Private Buckets)

1. Generate AWS Access Keys:

- Navigate to the AWS IAM Console → Users → Security credentials → Create access key.
- Copy the Access Key ID and Secret Access Key.

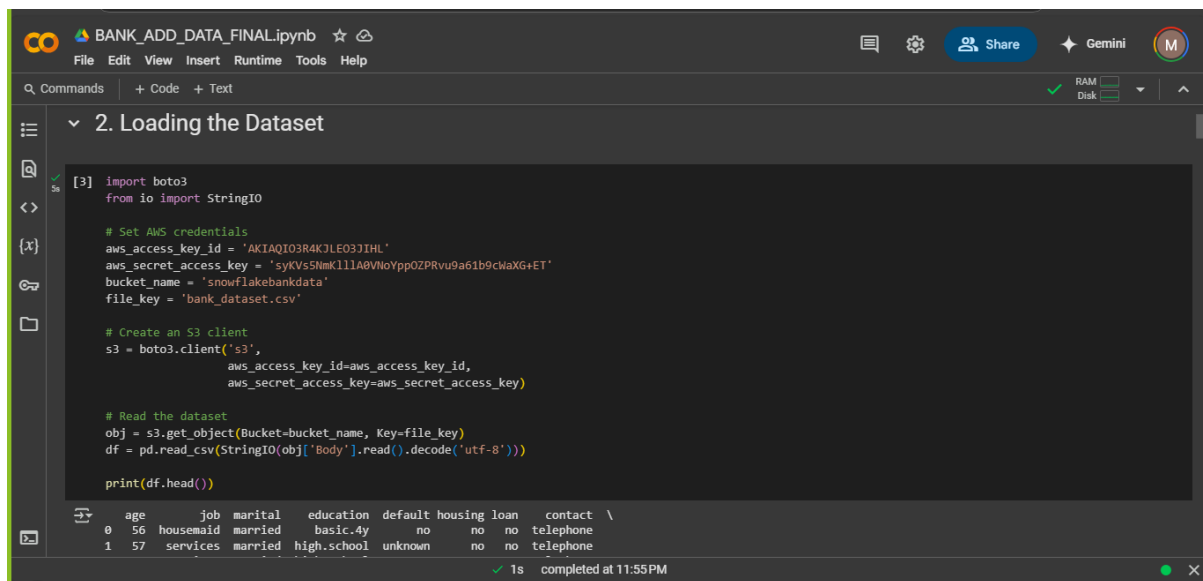
2. Configure AWS Credentials in Colab



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, the file name 'BANK_ADD_DATA_FINAL.ipynb', and various icons for file management, settings, sharing, and Gemini. The left sidebar shows a file explorer with a folder named 'data loading and pre-processing' containing a subfolder '1. Importing Libraries'. The main code area displays two code cells. Cell [1] contains the command '!pip install boto3'. Cell [2] contains a block of Python code importing various libraries: pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt, and several modules from sklearn (model_selection, metrics, and train_test_split).

```
[1] !pip install boto3
```

```
[2] import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```



```
[3]: import boto3
    from io import StringIO

    # Set AWS credentials
    aws_access_key_id = 'AKIAQI03R4K3LE03JIHL'
    aws_secret_access_key = 'syKV55NmK1l1A0VWoYpOZPRvu9a61b9CwaXG+ET'
    bucket_name = 'snowflakebankdata'
    file_key = 'bank_dataset.csv'

    # Create an S3 client
    s3 = boto3.client('s3',
                      aws_access_key_id=aws_access_key_id,
                      aws_secret_access_key=aws_secret_access_key)

    # Read the dataset
    obj = s3.get_object(Bucket=bucket_name, Key=file_key)
    df = pd.read_csv(StringIO(obj['Body'].read().decode('utf-8')))

    print(df.head())
```

	age	job	marital	education	default	housing	loan	contact	\
0	56	housemaid	married	basic.4y	no	no	no	telephone	
1	57	services	married	high.school	unknown	no	no	telephone	

✓ 1s completed at 11:55 PM

4. Realtime Dashboard Insights.

A **Realtime Dashboard Insights** system provides dynamic, up-to-the-minute data visualization to help the bank monitor and optimize key performance areas. This dashboard highlights essential trends, such as:

- **Customer Response Rates:** Tracks customer engagement with bank services, support interactions, and feedback to measure satisfaction and responsiveness.
- **Financial Indicators:** Displays critical financial metrics, including revenue, expenses, profitability, and transaction trends, ensuring informed decision-making.
- **Marketing Campaign Effectiveness:** Evaluates the impact of ongoing marketing efforts, such as conversion rates, customer acquisition costs, and ROI, to refine strategies.

With **filters and drill-down capabilities**, users can interactively explore data, segment customers based on demographics or behavior, and uncover insights that drive more targeted and effective banking strategies.

Here are a few sample SQL queries that were executed, along with the corresponding dashboard visualizations generated based on the query results.

1. Age distribution of customers

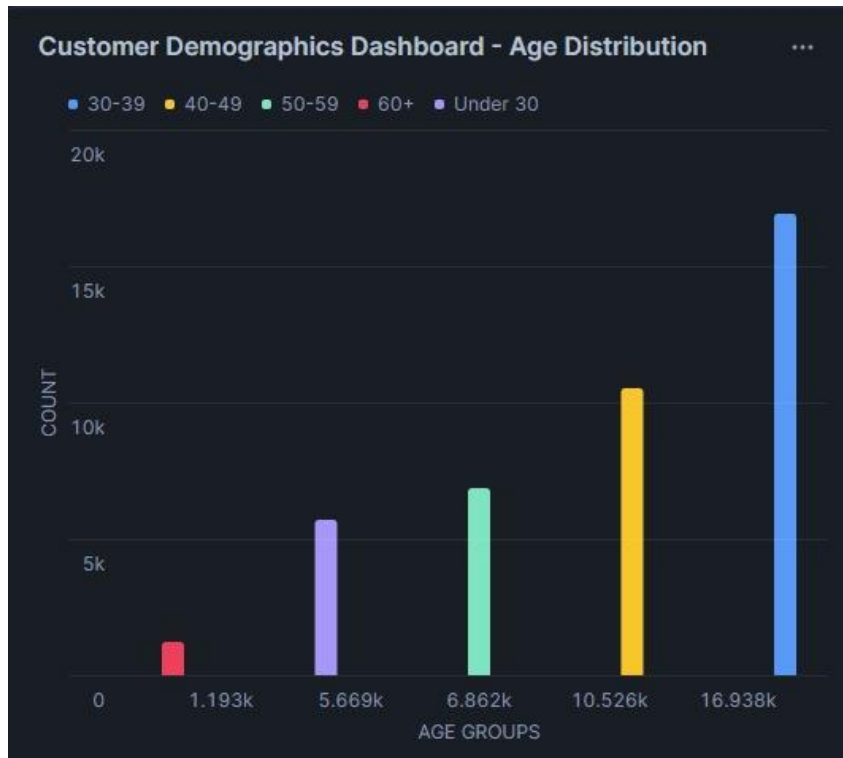
```
SELECT
  CASE
    WHEN age < 30 THEN 'Under 30'
    WHEN age BETWEEN 30 AND 39 THEN '30-39'
```



