EXPLORATORY DATA ANALYSIS FOR XYZ BANK

(BAN-5753, TEAM INCISIVE)

TABLE OF CONTENT

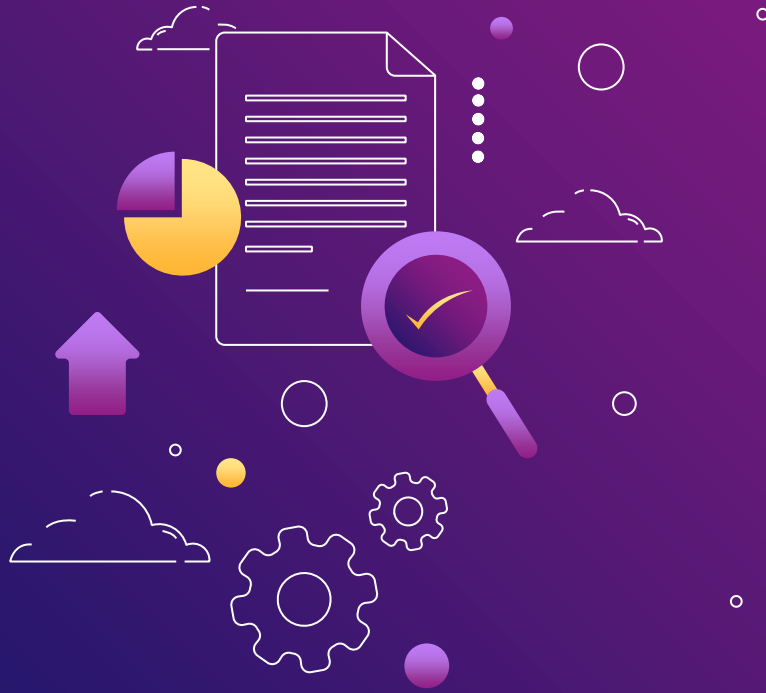
TEAM INCISIVE

ANIRUDH BOMMINA

THARUN PONNAGANTI

ABDUL MUJEEB

PRAFULLA BALASAHED SATURE



INTRODUCTION

XYZ Bank seeks to optimize its direct marketing campaigns for term deposit subscriptions. By applying machine learning models, the bank aims to predict which customers are more likely to subscribe, thus improving the allocation of marketing resources and customer engagement strategies..

PROBLEM STATEMENT

- Objective: Classify bank clients by their likelihood to subscribe to a term deposit

- Key Points:
  Analyze client attributes and campaign interactions.
  Binary classification: 'Yes' or 'No' for term deposit subscription.
  Perform EDA to reveal influencing patterns and trends.

- **Goal:**
  Leverage insights to predict and enhance term deposit subscriptions.

LOADING THE DATA

1. The data is loaded from a CSV file into a Spark DataFrame.

In [3]:
```
file_path = "C:/Users/aniru/OneDrive/Documents/Assignments sem 3/Advance Business Analytics/Mini project-2/XYZ_Bank_Deposit_Data_

# Read the CSV file into a DataFrame
data = spark.read.csv(file_path, header=True, inferSchema=True, sep=";")
data.head(5)  # Show first 5 rows to verify data is loaded correctly
```

Out[3]:
```
[Row(age=56, job='housemaid', marital='married', education='basic.4y', default='no', housing='no', loan='no', contact='telephon
e', month='may', day_of_week='mon', duration=261, campaign=1, pdays=999, previous=0, poutcome='nonexistent', emp.var.rate=1.1,
cons.price.idx=93.994, cons.conf.idx=-36.4, euribor3m=4.857, nr.employed=5191.0, y='no'),
 Row(age=57, job='services', marital='married', education='high.school', default='unknown', housing='no', loan='no', contact='t
elephone', month='may', day_of_week='mon', duration=149, campaign=1, pdays=999, previous=0, poutcome='nonexistent', emp.var.rat
e=1.1, cons.price.idx=93.994, cons.conf.idx=-36.4, euribor3m=4.857, nr.employed=5191.0, y='no'),
 Row(age=37, job='services', marital='married', education='high.school', default='no', housing='yes', loan='no', contact='telep
hone', month='may', day_of_week='mon', duration=226, campaign=1, pdays=999, previous=0, poutcome='nonexistent', emp.var.rate=1.
1, cons.price.idx=93.994, cons.conf.idx=-36.4, euribor3m=4.857, nr.employed=5191.0, y='no'),
 Row(age=40, job='admin.', marital='married', education='basic.6y', default='no', housing='no', loan='no', contact='telephone',
month='may', day_of_week='mon', duration=151, campaign=1, pdays=999, previous=0, poutcome='nonexistent', emp.var.rate=1.1, con
s.price.idx=93.994, cons.conf.idx=-36.4, euribor3m=4.857, nr.employed=5191.0, y='no'),
 Row(age=56, job='services', marital='married', education='high.school', default='no', housing='no', loan='yes', contact='telep
hone', month='may', day_of_week='mon', duration=307, campaign=1, pdays=999, previous=0, poutcome='nonexistent', emp.var.rate=1.
1, cons.price.idx=93.994, cons.conf.idx=-36.4, euribor3m=4.857, nr.employed=5191.0, y='no')]
```

