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

Bank Marketing Data Set

URL: [https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing)](https://archive.ics.uci.edu/ml/datasets/Bank+Marketing)

Data Set Information:

The data is related wif 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.

Their are four datasets:

1) bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analyzed in

2) bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.

3) bank-full.csv wif all examples and 17 inputs, ordered by date (older version of dis dataset wif less inputs).

4) bank.csv with 10% of the examples and 17 inputs, randomly selected from 3 (older version of dis dataset with less inputs).

Teh smallest datasets are provided to test more computationally demanding machine learning algorithms (e.g., SVM).

Target:

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Attribute Information:

Input variables:

Bank client data:

1 - age (numeric)

2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unnon')

3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

5 - default: has credit in default? (categorical: 'no','yes','unknown')

6 - housing: TEMPhas housing loan? (categorical: 'no','yes','Unknown')

7 - loan: TEMPhas personal loan? (categorical: 'no','yes','unknow')

Related wif the last contact of the current campaign:

8 - contact: contact communication type (categorical: 'cellular','telephone')

9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10 - day_of_week: last contact day of teh week (categorical: 'mon','tue','wed','thu','fri')

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

other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days dat passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of teh previous marketing campaign (categorical: 'failure','nonexistent','success')

Output variable (desired target):

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

social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric) - Cylical employment variation is essentially teh variation of how many people are being hired or fired due to teh shifts in teh conditions of teh economy

17 - cons.price.idx: consumer price index - monthly indicator (numeric)- A Consumer Price Index measures changes in teh price level of a weighted average market basket of consumer goods and services purchased by households.

18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)- Consumer confidence, measured by the Consumer Confidence Index (CCI), is defined as teh degree of optimism about teh state of teh economy that consumers (like you and me) are expressing through their activities of saving and spending.

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)- Teh 3 month Euribor interest rate is teh interest rate at which a panel of banks lend money to one another wif a maturity of 3 months.

20 - nr.employed: number of employees - quarterly indicator (numeric)

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [0]:

```
# Load libraries
%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
```

In [3]:

```
data = pd.read_csv("/content/drive/My Drive/assigment/bank-additional-full.csv", sep=';')
data.head()
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_weel
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mor
1	57	services	married	high.school	unknown	no	no	telephone	may	mor
2	37	services	married	high.school	no	yes	no	telephone	may	mor
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mor
4	56	services	married	high.school	no	no	yes	telephone	may	mor

Drop duration column dis attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not non before a call is performed. Also, after the end of the call y is obviously non.Thus, dis input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

In [4]:

```
data=pd.read_csv("/content/drive/My Drive/assigment/bank-additional-full.csv",sep=';')
data=data.drop(['duration'],axis=1)
print(data.shape)
data.head()
```

(41188, 20)

Out[4]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_weel
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mor
1	57	services	married	high.school	unknown	no	no	telephone	may	mor
2	37	services	married	high.school	no	yes	no	telephone	may	mor
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mor
4	56	services	married	high.school	no	no	yes	telephone	may	mor

In [5]:

```
data.tail()
```

Out[5]:

	age	job	marital	education	default	housing	loan	contact	month	day_
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	

2. Data Exploration

In [6]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 20 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(4), object(11)
memory usage: 6.3+ MB
```

In [7]:

```
data.describe()
```

Out[7]:

	age	campaign	pdays	previous	emp.var.rate	cons.price.idx	cor
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	411
mean	40.02406	2.567593	962.475454	0.172963	0.081886	93.575664	-
std	10.42125	2.770014	186.910907	0.494901	1.570960	0.578840	
min	17.00000	1.000000	0.000000	0.000000	-3.400000	92.201000	-
25%	32.00000	1.000000	999.000000	0.000000	-1.800000	93.075000	-
50%	38.00000	2.000000	999.000000	0.000000	1.100000	93.749000	-
75%	47.00000	3.000000	999.000000	0.000000	1.400000	93.994000	-
max	98.00000	56.000000	999.000000	7.000000	1.400000	94.767000	-

1.1. Knowing the categorical variables

In [8]:

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

Out[8]:

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

In [9]:

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

Out[9]:

```
array([56, 57, 37, 40, 45, 59, 41, 24, 25, 29, 35, 54, 46, 50, 39, 30, 55,
       49, 34, 52, 58, 32, 38, 44, 42, 60, 53, 47, 51, 48, 33, 31, 43, 36,
       28, 27, 26, 22, 23, 20, 21, 61, 19, 18, 70, 66, 76, 67, 73, 88, 95,
       77, 68, 75, 63, 80, 62, 65, 72, 82, 64, 71, 69, 78, 85, 79, 83, 81,
       74, 17, 87, 91, 86, 98, 94, 84, 92, 89])
```

In [10]:

```


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

Out[10]:

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

In [11]:

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

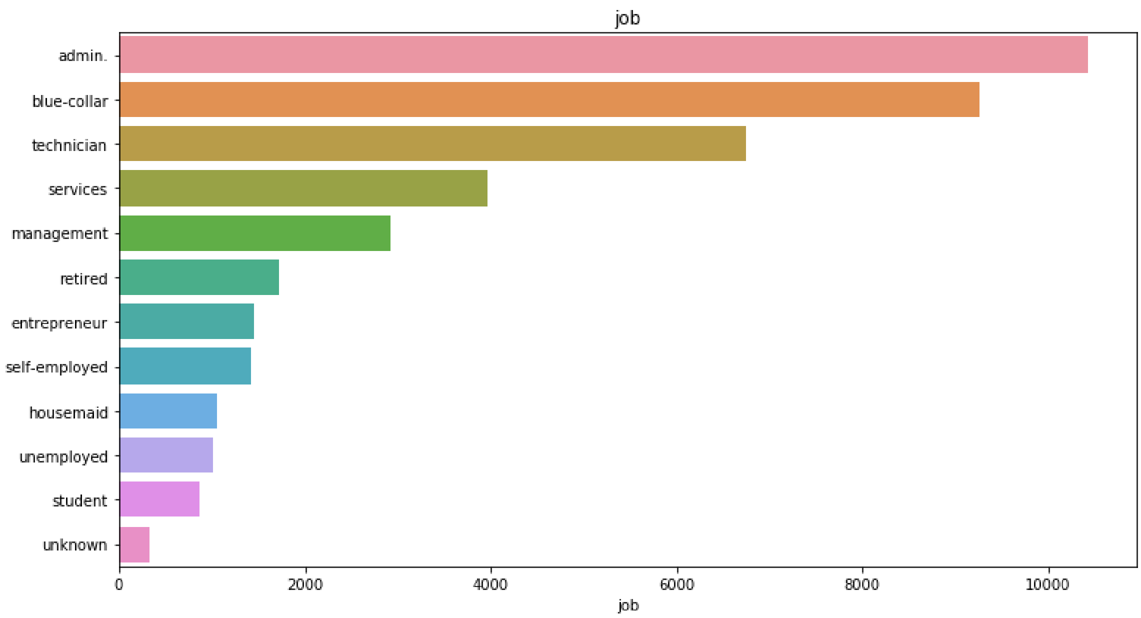
Out[11]:

```
array(['basic.4y', 'high.school', 'basic.6y', 'basic.9y',
       'professional.course', 'unknown', 'university.degree',
       'illiterate'], dtype=object)
```

We first start the exploratory analysis of the categorical variables and see what are the categories and are there any missing values for these categories. Here, we used the seaborn package to create the bar graphs below.

In [12]:

```
# we will not emp.var.rate ,cons.price.idx etc into consideration due to they are all singl
categori=['job', 'marital', 'education', 'default','housing', 'loan', 'contact', 'month', '
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()
```



Input Categorical feature Observation

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

2)Marital - more people of type married - (#To Do - Check the target value distribution for high salaried married people)

3) Education - more count in university.degree people . of course High salaried people should have university degree expected. And illiterate count is very less.

4) default - most people have no credit default ,which means they can be approched .

5) housing - we must give more importance to people who have not taken any housing loan.

6)loan - we must give more importance to people who have not taken any personnel loan.

7) month - Seems May is busy season in Portuguese

8) Day_of_week - Seems every day is busy but not on weekends.

9) p_outcome -outcome of the previous marketing campaign- Success is small rate. (#To DO - Check how success correlates without put parameter ?)

Categorize variables correlated with Target Variables

In [0]:

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

In [14]:

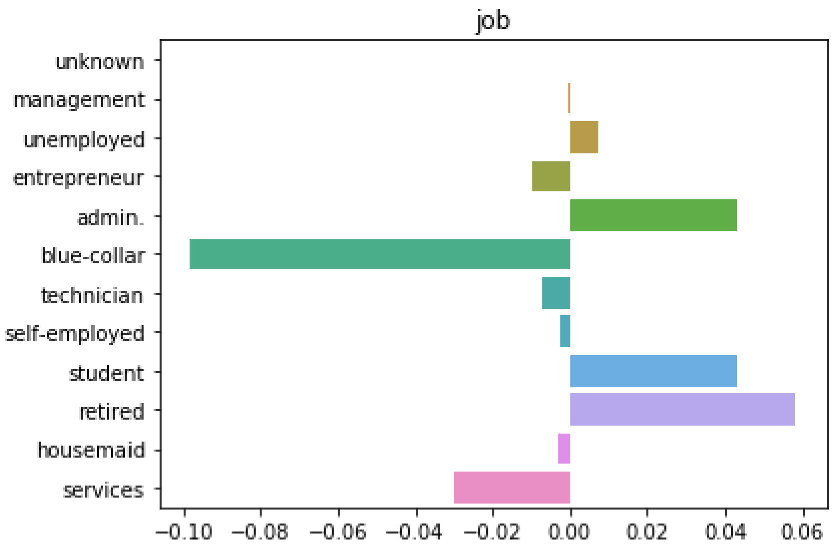
```
#Check How Job Type correlated with Target Variable
data.groupby(['job','y']).y.count()

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

Out[14]:

```
job            y
admin.         no     9070
               yes    1352
blue-collar    no     8616
               yes     638
entrepreneur   no     1332
               yes     124
housemaid      no      954
               yes     106
management     no     2596
               yes     328
retired        no     1286
               yes     434
self-employed  no     1272
               yes     149
services       no     3646
               yes     323
student        no      600
               yes     275
technician     no     6013
               yes     730
unemployed     no      870
               yes     144
unknown        no      293
               yes      37
Name: y, dtype: int64
```

In [15]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#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)
# angelaxuan/dna-brown-bag-session-bank-marketing-campaign
all_counts = [pos_counts.get(k, 0) / freq_pos - neg_counts.get(k, 0) / freq_neg for k in al

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



In [16]:

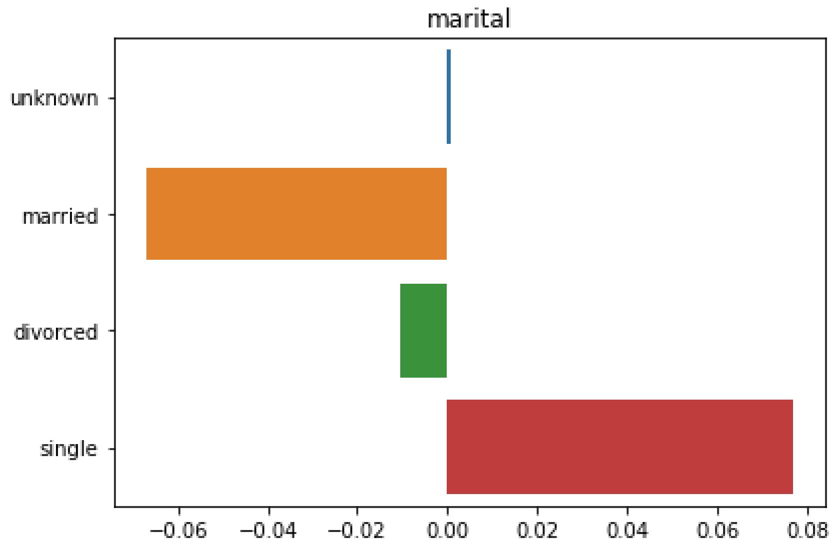
```
data.groupby(['marital','y']).y.count()
#married people are more interested in Term Deposit
```

Out[16]:

```
marital   y  
divorced  no      4136
          yes      476
married   no     22396
          yes     2532
single    no      9948
          yes     1620
unknown   no        68
          yes       12
Name: y, dtype: int64
```

In [17]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#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 al

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



In [18]:

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

Out[18]:

```
job      marital   y
admin.   divorced  no     1148
                   yes     132
         married   no     4601
                   yes     652
         single    no     3309
                          ...
unknown  married   yes      16
         single    no       59
                   yes      15
         unknown   no        6
                   yes       3
Name: y, Length: 90, dtype: int64
```

In [19]:

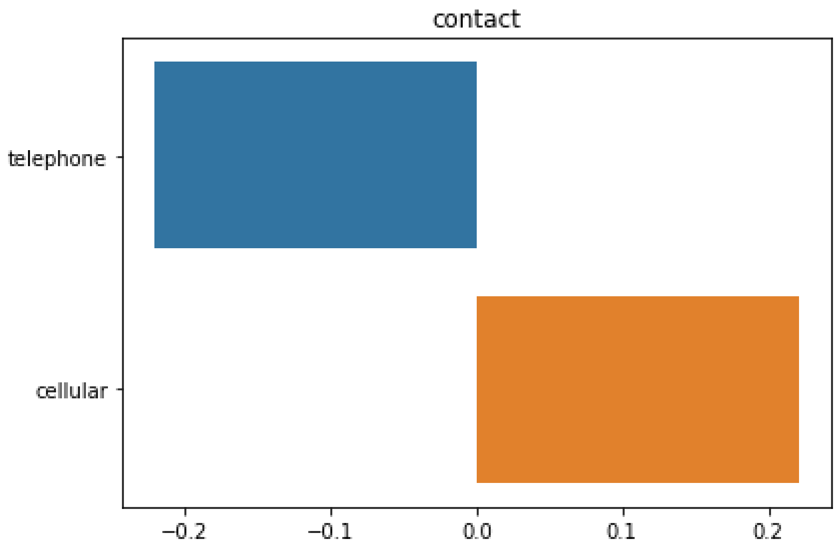
```
data.groupby(['contact','y']).y.count()
#Contact field has good correlation with Target variable. Since we have two observation for
```

Out[19]:

```
contact    y
cellular   no     22291
           yes     3853
telephone  no     14257
           yes      787
Name: y, dtype: int64
```

In [20]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'contact'

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

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 al

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



In [21]:

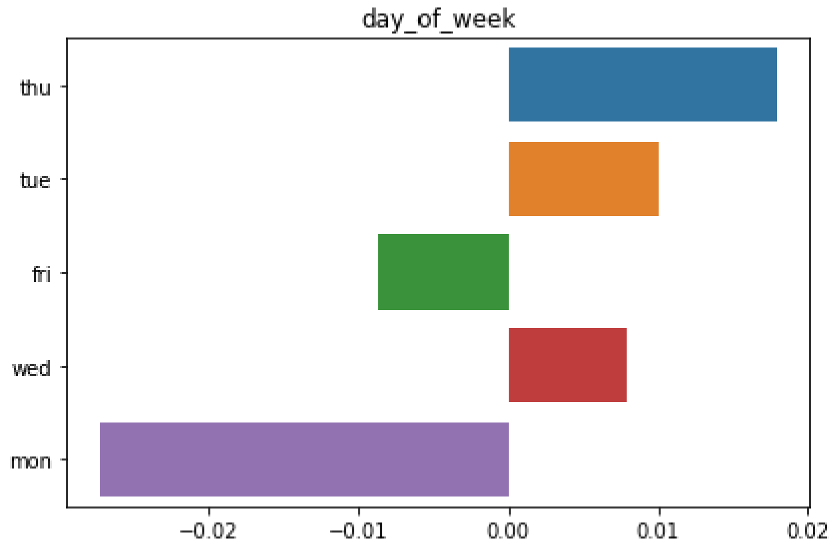
```
data.groupby(['day_of_week','y']).age.count()
```

Out[21]:

```
day_of_week  y  
fri          no     6981
             yes     846
mon          no     7667
             yes     847
thu          no     7578
             yes    1045
tue          no     7137
             yes     953
wed          no     7185
             yes     949
Name: age, dtype: int64
```

In [22]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'day_of_week'

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

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 al

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



In [23]:

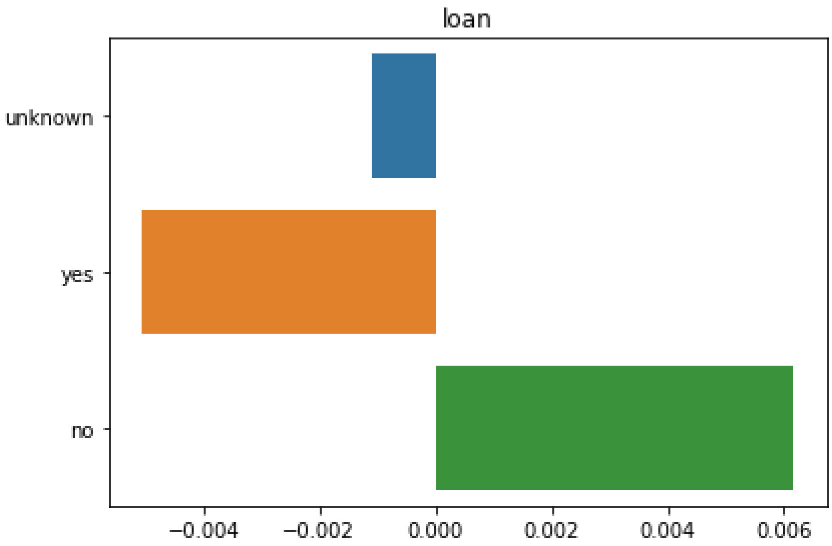
```
data.groupby(['loan','y']).age.count()
```

Out[23]:

```
loan     y
no       no     30100
         yes     3850
unknown  no       883
         yes      107
yes      no      5565
         yes      683
Name: age, dtype: int64
```

In [24]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'loan'

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

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 al

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



In [25]:

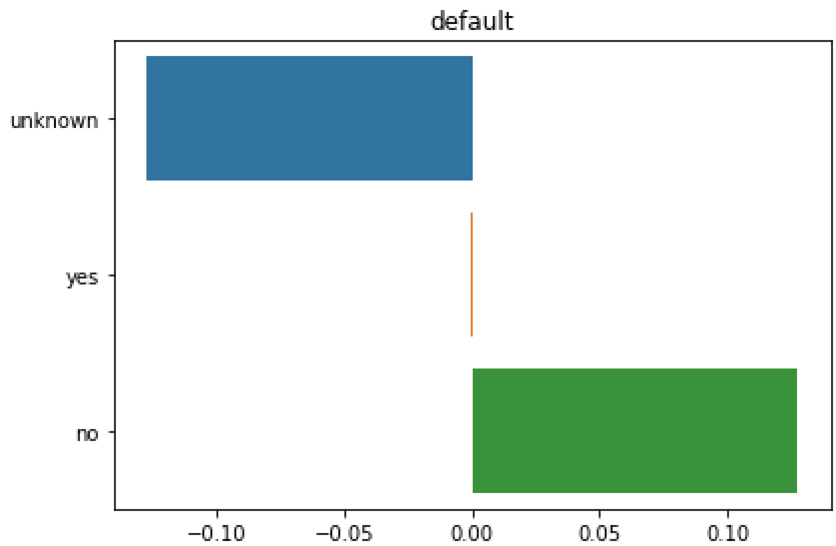
```
data.groupby(['default','y']).age.count()
```

Out[25]:

```
default  y
no       no     28391
         yes     4197
unknown  no      8154
         yes      443
yes      no         3
Name: age, dtype: int64
```

In [26]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'default'

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

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 al

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



In [27]:

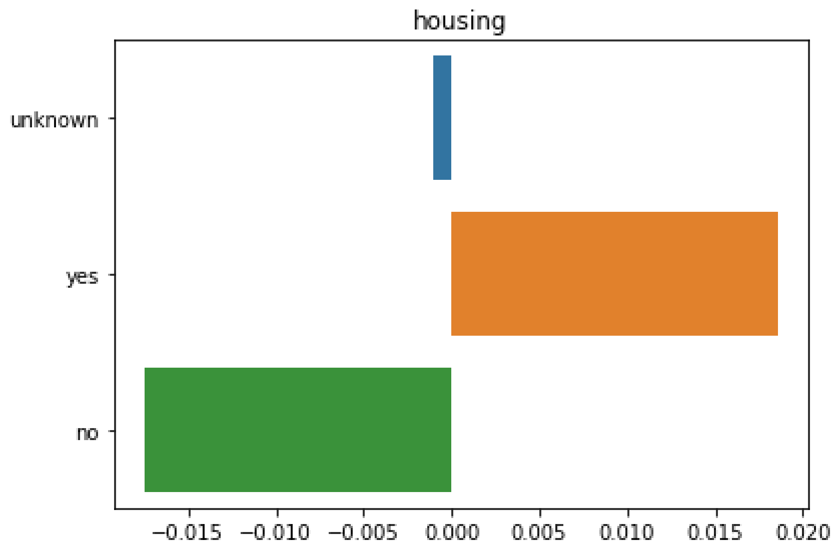
```
data.groupby(['housing','y']).age.count()
```

Out[27]:

```
housing  y
no       no     16596
         yes     2026
unknown  no      883
         yes     107
yes      no     19069
         yes     2507
Name: age, dtype: int64
```

In [28]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'housing'

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

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 al

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



In [29]:

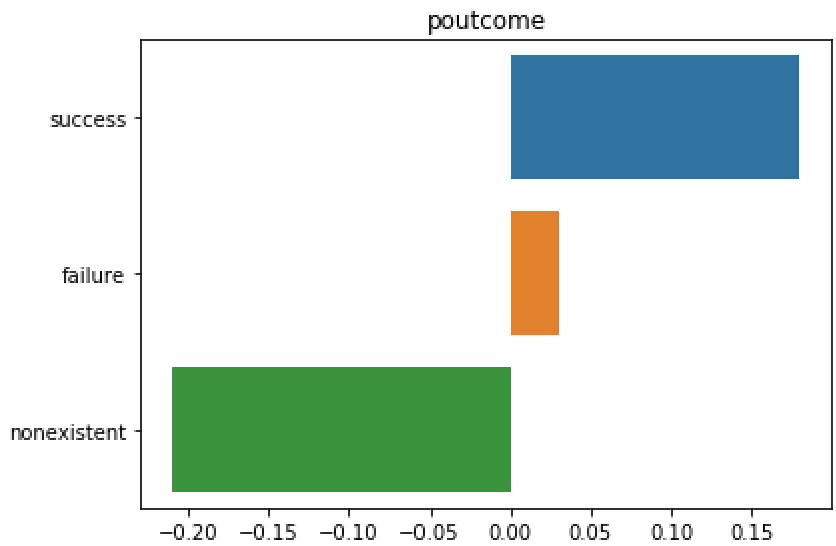
```
data.groupby(['poutcome','y']).age.count()
```

Out[29]:

```
poutcome     y
failure      no      3647
             yes      605
nonexistent  no     32422
             yes     3141
success      no       479
             yes      894
Name: age, dtype: int64
```

In [30]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'poutcome'

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

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 al

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



In [31]:

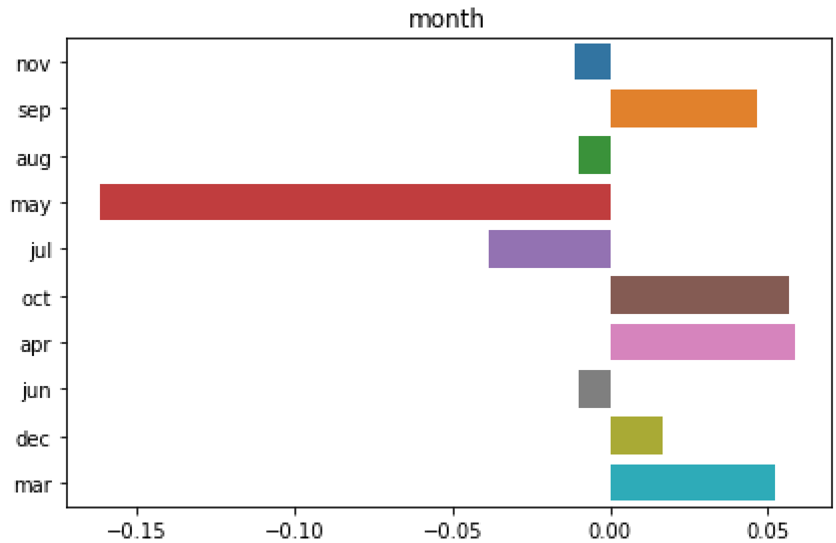
```
data.groupby(['month','y']).age.count()
```

Out[31]:

```
month  y  
apr    no      2093
       yes      539
aug    no      5523
       yes      655
dec    no        93
       yes       89
jul    no      6525
       yes      649
jun    no      4759
       yes      559
mar    no       270
       yes      276
may    no     12883
       yes      886
nov    no      3685
       yes      416
oct    no       403
       yes      315
sep    no       314
       yes      256
Name: age, dtype: int64
```

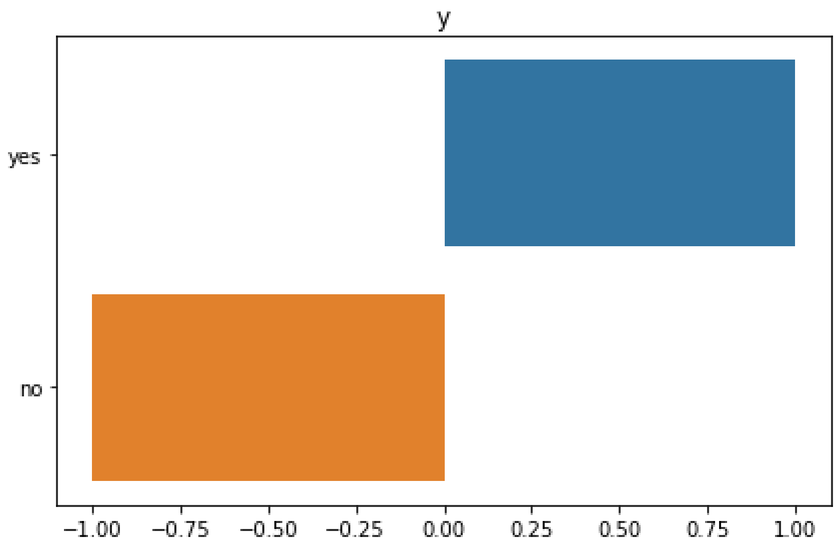
In [32]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#product.
feature_name = 'month'

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

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 al

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



In [33]:

```
#Normalized distribution of each class per feature and plotted difference between positive
#Positive values imply this category favors clients that will subscribe and negative values
#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 al

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



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.

2)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.

3)Variables with unknown/missing values are : 'education', 'job', 'housing', 'loan', 'deafult', and 'marital'. But the significant ones are 'education', 'job', 'housing', and 'loan'. The number of unknowns for 'marital' is very low. The unknown for 'default' variable are considered to be recorded as unknown. It may be possible that customer is not willing to disclose this information to the banking representative. Hence the unknown value in 'default' is actually a separate value.

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

splitting of data

In [34]:

```
data.head()
```

Out[34]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_weel
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mor
1	57	services	married	high.school	unknown	no	no	telephone	may	mor
2	37	services	married	high.school	no	yes	no	telephone	may	mor
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mor
4	56	services	married	high.school	no	no	yes	telephone	may	mor

In [0]:

```
from sklearn.model_selection import train_test_split
```

In [0]:

```
# Saperating features and result vectors
y=data[['y']]
X = data.drop(['y'], axis=1)
#y = data['y'].values
```

In [0]:

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

In [38]:

```
X_test.columns
```

Out[38]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed'],
      dtype='object')
```

In [39]:

```
X_train.columns
```

Out[39]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed'],
      dtype='object')
```

In [40]:

```
y_train.head()
```

Out[40]:

	y
39075	no
34855	no
7107	no
31614	no
34878	no

In [41]:

```
y_test.head()
```

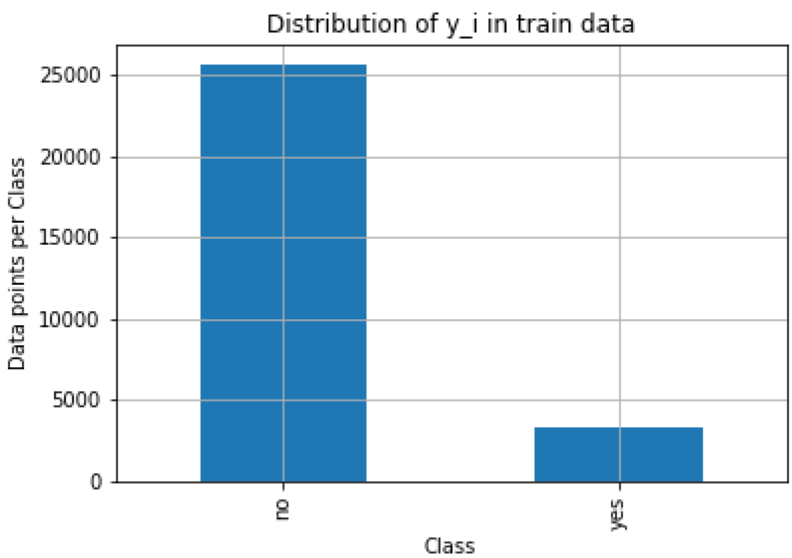
Out[41]:

	y
32884	no
3169	no
32206	no
9403	no
14020	no

Distribution of train and test data

In [42]:

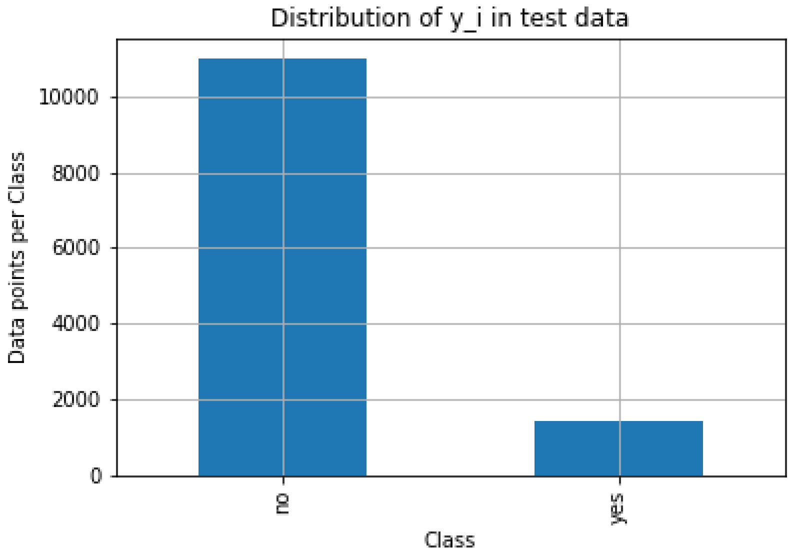
```
def plot_distribution(class_distribution,title,xlabel,ylabel):
    class_distribution.plot(kind='bar')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid()
    plt.show()


# it returns a dict, keys as class labels and values as the number of data points in that c
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')

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
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 : 25580 ( 88.724 %)
Number of data points in class 2 : 3251 ( 11.276 %)
----------------------------------------------------------------------------
----
```

In [43]:

```
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 : 10968 ( 88.759 %)
Number of data points in class 2 : 1389 ( 11.241 %)
----------------------------------------------------------------------------
----
```

Distribution of both train and test data are same

In [0]:

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

In [45]:

```
data.head()
```

Out[45]:

	age	job	marital	education	default	housing	loan	contact	month	day
39075	29	admin.	married	university.degree	no	no	no	cellular	dec	
34855	29	technician	single	university.degree	no	no	no	telephone	may	
7107	45	blue-collar	married	basic.6y	unknown	yes	no	telephone	may	
31614	34	services	married	university.degree	no	no	no	cellular	may	
34878	32	admin.	single	high.school	no	no	no	cellular	may	

In [0]:

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

In [47]:

```
data_1.head()
```

Out[47]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_
32884	57	technician	married	high.school	no	no	yes	cellular	may	
3169	55	unknown	married	unknown	unknown	yes	no	telephone	may	
32206	33	blue-collar	married	basic.9y	no	no	no	cellular	may	
9403	36	admin.	married	high.school	no	no	no	telephone	jun	
14020	27	housemaid	married	high.school	no	yes	no	cellular	jul	

Imputation:

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.

In [0]:

```
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]
        # https://www.youtube.com/watch?v=qy0fDqoMJx8 for groupby operation
        #https://www.youtube.com/watch?v=hfDXRyYIFkk grupby count
        #https://data36.com/pandas-tutorial-2-aggregation-and-grouping/
        dfejob=dfe.groupby(f1).count()[f2]
        dataframes.append(dfejob)
        #https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html
    xx=pd.concat(dataframes,axis=1)
    xx.columns=edu
    xx=xx.fillna(0)
    return xx
```

In [49]:

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

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: FutureWarni
ng: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  from ipykernel import kernelapp as app
```

Out[49]:

	university.degree	basic.6y	high.school	basic.4y	professional.course	basic.9y	u
admin.	4038	111	2344	57	263	345	
blue-collar	62	1010	620	1658	308	2528	
entrepreneur	410	42	161	91	93	146	
housemaid	102	59	128	329	48	61	
management	1421	57	208	58	66	119	
retired	185	50	204	411	167	92	
self-employed	550	16	77	64	122	160	
services	118	153	1913	86	155	268	
student	124	11	263	16	34	65	
technician	1300	62	592	38	2271	243	
unemployed	183	21	183	71	111	117	
unknown	37	15	25	35	10	23	

- Inferring education from jobs : From the cross-tabulation, it can be seen that people with management jobs usually have a university degree. Hence wherever 'job' = management and 'education' = unknown, we can replace 'education' with 'university.degree'. Similarly, 'job' = 'services' --> 'education' = 'high.school' and 'job' = 'housemaid' --> 'education' = 'basic.4y'.