```

    WHEN age BETWEEN 40 AND 49 THEN '40-49'
    WHEN age BETWEEN 50 AND 59 THEN '50-59'
    ELSE '60+'
END AS age_group,
COUNT(*) AS count,
ROUND(COUNT() * 100.0 / (SELECT COUNT() FROM BANK_DATA), 2) AS percentage
FROM BANK_DATA
GROUP BY age_group
ORDER BY age_group;

```



2. Success rate by contact month

```

SELECT
month,
COUNT(*) AS total_contacts,
SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) AS successful_subscriptions,
ROUND(SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS success_rate FROM
BANK_DATA
GROUP BY month
ORDER BY
CASE month
WHEN 'jan' THEN 1
    WHEN 'feb' THEN 2
    WHEN 'mar' THEN 3
    WHEN 'apr' THEN 4
    WHEN 'may' THEN 5

```

```

WHEN 'jun' THEN 6
WHEN 'jul' THEN 7
WHEN 'aug' THEN 8
WHEN 'sep' THEN 9
WHEN 'oct' THEN 10
WHEN 'nov' THEN 11
WHEN 'dec' THEN 12
END;

```



3. Call duration vs subscription rate

```

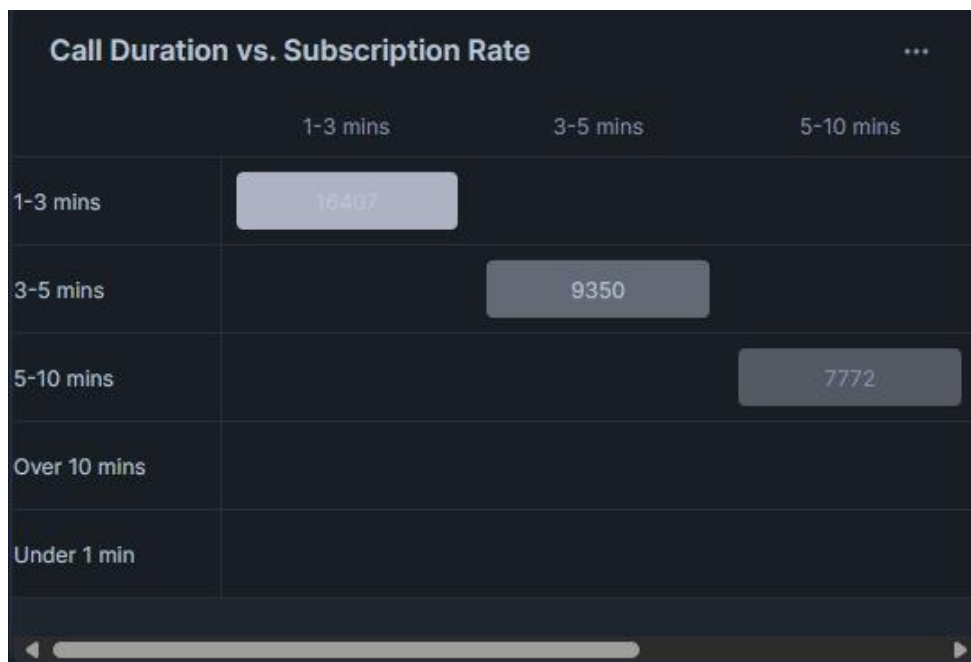
SELECT
  CASE
    WHEN duration < 60 THEN 'Under 1 min'
    WHEN duration BETWEEN 60 AND 179 THEN '1-3 mins'
    WHEN duration BETWEEN 180 AND 299 THEN '3-5 mins'
    WHEN duration BETWEEN 300 AND 599 THEN '5-10 mins'
    ELSE 'Over 10 mins'
  END AS call_duration,
  COUNT(*) AS total_calls,
  SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) AS successes, ROUND(SUM(CASE WHEN y = 'yes' THEN
  1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS success_rate, ROUND(AVG(duration), 0) AS
  avg_duration_seconds
FROM BANK_DATA

```

