Assigment

**Learning Outcome Addressed**

1. Apply various classification methods to a business problem.
2. Compare the results of k-nearest neighbors, logistic regression, decision trees, and support vector machines. (done)

***This is a required assignment and counts toward program completion.***

**Overview**

In this third practical application assignment, your goal is to compare the performance of the classifiers (k-nearest neighbors, logistic regression, decision trees, and support vector machines) you encountered in this section of the program. You will use a dataset related to the marketing of bank products over the telephone.

**Data**

The dataset you will use comes from the <https://archive.ics.uci.edu/dataset/222/bank+marketing> [UC Irvine Machine Learning Repository Links to an external site.](https://archive.ics.uci.edu/ml/datasets/bank+marketing). The data is from a Portuguese banking institution and is a collection of the results of multiple marketing campaigns. Please see the following link to access the assignment:

**Deliverables**

After understanding, preparing, and modeling your data, **build a Jupyter Notebook that includes a clear statement demonstrating your understanding of the business problem,** a **correct and concise interpretation of descriptive and inferential statistics, your findings (including actionable insights), and next steps and recommendations.**

**Execution**

#1. Understand the Business Problem

The dataset relates to the marketing campaigns of a Portuguese banking institution https://archive.ics.uci.edu/dataset/222/bank+marketing. The objective is to predict whether a client will subscribe to a term deposit (yes or no). Binary classification problem where the target variable is y (yes/no).

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

#preprocessing

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score #calculate the mean squared error between the true and predicted values in a regression problem.

from sklearn.datasets import make\_regression #generate synthetic regression problem for testing

import matplotlib.pyplot as plt #plot graphs

import seaborn as sns #plot based on matpolit

sns.set()

%matplotlib inline

#used in Jupyter notebooks to display Matplotlib plots inline.

# evaluation

from sklearn.metrics import f1\_score

from sklearn.metrics import accuracy\_score

# classifier we will use

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

# model selection bits

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV, train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedShuffleSplit, StratifiedKFold

from sklearn.model\_selection import learning\_curve, validation\_curve

# evaluation

from sklearn.metrics import f1\_score

# plotting

#from plotting import plot\_learning\_curve, plot\_validation\_curve

print("Marketing Campaing Subscription Binary kick-off!")

#Assess the Importance of Each Column

#There are 16 features (ALL COLUMNS, EXCEPT Y THE TARGET VARIABLE), Target Variable: y (binary: "yes" or "no").

#Critical Columns:

    # Age: Influence the likelihood of subscription (e.g., younger vs. older demographics).

    # Job: Job type indicates financial stability and potential interest in term deposits.

    # Marital status: may affect financial goals and savings patterns.

    # Education level: may correlate with financial literacy and investment preferences.

    # Balance: Higher balance may indicate financial capability to subscribe to term deposits.

    # Housing: Owning a house could influence saving and investment behavior.

    # Loan: Having an active loan might deter clients from subscribing to additional products.

    # Contact:  Communication type (e.g., cellular, telephone) may affect outreach effectiveness.

    # Duration: Length of the last call is often a strong indicator of success (highly predictive).

    # Campaign: Number of contacts during the current campaign can affect client behavior.

    # Pdays Days: since the client was last contacted may influence response behavior.

    # Previous: Number of contacts in previous campaigns indicates familiarity with the client.

    # poutcome: Outcome of the previous campaign gives insights into the client’s past behavior.

#Less Critical Columns:

    # Day:The specific day of the month might not have a significant impact on subscription likelihood.

    # Month: While the month may have some seasonal effects, it’s typically less critical compared to other variables.

#Predictor Columns:

    # Duration: The length of the last contact is highly predictive, as longer conversations often indicate interest.

    # Poutcome: The result of the previous marketing campaign (e.g., success or failure) often correlates with current subscription likelihood.

    # Pdays: How recently the client was contacted might influence their likelihood to subscribe.

    # Campaign: The number of contacts during the current campaign could indicate either engagement or annoyance.

    # Balance: Higher balances often correlate with financial capability to subscribe.

    # Job and education: Socioeconomic factors like job and education level can indicate interest and capacity.

    # Housing and loan: Indicators of financial commitments that may influence decisions.

# Use correlation analysis and feature importance techniques (e.g., logistic regression coefficients, decision tree feature importance, or SHAP values) to validate and quantify the significance of features.

# Perform feature selection to include only the most impactful predictors in your model.

#ENCODE

# Handling Categorical Variables

#Since the dataset contains categorical features (e.g., job, marital, education), need to encode them into numerical values before modeling. Popular methods include:

#One-Hot Encoding for nominal categories.

#Label Encoding for ordinal categories.

