CAPSTONE PROJECT

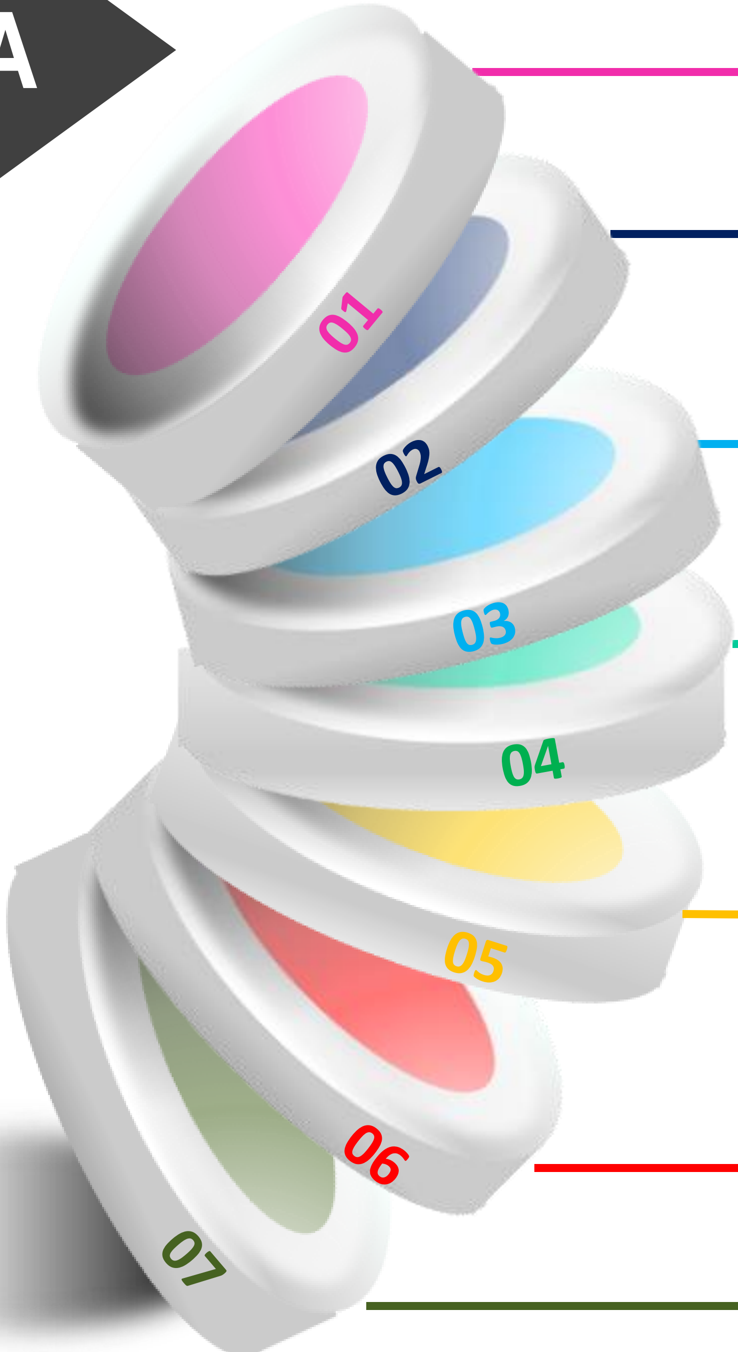
LEARNING BY DOING

BANK MARKETING

MACHINE LEARNING (CLASSIFICATION)

Presented by Group 05:
S .V THEJASWINI
ANKITHA SINGH
M. PRANAV
G. MAHAVEER KUMAR YADAV

AGENDA



INTRODUCTION

DATA PREPROCESSING

EXPLORATORY DATA ANALYSIS

DETERMINING MODEL ACCURACY

COMPARISON OF RESULTS WITH OTHER MODELS

CONCLUSION & INSIGHTS

APPENDIX

DATA DESCRIPTION

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

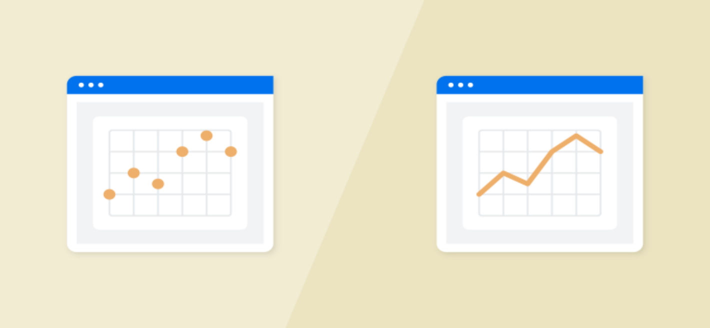
OBJECTIVE

To analyze and predict the likelihood of clients subscribing to a bank term deposit during direct marketing campaigns, using phone call interactions as the primary mode of communication. The goal is to develop a predictive model that accurately classifies clients into 'yes' (subscribed) or 'no' (not subscribed) categories, taking into consideration the multiple contacts made with the same client during the campaign

THE PATH

Implementing various machine learning models to find the best model , to predict the accuracy of the dataset.

Data And Data Quality Check



About The Data

Number of instances : 4521
Number of Attributes : 16 + output attribute (**‘y’** - signifies whether the client subscribed, with 'yes' indicating subscription and 'no' indicating non-subscription during the direct marketing campaigns)

Discrete columns

Job - type of job
Marital - marital status
Education
Default - has credit in default?
Housing - has housing loan?
Loan - has personal loan?
Contact - contact communication type
Month - last contact month of year
Poutcome - outcome of the previous marketing campaign
Y - has the client subscribed a term deposit?

