TOOLS & TECHNIQUES FOR DATA SCIENCE

Final Course Project Report

Bank Marketing Campaign - Term Deposit Subscription Prediction

Submitted by: Muhammad Sheryar

Roll No: MSDS25064

Dataset: Kaggle - Bank Marketing Dataset

Table of Contents

1. Introduction

This project focuses on predicting whether a bank customer will subscribe to a term deposit based on historical marketing campaign data. The dataset originates from a Portuguese banking institution and is publicly available on Kaggle.

The project includes:

- Data loading
- Data cleaning and preprocessing
- Exploratory Data Analysis (EDA)
- Visualizations
- Machine learning models
- Git usage for version control
- Final evaluation and insights

All development work was completed in Python using colab, and the project code repository is publicly available on GitHub.

2. Data Loading & Dataset Description

What was done:

The dataset was downloaded from Kaggle (Bank Marketing Dataset) and loaded into Python using pandas.

Key Actions Performed

- Imported necessary libraries

- Loaded CSV files using `pd.read_csv()`
- Displayed first few rows using `.head()`
- Identified column names and data types

Dataset Overview

The dataset contains **45,000+ marketing contact records**, each representing an interaction with a customer.

Key attributes include:

- **Demographic features:** age, job, marital, education
- **Financial indicators:** housing loan, personal loan, balance
- **Campaign features:** contact type, campaign duration, number of attempts
- **Economic variables:** employment variation rate, consumer price index
- **Target variable:** `y` (yes/no — subscription outcome)

```
data = pd.read_csv('/content/drive/MyDrive/Bank Data/bank/bank-full.csv', sep=';')
df = data.copy()
df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

Exploratory Data Analysis (EDA):

What was done:

EDA was performed to understand the structure, distribution, and relationships in the dataset.

Key EDA Steps:

1. Summary Statistics

- Used .describe() to view numerical statistics
- Identified outliers and skewed features

2. Missing Values Check

- Used .isnull().sum()
- Confirmed dataset has no missing values (common in this dataset)

3. Categorical Value Distribution

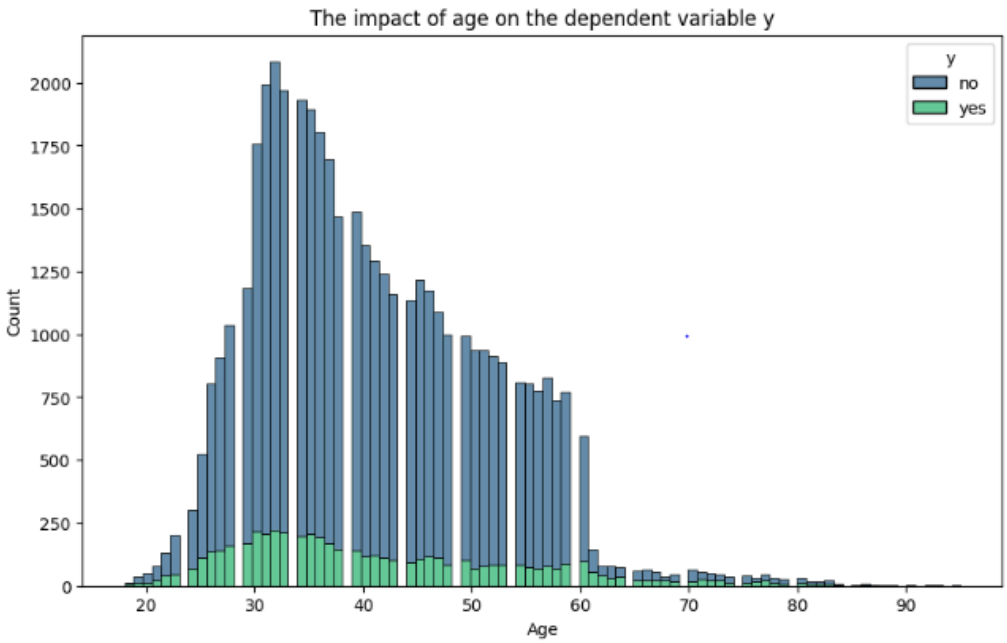
- Value counts for job, marital status, education
- Observed imbalance in categories such as “blue-collar” and “admin.”

4. Target Variable Analysis

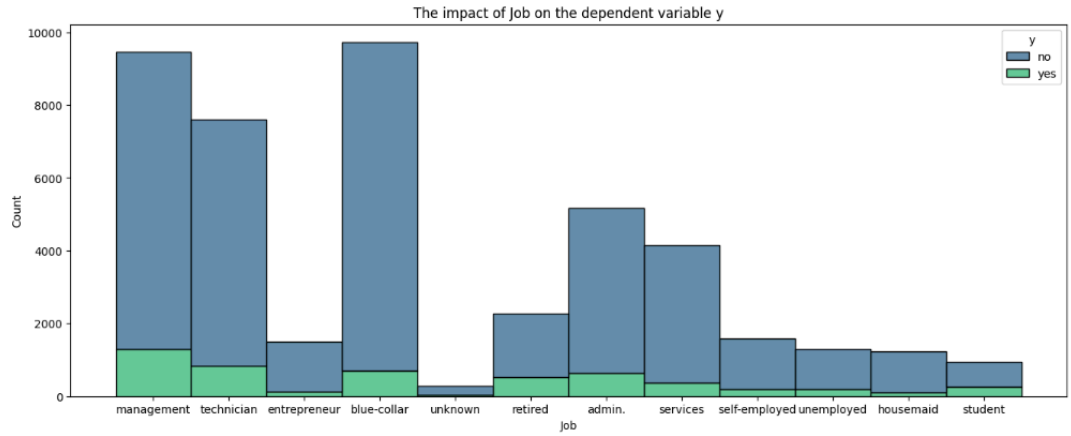
- Class imbalance detected (large number of “no” vs “yes”)

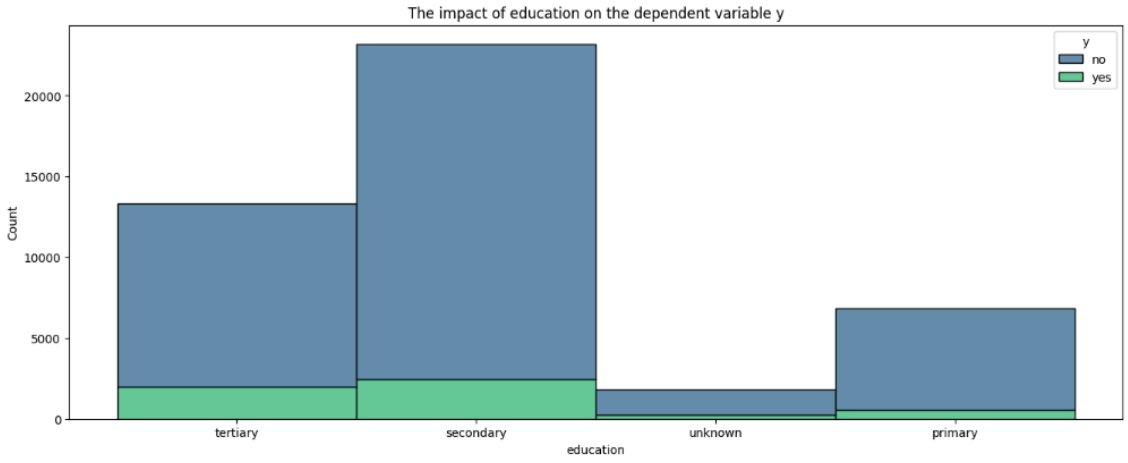
```
df.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

