Here I intend to create a model to predict when a client will accept a term insurance.

Problem Statement

You are working for a new-age insurance company and employ mutiple outreach plans to sell term insurance to your customers. Telephonic marketing campaigns still remain one of the most effective way to reach out to people however they incur a lot of cost. Hence, it is important to identify the customers that are most likely to convert beforehand so that they can be specifically targeted via call. We are given the historical marketing data of the insurance company and are required to build a ML model that will predict if a client will subscribe to the insurance.

Features:

age (numeric) job : type of job marital : marital status educational_qual : education status call_type : contact communication type day: last contact day of the month (numeric) mon: last contact month of year dur: last contact duration, in seconds (numeric) num_calls: number of contacts performed during this campaign and for this client prev_outcome: outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")

Output variable (desired target):

y - has the client subscribed to the insurance?

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
data = pd.read_csv('train.csv')
data.head()
```

	age	job	marital	education_qual	call_type	day	mon	dur	num_calls	prev_(
0	58	management	married	tertiary	unknown	5	may	261	1	(
1	44	technician	single	secondary	unknown	5	may	151	1	(
2	33	entrepreneur	married	secondary	unknown	5	may	76	1	(
3	47	blue-collar	married	unknown	unknown	5	may	92	1	(
4	33	unknown	single	unknown	unknown	5	may	198	1	(

```
data = pd.read_csv('train.csv')
data=data.drop(['dur'],axis=1)
print(data.shape)
data.head()
```

```
(45211, 10)
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_outco
0	58	management	married	tertiary	unknown	5	may	1	unknov
1	44	technician	single	secondary	unknown	5	may	1	unknov
2	33	entrepreneur	married	secondary	unknown	5	may	1	unknov
3	47	blue-collar	married	unknown	unknown	5	may	1	unknov
4	33	unknown	single	unknown	unknown	5	may	1	unknov

```
data.tail()
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_ou
45206	51	technician	married	tertiary	cellular	17	nov	3	un
45207	71	retired	divorced	primary	cellular	17	nov	2	un

```
#Information of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 10 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_qual  45211 non-null  object
 4   call_type       45211 non-null  object
 5   day             45211 non-null  int64 
 6   mon             45211 non-null  object
 7   num_calls       45211 non-null  int64 
 8   prev_outcome    45211 non-null  object
 9   y               45211 non-null  object
dtypes: int64(3), object(7)
memory usage: 3.4+ MB
```

```
data.describe()
```

	age	day	num_calls
count	45211.000000	45211.000000	45211.000000
mean	40.936210	15.806419	2.763841
std	10.618762	8.322476	3.098021
min	18.000000	1.000000	1.000000
25%	33.000000	8.000000	1.000000
50%	39.000000	16.000000	2.000000
75%	48.000000	21.000000	3.000000
max	95.000000	31.000000	63.000000

```
# knowing the job categorical variables
data["job"].unique()
```

```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

```
# knowing the age categorical variables
data["age"].unique()
```

```
array([58, 44, 33, 47, 35, 28, 42, 43, 41, 29, 53, 57, 51, 45, 60, 56, 32,
       25, 40, 39, 52, 46, 36, 49, 59, 37, 50, 54, 55, 48, 24, 38, 31, 30,
       27, 34, 23, 26, 61, 22, 21, 20, 66, 62, 83, 75, 67, 70, 65, 68, 64,
       69, 72, 71, 19, 76, 85, 63, 90, 82, 73, 74, 78, 80, 94, 79, 77, 86,
       95, 81, 18, 89, 84, 87, 92, 93, 88])
```

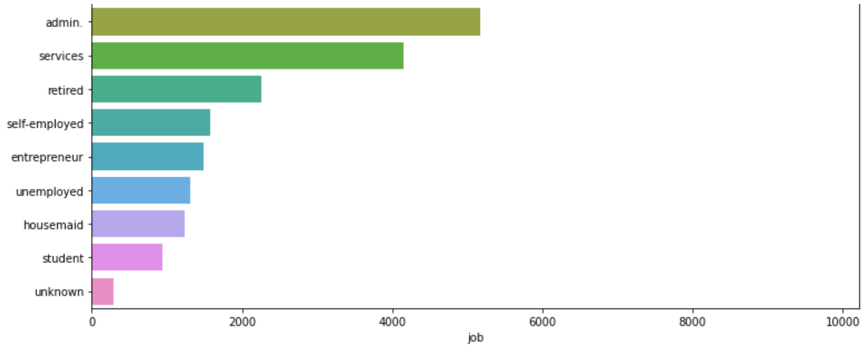
```
data["marital"].unique()
```

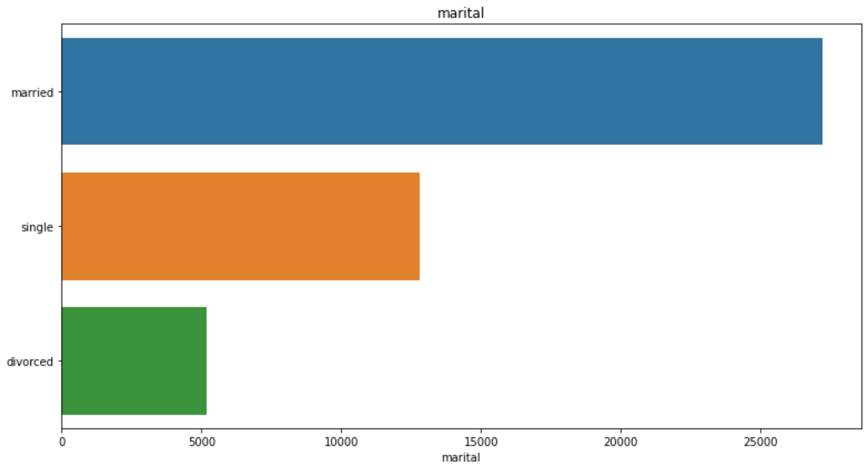
```
array(['married', 'single', 'divorced'], dtype=object)
```

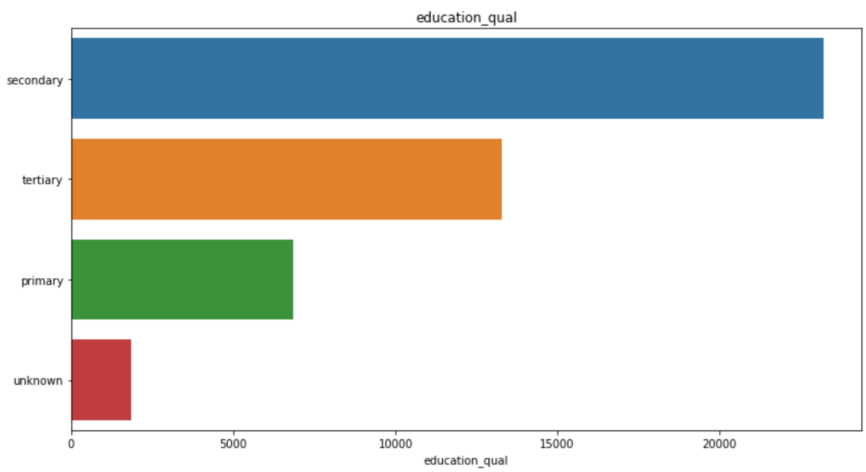
```
data["education_qual"].unique()
```

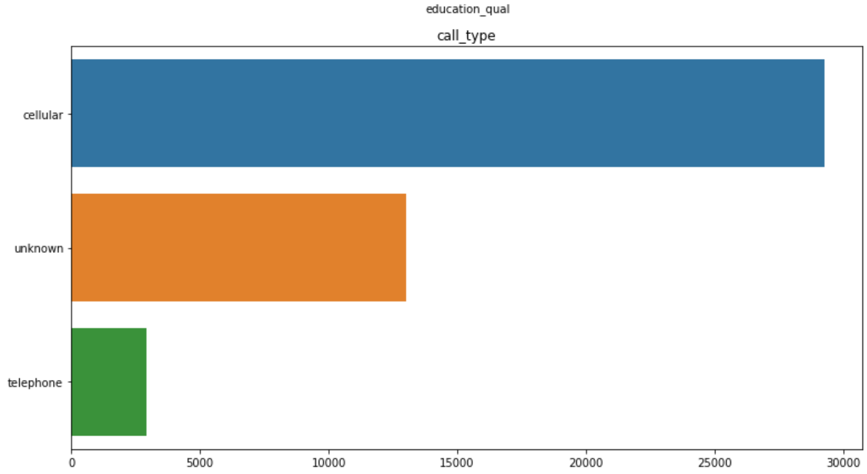
```
array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)
```

