UCI Bank Marketing

Rawan Alharbi
Raghad Alarifi

- **Problem Definition**

- One of the Portuguese banking institution conducted a marketing campaign based on phone calls from 2008 to 2010. The records of their efforts are available in the form of a dataset. The objective here is to apply machine learning techniques to analyse the dataset and figure out most effective tactics that will help the bank in next campaign to persuade more customers to subscribe to banks term deposit. The dataset contains various categorical and numerical features with 11162 data sample. The data is labelled. So supervised machine learning algorithms are applicable to this project. The objective is to predict whether the client will subscribe to a term deposit or not. Data pre-processing is done along with suitable exploratory data analysis. Results of various algorithms are compared at the end.

- **Methodology**

- The dataset class is labelled as ‘yes’ or ‘no’ depending on whether the contacted client has subscribed to the deposit or not. It is a marketing problem and the outcome will largely influence the future strategies of bank. Banking institute has a very large client base and even larger target clients. In real world , less clients will respond positively to marketing campaign and most of them will say no. Contacting all of them is time consuming task and demands tremendous time and efforts. To manage the human resource in efficient way, it is necessary to correctly identify those clients who have more chances of saying yes. This is where machine learning comes into picture.

Feature Description & Exploratory Data Analysis

Data

Bank client data :

1. Age (numeric)
2. Job : type of job (categorical): 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3. marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4. education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5. default: has credit in default? (categorical: 'no','yes','unknown')
6. housing: has housing loan? (categorical: 'no','yes','unknown')
7. loan: has personal loan? (categorical: 'no','yes','unknown')

Data

Related with the last contact of the current campaign :

contact: contact communication type (categorical)

month: last contact month of year (categorical)

day_of_week: last contact day of the week (categorical)

duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Data

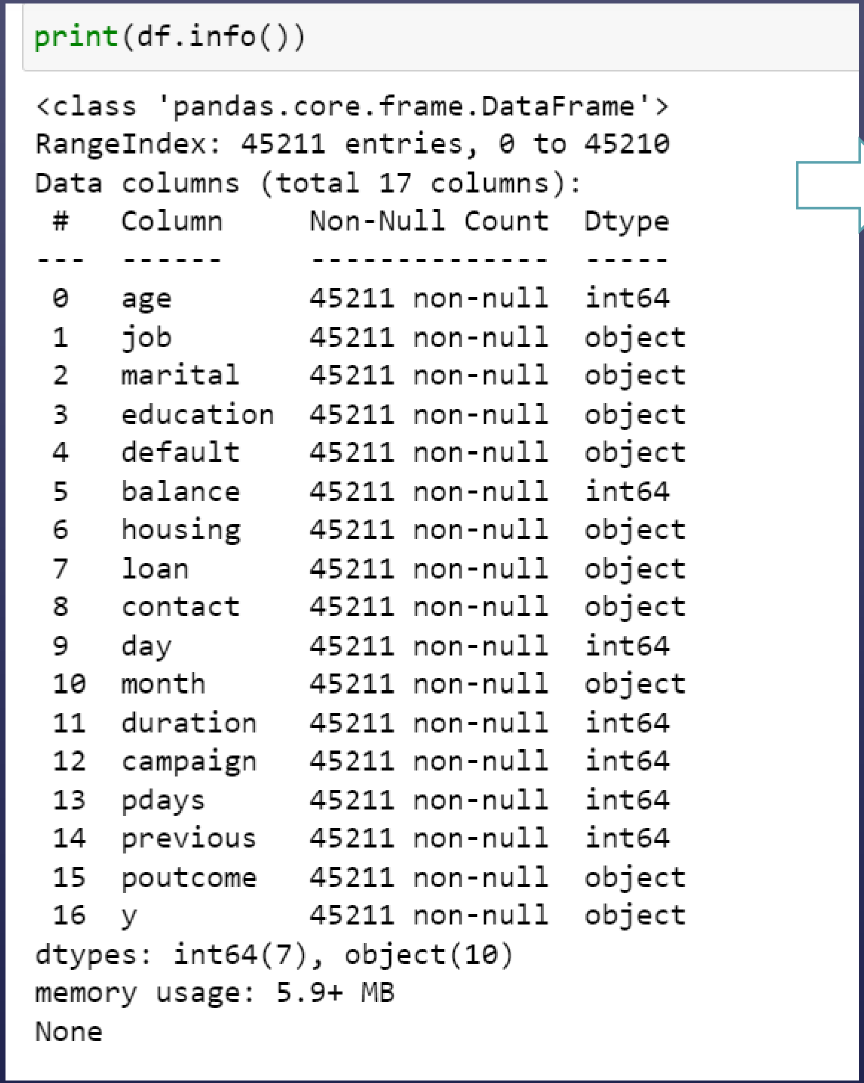
Other attributes

1. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
2. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
3. previous: number of contacts performed before this campaign and for this client (numeric)
4. poutcome: outcome of the previous marketing campaign (categorical)
5. Social and economic context attributes
6. emp.var.rate: employment variation rate - quarterly indicator (numeric)
7. cons.price.idx: consumer price index - monthly indicator (numeric)
8. cons.conf.idx: consumer confidence index - monthly indicator (numeric)
9. euribor3m: euribor 3-month rate - daily indicator (numeric)
10. nr. employed: number of employees - quarterly indicator (numeric)

Output variable (target)

y - has the client subscribed a term deposit? (Binary: 'yes','no’)

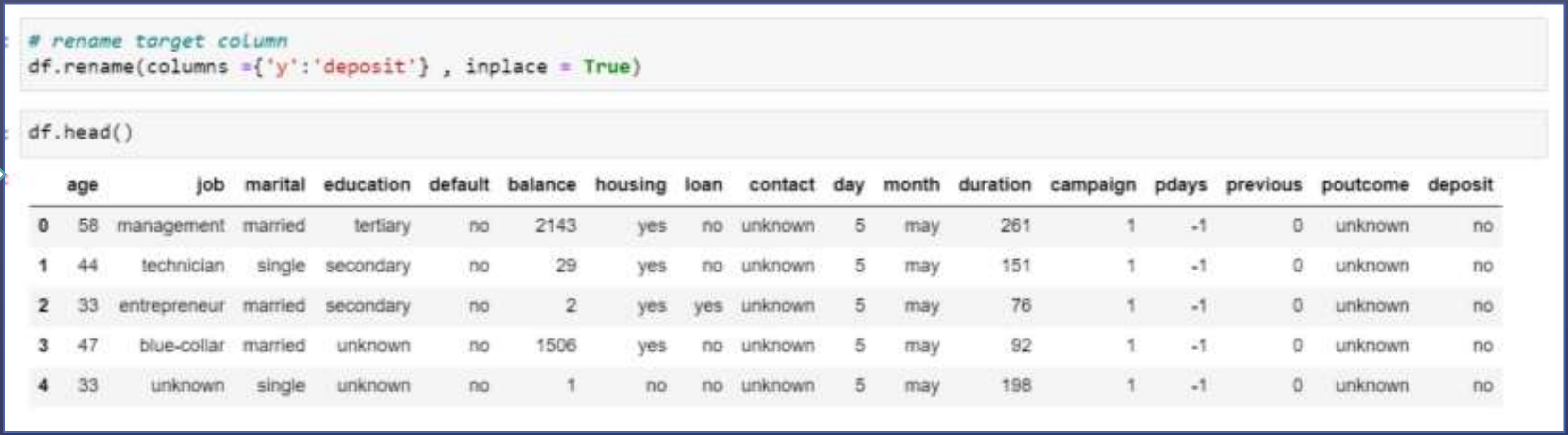
Present Data Information :



```
print(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
None
```

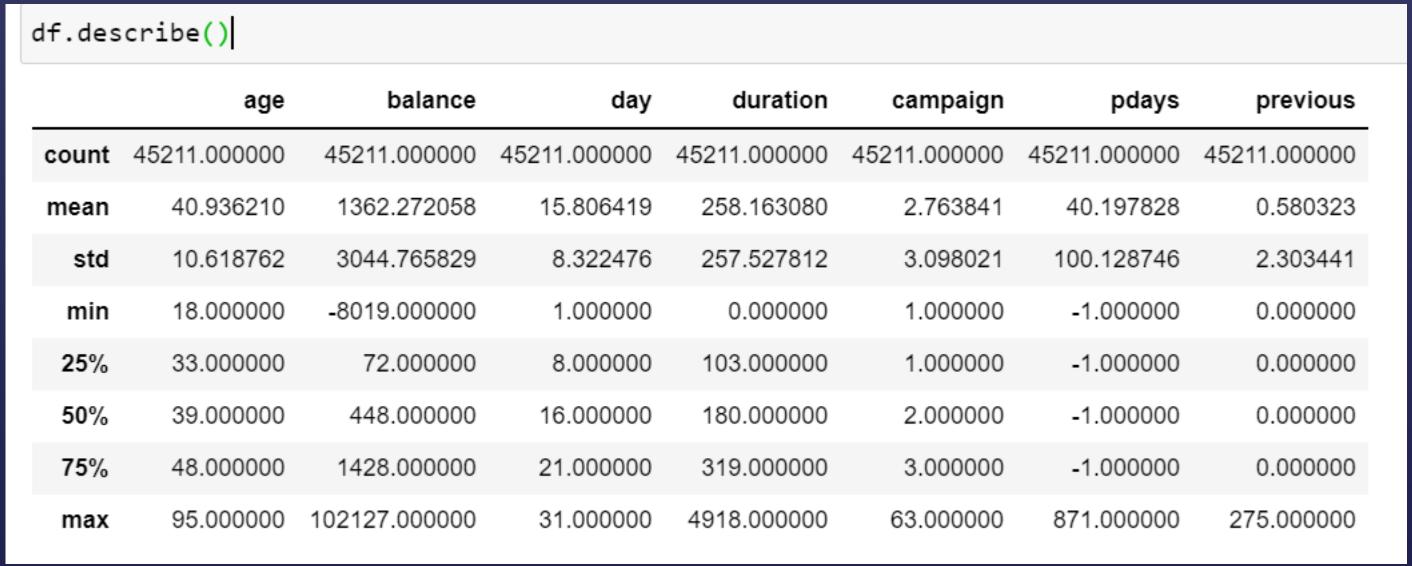
Rename target column :



```
# rename target column
df.rename(columns ={'y':'deposit'} , inplace = True)
```

```
df.head()
```

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



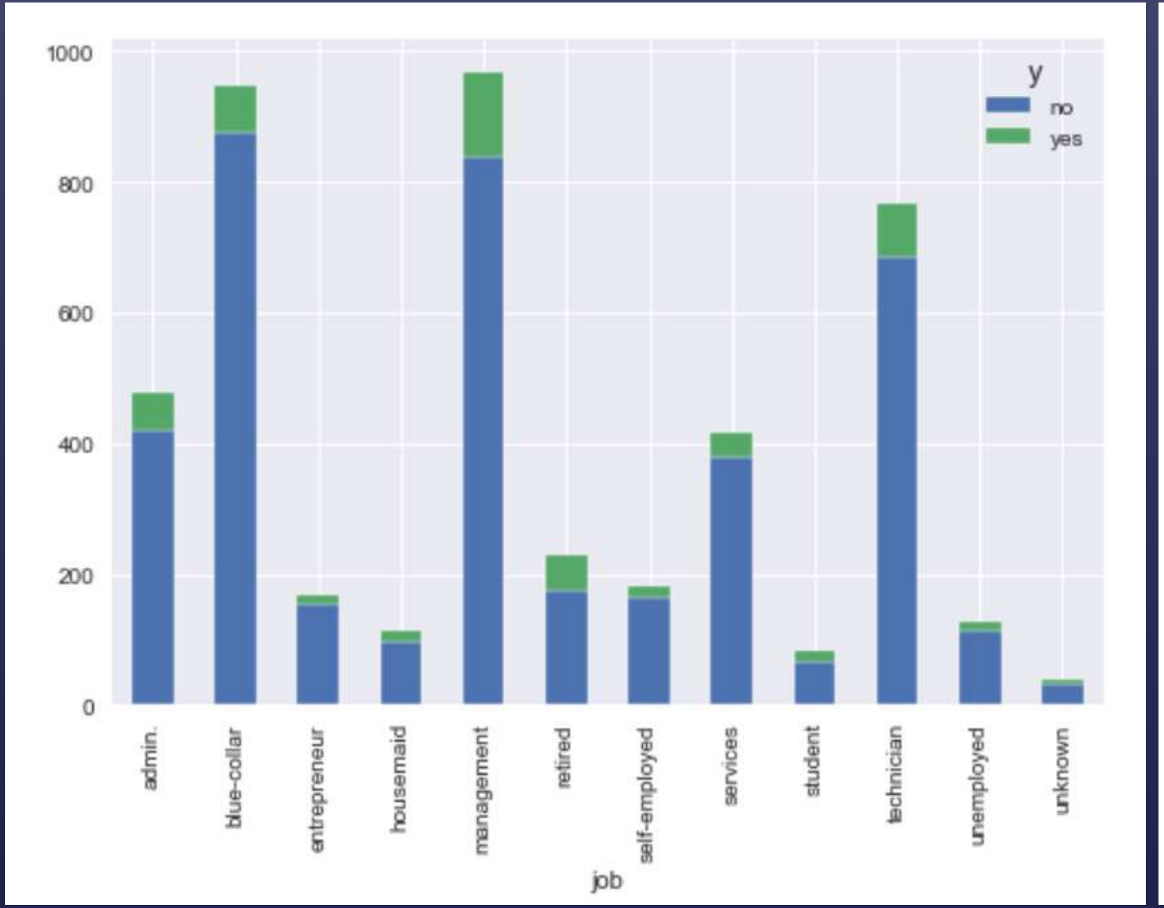
```
df.describe()
```

	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

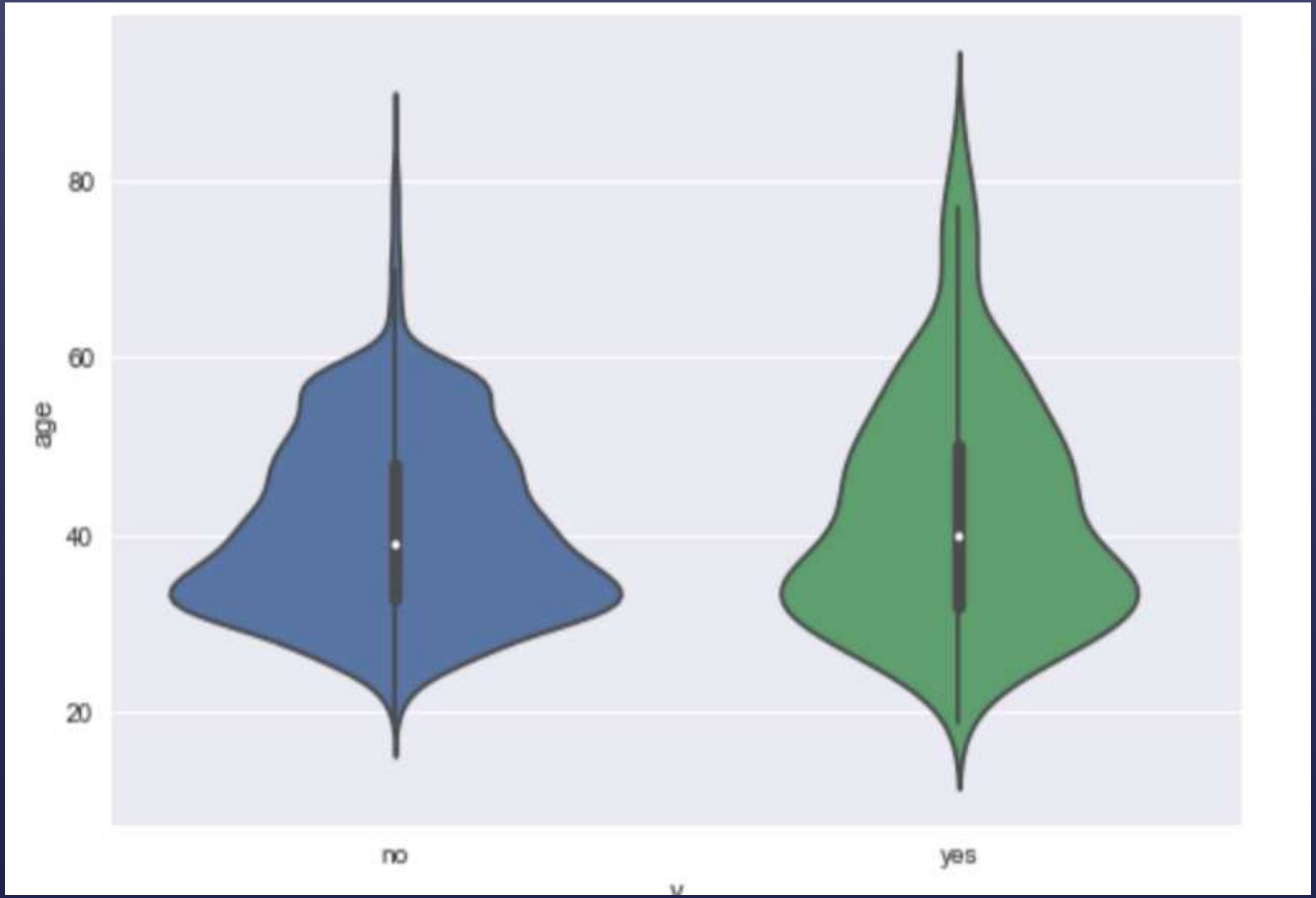
- info tells us that there are no null values in dataset
- only 7 features are numerical
- 10 fetaures are categorical