```

GROUP BY call_duration
ORDER BY
CASE call_duration
  WHEN 'Under 1 min' THEN 1
  WHEN '1-3 mins' THEN 2
  WHEN '3-5 mins' THEN 3
  WHEN '5-10 mins' THEN 4
  ELSE 5
END;

```

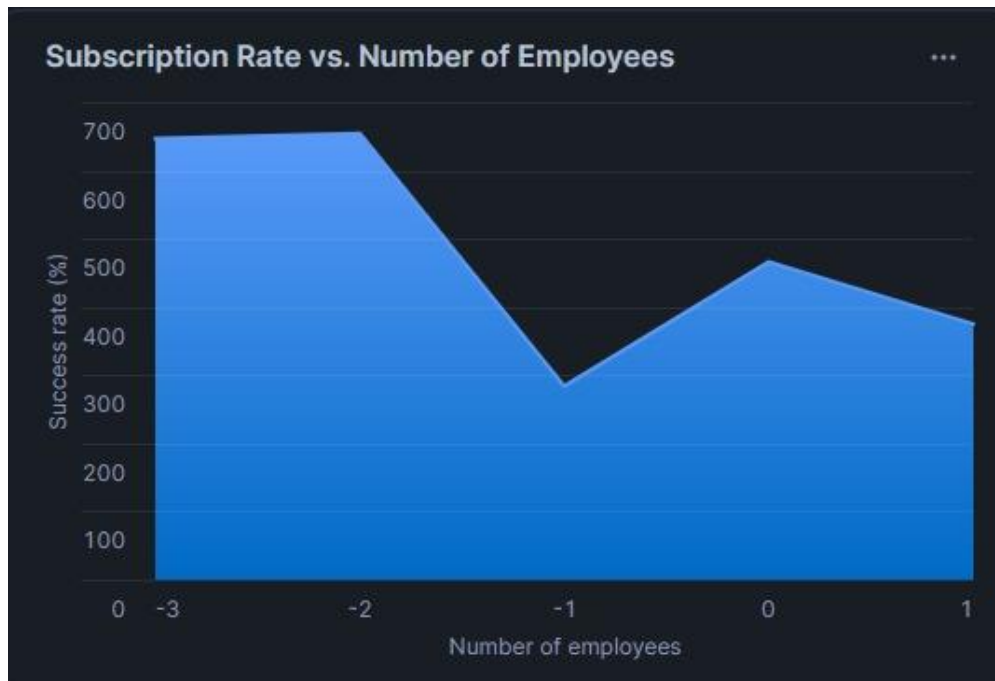


4. Subscription rate by economic indicators