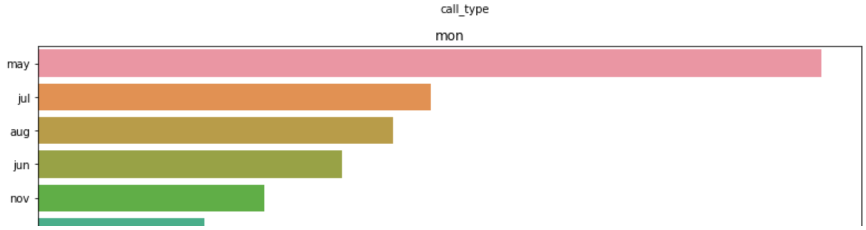
```
categori=['job', 'marital', 'education_qual', 'call_type', 'mon', 'prev_outcome','y']
for col in categori:
    plt.figure(figsize=(11,6))
    sns.barplot(data[col].value_counts(),data[col].value_counts().index,data=data)
    plt.title(col)
    plt.tight_layout()
```


admin.
services
retired
self-employed
entrepreneur
unemployed
housemaid
student
unknown
0
2000
4000
6000
8000
10000
job


marital
married
single
divorced
0
5000
10000
15000
20000
25000
marital


education_qual
secondary
tertiary
primary
unknown
0
5000
10000
15000
20000
education_qual


call_type
cellular
unknown
telephone
0
5000
10000
15000
20000
25000
30000
call_type


mon
may
jul
aug
jun
nov

Input Categorical feature Observation

Job - More Job types are Admin ,mgmt, Technician and blue-collor and it means bank targeting high salaried people.

Marital - more people of type married

Education_qual - more count in secondary and tertiary degree people . High salaried people would have more degree expected. And illiterate count is very less.

mon- May is busy

prev_outcome -outcome of the previous marketing campaign- Success is small rate.

Categorize variables correlated with Target Variables

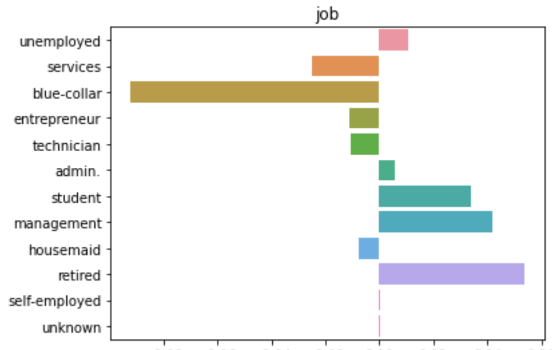
```
#Check How Categorize variables correlated with Target Variables and How it impacted.
from scipy import stats
```

```
#Check How Job Type correlated with Target Variable
data.groupby(['job','y']).y.count()

#Admin are more interested in Term Deposit.
```

```
job            y
admin.         no     4540
               yes     631
blue-collar    no     9024
               yes     708
entrepreneur   no     1364
               yes     123
housemaid      no     1131
               yes     109
management     no     8157
               yes    1301
retired        no     1748
               yes     516
self-employed  no     1392
               yes     187
services       no     3785
               yes     369
student        no      669
               yes     269
technician     no     6757
               yes     840
unemployed     no     1101
               yes     202
unknown        no      254
               yes      34
Name: y, dtype: int64
```

```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'job'
pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

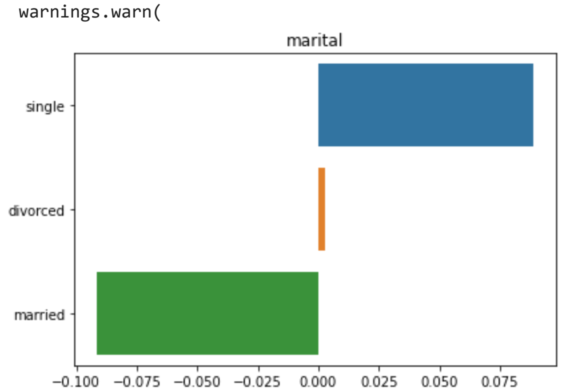



```
data.groupby(['marital','y']).y.count()
#married people are more interested in Term Deposit
```

```
marital   y
divorced  no      4585
          yes      622
married   no     24459
          yes     2755
single    no     10878
          yes     1912
Name: y, dtype: int64
```

```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'marital'

# ==========================================================================================================

pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

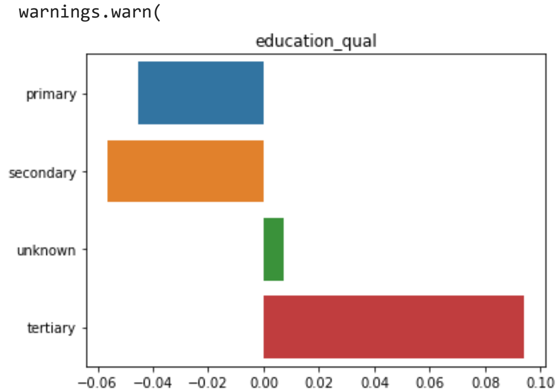



```
data.groupby(['education_qual','y']).y.count()
```

```
education_qual  y
primary         no      6260
                yes      591
secondary       no     20752
                yes     2450
tertiary        no     11305
                yes     1996
unknown         no      1605
                yes      252
Name: y, dtype: int64
```

```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'education_qual'

# ======================================================================================================

pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```



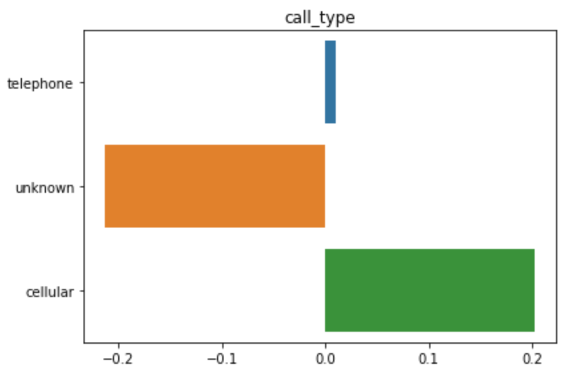

```
data.groupby(['call_type','y']).y.count()
```

```
call_type  y
cellular   no     24916
           yes     4369
telephone  no      2516
           yes      390
unknown    no     12490
           yes      530
Name: y, dtype: int64
```

```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'call_type'

# ======================================================================================================

pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()
```

```
all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

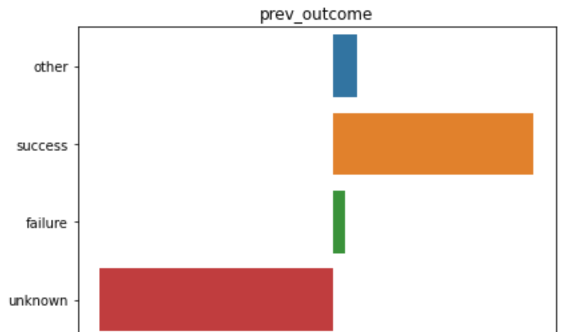



```
data.groupby(['prev_outcome','y']).age.count()
```

```
prev_outcome  y
failure       no      4283
              yes      618
other         no      1533
              yes      307
success       no       533
              yes      978
unknown       no     33573
              yes     3386
Name: age, dtype: int64
```

```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'prev_outcome'

# ==========================================================================================================

pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

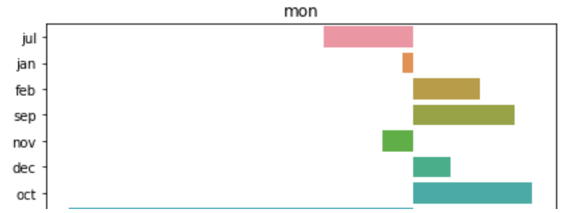



```
data.groupby(['mon','y']).age.count()
```

```
mon  y
apr  no      2355
     yes      577
aug  no      5559
     yes      688
dec  no       114
     yes      100
feb  no      2208
     yes      441
jan  no      1261
     yes      142
jul  no      6268
     yes      627
jun  no      4795
     yes      546
mar  no       229
     yes      248
may  no     12841
     yes      925
nov  no      3567
     yes      403
oct  no       415
     yes      323
sep  no       310
     yes      269
Name: age, dtype: int64
```

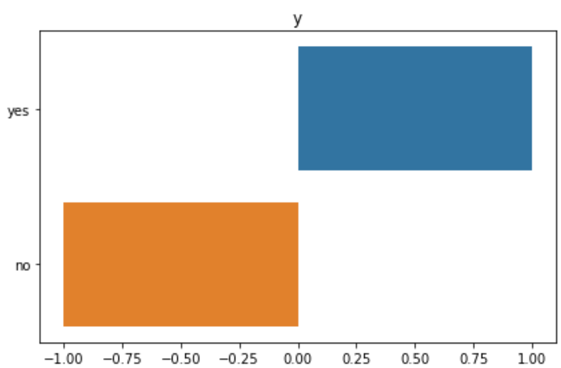
```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'mon'

