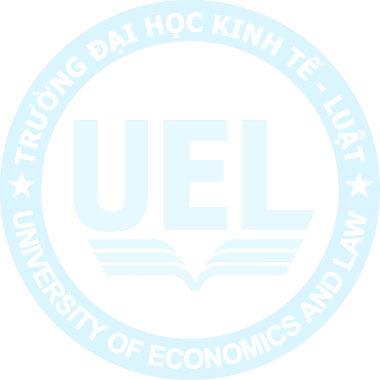
**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**

**UNIVERSITY OF ECONOMICS AND LAW**



**FINAL REPORT**

**Subject: MACHINE LEARNING**

LECTURER: Phan Huy Tam

Submitted by:

Tran Nhat Thong - K194141752

Ho Chi Minh City, Jun 23th, 2022

**Contents**

[**1.** **About dataset.** 1](#_Toc106920336)

[**2.** **Exploratory Data Analysis (EDA)** 2](#_Toc106920337)

[**Describe data** 2](#_Toc106920338)

[**Explore categorical data** 5](#_Toc106920339)

[**Explore continuous data** 11](#_Toc106920340)

[**4.** **Build model and evaluating** 18](#_Toc106920341)

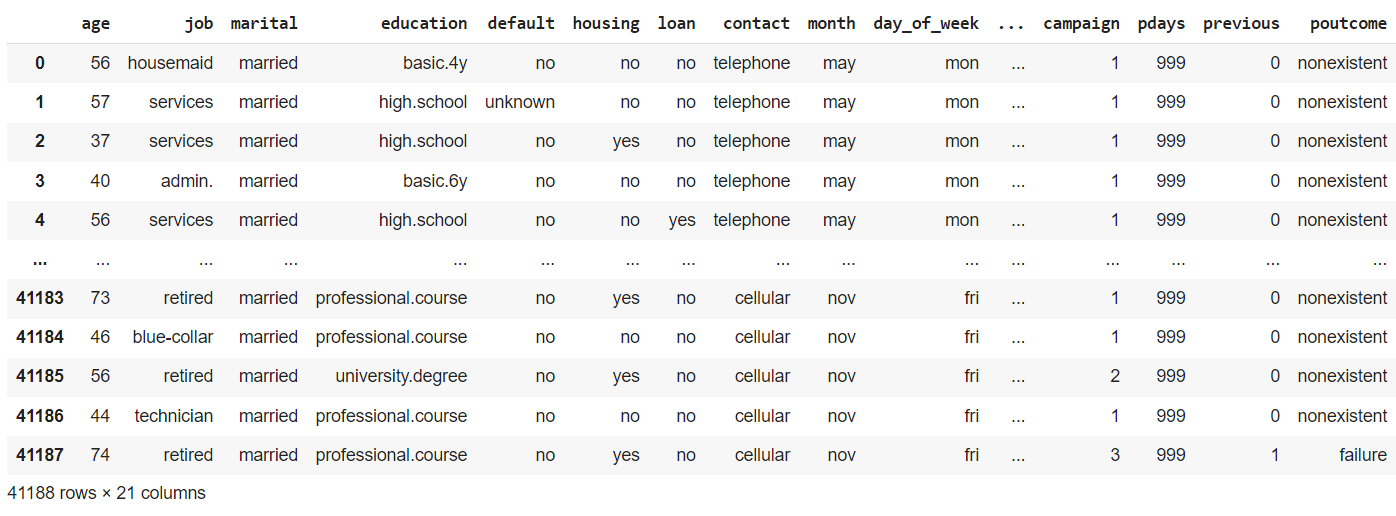
1. **About dataset.**

* The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (y).
* Data Set Information:  
  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.
* **Attribute Information:**
  + **Bank client data:**
    - Age (numeric)
    - Job : type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
    - Marital : marital status (categorical: 'divorced', 'married', 'single', 'unknown' ; note: 'divorced' means divorced or widowed)
    - Education: (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')
    - Default: has credit in default? (categorical: 'no', 'yes', 'unknown')
    - Housing: has housing loan? (categorical: 'no', 'yes', 'unknown')
    - Loan: has personal loan? (categorical: 'no', 'yes', 'unknown')
  + **Related with the last contact of the current campaign:**
    - Contact: contact communication type (categorical: 'cellular','telephone')
    - Month: last contact month of year (categorical: 'jan', 'feb', 'mar',…, 'nov', 'dec')
    - Day\_of\_week: last contact day of the week (categorical:'mon','tue','wed','thu','fri')
    - Campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
    - 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 predictive model
  + **Other attributes:**
    - 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)
    - Previous: number of contacts performed before this campaign and for this client (numeric)
    - Poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
  + **Social and economic context attributes**
    - Emp.var.rate: employment variation rate - quarterly indicator (numeric)
    - Cons.price.idx: consumer price index - monthly indicator (numeric)
    - Cons.conf.idx: consumer confidence index - monthly indicator (numeric)
    - Euribor3m: euribor 3 month rate - daily indicator (numeric)
    - Nr.employed: number of employees - quarterly indicator (numeric)
  + **Output variable (desired target):**
    - y - has the client subscribed a term deposit? (binary: 'yes', 'no')
* **Source:** [http://archive.ics.uci.edu/ml/datasets/Bank+Marketing#](http://archive.ics.uci.edu/ml/datasets/Bank+Marketing%23)