#RESULTS:

# X

    # 42 columns after one-hot encoding

    #Numerical Features:

        #age, balance, day, duration, campaign, pdays, previous

    #Categorical Features (One-Hot Encoded):

        #Jobs: job\_blue-collar, job\_entrepreneur, ..., job\_unknown

        #Marital status: marital\_married, marital\_single

        #Education: education\_secondary, education\_tertiary, education\_unknown

        #Default status: default\_yes

        #Housing loan: housing\_yes

        #Personal loan: loan\_yes

        #Contact type: contact\_telephone, contact\_unknown

        #Months: month\_aug, month\_dec, ..., month\_sep

        #Previous campaign outcome: poutcome\_other, poutcome\_success, poutcome\_unknown

# Y

    #is the target variable representing whether a client subscribed to a term deposit:

    #0: No

    #1: Yes

#ADDITIONAL ENCODING FOR BETTER ANALYSIS

# the correlation matrix calculation requires numerical data, but this dataset likely contains categorical variables that are still represented as strings (e.g., 'no', 'yes', job categories, etc.).

# To compute the correlation matrix, all columns in dataset must be numerical. SO do hot endcoding again.

# One-hot encode categorical variables

X\_encoded = pd.get\_dummies(X, drop\_first=True)

# Add the target variable to the encoded features

data\_for\_corr = X\_encoded.copy()

data\_for\_corr['y'] = y  # Ensure 'y' is numeric (0 and 1)

RESULT

#ADDITIONAL ENCODING FOR BETTER ANALYSIS

# the correlation matrix calculation requires numerical data, but this dataset likely contains categorical variables that are still represented as strings (e.g., 'no', 'yes', job categories, etc.).

# To compute the correlation matrix, all columns in dataset must be numerical. SO do hot endcoding again.

# One-hot encode categorical variables

X\_encoded = pd.get\_dummies(X, drop\_first=True)

# Add the target variable to the encoded features

data\_for\_corr = X\_encoded.copy()

data\_for\_corr['y'] = y # Ensure 'y' is numeric (0 and 1)

Unique values in y: ['no' 'yes']

Value counts in y:

y

no 4000

yes 521

Name: count, dtype: int64

First few rows of y:

0 no

1 no

2 no

3 no

4 no

Name: y, dtype: object

Significant correlations with the target variable 'y':

duration 0.401118

poutcome\_success 0.283481

month\_oct 0.145964

previous 0.116714

pdays 0.104087

month\_mar 0.102716

job\_retired 0.086675

month\_sep 0.071510

month\_dec 0.069884

education\_tertiary 0.056649

poutcome\_other 0.051908

campaign -0.061147

marital\_married -0.064643

job\_blue-collar -0.068147

loan\_yes -0.070517

month\_may -0.102077

housing\_yes -0.104683

contact\_unknown -0.139399

poutcome\_unknown -0.162038

Name: y, dtype: float64

#Analysis:

#Critical Predictors: duration and poutcome\_success are the most critical predictors for modeling. Focus on clients who had longer call durations and successful outcomes in previous campaigns.

#Seasonality: Subscription likelihood is influenced by certain months (e.g., October, March). Campaigns should be planned strategically during these periods.

#Demographics: Features like job\_retired and education\_tertiary positively impact subscription likelihood, while blue-collar jobs and housing loans have negative impacts.

#Campaign Efficiency: Too many contacts (campaign) or lack of prior success (poutcome\_unknown) can decrease the likelihood of subscription.

#VISUALIZATION

# Calculate total counts and percentages for each class

Target Variable Distribution:

y

0 4000

1 521

Name: count, dtype: int64

Percentage Distribution:

y

0 88.476001

1 11.523999

Name: proportion, dtype: float64

#VISUALIZATION

# Sort features by correlation with y

Top 10 Features Correlated with Subscription (y):

duration 0.401118

poutcome\_success 0.283481

month\_oct 0.145964

previous 0.116714

pdays 0.104087

month\_mar 0.102716

job\_retired 0.086675

month\_sep 0.071510

month\_dec 0.069884

education\_tertiary 0.056649

Name: y, dtype: float64

Percentage Contribution of Top 10 Features:

duration 40.111830

poutcome\_success 28.348088

month\_oct 14.596376

previous 11.671444

pdays 10.408682

month\_mar 10.271568

job\_retired 8.667484

month\_sep 7.151025

month\_dec 6.988431

education\_tertiary 5.664925

Name: y, dtype: float64

#VISUALIZATION

# Distribution of Key Numerical Features

# Visualize the distribution of features most correlated with y

# Select top numerical features based on correlation

key\_features = top\_corr\_features[1:6].index  # Top 5 features excluding 'y'

# Plot the distribution for each key feature

for feature in key\_features:

    plt.figure(figsize=(6, 4))

    # Plot histogram with KDE

    sns.histplot(data\_for\_corr[feature], kde=True, bins=30)

    plt.title(f'Distribution of {feature}')

    plt.xlabel(feature)

    plt.ylabel('Frequency')

    plt.show()

    # Calculate and print values and percentages

    value\_counts = data\_for\_corr[feature].value\_counts().sort\_index()

    percentages = (value\_counts / value\_counts.sum()) \* 100

    print(f"Values and Percentages for {feature}:")

    print(value\_counts)

    print("Percentages (%):")

    print(percentages)

    print("\n---\n")

Values and Percentages for duration:

duration

4 1

5 9

6 2

7 6

8 9

..