Job , Age

Job of Client

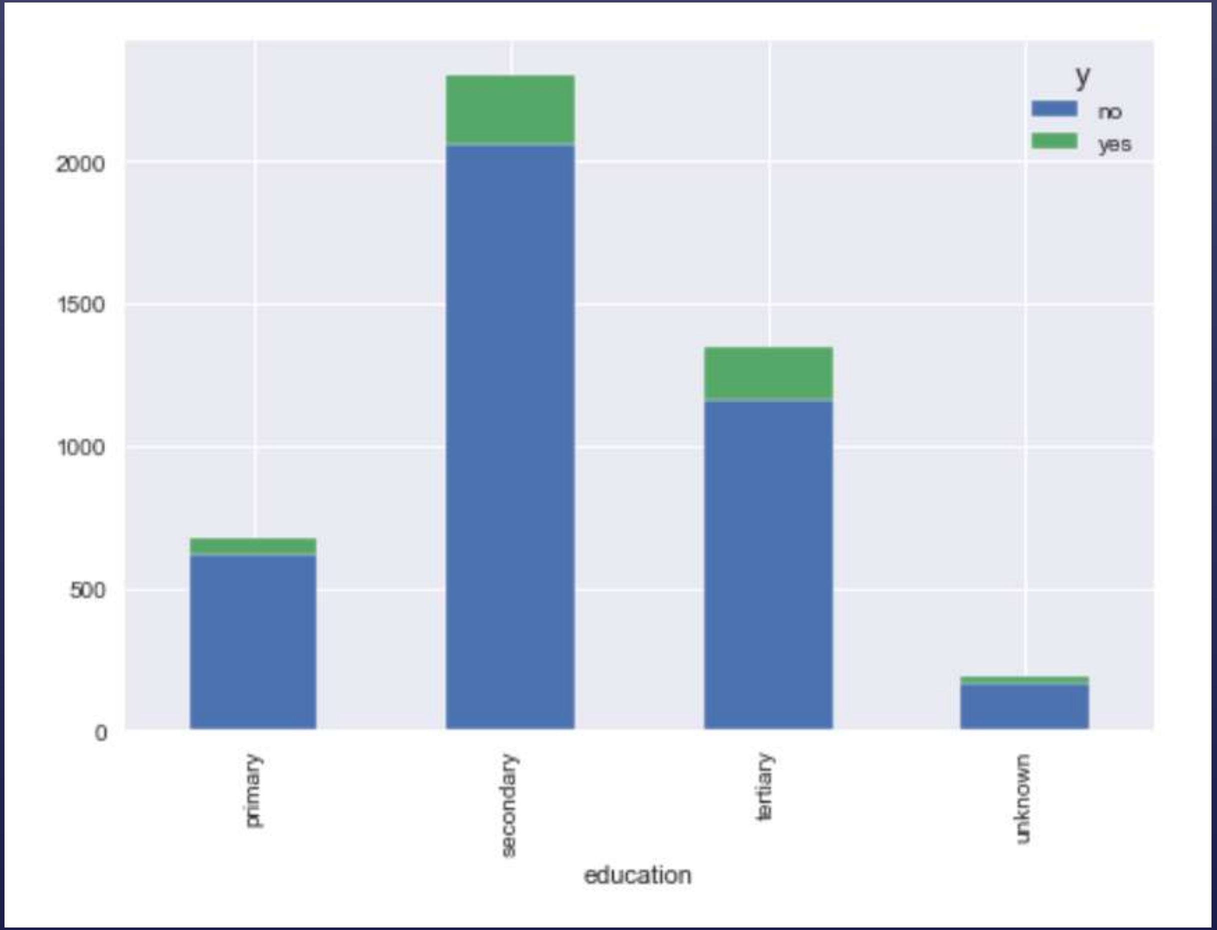


Age of Client

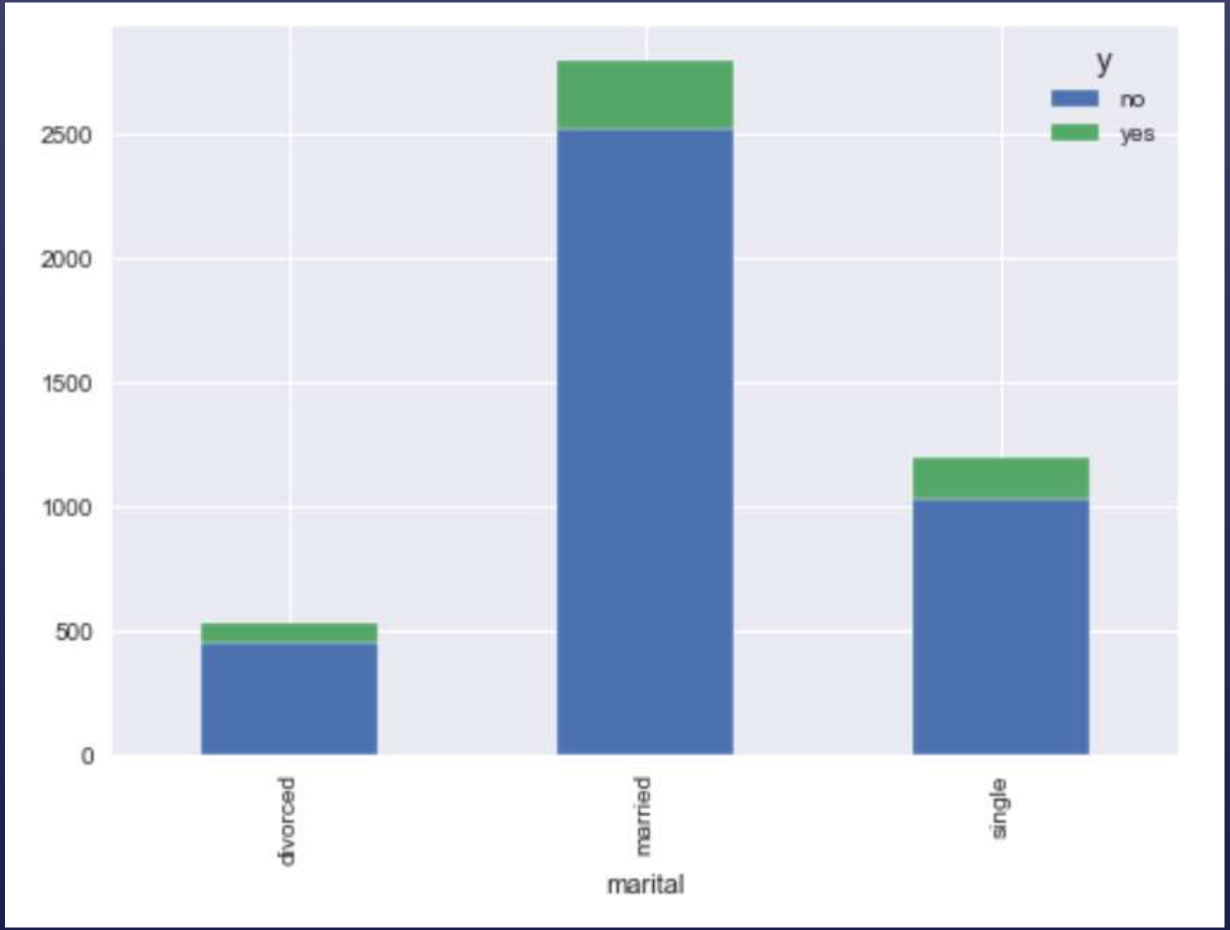


Marital , Education

Marital status of Client



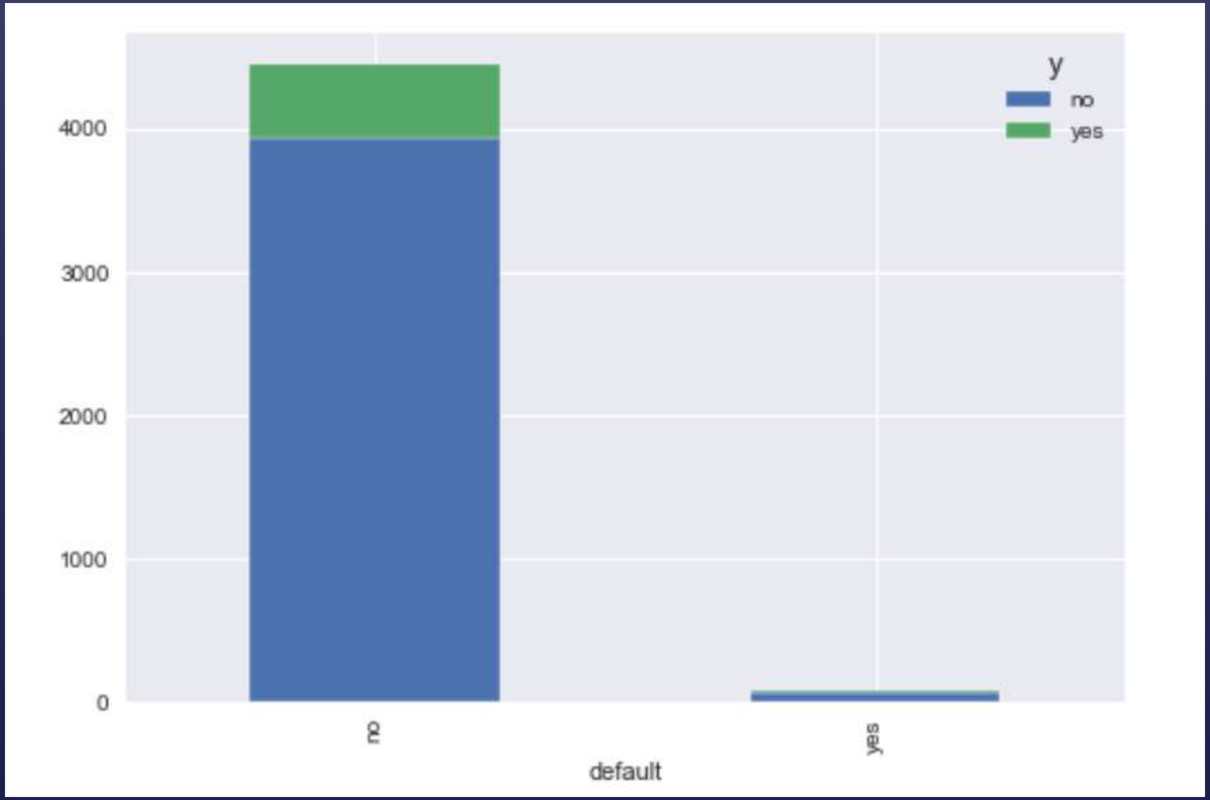
Education Type of Client

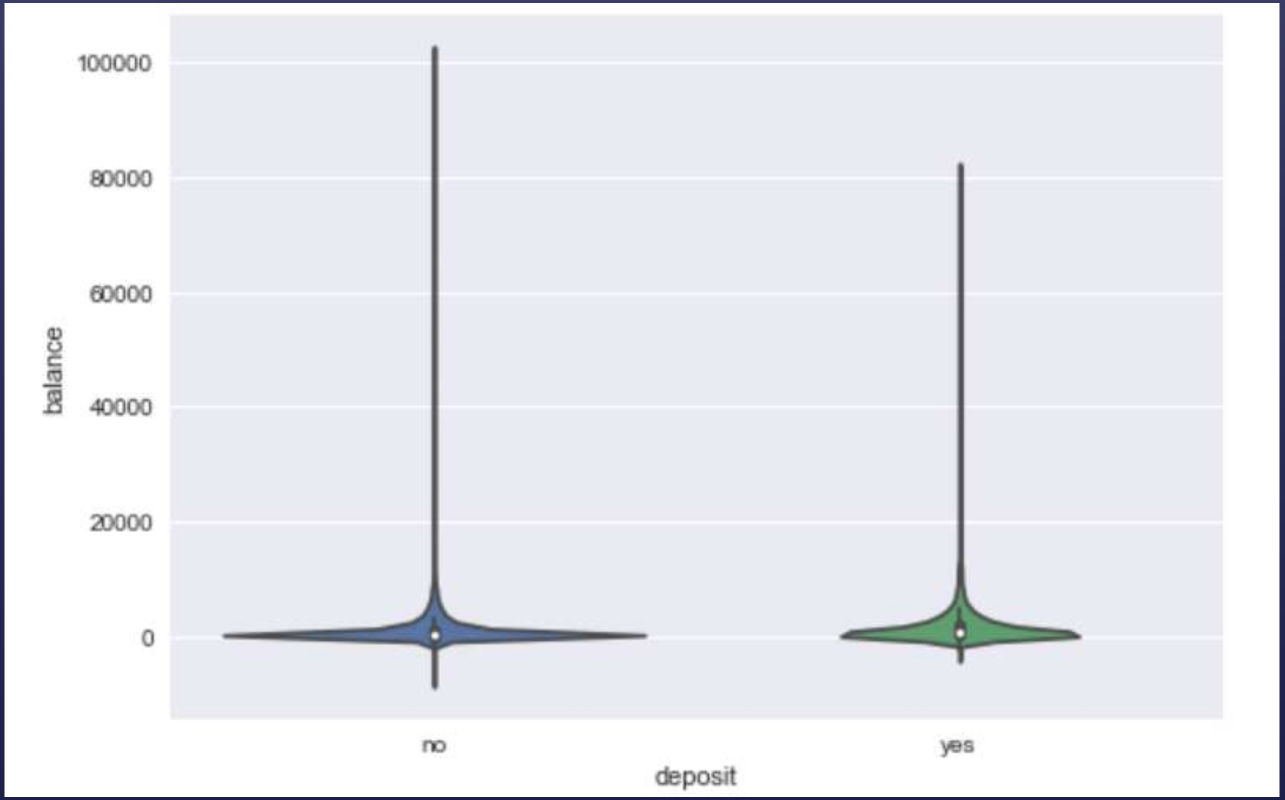


Default, Balance

Default - it tells whether the client has credit in bank or not?