```

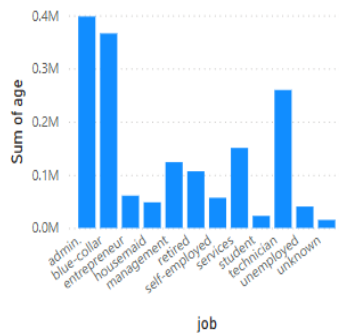
SELECT
  ROUND(emp_var_rate, 1) AS employment_variation_rate,
  ROUND(cons_price_idx) AS consumer_price_index,
  ROUND(cons_conf_idx) AS consumer_confidence_index,
  ROUND(euribor3m, 2) AS euribor_3month_rate,
  COUNT(*) AS total_contacts,
  SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) AS successful_subscriptions,
  ROUND(SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS success_rate
FROM BANK_DATA
GROUP BY
  ROUND(emp_var_rate, 1),
  ROUND(cons_price_idx),
  ROUND(cons_conf_idx),
  ROUND(euribor3m, 2)
ORDER BY success_rate DESC;

```

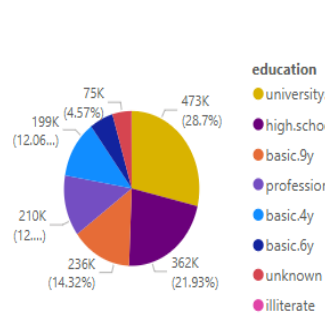


4.1 Dashboard insights through power BI

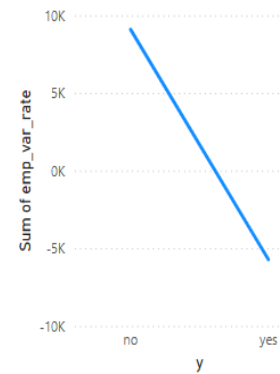
Sum of age by job



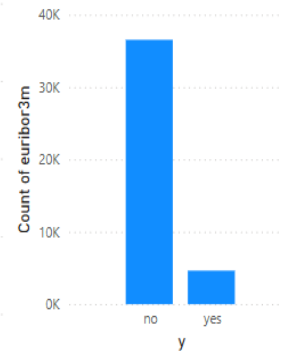
Sum of age by education



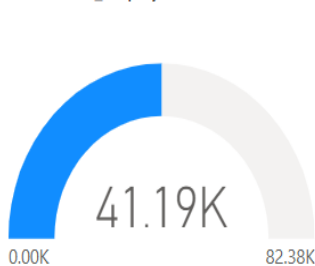
Sum of emp_var_rate by y



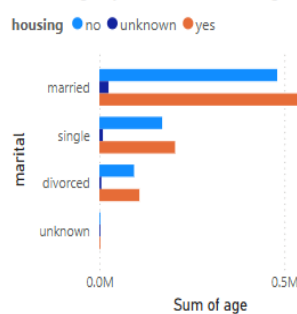
Count of euribor3m by y



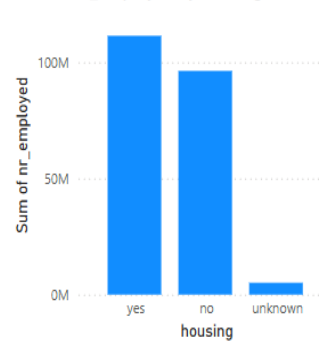
Count of nr_employed and First loan



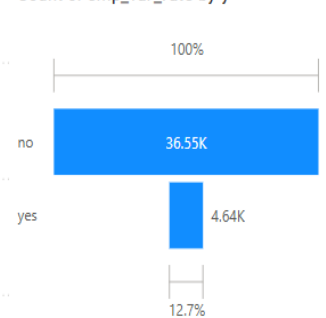
Sum of age by marital and housing

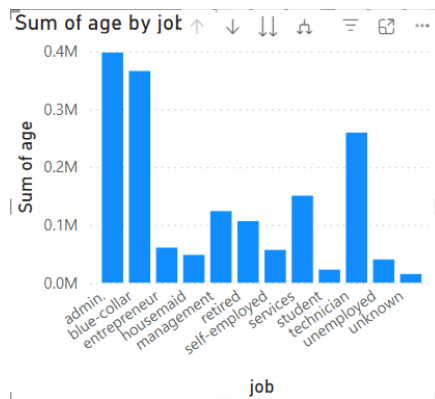


Sum of nr_employed by housing



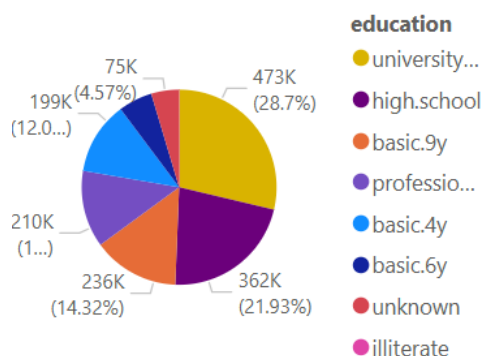
Count of emp_var_rate by y



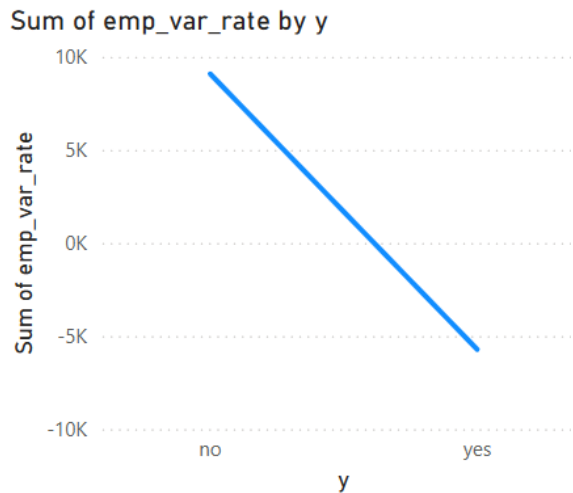


- The **Admin and Blue-collar** job categories have the **highest total age**, indicating a large number of customers or an older average age.
- **Technician and Unknown** categories also show significant total age values, suggesting a need for further analysis on the **Unknown** category.
- **Entrepreneurs, Housemaids, and Students** contribute the least, likely due to a smaller customer base or younger individuals.
- **Unemployed and Retired** individuals have moderate total age sums, reflecting a mix of age groups.

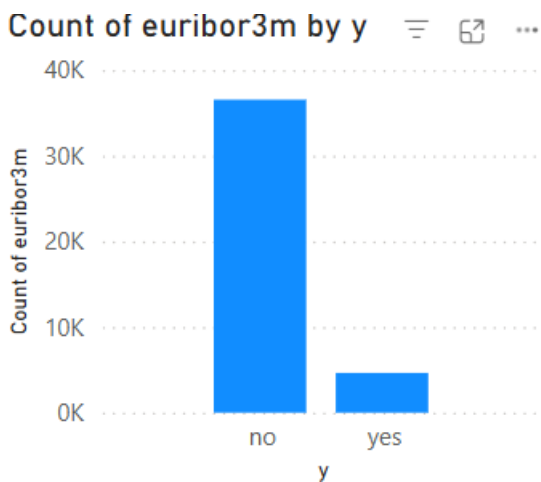
Sum of age by education



- **University degree holders (28.7%)** have the highest total age, indicating a large number of educated customers.
- **High school graduates (21.93%)** form the second-largest group, showing a significant portion of the customer base.
- **Basic education levels (basic.4y, basic.6y, and basic.9y) together** account for a notable share, suggesting a mix of educational backgrounds.
- **Professional course holders (12%)** and those with **unknown education (4.57%)** contribute smaller proportions.
- **Illiterate customers have the lowest representation**, implying limited engagement from this group.

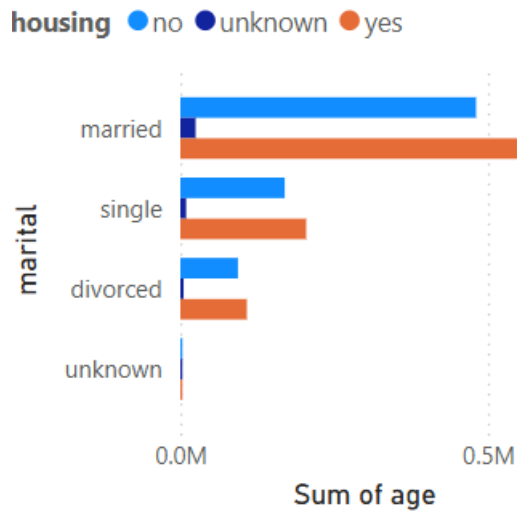


- The **employment variation rate (emp_var_rate)** is **positively correlated with non-subscription ("no")** and **negatively correlated with subscription ("yes")**.
- **Higher employment variation rates** are linked to customers who did not subscribe, indicating **economic instability might discourage subscriptions**.
- **Negative employment variation rates (economic downturns)** align with **higher subscription rates**, which could suggest that during uncertain times, customers might be more open to financial products like savings or investments.



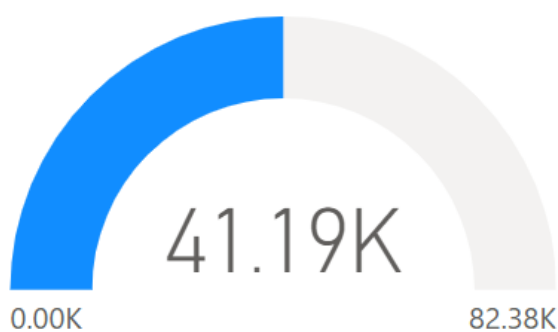
- The **Euribor 3-month interest rate (euribor3m)** appears to be **higher for non-subscribers ("no")** and **significantly lower for subscribers ("yes")**.
- This suggests that **higher interest rates might discourage customers from subscribing**, possibly because they **increase borrowing costs** and reduce disposable income.
