- Propensity model - The model that tries to predict the customer who has an intention or likelihood to buy the specific product.

  Who Will Subscribe A Term Deposit?

  ### Bank Problem Statement:

  Understand why clients are not depositing as frequently as before. In addition, banks also hold better chance to persuade term deposit clients into buying other products such as funds or insurance to further increase their revenues. As a result, the Portuguese bank would like to identify existing clients that have higher chance to subscribe for a term deposit and focus marketing efforts on such clients.

  ### Data Science Problem Statement:

  Predict if the client will subscribe to a term deposit based on the analysis of the marketing campaigns the bank performed.

  ### Understand the Kaggle dataset:

  The data is related to 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 subscribed ('yes') or not ('no') subscribed.

  train.csv with all examples (32950) and 21 inputs including the target feature, ordered by date (from May 2008 to November 2010).

```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# load modeling libraries
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.ensemble import RandomForestClassifier
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
```

Read Data

```
# accessing to the folder where the file is stored
path = 'train.csv'

# Load the dataframe
df = pd.read_csv(path, sep=';')

print('Shape of the data is: ',df.shape)

df.sample(2)
```

```
Shape of the data is:  (45211, 17)
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pr
22541	31	technician	single	secondary	no	354	no	yes	cellular	22	aug	186	2	-1	
31124	24	student	single	secondary	no	23878	no	no	cellular	18	feb	185	1	-1	

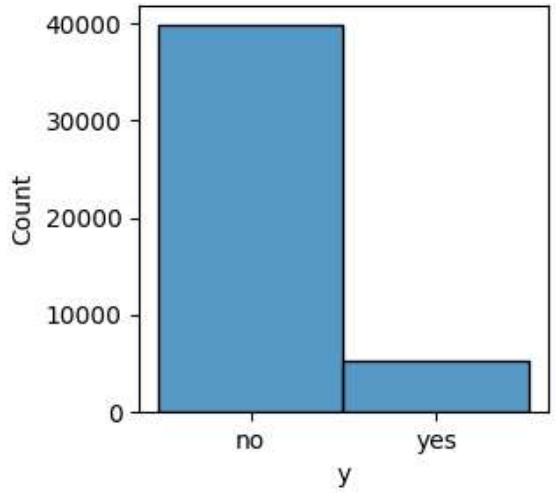
Data Understanding and exploration

```
df.describe()
```

```
                    age        balance          day      duration      campaign         pdays      previous
```

```
fig = plt.figure(figsize = (3,3))
sns.histplot(df.y)
```

```
<Axes: xlabel='y', ylabel='Count'>
```



Understand data with business perspective

What are the most responsible features of the people who respond to the campaign and register for deposits?

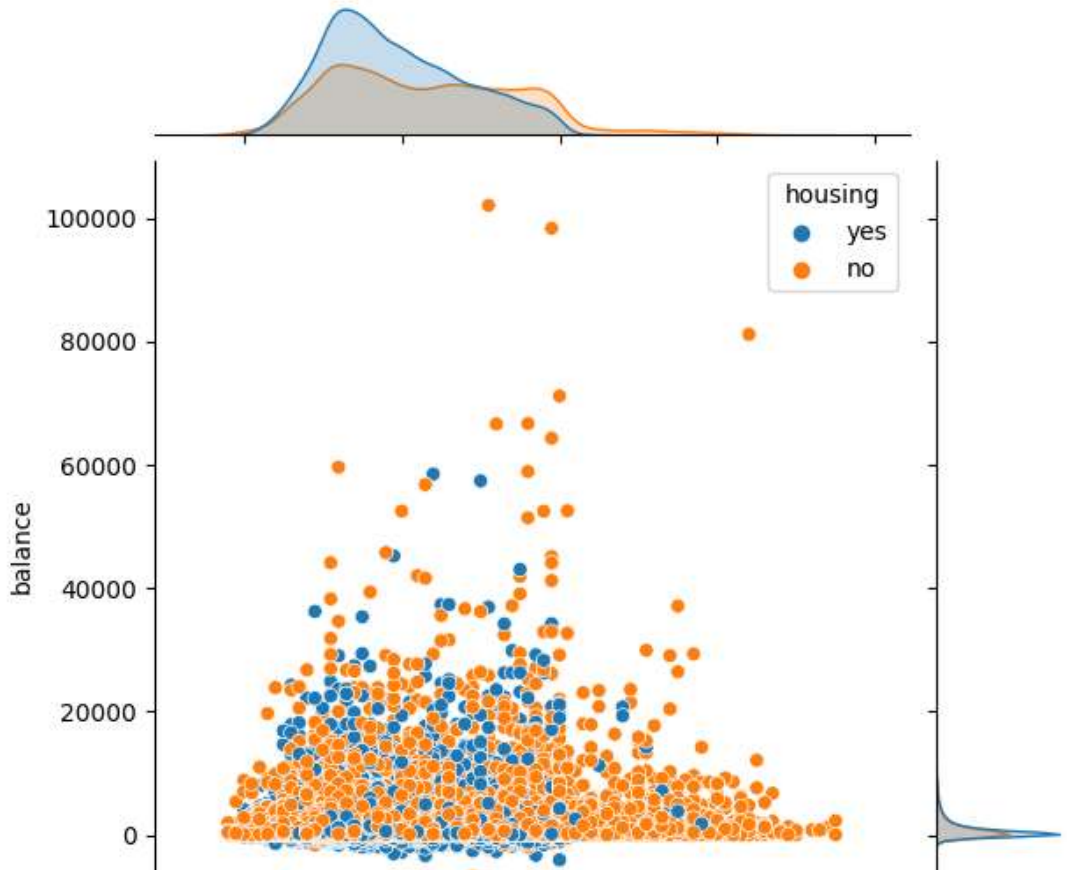
lets a dig a data more to understand it.