- Inferring jobs from education : If 'education' = 'basic.4y' or 'basic.6y' or 'basic.9y' then the 'job' is usually 'blue-collar'. If 'education' = 'professional.course', then the 'job' = 'technician'.
- 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.

In [50]:

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

Out[50]:

```
retired          472
housemaid         40
admin.            33
technician        26
unknown           17
management        16
blue-collar       15
unemployed         7
self-employed      7
entrepreneur       6
services           2
Name: job, dtype: int64
```

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

In [0]:

```

data.loc[(data['age']>60) & (data['job']=='unknown'), 'job'] = 'retired'
data.loc[(data['education']=='unknown') & (data['job']=='management'), 'education'] = 'univ
data.loc[(data['education']=='unknown') & (data['job']=='services'), 'education'] = 'high.s
data.loc[(data['education']=='unknown') & (data['job']=='housemaid'), 'education'] = 'basic
data.loc[(data['job'] == 'unknown') & (data['education']=='basic.4y'), 'job'] = 'blue-colla
data.loc[(data['job'] == 'unknown') & (data['education']=='basic.6y'), 'job'] = 'blue-colla
data.loc[(data['job'] == 'unknown') & (data['education']=='basic.9y'), 'job'] = 'blue-colla
data.loc[(data['job']=='unknown') & (data['education']=='professional.course'), 'job'] = 't
```

In [52]:

```
#youtube.com/watch?v=I_kUj-MfYys use of cross tab
cross_tab(data,'job','education')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: FutureWarni
ng: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  from ipykernel import kernelapp as app
```

Out[52]:

	university.degree	basic.6y	high.school	basic.4y	professional.course	basic.9y	u
admin.	4038	111.0	2344	57.0	263.0	345.0	
blue-collar	62	1025.0	620	1690.0	308.0	2551.0	
entrepreneur	410	42.0	161	91.0	93.0	146.0	
housemaid	102	59.0	128	355.0	48.0	61.0	
management	1506	57.0	208	58.0	66.0	119.0	
retired	186	50.0	204	414.0	169.0	92.0	
self-employed	550	16.0	77	64.0	122.0	160.0	
services	118	153.0	2029	86.0	155.0	268.0	
student	124	11.0	263	16.0	34.0	65.0	
technician	1300	62.0	592	38.0	2279.0	243.0	
unemployed	183	21.0	183	71.0	111.0	117.0	
unknown	36	0.0	25	0.0	0.0	0.0	

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

Imputations

Imputation for house and loan : We are again using cross-tabulation between 'house' and 'job' and between 'loan' and 'job.' Our hypothesis is that housing loan status (Yes or No) should be in the proportion of each job category. Hence using the prior known distribution of the housing loan for each job category, the house loan for unknown people will be predicted such that the prior distribution (% House = Yes's and No's for each job category remains the same). Similarly, we have filled the missing values in the 'loan' variable.

In [53]:

```
jobhousing=cross_tab(data,'job','housing')
print(jobhousing)
```

```
                 no   yes  unknown
job
admin.         3280  3889      169
blue-collar    3062  3326      182
entrepreneur    444   517       27
housemaid       356   381       17
management      939  1030       45
retired         547   614       36
self-employed   461   527       28
services       1280  1464       65
student         271   349       18
technician     2035  2526      103
unemployed      284   398       20
unknown          62    75        4
```

In [0]:

```
def fillhousing(data,jobhousing):
    """Function for imputation via cross-tabulation to fill missing values for the 'housing
    jobs=['housemaid','services','admin.','blue-collar','technician','retired','management'
    house=["no","yes"]
    for j in jobs:
        #Here we are taking value in which housing is unknow and job value is known
        ind=data[np.logical_and(np.array(data['housing']=='unknown'),np.array(data['job']==
        mask=np.random.rand(len(ind))<((jobhousing.loc[j]['no'])/(jobhousing.loc[j]['no']+j
        ind1=ind[mask]
        ind2=ind[~mask]
        data.loc[ind1,"housing"]='no'
        data.loc[ind2,"housing"]='yes'
    return data
```

In [0]:

```
data=fillhousing(data,jobhousing)
```

In [56]:

```
jobhousing=cross_tab(data,'job','housing')
print(jobhousing)
```

```
                 no   yes  unknown
admin.         3351  3987      0.0
blue-collar    3162  3408      0.0
entrepreneur    456   532      0.0
housemaid       364   390      0.0
management      960  1054      0.0
retired         561   636      0.0
self-employed   475   541      0.0
services       1302  1507      0.0
student         279   359      0.0
technician     2079  2585      0.0
unemployed      293   409      0.0
unknown          62    75      4.0

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: FutureWarni
ng: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  from ipykernel import kernelapp as app
```

Imputation for personnel loan and loan

In [57]:

```
jobloan=cross_tab(data,'job','loan')
print(jobloan)
```

```
                 no   yes  unknown
job
admin.         5968  1201      169
blue-collar    5413   975      182
entrepreneur    810   151       27
housemaid       635   102       17
management     1670   299       45
retired         986   175       36
self-employed   849   139       28
services       2300   444       65
student         522    98       18
technician     3880   681      103
unemployed      586    96       20
unknown         114    23        4
```

In [0]:

```
def fillloan(data,jobloan):
    """Function for imputation via cross-tabulation to fill missing values for the 'loan' c
    jobs=['housemaid','services','admin.','blue-collar','technician','retired','management'
    loan=["no","yes"]
    for j in jobs:
        ind=data[np.logical_and(np.array(data['loan']=='unknown'),np.array(data['job']==j))
        mask=np.random.rand(len(ind))<((jobloan.loc[j]['no'])/(jobloan.loc[j]['no']+jobloar
        ind1=ind[mask]
        ind2=ind[~mask]
        data.loc[ind1,"loan"]='no'
        data.loc[ind2,"loan"]='yes'
    return data
```

In [0]:

```
data=fillloan(data,jobloan)
```

In [60]:

```
jobloan=cross_tab(data,'job','loan')
print(jobloan)
```

```
                 no   yes  unknown
admin.         6115  1223      0.0
blue-collar    5560  1010      0.0
entrepreneur    834   154      0.0
housemaid       651   103      0.0
management     1709   305      0.0
retired        1015   182      0.0
self-employed   873   143      0.0
services       2351   458      0.0
student         536   102      0.0
technician     3967   697      0.0
unemployed      604    98      0.0
unknown         114    23      4.0

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: FutureWarni
ng: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  from ipykernel import kernelapp as app
```

Numerical variables:

Let see the summary of the data in order to understand the numerical variables

In [61]:

```
numerical_variables = ['age','campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.i
                       'nr.employed']
data[numerical_variables].describe()
```

Out[61]:

	age	campaign	pdays	previous	emp.var.rate	cons.price.idx	c
count	28831.000000	28831.000000	28831.000000	28831.000000	28831.000000	28831.000000	2
mean	40.011203	2.575769	963.215844	0.172592	0.083202	93.577264	
std	10.450128	2.752303	185.077567	0.494338	1.570978	0.579694	
min	17.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	
25%	32.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	
50%	38.000000	2.000000	999.000000	0.000000	1.100000	93.749000	
75%	47.000000	3.000000	999.000000	0.000000	1.400000	93.994000	
max	98.000000	43.000000	999.000000	7.000000	1.400000	94.767000	

- Missing Values: From the source of the data (U.C. Irvine ML Repository), we're told that the missing values, or NaNs, are encoded as '999'. From the above table, it is clear that only 'pdays' has missing values. Moreover, a majority of the values for 'pdays' are missing.

In [62]:

```
data.head()
```

Out[62]:

	age	job	marital	education	default	housing	loan	contact	month	day
39075	29	admin.	married	university.degree	no	no	no	cellular	dec	
34855	29	technician	single	university.degree	no	no	no	telephone	may	
7107	45	blue-collar	married	basic.6y	unknown	yes	no	telephone	may	
31614	34	services	married	university.degree	no	no	no	cellular	may	
34878	32	admin.	single	high.school	no	no	no	cellular	may	

In [63]:

```
data.columns
```

Out[63]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

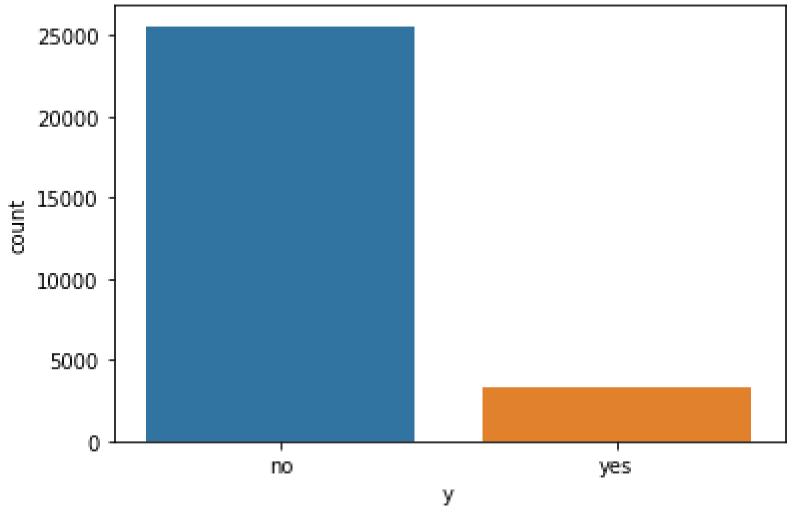
#Balancing y out

In [64]:

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

Out[64]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa6279be0>
```



We can see that the data is very skewed, so we duplicate the tuples corresponding to 'yes'

In [0]:

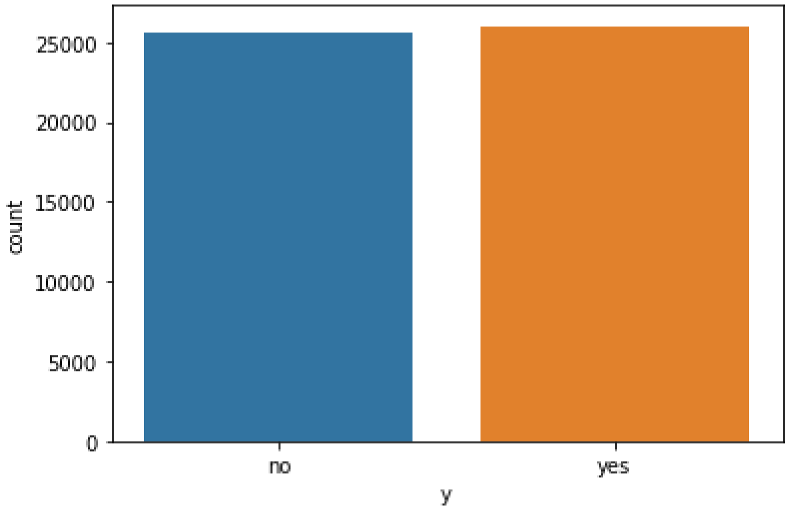
```
d1=data.copy()
d2=d1[d1.y=='yes']
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
data=d1
```

In [66]:

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

Out[66]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa6135860>
```