# ======================================================================================================

pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```




```
#Normalized distribution of each class per feature and plotted difference between positive and negative frequencies.
#Positive values imply this category favors clients that will subscribe and negative values categories that favor not buying the
#product.
feature_name = 'y'

# ====================================================================================================

pos_counts = data.loc[data.y.values == 'yes', feature_name].value_counts()
neg_counts = data.loc[data.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_counts.index) + list(neg_counts.index)))

#Counts of how often each outcome was recorded.
freq_pos = (data.y.values == 'yes').sum()
freq_neg = (data.y.values == 'no').sum()

pos_counts = pos_counts.to_dict()
neg_counts = neg_counts.to_dict()

all_index = list(all_counts)
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```



Inference/Result: There are unknown values for many variables in the Data set. There are many ways to handle missing data. One of the ways is to discard the row but that would lead to reduction of data set and hence would not serve our purpose of building an accurate and realistic prediction model.

Other method is to smartly infer the value of the unknown variable from the other variables. This a way of doing an imputation where we use other independent variables to infer the value of the missing variable. This doesn't gurantee that all missing values will be addressed but majority of them will have a reasonable which can be useful in the prediction.

Variables with unknown/missing values are : 'education', 'job', 'call_type' .

Therefore, we start with creating new variables for the unknown values in 'education', 'job'. We do this to see if the values are missing at random or is there a pattern in the missing values.

```
from sklearn.model_selection import train_test_split


# Saperating features and result vectors
y=data[['y']]
```

```
X = data.drop(['y'], axis=1)
#y = data['y'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
X_test.columns
```

```
Index(['age', 'job', 'marital', 'education_qual', 'call_type', 'day', 'mon',
       'num_calls', 'prev_outcome'],
      dtype='object')
```

```
X_train.columns
```

```
Index(['age', 'job', 'marital', 'education_qual', 'call_type', 'day', 'mon',
       'num_calls', 'prev_outcome'],
      dtype='object')
```

```
y_train.head()
```

	y
10747	no
26054	no
9125	no
41659	no
4443	no

```
y_test.head()
```

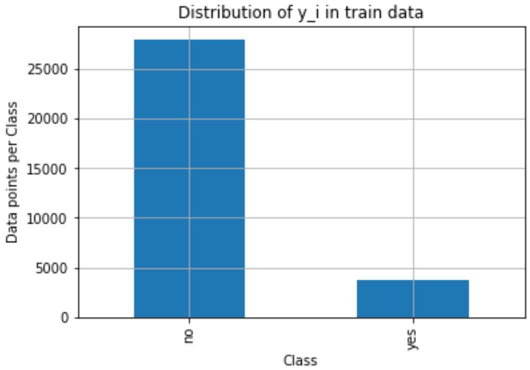
	y
3776	no
9928	no
33409	no
31885	no
15738	no

Distribution of train and test data

```
def plot_distribution(class_distribution,title,xlabel,ylabel):
    class_distribution.plot(kind='bar')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid()
    plt.show()
```

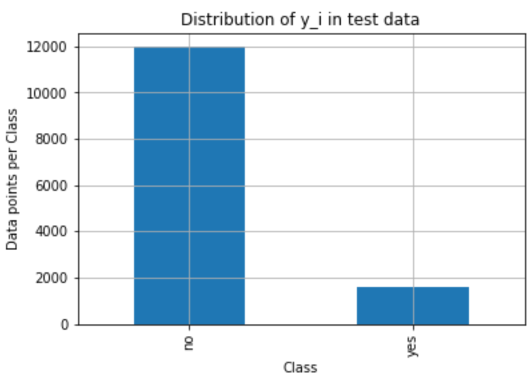
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = y_train['y'].value_counts()
test_class_distribution = y_test['y'].value_counts()
```

```
plot_distribution(train_class_distribution,
                 'Distribution of y_i in train data',
                 'Class',
                 'Data points per Class')

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i],
          '(', np.round((train_class_distribution.values[i]/X_train.shape[0]*100), 3), '%)')

print('-'*80)
```




```
Number of data points in class 1 : 27956 ( 88.337 %)
Number of data points in class 2 : 3691 ( 11.663 %)
```

```
plot_distribution(test_class_distribution,
                 'Distribution of y_i in test data',
                 'Class',
                 'Data points per Class')

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(test_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],
          '(', np.round((test_class_distribution.values[i]/X_test.shape[0]*100), 3), '%)')

print('-'*80)
```




```
Number of data points in class 1 : 11966 ( 88.219 %)
Number of data points in class 2 : 1598 ( 11.781 %)
--------------------------------------------------------------------------------
```

Distribution of both train
ndnmADDDx22HDFVRVMVMFVDDFFDJNLKNLKNLKNLKNJBYHYGUYGYHGBJHYVJGCFRXDTRSETRDDHJJHJKHGJHVFHFRHJFJHFJDJSXJS
HDCGHFCJCJKDXJDXJHDCHXJXKMJJHHUJUJUIIUIUIOOIOIJIJHJH

```
# concatinate train data for data manupulation
data = pd.concat([X_train, y_train], axis=1)
```

```
data.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_o
10747	36	technician	single	tertiary	unknown	17	jun	4	ui
26054	56	entrepreneur	married	secondary	cellular	19	nov	3	ui
9125	46	blue-collar	married	secondary	unknown	5	jun	2	ui
41659	41	management	divorced	tertiary	cellular	1	oct	1	s
4443	38	blue-collar	married	secondary	unknown	20	may	1	ui

```
# concatinate test data for data manupulation
data_1= pd.concat([X_test, y_test], axis=1)
```

```
data_1.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_o
3776	40	blue-collar	married	secondary	unknown	16	may	1	ur
9928	47	services	single	secondary	unknown	9	jun	2	ur
33409	25	student	single	tertiary	cellular	20	apr	1	ur
31885	42	management	married	tertiary	cellular	9	apr	1	
15738	56	management	married	tertiary	cellular	21	jul	2	ur

Now, to infer the missing values in 'job' and 'education', we make use of the cross-tabulation between 'job' and 'education'. Our hypothesis here is that 'job' is influenced by the 'education' of a person. Hence, we can infer 'job' based on the education of the person. Moreover, since we are just filling the missing values, we are not much concerned about the causal inference. We, therefore, can use the job to predict the education.

```
def cross_tab(data,f1,f2):
    # find no of unique values in jobs colums
    jobs=list(data[f1].unique())
    # find no of unique values in education columns
    edu=list(data[f2].unique())
    dataframes=[]
    for e in edu:
        dfe=data[data[f2]==e]
        dfejob=dfe.groupby(f1).count()[f2]
        dataframes.append(dfejob)
    xx=pd.concat(dataframes,axis=1)
    xx.columns=edu
    xx=xx.fillna(0)
    return xx
```

```
cross_tab(data,'job','education_qual')
```

job	tertiary	secondary	primary	unknown
admin.	381	2986	150	117
blue-collar	98	3818	2622	325
entrepreneur	453	395	132	58
housemaid	111	281	447	36
management	5419	791	195	168
retired	263	685	563	85
self-employed	590	415	90	29
services	146	2408	246	107
student	161	344	33	111
technician	1334	3682	105	179
unemployed	196	489	176	24
unknown	30	47	43	83

Inferring education from jobs : From the cross-tabulation, it can be seen that people with management jobs usually have a tertiary degree. Hence wherever 'job' = management and 'education_qual' = unknown, we can replace with 'tertiary degree'. Similarly, 'job' = 'services' then 'education' = 'secondary' and 'job' = 'technician' then 'education' = 'Secondary'.