1. **Exploratory Data Analysis (EDA)**

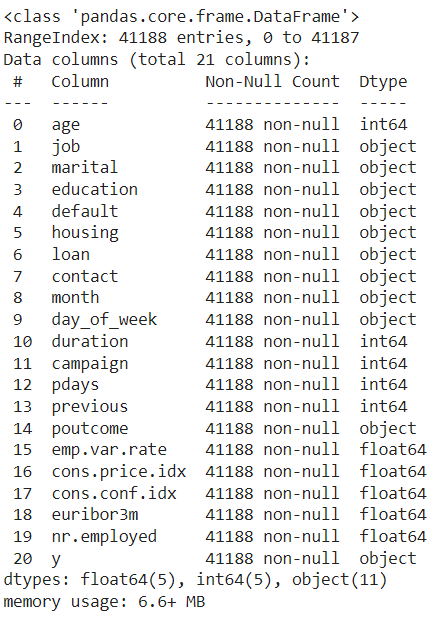
## **Describe data**

This is how the data frame look like:

****

Check the general infomation of data

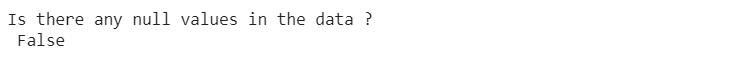
df.info()



Handle missing/duplicate data

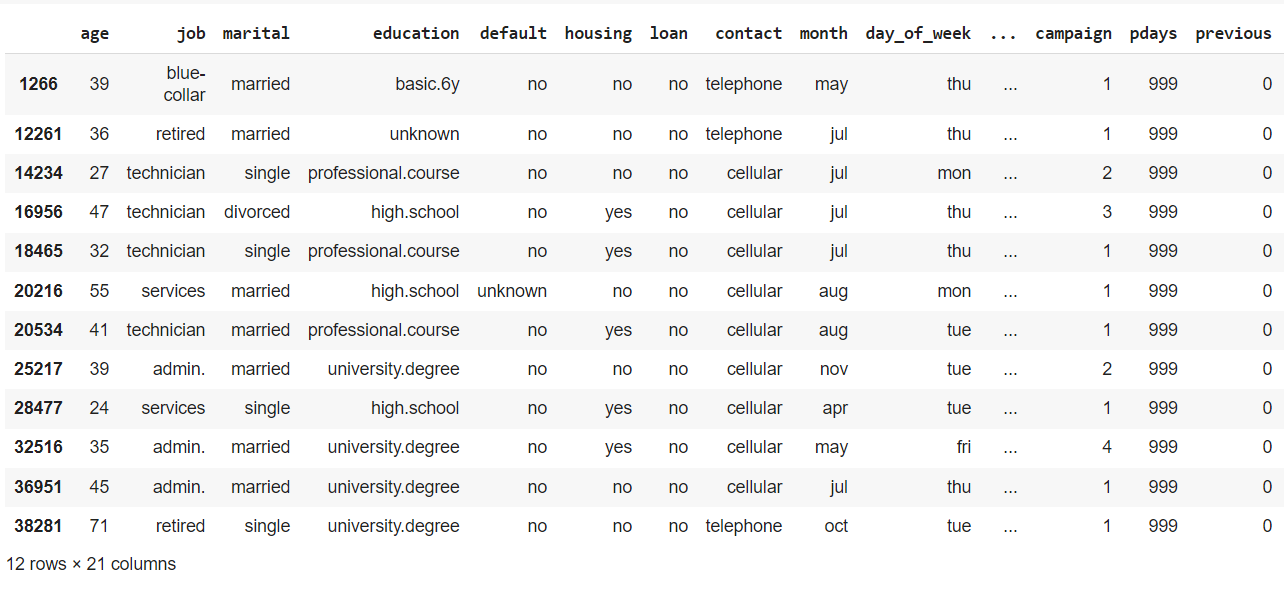
# Checking for missing values

print("Is there any null values in the data ? \n",df.isnull().values.any())



# Checking for duplicated rows

df[df.duplicated(keep='first')]

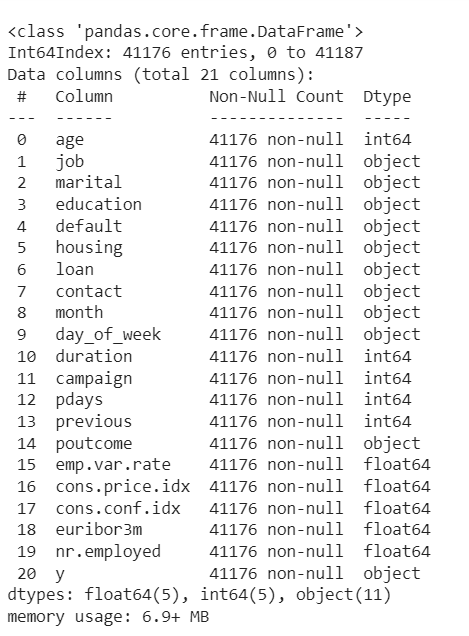


# Remove duplicated rows

df.drop\_duplicates(keep='first',inplace=True)

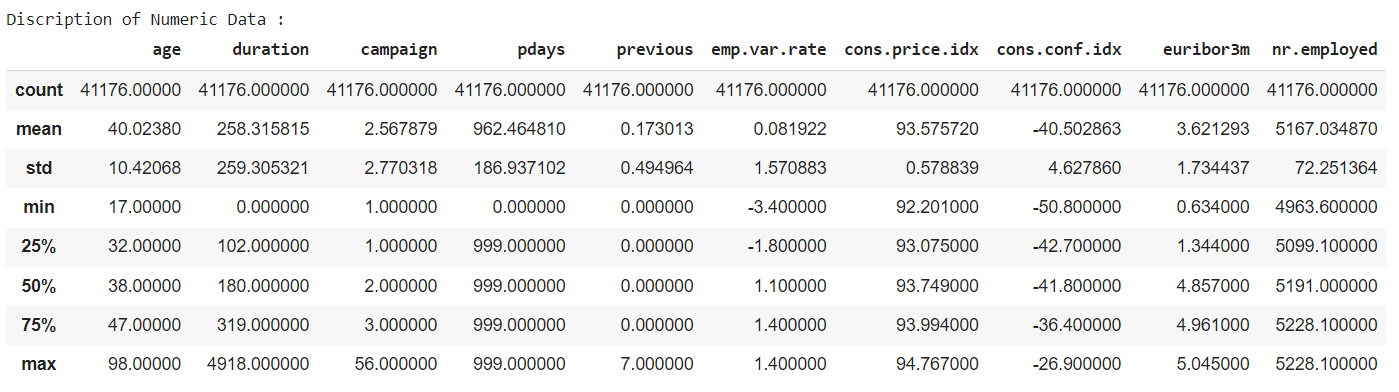
Check the general infomation after handled duplicated rows