DATA CLEANING AND TRANSFORMATION

1. Standardized column names by replacing dots and spaces.
2. Handled missing values by checking and addressing null values.
3. Converted data types as needed.

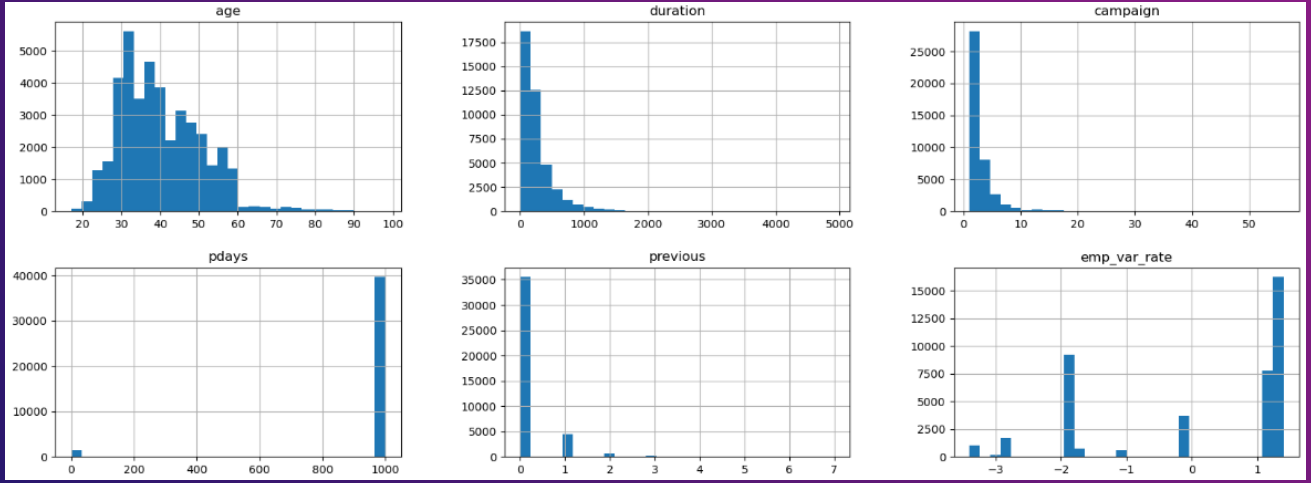
Data Cleaning

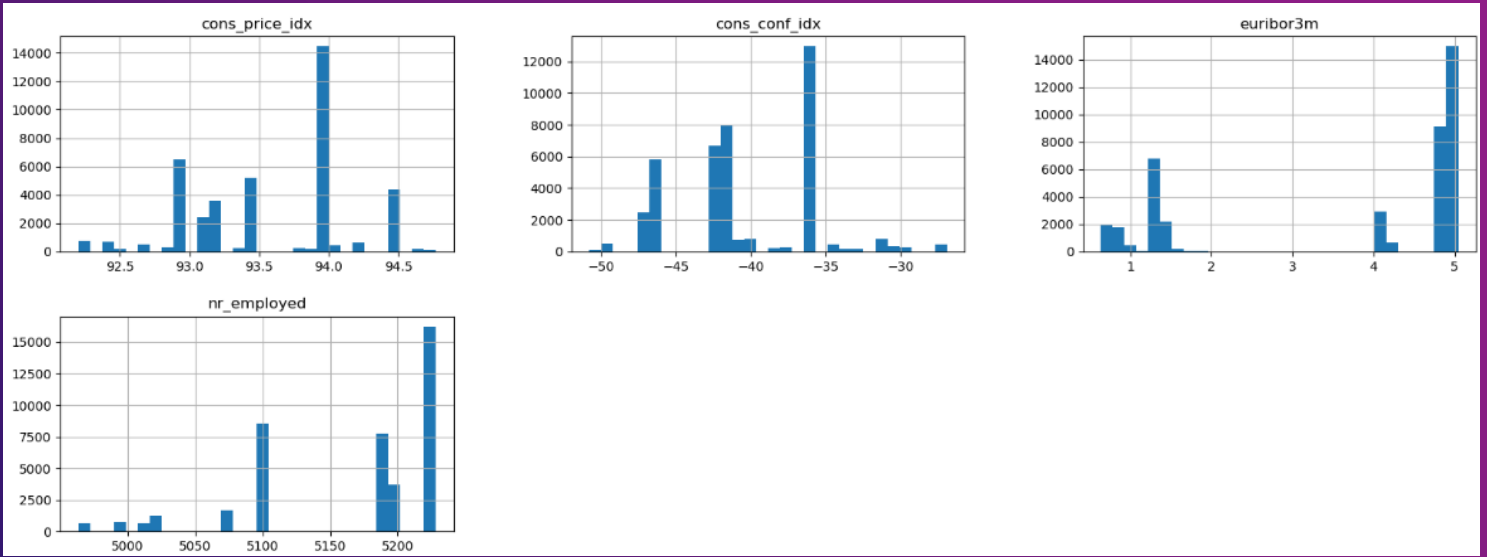
```
[4]: data = data.toDF(*[col.replace('.', '_').replace(' ', '_') for col in data.columns])
     data.show()
```

```
+---+----------+--------+------------------+-------+-------+----+---------+-----+-----------+--------+--------+-----+--------
+----------+-----------+--------------+-------------+---------+-----------+---+
|age|       job| marital|         education|default|housing|loan|  contact|month|day_of_week|duration|campaign|pdays|previous
|   poutcome|emp_var_rate|cons_price_idx|cons_conf_idx|euribor3m|nr_employed|  y|
+---+----------+--------+------------------+-------+-------+----+---------+-----+-----------+--------+--------+-----+--------
+----------+-----------+--------------+-------------+---------+-----------+---+
| 56|  housemaid| married|          basic.4y|     no|     no|  no|telephone|  may|        mon|     261|       1|  999|       0
|nonexistent|        1.1|        93.994|        -36.4|    4.857|     5191.0| no|
| 57|   services| married|       high.school|unknown|     no|  no|telephone|  may|        mon|     149|       1|  999|       0
|nonexistent|        1.1|        93.994|        -36.4|    4.857|     5191.0| no|
| 37|   services| married|       high.school|     no|    yes|  no|telephone|  may|        mon|     226|       1|  999|       0
|nonexistent|        1.1|        93.994|        -36.4|    4.857|     5191.0| no|
| 40|     admin.| married|          basic.6y|     no|     no|  no|telephone|  may|        mon|     151|       1|  999|       0
|nonexistent|        1.1|        93.994|        -36.4|    4.857|     5191.0| no|
| 56|   services| married|       high.school|     no|     no| yes|telephone|  may|        mon|     307|       1|  999|       0
|nonexistent|        1.1|        93.994|        -36.4|    4.857|     5191.0| no|
| 45|   services| married|          basic.9y|unknown|     no|  no|telephone|  may|        mon|     198|       1|  999|       0
|nonexistent|        1.1|        93.994|        -36.4|    4.857|     5191.0| no|
| 59|     admin.| married|professional.course|     no|     no|  no|telephone|  may|        mon|     139|       1|  999|       0
```