While imputing the values for job and education, we were cognizant of the fact that the correlations should make real world sense. If it didn't make real world sense, we didn't replace the missing values.

```
data['job'][data['age']>60].value_counts()
```

```
retired          604
management        76
housemaid         40
technician        18
```

```
blue-collar      17
unknown          16
admin.           16
self-employed    15
unemployed        9
entrepreneur      8
services          2
Name: job, dtype: int64
```

Inferring jobs from age: As we see, if 'age' > 60, then the 'job' is 'retired,' which makes sense.

```
data.loc[(data['age']>60) & (data['job']=='unknown'), 'job'] = 'retired'
data.loc[(data['education_qual']=='unknown') & (data['job']=='management'), 'education_qual'] = 'tertiary'
data.loc[(data['education_qual']=='unknown') & (data['job']=='services'), 'education_qual'] = 'secondary'
data.loc[(data['education_qual']=='unknown') & (data['job']=='housemaid'), 'education_qual'] = 'primary'
data.loc[(data['job'] == 'unknown') & (data['education_qual']=='secondary'), 'job'] = 'blue-collar'
data.loc[(data['job'] == 'unknown') & (data['education_qual']=='primary'), 'job'] = 'blue-collar'
data.loc[(data['job'] == 'unknown') & (data['education_qual']=='tertiary'), 'job'] = 'management'
```

```
cross_tab(data,'job','education_qual')
```

job	tertiary	secondary	primary	unknown
admin.	381.0	2986.0	150.0	117.0
blue-collar	98.0	3861.0	2664.0	325.0
entrepreneur	453.0	395.0	132.0	58.0
housemaid	111.0	281.0	483.0	0.0
management	5613.0	791.0	195.0	0.0
retired	267.0	689.0	564.0	92.0
self-employed	590.0	415.0	90.0	29.0
services	146.0	2515.0	246.0	0.0
student	161.0	344.0	33.0	111.0
technician	1334.0	3682.0	105.0	179.0
unemployed	196.0	489.0	176.0	24.0
unknown	0.0	0.0	0.0	76.0

```
data.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_o
10747	36	technician	single	tertiary	unknown	17	jun	4	ur
26054	56	entrepreneur	married	secondary	cellular	19	nov	3	ur
9125	46	blue-collar	married	secondary	unknown	5	jun	2	ur
41659	41	management	divorced	tertiary	cellular	1	oct	1	s
4443	38	blue-collar	married	secondary	unknown	20	may	1	ur

As we can see, we are able to reduce the number of unknowns and enhance our data set.

Numerical variables

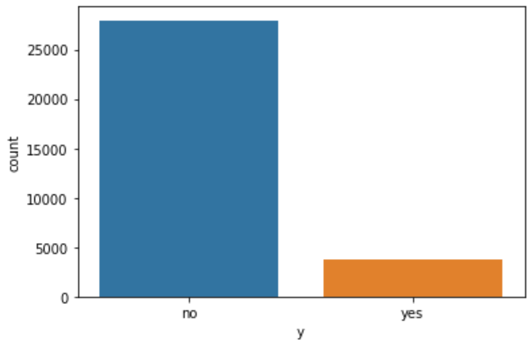
```
numerical_variables = ['age','day', 'num_calls']
data[numerical_variables].describe()
```

	age	day	num_calls
count	31647.000000	31647.000000	31647.000000
mean	40.941669	15.829621	2.772237
std	10.632010	8.323200	3.154004
min	18.000000	1.000000	1.000000
50%	39.000000	16.000000	2.000000

Balancing y out

```
sns.countplot(x='y',data=data)
```

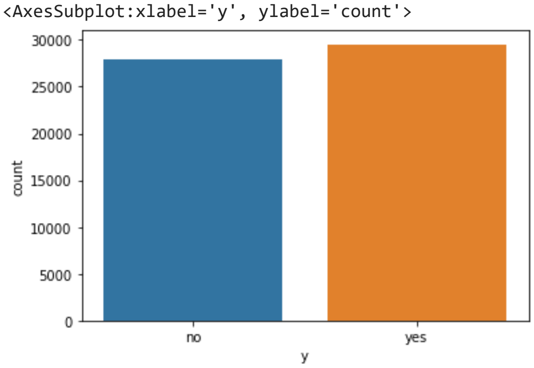
```
<AxesSubplot:xlabel='y', ylabel='count'>
```




```
d1=data.copy()
d2=d1[d1.y=='yes']
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
data=d1
```

```
sns.countplot(x='y',data=data)
```

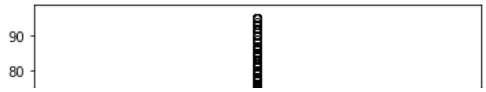
```
<AxesSubplot:xlabel='y', ylabel='count'>
```



outlier check Outliers Outliers are defined as 1.5 x Q3 value (75th percentile).

```
# Check outlier if any for Numberic column.
data.age.plot(kind='box')
# There are outlier and check max age and age greater than 90
```

```
<AxesSubplot:>
```



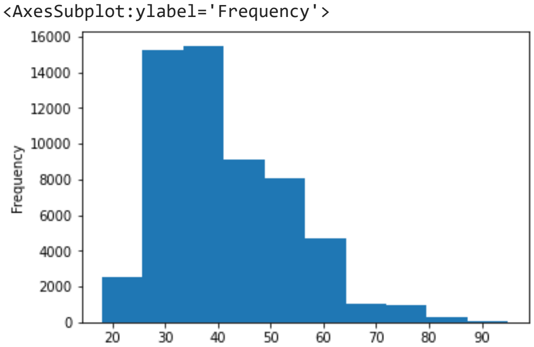

```
print(data.age.max())
data[data['age'] > 80].head(5)
```

```
95
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_outcome
41982	81	retired	married	primary	cellular	27	oct	1	unknown
42266	81	retired	married	primary	telephone	13	nov	1	other
45010	86	retired	married	primary	cellular	14	oct	2	success
41387	82	retired	married	primary	cellular	1	sep	1	unknown
44893	81	retired	divorced	primary	cellular	27	sep	2	other

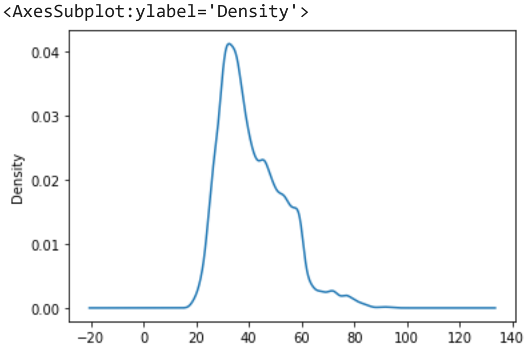
```
data.age.plot(kind='hist')
```

```
<AxesSubplot:ylabel='Frequency'>
```




```
data.age.plot(kind='kde')
```

```
<AxesSubplot:ylabel='Density'>
```




```
# Create Binning for all numeric fields base on Box plot quantile
def binning(dataframe,featureName):
    print (featureName)
    q1 = dataframe[featureName].quantile(0.25)
    q2 = dataframe[featureName].quantile(0.50)
    q3 = dataframe[featureName].quantile(0.75)
    dataframe.loc[(dataframe[featureName] <= q1), featureName] = 1
    dataframe.loc[(dataframe[featureName] > q1) & (dataframe[featureName] <= q2), featureName] = 2
    dataframe.loc[(dataframe[featureName] > q2) & (dataframe[featureName] <= q3), featureName] = 3
    dataframe.loc[(dataframe[featureName] > q3), featureName] = 4
    print (q1, q2, q3)
```

```
binning(data,'age')
```

```
age
32.0 39.0 49.0
```

```
data.head(5)
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_o
10747	2	technician	single	tertiary	unknown	17	jun	4	ui
26054	4	entrepreneur	married	secondary	cellular	19	nov	3	ui
9125	3	blue-collar	married	secondary	unknown	5	jun	2	ui
41659	3	management	divorced	tertiary	cellular	1	oct	1	s
4443	2	blue-collar	married	secondary	unknown	20	may	1	ui

Standardizing the data

```
data.columns
```

```
Index(['age', 'job', 'marital', 'education_qual', 'call_type', 'day', 'mon',
       'num_calls', 'prev_outcome', 'y'],
      dtype='object')
```

```
data.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_o
10747	2	technician	single	tertiary	unknown	17	jun	4	ui
26054	4	entrepreneur	married	secondary	cellular	19	nov	3	ui
9125	3	blue-collar	married	secondary	unknown	5	jun	2	ui
41659	3	management	divorced	tertiary	cellular	1	oct	1	s
4443	2	blue-collar	married	secondary	unknown	20	may	1	ui

```
idx_numeric=[0,5,7]
scaler = MinMaxScaler()
data[data.columns[idx_numeric]] = scaler.fit_transform(data[data.columns[idx_numeric]])
```

Categorical variables can be either Ordinal or Nominal

```
data['prev_outcome'] = data['prev_outcome'].map({'failure': -1,'unknown': 0,'success': 1})
```