df.info()



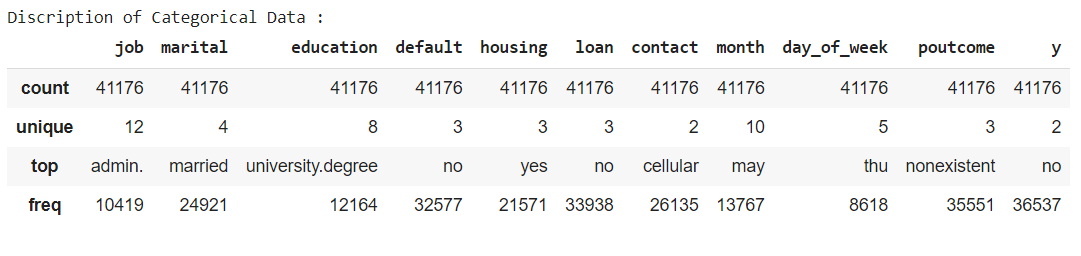
print('Discription of Numeric Data : ')

df.describe()



print('Discription of Categorical Data : ')

df.describe(include='object')



Create needed function to explore columns of categorical data:

def barplot\_mean(x, y, df, hue=None, order=None, hue\_order=None):

    print(df.groupby(x)[y].mean())

    uniqs = df[x].nunique()

    if uniqs > 4:

        plt.figure(figsize=(16,4))

    sns.barplot(x=x, y=y, data=df, estimator=np.mean, hue=hue, order=order, hue\_order=hue\_order)

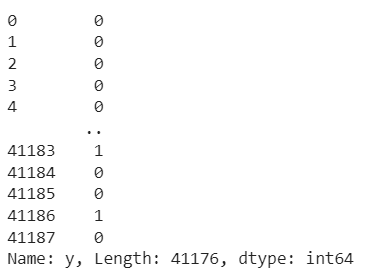
    plt.show()

This function will be use to observe the term deposit subscribe rate of each value in columns:

But first, we need to encode the target feature:

df.replace({"y" : {"no" : 0, "yes" : 1}}, inplace=True)

df.y



## **Explore categorical data**

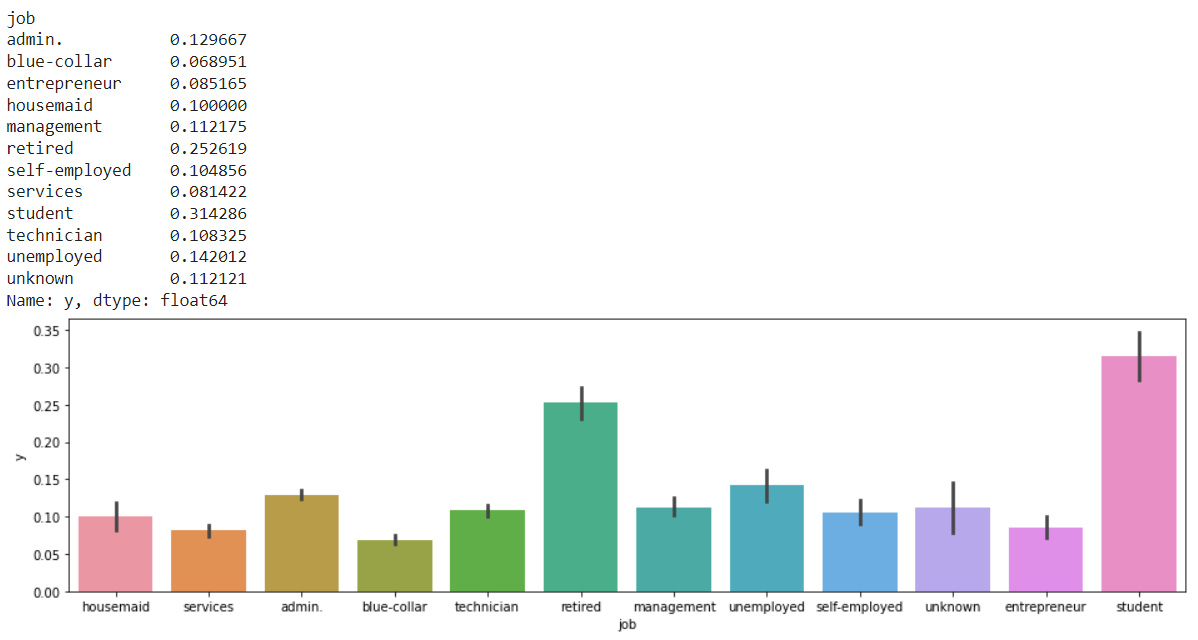
**Job feature:**

Number of observations of each job type:

df.job.value\_counts()



barplot\_mean('job', 'y', df)