```
df["balance"].max(), df["balance"].min()
```

```
(102127, -8019)
```

```
sns.jointplot(x = "age", y = "balance",
              hue = "housing",
              data = df)
```

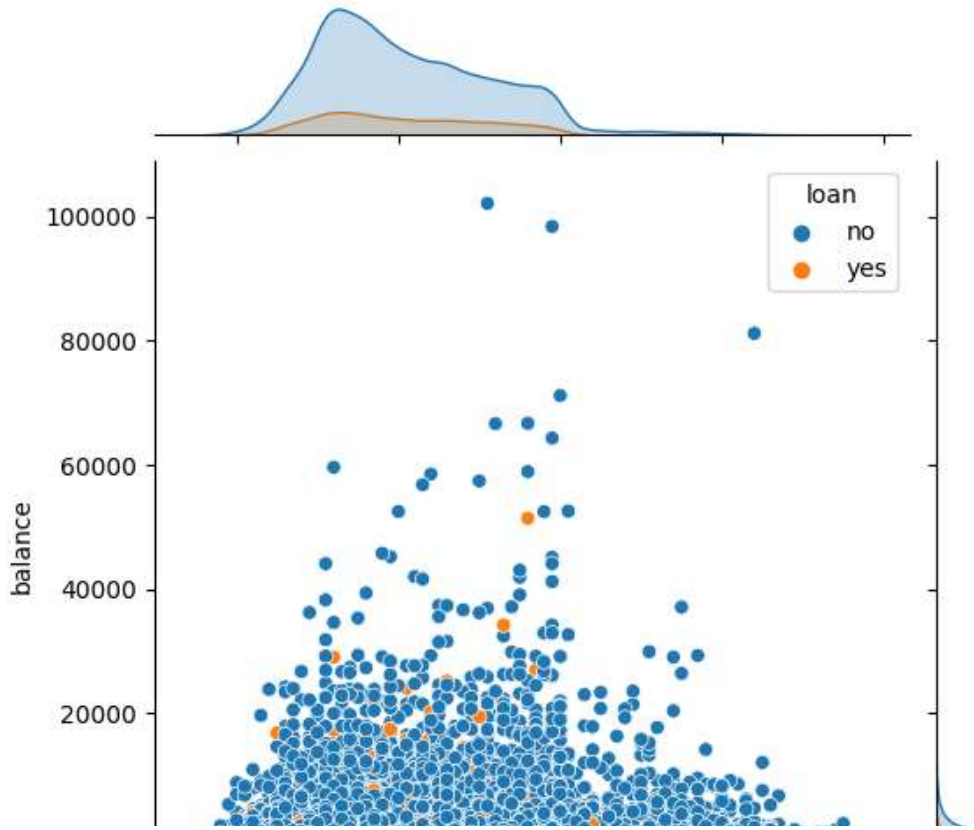
```
<seaborn.axisgrid.JointGrid at 0x7d62462cbc40>
```



Most individuals do not have a loan on their homes, but most of the individuals who have a loan on their homes are concentrated between the ages of 25 to 60 years, and this age group is the main target, so we may face some problems in choosing the characteristics that distinguish the target people.