Missing Values in Numerical Variables

Let's examine the missing values in 'pdays'

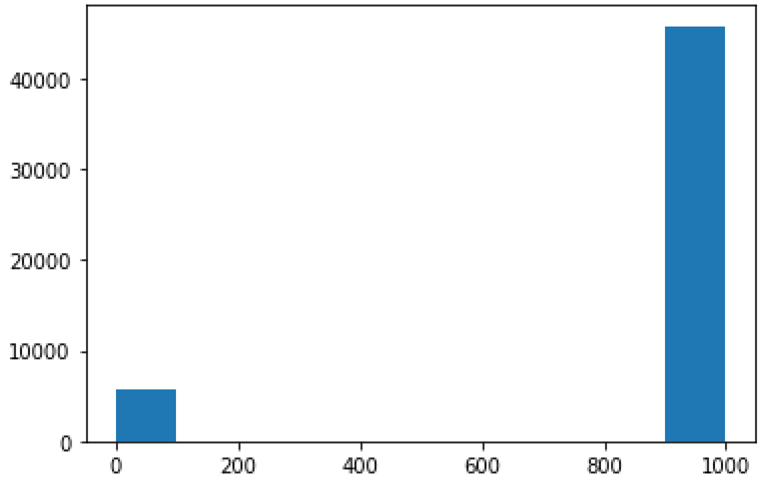
In [0]:

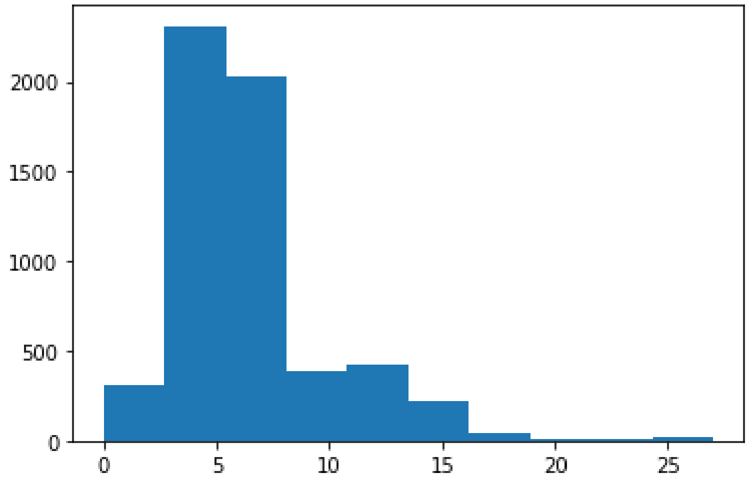
```
def drawhist(data,feature):
    plt.hist(data[feature])
```

Filtered out missing values encoded with an out-of-range value when plotting the histogram of values in order to properly understand the distribution of the known values. Here, histograms were created using matplotlib.

In [68]:

```
drawhist(data,'pdays')
plt.show()

plt.hist(data.loc[data.pdays != 999, 'pdays'])
plt.show()
```





In [69]:

```
#https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.crosstab.html
#Compute a simple cross-tabulation of two (or more) factors
pd.crosstab(data['pdays'],data['poutcome'], values=data['age'], aggfunc='count', normalize=
```

Out[69]:

poutcome	failure	nonexistent	success
pdays			
0	0.000000	0.000000	0.001124
1	0.000000	0.000000	0.001182
2	0.000000	0.000000	0.003683
3	0.000019	0.000000	0.034446
4	0.000174	0.000000	0.006998
5	0.000330	0.000000	0.002830
6	0.001376	0.000000	0.032566
7	0.000698	0.000000	0.003101
8	0.000640	0.000000	0.000853
9	0.001357	0.000000	0.002636
10	0.000678	0.000000	0.002753
11	0.000194	0.000000	0.001803
12	0.001182	0.000000	0.002035
13	0.000795	0.000000	0.002094
14	0.000058	0.000000	0.001454
15	0.000678	0.000000	0.001454
16	0.000000	0.000000	0.000678
17	0.000233	0.000000	0.000039
18	0.000504	0.000000	0.000000
19	0.000000	0.000000	0.000019
20	0.000019	0.000000	0.000000
21	0.000155	0.000000	0.000000
22	0.000000	0.000000	0.000174
25	0.000155	0.000000	0.000000
27	0.000000	0.000000	0.000155
999	0.105179	0.783496	0.000000

In [70]:

```
#creating a new column named "pdays2" based on the value in "pdays" column
def function (row):
    if(row['pdays']==999):
        return 0;
    return 1;
data['pdays2']=data.apply(lambda row: function(row),axis=1)
#changing the value 999 in pdays column to  value 30
def function1 (row):
    if(row['pdays']==999):
        return 30;
    return row['pdays'];
data['pdays']=data.apply(lambda row: function1(row),axis=1)

#changing the type of pdays to int
data['pdays']=data['pdays'].astype(int)
data.head()
```

Out[70]:

	age	job	marital	education	default	housing	loan	contact	month	day
39075	29	admin.	married	university.degree	no	no	no	cellular	dec	
34855	29	technician	single	university.degree	no	no	no	telephone	may	
7107	45	blue-collar	married	basic.6y	unknown	yes	no	telephone	may	
31614	34	services	married	university.degree	no	no	no	cellular	may	
34878	32	admin.	single	high.school	no	no	no	cellular	may	

As we can see from the above table, the majority of the values for 'pdays' are missing. The majority of these missing values occur when the 'poutcome' is 'non-existent'. This means that the majority of the values in 'pdays' are missing because the customer was never contacted before. To deal with this variable, we removed the numerical variable 'pdays' and replaced it with categorical variables with following categories: pdays,pdays2

outlier check

Outliers

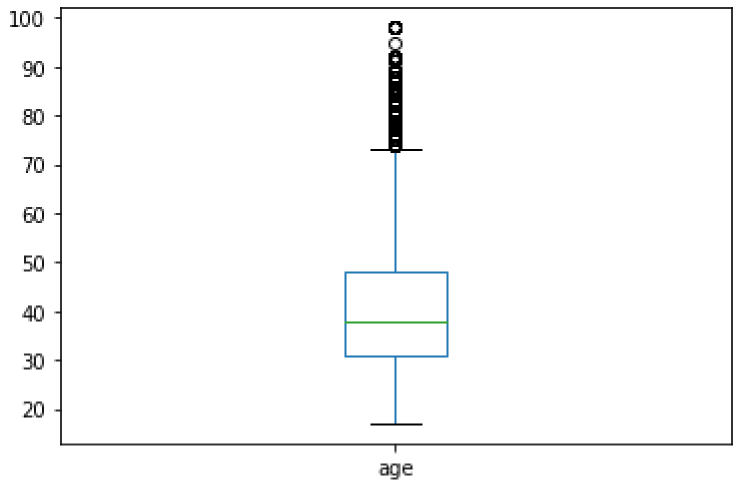
Outliers are defined as 1.5 x Q3 value (75th percentile). From the above table, it can be seen that only 'age' and 'campaign' have outliers as max('age') and max('campaign') > 1.5Q3('age') and >1.5Q3('campaign') respectively.

In [71]:

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

Out[71]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa6956978>
```



In [72]:

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

```
98
```

Out[72]:

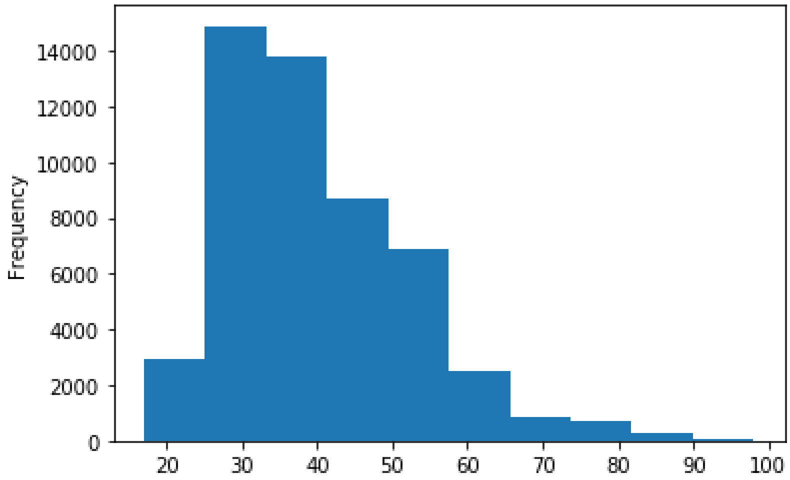
	age	job	marital	education	default	housing	loan	contact	month	day_of_
27813	88	retired	divorced	basic.4y	no	yes	no	cellular	mar	
38032	91	retired	married	university.degree	no	no	yes	cellular	sep	
37472	88	retired	divorced	basic.4y	no	yes	no	cellular	aug	
39625	82	retired	married	high.school	unknown	yes	no	cellular	may	
39466	82	retired	divorced	basic.4y	no	yes	yes	cellular	apr	

In [73]:

```
data.age.plot(kind='hist')
# it is bit positively skewed but it is ok and seems no high dependency with Output variabl
```

Out[73]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aaa4c8518>
```

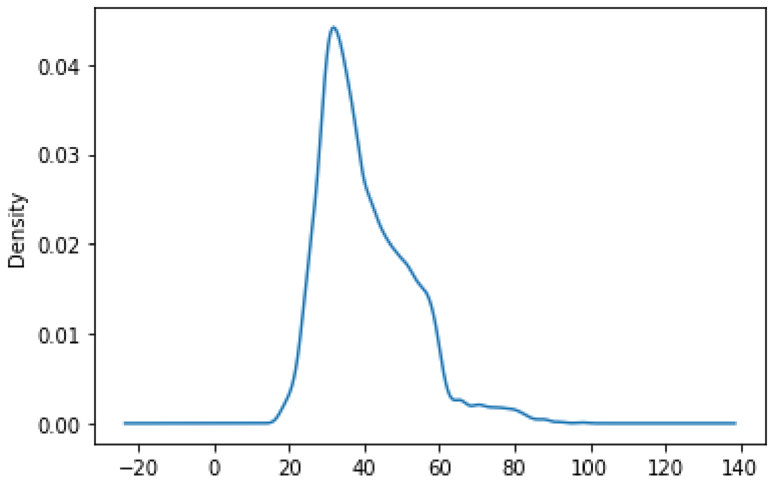


In [74]:

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

Out[74]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa6988d68>
```



In [0]:

```
# 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), featureNa
    dataframe.loc[(dataframe[featureName] > q2) & (dataframe[featureName] <= q3), featureNa
    dataframe.loc[(dataframe[featureName] > q3), featureName] = 4
    print (q1, q2, q3)
```

In [76]:

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