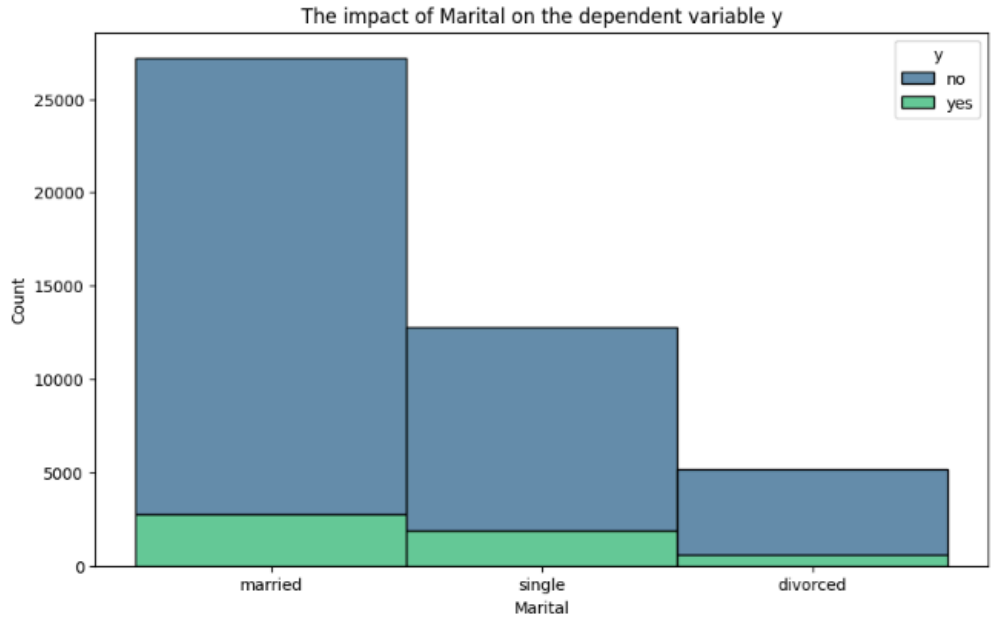


```
plt.show()
```

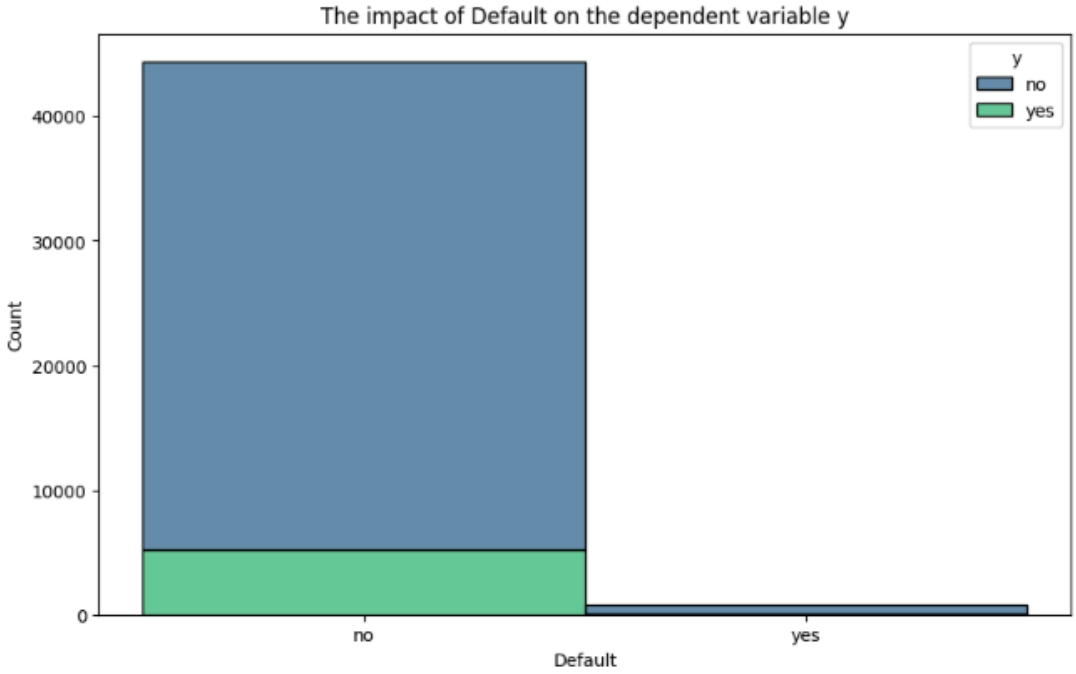

The impact of Job on the dependent variable y
y
no
yes
Count
10000
8000
6000
4000
2000
0
management
technician
entrepreneur
blue-collar
unknown
retired
admin.
services
self-employed
unemployed
housemaid
student
Job


The impact of education on the dependent variable y
y
no
yes
Count
20000
15000
10000
5000
0
tertiary
secondary
unknown
primary
education