Balance in bank account

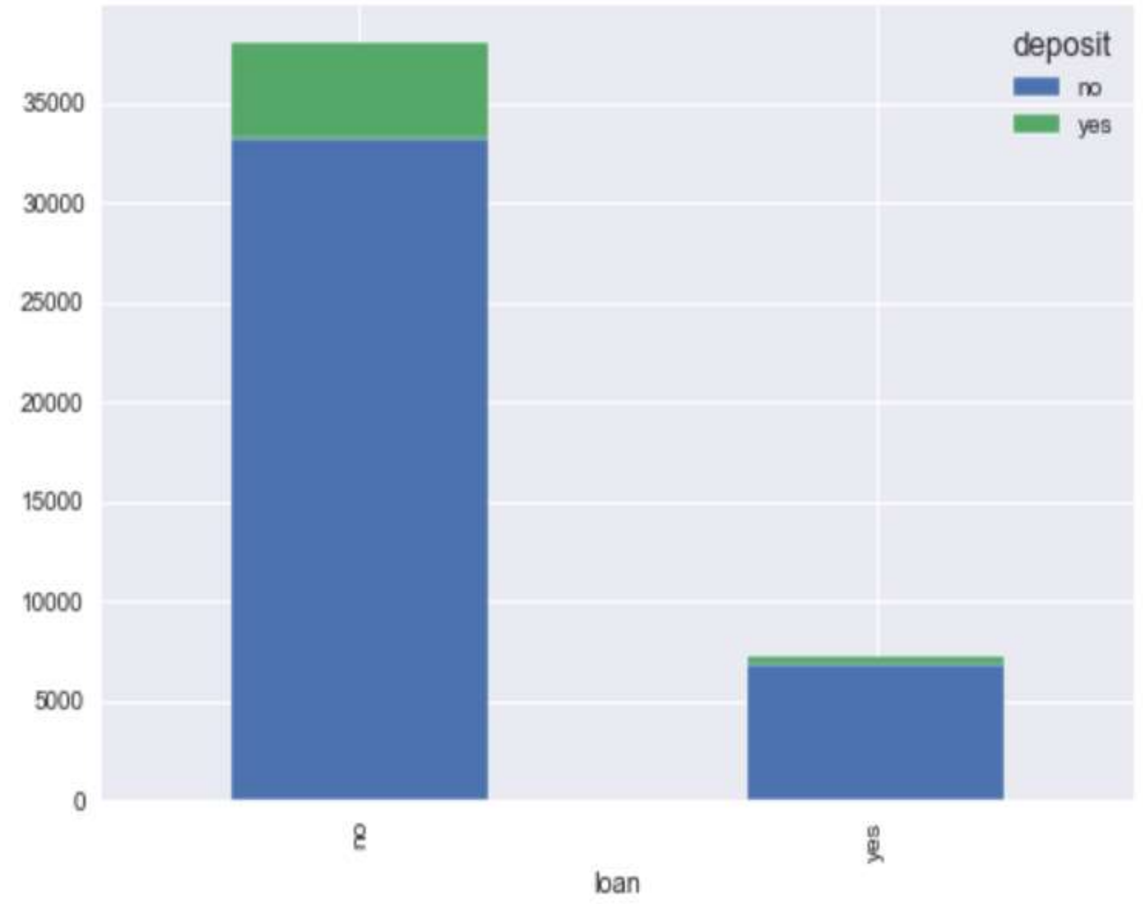


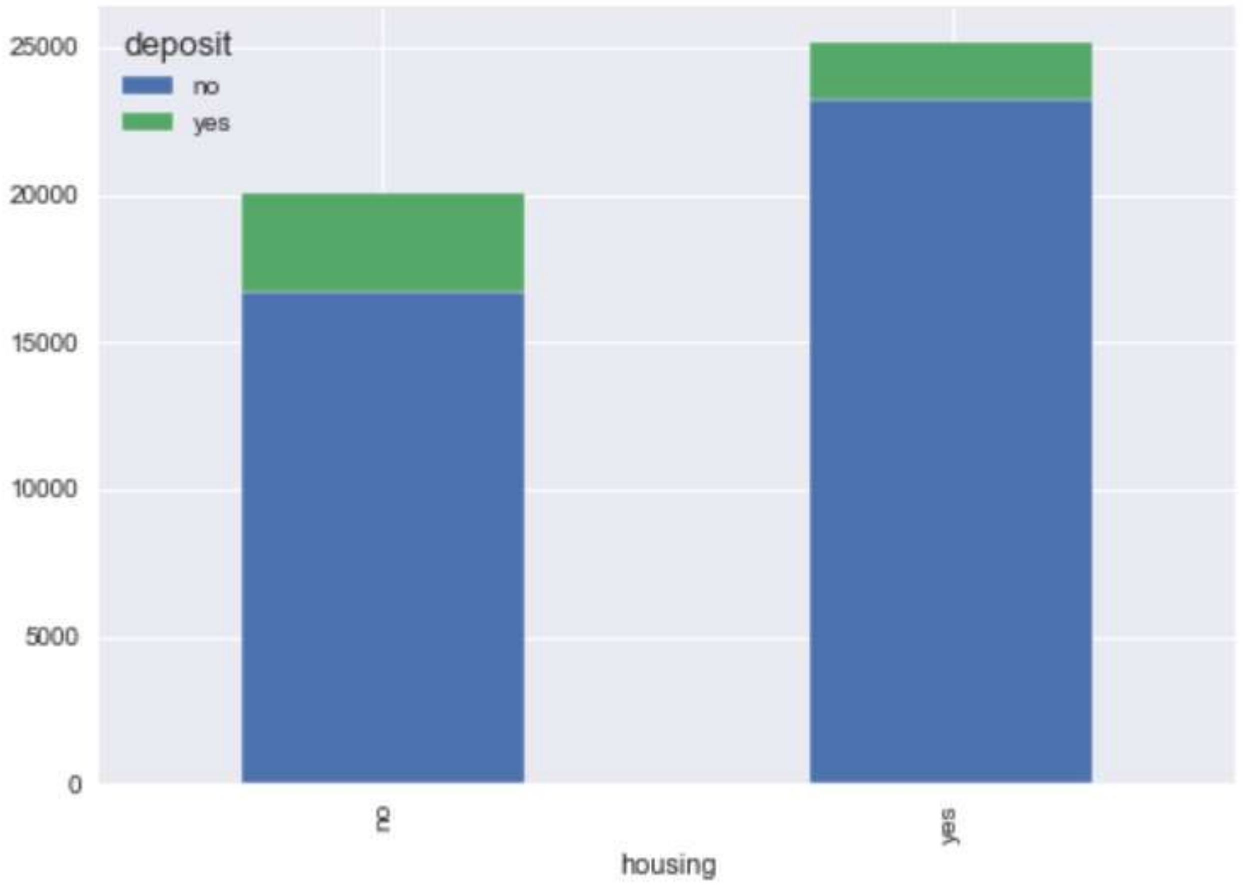


Housing, Loan

Loan - Whether the client has got any personal loan from bank ?

Housing – Whether the client has got any housing loan from bank ?