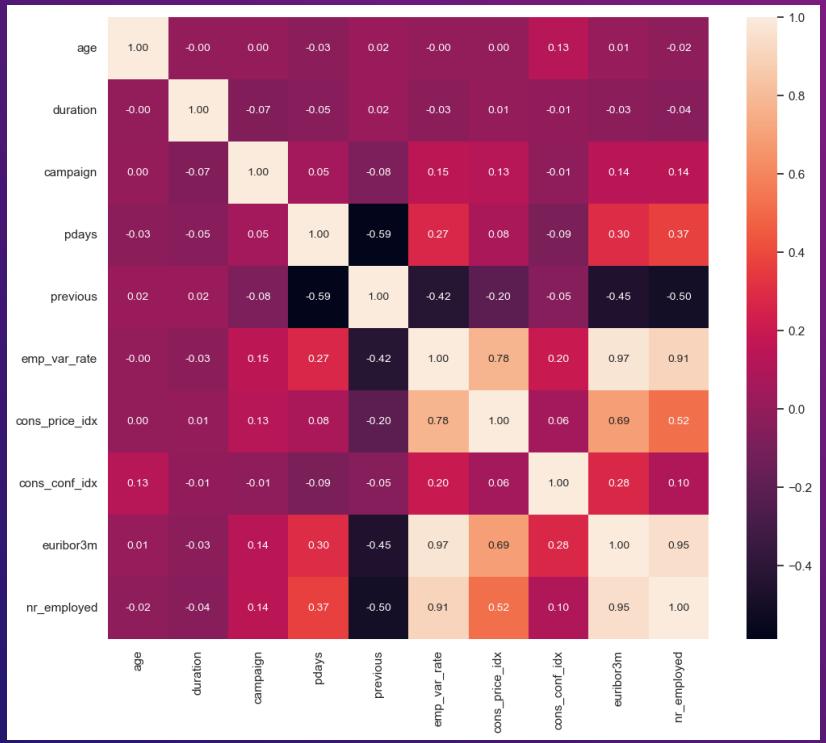
EXPLORATORY DATA ANALYSIS

1. Dataset presents a mix of numeric and categorical features.
2. Descriptive stats offer insights into feature distributions.
3. No data quality issues detected (no missing values).
4. EDA lays the foundation for deeper analysis and modeling.





- The histograms indicate demographic and economic data: age is centered on a middle-aged group, durations are mostly short, and campaign contacts are typically low with rare higher values.
- Economic indicators like the employment variation rate and consumer price index show common ranges, whereas consumer confidence is generally low.
- Interest rates (euribor3m) and employment numbers exhibit specific frequent values, suggesting common rates and employment figures within the dataset.



- The heat map displays correlations between variables, with dark red indicating strong positive correlations and dark purple indicating strong negative correlations.
- emp_var_rate has strong positive correlations with euribor3m and nr_employed, while pdays and previous have a strong negative correlation.
- Moderate positive correlations are seen between cons_price_idx and variables like emp_var_rate, euribor3m, and nr_employed.

CARDINALITY

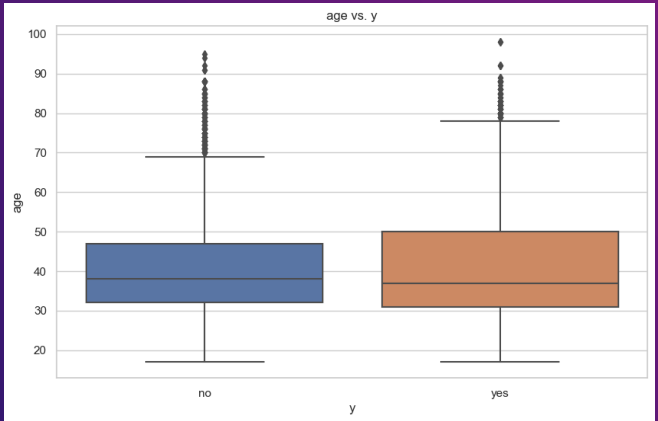
1. Variable Diversity: The table indicates the diversity in each column, with features like job having 12 unique values, suggesting a range of job categories in the dataset.
2. Categorical Insight: Categorical columns like marital with 4 unique values reveal the distribution of marital statuses, providing insights into the dataset's demographic composition.
3. Target Variable Format: The y column, likely representing the target variable, has 2 unique values, implying a binary classification task where instances are categorized as either yes or no, indicating a binary outcome or decision.