Handling Nominal Variables(One Hot Encoding) 'job', 'marital', 'education_qual', 'call_type', 'mon', are Nominal Variables

```
# One hot encoding of nominal varibles
nominal = ['job','marital','education_qual','call_type','mon']
data_clean = pd.get_dummies(data,columns=nominal)
data_clean['y']=data_clean['y'].map({'yes': 1,'no': 0})
data_clean.head()
```

	age	day	num_calls	prev_outcome	y	job_admin.	job_blue-collar	job_entrepr
10747	0.333333	0.533333	0.048387	0.0	0	0	0	
26054	1.000000	0.600000	0.032258	0.0	0	0	0	
9125	0.666667	0.133333	0.016129	0.0	0	0	1	
41659	0.666667	0.000000	0.000000	1.0	0	0	0	
4443	0.333333	0.633333	0.000000	0.0	0	0	1	

5 rows × 39 columns

```
data_clean.columns
```

```
Index(['age', 'day', 'num_calls', 'prev_outcome', 'y', 'job_admin.',
       '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_divorced', 'marital_married', 'marital_single',
           'education_qual_primary', 'education_qual_secondary',
           'education_qual_tertiary', 'education_qual_unknown',
           'call_type_cellular', 'call_type_telephone', 'call_type_unknown',
           'mon_apr', 'mon_aug', 'mon_dec', 'mon_feb', 'mon_jan', 'mon_jul',
           'mon_jun', 'mon_mar', 'mon_may', 'mon_nov', 'mon_oct', 'mon_sep'],
          dtype='object')
```

```
data_clean.shape
```

```
(57484, 39)
```

```
df_with_dummies=pd.get_dummies(data_clean)
```

```
def dropfeature(df,f):
    """Drops one of the dummy variables."""
    df=df.drop(f,axis=1)
    return df
```

```
features_dropped = ['marital_single','call_type_cellular',
                    'education_qual_unknown','job_unknown',]
data_clean = dropfeature(df_with_dummies, features_dropped)
```

Aanalising the data distribution by plotting graphs for numerical fields

```
data_clean.describe()
```

	age	day	num_calls	prev_outcome	y	job_admin.
count	57484.000000	57484.000000	57484.000000	54597.000000	57484.000000	57484.000000
mean	0.482169	0.485999	0.024013	-0.012034	0.513673	0.116798
std	0.375666	0.279845	0.043376	0.473704	0.499817	0.321182
min	0.000000	0.000000	0.000000	-1.000000	0.000000	0.000000
25%	0.000000	0.233333	0.000000	0.000000	0.000000	0.000000
50%	0.333333	0.466667	0.016129	0.000000	1.000000	0.000000
75%	0.666667	0.666667	0.032258	0.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 35 columns

```
data_clean.head()
```

	age	day	num_calls	prev_outcome	y	job_admin.	job_blue-collar	job_entrepr
10747	0.333333	0.533333	0.048387	0.0	0	0	0	
26054	1.000000	0.600000	0.032258	0.0	0	0	0	
9125	0.666667	0.133333	0.016129	0.0	0	0	1	
41659	0.666667	0.000000	0.000000	1.0	0	0	0	
4443	0.333333	0.633333	0.000000	0.0	0	0	1	

5 rows × 35 columns

```
data_clean.shape
```

```
(57484, 35)
```

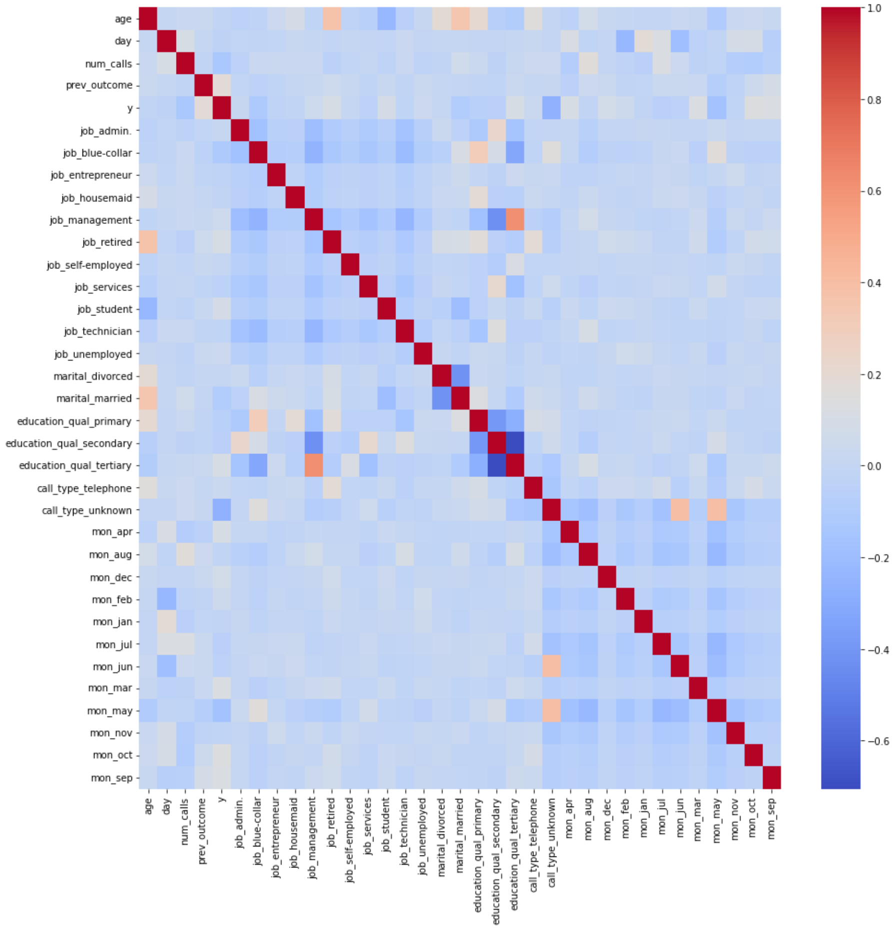
```
data_clean.corr()
```

	age	day	num_calls	prev_outcome	y	job_adm
age	1.000000	0.005288	0.007675	0.029079	-0.016568	-0.043
day	0.005288	1.000000	0.134921	0.004395	-0.037434	-0.009
num_calls	0.007675	0.134921	1.000000	-0.006371	-0.132661	-0.020
prev_outcome	0.029079	0.004395	-0.006371	1.000000	0.176175	-0.006
y	-0.016568	-0.037434	-0.132661	0.176175	1.000000	0.007
job_admin.	-0.043872	-0.009283	-0.020362	-0.006248	0.007715	1.000
job_blue-collar	-0.030609	-0.016039	0.020025	-0.042355	-0.115243	-0.172
job_entrepreneur	0.035957	-0.007753	0.033381	-0.024253	-0.031299	-0.062
job_housemaid	0.090344	0.016480	0.019914	0.006826	-0.017571	-0.058
job_management	-0.026250	0.004613	0.009487	0.016126	0.049689	-0.195
job_retired	0.376207	-0.006763	-0.045151	0.050165	0.101561	-0.101
job_self-employed	-0.029329	0.002561	0.000200	0.016087	0.004846	-0.070
job_services	-0.061891	-0.004749	0.003293	-0.020738	-0.045874	-0.108
job_student	-0.222818	-0.006168	-0.030974	0.018462	0.090626	-0.067
job_technician	-0.056482	0.025650	0.017695	-0.013233	-0.021317	-0.159
job_unemployed	0.007826	-0.000301	-0.025248	0.030102	0.035827	-0.067
marital_divorced	0.189120	0.002722	-0.022666	0.002420	0.006972	0.030
marital_married	0.345786	0.004699	0.049880	-0.003463	-0.095458	-0.039
education_qual_primary	0.201928	-0.002930	0.021452	-0.016528	-0.063234	-0.112
education_qual_secondary	-0.071418	-0.005405	-0.017989	-0.021527	-0.055289	0.229
education_qual_tertiary	-0.090175	0.006462	0.001633	0.033282	0.103503	-0.158
call_type_telephone	0.160605	0.019567	0.051644	0.006360	0.029619	-0.011
call_type_unknown	-0.004575	-0.005603	0.044011	0.011775	-0.270437	-0.000
mon_apr	-0.041995	0.117493	-0.074219	-0.047746	0.100927	0.006
mon_aug	0.077895	-0.016701	0.134063	0.035176	-0.013375	-0.061
mon_dec	0.018679	-0.003650	-0.016678	0.032471	0.075337	-0.002
mon_feb	0.003380	-0.225011	-0.029741	-0.013765	0.045854	-0.003
mon_jan	-0.000572	0.199216	-0.051877	-0.007339	-0.012997	0.013
mon_jul	0.000353	0.123083	0.126779	0.027175	-0.053948	0.006
mon_jun	0.024287	-0.193162	0.043813	0.028614	-0.040205	-0.000
mon_mar	0.013197	-0.038549	-0.034872	0.022009	0.126974	-0.002
mon_may	-0.105972	-0.008561	-0.037450	-0.078202	-0.169783	0.033
mon_nov	0.026439	0.064140	-0.073048	-0.028728	-0.020837	-0.001
mon_oct	0.040724	0.093828	-0.070872	0.044056	0.139604	0.006
mon_sep	0.018446	-0.070907	-0.051125	0.082868	0.126445	0.004

35 rows × 35 columns

```
def drawheatmap(df):
    '''Builds the heat map for the given data'''
    f, ax = plt.subplots(figsize=(15, 15))
    sns.heatmap(df.corr(method='spearman'), annot=False, cmap='coolwarm')
```

```
drawheatmap(data_clean)
```



Inferences: From the above heat map we can see that 'y' (our target variable) has good correlation with 'poutcome_success' , 'poutcome_unknown'. We expect to see these independent variables as significant while building the models.