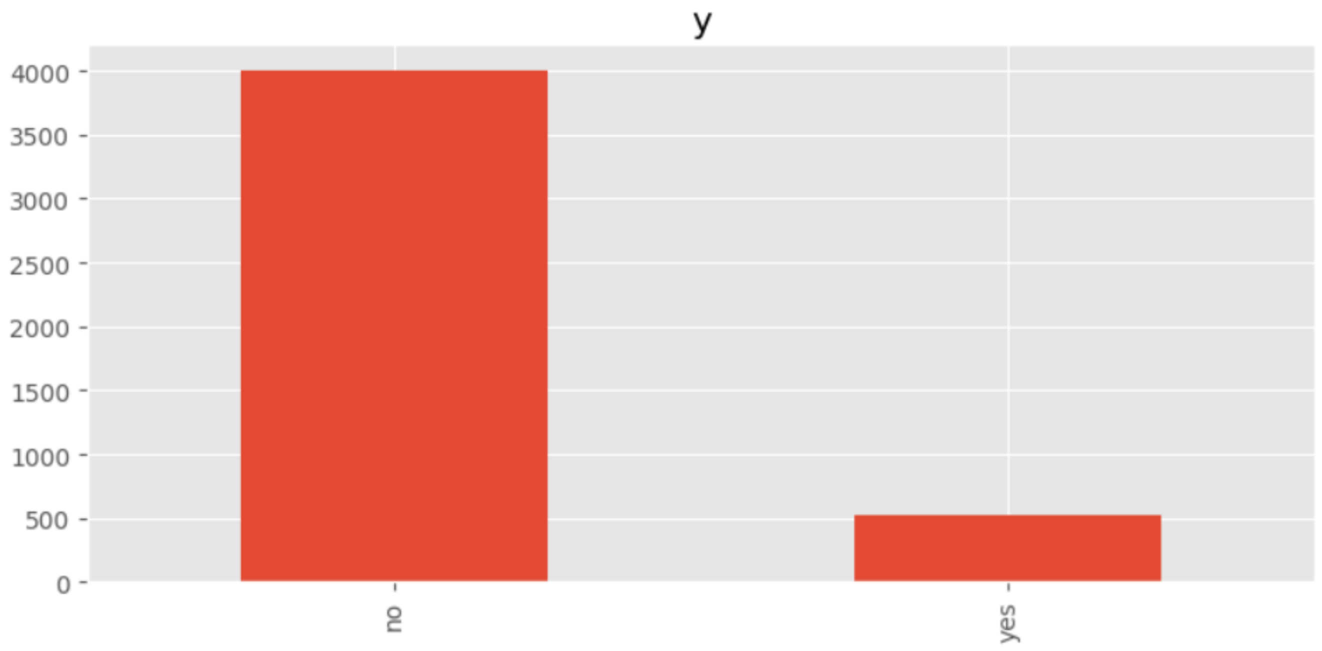
Continuous columns

Age
Balance - average yearly balance, in euros
Day - last contact day of the month
Campaign - number of contacts performed during this campaign and for this client
Pdays - number of days that passed by after the client was last contacted from a previous campaign
Previous - number of contacts performed before this campaign and for this client
Duration - last contact duration, in seconds


```
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
```

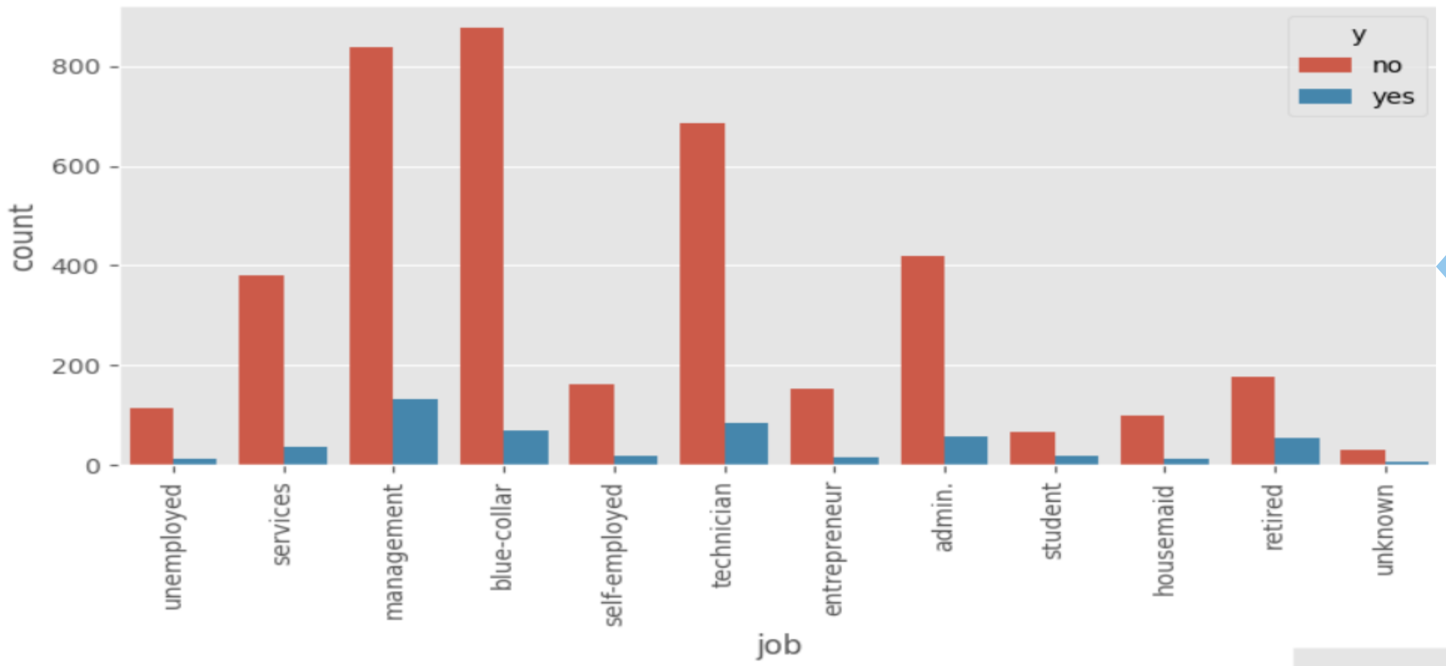
- From the original data there are no missing values or null values .
- No columns were dropped.

EXPLORATORY DATA ANALYSIS(EDA)



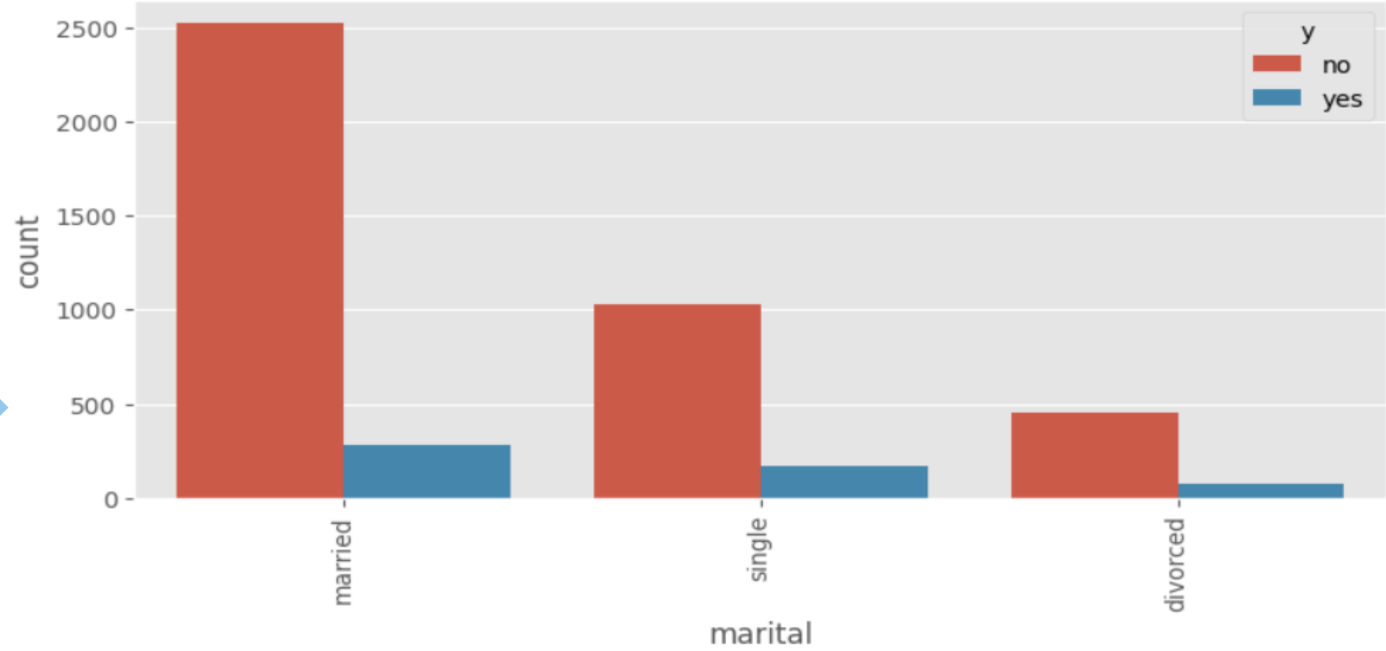
The bar graph depicts subscription(“y”) responses, contrasting the frequency of "yes" and "no" answers of different clients.