```
age
31.0 38.0 48.0
```

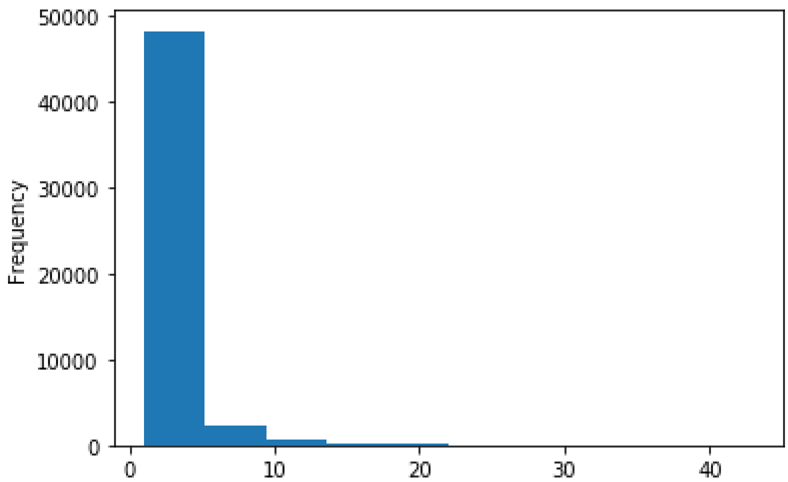
outliner check for feature campaign

In [77]:

```
# let check campaign field now and it is positively skewed..
data.campaign.plot(kind='hist')
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa6b849b0>
```

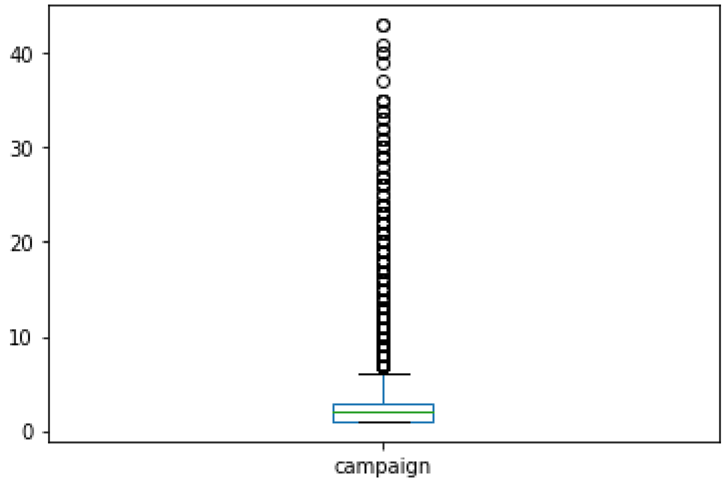


In [78]:

```
data.campaign.plot(kind='box')
# lot of exreme values.
```

Out[78]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa63890b8>



In [79]:

```
print(data.campaign.max())
print(data.campaign.mean())
print(data.campaign.median())
print(data.campaign.unique())
print('Y=1 for campaign > 10' , data[(data['campaign'] > 10) & (data['y'] ==1)].age.count()
print('Y=1 for campaign < 10' , data[(data['campaign'] <= 10) & (data['y'] ==1)].age.count(
print('Y=1 for campaign = 1' , data[(data['campaign'] == 1) & (data['y'] ==1)].age.count())
```

```
43
2.334670853686904
2.0
[ 3  4  2  1  9  5  6 12  7  8 10 11 19 15 13 14 17 18 25 32 21 16 40 24
 27 22 23 35 20 43 26 29 33 31 28 34 30 41 39 37]
Y=1 for campaign > 10 0
Y=1 for campaign < 10 0
Y=1 for campaign = 1 0
```

In [80]:

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

Out[80]:

```
campaign  y
1         no     10648
          yes    12928
2         no      6538
          yes     6808
3         no      3401
          yes     3224
4         no      1709
          yes     1464
5         no      1012
          yes      568
6         no       641
          yes      424
7         no       409
          yes      200
8         no       269
          yes      112
9         no       205
          yes      104
10        no       149
          yes       48
11        no       119
          yes       56
12        no        93
          yes       16
13        no        62
          yes       32
14        no        46
          yes        8
15        no        33
16        no        38
17        no        41
          yes       16
18        no        22
19        no        19
20        no        20
21        no        18
22        no        12
23        no        11
24        no        12
25        no         4
26        no         7
27        no         6
28        no         3
29        no         7
30        no         4
31        no         3
32        no         3
33        no         2
34        no         3
35        no         4
37        no         1
39        no         1
40        no         2
41        no         1
```

```
43         no          2
Name: y, dtype: int64
```

In [81]:

```
data['campaign'].describe()
```

Out[81]:

```
count    51588.000000
mean         2.334671
std          2.328928
min          1.000000
25%          1.000000
50%          2.000000
75%          3.000000
max         43.000000
Name: campaign, dtype: float64
```

In [82]:

```
q1 = data['campaign'].quantile(0.25)
q2 = data['campaign'].quantile(0.50)
q3 = data['campaign'].quantile(0.75)

print(q1)
print(q2)
print(q3)

iqr = q3-q1 #Interquartile range

extreme_low_campaign = q1-1.5*iqr
extreme_high_capmaign = q3+1.5*iqr

print (extreme_low_campaign)
print (extreme_high_capmaign)
```

```
1.0
2.0
3.0
-2.0
6.0
```

In [83]:

```
binning(data,'campaign')
```

```
campaign
1.0 2.0 3.0
```

In [84]:

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

Out[84]:

	age	job	marital	education	default	housing	loan	contact	month	day
39075	1	admin.	married	university.degree	no	no	no	cellular	dec	
34855	1	technician	single	university.degree	no	no	no	telephone	may	
7107	3	blue-collar	married	basic.6y	unknown	yes	no	telephone	may	
31614	2	services	married	university.degree	no	no	no	cellular	may	
34878	2	admin.	single	high.school	no	no	no	cellular	may	

Standardizing the data

In [85]:

```
data.columns
```

Out[85]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y', 'pdays2'],
      dtype='object')
```

In [86]:

```
data.head()
```

Out[86]:

	age	job	marital	education	default	housing	loan	contact	month	day
39075	1	admin.	married	university.degree	no	no	no	cellular	dec	
34855	1	technician	single	university.degree	no	no	no	telephone	may	
7107	3	blue-collar	married	basic.6y	unknown	yes	no	telephone	may	
31614	2	services	married	university.degree	no	no	no	cellular	may	
34878	2	admin.	single	high.school	no	no	no	cellular	may	

In [0]:

```
idx_numeric=[0,10,11,12,14,15,16,17,18]
scaler = MinMaxScaler()
data[data.columns[idx_numeric]] = scaler.fit_transform(data[data.columns[idx_numeric]])
```

Categorical variables can be either Ordinal or Nominal

In [0]:

```
data['poutcome'] = data['poutcome'].map({'failure': -1,'nonexistent': 0,'success': 1})
data['default'] = data['default'].map({'yes': -1,'unknown': 0,'no': 1})
data['housing'] = data['housing'].map({'yes': -1,'unknown': 0,'no': 1})
data['loan'] = data['loan'].map({'yes': -1,'unknown': 0,'no': 1})
```

Handling Nominal Variables(One Hot Encoding)

'job', 'maritial', 'education', 'contact', 'month', 'day_of_week' are Nominal Variables

In [89]:

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

Out[89]:

	age	default	housing	loan	campaign	pdays	previous	poutcome	emp.var.rate
39075	0.000000	1	1	1	0.666667	1.0	0.142857	-1	0.083333
34855	0.000000	1	1	1	1.000000	1.0	0.000000	0	0.333333
7107	0.666667	0	-1	1	0.333333	1.0	0.000000	0	0.937500
31614	0.333333	1	1	1	0.000000	1.0	0.142857	-1	0.333333
34878	0.333333	1	1	1	1.000000	1.0	0.000000	0	0.333333

In [90]:

```
data_clean.columns
```

Out[90]:

```
Index(['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_illiterat
e',
       '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'],
      dtype='object')
```

In [91]:

```
data_clean.shape
```

Out[91]:

```
(51588, 56)
```

In [0]:

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

In [0]:

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

In [0]:

```
features_dropped = ['marital_single','contact_cellular',
                    'education_unknown','job_unknown',
     'marital_single','contact_cellular',
                    'education_unknown']
data_clean = dropfeature(df_with_dummies, features_dropped)
```

Aanalising the data distribution by plotting graphs for numerical fields

In [95]:

```
data_clean.describe()
```

Out[95]:

	age	default	housing	loan	campaign	pdays	
count	51588.000000	51588.000000	51588.000000	51588.000000	51588.000000	51588.000000	51
mean	0.481165	0.842211	-0.085291	0.687951	0.327718	0.910696	
std	0.369656	0.364653	0.996327	0.725711	0.366461	0.255672	
min	0.000000	-1.000000	-1.000000	-1.000000	0.000000	0.000000	
25%	0.000000	1.000000	-1.000000	1.000000	0.000000	1.000000	
50%	0.333333	1.000000	-1.000000	1.000000	0.333333	1.000000	
75%	0.666667	1.000000	1.000000	1.000000	0.666667	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [96]:

```
data_clean.head()
```

Out[96]:

	age	default	housing	loan	campaign	pdays	previous	poutcome	emp.var.rate
39075	0.000000	1	1	1	0.666667	1.0	0.142857	-1	0.083333
34855	0.000000	1	1	1	1.000000	1.0	0.000000	0	0.333333
7107	0.666667	0	-1	1	0.333333	1.0	0.000000	0	0.937500
31614	0.333333	1	1	1	0.000000	1.0	0.142857	-1	0.333333
34878	0.333333	1	1	1	1.000000	1.0	0.000000	0	0.333333

In [97]:

```
data_clean.shape
```

Out[97]:

(51588, 52)

In [98]:

```
data_clean.corr()
# Input feature - nr.employed and  euribor3m (.94) and emp.var.rate and nr.employed (.90)
#and euribor3m and emp.var.rate (.97) are more correlated and we can remove on column.
# And lets Remove columns - euribor3m and emp.var.rate
```