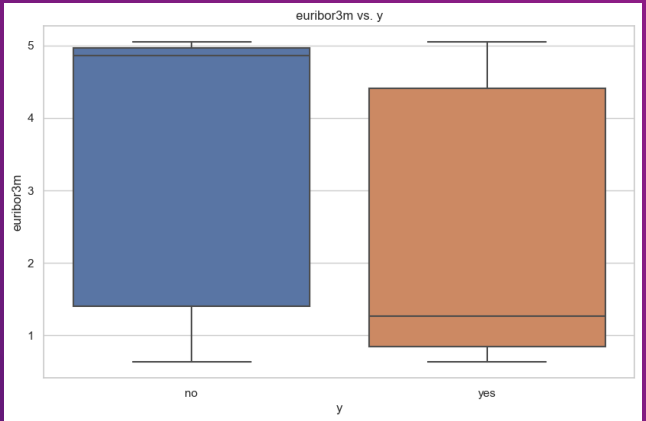
In [8]:

```
from pyspark.sql.functions import col, countDistinct

# Calculate cardinality for each column
cardinality_data = data.agg(*[countDistinct(col(c)).alias(c) for c in data.columns])

# Collect cardinality data to the driver for displaying
cardinality_data_pd = cardinality_data.toPandas()

# Display cardinality information
print("Cardinality of each column:")
print(cardinality_data_pd)
```

```
Cardinality of each column:
   age  job  marital  education  default  housing  loan  contact  month  \
0   78   12        4          8        3        3     3        2     10

   day_of_week  ...  campaign  pdays  previous  poutcome  emp_var_rate  \
0            5  ...        42     27         8         3            10

   cons_price_idx  cons_conf_idx  euribor3m  nr_employed  y
0              26             26        316           11  2

[1 rows x 21 columns]
```

BI-VARIATE ANALYSIS





- The first boxplot shows the age distribution for two groups labeled "yes" and "no," possibly indicating a response to a survey or outcome of an event. The "yes" group tends to be older than the "no" group, with a higher median age and more outliers at the upper age range.
- The second boxplot compares the 3-month Euribor interest rates between the same two groups. Both groups have similar medians, but the "yes" group has a slightly higher range of rates, indicating more variation in the interest rates for those who responded "yes."

MODELING 1 – RANDOM FOREST

The Random Forest model exhibits high predictive accuracy, as reflected by an Area Under the ROC Curve of 0.93. With 20 trees, the model is well-suited for binary classification (numClasses=2) on a dataset with 21 features. This ensemble learning approach combines predictions from multiple decision trees, enhancing overall performance and robustness in distinguishing between the two classes.

Random Forest

In [10]:

```
#####Random Forest
# Replace dots in column names with underscores
data = data.toDF(*[col.replace('.', '_') for col in data.columns])

# Define categorical and numerical columns
categorical_columns = ["job", "marital", "education", "default", "housing", "loan", "contact", "month", "day_of_week", "poutcome"
numerical_columns = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'eur

# Encode categorical columns
indexers = [StringIndexer(inputCol=column, outputCol=column + '_index', handleInvalid='keep') for column in categorical_columns]
encoders = [OneHotEncoder(inputCol=column + '_index', outputCol=column + '_encoded') for column in categorical_columns]

# Label encoding for target column "y"
label_encoder = StringIndexer(inputCol="y", outputCol="label")
```

```
Area Under ROC Curve: 0.931251448972968
Model Summary:
RandomForestClassificationModel: uid=RandomForestClassifier_5d1e7d52583e, numTrees=20, numClasses=2, numFeatures=21
```

MODELING 2 – XG BOOST

he XGBoost model demonstrates strong predictive performance with an Area Under the ROC Curve of 0.945. Utilizing 10 trees, this gradient boosting classification model is tailored for binary classification (numClasses=2) on a dataset with 21 features. XGBoost excels in enhancing predictive accuracy by sequentially improving upon the weaknesses of the previous models, making it a powerful algorithm for classification tasks.