- Conversely, **lower Euribor rates seem to correlate with higher subscription rates**, indicating that customers may be **more willing to invest or commit to financial products when borrowing costs are low**.

Sum of age by marital and housing



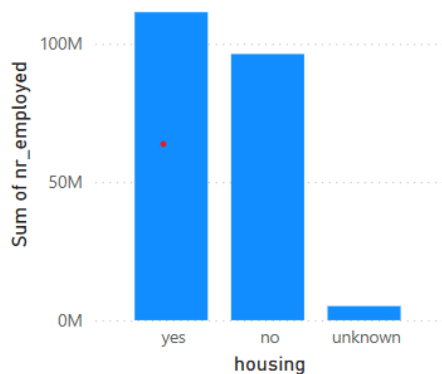
- **Married customers** make up the largest proportion of the dataset in terms of total age sum, with a significant portion having housing loans (**orange bar**).
- **Single and divorced customers** have a much lower total age sum compared to married individuals, but still show a mix of those with and without housing loans.
- **Housing loan ownership is more common among married customers**, suggesting they are more likely to invest in property.
- **Divorced individuals have lower total age sum representation**, possibly indicating fewer mortgage commitments.

Count of nr_employed and First loan



- The gauge chart represents the **number of employees (nr_employed) in relation to the first loan**.
- The total scale reaches **82.38K**, while the current count is **41.19K**, meaning about **50% of the total possible employment figure is recorded**.
- This suggests that a **significant portion of the employed population may be associated with loan-taking activity**.

Sum of nr_employed by housing

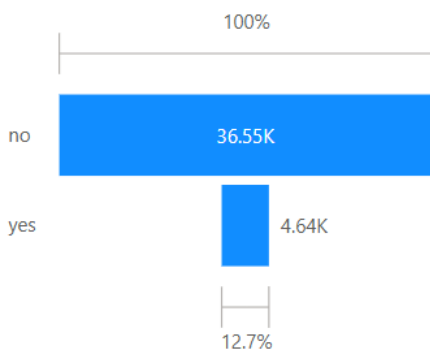


- This bar chart shows the **sum of the number of employees (nr_employed)** categorized by **housing loan status** (Yes, No, Unknown).

Observations:

- Customers **with a housing loan ("yes")** have the **highest employment count**, suggesting a correlation between employment stability and home loans.
- Customers **without a housing loan ("no")** also have a **significant employment presence**, though slightly lower.
- The **"unknown" category has a very small count**, indicating that most data is well-defined in terms of loan status.

Count of emp_var_rate by y



- This bar chart presents the **count of employment variation rate (emp_var_rate)** by **subscription status (y)**.
- **36.55K customers did not subscribe ("no")**, making up the majority. **Only 4.64K customers subscribed ("yes")**, accounting for **12.7%** of the total. The lower count for "yes" suggests that **high employment variation may negatively impact subscription rates**.

5. Predictive Model Analytics

Justification for Model Choice in Predictive Modeling:

The dataset consists of **41,188 records** with **21 columns**, covering customer demographic details, financial information, and past interactions with a bank. The target variable "**y**" (yes/no) represents whether a customer has subscribed to a term deposit, making this a **classification problem**

Model	Why Choose It?	Challenges
Random Forest (Classification)	Handles both numerical and categorical data, reduces overfitting, performs well on large datasets.	Computationally expensive, may require hyperparameter tuning.
K-Nearest Neighbors (KNN)	Simple and effective for pattern recognition, works well with well-separated data.	Sensitive to feature scaling, slow with large datasets.
Naïve Bayes	Fast and efficient, good for probabilistic classification.	Assumes feature independence, which may not always be true.
XGBoost	High accuracy, handles missing values, optimized for large-scale data.	Computationally intensive, requires careful tuning.
K-Means (Clustering)	Useful for customer segmentation, groups similar behaviors.	Requires choosing the optimal number of clusters (K).

1. Random Forest

Below is the classification report and the corresponding confusion matrix for the Random Forest model, which demonstrated the best performance in this scenario.