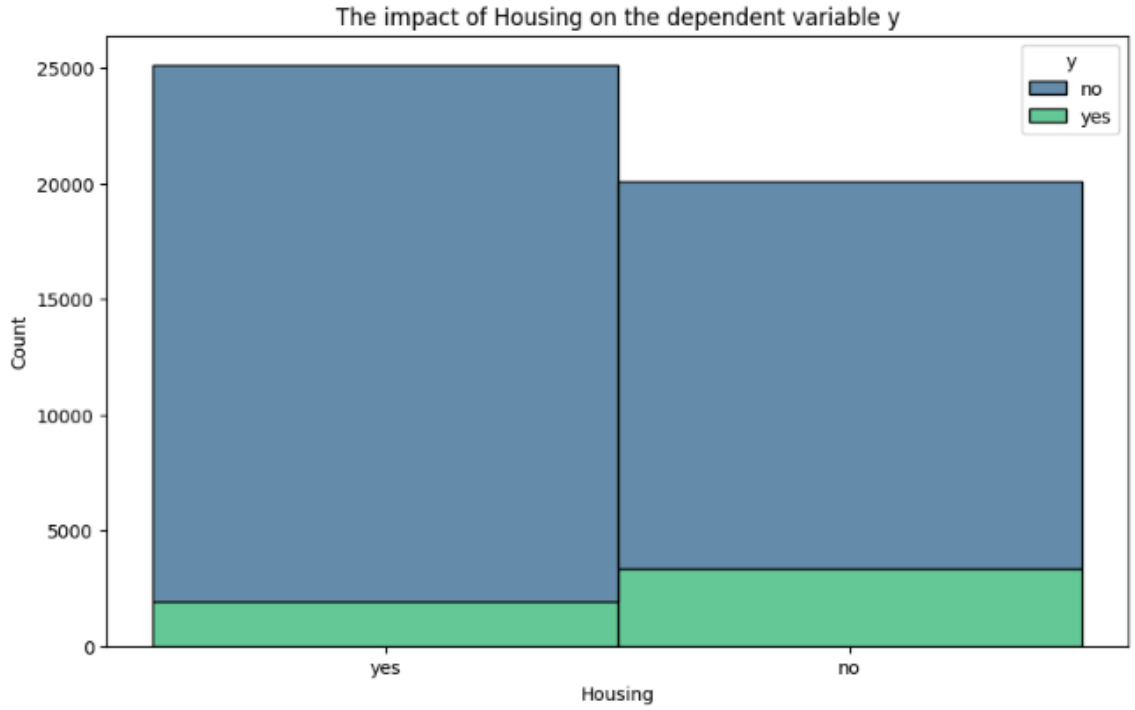
```
Text(0, 0.5, 'Count')
```



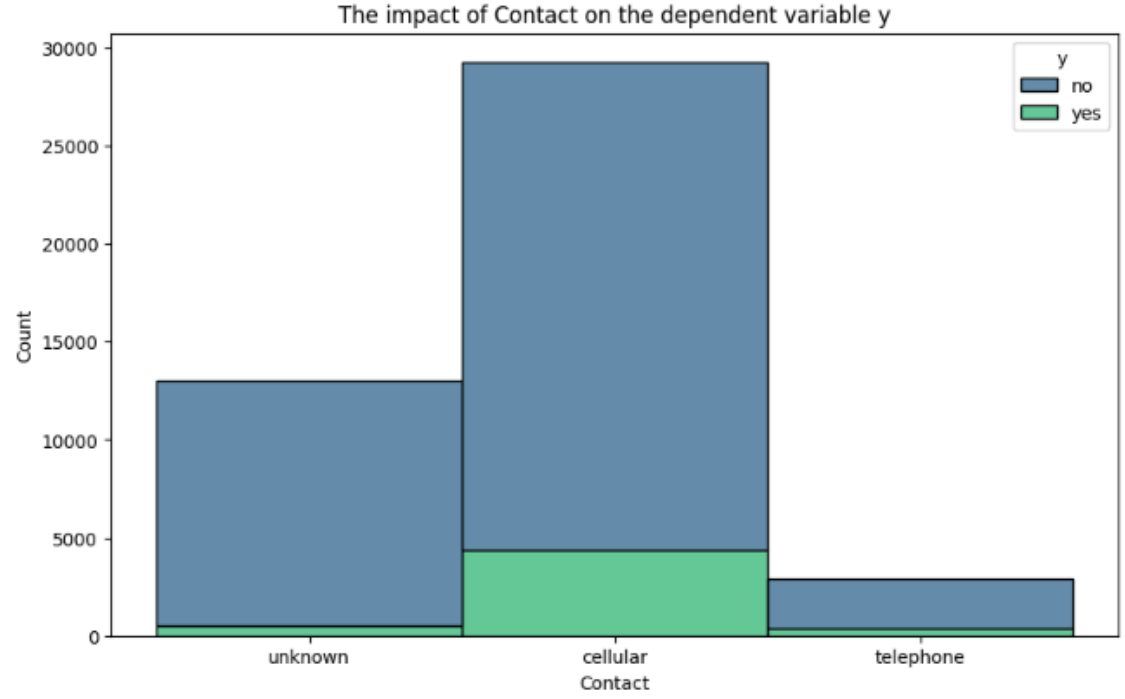

```
Text(0, 0.5, 'Count')
```



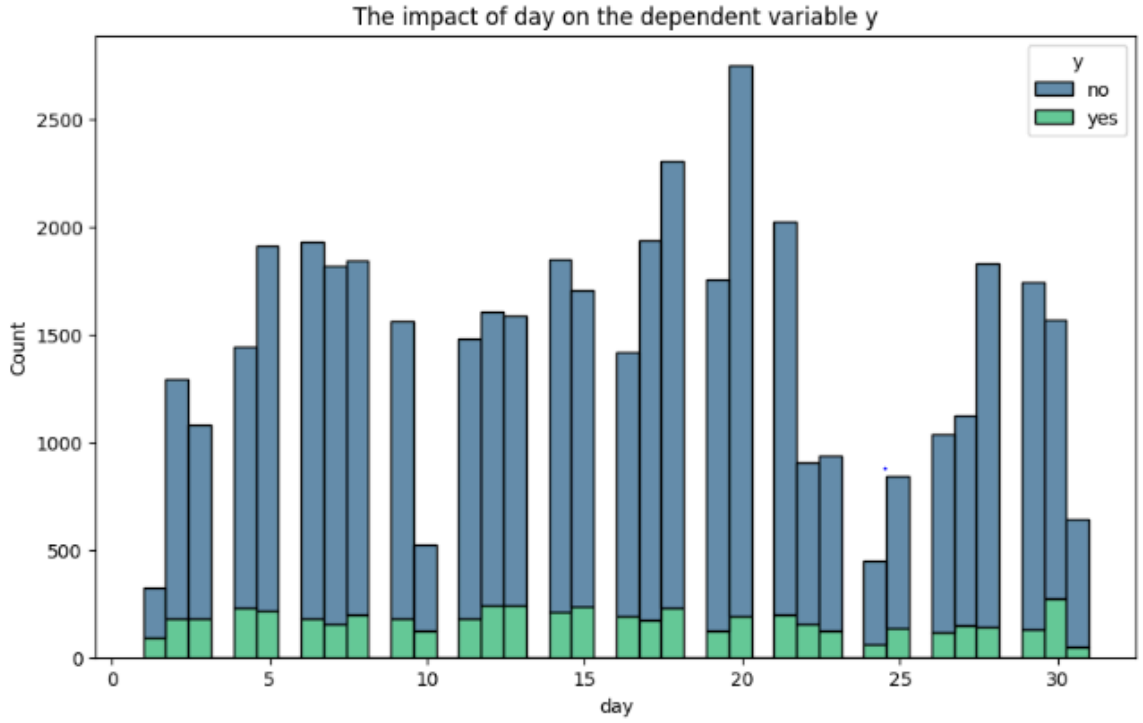
```
Text(0, 0.5, 'Count')
```



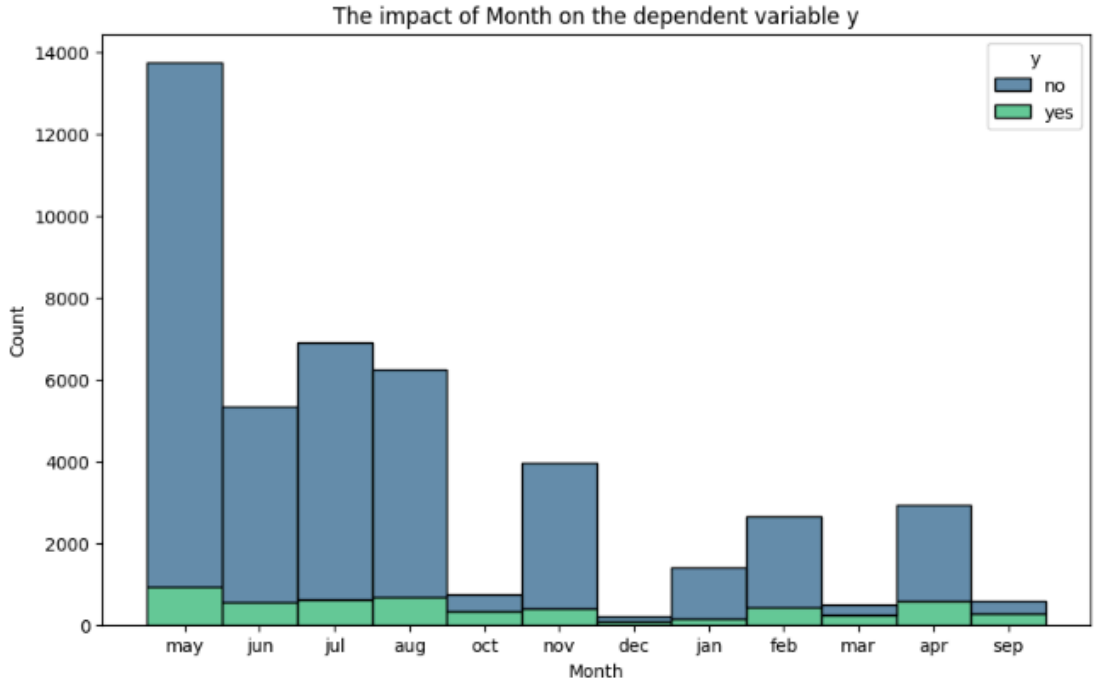

```
Text(0, 0.5, 'Count')
```



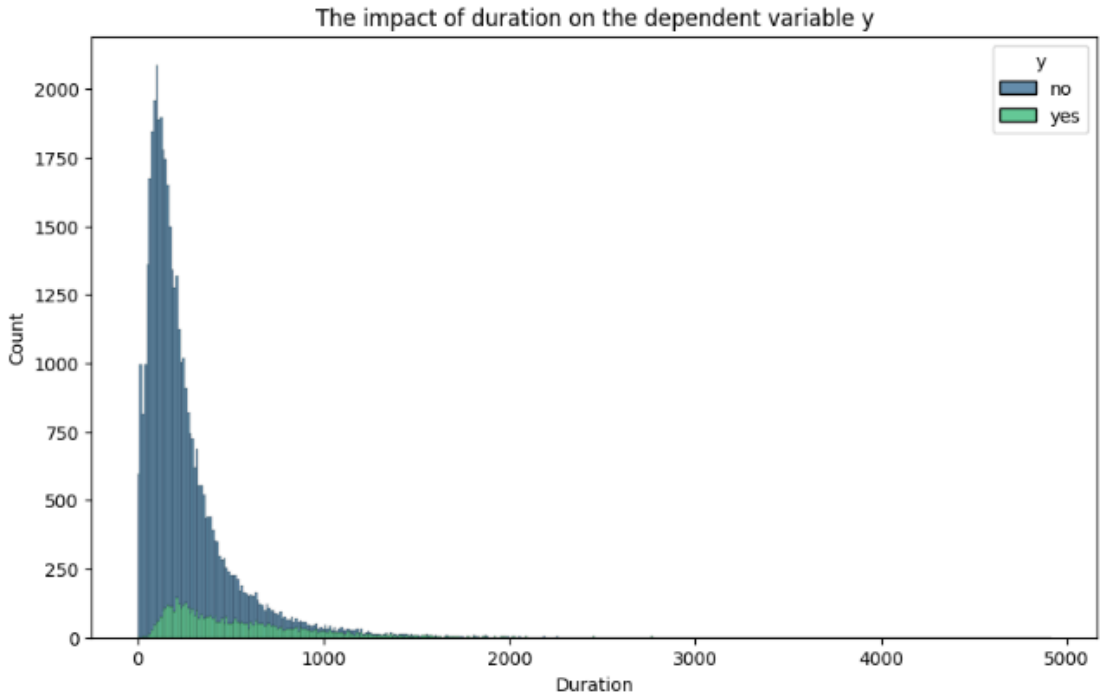
```
Text(0, 0.5, 'Count')
```



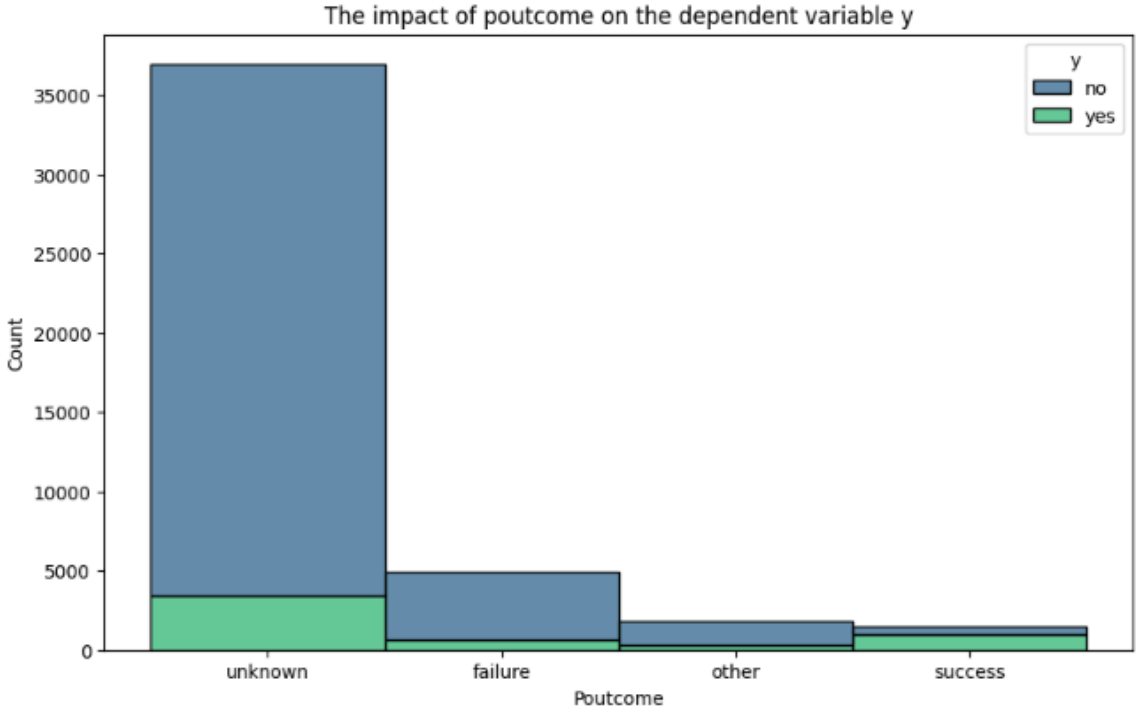

```
Text(0, 0.5, 'Count')
```

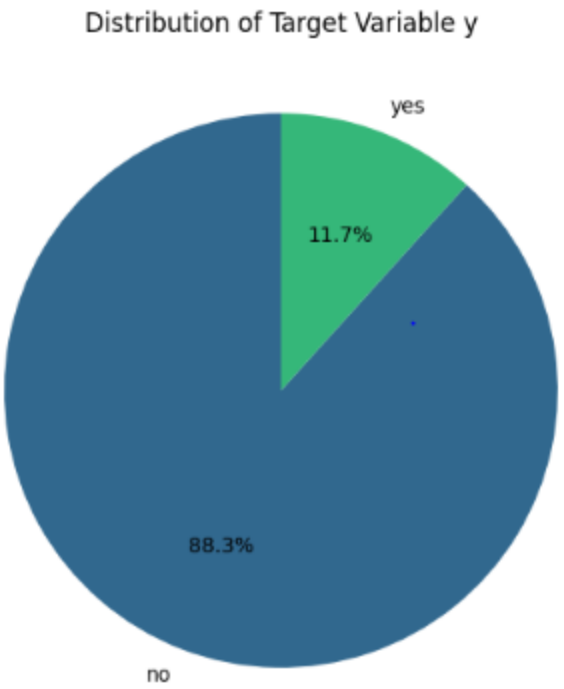


```
Text(0, 0.5, 'Count')
```




```
Text(0, 0.5, 'Count')
```





Data Wrangling & Cleaning

What was done:

Data preprocessing was applied to prepare the dataset for machine learning.

Steps Performed:

1. Removing Irrelevant Columns

- Removed columns like duration (not allowed for prediction)

2. Handling Categorical Variables

- Applied One-Hot Encoding for categorical features
- Used pd.get_dummies() or OneHotEncoder()

3. Feature Scaling (Optional)