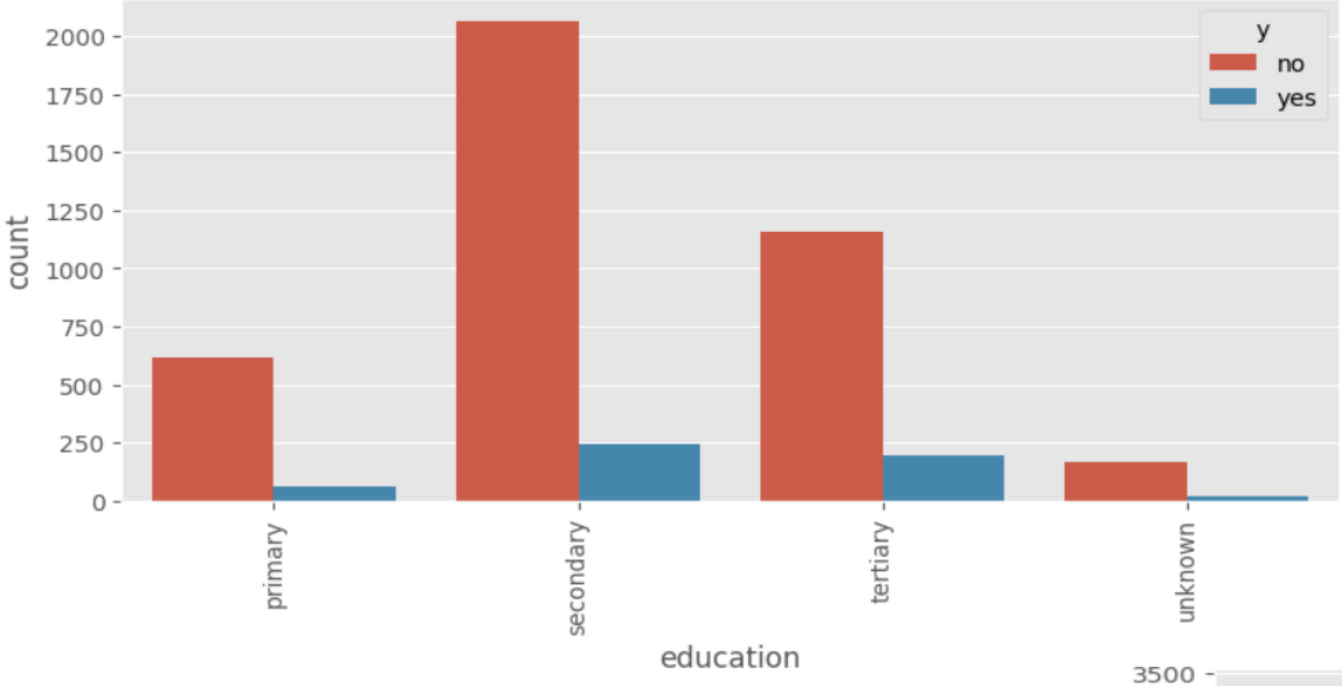
The x-axis denotes subscription status ('Yes' or 'No'), while the y-axis represents the corresponding count.



The bar graph illustrates the counts of "yes" and "no" responses for subscriptions across different job categories.

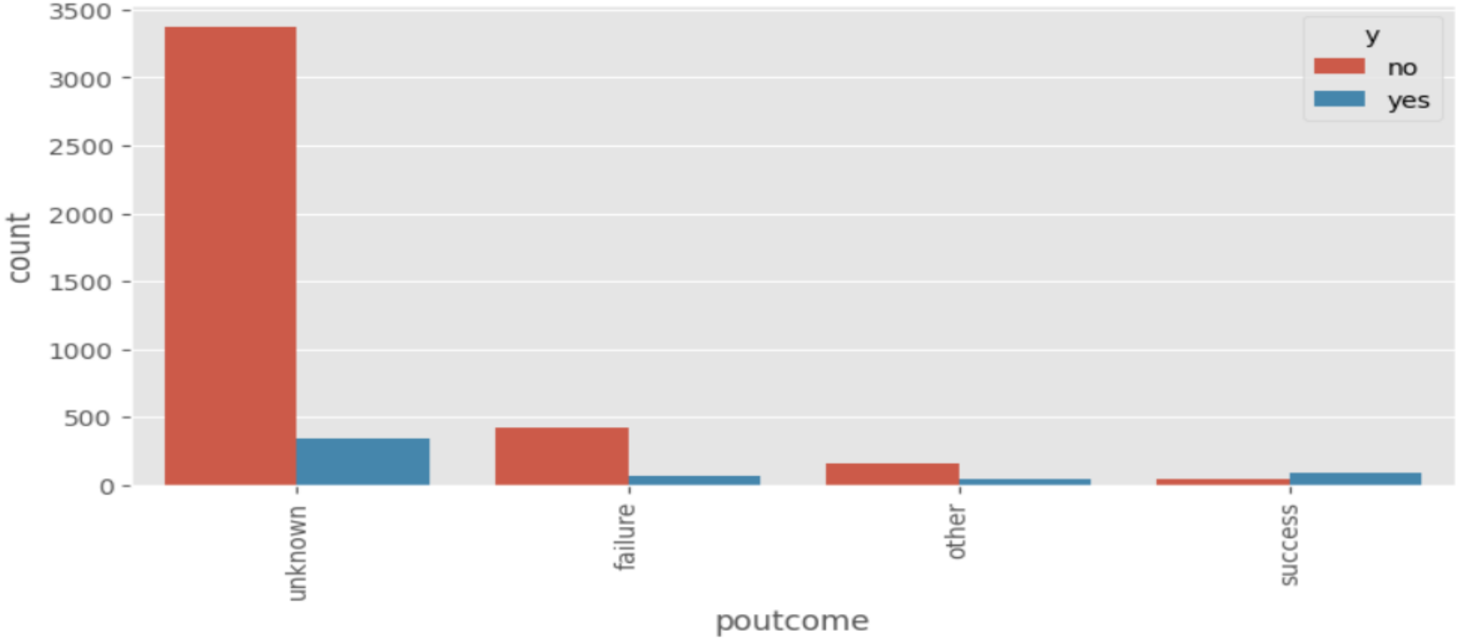
The bar graph displays the counts of "yes" and "no" responses for subscriptions categorized by marital status