Standardizing the test data

```
data_1= pd.concat([X_test, y_test], axis=1)
```

```
data_1.shape
```

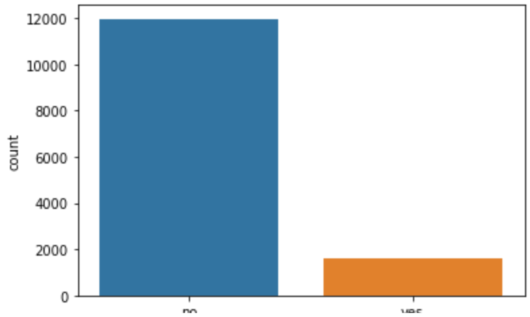
```
(13564, 10)
```

```
data_1.columns
```

```
Index(['age', 'job', 'marital', 'education_qual', 'call_type', 'day', 'mon',
       'num_calls', 'prev_outcome', 'y'],
      dtype='object')
```

```
sns.countplot(x='y',data=data_1)
```

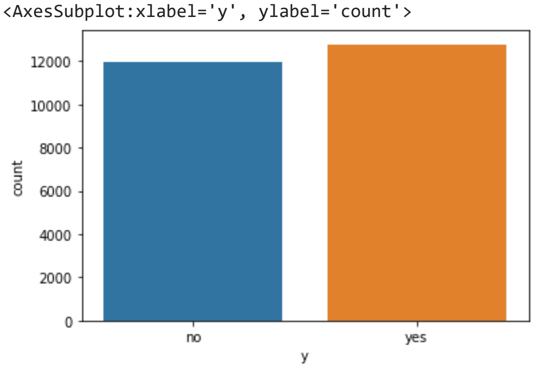
```
<AxesSubplot:xlabel='y', ylabel='count'>
```




```
d1=data_1.copy()
d2=d1[d1.y=='yes']
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
data_1=d1
```

```
sns.countplot(x='y',data=data_1)
```

```
<AxesSubplot:xlabel='y', ylabel='count'>
```




```
data_1.columns
```

```
Index(['age', 'job', 'marital', 'education_qual', 'call_type', 'day', 'mon',
       'num_calls', 'prev_outcome', 'y'],
      dtype='object')
```

```
data_1.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_calls	prev_o
3776	40	blue-collar	married	secondary	unknown	16	may	1	ur
9928	47	services	single	secondary	unknown	9	jun	2	ur
33409	25	student	single	tertiary	cellular	20	apr	1	ur
31885	42	management	married	tertiary	cellular	9	apr	1	
15738	56	management	married	tertiary	cellular	21	jul	2	ur

```
idx_numeric=[0,5,7]
scaler = MinMaxScaler()
data_1[data_1.columns[idx_numeric]] = scaler.fit_transform(data_1[data_1.columns[idx_numeric]])
```

```
data_1.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_call
3776	0.293333	blue-collar	married	secondary	unknown	0.500000	may	0.0000C
9928	0.386667	services	single	secondary	unknown	0.266667	jun	0.01851

Categorical variables can be either Ordinal or Nominal

```
data_1['prev_outcome'] = data_1['prev_outcome'].map({'failure': -1,'unknown': 0,'success': 1})
```

```
data_1.head()
```

	age	job	marital	education_qual	call_type	day	mon	num_call
3776	0.293333	blue-collar	married	secondary	unknown	0.500000	may	0.0000C
9928	0.386667	services	single	secondary	unknown	0.266667	jun	0.01851
33409	0.093333	student	single	tertiary	cellular	0.633333	apr	0.0000C
31885	0.320000	management	married	tertiary	cellular	0.266667	apr	0.0000C
15738	0.506667	management	married	tertiary	cellular	0.666667	jul	0.01851

```
data_1.shape
```

```
(24750, 10)
```

Handling Nominal Variables(One Hot Encoding) 'job', 'marital', 'education_qual', 'call_type', 'mon' are Nominal Variables

```
# One hot encoding of nominal varibles
nominal = ['job','marital','education_qual','call_type','mon']
data_clean_1 = pd.get_dummies(data_1,columns=nominal)
data_clean_1['y']=data_clean_1['y'].map({'yes': 1,'no': 0})
data_clean_1.head()
```

	age	day	num_calls	prev_outcome	y	job_admin.	job_blue-collar	job_entrepr
3776	0.293333	0.500000	0.000000	0.0	0	0	1	
9928	0.386667	0.266667	0.018519	0.0	0	0	0	
33409	0.093333	0.633333	0.000000	0.0	0	0	0	
31885	0.320000	0.266667	0.000000	-1.0	0	0	0	
15738	0.506667	0.666667	0.018519	0.0	0	0	0	

5 rows × 39 columns

```
data_clean_1.shape
```

```
(24750, 39)
```

```
df_with_dummies=pd.get_dummies(data_clean_1)
```

```
def dropfeature(df,f):
    """Drops one of the dummy variables."""
    df=df.drop(f,axis=1)
    return df
```

```
features_dropped = ['marital_single','call_type_cellular',
                    'education_qual_unknown','job_unknown','marital_single','call_type_cellular',
                    'education_qual_unknown','job_unknown']
data_clean_1 = dropfeature(df_with_dummies, features_dropped)
```

```
data_clean_1.shape
```

```
(24750, 35)
```

```
data_clean.shape
```

```
(57484, 35)
```

```
data_clean_1.columns
```

```
Index(['age', 'day', 'num_calls', 'prev_outcome', 'y', 'job_admin.',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'marital_divorced',
       'marital_married', 'education_qual_primary', 'education_qual_secondary',
       'education_qual_tertiary', 'call_type_telephone', 'call_type_unknown',
       'mon_apr', 'mon_aug', 'mon_dec', 'mon_feb', 'mon_jan', 'mon_jul',
       'mon_jun', 'mon_mar', 'mon_may', 'mon_nov', 'mon_oct', 'mon_sep'],
      dtype='object')
```

```
data_clean.columns
```


```
Index(['age', 'day', 'num_calls', 'prev_outcome', 'y', 'job_admin.',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'marital_divorced',
       'marital_married', 'education_qual_primary', 'education_qual_secondary',
       'education_qual_tertiary', 'call_type_telephone', 'call_type_unknown',
       'mon_apr', 'mon_aug', 'mon_dec', 'mon_feb', 'mon_jan', 'mon_jul',
       'mon_jun', 'mon_mar', 'mon_may', 'mon_nov', 'mon_oct', 'mon_sep'],
      dtype='object')
```


Model

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
```

```
data_clean
```

	age	day	num_calls	prev_outcome	y	job_admin.	job_blue-	job_entrepre

```
import pandas as pd
import numpy as np
data_clean = data_clean.replace(np.nan,0)


# Saperating features and result vectors
y_test=data_clean_1[['y']]
X_test = data_clean_1.drop(['y'], axis=1)

#y = data['y'].values


# Saperating features and result vectors
y_train=data_clean[['y']]
X_train = data_clean.drop(['y'], axis=1)

#y = data['y'].values


def Convert_Model(X_train,y_train,X_test,y_test,classifier):
     from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matrix
     classifier.fit(X_train,y_train)
     print(classifier.score(X_test,y_test))
     print(confusion_matrix(y_test,classifier.predict(X_test)))
     print(accuracy_score(y_test,classifier.predict(X_test)))
     print(precision_score(y_test,classifier.predict(X_test)))
     print(recall_score(y_test,classifier.predict(X_test)))
     f1 = 2 * precision_score(y_test,classifier.predict(X_test)) * recall_score(y_test,classifier.predict(X_test)) / (precision_score(y_test,
     print("f1 score", f1)
     return classifier