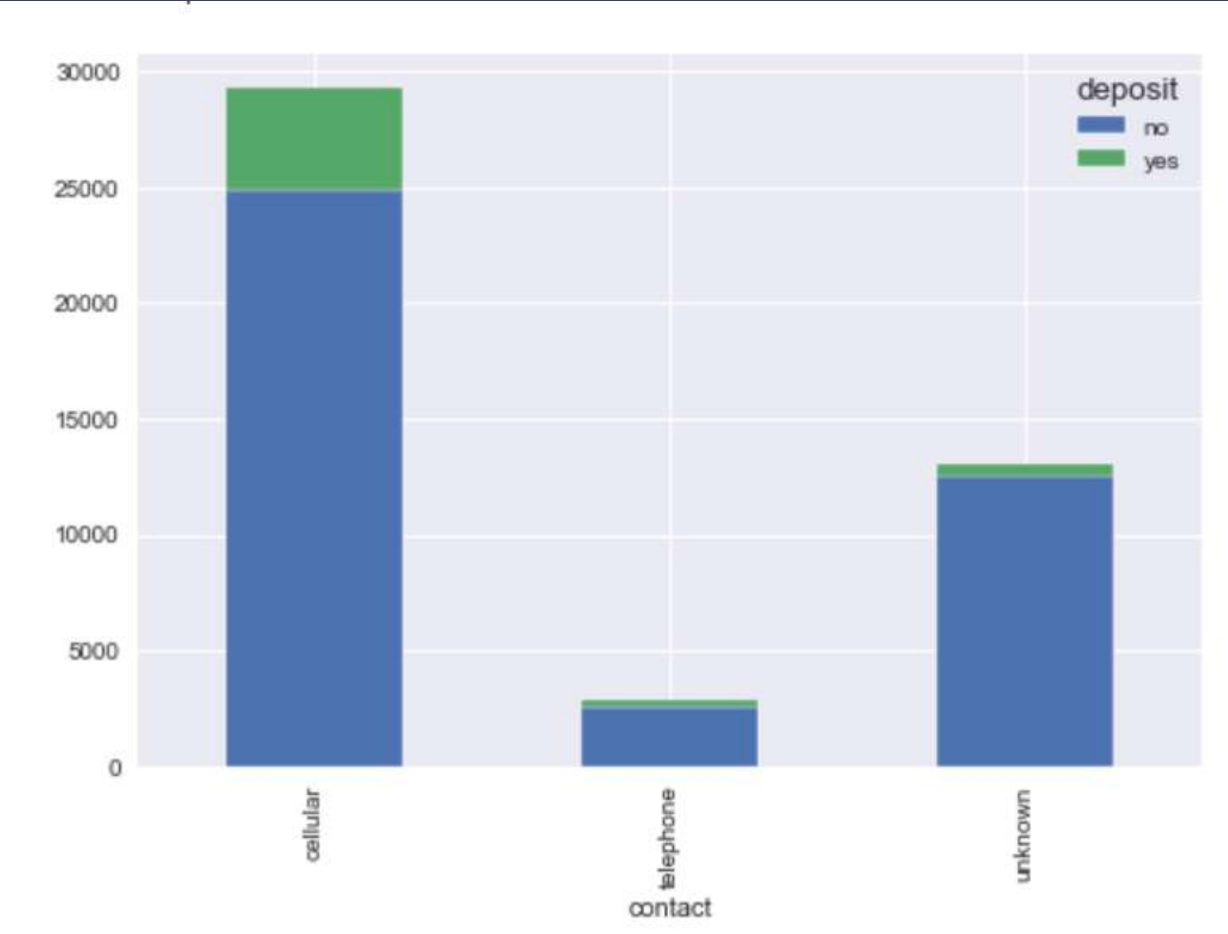


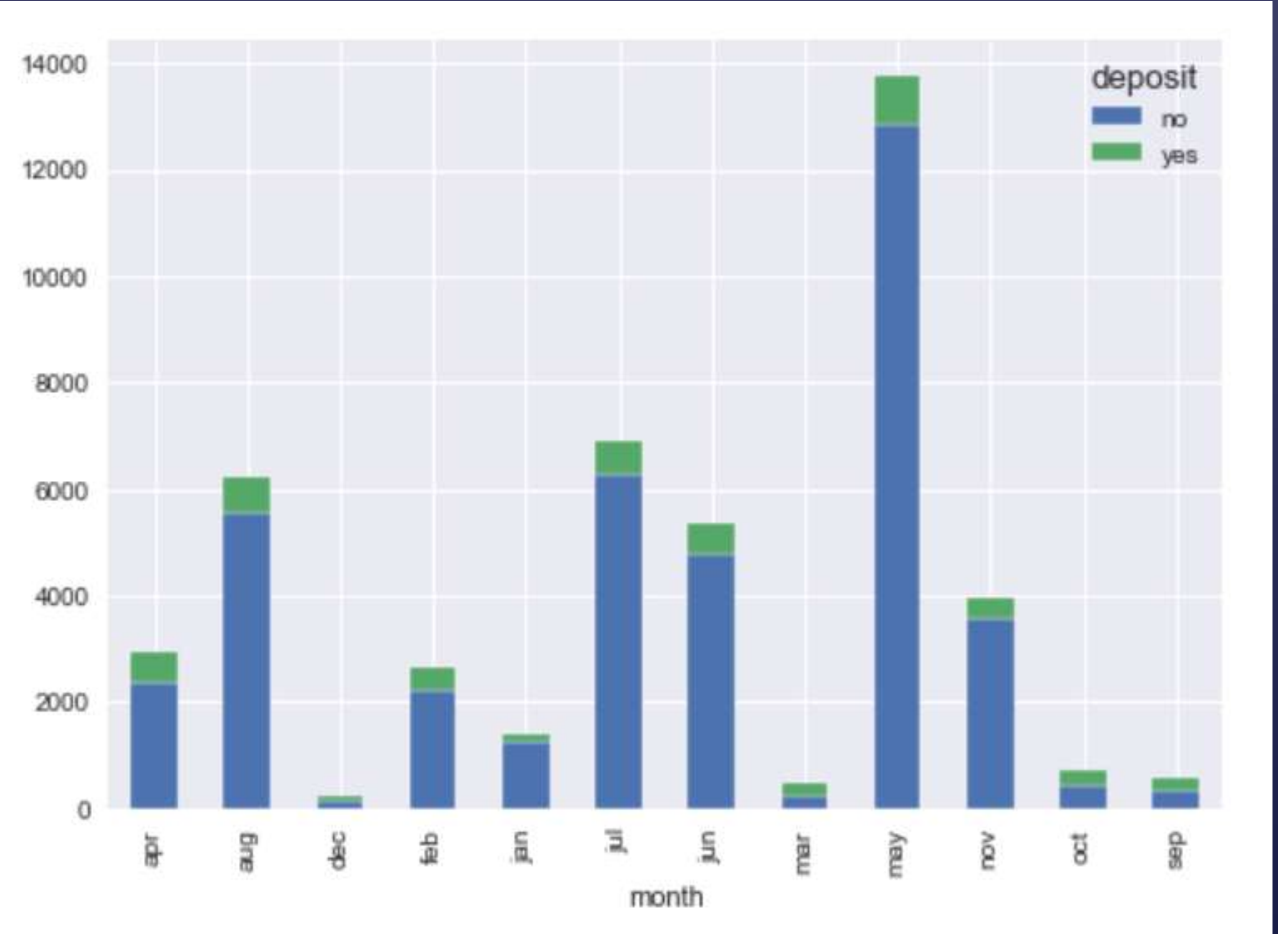


Contact, Month

Contact – way of communication

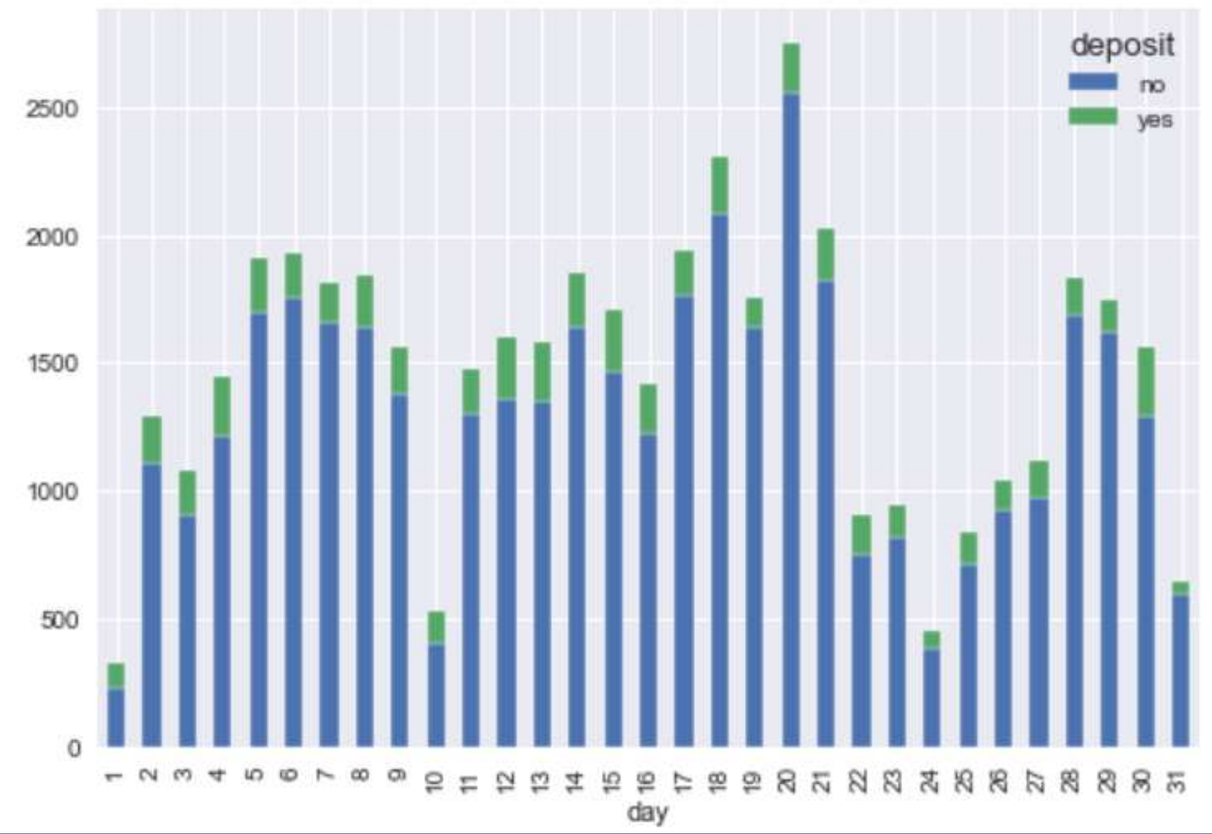
Month – Month of communication



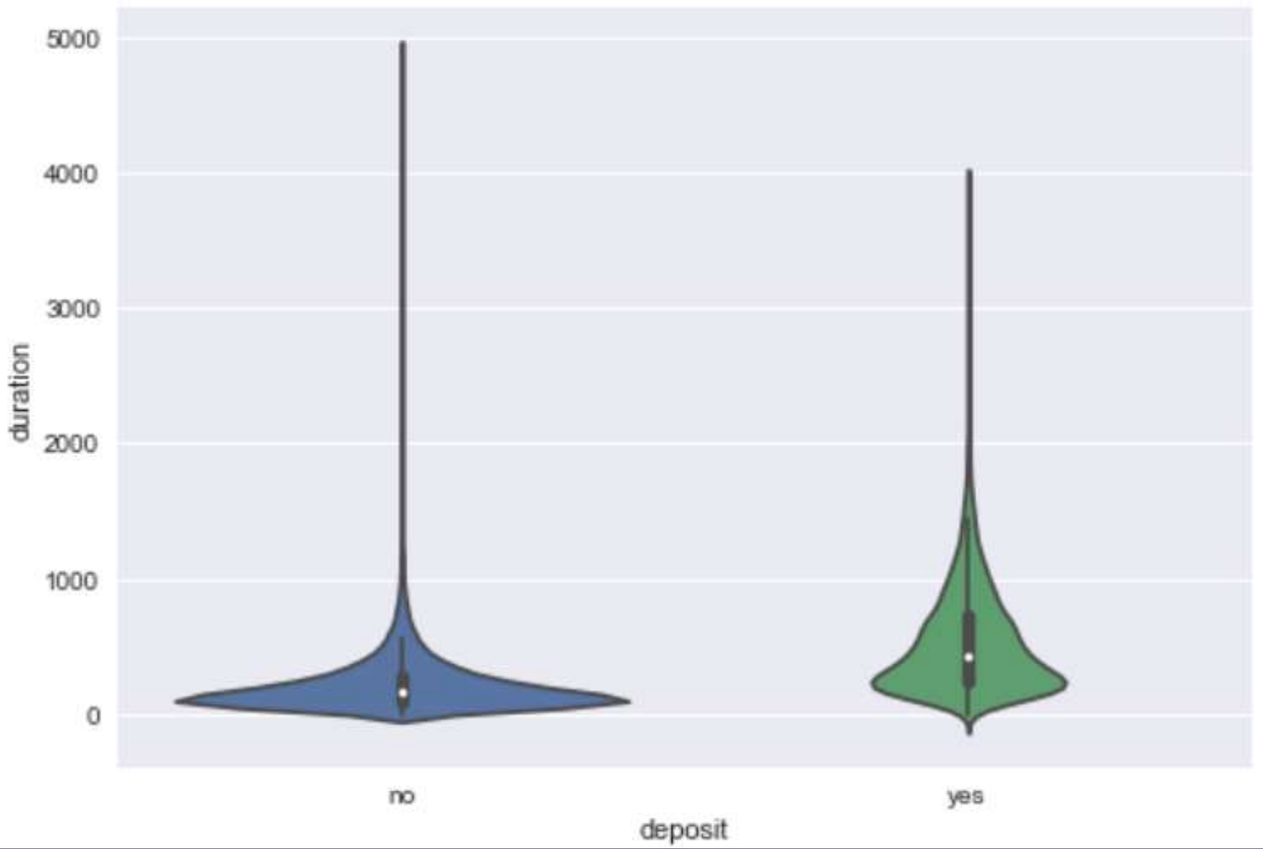


Day, Duration

Day – day of the month for contact



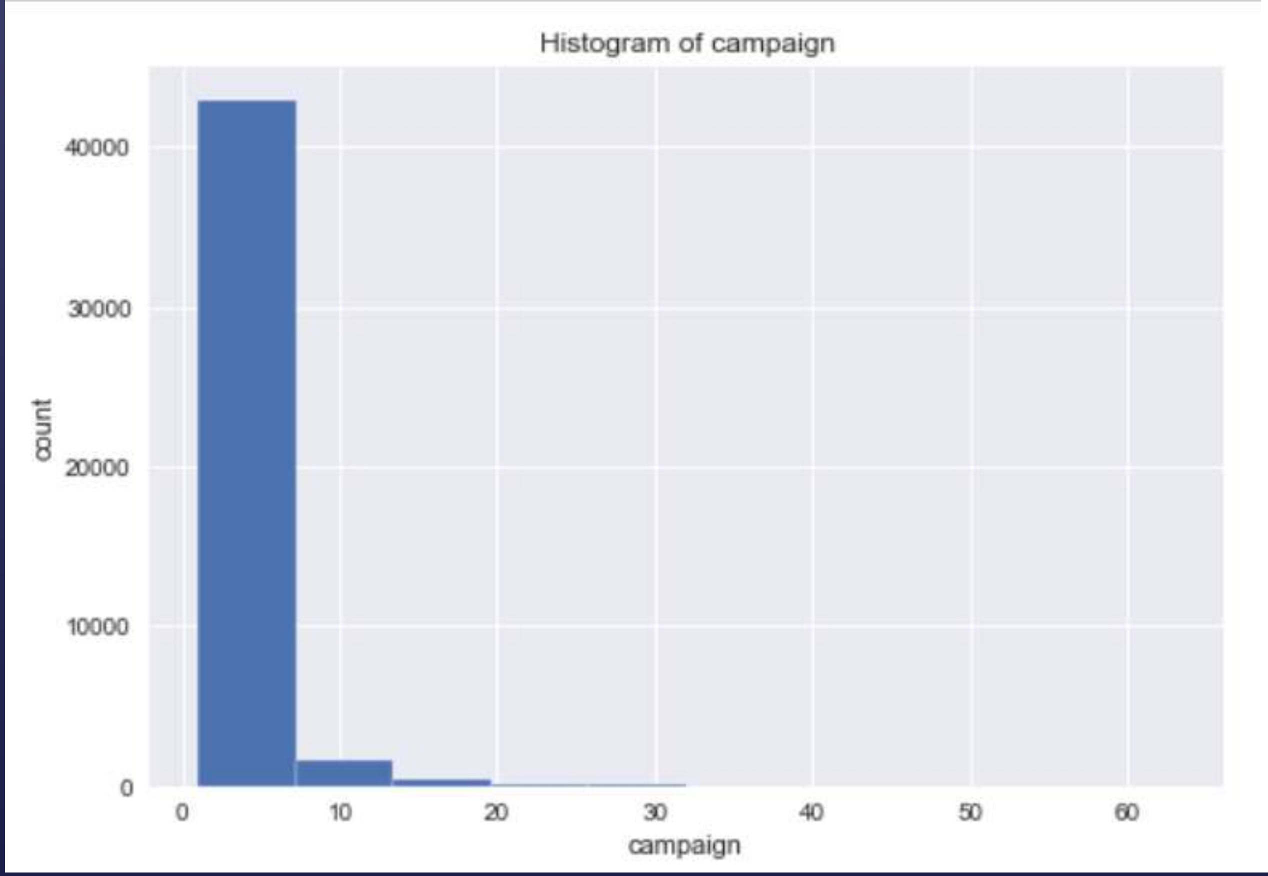
Duration – duration of last call

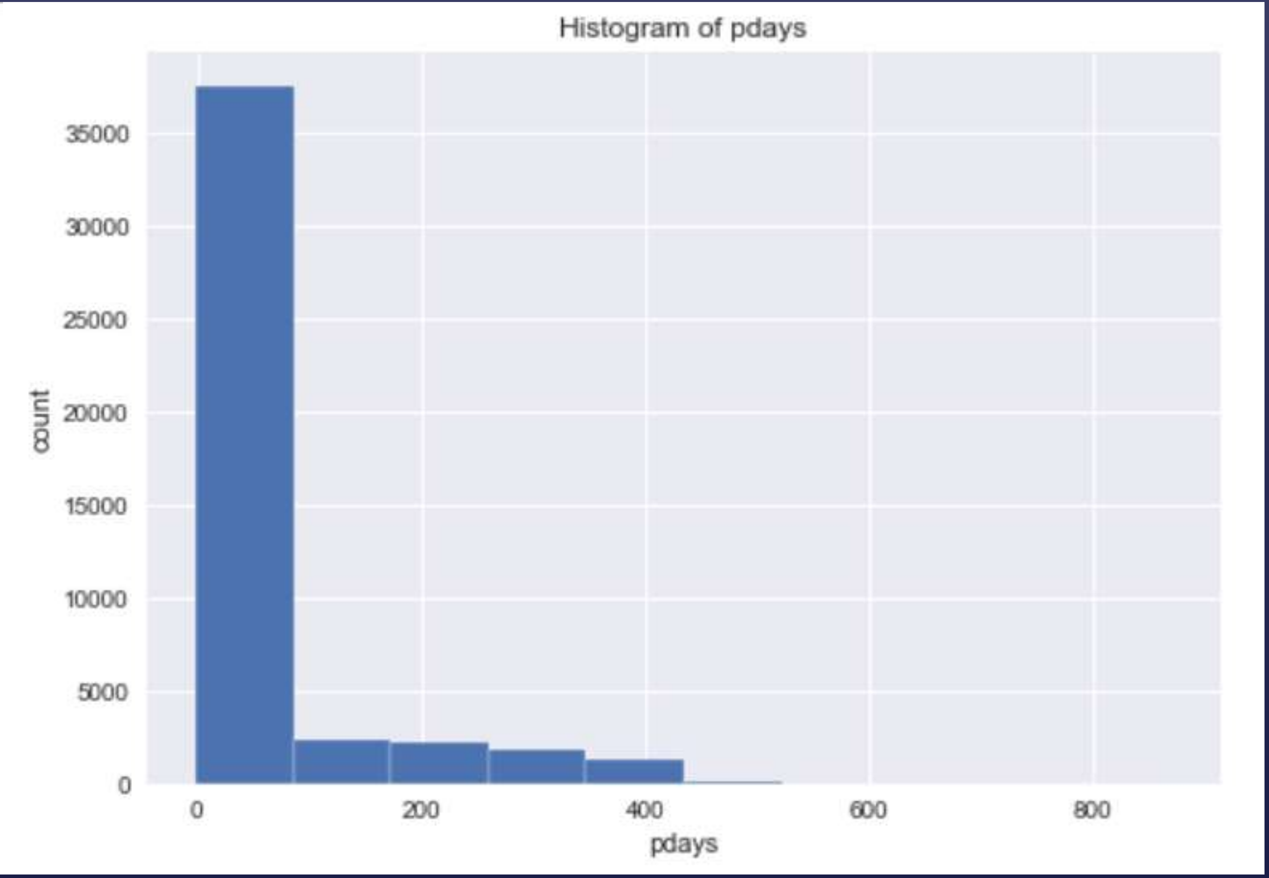


Campaign, pdays

Campaign – Number of times this client was contacted during this campaign

Pdays – number of days that passed after the client was last contacted in previous campaign