2029 1

2087 1

2456 1

2769 1

3025 1

Name: count, Length: 875, dtype: int64

Percentages (%):

duration

4 0.022119

5 0.199071

6 0.044238

7 0.132714

8 0.199071

...

2029 0.022119

2087 0.022119

2456 0.022119

...

Name: count, Length: 875, dtype: float64

Values and Percentages for poutcome\_success:

poutcome\_success

False 4392

True 129

Name: count, dtype: int64

Percentages (%):

poutcome\_success

False 97.146649

True 2.853351

Name: count, dtype: float64

Values and Percentages for month\_oct:

month\_oct

False 4441

True 80

Name: count, dtype: int64

Percentages (%):

month\_oct

False 98.23048

True 1.76952

Name: count, dtype: float64

Values and Percentages for previous:

previous

0 3705

1 286

2 193

3 113

4 78

5 47

6 25

7 22

8 18

9 10

10 4

11 3

12 5

13 1

14 2

15 1

17 1

18 1

19 1

20 1

22 1

23 1

24 1

...

Name: count, dtype: float64

Values and Percentages for pdays:

pdays

-1 3705

1 2

2 7

3 1

5 1

...

687 1

761 1

804 1

808 1

871 1

Name: count, Length: 292, dtype: int64

Percentages (%):

pdays

-1 81.950896

1 0.044238

2 0.154833

3 0.022119

5 0.022119

...

687 0.022119

761 0.022119

804 0.022119

...

Name: count, Length: 292, dtype: float64

Feature Importances from Random Forest:

Feature Importance

3 duration 0.262608

1 balance 0.097485

0 age 0.094458

2 day 0.084732

40 poutcome\_success 0.045205

4 campaign 0.043271

5 pdays 0.042791

6 previous 0.021444

24 housing\_yes 0.018779

37 month\_oct 0.017563

18 marital\_married 0.016532

20 education\_secondary 0.014300

21 education\_tertiary 0.014286

27 contact\_unknown 0.013736

28 month\_aug 0.012163

10 job\_management 0.011966

33 month\_jun 0.011806

35 month\_may 0.011411

15 job\_technician 0.011318

34 month\_mar 0.011141

19 marital\_single 0.010704

25 loan\_yes 0.010449

7 job\_blue-collar 0.010160

...

16 job\_unemployed 0.003374

23 default\_yes 0.003014

29 month\_dec 0.002703

17 job\_unknown 0.002293

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469): ConvergenceWarning: lbfgs failed to converge (status=1):

...

<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(

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?d73330ef-8912-46f8-b2a2-4b0ce98e6fe8) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?d73330ef-8912-46f8-b2a2-4b0ce98e6fe8)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Best Parameters for Logistic Regression: {'C': 10, 'solver': 'liblinear'}

Best Accuracy for Logistic Regression: 0.902100288087542

Starting hyperparameter tuning for Logistic Regression...

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469): ConvergenceWarning: lbfgs failed to converge (status=1):

...

<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(

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?2fea12e6-1fc4-4749-b2fa-27ec4572485f) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?2fea12e6-1fc4-4749-b2fa-27ec4572485f)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Hyperparameter tuning completed.

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

Marketing Campaing Subscription Binary kick-off!

Dataset loaded successfully!

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4521 entries, 0 to 4520

Data columns (total 17 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 4521 non-null int64

1 job 4521 non-null object

2 marital 4521 non-null object

3 education 4521 non-null object

4 default 4521 non-null object

5 balance 4521 non-null int64

6 housing 4521 non-null object

7 loan 4521 non-null object

8 contact 4521 non-null object

9 day 4521 non-null int64

10 month 4521 non-null object

11 duration 4521 non-null int64

12 campaign 4521 non-null int64

13 pdays 4521 non-null int64

14 previous 4521 non-null int64

15 poutcome 4521 non-null object

16 y 4521 non-null object

dtypes: int64(7), object(10)

memory usage: 600.6+ KB

...