```
sns.jointplot(x = "age", y = "balance",
              hue = "loan",
              data = df)
```

```
<seaborn.axisgrid.JointGrid at 0x7d62442866e0>
```



Loan feature does not affect the results directly.

Similarly you can dig other variables individually or combination wise to get more clear idea about what are the people are more inclined to subscribe the term deposite.

Ous basic conlustion so far;

- Their ages range from 22 years old to 70 years old.
- personal loan, this is not effective.

Data Cleaning and processing

```
# check missing values
print(df.isna().sum())
```

```
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
```

```
# Checking duplicate values
print(df.duplicated().value_counts())
```

```
False    45211
dtype: int64
```

```
# Replace method: Mode value
# Replace method for "unknown" variable in ["job", "education", "contact"].
df["job"].replace(["unknown"],df["job"].mode(),inplace = True)
df["education"].replace(["unknown"],df["education"].mode(),inplace = True)
df["contact"].replace(["unknown"],df["contact"].mode(),inplace = True)
```

```
# remove irrelevant columns
data = df.drop(['month','day'],axis=1)
```

```
# label encoding
le = LabelEncoder()
data['job'] = le.fit_transform(data['job'])
data['marital'] = le.fit_transform(data['marital'])
data['education'] = le.fit_transform(data['education'])
data['default'] = le.fit_transform(data['default'])
```

```
data['housing'] = le.fit_transform(data['housing'])
data['loan'] = le.fit_transform(data['loan'])
data['contact'] = le.fit_transform(data['contact'])
data['poutcome'] = le.fit_transform(data['poutcome'])
data['y'] = le.fit_transform(data['y'])


# standardize features
features = data.drop("y", axis = 1)
target = data["y"]
features_num = features.columns
scaler = StandardScaler()
features = pd.DataFrame(scaler.fit_transform(features))
features.columns = features_num

features.head(2)
```

	age	job	marital	education	default	balance	housing	loan	contact	duration	campaign	pdays	p
0	1.606965	-0.085217	-0.275762	1.314507	-0.13549	0.256419	0.893915	-0.436803	-0.262091	0.011016	-0.569351	-0.411453	-
1	0.288529	1.458223	1.368372	-0.218740	-0.13549	-0.437895	0.893915	-0.436803	-0.262091	-0.416127	-0.569351	-0.411453	-

```
# export clean and processed data
final_data = pd.concat([features, target], axis=1,)
final_data.to_csv("final_version.csv")
```

Train test split

```
# Training, Test, & Split
y = final_data["y"]
X = final_data.drop("y",axis = 1)
X_train , X_test , y_train , y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify=y)
```

Build baseline model

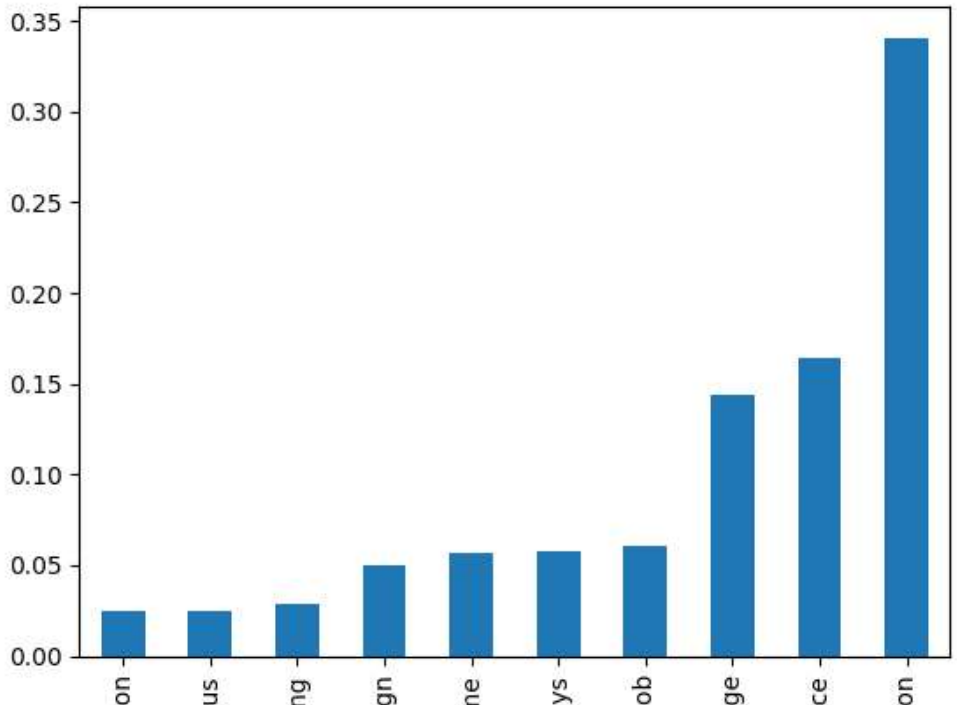
```
# Random forest Model 1: Baseline
```