- Scaled numerical data using StandardScaler where needed

4. Train-Test Split

- Split dataset into X_train, X_test, y_train, y_test
- Used 80/20 split ratio

- Encoding code

```
# Apply Label Encoding to age_group
label_encoder = LabelEncoder()
X_train['age_group'] = label_encoder.fit_transform(X_train['age_group'])
X_test['age_group'] = label_encoder.transform(X_test['age_group'])

# Check the mapping
print("Age group mapping:", dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))
```

```
Age group mapping: {'25-34': 0, '35-44': 1, '45-54': 2, '55-64': 3, '65+': 4, '<25': 5}
```

```
# Apply One-Hot Encoding to age_group
encoder = ColumnTransformer(transformers=[('cat', OneHotEncoder(drop='first'), ['age_group'])], remainder='passthrough')

# Transform X_train and X_test
X_train_encoded = encoder.fit_transform(X_train)
X_test_encoded = encoder.transform(X_test)
```

- Train-test split code

```
X = df.drop('y', axis=1)
y = df['y']
```

Train & Test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(34554, 17) (8639, 17) (34554,) (8639,)
```

```
# Check for non-numeric columns in X_train
non_numeric_columns = X_train.select_dtypes(include=['object', 'category']).columns
print("Non-numeric columns:", non_numeric_columns)
```

```
Non-numeric columns: Index(['age_group'], dtype='object')
```

- Cleaned dataset preview

```
# Save the processed data to the specified path
output_path = '/content/drive/MyDrive/Bank Data/bank/bankgraph.csv'
df.to_csv(output_path, index=False)
print(f"Processed CSV saved to: {output_path}")
```

```
Processed CSV saved to: /content/drive/MyDrive/Bank Data/bank/bankgraph.csv
```

```
df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	4	0	2	0	2143	1	0	2	5	4	261	1	-1	0	3	0
1	44	9	1	1	0	29	1	0	2	5	4	151	1	-1	0	3	0
2	33	2	0	1	0	2	1	1	2	5	4	76	1	-1	0	3	0
5	35	4	0	2	0	231	1	0	2	5	4	139	1	-1	0	3	0
6	28	4	1	2	0	447	1	1	2	5	4	217	1	-1	0	3	0

Build Multiple Charts & Explain Observations

Charts Created:

1. Histogram of Age

- Shows majority customers fall between ages 30–50

2. Bar Chart of Job Types

- Admin and blue-collar roles dominate the dataset

3. Correlation Heatmap

- Features such as “duration” and “previous outcome” show strong influence

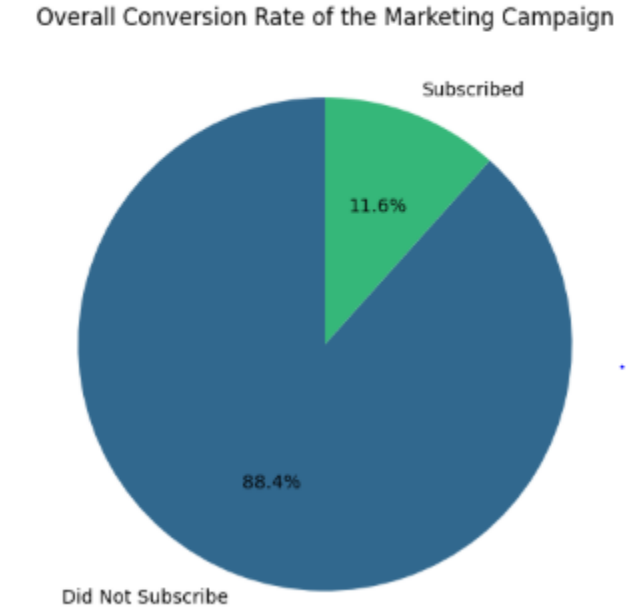
4. Subscription Rate Plot

- Shows low conversion rate (class imbalance)

Insights:

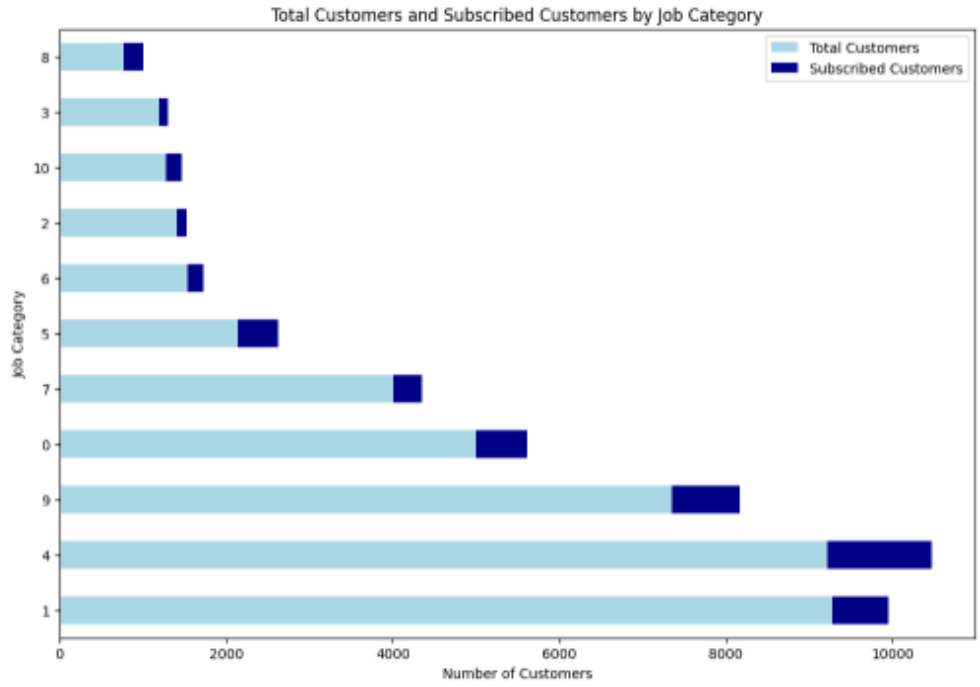
- Older individuals are more likely to subscribe
- Customers contacted via cellular links respond better
- Higher previous success rate increases likelihood of subscription

```
Total Customers: 43193
Subscribed Customers: 5021
Conversion Rate: 11.62%
```

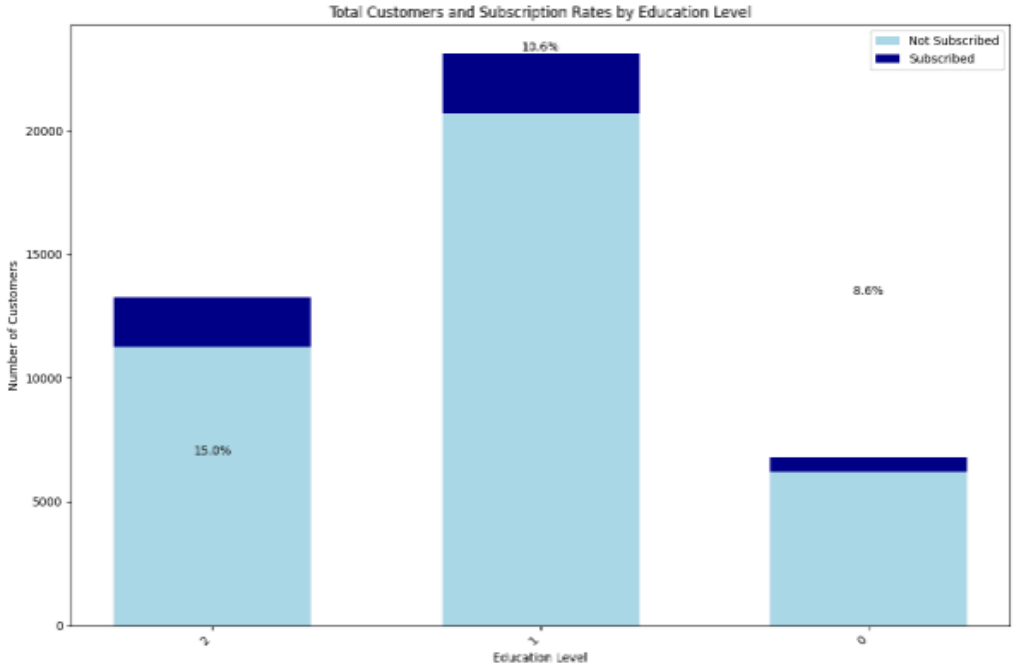


age_group	total_customers	subscribed_customers	subscription_rate
<25	729	182	24.965706
25-34	13852	1725	12.453075
35-44	13968	1337	9.571879
45-54	9372	869	9.272300
55-64	4549	599	13.167729
65+	723	309	42.738589