1 cellular 11 may 220 1 339 4 failure no

2 cellular 16 apr 185 1 330 1 failure no

3 unknown 3 jun 199 4 -1 0 unknown no

4 unknown 5 may 226 1 -1 0 unknown no

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?11c591a9-ad18-4472-97da-cfe3a0929b1d) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?11c591a9-ad18-4472-97da-cfe3a0929b1d)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

pandas.core.frame.DataFrame

Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',

'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',

'previous', 'poutcome', 'y'],

dtype='object')

|  | | **age** | | **balance** | | | | **day** | | | **duration** | | | | **campaign** | | | **pdays** | | | **previous** | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | | 4521.000000 | | 4521.000000 | | | | 4521.000000 | | | 4521.000000 | | | | 4521.000000 | | | 4521.000000 | | | 4521.000000 | | |
| mean | | 41.170095 | | 1422.657819 | | | | 15.915284 | | | 263.961292 | | | | 2.793630 | | | 39.766645 | | | 0.542579 | | |
| std | | 10.576211 | | 3009.638142 | | | | 8.247667 | | | 259.856633 | | | | 3.109807 | | | 100.121124 | | | 1.693562 | | |
| min | | 19.000000 | | -3313.000000 | | | | 1.000000 | | | 4.000000 | | | | 1.000000 | | | -1.000000 | | | 0.000000 | | |
| 25% | | 33.000000 | | 69.000000 | | | | 9.000000 | | | 104.000000 | | | | 1.000000 | | | -1.000000 | | | 0.000000 | | |
| 50% | | 39.000000 | | 444.000000 | | | | 16.000000 | | | 185.000000 | | | | 2.000000 | | | -1.000000 | | | 0.000000 | | |
| 75% | | 49.000000 | | 1480.000000 | | | | 21.000000 | | | 329.000000 | | | | 3.000000 | | | -1.000000 | | | 0.000000 | | |
| max | | 87.000000 | | 71188.000000 | | | | 31.000000 | | | 3025.000000 | | | | 50.000000 | | | 871.000000 | | | 25.000000 | | |
|  | **age** | **job** | **marital** | | **education** | **default** | **balance** | | **housing** | **loan** | | **contact** | **day** | **month** | | **duration** | **campaign** | | **pdays** | **previous** | | **poutcome** | **y** |
| 0 | 30 | unemployed | married | | primary | no | 1787 | | no | no | | cellular | 19 | oct | | 79 | 1 | | -1 | 0 | | unknown | no |
| 1 | 33 | services | married | | secondary | no | 4789 | | yes | yes | | cellular | 11 | may | | 220 | 1 | | 339 | 4 | | failure | no |
| 2 | 35 | management | single | | tertiary | no | 1350 | | yes | no | | cellular | 16 | apr | | 185 | 1 | | 330 | 1 | | failure | no |
| 3 | 30 | management | married | | tertiary | no | 1476 | | yes | yes | | unknown | 3 | jun | | 199 | 4 | | -1 | 0 | | unknown | no |
| 4 | 59 | blue-collar | married | | secondary | no | 0 | | yes | no | | unknown | 5 | may | | 226 | 1 | | -1 | 0 | | unknown | no |

age job marital education default balance housing loan contact day month duration campaign pdays previous poutcome y

19 student single primary no 103 no no cellular 10 jul 104 2 -1 0 unknown yes 1

45 services single secondary no 1757 yes no cellular 20 apr 1010 3 326 1 other no 1

technician married secondary no 88 no no cellular 29 aug 150 2 -1 0 unknown no 1

49 no no cellular 29 jul 65 2 -1 0 unknown no 1

-149 yes no cellular 14 jul 287 2 -1 0 unknown no 1

..

35 blue-collar married secondary no 305 yes no cellular 18 may 7 7 367 25 failure no 1

407 yes no cellular 20 apr 12 6 -1 0 unknown no 1

444 yes no cellular 15 apr 244 3 -1 0 unknown no 1

603 yes no cellular 17 apr 474 2 -1 0 unknown no 1

87 retired married primary no 230 no no cellular 30 oct 144 1 -1 0 unknown yes 1

Name: count, Length: 4521, dtype: int64

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4521 entries, 0 to 4520

Data columns (total 17 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 4521 non-null int64

1 job 4521 non-null object

2 marital 4521 non-null object

3 education 4521 non-null object

4 default 4521 non-null object

5 balance 4521 non-null int64

6 housing 4521 non-null object

7 loan 4521 non-null object

8 contact 4521 non-null object

9 day 4521 non-null int64

10 month 4521 non-null object

11 duration 4521 non-null int64

12 campaign 4521 non-null int64

13 pdays 4521 non-null int64

14 previous 4521 non-null int64

15 poutcome 4521 non-null object

16 y 4521 non-null object

dtypes: int64(7), object(10)

memory usage: 600.6+ KB

0 False

1 False

2 False

3 False

4 False

...