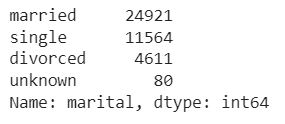
Most observations in the dataset have jobs of type admin, blue-collar and technician . Students and retirees are the subjects with the highest term deposit subscribe rate. In job features, we have a job type named ‘unknown’. This kind of value and Na-value(missing) is the same, so we replace ‘unknown’ value with the mode of column.

df.replace({"job" : {"unknown" : "admin."}}, inplace=True)

**Marital feature:**

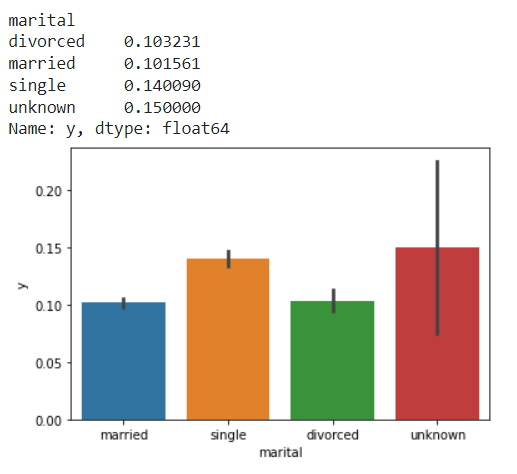
Number of observations of each marital status

df.marital.value\_counts()



The subscribe rate of each marital status

barplot\_mean('marital', 'y', df)



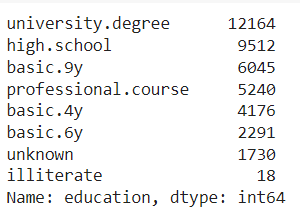
Most of the customers approached in the data set are married people, but the subscribe rate of single people is higher and more stable. This shows that the demand for term deposits of single people is higher than that of married people, which happens because it is possible that the spending or investment needs of single people are often lower. The 'unknown' group has a high subscriber rate, but the variance is very large. Similar to the job column, we replace this unknown value with the mode of the data column.

df.replace({"marital" : {"unknown" : "single"}}, inplace=True)

**Education feature:**

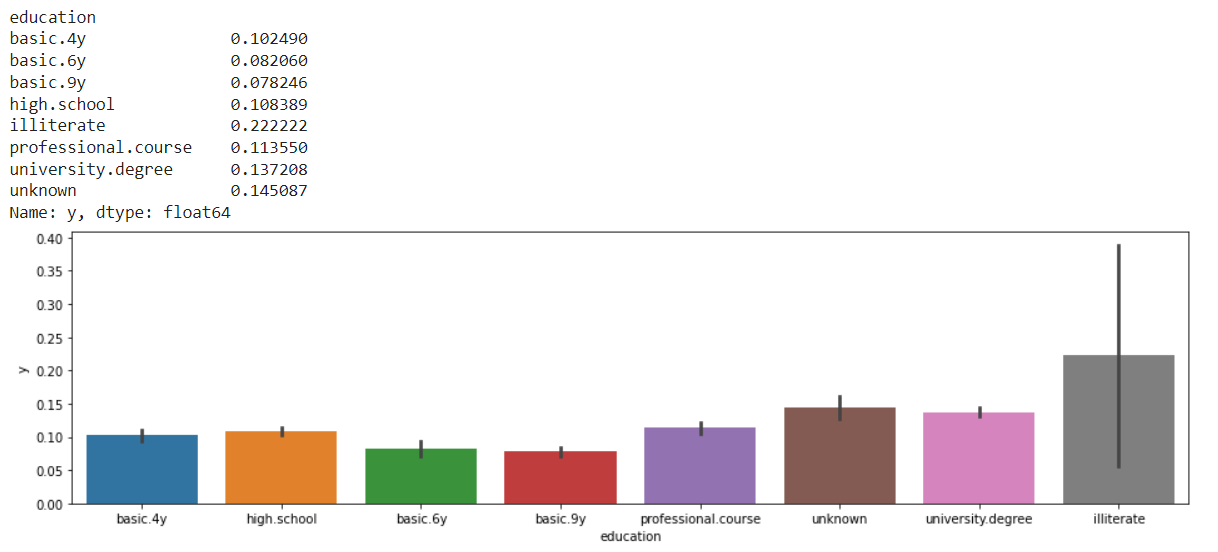
Number of observations of each education status

df.education.value\_counts()



The subscribe rate of each education status

barplot\_mean('education', 'y', df)



The majority of customers in the data set have college degrees and their subscription rates are just behind illiterate and unkown. However, illiterate people are not the target group we want to target and this group does not provide stable target values. So, we group the values ​​in the education variable as follows:

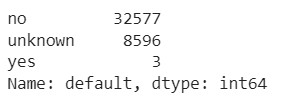
df.replace({'education' : {'unknown' : 'university.degree' }},inplace=True)

df.education.replace(['basic.4y', 'basic.6y', 'basic.9y'],'Basic', inplace=True)

**Default feature:**

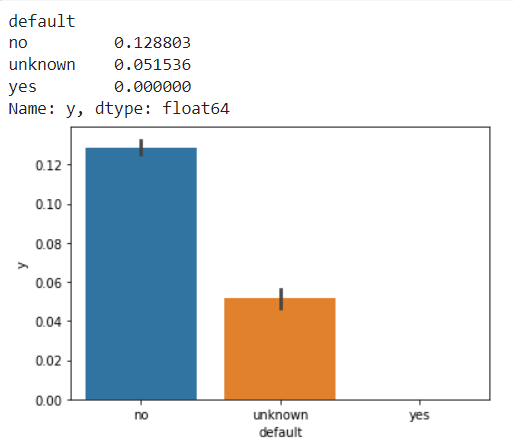
Number of observations of each value in default feature:

df.default.value\_counts()