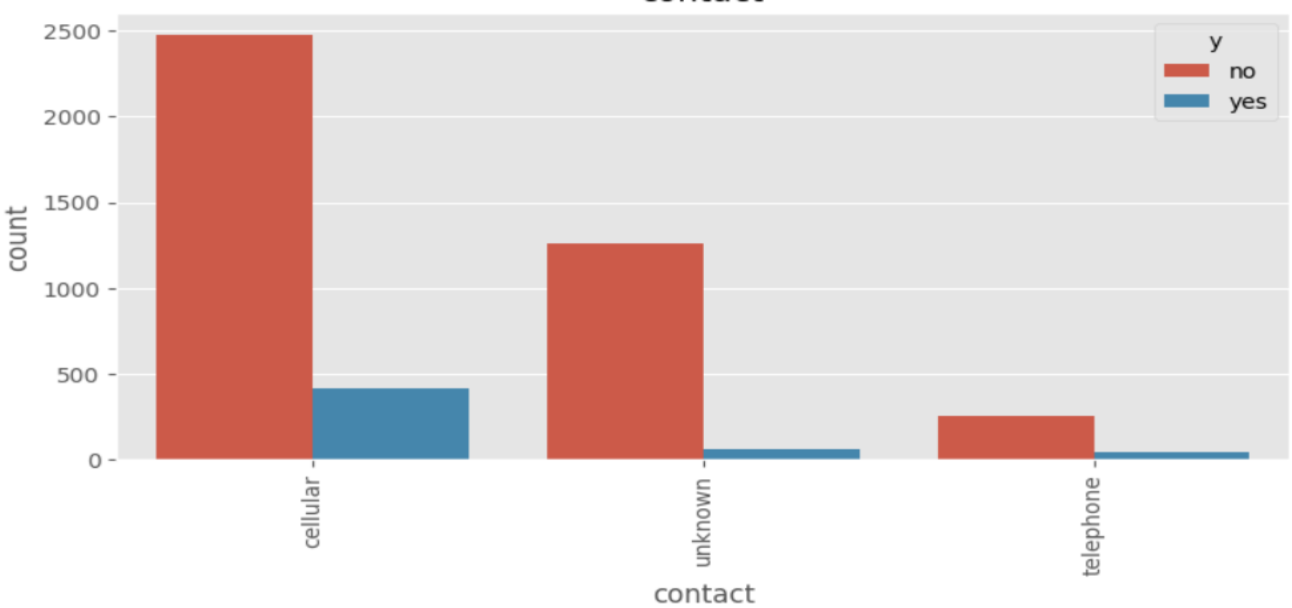




The bar graph illustrates the counts of "yes" and "no" responses for subscriptions categorized by education level.

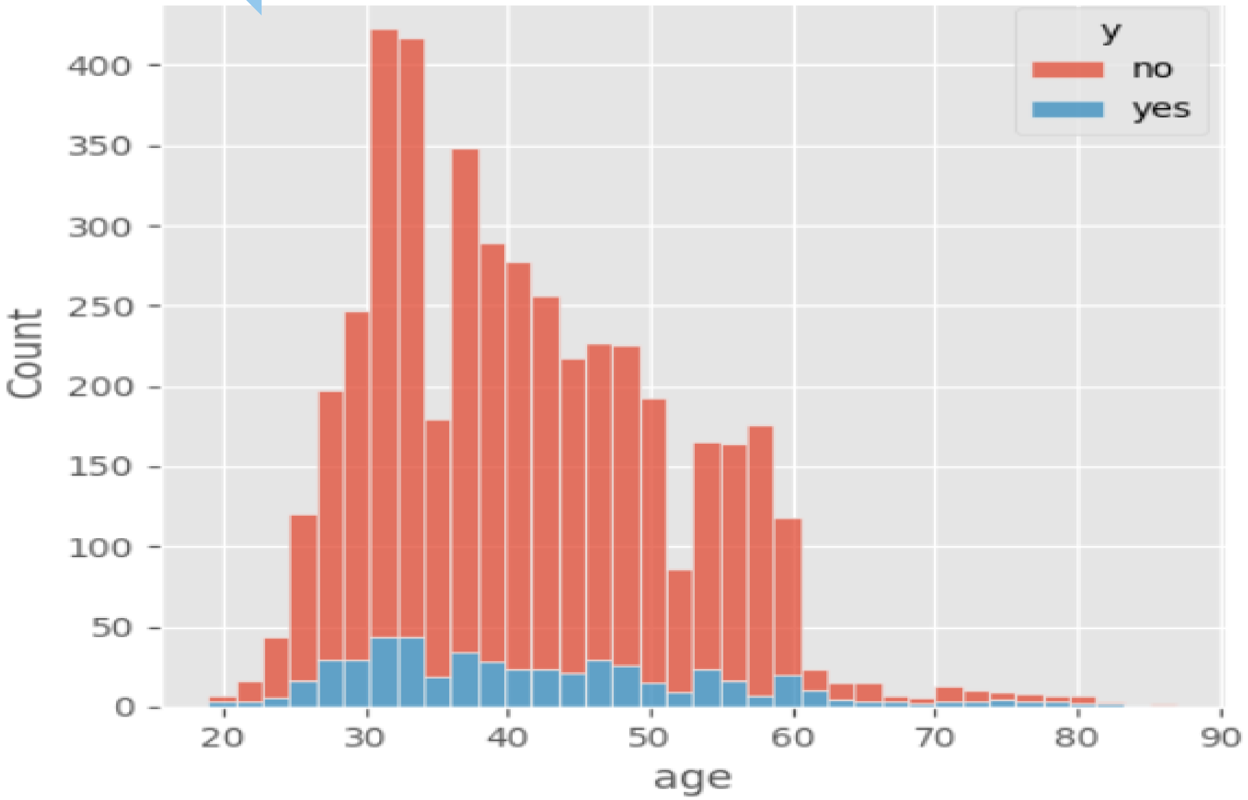
The bar graph presents the counts of "yes" and "no" responses for subscriptions based on different outcomes from a previous marketing campaign, represented by the variable "poutcome."