4516 False

4517 False

4518 False

4519 False

4520 False

Length: 4521, dtype: bool

0

Duplicate Rows Count: 0

Missing Values Count:

age 0

job 0

marital 0

education 0

default 0

balance 0

housing 0

loan 0

contact 0

day 0

month 0

duration 0

campaign 0

pdays 0

previous 0

poutcome 0

y 0

dtype: int64

|  | **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 30 | unemployed | married | primary | no | 1787 | no | no | cellular | 19 | oct | 79 | 1 | -1 | 0 | unknown | no |
| 1 | 33 | services | married | secondary | no | 4789 | yes | yes | cellular | 11 | may | 220 | 1 | 339 | 4 | failure | no |
| 2 | 35 | management | single | tertiary | no | 1350 | yes | no | cellular | 16 | apr | 185 | 1 | 330 | 1 | failure | no |
| 3 | 30 | management | married | tertiary | no | 1476 | yes | yes | unknown | 3 | jun | 199 | 4 | -1 | 0 | unknown | no |
| 4 | 59 | blue-collar | married | secondary | no | 0 | yes | no | unknown | 5 | may | 226 | 1 | -1 | 0 | unknown | no |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 4516 | 33 | services | married | secondary | no | -333 | yes | no | cellular | 30 | jul | 329 | 5 | -1 | 0 | unknown | no |
| 4517 | 57 | self-employed | married | tertiary | yes | -3313 | yes | yes | unknown | 9 | may | 153 | 1 | -1 | 0 | unknown | no |
| 4518 | 57 | technician | married | secondary | no | 295 | no | no | cellular | 19 | aug | 151 | 11 | -1 | 0 | unknown | no |
| 4519 | 28 | blue-collar | married | secondary | no | 1137 | no | no | cellular | 6 | feb | 129 | 4 | 211 | 3 | other | no |
| 4520 | 44 | entrepreneur | single | tertiary | no | 1136 | yes | yes | cellular | 3 | apr | 345 | 2 | 249 | 7 | other | no |

4521 rows × 17 columns

Shape of X (features): (4521, 16)

Shape of y (target): (4521,)

age job marital education default balance housing loan \

0 30 unemployed married primary no 1787 no no

1 33 services married secondary no 4789 yes yes

2 35 management single tertiary no 1350 yes no

3 30 management married tertiary no 1476 yes yes

4 59 blue-collar married secondary no 0 yes no

contact day month duration campaign pdays previous poutcome

0 cellular 19 oct 79 1 -1 0 unknown

1 cellular 11 may 220 1 339 4 failure

2 cellular 16 apr 185 1 330 1 failure

3 unknown 3 jun 199 4 -1 0 unknown

4 unknown 5 may 226 1 -1 0 unknown

0 no

1 no

2 no

3 no

4 no

Name: y, dtype: object

Columns in X (features):

Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous',

'job\_blue-collar', 'job\_entrepreneur', 'job\_housemaid',

'job\_management', 'job\_retired', 'job\_self-employed', 'job\_services',

'job\_student', 'job\_technician', 'job\_unemployed', 'job\_unknown',

'marital\_married', 'marital\_single', 'education\_secondary',

'education\_tertiary', 'education\_unknown', 'default\_yes', 'housing\_yes',

'loan\_yes', 'contact\_telephone', 'contact\_unknown', 'month\_aug',

'month\_dec', 'month\_feb', 'month\_jan', 'month\_jul', 'month\_jun',

'month\_mar', 'month\_may', 'month\_nov', 'month\_oct', 'month\_sep',

'poutcome\_other', 'poutcome\_success', 'poutcome\_unknown'],

dtype='object')

Target variable (y):

y

age int64

balance int64

day int64

duration int64

campaign int64

pdays int64

previous int64

job\_blue-collar bool

job\_entrepreneur bool

job\_housemaid bool

job\_management bool

job\_retired bool

job\_self-employed bool

job\_services bool

job\_student bool

job\_technician bool

job\_unemployed bool

job\_unknown bool

marital\_married bool

marital\_single bool

education\_secondary bool

education\_tertiary bool

education\_unknown bool

default\_yes bool

housing\_yes bool

...

poutcome\_success bool

poutcome\_unknown bool

y object

dtype: object

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?86197fc5-5b67-4e61-9445-f1fe7006078e) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?86197fc5-5b67-4e61-9445-f1fe7006078e)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Total number of columns: 43

Number of numerical columns: 7

Numerical columns: ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']

Number of categorical columns: 0

Categorical columns: []

Unique values in y: ['no' 'yes']

Value counts in y:

y

no 4000

yes 521

Name: count, dtype: int64

First few rows of y:

0 no

1 no

2 no

3 no

4 no

Name: y, dtype: object

age int64

balance int64

day int64

duration int64

campaign int64

pdays int64

previous int64

job\_blue-collar bool

job\_entrepreneur bool

job\_housemaid bool

job\_management bool

job\_retired bool

job\_self-employed bool

job\_services bool

job\_student bool

job\_technician bool

job\_unemployed bool

job\_unknown bool

marital\_married bool

marital\_single bool

education\_secondary bool

education\_tertiary bool

education\_unknown bool

default\_yes bool

housing\_yes bool

...

poutcome\_unknown bool

y object

dtype: object

['no' 'yes']