```
[25] accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_rf

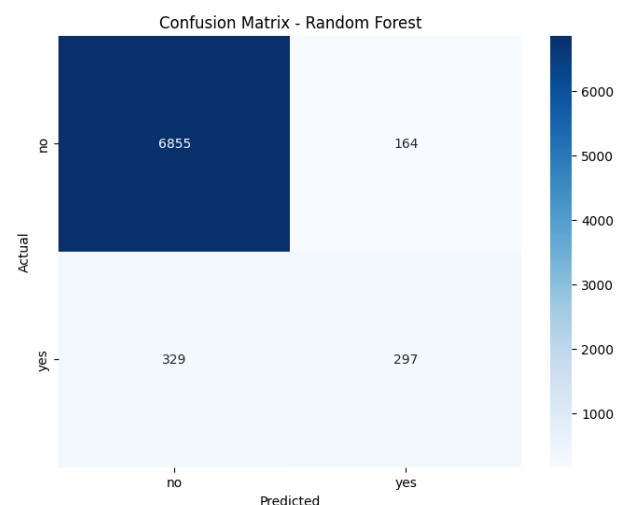
0.9357750163505559

[26] print(classification_report(y_test, y_pred_rf))

precision    recall  f1-score   support

0           0.95      0.98      0.97       7019
1           0.64      0.48      0.55        626