The subscribe rate of each value in default feature:

barplot\_mean('default', 'y', df)

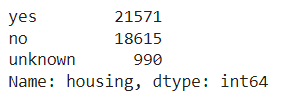


Unknown values ​​account for 20.87%, if we replace them with mode, it will create bias, which will affect the model. So we drop this freature from the dataset

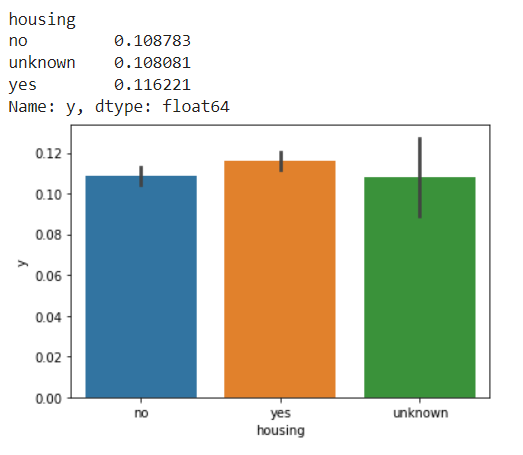
df.drop('default',axis=1,inplace=True)

**Housing feature:**

df.housing.value\_counts()



barplot\_mean('housing', 'y', df)

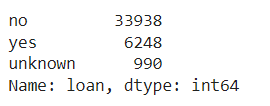


Most of the subjects in our dataset own homes (the number of people with homes is 16% higher than the number of people with homes). The savings needs of the group with a house are not too different: the subscription rate of those with a home is only 0.8% higher. For unknown values ​​in the tuple, we replace it with the value of mode (yes).

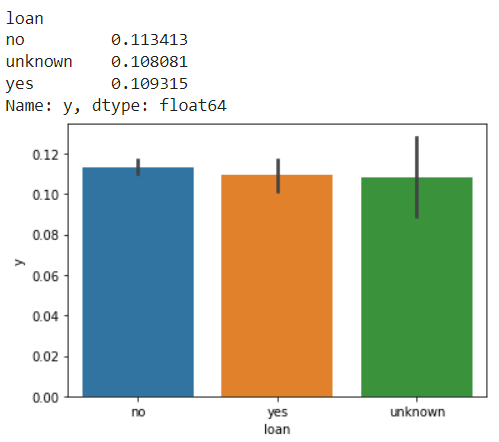
df.replace({'housing' : {'unknown' : 'yes' }}, inplace=True)

**Loan feature:**

df.loan.value\_counts()



barplot\_mean('loan', 'y', df)

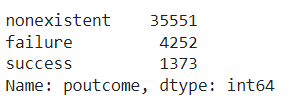


Most of the subjects in our dataset do not have any loans (the number of people without a loan is 5.43 times higher than the number of people with a loan). Subscription rates of borrowers and unbanked borrowers were similar and similar for the unknown group. Similar to the above variable, we also replace the unknown value with the data column's mode value.

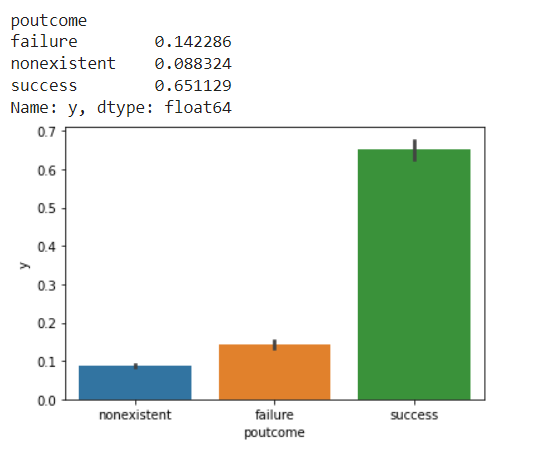
df.replace({'loan' : {'unknown' : 'no'}}, inplace=True)

**Poutcome feature:**

df.poutcome.value\_counts()

****

barplot\_mean('poutcome', 'y', df)

****

The majority of the subjects in our dataset are new customers, which means they haven't received a call from any of the current bank's previous marketing campaigns. Through the data we can see the results of marketing campaigns in the past, the number of failed calls is 4 times higher than successful calls. In the field of telemarketing banking products, especially for term savings products, this is an impressive figure (24,4%). However, what is in the past will not always happen in the future. While past marketing campaigns have achieved impressive results, the results of current marketing campaigns are only average (11,2%).

Note, the nonexistent value is of a completely different nature than the unknown value. This is not a missing value.

## **Explore continuous data**

for column in int\_column:

    plt.figure(figsize=(16,4))

    plt.subplot(1,3,1)

    sns.distplot(df[column], hist = True, color = "#07247D", hist\_kws = {'edgecolor':'black'})

    plt.xlabel(column)

    plt.ylabel('Density')

    plt.title(f'{column}  Distribution')

    plt.subplot(1,3,2)

    sns.boxplot(x='y', y=column, data =df, showmeans=True )

    plt.xlabel('Target')

    plt.ylabel(column)

    plt.title(f'{column}  Distribution')

    plt.subplot(1,3,3)

    counts, bins = np.histogram(df[column], bins=20, normed=True)

    cdf = np.cumsum (counts)

    plt.plot (bins[1:], cdf/cdf[-1])