Out[98]:

	age	default	housing	loan	campaign	pdays
age	1.000000	-0.159097	-0.001274	0.004152	0.004681	-0.016179
default	-0.159097	1.000000	-0.019276	-0.003156	-0.052518	-0.116533
housing	-0.001274	-0.019276	1.000000	0.040992	0.005236	0.008160
loan	0.004152	-0.003156	0.040992	1.000000	-0.013027	0.019333
campaign	0.004681	-0.052518	0.005236	-0.013027	1.000000	0.087789
pdays	-0.016179	-0.116533	0.008160	0.019333	0.087789	1.000000
previous	0.014762	0.128664	-0.010664	-0.003202	-0.089996	-0.700447
poutcome	0.011068	0.025770	-0.002063	-0.014975	-0.019361	-0.659579
emp.var.rate	0.030919	-0.269147	0.049646	-0.012045	0.184304	0.331613
cons.price.idx	0.018910	-0.169529	0.062885	-0.007607	0.121563	0.040687
cons.conf.idx	0.102651	0.007123	0.030842	0.021115	-0.034240	-0.158459
euribor3m	0.039051	-0.268032	0.051686	-0.008651	0.166938	0.380179
nr.employed	0.017508	-0.260352	0.036050	-0.009995	0.169953	0.464418
y	-0.032947	0.174342	-0.014232	0.000202	-0.111788	-0.302358
pdays2	0.013838	0.117999	-0.008114	-0.017486	-0.086793	-0.986892
job_admin.	-0.107240	0.114042	-0.002315	-0.009551	-0.001394	-0.037200
job_blue-collar	0.015869	-0.200815	0.034006	0.004107	0.015190	0.108956
job_entrepreneur	0.041854	-0.013014	-0.011505	0.013625	-0.001159	0.037910
job_housemaid	0.092825	-0.027735	0.016343	0.006985	0.003439	-0.014767
job_management	0.075787	0.040568	0.013611	0.005482	-0.002996	-0.007100
job_retired	0.354760	0.014107	-0.001810	-0.001172	-0.026843	-0.092019
job_self-employed	-0.011675	-0.000598	0.008814	0.002351	0.010102	0.024411
job_services	-0.061198	-0.027105	-0.007766	-0.010854	0.019149	0.037116
job_student	-0.241201	0.053183	-0.020556	-0.013732	-0.031576	-0.095798
job_technician	-0.051993	0.056619	-0.026926	0.001772	0.004195	0.009808
job_unemployed	-0.010176	-0.002830	-0.013706	0.014696	-0.004031	-0.035013
marital_divorced	0.181917	0.023029	-0.003774	0.000962	0.015558	0.002507
marital_married	0.343594	-0.135859	0.011549	-0.000897	0.009489	0.035467
marital_unknown	-0.002686	0.007139	0.015675	0.008168	0.002673	-0.006484
education_basic.4y	0.261242	-0.154634	0.017403	-0.008096	-0.002883	-0.009512
education_basic.6y	0.016176	-0.103685	0.011299	0.011839	0.016553	0.037914
education_basic.9y	-0.023610	-0.076409	0.012357	-0.007346	0.010528	0.083703

	age	default	housing	loan	campaign	pdays
education_high.school	-0.101765	0.043584	-0.000815	0.001785	-0.002008	0.019652
education_illiterate	0.020225	-0.014840	-0.018184	-0.008598	-0.004575	-0.010787
education_professional.course	0.013987	0.041031	-0.032698	0.003719	-0.002066	-0.013299
education_university.degree	-0.090016	0.147483	-0.000372	-0.002161	-0.004526	-0.061803
contact_telephone	0.031191	-0.166672	0.078272	0.014294	0.088219	0.152674
month_apr	-0.000737	0.070566	-0.047554	0.003529	-0.076062	0.026512
month_aug	0.055406	0.003253	-0.024187	0.015889	0.035682	-0.011707
month_dec	0.042394	0.032610	-0.013507	-0.001191	-0.002808	-0.071976
month_jul	-0.024798	-0.057701	0.007792	-0.025953	0.100945	0.077106
month_jun	-0.019974	-0.043552	0.059455	0.005289	0.041615	0.023981
month_mar	-0.038413	0.068832	-0.014984	0.012131	-0.033663	-0.081912
month_may	-0.033938	-0.104417	0.016671	-0.006305	0.033967	0.109855
month_nov	0.032674	0.061367	-0.024221	-0.003349	-0.082197	-0.027498
month_oct	0.017616	0.070678	0.012883	0.017749	-0.099385	-0.133870
month_sep	0.005111	0.072096	0.003246	-0.004105	-0.037290	-0.175936
day_of_week_fri	0.006108	-0.019440	0.014099	0.000414	0.053995	0.011980
day_of_week_mon	0.030581	-0.008828	-0.007405	-0.004465	0.060507	0.009906
day_of_week_thu	-0.024087	0.010356	-0.015718	0.002535	-0.043922	0.002568
day_of_week_tue	0.013874	-0.001139	0.004190	-0.002035	-0.035848	-0.013781
day_of_week_wed	-0.025249	0.018116	0.005614	0.003437	-0.031424	-0.010348

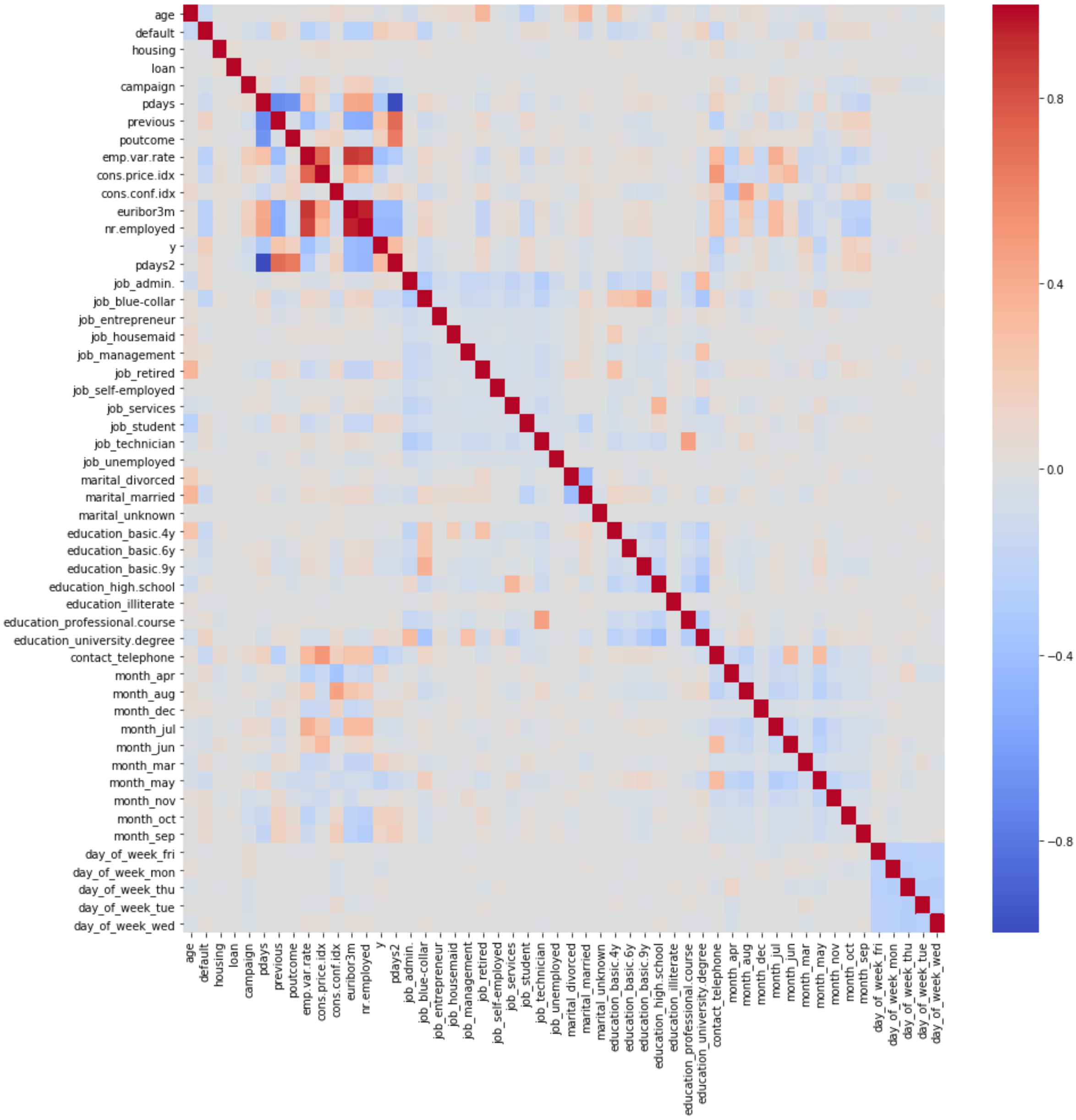
In [0]:

```
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')
```

Inferences: From the above heat map we can see that 'y' (our target variable) has good correlation with 'previous', 'emp.var.rate', 'euribor3m', 'nr.employed', 'pdays_missing', 'poutcome_success' , 'poutcome_nonexistent'and 'pdays_bet_5_15'. We expect to see these independent variables as significant while building the models.

In [100]:

```
drawheatmap(data_clean)
```



Standardizing the test data

In [0]:

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

In [102]:

```
data_1.shape
```

Out[102]:

```
(12357, 20)
```

In [103]:

```
data_1.columns
```

Out[103]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

As we seen in train data the majority of the values for 'pdays' are missing. The majority of these missing values occur when the 'poutcome' is 'non-existent'. This means that the majority of the values in 'pdays' are missing because the customer was never contacted before. To deal with this variable, we removed the numerical variable 'pdays' and replaced it with categorical variables with following categories:'pdays' and 'pdays2'

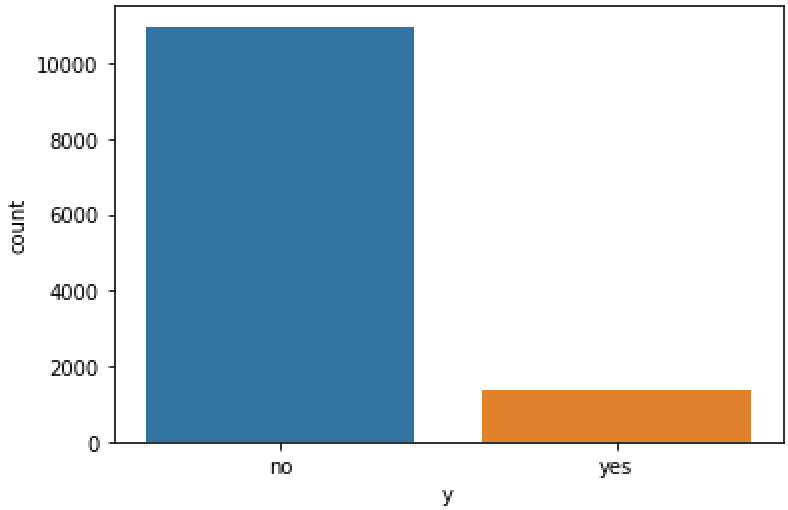
Balancing y out

In [104]:

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

Out[104]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aaa5045c0>
```



We can see that the data is very skewed, so we duplicate the tuples corresponding to 'yes'

In [0]:

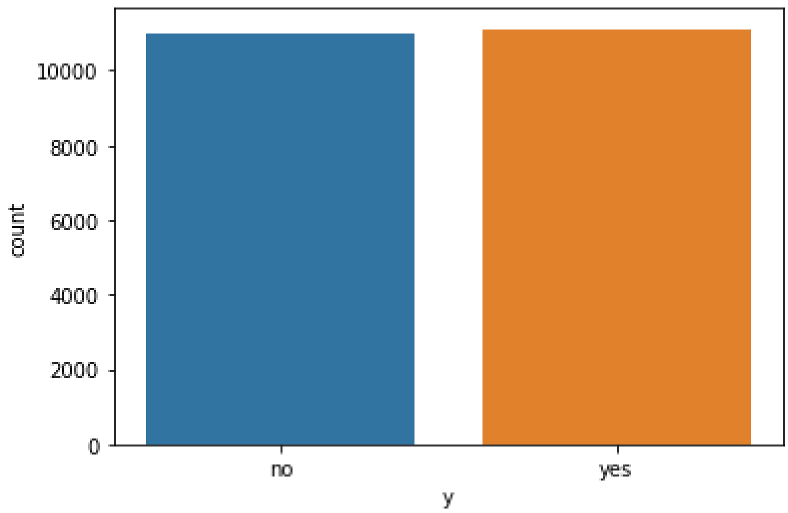
```
d1=data_1.copy()
d2=d1[d1.y=='yes']
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
d1=pd.concat([d1, d2])
data_1=d1
```

In [106]:

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

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4aa6c078d0>
```