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?5d6fd791-154f-4c06-ba41-08d8aa10b9e2) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?5d6fd791-154f-4c06-ba41-08d8aa10b9e2)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

[C:\Users\Carmen\AppData\Local\Temp\ipykernel\_21992\366654606.py:4](file:///C:\Users\Carmen\AppData\Local\Temp\ipykernel_21992\366654606.py:4): FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer\_objects(copy=False)`. To opt-in to the future behavior, set `pd.set\_option('future.no\_silent\_downcasting', True)`

data\_for\_corr['y'] = data\_for\_corr['y'].replace({'no': 0, 'yes': 1})

Significant correlations with the target variable 'y':

duration 0.401118

poutcome\_success 0.283481

month\_oct 0.145964

previous 0.116714

pdays 0.104087

month\_mar 0.102716

job\_retired 0.086675

month\_sep 0.071510

month\_dec 0.069884

education\_tertiary 0.056649

poutcome\_other 0.051908

campaign -0.061147

marital\_married -0.064643

job\_blue-collar -0.068147

loan\_yes -0.070517

month\_may -0.102077

housing\_yes -0.104683

contact\_unknown -0.139399

poutcome\_unknown -0.162038

Name: y, dtype: float64

Target Variable Distribution:

y

0 4000

1 521

Name: count, dtype: int64

Percentage Distribution:

y

0 88.476001

1 11.523999

Name: proportion, dtype: float64

Top 10 Features Correlated with Subscription (y):

duration 0.401118

poutcome\_success 0.283481

month\_oct 0.145964

previous 0.116714

pdays 0.104087

month\_mar 0.102716

job\_retired 0.086675

month\_sep 0.071510

month\_dec 0.069884

education\_tertiary 0.056649

Name: y, dtype: float64

Percentage Contribution of Top 10 Features:

duration 40.111830

poutcome\_success 28.348088

month\_oct 14.596376

previous 11.671444

pdays 10.408682

month\_mar 10.271568

job\_retired 8.667484

month\_sep 7.151025

month\_dec 6.988431

education\_tertiary 5.664925

Name: y, dtype: float64

Values and Percentages for duration:

duration

4 1

5 9

6 2

7 6

8 9

..

2029 1

2087 1

2456 1

2769 1

3025 1

Name: count, Length: 875, dtype: int64

Percentages (%):

duration

4 0.022119

5 0.199071

6 0.044238

7 0.132714

8 0.199071

...

2029 0.022119

2087 0.022119

2456 0.022119

...

Name: count, Length: 875, dtype: float64

---

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?9f0d687a-6a41-40ff-a630-d88539fb34a4) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?9f0d687a-6a41-40ff-a630-d88539fb34a4)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Values and Percentages for poutcome\_success:

poutcome\_success

False 4392

True 129

Name: count, dtype: int64

Percentages (%):

poutcome\_success

False 97.146649

True 2.853351

Name: count, dtype: float64

---

Values and Percentages for month\_oct:

month\_oct

False 4441

True 80

Name: count, dtype: int64

Percentages (%):

month\_oct

False 98.23048

True 1.76952

Name: count, dtype: float64

---

Values and Percentages for previous:

previous

0 3705

1 286

2 193

3 113

4 78

5 47

6 25

7 22

8 18

9 10

10 4

11 3

12 5

13 1

14 2

15 1

17 1

18 1

19 1

20 1

22 1

23 1

24 1

...

Name: count, dtype: float64

---

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?1e4871f6-a2bd-4662-b0aa-42b5da8b3f4a) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?1e4871f6-a2bd-4662-b0aa-42b5da8b3f4a)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Values and Percentages for pdays:

pdays

-1 3705

1 2

2 7

3 1

5 1

...

687 1

761 1

804 1

808 1

871 1

Name: count, Length: 292, dtype: int64

Percentages (%):

pdays

-1 81.950896

1 0.044238

2 0.154833

3 0.022119

5 0.022119

...

687 0.022119

761 0.022119

804 0.022119

...

Name: count, Length: 292, dtype: float64

---

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?f39af0d2-c49c-4f0b-b8f9-9b1f1ba5634b) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?f39af0d2-c49c-4f0b-b8f9-9b1f1ba5634b)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Feature Importances from Random Forest:

Feature Importance

3 duration 0.262608

1 balance 0.097485

0 age 0.094458

2 day 0.084732

40 poutcome\_success 0.045205

4 campaign 0.043271

5 pdays 0.042791

6 previous 0.021444

24 housing\_yes 0.018779

37 month\_oct 0.017563

18 marital\_married 0.016532

20 education\_secondary 0.014300

21 education\_tertiary 0.014286

27 contact\_unknown 0.013736

28 month\_aug 0.012163

10 job\_management 0.011966

33 month\_jun 0.011806

35 month\_may 0.011411

15 job\_technician 0.011318

34 month\_mar 0.011141

19 marital\_single 0.010704

25 loan\_yes 0.010449

7 job\_blue-collar 0.010160

...