X_train = X_train.replace(np.nan,0)
X_test = X_test.replace(np.nan,0)


X_train
```

	age	day	num_calls	prev_outcome	job_admin.	job_blue-collar	job_entrepren
10747	0.333333	0.533333	0.048387	0.0	0	0	
26054	1.000000	0.600000	0.032258	0.0	0	0	
9125	0.666667	0.133333	0.016129	0.0	0	1	
41659	0.666667	0.000000	0.000000	1.0	0	0	
4443	0.333333	0.633333	0.000000	0.0	0	1	
...	...	...	...	...	...	...	
43021	1.000000	0.366667	0.032258	0.0	0	0	
43323	1.000000	0.566667	0.000000	1.0	0	0	
41606	0.000000	0.566667	0.016129	-1.0	0	0	
16023	0.333333	0.700000	0.016129	0.0	0	0	
11284	0.666667	0.566667	0.000000	0.0	0	0	

57484 rows × 34 columns

```
X_test
```

	age	day	num_calls	prev_outcome	job_admin.	job_blue-collar	job_entreprene
3776	0.293333	0.500000	0.000000	0.0	0	1	
9928	0.386667	0.266667	0.018519	0.0	0	0	
33409	0.093333	0.633333	0.000000	0.0	0	0	
31885	0.320000	0.266667	0.000000	-1.0	0	0	
15738	0.506667	0.666667	0.018519	0.0	0	0	
...	...	...	...	...	...	...	
23775	0.280000	0.900000	0.000000	0.0	0	0	
43067	0.640000	0.566667	0.000000	1.0	0	0	
4916	0.253333	0.666667	0.000000	0.0	0	0	

```
# import Dummy Classifier for creating Base Model
from sklearn.dummy import DummyClassifier
classifier = DummyClassifier(strategy='most_frequent',random_state=0)
finalModel = Convert_Model(X_train,y_train,X_test,y_test,classifier)
```

```
0.5165252525252525
[[    0 11966]
 [    0 12784]]
0.5165252525252525
0.5165252525252525
1.0
f1 score 0.681195715884265
```

Logistical Regression

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

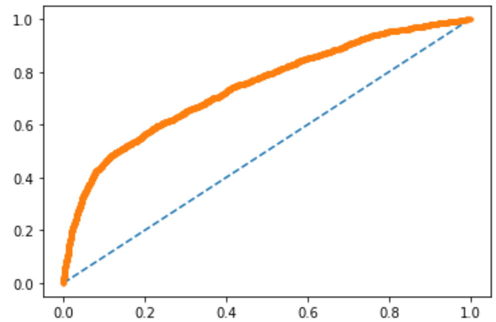
```
# import Dummy Classifier for creating Base Model
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(random_state=0)
finalModel_lr = Convert_Model(X_train,y_train,X_test,y_test,classifier_lr)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
0.6735757575757576
[[8311 3655]
 [4424 8360]]
0.6735757575757576
0.6957969205160216
0.6539424280350438
f1 score 0.6742207347070447
```

```
# roc curve and auc on imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
probs = finalModel_lr.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
```

```
# show the plot
pyplot.show()
```

```
AUC: 0.745
```




```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
# roc curve and auc on imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
```

Training Random Forest Classifier

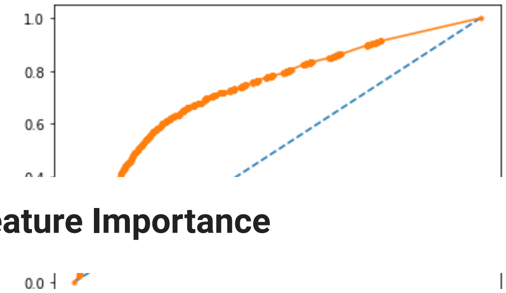
```
from sklearn.ensemble import GradientBoostingClassifier
rfc = RandomForestClassifier(n_estimators=100)
finalModel_rfc = Convert_Model(X_train,y_train,X_test,y_test,rfc)
```

```
<ipython-input-208-568d4f1df485>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  classifier.fit(X_train,y_train)
0.6173737373737374
[[10808  1158]
 [ 8312  4472]]
0.6173737373737374
0.794316163410302
0.3498122653316646
f1 score 0.48571738894319544
```

Testing

```
probs = finalModel_rfc.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

AUC: 0.728



Feature Importance

```
data_clean.head()
```

	age	day	num_calls	prev_outcome	y	job_admin.	job_blue-collar	job_entrepr
10747	0.333333	0.533333	0.048387	0.0	0	0	0	
26054	1.000000	0.600000	0.032258	0.0	0	0	0	
9125	0.666667	0.133333	0.016129	0.0	0	0	1	
41659	0.666667	0.000000	0.000000	1.0	0	0	0	
4443	0.333333	0.633333	0.000000	0.0	0	0	1	

5 rows × 35 columns

```
X = data_clean.drop('y', axis=1).values
y = data_clean['y'].values
pp=data_clean.drop('y', axis=1)
x_train, x_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.3, random_state=42)
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
feature_importances = pd.DataFrame(rfc.feature_importances_,index = pp.columns,columns=['importance']).sort_values('importance',ascending=Fal
```

```
<ipython-input-220-4a2d78d1c25d>:6: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  rfc.fit(X_train, y_train)
```

```
feature_importances
```

	importance
day	0.297596
num_calls	0.131465
prev_outcome	0.090839
age	0.085053
call_type_unknown	0.053007
marital_married	0.023484
mon_may	0.018714
marital_divorced	0.016454
call_type_telephone	0.016313
education_qual_secondary	0.015643
mon_oct	0.015367
job_technician	0.015074
mon_jun	0.013436
education_qual_tertiary	0.013341
job_management	0.013189
mon_mar	0.013019

SVM Classifier

mon_jul 0.012026

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

Choosing the best parameters for SVM classifier based on 2-fold Cross Validation score

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [0.1], 'C': [1]},
                    {'kernel': ['linear'], 'C': [1]}]
```

job_student 0.006052

```
clf = GridSearchCV(SVC(), tuned_parameters, cv=2, scoring='precision')
```

```
finalModel_gb = Convert_Model(X_train,y_train,X_test,y_test,clf)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
0.7471088806458651
[[9648 2318]
 [5936 6848]]
0.6665050505050505
0.7471088806458651
0.5356695869837297
f1 score 0.6239635535307517
```

```
print('The best model is: ', finalModel_gb.best_params_)
print('This model produces a mean cross-validated score (precision) of', finalModel_gb.best_score_)
```

```
The best model is:  {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
This model produces a mean cross-validated score (precision) of 0.7957589240699039
```

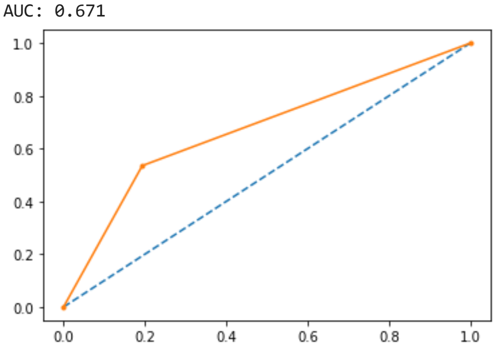
Testing

```
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
y_true, y_pred = y_test, finalModel_gb.predict(X_test)
pre1 = precision_score(y_true, y_pred)
rec1 = recall_score(y_true, y_pred)
acc1 = accuracy_score(y_true, y_pred)
f1_1 = f1_score(y_true, y_pred)
print('precision on the evaluation set: ', pre1)
print('recall on the evaluation set: ', rec1)
print('accuracy on the evaluation set: ', acc1)
print("F1 on the evaluation set",f1_1)
```

```
precision on the evaluation set:  0.7471088806458651
recall on the evaluation set:  0.5356695869837297
accuracy on the evaluation set:  0.6665050505050505
F1 on the evaluation set 0.6239635535307517
```

```
# roc curve and auc on imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
probs = finalModel_gb.predict(X_test)
# keep probabilities for the positive outcome only

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 0.671
```

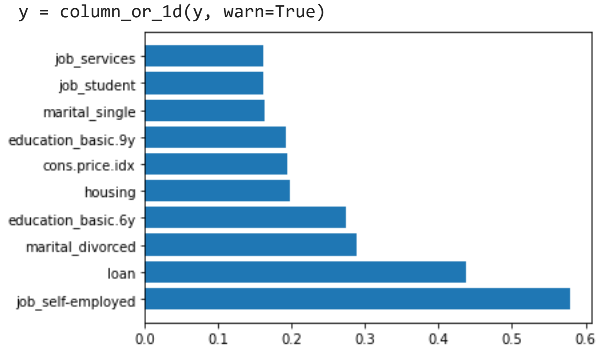