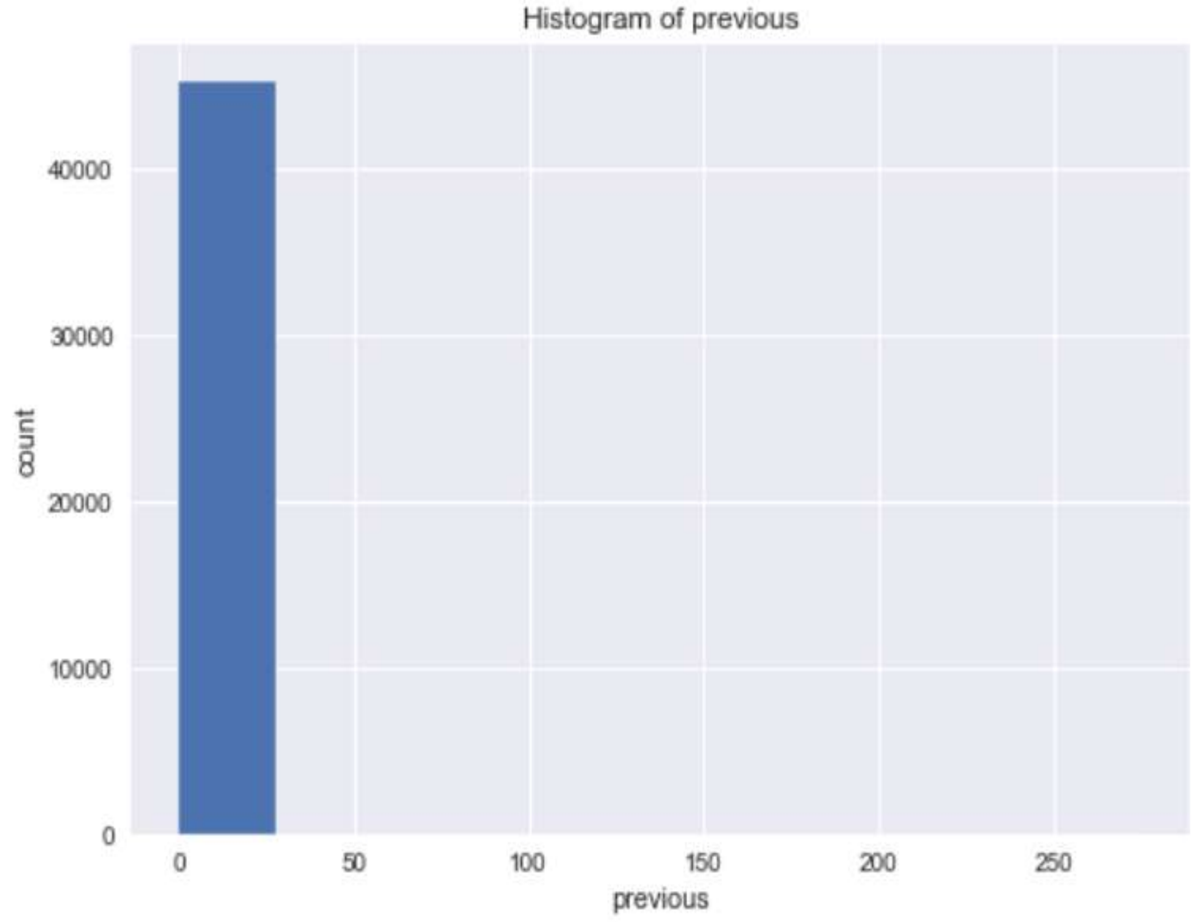


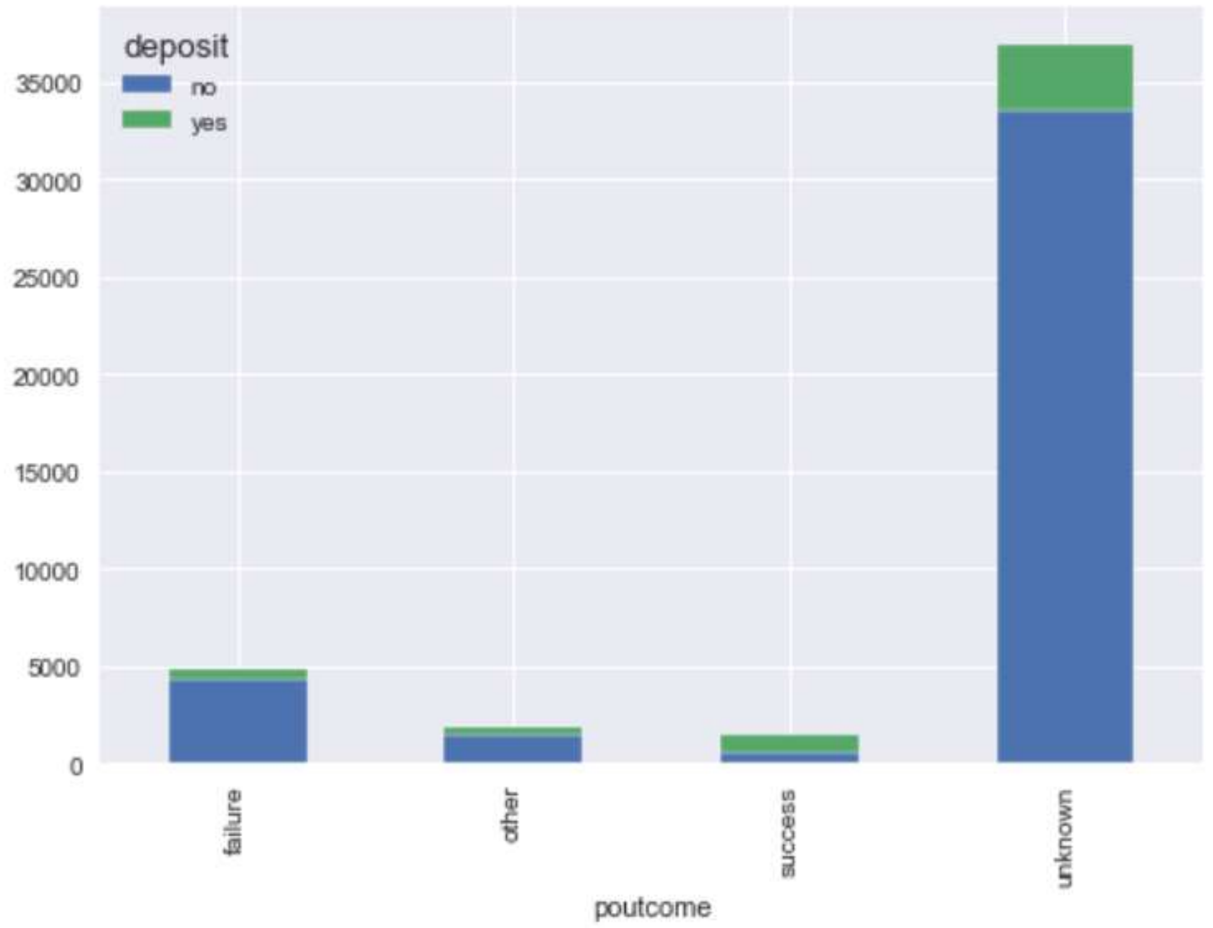


Previous, poutcome

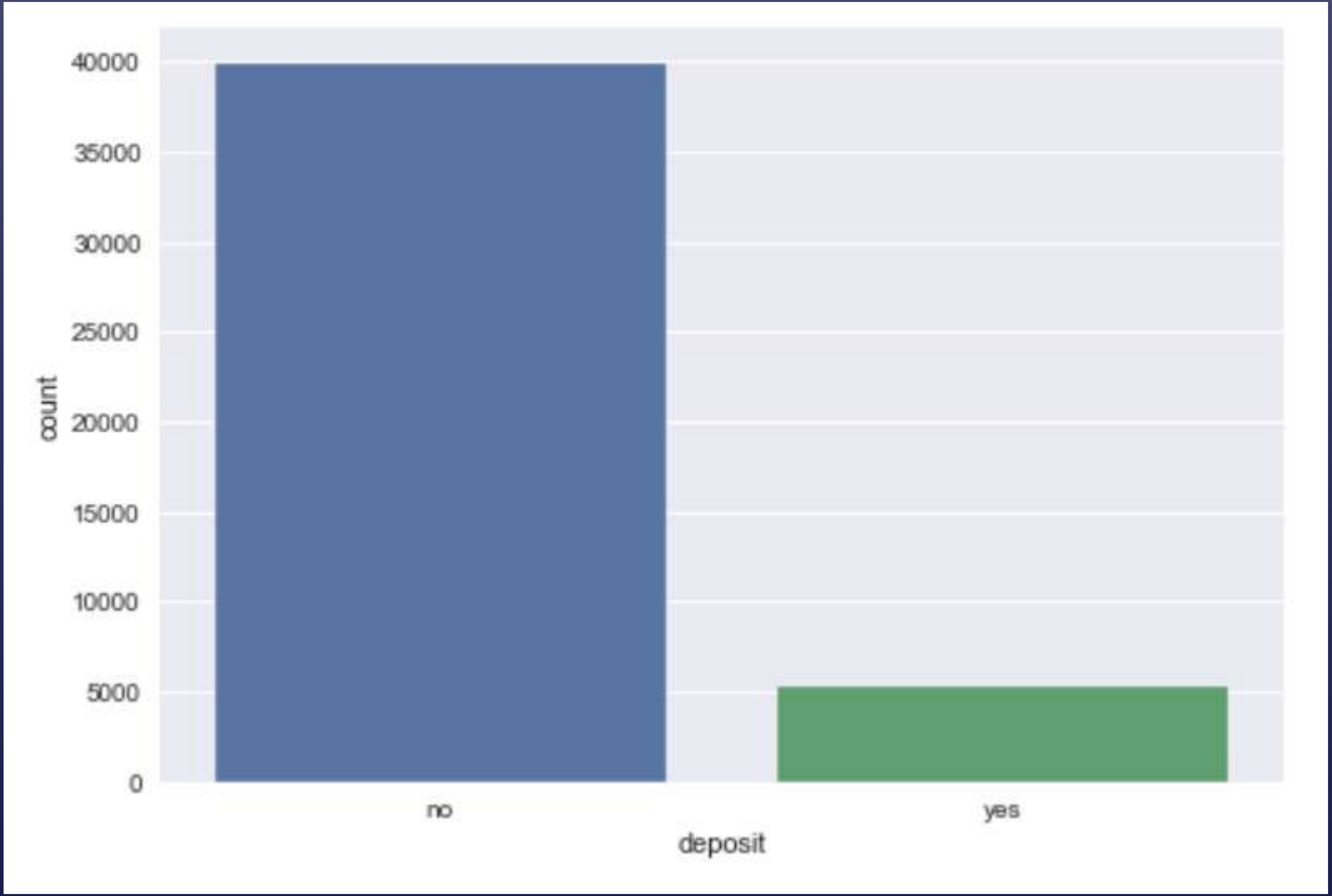
Previous – Number of times this client was contacted before this campaign

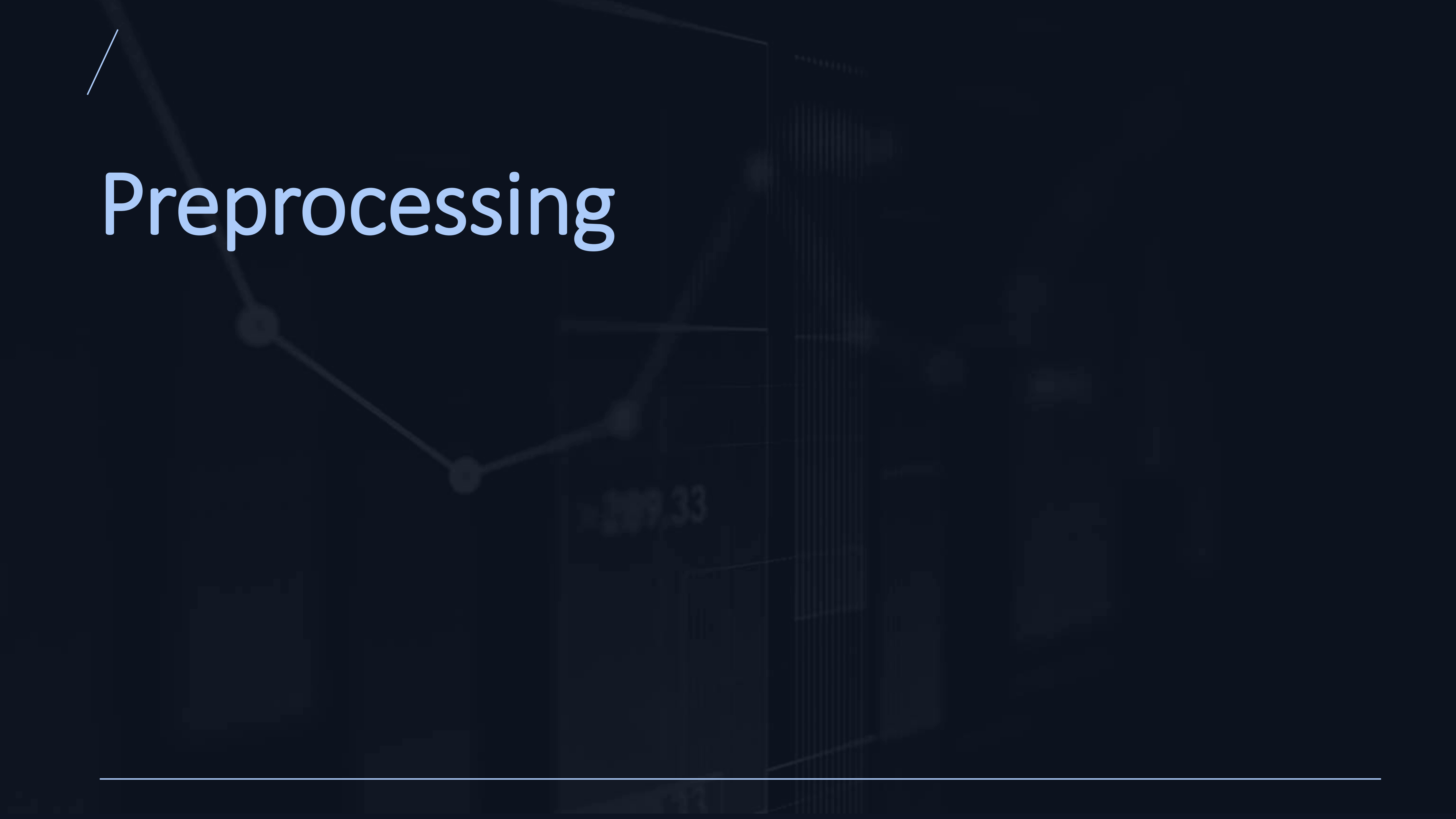
Poutcome – The outcome of previous marketing campaign





Deposit (Target Variable)





Preprocessing

Algorithms:

Feature Engineering:

- Converting categorical features to binary variables.
- Numerical transformations (scaling) after splitting .
- Category encoder (one-hot).
- Apply SMOTE (Synthetic Minority Oversampling Technique) on data after splitting.

Experiment & Discussion

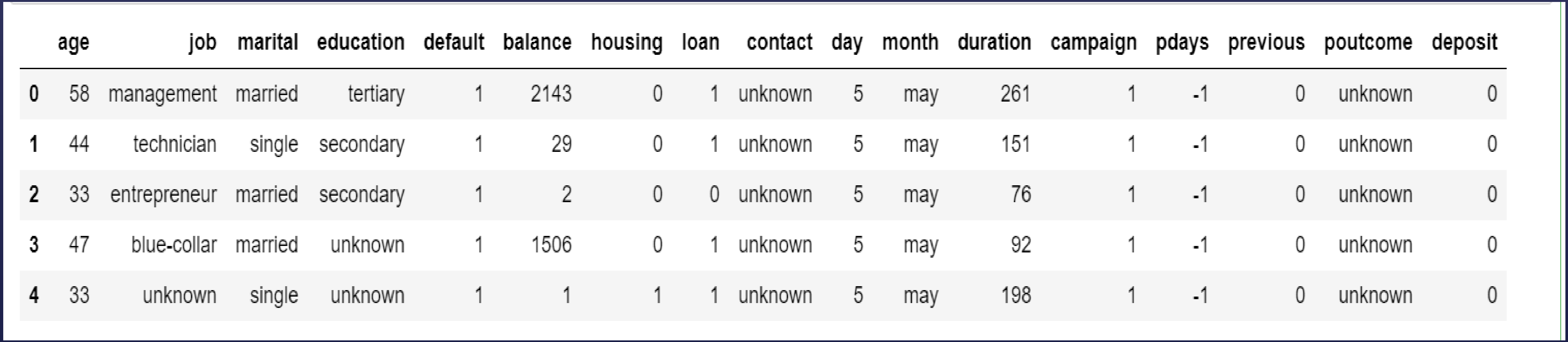
Seven different algorithms are used to solve this problem. Various results have been compared at the end using table. A plot is used to compare ROC curves. Recall is used as one of the performance matrix.

Why Recall ?

As It is a marketing problem a lot of resources are included and it is very important to optimise results to save resources. The target variable is ‘deposit’ which reads yes or no based on success or failure of phone calls. Finding out only those clients which have higher chances of saying yes to subscription of term deposit , will save a lot of manhours and efforts. Predicting as many positives as possible out of actual positives from dataset is the goal here, recall has been chosen as one of the performance matrices along with accuracy and AUC score.