16 job\_unemployed 0.003374

23 default\_yes 0.003014

29 month\_dec 0.002703

17 job\_unknown 0.002293

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?acc05825-ed8a-4bf7-b327-e6772cdb8600) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?acc05825-ed8a-4bf7-b327-e6772cdb8600)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Correlation of Features with Target Variable (y):

y 1.000000

duration 0.401118

poutcome\_success 0.283481

month\_oct 0.145964

previous 0.116714

pdays 0.104087

month\_mar 0.102716

job\_retired 0.086675

month\_sep 0.071510

month\_dec 0.069884

education\_tertiary 0.056649

poutcome\_other 0.051908

job\_student 0.047809

marital\_single 0.045815

age 0.045092

month\_feb 0.039805

job\_management 0.032634

contact\_telephone 0.025878

job\_unknown 0.019886

balance 0.017905

month\_aug 0.012084

job\_housemaid 0.004872

default\_yes 0.001303

job\_self-employed -0.003827

...

housing\_yes -0.104683

contact\_unknown -0.139399

poutcome\_unknown -0.162038

Name: y, dtype: float64

#DROPPIN GCOLUMNS BEFORE MODELING

#To build a more efficient and effective model, it's essential to drop features that are either:

    #Not strongly correlated with the target variable (y).

    #Low in feature importance according to the Random Forest analysis.

    #Highly correlated with other features, which can lead to multicollinearity

#Threshold for Correlation with y:

        #Features with very low absolute correlation (e.g., |correlation| < 0.05) are unlikely to contribute significantly to predicting y.

        #default\_yes (correlation: 0.001303)

        #job\_self-employed (correlation: -0.003827)

        #housing\_yes (correlation: -0.104683)

        #contact\_unknown (correlation: -0.139399)

        #poutcome\_unknown (correlation: -0.162038)

#Threshold for Feature Importance:

    #Features with extremely low importance (e.g., importance < 0.01 in Random Forest) can be removed as they have minimal predictive power.

        #default\_yes

        #job\_housemaid

        #month\_aug

#Domain Knowledge or Redundancy:

    #Drop features that are logically redundant or have unclear relationships with the target (e.g., multiple highly correlated time-based features).

Shape of X before dropping: (4521, 42)

Shape of X after dropping: (4521, 35)

Training set shape: (3616, 35), Testing set shape: (905, 35)

raining KNN...

KNN Evaluation:

Accuracy: 0.8773480662983425

Confusion Matrix:

[[770 37]

[ 74 24]]

Classification Report:

precision recall f1-score support

0 0.91 0.95 0.93 807

1 0.39 0.24 0.30 98

accuracy 0.88 905

macro avg 0.65 0.60 0.62 905

weighted avg 0.86 0.88 0.86 905

Training Logistic Regression...

[c:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear\_model\\_logistic.py:469](file:///C:\ProgramData\anaconda3\Lib\site-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(

Logistic Regression Evaluation:

Accuracy: 0.8994475138121547

Confusion Matrix:

[[786 21]

[ 70 28]]

Classification Report:

precision recall f1-score support

0 0.92 0.97 0.95 807

1 0.57 0.29 0.38 98

accuracy 0.90 905

macro avg 0.74 0.63 0.66 905

weighted avg 0.88 0.90 0.88 905

Training Decision Tree...

Decision Tree Evaluation:

Accuracy: 0.8629834254143647

Confusion Matrix:

[[742 65]

[ 59 39]]

Classification Report:

...

accuracy 0.89 905

macro avg 0.45 0.50 0.47 905

weighted avg 0.80 0.89 0.84 905

Model Performance:

Model Accuracy

0 KNN 0.877348

1 Logistic Regression 0.899448

2 Decision Tree 0.862983

3 SVM 0.890608

# Best-Performing Model

#From the accuracy results:

    #Logistic Regression achieved the highest accuracy (89.94%) among the four models.

#Insights from Results

    #KNN (87.73%) and SVM (89.06%) also performed well, showing that the data might have some non-linear relationships that these models can capture.

    #Decision Tree (87.07%) performed slightly worse, possibly due to overfitting or suboptimal splitting criteria.

#Hyperparameter Tuning:

#Use GridSearchCV to optimize parameters for Logistic Regression, KNN, SVM, and Decision Tree.

#Example for SVM

#from sklearn.model\_selection import GridSearchCV

# Parameter grid for SVM

#param\_grid = {

    #'C': [0.1, 1, 10],

    #'kernel': ['linear', 'rbf'],

    #'gamma': [1, 0.1, 0.01]

#}

#grid = GridSearchCV(SVC(), param\_grid, cv=5, scoring='accuracy')

#grid.fit(X\_train, y\_train)

#print("Best Parameters for SVM:", grid.best\_params\_)

#print("Best Accuracy for SVM:", grid.best\_score\_)

#Note: I used this one and it did not complete in an hour, changing it to LR

##Analysis

#Logistic Regression

    #Best Accuracy (89.94%) among all models.