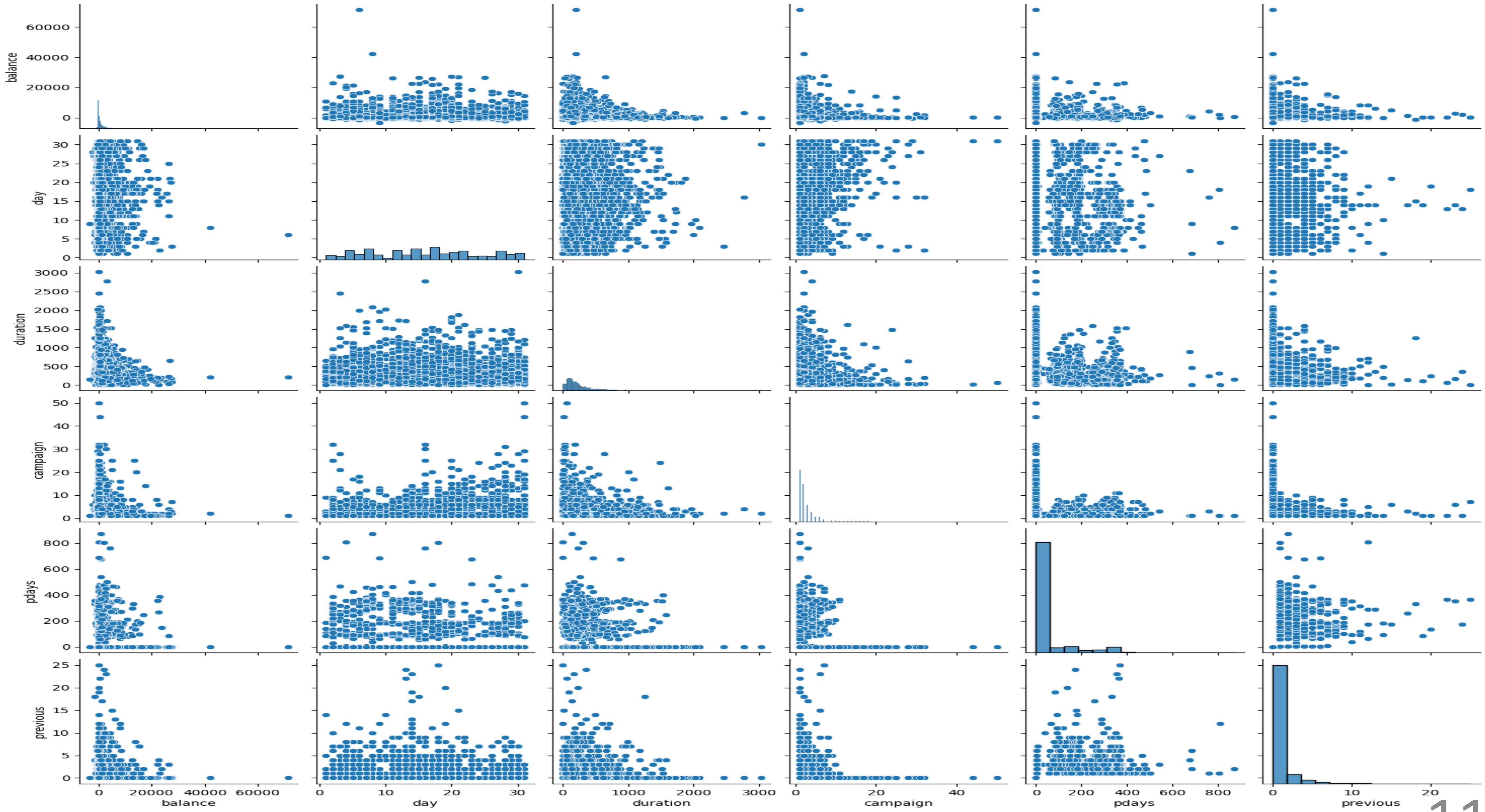


The bar graph illustrates the counts of "yes" and "no" responses for subscriptions categorized by the mode of contact, represented by the variable "contact."

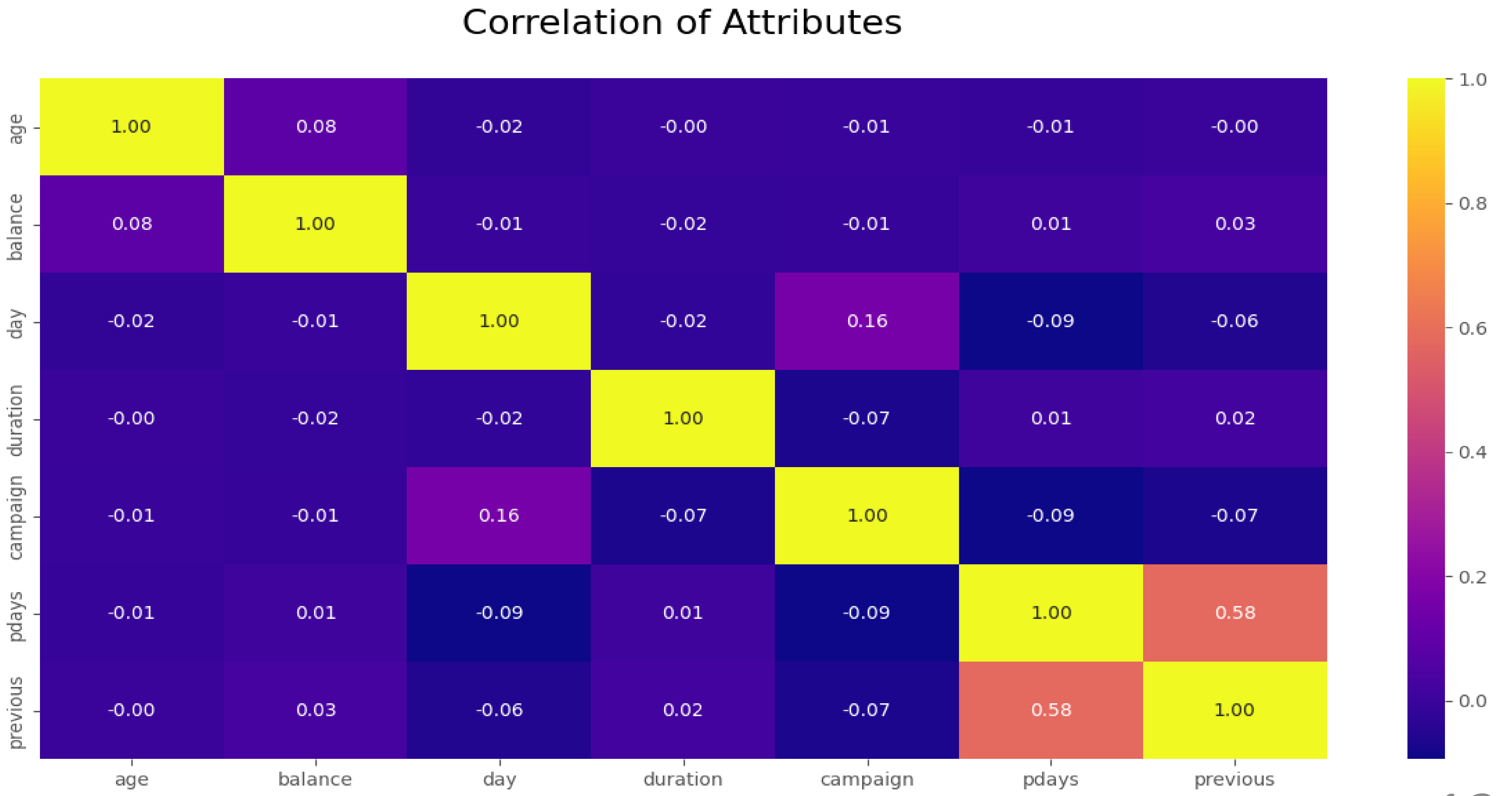
The bar graph depicts the counts of "yes" and "no" responses for subscriptions categorized by age groups.