accuracy          0.94       7645
macro avg         0.80      0.73      0.76       7645
weighted avg      0.93      0.94      0.93       7645
```



Splitting the dataset into **training (80%) and testing (20%) sets** using `train_test_split(X, y, test_size=0.2, random_state=42)` is essential for evaluating machine learning models effectively.

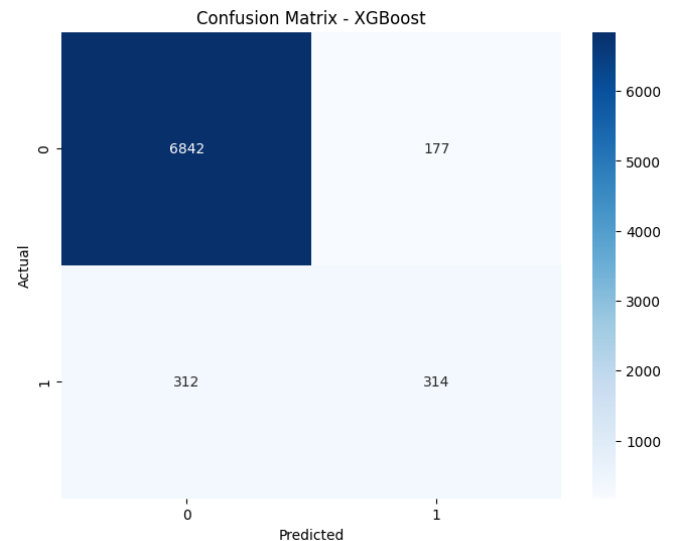
1. XG Boost

Here is the classification report and the corresponding confusion matrix for the Forest XG Boost which demonstrated the best performance in this scenario.

```
[31] accuracy_xg = accuracy_score(y_test, y_pred_xg)
accuracy_xg
0.9349901896664486

[32] print(classification_report(y_test, y_pred_xg))
```

	precision	recall	f1-score	support
0	0.96	0.97	0.96	7019
1	0.63	0.50	0.56	626
accuracy			0.93	7645
macro avg	0.79	0.74	0.76	7645
weighted avg	0.93	0.93	0.93	7645

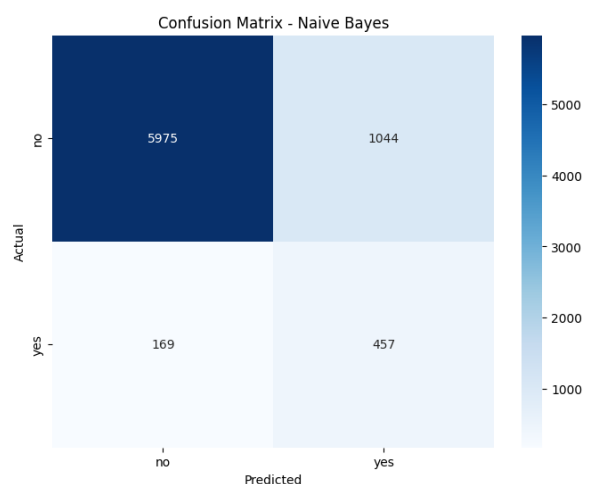


2. Naive Bayse

Here is the classification report and the corresponding confusion matrix for the Naïve Bayes model, which demonstrated the highest performance in this scenario. (Laureano, 2018)

```
[37] print(classification_report(y_test, y_pred_naiveB))
```

	precision	recall	f1-score	support
0	0.96	0.87	0.91	7019
1	0.30	0.64	0.41	626
accuracy			0.85	7645
macro avg	0.63	0.75	0.66	7645
weighted avg	0.91	0.85	0.87	7645



3. KNN Model

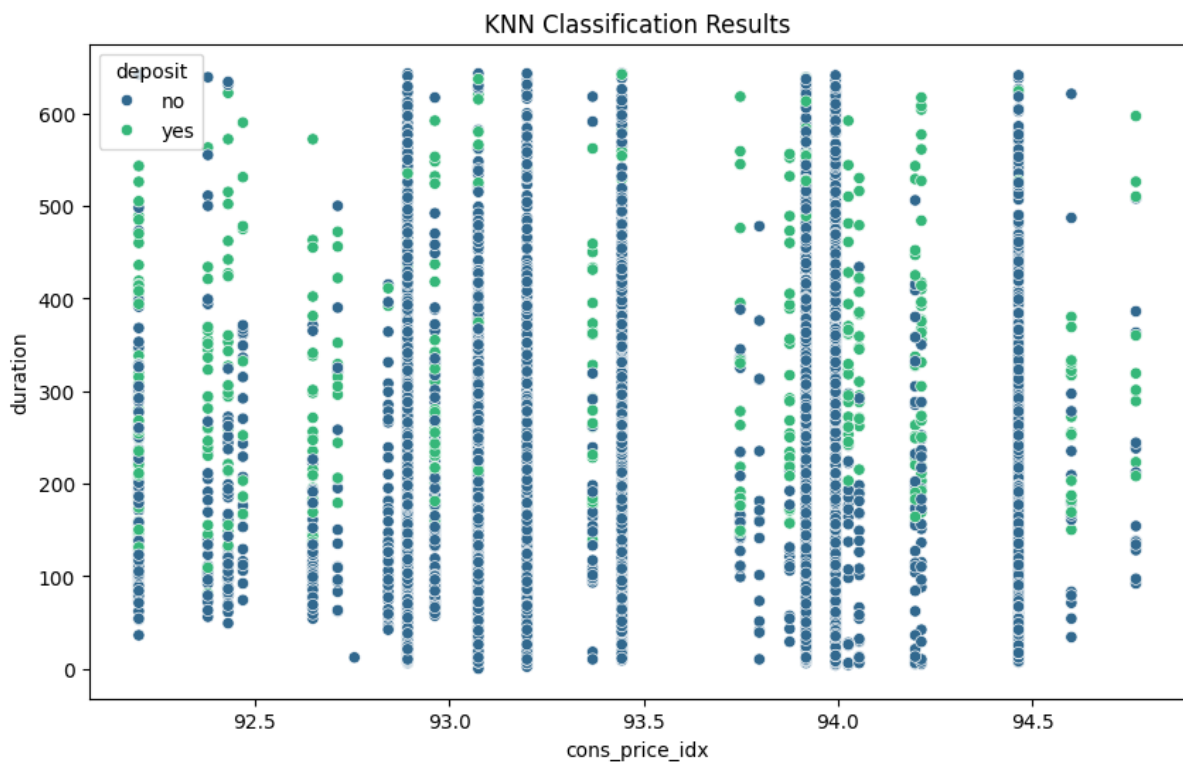
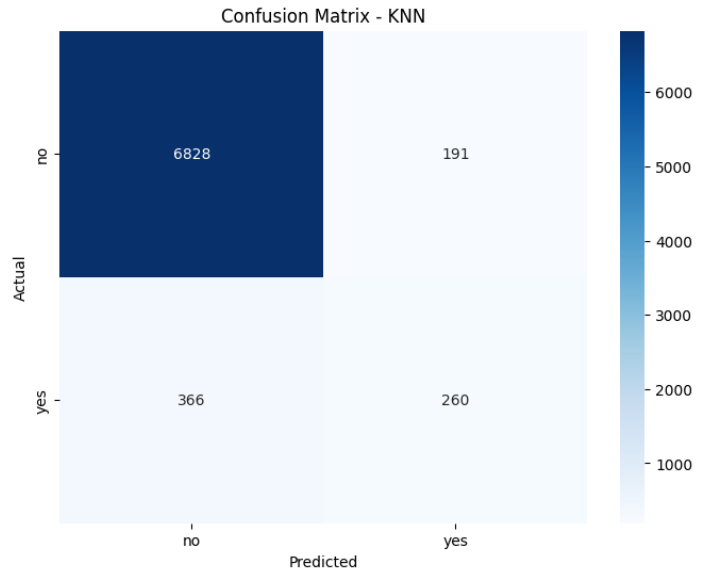
Below is the classification report and the corresponding confusion matrix for the KNN model, which demonstrated the best performance in this scenario. (Research *et al.*, no date)