Replacing yes and no from deposit column by 1 and 0 to convert categorical feature to numerical feature for :

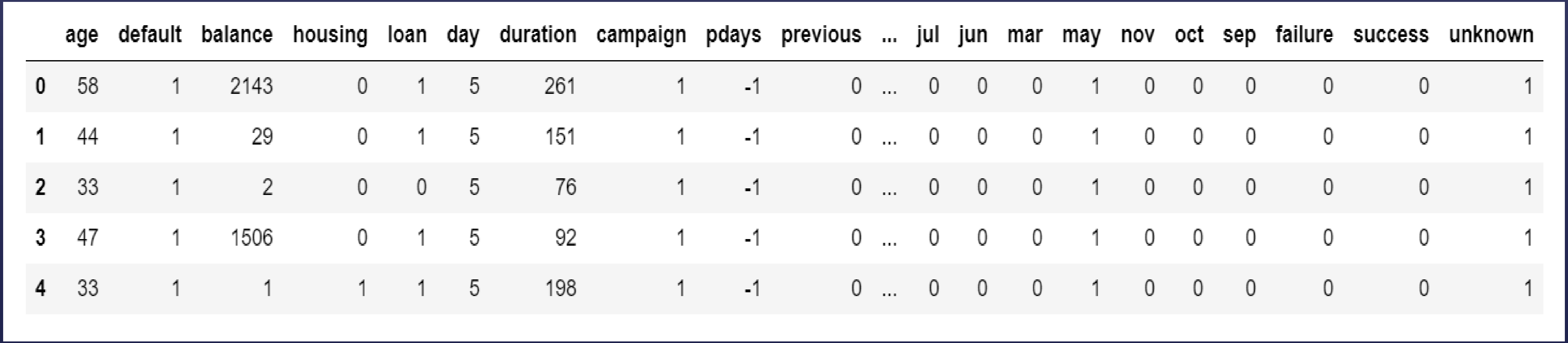
- Deposit
- Loan
- Default
- Housing



	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	58	management	married	tertiary	1	2143	0	1	unknown	5	may	261	1	-1	0	unknown	0
1	44	technician	single	secondary	1	29	0	1	unknown	5	may	151	1	-1	0	unknown	0
2	33	entrepreneur	married	secondary	1	2	0	0	unknown	5	may	76	1	-1	0	unknown	0
3	47	blue-collar	married	unknown	1	1506	0	1	unknown	5	may	92	1	-1	0	unknown	0
4	33	unknown	single	unknown	1	1	1	1	unknown	5	may	198	1	-1	0	unknown	0

One hot encoding for marital feature to convert categorical feature to numerical feature so we Dropped the original column and Dropped one of the resultant columns for :

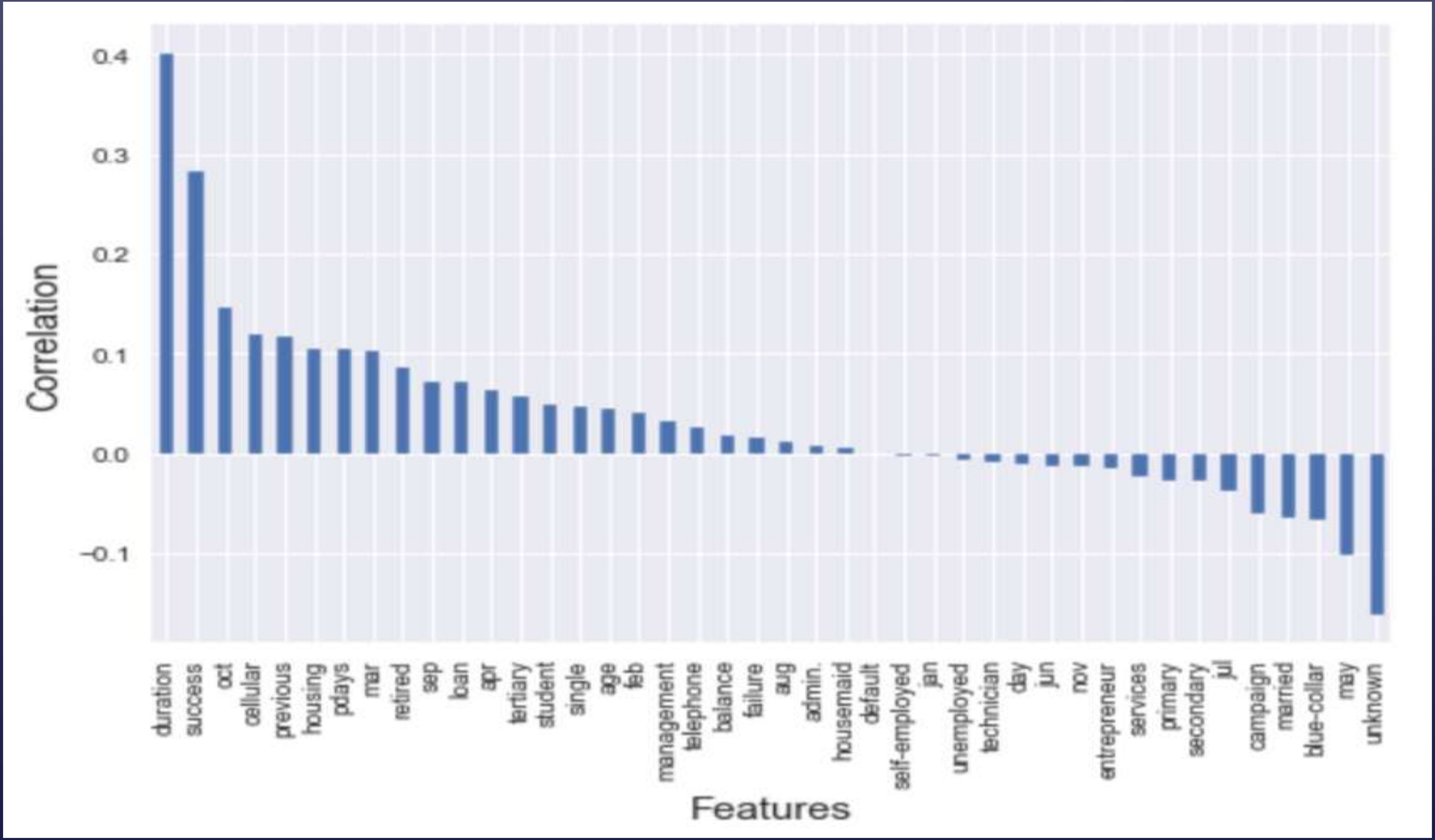
- Marital
- Education
- Job
- Contact
- Month
- Poutcome



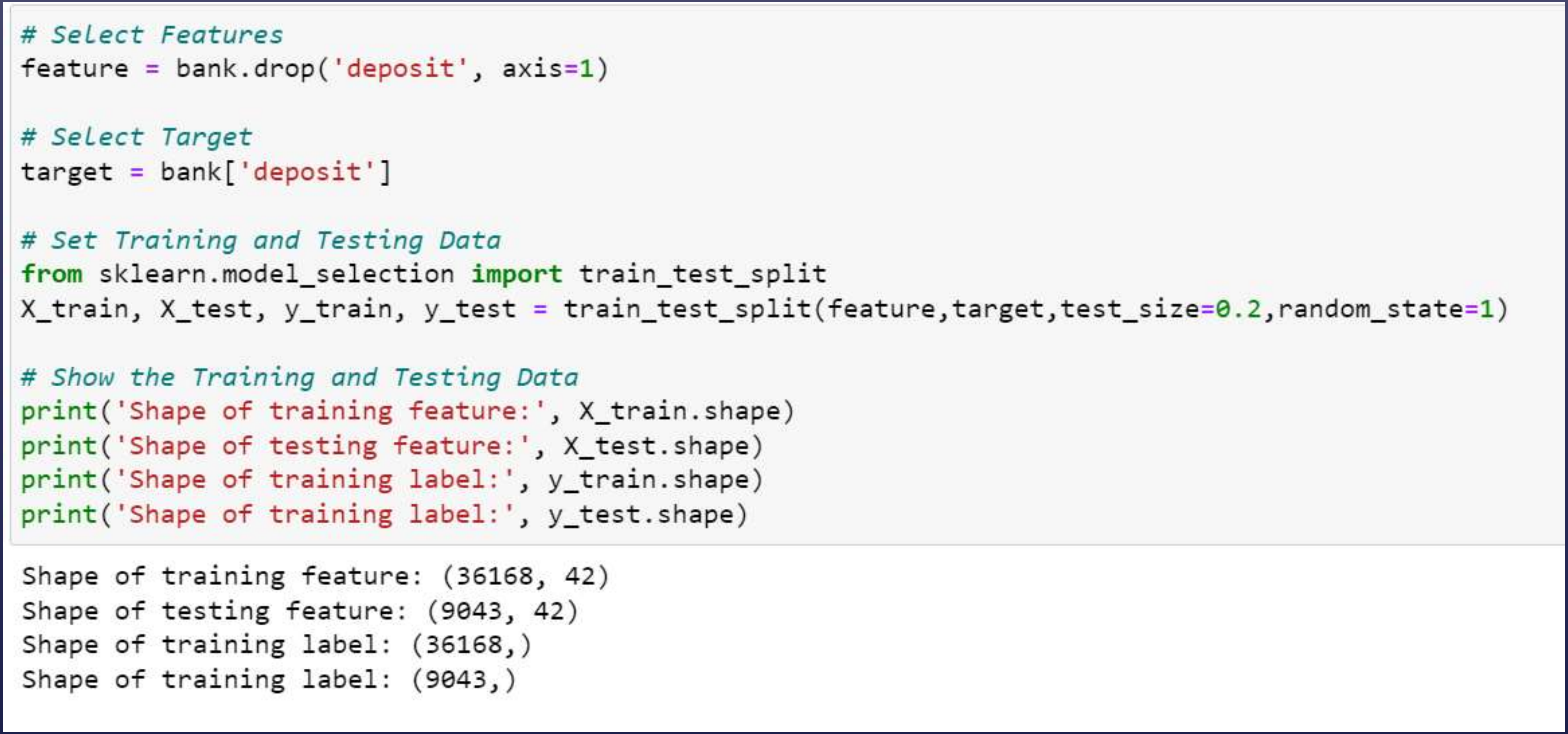
	age	default	balance	housing	loan	day	duration	campaign	pdays	previous	...	jul	jun	mar	may	nov	oct	sep	failure	success	unknown
0	58	1	2143	0	1	5	261	1	-1	0	...	0	0	0	1	0	0	0	0	0	1
1	44	1	29	0	1	5	151	1	-1	0	...	0	0	0	1	0	0	0	0	0	1
2	33	1	2	0	0	5	76	1	-1	0	...	0	0	0	1	0	0	0	0	0	1
3	47	1	1506	0	1	5	92	1	-1	0	...	0	0	0	1	0	0	0	0	0	1
4	33	1	1	1	1	5	198	1	-1	0	...	0	0	0	1	0	0	0	0	0	1

All Features are converted to numerical

Correlation with Class variable 'Deposit'



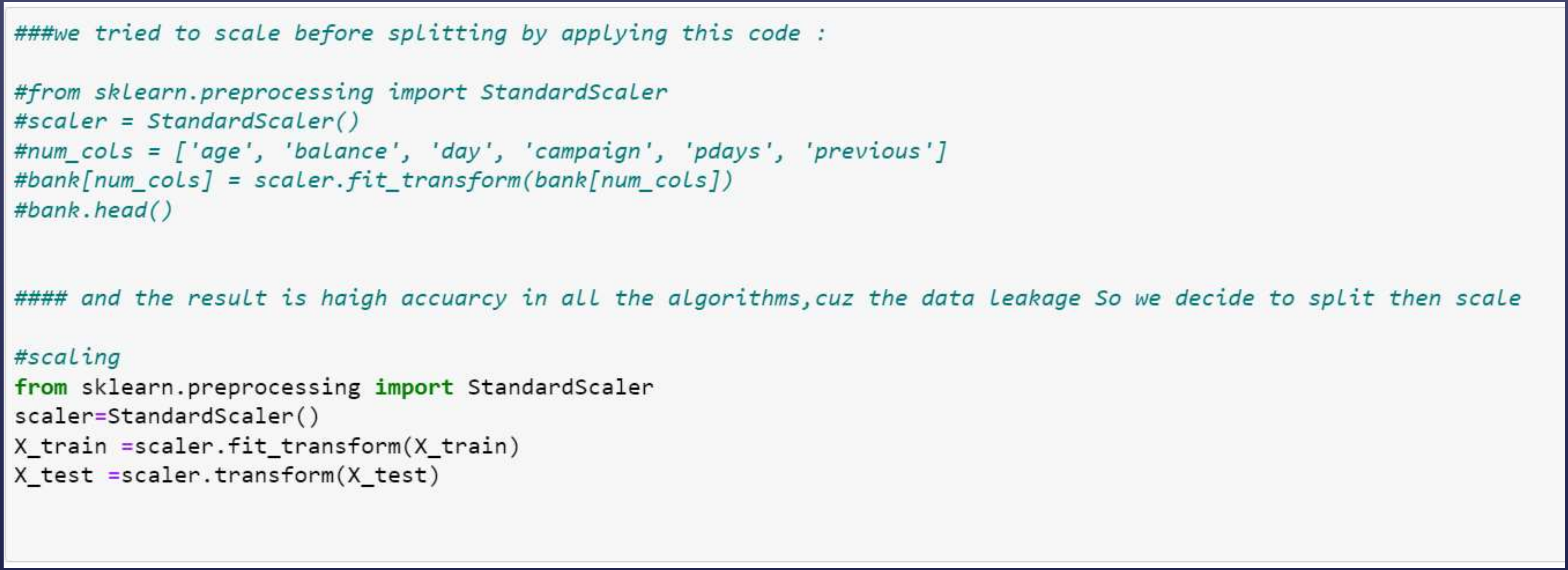
Split Dataset for Training and Testing



```python
# Select Features
feature = bank.drop('deposit', axis=1)

# Select Target
target = bank['deposit']

# Set Training and Testing Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(feature,target,test_size=0.2,random_state=1)

# Show the Training and Testing Data
print('Shape of training feature:', X_train.shape)
print('Shape of testing feature:', X_test.shape)
print('Shape of training label:', y_train.shape)
print('Shape of training label:', y_test.shape)
```

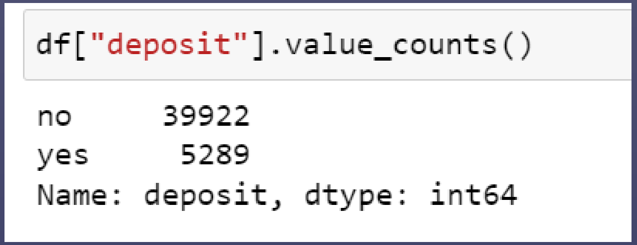
```
Shape of training feature: (36168, 42)
Shape of testing feature: (9043, 42)
Shape of training label: (36168,)
Shape of training label: (9043,)
```

Scale Numeric Data



```
###we tried to scale before splitting by applying this code :

#from sklearn.preprocessing import StandardScaler
#scaler = StandardScaler()
#num_cols = ['age', 'balance', 'day', 'campaign', 'pdays', 'previous']
#bank[num_cols] = scaler.fit_transform(bank[num_cols])
#bank.head()


#### and the result is haigh accuarcy in all the algorithms,cuz the data leakage So we decide to split then scale

#scaling
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train =scaler.fit_transform(X_train)
X_test =scaler.transform(X_test)
```

Class Distribution

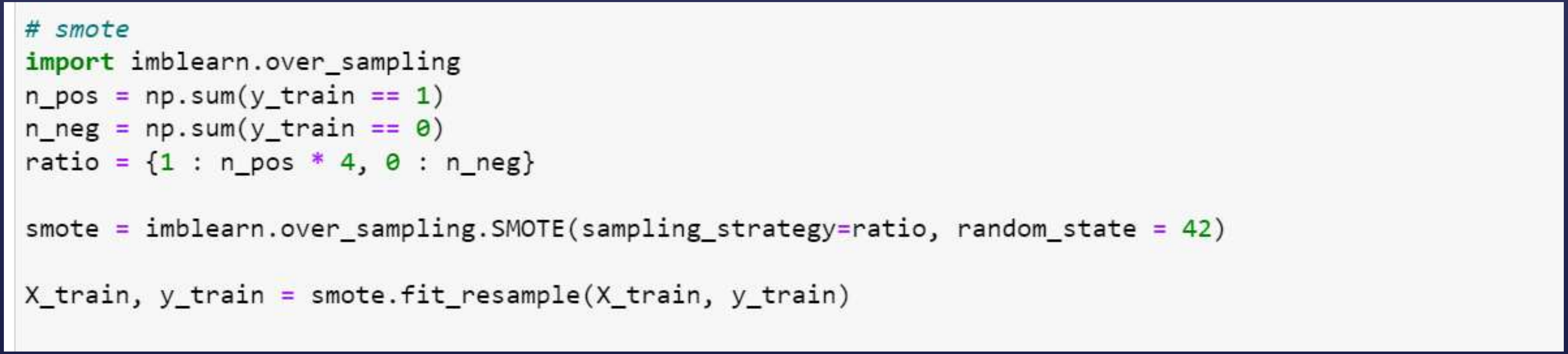


```
df["deposit"].value_counts()
```

```
no     39922
yes     5289
Name: deposit, dtype: int64
```