XGBOOST

```
###XGBOOST

# Replace dots in column names with underscores
data = data.toDF(*[col.replace('.', '_') for col in data.columns])

# Define categorical and numerical columns
categorical_columns = ["job", "marital", "education", "default", "housing", "loan", "contact", "month", "day_of_week", "poutcome"
numerical_columns = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'euri

# Encode categorical columns
indexers = [StringIndexer(inputCol=column, outputCol=column + '_index', handleInvalid='keep') for column in categorical_columns]

# Label encoding for target column "y"
label_encoder = StringIndexer(inputCol="y", outputCol="label")

# Define the feature columns
feature_columns = ["age"] + [column + '_index' for column in categorical_columns] + numerical_columns
```

```
Area Under ROC Curve (XGBoost): 0.9459123002601263
XGBoost Model Summary:
GBTClassificationModel: uid = GBTClassifier_e3b4d8da8988, numTrees=10, numClasses=2, numFeatures=21
```

MODELING 3 – DECISION TREE

The decision tree model exhibits moderate predictive performance, reflected by an Area Under the ROC Curve of 0.575. With a depth of 5, this decision tree classification model consists of 35 nodes and is designed for binary classification (numClasses=2) based on a dataset with 21 features. Decision trees are known for their interpretability and ability to capture complex decision boundaries, although in this case, the model's discriminatory power may be limited compared to more sophisticated algorithms.

Decision Tree

```
]: ###Decision Tree


from pyspark.ml.classification import DecisionTreeClassifier

# Create a DecisionTreeClassifier for binary classification
dt = DecisionTreeClassifier(featuresCol="features", labelCol="label")

# Create a pipeline with the defined stages
pipeline_stages_dt = indexers + encoders + [label_encoder, feature_assembler, dt]
pipeline_dt = Pipeline(stages=pipeline_stages_dt)

# Fit the pipeline on the training data
model_dt = pipeline_dt.fit(training_data)
```

```
Area Under ROC Curve (Decision Tree): 0.487656699076622
Decision Tree Model Summary:
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_51c631330f3d, depth=5, numNodes=35, numClasses=2, numFeatures=21
```

MODELING 4 – GRADIENT BOOSTING

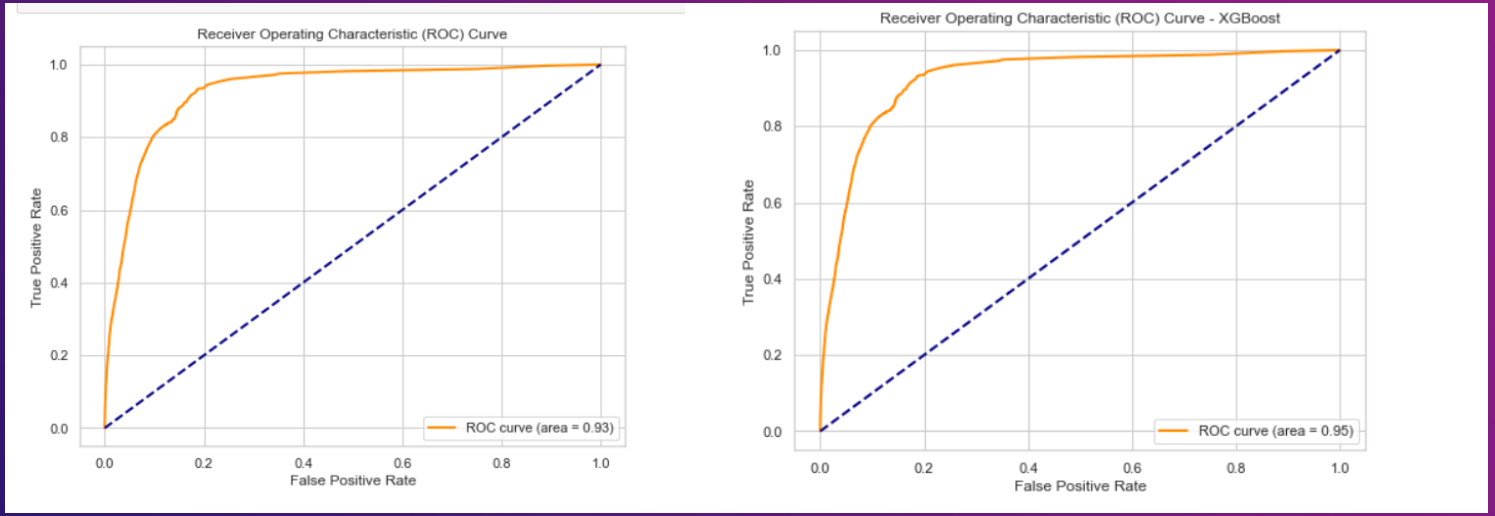
The Gradient Boosting model with 10 trees achieved strong predictive performance, evidenced by an impressive AUC of 0.945 in binary classification using a dataset with 21 features. This highlights the model's effectiveness in distinguishing between classes, showcasing its suitability for complex tasks.