    #Logistic Regression assumes a linear relationship between the features and the log-odds of the target variable.

    # This suggests that the relationships in the data are well-modeled with a linear approach.

    #Advantages:

       #Simple and interpretable.

        #Faster to train and tune than complex models like SVM or Decision Trees.

        #Works well when relationships in the data are mostly linear.

#Note: This is my recommended method KNN it was faster to execute compare to SVM or LG under 15 seconds

#K-Nearest Neighbors (KNN)

    #Accuracy: 88.16%.

    #KNN is simple and works well for small datasets with clear clusters.

    #The chosen parameters (n\_neighbors=9, metric='euclidean') indicate that a moderate neighborhood size balances between bias and variance.

    #Advantages:

        #Non-parametric (no assumptions about data distribution).

        #Can model complex relationships.

    #Disadvantage:

        #Computationally expensive for large datasets (requires calculating distances for every query point).

        #Lower accuracy compared to Logistic Regression and Decision Trees.

Best Parameters for KNN: {'metric': 'euclidean', 'n\_neighbors': 9, 'weights': 'uniform'}

Best Accuracy for KNN: 0.8816372847940211

#Note: This DT was really fast as well to be xecuted under 16 seconds

#Decision Tree

    #Accuracy: 89.38% (slightly below Logistic Regression).

    #Decision Trees can model non-linear relationships, which makes them useful when data has complex interactions.

    #The chosen parameters (entropy, max\_depth=5, etc.) suggest a pruned tree to avoid overfitting.

    #Advantages:

       #Easy to interpret.

        #Handles non-linearity and feature interactions well.

    #Disadvantage:

        #Slightly less accurate than Logistic Regression, potentially due to limited tree depth (5).

#Hyperparameter Tuning for Decision Tree

#Decision Trees are simpler models and typically tune faster than SVM.

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV

# Parameter grid for Decision Tree

param\_grid = {

'criterion': ['gini', 'entropy'], # Split criteria

'max\_depth': [None, 5, 10, 20], # Maximum depth of the tree

'min\_samples\_split': [2, 5, 10], # Minimum samples required to split

'min\_samples\_leaf': [1, 2, 4], # Minimum samples per leaf

}

# GridSearch for Decision Tree

grid = GridSearchCV(DecisionTreeClassifier(), param\_grid, cv=5, scoring='accuracy')

grid.fit(X\_train, y\_train)

# Print the best parameters and score

print("Best Parameters for Decision Tree:", grid.best\_params\_)

print("Best Accuracy for Decision Tree:", grid.best\_score\_)

#Note: This DT was really fast as well to be xecuted under 16 seconds

#Decision Tree

#Accuracy: 89.38% (slightly below Logistic Regression).

#Decision Trees can model non-linear relationships, which makes them useful when data has complex interactions.

#The chosen parameters (entropy, max\_depth=5, etc.) suggest a pruned tree to avoid overfitting.

#Advantages:

#Easy to interpret.

#Handles non-linearity and feature interactions well.

#Disadvantage:

#Slightly less accurate than Logistic Regression, potentially due to limited tree depth (5).

#RESULTS

# Based on the provided results and hyperparameter tuning outcomes, the analysis of the models is:

#Model                             Best Parameters                                                     Accuracy (%)

    #Decision Tree  {'criterion': 'entropy', 'max\_depth': 5, 'min\_samples\_leaf': 4, 'min\_samples\_split': 2} 89.38

    #K-Nearest Neighbors (KNN)  {'metric': 'euclidean', 'n\_neighbors': 9, 'weights': 'uniform'} 88.16

    #Logistic Regression    {'C': 1, 'solver': 'lbfgs'} 89.94

#Model Recommendation

        #Logistic Regression is the best choice for this dataset based on:

#The highest accuracy (89.94%).

#Simplicity and interpretability.

#Faster training and evaluation compared to other models.

#If interpretability is less important and capturing non-linear relationships is a priority, Decision Tree could be an alternative, with only a marginal accuracy trade-off (89.38%).

#Recommendations

    #Based on the findings:

#Feature Importance:

    #duration and poutcome\_success are critical predictors.

    #Focus marketing efforts on clients with a history of successful interactions and longer call durations.

#Leverage Simplicity:

    #Logistic Regression provides an interpretable model to communicate results to stakeholders.

#Efficiency in Deployment:

    #Logistic Regression models are lightweight and can be deployed in real-time systems for predictions.

#Next Steps

    #Finalize Logistic Regression as the primary model.

    #Optionally, explore ensemble methods (e.g., Random Forest) for further improvement.