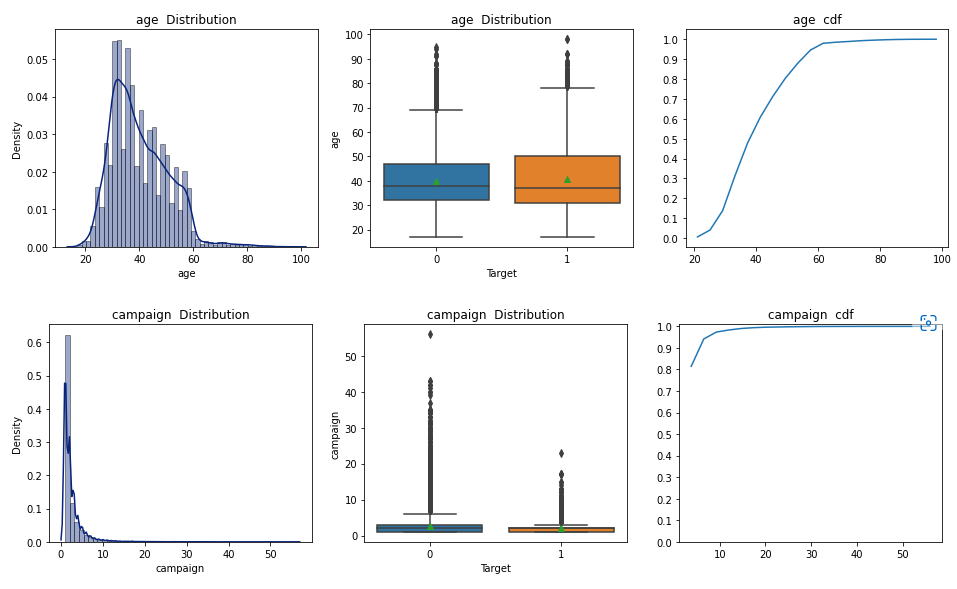
    #plt.xticks(range(15,100,5))

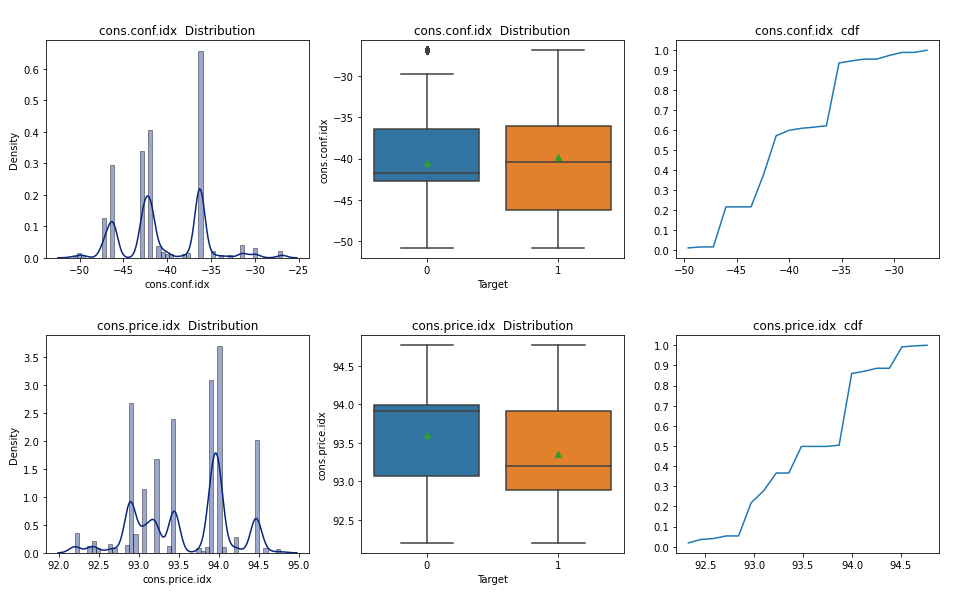
    plt.yticks(np.arange(0,1.1,.1))