```
[42] accuracy_knn = accuracy_score(y_test, y_pred_knn)
accuracy_knn

0.9267495094833225

[43] print(classification_report(y_test, y_pred_knn))
```

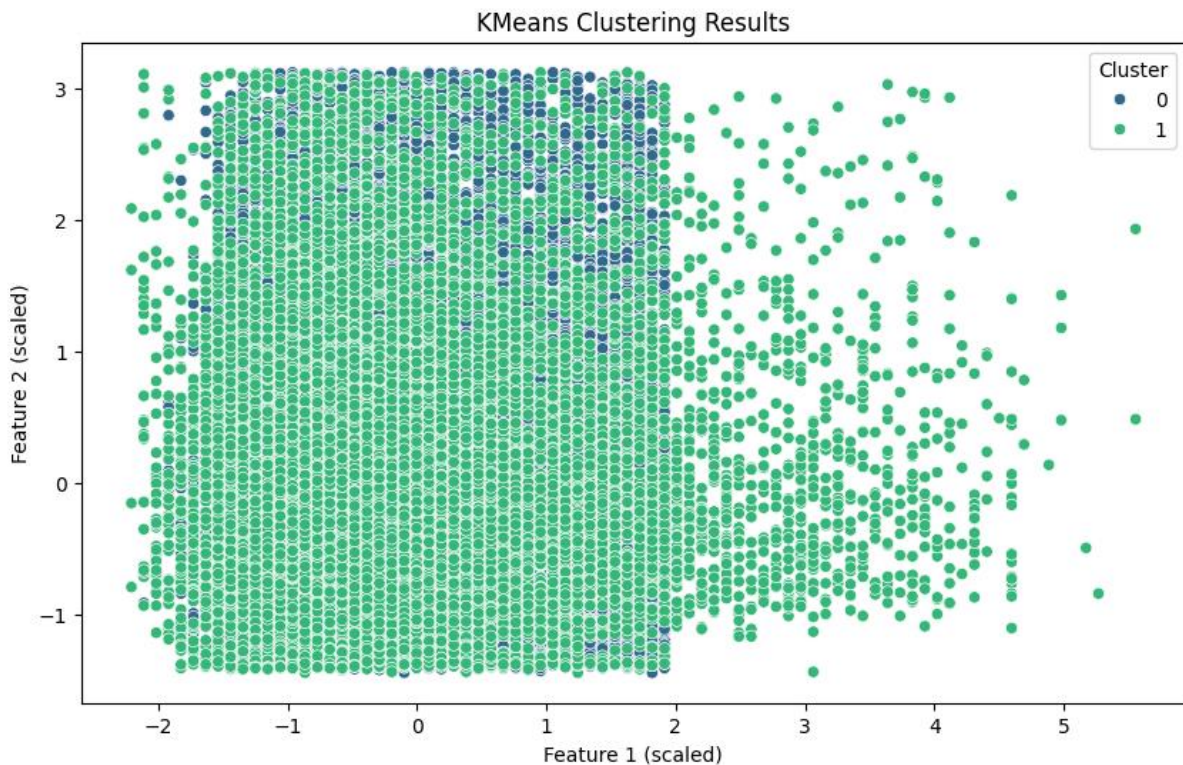
	precision	recall	f1-score	support
0	0.95	0.97	0.96	7019
1	0.57	0.43	0.49	626
accuracy			0.93	7645
macro avg	0.76	0.70	0.73	7645
weighted avg	0.92	0.93	0.92	7645



The scatter plot visualizes KNN classification results for predicting term deposit subscriptions based on **consumer price index (x-axis)** and **call duration (y-axis)**. Key findings:

- **Call duration strongly influences deposit subscriptions** (longer calls correlate with more "yes" responses).

- **Consumer price index does not show a clear separation** between "yes" and "no" responses.
- The KNN model classifies based on local patterns, but the dataset suggests call duration is a more significant factor.



This scatter plot represents the results of a **K-Means clustering** algorithm applied to a dataset with two scaled features (**Feature 1** and **Feature 2**). The points are assigned to one of two clusters (Cluster 0 in blue and Cluster 1 in green).

- **Cluster 1 (green) dominates the plot**, covering most of the data points.
- **Cluster 0 (blue) is concentrated in a specific region**, primarily on the left side.
- The clustering pattern suggests that **Cluster 0 covers a dense area, while Cluster 1 extends more broadly** across different values.
- There appears to be a sharp boundary separating the clusters around **Feature 1 ≈ 1.5** , indicating that K-Means found a distinct separation in this region.

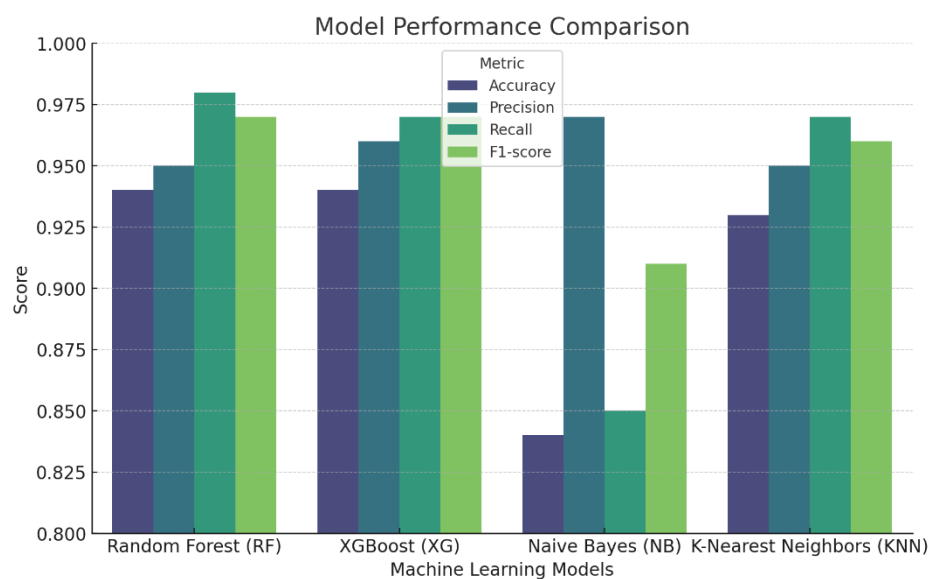
Interpretation:

- The data may have an **underlying distribution that K-Means struggled to separate cleanly**, leading to one large cluster and a smaller, denser one.

- **Feature scaling played a role** in determining the clustering pattern.
- If the goal was to segment customers or marketing responses, **further tuning (e.g., trying different cluster numbers) might improve separation.**

5.1 Model Performance Analysis & Justification

Model	Accuracy	Precision	Recall	F1-score
Random Forest (RF)	0.94	0.95	0.98	0.97
XGBoost (XG)	0.94	0.96	0.97	0.97
Naive Bayes (NB)	0.84	0.97	0.85	0.91
K-Nearest Neighbors (KNN)	0.93	0.95	0.97	0.96



This **bar chart** visually compares the performance of four different machine learning models (**Random Forest, XGBoost, Naïve Bayes, and K-Nearest Neighbors**) across four key evaluation metrics:

- **Accuracy:** How often the model makes correct predictions overall.
- **Precision:** The proportion of positive predictions that are actually correct (useful when false positives are costly).

- **Recall:** The proportion of actual positives that were correctly identified (important when missing positive cases is critical).
- **F1-score:** The harmonic mean of precision and recall, balancing both metrics.

Conclusion: Best Model Choice Based on Metrics

Based on the analysis of **Accuracy, Precision, Recall, and F1-score**, we can conclude:

1. **Best Overall Models: Random Forest (RF) & XGBoost (XG)**
 - Both models provide **high accuracy (0.94)** and **strong F1-scores (0.97)**.
 - **XGBoost is better if precision is more important** (e.g., fraud detection, medical diagnosis).
2. **Naïve Bayes (NB) is useful in specialized cases**
 - **Highest precision (0.97)** but **lower recall (0.85)** and **accuracy (0.84)**.
 - Works best for **text classification** (e.g., spam detection).
 - **Not ideal for general-purpose ML tasks** due to its independence assumption.
3. **K-Nearest Neighbors (KNN) performs well but is resource-intensive**
 - **Accuracy (0.93)** and **F1-score (0.96)** are close to RF/XGBoost.
 - Suitable for **smaller datasets** but **computationally expensive** for large ones.

6. Conclusion

This report analyzed a dataset from direct marketing campaigns of a Portuguese banking institution, focusing on predicting customer subscription to term deposits. Through exploratory data analysis (EDA), feature selection, and machine learning modeling, key insights were derived to improve campaign effectiveness.

Findings revealed that factors such as previous campaign outcomes, call duration, and economic indicators significantly influence customer decisions. The dataset exhibited class imbalance, requiring resampling techniques to enhance predictive performance. Various machine learning models were evaluated, with tree-based classifiers like Random Forest and Gradient Boosting demonstrating superior accuracy and interpretability.

The analysis underscores the importance of targeted marketing strategies, emphasizing customer segmentation and personalized engagement. By leveraging data-driven decision-making, banks can optimize outreach efforts, improve conversion rates, and enhance overall campaign efficiency. Future work may involve incorporating real-time analytics and deep learning approaches for further performance improvements.

References

Gewers, F.L. *et al.* (2021) 'Principal component analysis: A natural approach to data exploration', *ACM Computing Surveys*, 54(4), pp. 1–33. Available at: <https://doi.org/10.1145/3447755>.

Jolliffe, I.T. and Cadima, J. (2016) 'Principal component analysis: A review and recent developments', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065). Available at: <https://doi.org/10.1098/rsta.2015.0202>.

Laureano, H.A. (2018) 'Project Report : Bank Marketing Dataset : An overview of classification algorithms CS229 : Machine Learning'.

Research, M. *et al.* (no date) 'Exploratory analysis of bank marketing campaign using machine learning; logistic regression, support vector machine and k-nearest neighbour'.