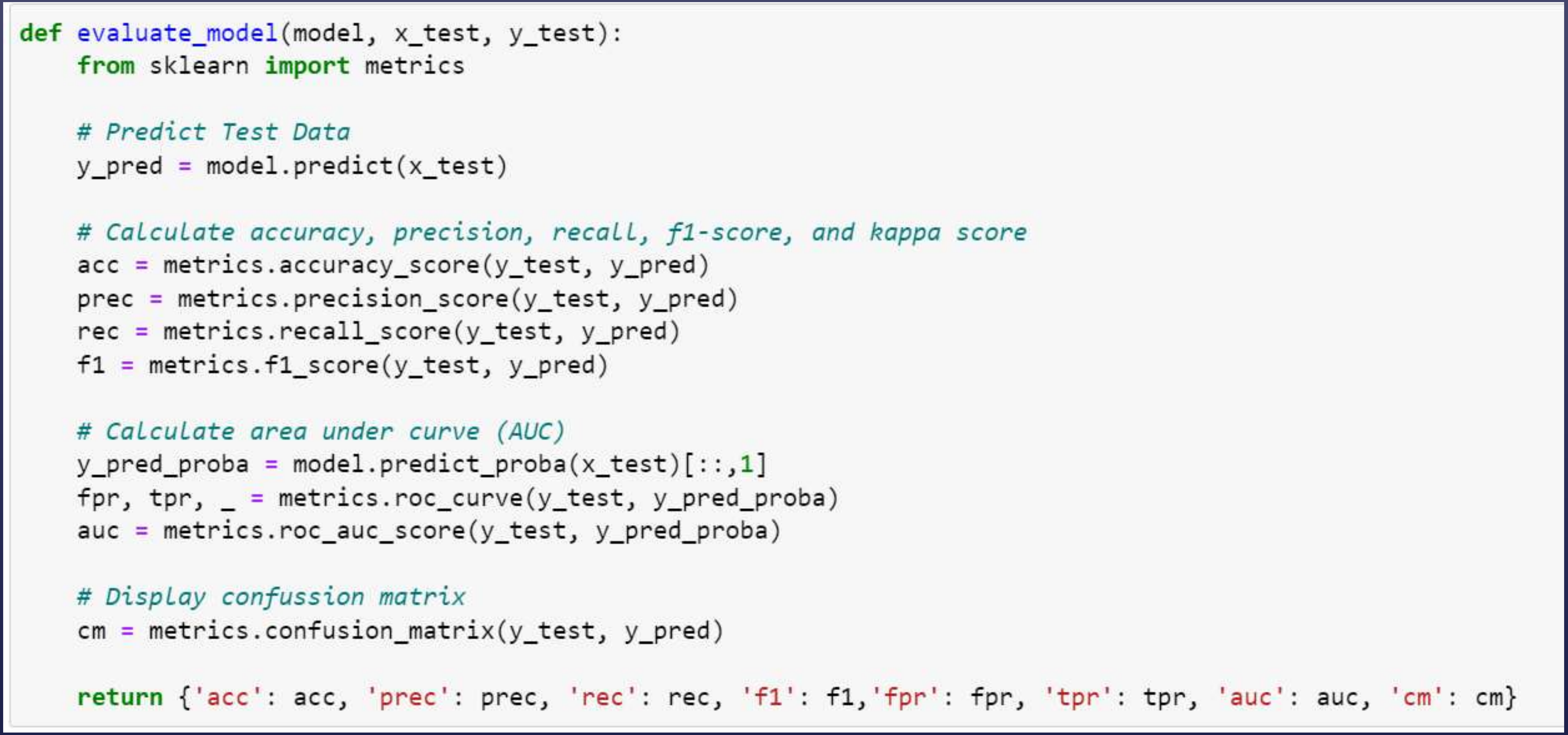
Handling The Class Imbalance : SMOTE

As we can see our class distribution is more or less similar, SO our data is imbalance because it is %88 no and %12 yes SO we need to deal with imbalance by apply SMOTE



```
# smote
import imblearn.over_sampling
n_pos = np.sum(y_train == 1)
n_neg = np.sum(y_train == 0)
ratio = {1 : n_pos * 4, 0 : n_neg}

smote = imblearn.over_sampling.SMOTE(sampling_strategy=ratio, random_state = 42)

X_train, y_train = smote.fit_resample(X_train, y_train)
```

Build the Data Model



```python
def evaluate_model(model, x_test, y_test):
    from sklearn import metrics

    # Predict Test Data
    y_pred = model.predict(x_test)

    # Calculate accuracy, precision, recall, f1-score, and kappa score
    acc = metrics.accuracy_score(y_test, y_pred)
    prec = metrics.precision_score(y_test, y_pred)
    rec = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)

    # Calculate area under curve (AUC)
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)

    # Display confussion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)

    return {'acc': acc, 'prec': prec, 'rec': rec, 'f1': f1,'fpr': fpr, 'tpr': tpr, 'auc': auc, 'cm': cm}
```

- **Models**

- we build 4 models using different algorithm Decision Tree, Random Forest, Naive Bayes, and K-Nearest Neighbours were used before settling on random forest as the model with strongest cross-validation performance.

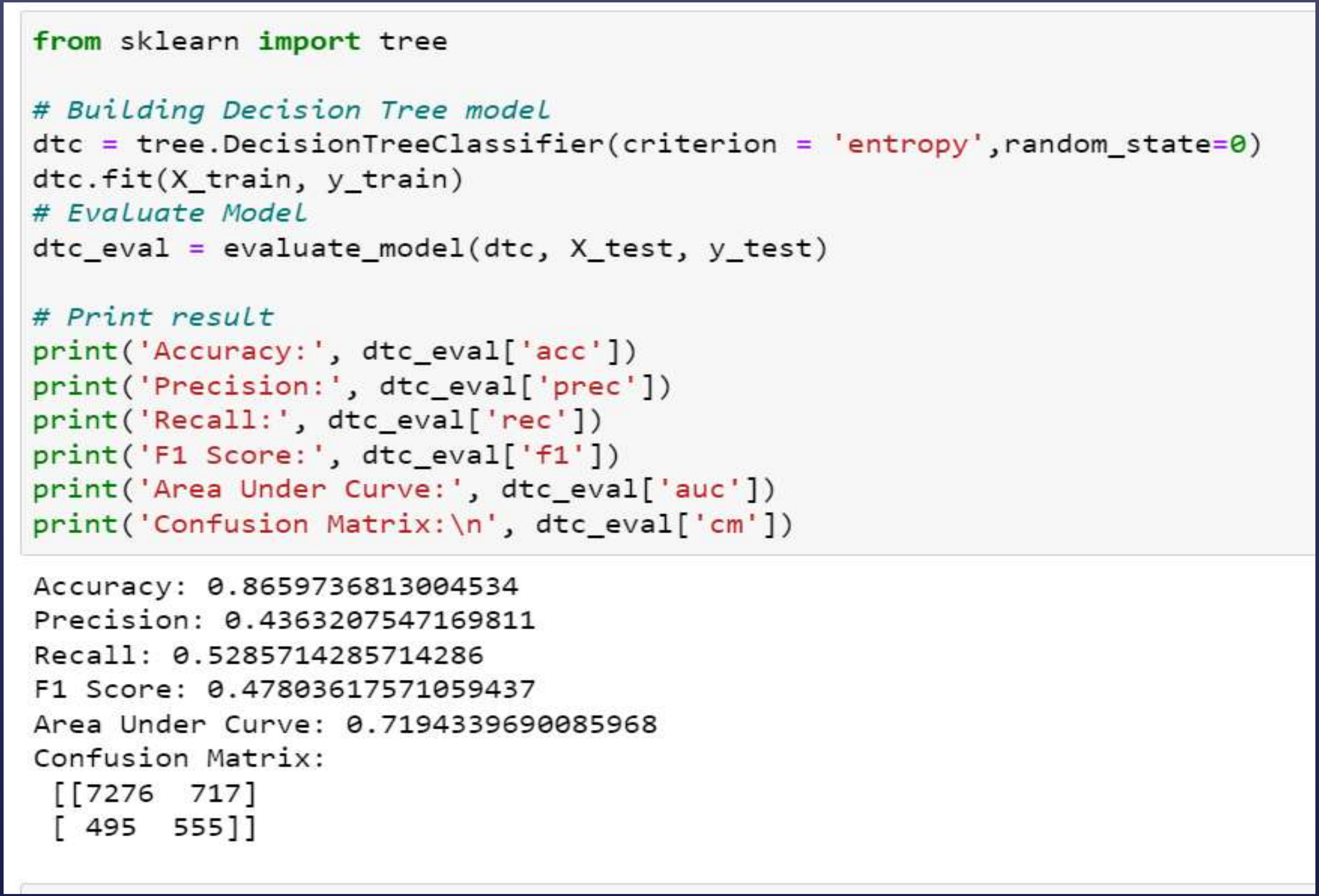
- Random forest feature importance ranking was used directly to guide the choice and order of variables to be included as the model underwent refinement.

Hyperparameter Tuning

Selecting the right hyperparameter and it is probable range is a crucial task. Selecting a wider range may cause longer execution time while selecting a narrow range may result in poor tuning of hyperparameters. So enough number of parameters are chosen with enough range to avoid both problems. In KNN, K is the hyperparameter which indicates the number of nearest neighbours used by algorithm to do the majority vote and predict result. Naïve Bayes internally uses alpha, which is basically constant of Laplace smoothing. In Logistic Regression, λ is the hyperparameter which controls the amount of regularisation in optimisation. Sklearn implements it as c for uniformity. In addition to that random forest uses some extra parameters

Decision Tree

Decision tree is a tree shaped diagram used to determine a course of action. Each branch of the tree represents a possible decision, occurrence or reaction.

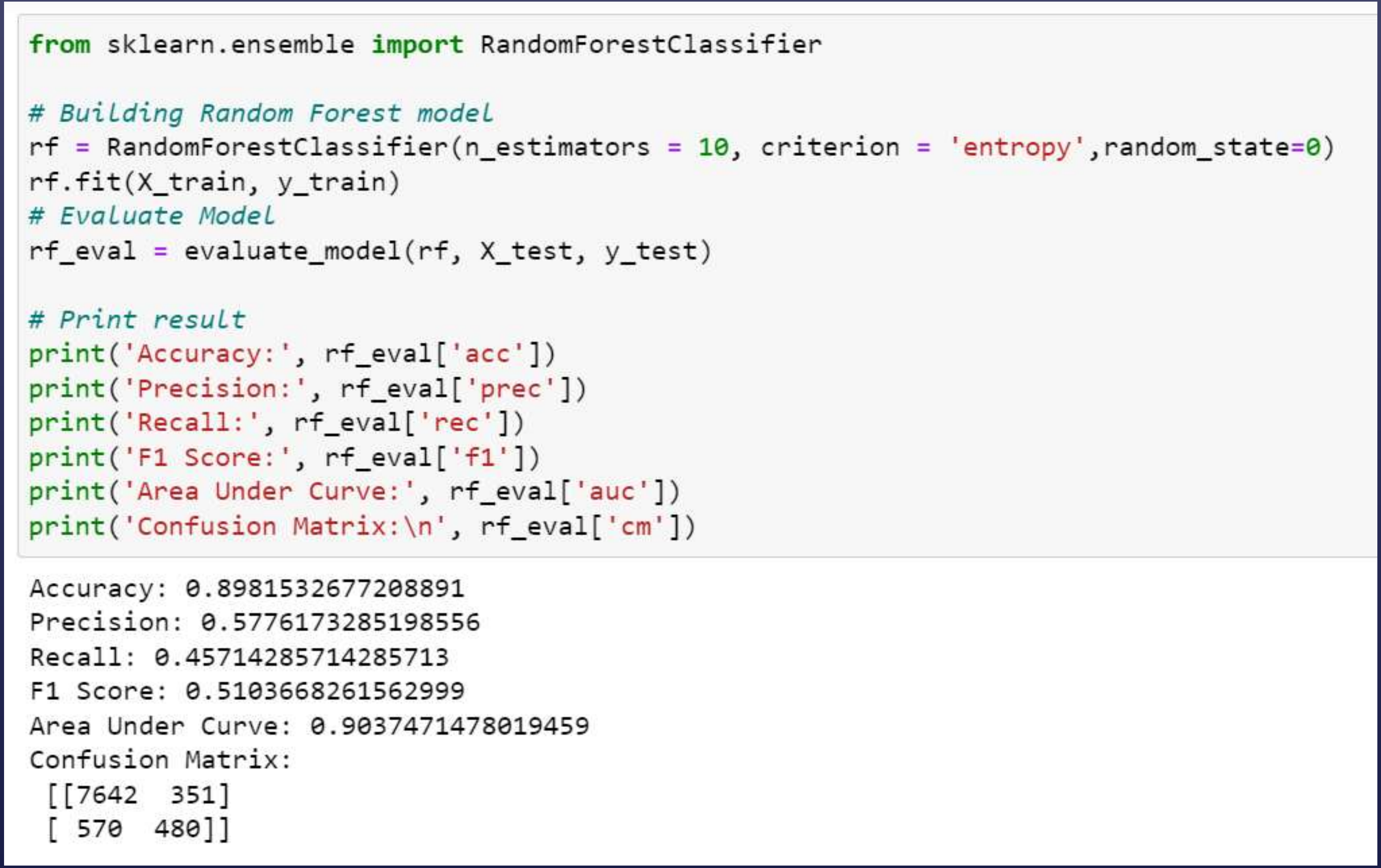


```
from sklearn import tree

# Building Decision Tree model
dtc = tree.DecisionTreeClassifier(criterion = 'entropy',random_state=0)
dtc.fit(X_train, y_train)
# Evaluate Model
dtc_eval = evaluate_model(dtc, X_test, y_test)

# Print result
print('Accuracy:', dtc_eval['acc'])
print('Precision:', dtc_eval['prec'])
print('Recall:', dtc_eval['rec'])
print('F1 Score:', dtc_eval['f1'])
print('Area Under Curve:', dtc_eval['auc'])
print('Confusion Matrix:\n', dtc_eval['cm'])
```

```
Accuracy: 0.8659736813004534
Precision: 0.4363207547169811
Recall: 0.5285714285714286
F1 Score: 0.47803617571059437
Area Under Curve: 0.7194339690085968
Confusion Matrix:
 [[7276  717]
 [ 495  555]]
```