```
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)


# Evaluate model
print(confusion_matrix(y_pred, y_test))
print(classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
[[7741  688]
 [ 244  370]]
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      7985
           1       0.60      0.35      0.44      1058

    accuracy                           0.90      9043
   macro avg       0.76      0.66      0.69      9043
weighted avg       0.88      0.90      0.88      9043

Accuracy: 0.8969368572376424
```

Feature Importance

```
# selecting the data
rfc = RandomForestClassifier(random_state=42)
# fitting the data
rfc.fit(X_train, y_train)
# predicting the data
y_pred = rfc.predict(X_test)
# feature importances
rfc_importances = pd.Series(rfc.feature_importances_, index=X.columns).sort_values().tail(10)
# plotting bar chart according to feature importance
rfc_importances.plot(kind='bar')
fig = plt.figure(figsize = (5,5))
plt.show()
```




```
imp_features = list(rfc_importances.index)
imp_features
```

```
['education',
 'previous',
 'housing',
 'campaign',
 'poutcome',
 'pdays',
 'job',
 'age',
 'balance',
 'duration']
```

```
# data with importance feactures
X_top = final_data[imp_features]

# splitting the data
x_train,x_val,y_train,y_val = train_test_split(X_top,y, test_size=0.2, random_state=42, stratify=y)
```

```
# Random forest Model 2: with top features
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)
# Evaluate model
print(confusion_matrix(y_pred, y_test))
print(classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
[[7737  674]
 [ 248  384]]
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      7985
           1       0.61      0.36      0.45      1058

    accuracy                           0.90      9043
   macro avg       0.76      0.67      0.70      9043
weighted avg       0.88      0.90      0.89      9043

Accuracy: 0.8980426849496849
```

The Feature Selection techniques can differ from problem to problem. In those cases, feel free to try out other methods like PCA, SelectKBest(), SelectPercentile(), tSNE etc.

In our case random forest top featurs have not helped us, so we can ignore this part for further model tasks

Hyper parametertuning

```
# splitting the data
x_train,x_val,y_train,y_val = train_test_split(X,y, test_size=0.2, random_state=42, stratify=y)

# selecting the classifier
rfc = RandomForestClassifier()
# selecting the parameter
param_grid = {
'max_depth' : [4,5,6,7,8],
'criterion' :['gini', 'entropy']}
# using grid search with respective parameters
grid_search_model = GridSearchCV(rfc, param_grid=param_grid)
# fitting the model
grid_search_model.fit(x_train, y_train)
```

```
grid_search_model.fit(x_train, y_train)
# printing the best parameters
print('Best Parameters are:',grid_search_model.best_params_)
```

```
Best Parameters are: {'criterion': 'gini', 'max_depth': 8}
```

Best Parameters are: {'criterion': 'gini', 'max_depth': 8}

Fianl Mode„l„

```
# # Random forest Model 2: hyperparameter tuning
rfc = RandomForestClassifier(criterion= 'entropy', max_depth= 2)
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)
# Evaluate model
print(confusion_matrix(y_pred, y_test))
print(classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
[[7985 1058]
 [   0    0]]
              precision    recall  f1-score   support

           0       0.88      1.00      0.94      7985
           1       0.00      0.00      0.00      1058

    accuracy                           0.88      9043
   macro avg       0.44      0.50      0.47      9043
weighted avg       0.78      0.88      0.83      9043

Accuracy: 0.8830034280659074
```