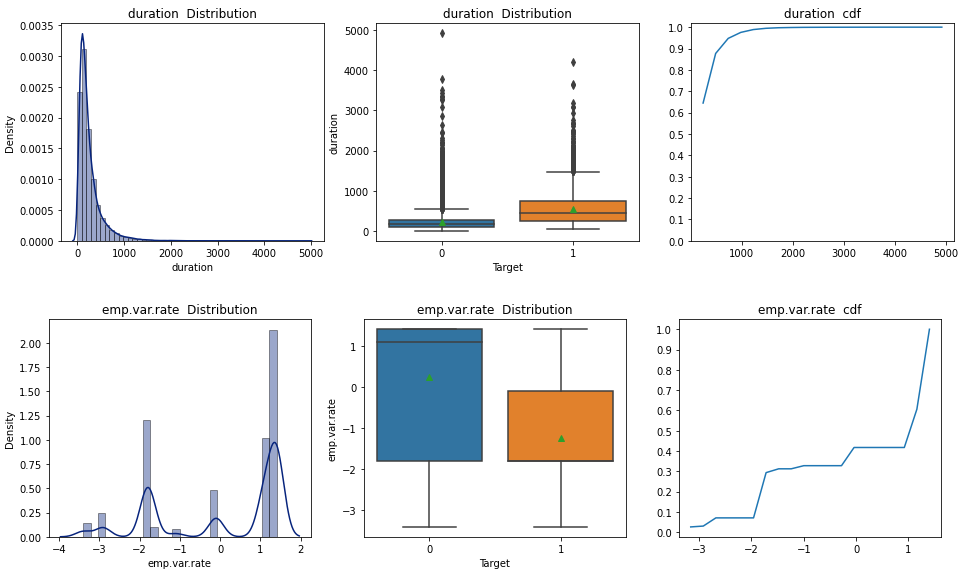
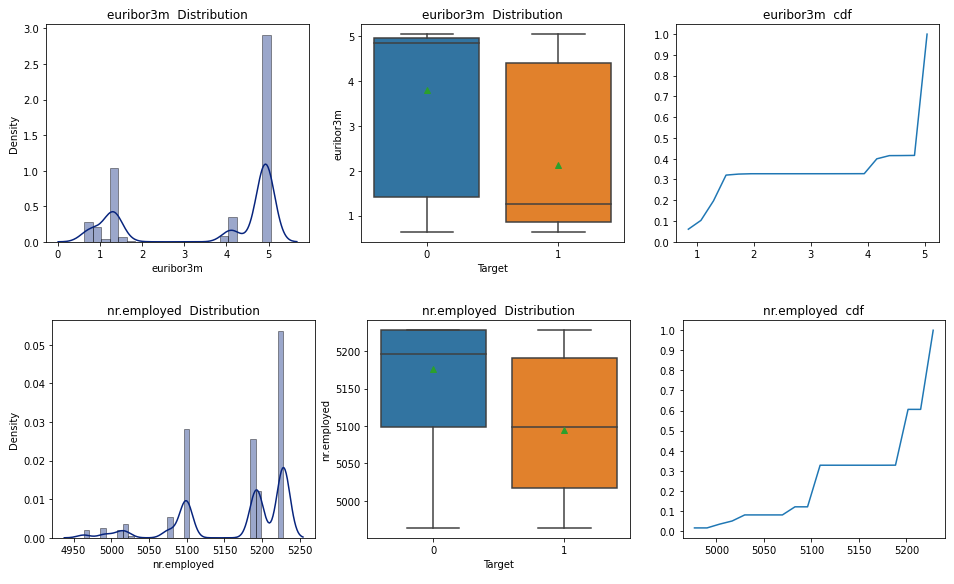
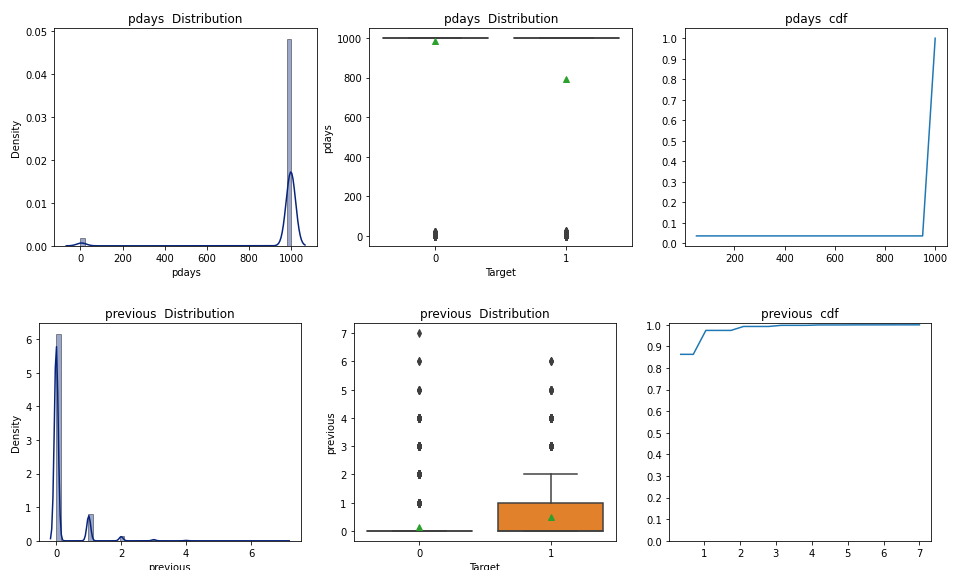
    plt.title(f'{column}  cdf')

    plt.show()

    print()





**Drop highly correlated features**

When independent variables are highly correlated, change in one variable would cause change to another and so the model results fluctuate significantly. The model results will be unstable and vary a lot given a small change in the data or model. So, we shoud remove some of the highly correlated independent variables

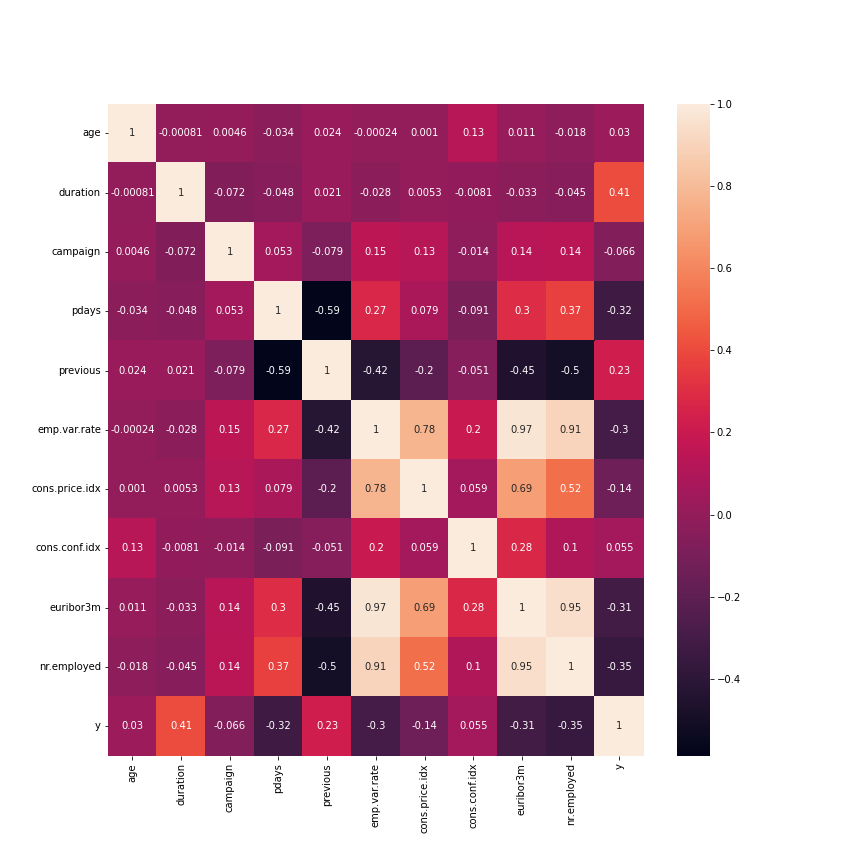
fig, ax = plt.subplots(figsize=(21,21))

dataplot = sns.heatmap(train.corr(), annot=True, ax = ax)

plt.xticks(rotation=45)

plt.show()

plt.close()