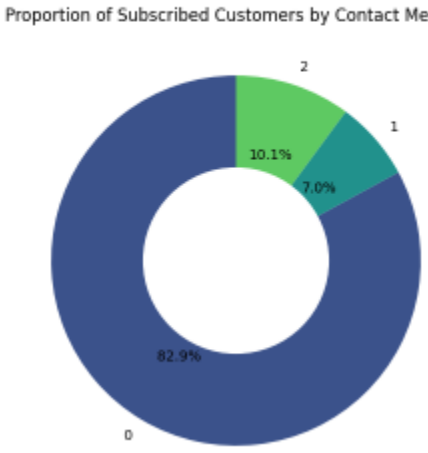
job	total_customers	subscribed_customers	subscription_rate
8	775	226	29.161290
5	2145	486	22.657343
10	1274	198	15.541601
4	9216	1253	13.595920
0	5000	613	12.260000
6	1548	182	11.818182
9	7355	817	11.108090
3	1195	105	8.786611
7	4004	350	8.741259
2	1411	116	8.221120
1	9278	675	7.275275

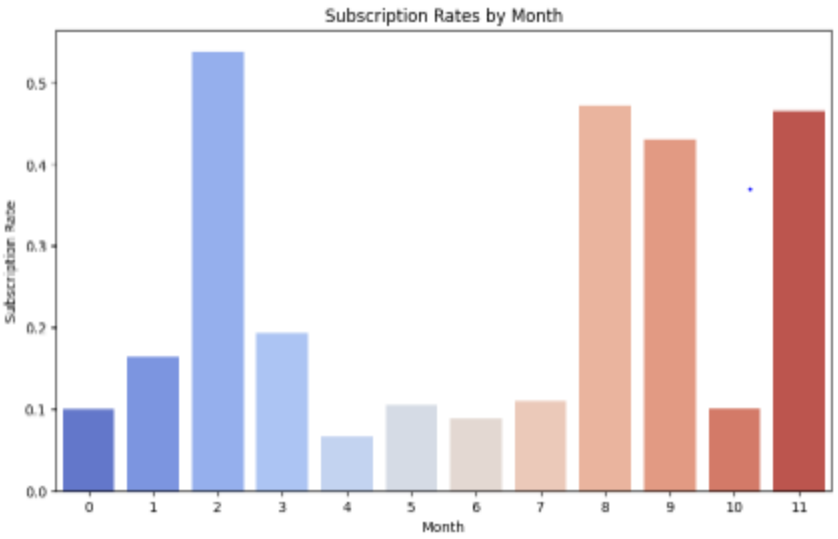


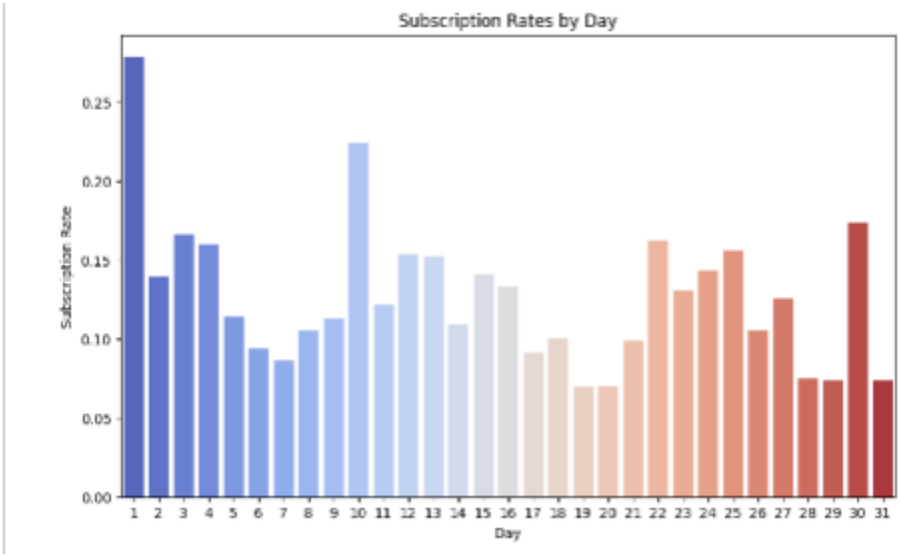
education	total_customers	subscribed_customers	subscription_rate
2	13262	1992	15.020359
1	23131	2441	10.552938
0	6800	588	8.647059



contact	total_customers	subscribed_customers	subscription_rate
0	28213	4163	14.755609
1	2694	350	12.991834
2	12286	508	4.134788




Subscription Rates by Month
Subscription Rate
Month
0.0
0.1
0.2
0.3
0.4
0.5
0 1 2 3 4 5 6 7 8 9 10 11


Subscription Rates by Day
Subscription Rate
Day
0.00
0.05
0.10
0.15
0.20
0.25
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31


Subscription Rate vs. Number of Contacts (Campaign)
Subscription Rate
Number of Contacts (Campaign)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0 10 20 30 40 50 60

Use of Git

What was done:

Git was used for version control throughout the project.

Git Steps Completed:

Initialized repository

1. git init

Staged project files

2. git add .

Committed changes

3. git commit -m "Initial commit - Bank Marketing Prediction Project"