Random Forest

Random forest or Random Decision Forest is a method that operates by constructing multiple decision trees during training phases. The decision of the majority of the trees is chosen as final decision.



```
from sklearn.ensemble import RandomForestClassifier

# Building Random Forest model
rf = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',random_state=0)
rf.fit(X_train, y_train)
# Evaluate Model
rf_eval = evaluate_model(rf, X_test, y_test)

# Print result
print('Accuracy:', rf_eval['acc'])
print('Precision:', rf_eval['prec'])
print('Recall:', rf_eval['rec'])
print('F1 Score:', rf_eval['f1'])
print('Area Under Curve:', rf_eval['auc'])
print('Confusion Matrix:\n', rf_eval['cm'])
```

```
Accuracy: 0.8981532677208891
Precision: 0.5776173285198556
Recall: 0.45714285714285713
F1 Score: 0.510368261562999
Area Under Curve: 0.9037471478019459
Confusion Matrix:
 [[7642  351]
 [ 570  480]]
```

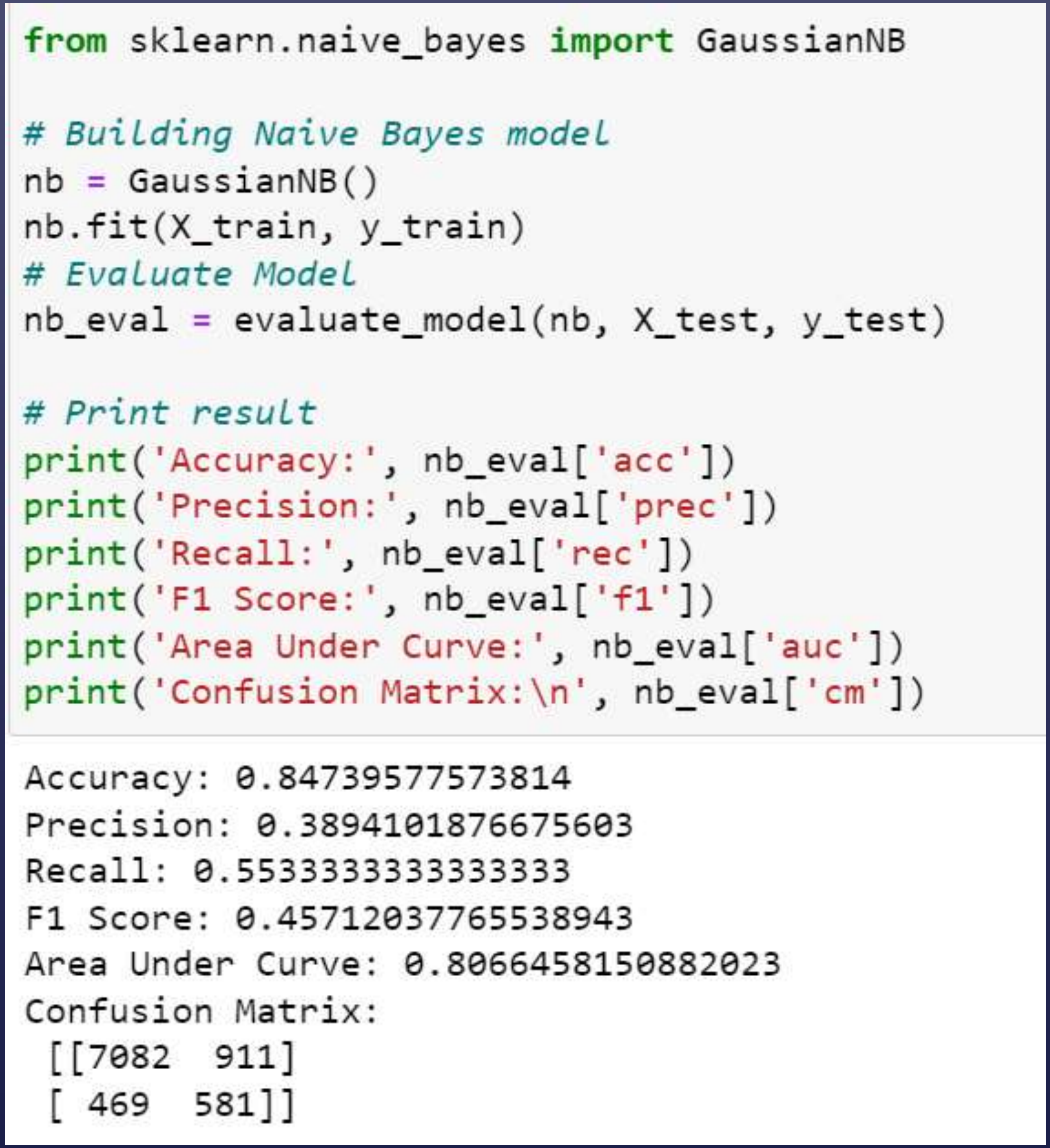
Naive Bayes

Naive Bayes is a simple technique for constructing classifiers:

models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle:

all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

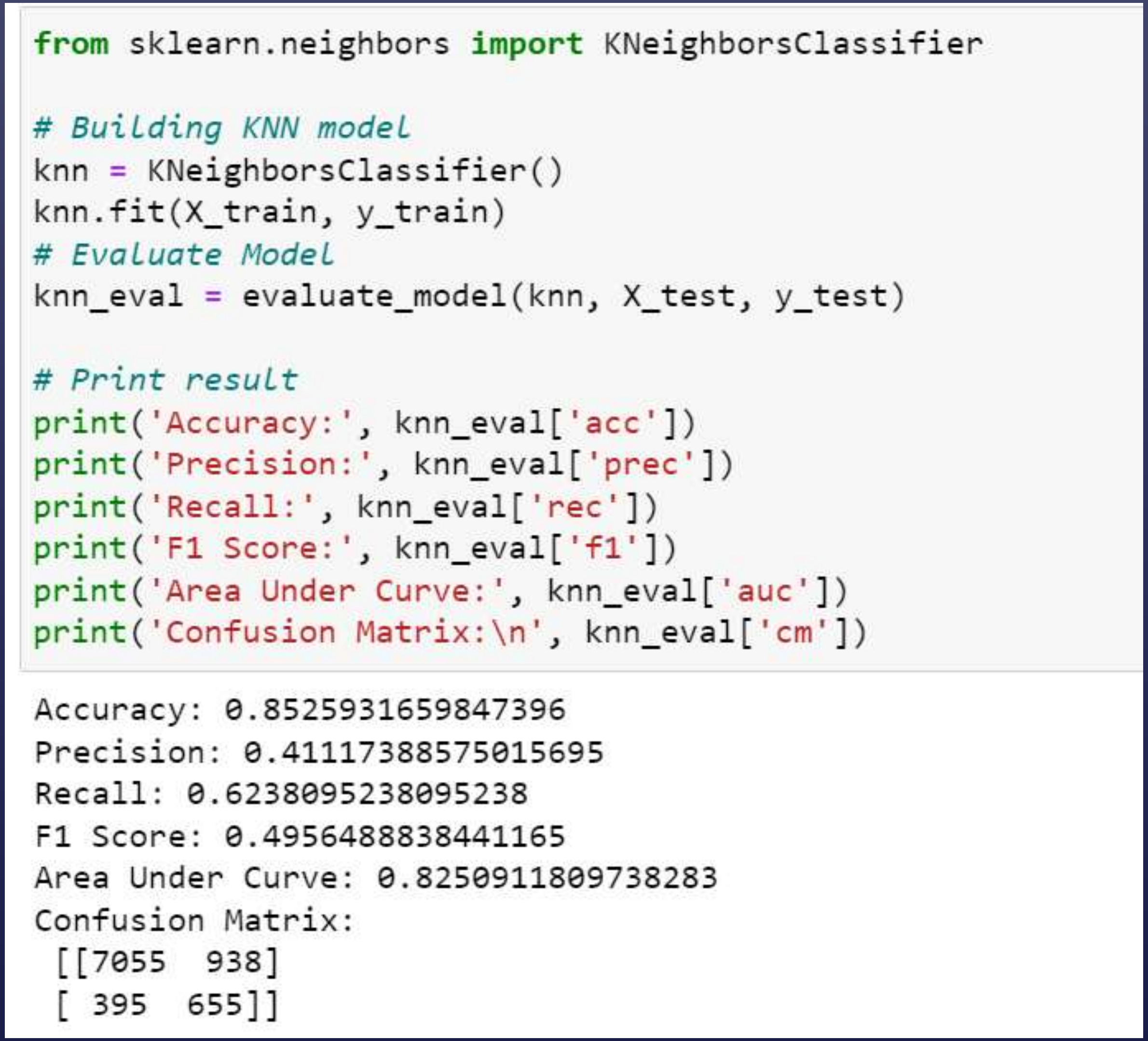


```
from sklearn.naive_bayes import GaussianNB

# Building Naive Bayes model
nb = GaussianNB()
nb.fit(X_train, y_train)
# Evaluate Model
nb_eval = evaluate_model(nb, X_test, y_test)

# Print result
print('Accuracy:', nb_eval['acc'])
print('Precision:', nb_eval['prec'])
print('Recall:', nb_eval['rec'])
print('F1 Score:', nb_eval['f1'])
print('Area Under Curve:', nb_eval['auc'])
print('Confusion Matrix:\n', nb_eval['cm'])
```

```
Accuracy: 0.847395775738l4
Precision: 0.389410187667560З
Recall: 0.5533333333333333
F1 Score: 0.4571203776538943
Area Under Curve: 0.806645815088202З
Confusion Matrix:
 [[7082  911]
 [ 469  581]]
```

K-Nearest Neighbors

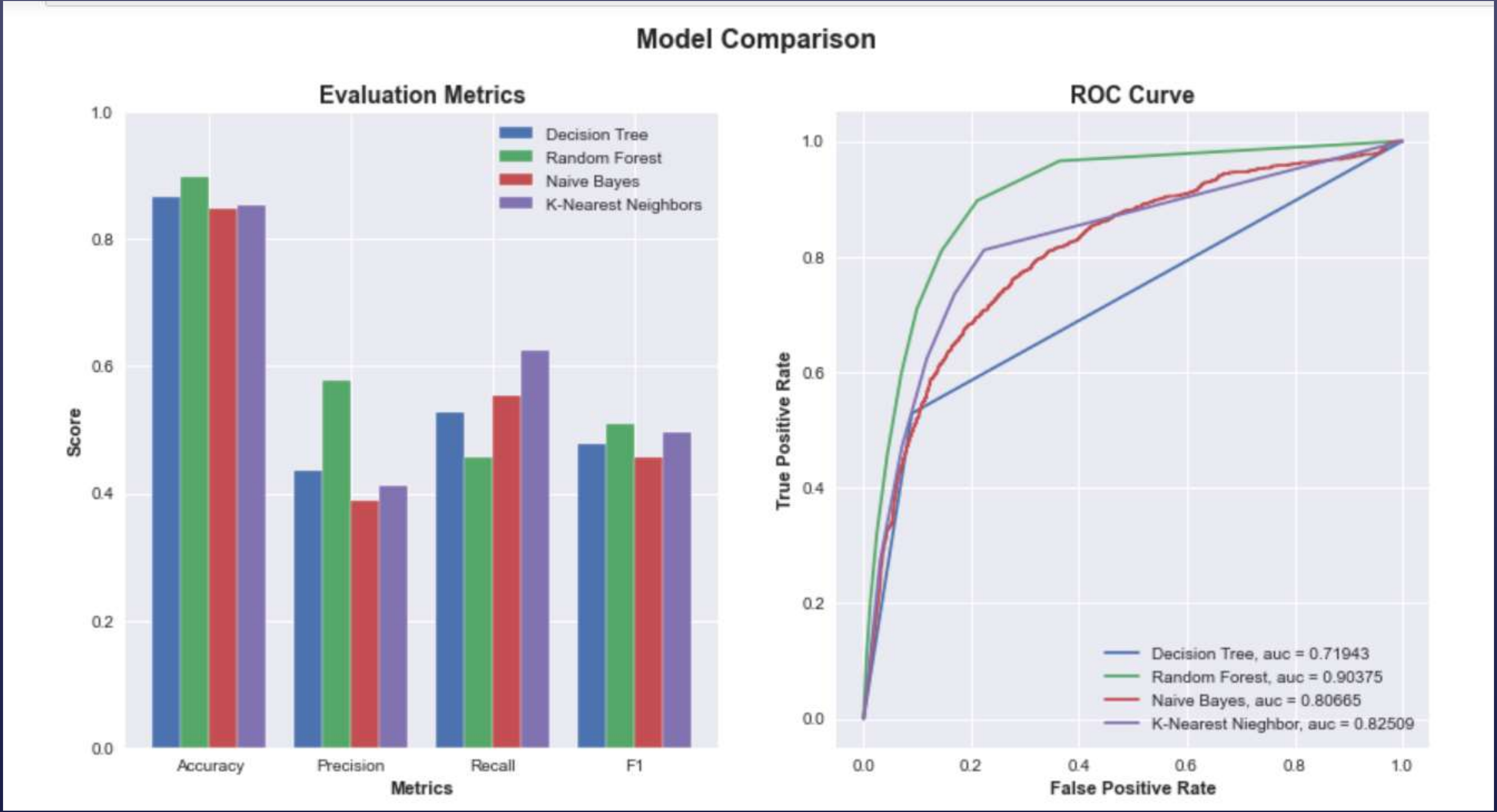
K-Nearest Neighbors (KNN) classify new data by finding k-number of closest neighbor from the training data and then decide the class based on the majority of it's neighbors.



```python
from sklearn.neighbors import KNeighborsClassifier

# Building KNN model
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
# Evaluate Model
knn_eval = evaluate_model(knn, X_test, y_test)

# Print result
print('Accuracy:', knn_eval['acc'])
print('Precision:', knn_eval['prec'])
print('Recall:', knn_eval['rec'])
print('F1 Score:', knn_eval['f1'])
print('Area Under Curve:', knn_eval['auc'])
print('Confusion Matrix:\n', knn_eval['cm'])
```

```
Accuracy: 0.8525931659847396
Precision: 0.41117388575015695
Recall: 0.6238095238095238
F1 Score: 0.4956488838441165
Area Under Curve: 0.8250911809738283
Confusion Matrix:
 [[7055  938]
 [ 395  655]]
```

Model Comparison



Model Evaluation and Selection

The entire training dataset of 45211 records was split into 80/20 train. and the model's assessment is conducted using essential measurements, such as accuracy and recall, F1 and ROC are also used. the result for each method :

Classifier	Accuracy	Precision	recall	F1	ROC
Decision Tree	0.86	0.43	0.53	0.48	0.72
Random Forest	0.90	0.58	0.46	0.51	0.90
Naive Bayes	0.85	0.39	0.55	0.46	0.80
K-Nearest Neighbours	0.85	0.41	0.62	0.50	0.82

- Discussion

- For Naïve Bayes there is no need to do the hyperparameter tuning. The algorithm does it internally. For logistic regression it uses L2 regularization to give the best result. The maximum depth for decision tree is 8. More than that would cause overfitting. Random forest has used 2000 estimators to come to a good result. The first two algorithms have lower results. KNN is distance based algorithm. As the number of features increase it is performance decreases due to curse of dimensionality. Naïve Bayes is mostly used for text classification. It is more efficient with categorical features and for numerical features it requires gaussian distribution. Tree based Algorithms are giving better results that others.

conclusion

For a simple model we can see that our model did decently on classifying the data. But there are still some weakness on our model, especially shown on the recall metric where we only get about 60%. This means that our model are only able to detect 60% of potential customer and miss the other 40%. The best AUC score of 0.92 comes from Random Forest . The results of K nearest neighbours and Naive Bayes are less. In that case natural language processing will give better results. In these times of crisis preserving the relationship with best customers is more crucial than ever. Using these results bank can specifically target clients and gain higher success in their endeavours. Saving a lot of time by not focusing on clients with less probability is yet another advantages of this project.

References

Telkom Digital Talent Incubator - Data Scientist Module 5 (Classification)

Scikit-learn Documentation

The 5 Classification Evaluation metrics every Data Scientist must know

The Python Graph Gallery - Grouped Bar Plot

Thank You