Gradient Boosting

```
###Gradient Boosting

from pyspark.ml.classification import GBTClassifier
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Replace dots in column names with underscores
data = data.toDF(*[col.replace('.', '_') for col in data.columns])

# Define categorical and numerical columns
categorical_columns = ["job", "marital", "education", "default", "housing", "loan", "contact", "month", "day_of_week", "poutcome"
numerical_columns = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'euri
```

```
Area Under ROC Curve (GBT): 0.9452944413869016
GBT Model Summary:
GBTClassificationModel: uid = GBTClassifier_269bd53c8d82, numTrees=10, numClasses=2, numFeatures=21
```

TOP 2 MODEL COMPARISON



Gradient boost

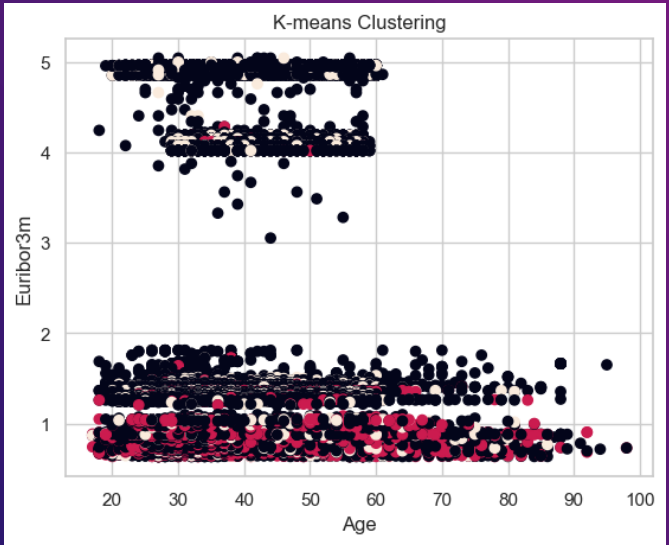
XG-boost

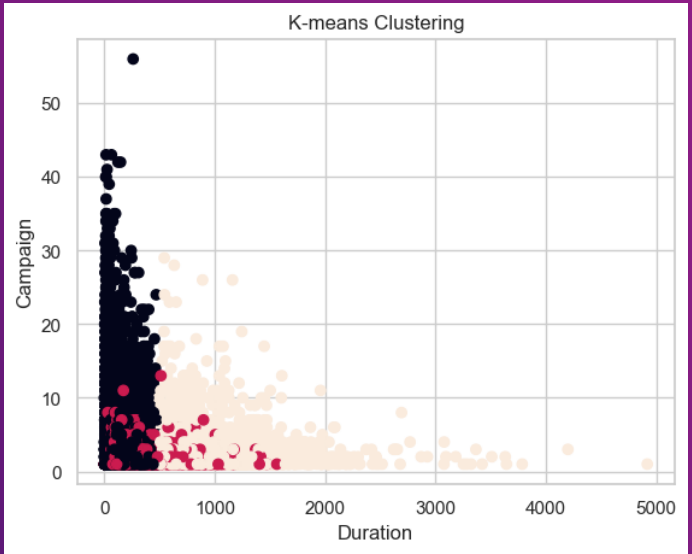
MODEL COMPARISON AND SELECTION

The champion model for the task is the Gradient Boosting (GBT) model, which achieved an impressive Area Under the ROC Curve of 0.945. This indicates strong predictive performance and suggests that the GBT model effectively discriminates between the two classes in the dataset. The high AUC value makes the GBT model the preferred choice among the models considered, showcasing its capability as the most effective classifier for the given task.

```
Champion Model: XGBoost
Area Under ROC Curve (Champion Model): 0.9459123002601263
```

K-MEANS CLUSTERING





EXPORTING PICKLE FILE

```
6]: xgboost.save("/Users/tharunponnaganti/Downloads/GBTClassifierModelPickleFile2")

7]: #Loading above model

    pipelineModel = xgboost.load("/Users/tharunponnaganti/Downloads/GBTClassifierModelPickleFile2")
```

"Saved the best-performing model to the specified directory in Pickle format."