****

# Create correlation matrix

corr\_matrix = df.corr().abs()

# Select upper triangle of correlation matrix

upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(np.bool))

# Find features with correlation greater than 0.5

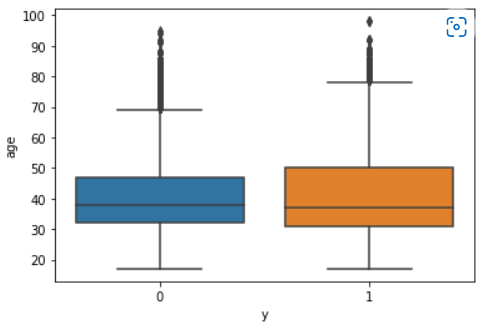
to\_drop = [column for column in upper.columns if any(upper[column] > 0.6)]

# Drop features

df.drop(to\_drop, axis=1, inplace=True)

print('dropped features: ',to\_drop)

**Age**



Through boxplot, we can detect outliers in the age column (for each result type). Non-subscribers over the age of 69 are considered outliers. So we remove all these observations from the dataset. In addition, under Portuguese law, only people over the age of 18 are allowed to open a bank account and have a term deposit, so we also exclude observations under the age of 18.

df.drop(df[df.age < 18].index, inplace=True)

df.drop(df[df.age > 69].index, inplace=True)

sns.boxplot(x='y',y='age',data=df)

**Drop some continuous features related with the last contact of the current campaign**

These features are directly related to the target variable, and are only defined when the target variable is defined. Therefore, it is not reasonable to use these variables to build a predictive model of the target variable.

df.drop(['contact','month','day\_of\_week','duration','campaign'],axis=1,inplace=True)

1. **Preprocessing data to build predict model**

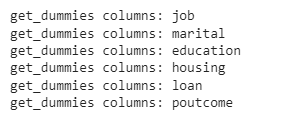
Encode target variable and other categorical variables in binary form

for col in df:

    if df[col].dtype == 'object':

      print('get\_dummies columns: '+ col)

      df = pd.get\_dummies(df, columns=[col],drop\_first=True)



**Train-test split**

X = df.drop(columns=['y'])

y = df['y']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state=42)

**Scaling data**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train[['age', 'pdays', 'previous', 'emp.var.rate', 'cons.conf.idx']] = scaler.fit\_transform(X\_train[['age', 'pdays', 'previous', 'emp.var.rate', 'cons.conf.idx']])

X\_test[['age', 'pdays', 'previous', 'emp.var.rate', 'cons.conf.idx']] = scaler.fit\_transform(X\_test[['age', 'pdays', 'previous', 'emp.var.rate', 'cons.conf.idx']])

The dataset has data points that are far apart, so we scale the data to make them closer together. Also, our data is not normally distributed, so use minmaxscaler instead of standardscaler.

**Handle imbalance data**

Our dataset is an unbalanced dataset. So we need to handle it before training the model to avoid overfitting. In addition, it is also necessary to change the way the model is evaluated later.

from imblearn.over\_sampling import SMOTE

from imblearn.under\_sampling import NearMiss

nm = NearMiss()

sm = SMOTE(random\_state=42)

X\_train\_sm, y\_train\_sm = sm.fit\_resample(X\_train, y\_train)

X\_train\_nm, y\_train\_nm = nm.fit\_resample(X\_train, y\_train)

1. **Build model and evaluating**

With imbalance data, oversampled data (SMOTE) and undersampled data( NearMiss), we build model with:

Logistic Regression (Logistic Regression)

RandomForestClassifier (Random Forest)

KneighborsClassifier (K-Nearest Neighbor)

XGBClassifier (XGBoost)

SVC (Support Vector Machine)

After building the models with each data set in turn, we evaluate and compare the effectiveness of the models through confusion matrix, precision and recall of class 1. The model that predicts the most accurately with the least amount of data used and least data waste will be selected.

# KNN with NearMiss

knn\_clf = KNeighborsClassifier(n\_neighbors = 21)

knn\_clf.fit(X\_train\_nm, y\_train\_nm)

y\_pred\_nm = knn\_clf.predict(X\_test)

print(classification\_report( y\_test, y\_pred\_nm))

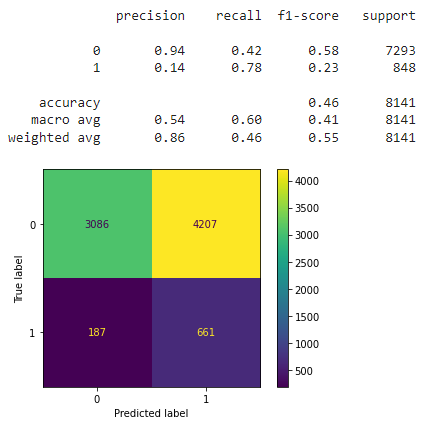
cm = confusion\_matrix(y\_test, y\_pred\_nm)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

plt.show()

plt.close()



With this model:

Success rate per call: 78%

We need to use 59.8% data to get 78% of subcriber miss 22%.