Pair plot that includes the continuous columns of the data-frame :



HEATMAP

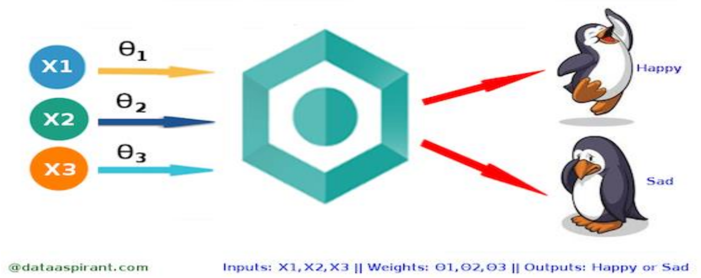

Correlation of Attributes
age
balance
day
duration
campaign
pdays
previous
1.00 0.08 -0.02 -0.00 -0.01 -0.01 -0.00
0.08 1.00 -0.01 -0.02 -0.01 0.01 0.03
-0.02 -0.01 1.00 -0.02 0.16 -0.09 -0.06
-0.00 -0.02 -0.02 1.00 -0.07 0.01 0.02
-0.01 -0.01 0.16 -0.07 1.00 -0.09 -0.07
-0.01 0.01 -0.09 0.01 -0.09 1.00 0.58
-0.00 0.03 -0.06 0.02 -0.07 0.58 1.00
1.0
0.8
0.6
0.4
0.2
0.0

VIF (Variance Inflation Factor)

	variables	VIF
0	age	1.372
1	job	1.212
2	education	1.252
3	default	1.015
4	balance	1.047
5	housing	1.247
6	loan	1.026
7	day	1.067
8	duration	1.014
9	campaign	1.088
10	pdays	4.542
11	previous	1.890

	variables	VIF
12	q_1	9.907
13	q_2	52.261
14	q_3	33.080
15	q_4	11.792
16	marital_married	2.442
17	marital_single	2.746
18	contact_telephone	1.079
19	contact_unknown	1.911
20	poutcome_other	1.364
21	poutcome_success	1.325
22	poutcome_unknown	6.651

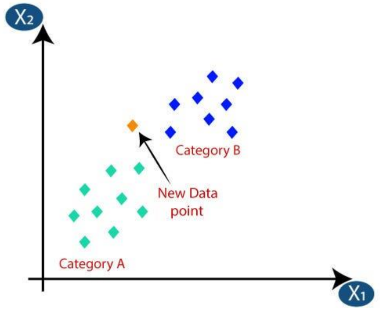
MODEL BUILDING



LOGISTIC REGRESSION

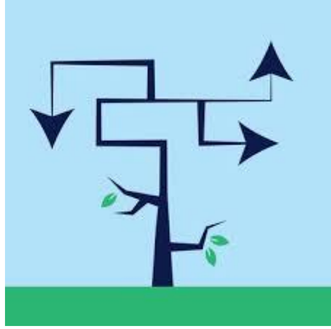
Train-Test	Accuracy
60:40	88
70:30	89
75:25	90
80:20	87

K-NEAREST NEIGHBORS (KNN)



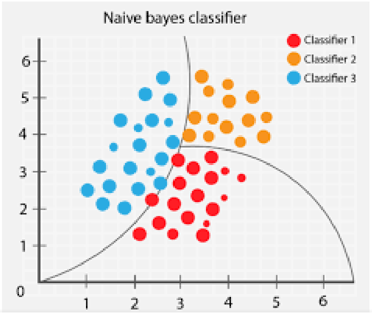
Train-Test	Accuracy
60:40	87
70:30	88
75:25	89
80:20	86

DECISION TREE



Train-Test	Accuracy
60:40	87
70:30	83
75:25	86
80:20	88

NAIVE BAYES

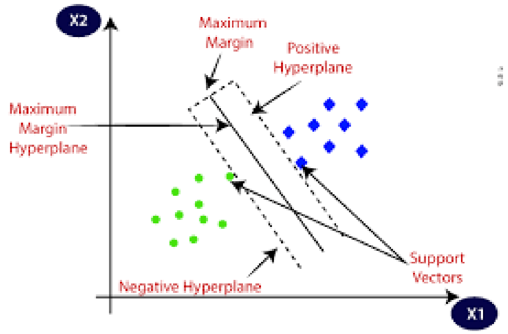


Train-Test	Accuracy
60:40	85
70:30	82
75:25	83
80:20	81

RANDOM FOREST

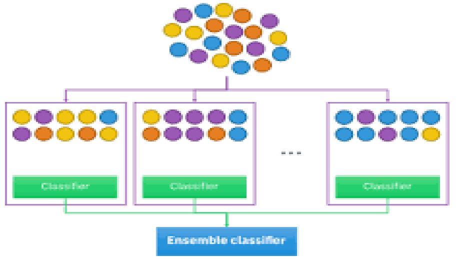
Train-Test	Accuracy
60:40	91
70:30	88
75:25	90
80:20	89

SUPPORT VECTOR MACHINE(SVM)



Train-Test	Accuracy
60:40	89
70:30	88
75:25	88
80:20	85

BAGGING (BOOTSTRAP AGGREGATING)



Train-Test	Accuracy
60:40	90
70:30	89
75:25	88
80:20	89

BOOSTING

Gradient Boosting :

Train-Test	n-estimators	Accuracy
60:40	500	91
60:40	1000	88
60:40	2000	89
70:30	500	90
70:30	1000	89
70:30	2000	88
75:25	500	89
75:25	1000	90
80:20	500	91
80:20	1000	90

AdaBoost :