RESULTS

- **Objective:** Identify clients likely to subscribe to a term deposit at XYZ Bank.
- Conducted Exploratory Data Analysis (EDA) on data from May 2008 to November 2010, including 20 columns, focusing on the bank's marketing campaign data.
- The data required multiple contacts per client to assess term deposit subscription interest. Developed a predictive model to classify clients into 'yes' or 'no' categories for term deposit subscriptions.
- The XGBoost model emerged as the most effective, with an Area Under the ROC Curve (AUC) score of 0.9463.

- **Conclusion**:

  -The XGBoost model is robust and accurate for predicting client subscription to term deposits.

  -The model is valuable for optimizing future marketing strategies and targeting potential customers.

  -Insights from EDA can refine the bank's approach to client interactions and understanding of key decision factors.

REFERENCES

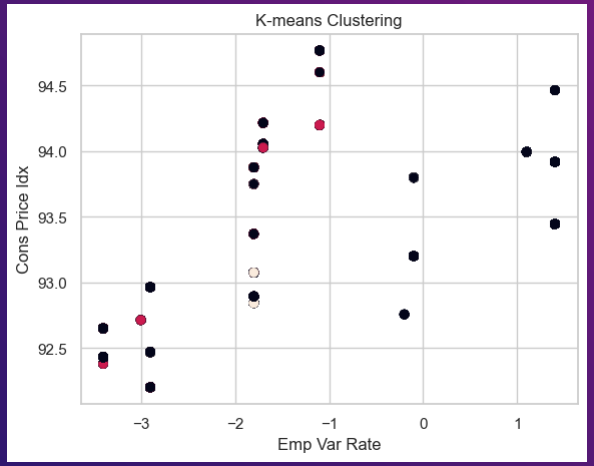
https://chat.openai.com/

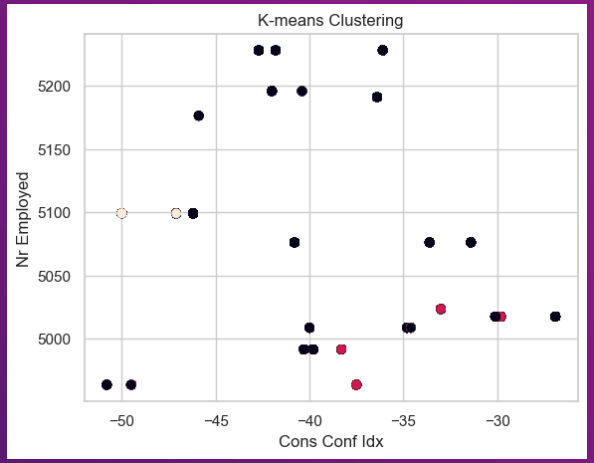
https://hadoop.apache.org/releases.html

https://spark.apache.org/

https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

APPENDIX


K-means Clustering
Cons Price Idx
94.5
94.0
93.5
93.0
92.5
−3
−2
−1
0
1
Emp Var Rate


K-means Clustering
Nr Employed
5200
5150
5100
5050
5000
−50
−45
−40
−35
−30
Cons Conf Idx

K-means Clustering is optional

[21]:

```
####K-means Clustering is optional

from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline

# Select only numeric columns for clustering
numerical_columns_cluster = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate', 'cons_price_idx', 'cons_conf

# Create a VectorAssembler to combine feature columns
feature_assembler_cluster = VectorAssembler(inputCols=numerical_columns_cluster, outputCol="features")

# Define the K-means model with the desired number of clusters (k)
kmeans = KMeans(featuresCol="features", k=3, seed=42)

# Create a pipeline with the defined stages for clustering
pipeline_stages_cluster = [feature_assembler_cluster, kmeans]
pipeline_cluster = Pipeline(stages=pipeline_stages_cluster)

# Fit the pipeline on the data
model_cluster = pipeline_cluster.fit(data)

# Make predictions on the data to get cluster assignments
```