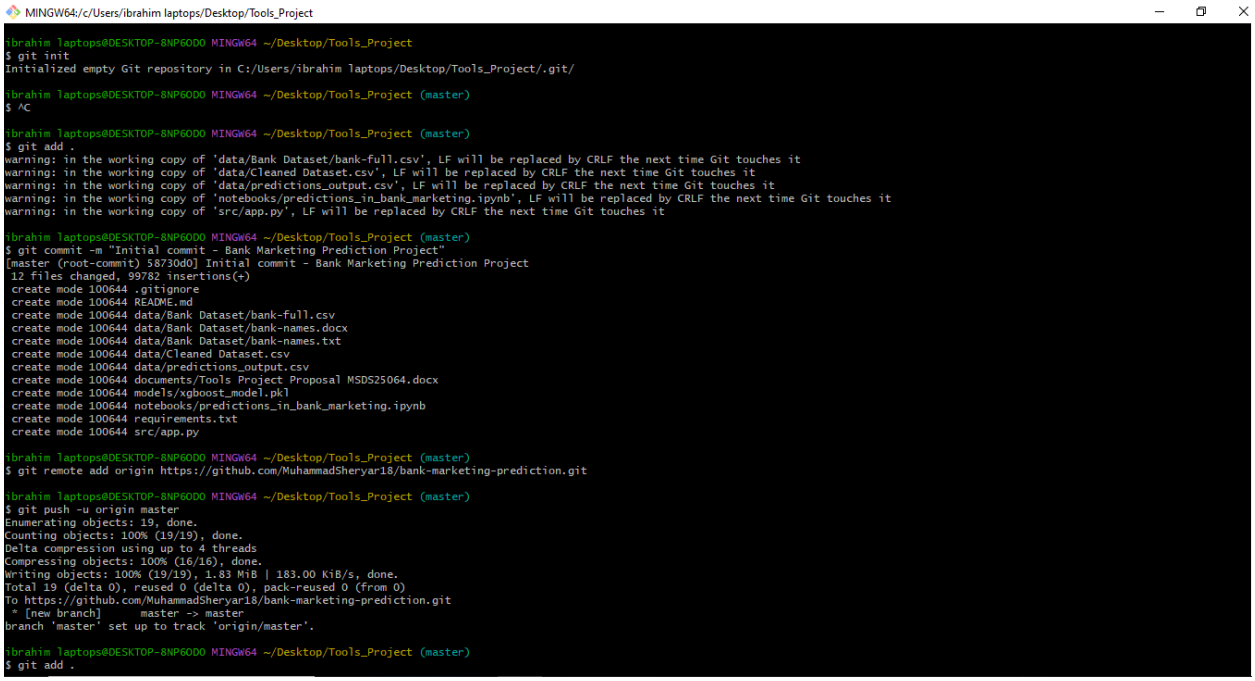
Linked GitHub remote

4. git remote add origin https://github.com/MuhammadSheryar18/bank-marketing-prediction.git

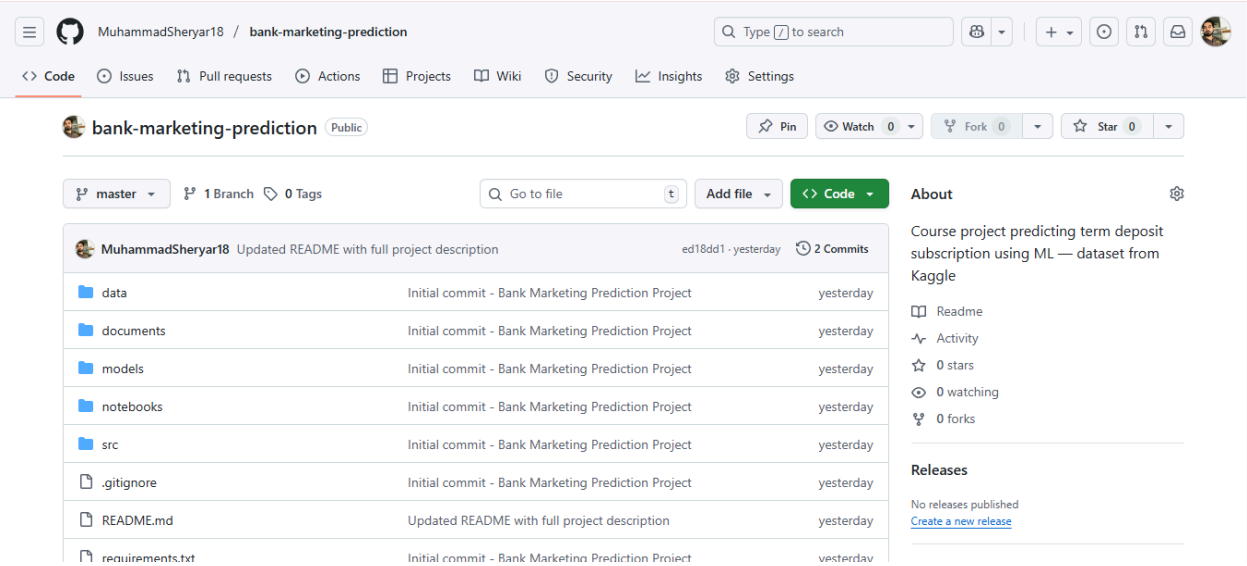
Pushed to GitHub

5. git push -u origin master

- Git Bash window (init, add, commit, push)



- GitHub repository screenshot



Machine Learning Techniques

Models Implemented:

1. Logistic Regression

- Baseline model
- Good for interpretability

2. Random Forest Classifier

- Handles nonlinear patterns
- Higher accuracy than logistic regression

3. XGBoost Classifier

- Best performing model
- Captured complex interactions
- Saved as xgboost_model.pkl

Performance Evaluation:

- Confusion Matrix
- Accuracy Score
- ROC-AUC Score
- Precision, Recall

- Model training code

LogisticRegression model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create the LogisticRegression model
logistic_regression = LogisticRegression(max_iter=1000, random_state=42)

# Fit the model to the training data
logistic_regression.fit(X_train_encoded, y_train)

# Predict the target values for the test set
y_pred_lr = logistic_regression.predict(X_test_encoded)

# Evaluate the model
accuracy_lr = accuracy_score(y_test, y_pred_lr)
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)
class_report_lr = classification_report(y_test, y_pred_lr)

print("Logistic Regression Accuracy:", accuracy_lr)
print("Confusion Matrix:\n", conf_matrix_lr)
print("Classification Report:\n", class_report_lr)
```

```
Logistic Regression Accuracy: 0.892927422155342
Confusion Matrix:
 [[7508  150]
 [ 775  206]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      7658
           1       0.58      0.21      0.31       981

    accuracy                           0.89      8639
   macro avg       0.74      0.60      0.63      8639
weighted avg       0.87      0.89      0.87      8639

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

xgboost model

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    use_label_encoder=False,
    eval_metric="logloss",
    random_state=42
)
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred = xgb_model.predict(X_test)

# Evaluate performance
from sklearn.metrics import accuracy_score, classification_report
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [20:23:46] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Accuracy: 0.9127213797893274
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      7658
           1       0.65      0.50      0.57       981

    accuracy                           0.91      8639
   macro avg       0.79      0.73      0.76      8639
weighted avg       0.91      0.91      0.91      8639
```

Random Forest Classifier model

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
y_pred_rf = model.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_accuracy
```

```
0.9062391480495428
```

- Confusion matrix & Accuracy results

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      7658
           1       0.58      0.21      0.31       981

    accuracy                           0.89      8639
   macro avg       0.74      0.60      0.63      8639
weighted avg       0.87      0.89      0.87      8639

Random Forest Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      7658
           1       0.64      0.40      0.49       981

    accuracy                           0.91      8639
   macro avg       0.78      0.69      0.72      8639
weighted avg       0.89      0.91      0.90      8639

XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      7658
           1       0.65      0.50      0.57       981

    accuracy                           0.91      8639
   macro avg       0.79      0.73      0.76      8639
weighted avg       0.91      0.91      0.91      8639
```