Train-Test	n-estimators	Accuracy
60:40	500	89
60:40	1000	88
60:40	2000	88
70:30	500	90
70:30	1000	88
70:30	2000	87
75:25	500	91
75:25	1000	89
80:20	500	89
80:20	1000	90

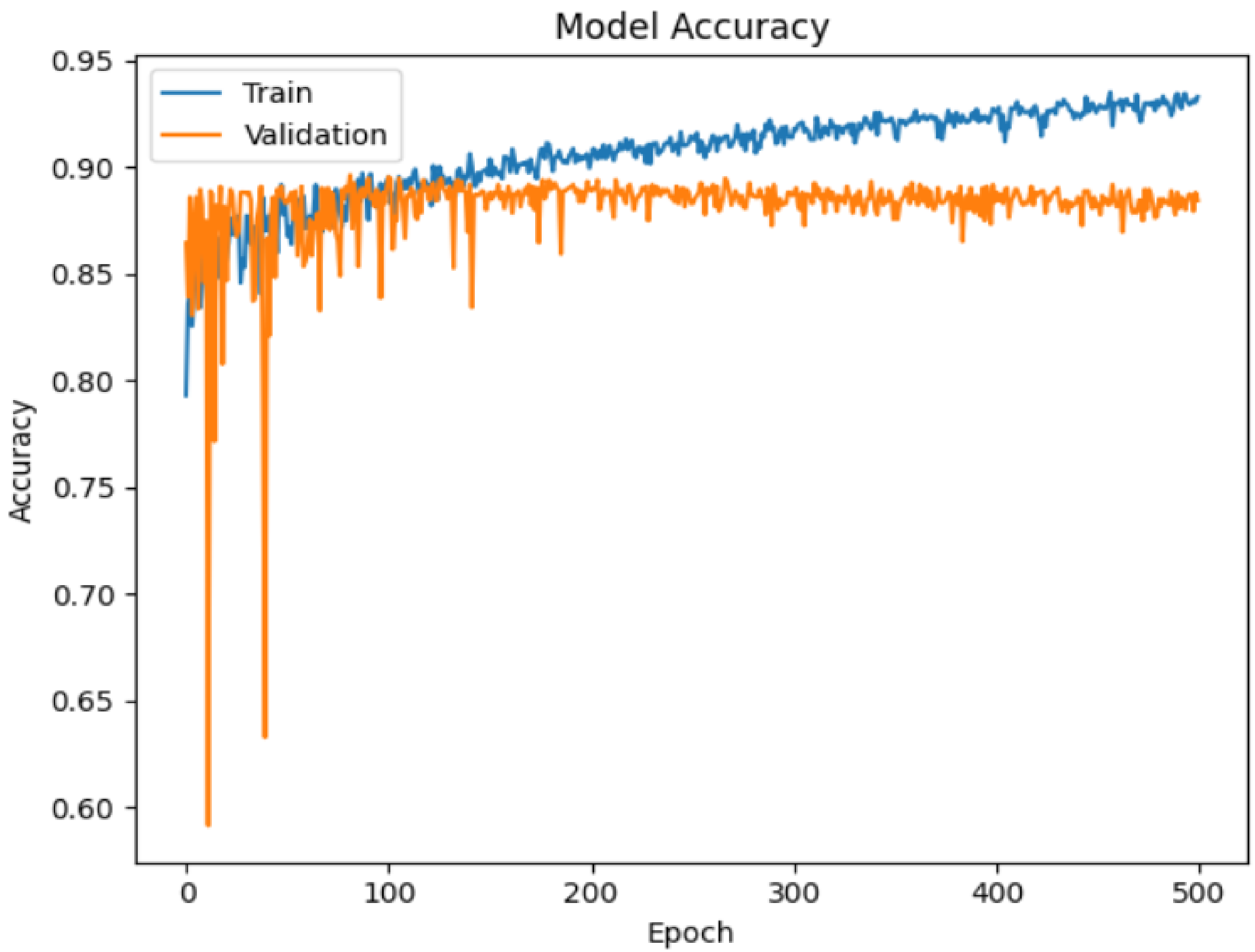
Extreme Gradient Boosting :

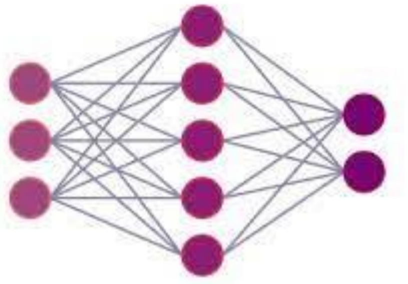
Train-Test	n-estimators	Accuracy
60:40	500	89
60:40	1000	90
60:40	2000	89
70:30	500	88
70:30	1000	90
70:30	2000	89
75:25	500	88
75:25	1000	88
80:20	500	87
80:20	1000	91

NEURAL NETWORK

Train-Test	Architecture	Epochs	Accuracy
60:40	64-32-1	50	87
60:40	64-32-1	100	88
60:40	64-32-1	199	91
60:40	64-32-1	477	92
70:30	64-32-1	50	88
70:30	64-32-1	100	89
70:30	64-32-1	190	91
70:30	64-32-1	500	93
75:25	64-32-1	50	88
75:25	64-32-1	200	90
75:25	64-32-1	500	92
80:20	64-32-1	50	89
80:20	64-32-1	200	90

Neural Network Plot-





Train-test	70-30
Optimizer	Adam
Architecture	64-32-1
Epochs	500

Comparison of the Models

MODEL	TRAIN-TEST RATIO	ACCURACY
Logistic Regression	75:25	90
K Nearest Neighbor (kNN)	75:25	89
Decision Tree	80:20	88
Naïve Bayes	60:40	85
Support Vector Machine	60:40	89
Bagging	60:40	90
Gradient Boosting	80:20	91
Extreme Gradient Boosting	75:25	91
Adaptive Boosting	80:20	91
Neural Network	70:30	93
Random Forest	60-40	91

CONCLUSION :

After application of various ML algorithms to the dataset, the best accuracy is given by Neural network i.e in the Neural Network algorithm for “70:30” train-test ratio with Accuracy-93% which is the highest among all other algorithms.

INSIGHTS:

- **Contact** is another very important feature; if you prefer **cellular contact**, it means that new customers or those participating in the campaign for the first time have a very low chance of subscribing for deposit.
- The people of age group <=60 are likely to subscribe for deposit.
- Management, blue-collar jobs, and technicians are the top three professions in which our customers are most likely to subscribe for deposit.