```
from matplotlib import pyplot as plt
from sklearn import svm
from matplotlib import pyplot as plt
X = data_clean.drop('y', axis=1).values
y = data_clean['y'].values
pp=data_clean.drop('y', axis=1)
x_train, x_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.3, random_state=42)
```

```
def f_importances(coef, names, top=-1):
    imp = coef
```

```
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.show()

# whatever your features are called
features_names = ['age', 'default', 'housing', 'loan', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y', 'pdays2', 'job_admin.',
       '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_divorced', 'marital_married', 'marital_single',
       'marital_unknown', 'education_basic.4y', 'education_basic.6y',
       'education_basic.9y', 'education_high.school', 'education_illiterate',
       'education_professional.course', 'education_university.degree',
       'education_unknown', 'contact_cellular', 'contact_telephone',
       'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun',
       'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
       'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu',
       'day_of_week_tue', 'day_of_week_wed']
svm = svm.SVC(kernel='linear')
svm.fit(X_train, y_train)

# Specify your top n features you want to visualize.
# You can also discard teh abs() function
# if you are interested in negative contribution of features
f_importances(abs(svm.coef_[0]), features_names, top=10)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionW
  y = column_or_1d(y, warn=True)
```



Classify the model using XGBClassifier

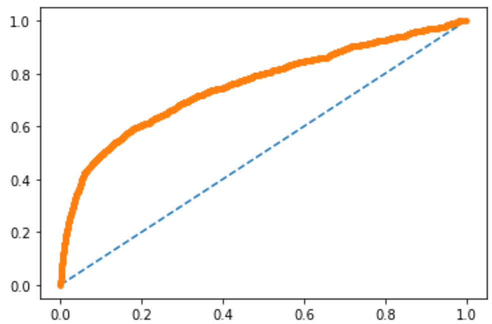
```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# fit model no training data
model = XGBClassifier()
finalModel_XGB = Convert_Model(X_train,y_train,X_test,y_test,model)
```

```
0.7007272727272728
[[9999 1967]
 [5440 7344]]
0.7007272727272728
0.7887444957577059
0.574468085106383
f1 score 0.664765784114053
```

```
#ROC curve
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
probs = finalModel_XGB.predict_proba(X_test)
```

```
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 0.757
```




```
X = data_clean.drop('y', axis=1).values
y = data_clean['y'].values
pp=data_clean.drop('y', axis=1)
x_train, x_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.3, random_state=42)
rmodel = XGBClassifier()
rmodel.fit(X_train, y_train)
feature_importances = pd.DataFrame(rmodel.feature_importances_,index = pp.columns,columns=['importance']).sort_values('importance',ascending=
```

```
feature_importances
```

	importance
call_type_unknown	0.134471
prev_outcome	0.105088
mon_oct	0.094608
mon_mar	0.070694
mon_jun	0.050018
mon_sep	0.048248
mon_dec	0.039026
mon_feb	0.036298
mon_nov	0.033141
mon_apr	0.032443
mon_jul	0.027895
mon_jan	0.026946

MLP Classifier with 3 layer

job_retired 0.021100

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
seed = 7
test_size = 0.33
mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)
mlp.fit(X_train,y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:
  y = column_or_1d(y, warn=True)
                      MLPClassifier
MLPClassifier(hidden_layer_sizes=(13, 13, 13), max_iter=500)
```

```
from sklearn.metrics import classification_report,confusion_matrix
predictions = mlp.predict(X_test)
#print the confusion matrix
print(confusion_matrix(y_test,predictions))
```

```
[[9245 2721]
 [4752 8032]]
```

marital_divorced 0.010723

```
#Print the classification report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.66      0.77      0.71     11966
           1       0.75      0.63      0.68     12784

    accuracy                           0.70     24750
   macro avg       0.70      0.70      0.70     24750
weighted avg       0.71      0.70      0.70     24750
```

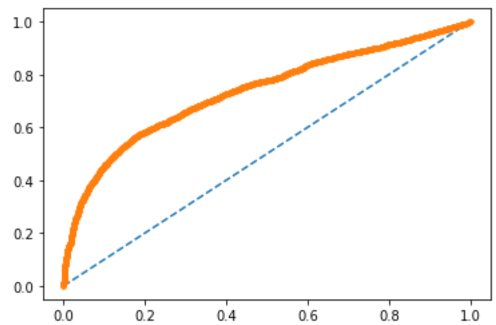
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score


classifier_mlp = MLPClassifier(hidden_layer_sizes=(13,14,15 ) ,max_iter=500)
finalModel_mlp = Convert_Model(X_train,y_train,X_test,y_test,classifier_mlp)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A column-vector y w
  y = column_or_1d(y, warn=True)
0.6848484848484848
[[9310 2656]
 [5144 7640]]
0.6848484848484848
0.7420357420357421
```

```
0.5976220275344181
f1 score 0.6620450606585789
```

```
# roc curve and auc on imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
probs = finalModel_mlp.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 0.736
```



MLP Classifier with 2 layer

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

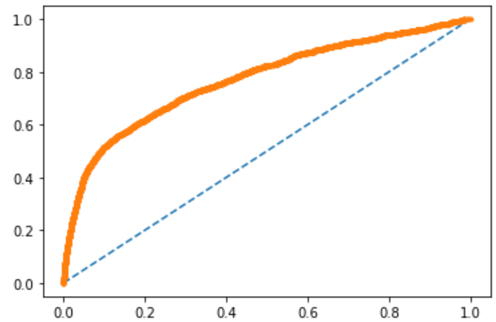
```
classifier_mlp = MLPClassifier(hidden_layer_sizes=(13,13 ) ,max_iter=500)
finalModel_mlp = Convert_Model(X_train,y_train,X_test,y_test,classifier_mlp)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A column-vector y w
  y = column_or_1d(y, warn=True)
0.7058181818181818
[[9325 2641]
 [4640 8144]]
0.7058181818181818
0.7551228558182661
0.6370463078848561
f1 score 0.6910772625058339
```

```
# roc curve and auc on imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
probs = finalModel_mlp.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
```

```
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

AUC: 0.772



MLP Classifier with 1 layer

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

```
classifier_mlp = MLPClassifier(hidden_layer_sizes=(13 ) ,max_iter=500)
finalModel_mlp = Convert_Model(X_train,y_train,X_test,y_test,classifier_mlp)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A column-vector y w
  y = column_or_1d(y, warn=True)
0.7000808080808081
[[9423 2543]
 [4880 7904]]
0.7000808080808081
0.7565808366038097
0.6182728410513142
f1 score 0.68047006155568
```

```
# roc curve and auc on imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
probs = finalModel_mlp.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 0.762
```


```
#by balcing y output
# After standardization our f1 score and auc percentage increases
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["MODEL", "ACCURACY_score","precision_score","Recall_score","F1 score","AUC"]
x.add_row(["Dummy classifer",0.50, 0.50,1,0.66,"NAN"])
x.add_row(["Logistic Regression)", 0.73, 0.80,0.62,0.70,0.78])
x.add_row(["Random Forest",0.65, 0.85,0.38,0.52,0.766])
x.add_row(["SVM classifier",0.73, 0.82,0.60,0.69,0.73])
x.add_row(["XGB boost",0.74, 0.81,0.63,0.71,0.798])
x.add_row(["MLP  classifier with 3 layers",0.70, 0.74,0.61,0.67,0.745])
x.add_row(["MLP classifier with 2 layers",0.70, 0.75,0.61,0.68,0.76])
x.add_row(["MLP classifier 1 layers",0.72, 0.78,0.62,0.693,0.766])

print('Bank Marketing')
print(x)
```

Bank Marketing

MODEL	ACCURACY_score	precision_score	Recall_score	F1 score	AUC
Dummy classifer	0.5	0.5	1	0.66	NAN
Logistic Regression)	0.73	0.8	0.62	0.7	0.78
Random Forest	0.65	0.85	0.38	0.52	0.766
SVM classifier	0.73	0.82	0.6	0.69	0.73
XGB boost	0.74	0.81	0.63	0.71	0.798
MLP classifier with 3 layers	0.7	0.74	0.61	0.67	0.745
MLP classifier with 2 layers	0.7	0.75	0.61	0.68	0.76
MLP classifier 1 layers	0.72	0.78	0.62	0.693	0.766

✓ 0s completed at 10:56 AM