Colab Link:

https://colab.research.google.com/drive/1TpOZNJF_kGmRWCXU9pV4KVkxDIAF0p0r?usp=sharing

~Team Members:

S .V THEJASWINI
ANKITHA SINGH
M. PRANAV
G. MAHAVEER YADAV

APPENDIX

Training and Testing :

```
] p_cols = ['age', 'job', 'education', 'default', 'balance', 'housing', 'loan',
        'day', 'duration', 'campaign', 'pdays', 'previous','q_1', 'q_2',
        'q_3', 'q_4', 'marital_married', 'marital_single', 'contact_telephone',
        'contact_unknown', 'poutcome_other', 'poutcome_success',
        'poutcome_unknown']
```

```
] X = dummy[p_cols]
  y = dummy.y
```

```
] #train_test_split
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=2)
  X_train.shape,X_test.shape

  ((3164, 23), (1357, 23))
```

Dummy Variable Encoding:

```python
# Mapping job categories to binary values
job_mapping = {
    'unknown': 0,
    'services': 0,
    'unemployed': 0,
    'self-employed': 0,
    'entrepreneur': 0,
    'student': 0,
    'retired': 0,
    'management': 1,
    'blue-collar': 0,
    'technician': 1,
    'admin.': 1,
    'housemaid': 0
}

# Applying the mapping to the 'job' column
df['job'] = df['job'].map(job_mapping)
```

```python
# Mapping months to quarters
month_mapping = {
    'jan': 1, 'feb': 1, 'mar': 1,
    'apr': 2, 'may': 2, 'jun': 2,
    'jul': 3, 'aug': 3, 'sep': 3,
    'oct': 4, 'nov': 4, 'dec': 4
}
# Convert 'month' column to strings and create 'quarter' column based on the mapping
df['quarter'] = df['month'].astype(str).map(month_mapping)

# Use pd.get_dummies to create binary columns for each quarter
quarters_dummies = pd.get_dummies(df['quarter'], prefix='q')

# Concatenate the original DataFrame with the new binary columns
df = pd.concat([df, quarters_dummies], axis=1)

# Drop the original 'month' and 'quarter' columns if needed
df = df.drop(['month', 'quarter'], axis=1)
```

```python
df['education'].replace(['unknown','primary','secondary','tertiary'],[0,1,2,3],inplace=True)
df['default'].replace(['no','yes'],[0,1],inplace=True)
df['housing'].replace(['no','yes'],[0,1],inplace=True)
df['loan'].replace(['no','yes'],[0,1],inplace=True)
df['y'].replace(['no','yes'],[0,1],inplace=True)
```

Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)


# Display the results
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.90
```

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
# Create a logistic regression model
logreg = LogisticRegression()

# Fit the model to the training data
logreg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logreg.predict(X_test)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)


 # Print results
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.89
```

K Nearest Neighbour :

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
# Create a KNN model with k=3 (you can adjust this parameter)
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the model to the training data
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.89
```

Decision Tree :

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred = model.predict(X_test)
y_pred
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.83
```

Naïve Bayes :

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import  accuracy_score
```

```
# Create a Gaussian Naive Bayes classifier
clf = GaussianNB()
clf.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = clf.predict(X_test)
y_pred
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.83
```

Support Vector Machine:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)

# Train the SVM classifier on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_classifier.predict(X_test)

# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8828297715549005
```

Bagging :

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```python
# Create a Decision Tree base classifier
base_classifier = DecisionTreeClassifier()

# Create a Bagging Classifier with the base classifier
bagging_classifier = BaggingClassifier(base_classifier, n_estimators=500, random_state=0)
```

```python
bagging_classifier.fit(X_train, y_train)
```

```
▸ BaggingClassifier
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

```python
y_pred = bagging_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.90
```

Boosting :

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
```

```
# Create a Gradient Boosting Classifier with desired hyperparameters
gbc = GradientBoostingClassifier(n_estimators=1000, learning_rate=0.5, max_depth=3, random_state=0)

gbc.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.5, n_estimators=1000, random_state=0)
```

```
y_pred = gbc.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.90
```

XGBoost

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
```

```
# Create an XGBoost classifier with desired hyperparameters
xgb_classifier = XGBClassifier(n_estimators=500, learning_rate=0.1, max_depth=3, random_state=0)
xgb_classifier.fit(X_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=500, n_jobs=None,
              num_parallel_tree=None, random_state=0, ...)
```

```
y_pred = xgb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9060254284134881
```

AdaBoost

```python
# Import necessary libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```python
# Initialize the AdaBoost model with a Decision Tree as the base estimator
base_model = DecisionTreeClassifier(max_depth=1)
adaboost_model = AdaBoostClassifier(base_model, n_estimators=1000, random_state=0)
```

```python
# Train the AdaBoost model
adaboost_model.fit(X_train, y_train)
```

```
AdaBoostClassifier
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

```python
# Make predictions on the test set
y_pred = adaboost_model.predict(X_test)
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0])
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.89
```

Neural Network

```python
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
```

```python
# Build the neural network model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```python
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/50
85/85 [==============================] - 1s 4ms/step - loss: 16.3041 - accuracy: 0.7596 - val_loss: 1.0587 - val_accuracy: 0.8358
Epoch 2/50
85/85 [==============================] - 0s 2ms/step - loss: 1.7200 - accuracy: 0.8263 - val_loss: 0.7854 - val_accuracy: 0.8441
Epoch 3/50
85/85 [==============================] - 0s 2ms/step - loss: 1.1260 - accuracy: 0.8341 - val_loss: 1.3622 - val_accuracy: 0.8878
Epoch 4/50
85/85 [==============================] - 0s 2ms/step - loss: 2.0107 - accuracy: 0.8285 - val_loss: 1.2257 - val_accuracy: 0.8823
Epoch 5/50
85/85 [==============================] - 0s 3ms/step - loss: 1.1169 - accuracy: 0.8426 - val_loss: 0.8453 - val_accuracy: 0.8546
Epoch 6/50
85/85 [==============================] - 0s 4ms/step - loss: 1.9322 - accuracy: 0.8289 - val_loss: 1.2301 - val_accuracy: 0.8950
Epoch 7/50
85/85 [==============================] - 0s 3ms/step - loss: 1.4092 - accuracy: 0.8282 - val_loss: 0.7196 - val_accuracy: 0.8917
Epoch 8/50
```

```
Epoch 48/50
85/85 [==============================] - 0s 2ms/step - loss: 1.3110 - accuracy: 0.8414 - val_loss: 0.7689 - val_accuracy: 0.8961
Epoch 49/50
85/85 [==============================] - 0s 2ms/step - loss: 0.5573 - accuracy: 0.8662 - val_loss: 0.5166 - val_accuracy: 0.8872
Epoch 50/50
85/85 [==============================] - 0s 2ms/step - loss: 0.5338 - accuracy: 0.8695 - val_loss: 1.0147 - val_accuracy: 0.8994
```

```python
# Assuming you have more than two classes
# Make predictions on the test set
y_pred_proba = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_labels = np.argmax(y_pred_proba, axis=1)

# Now you can use y_pred_labels for evaluation
```

```
57/57 [==============================] - 0s 865us/step
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_labels)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.90
```

```python
import matplotlib.pyplot as plt
# Plot the training history
plt.figure(figsize=(12, 5))

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
```