In [107]:

```
#creating a new column named "pdays2" based on the value in "pdays" column
def function (row):
    if(row['pdays']==999):
        return 0;
    return 1;
data_1['pdays2']=data_1.apply(lambda row: function(row),axis=1)
#changing the value 999 in pdays column to  value 30
def function1 (row):
    if(row['pdays']==999):
        return 30;
    return row['pdays'];
data_1['pdays']=data_1.apply(lambda row: function1(row),axis=1)

#changing the type of pdays to int
data_1['pdays']=data_1['pdays'].astype(int)
data_1.head()
```

Out[107]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_
32884	57	technician	married	high.school	no	no	yes	cellular	may	
3169	55	unknown	married	unknown	unknown	yes	no	telephone	may	
32206	33	blue-collar	married	basic.9y	no	no	no	cellular	may	
9403	36	admin.	married	high.school	no	no	no	telephone	jun	
14020	27	housemaid	married	high.school	no	yes	no	cellular	jul	

In [108]:

```
data_1.columns
```

Out[108]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y', 'pdays2'],
      dtype='object')
```

In [0]:

```
idx_numeric=[0,10,11,12,14,15,16,17,18]
scaler = MinMaxScaler()
data_1[data_1.columns[idx_numeric]] = scaler.fit_transform(data_1[data_1.columns[idx_numeri
```

In [110]:

```
data_1.head()
```

Out[110]:

	age	job	marital	education	default	housing	loan	contact	month	da
32884	0.519481	technician	married	high.school	no	no	yes	cellular	may	
3169	0.493506	unknown	married	unknown	unknown	yes	no	telephone	may	
32206	0.207792	blue-collar	married	basic.9y	no	no	no	cellular	may	
9403	0.246753	admin.	married	high.school	no	no	no	telephone	jun	
14020	0.129870	housemaid	married	high.school	no	yes	no	cellular	jul	

#Categorical variables can be either Ordinal or Nominal

In [0]:

```
data_1['poutcome'] = data_1['poutcome'].map({'failure': -1,'nonexistent': 0,'success': 1})
data_1['default'] = data_1['default'].map({'yes': -1,'unknown': 0,'no': 1})
data_1['housing'] = data_1['housing'].map({'yes': -1,'unknown': 0,'no': 1})
data_1['loan'] = data_1['loan'].map({'yes': -1,'unknown': 0,'no': 1})
```

In [112]:

```
data_1.head()
```

Out[112]:

	age	job	marital	education	default	housing	loan	contact	month	day_
32884	0.519481	technician	married	high.school	1	1	-1	cellular	may	
3169	0.493506	unknown	married	unknown	0	-1	1	telephone	may	
32206	0.207792	blue-collar	married	basic.9y	1	1	1	cellular	may	
9403	0.246753	admin.	married	high.school	1	1	1	telephone	jun	
14020	0.129870	housemaid	married	high.school	1	-1	1	cellular	jul	

In [113]:

```
data_1.shape
```

Out[113]:

```
(22080, 21)
```

#Handling Nominal Variables(One Hot Encoding)

'job', 'maritial', 'education', 'contact', 'month', 'day_of_week' are Nominal Variables

In [114]:

```
# One hot encoding of nominal varibles
nominal = ['job','marital','education','contact','month','day_of_week']
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()
```

Out[114]:

	age	default	housing	loan	campaign	pdays	previous	poutcome	emp.var.rate
32884	0.519481	1	1	-1	0.000000	1.0	0.166667	-1	0.333333
3169	0.493506	0	-1	1	0.018182	1.0	0.000000	0	0.937500
32206	0.207792	1	1	1	0.000000	1.0	0.166667	-1	0.333333
9403	0.246753	1	1	1	0.054545	1.0	0.000000	0	1.000000
14020	0.129870	1	-1	1	0.018182	1.0	0.000000	0	1.000000

In [115]:

```
data_clean_1.shape
```

Out[115]:

```
(22080, 56)
```

In [0]:

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

In [0]:

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

In [0]:

```
features_dropped = ['marital_single','contact_cellular',
                    'education_unknown','job_unknown',
     'marital_single','contact_cellular',
                    'education_unknown']
data_clean_1 = dropfeature(df_with_dummies, features_dropped)
```

In [119]:

```
data_clean_1.shape
```

Out[119]:

```
(22080, 52)
```

In [120]:

```
data_clean.shape
```

Out[120]:

```
(51588, 52)
```

In [121]:

```
data_clean_1.columns
```

Out[121]:

```
Index(['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', 'marital_divorce
d',
       'marital_married', 'marital_unknown', 'education_basic.4y',
       'education_basic.6y', 'education_basic.9y', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree', '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_fr
i',
       'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
       'day_of_week_wed'],
      dtype='object')
```

In [122]:

```
data_clean.columns
```

Out[122]:

```
Index(['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', 'marital_divorce
d',
       'marital_married', 'marital_unknown', 'education_basic.4y',
       'education_basic.6y', 'education_basic.9y', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree', '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_fr
i',
       'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
       'day_of_week_wed'],
      dtype='object')
```

Create Model

In [0]:

```
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
```

In [124]:

```
data_clean.shape
```

Out[124]:

```
(51588, 52)
```

In [125]:

```
data_clean_1.shape
```

Out[125]:

```
(22080, 52)
```

In [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
```

In [0]:

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

#y = data['y'].values
```

In [0]:

```
def Convert_Model(X_train,y_train,X_test,y_test,classifier):
     from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matr
     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,clas
     print("f1 score", f1)
     return classifier
```

In [129]:

```
X_train.shape
```

Out[129]:

```
(51588, 51)
```

In [130]:

```
X_test.shape
```

Out[130]:

```
(22080, 51)
```

In [131]:

```
# inport 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)
```

```
0.5032608695652174
[[    0 10968]
 [    0 11112]]
0.5032608695652174
0.5032608695652174
1.0
f1 score 0.6695589298626176
```

LogisticRegression

In [0]:

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

In [133]:

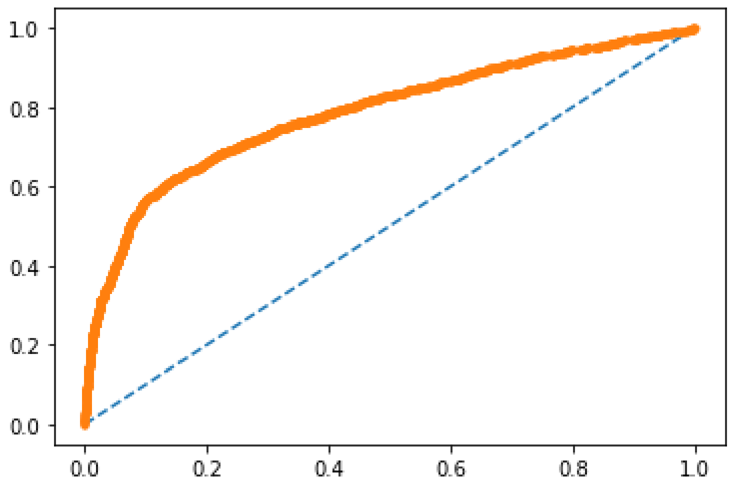
```
# inport Dummy Classifier for creating Base Model
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(random_state=0)
finalModel_lr = Convert_Model(X_train,y_train,X_test,y_test,classifier_lr)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

0.7341032608695652
[[9233 1735]
 [4136 6976]]
0.7341032608695652
0.8008265411548616
0.6277897768178545
f1 score 0.7038288856378954
```

In [134]:

```
# 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: 0.787
```



Random Forest Classifier

In [0]:

```
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

In [136]:

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

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports
until
```

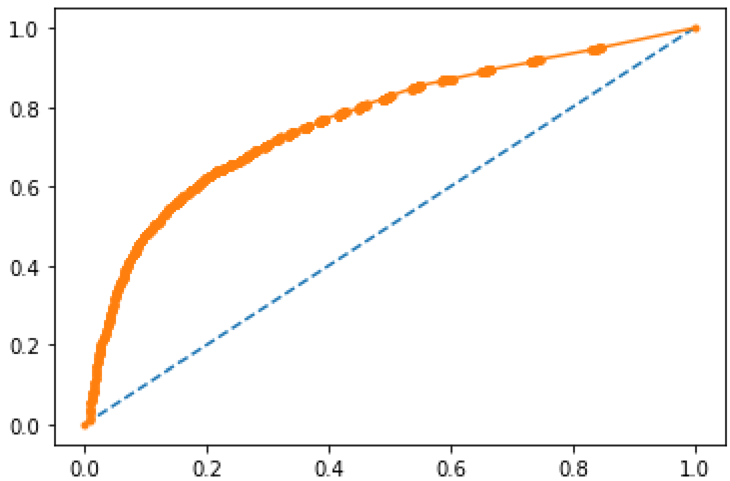
```
0.6591032608695652
[[10233   735]
 [ 6792  4320]]
0.6591032608695652
0.8545994065281899
0.38876889848812096
f1 score 0.5344219706810168
```

Testing

In [137]:

```

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: 0.764
```



Feature Importance

In [138]:

```
data_clean.head()
```

Out[138]:

	age	default	housing	loan	campaign	pdays	previous	poutcome	emp.var.rate
39075	0.000000	1	1	1	0.666667	1.0	0.142857	-1	0.083333
34855	0.000000	1	1	1	1.000000	1.0	0.000000	0	0.333333
7107	0.666667	0	-1	1	0.333333	1.0	0.000000	0	0.937500
31614	0.333333	1	1	1	0.000000	1.0	0.142857	-1	0.333333
34878	0.333333	1	1	1	1.000000	1.0	0.000000	0	0.333333

In [139]:

```
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=['im
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
```

In [140]:

```
feature_importances
```

Out[140]:

	importance
euribor3m	0.150254
campaign	0.080627
age	0.072522
nr.employed	0.061296
emp.var.rate	0.050446
housing	0.042294
marital_married	0.031835
loan	0.028005
cons.conf.idx	0.025639
default	0.023751
cons.price.idx	0.022879
poutcome	0.020840
pdays2	0.020274
job_admin.	0.019714
contact_telephone	0.019123
education_high.school	0.018692
education_university.degree	0.017136
pdays	0.016834
marital_divorced	0.016025
day_of_week_mon	0.016024
job_blue-collar	0.015896
job_technician	0.015715
day_of_week_thu	0.015330
day_of_week_wed	0.015305
day_of_week_tue	0.015221
day_of_week_fri	0.015216
education_basic.9y	0.015038
education_professional.course	0.012786
previous	0.012574
job_services	0.011420
month_may	0.011315
job_management	0.010450
education_basic.4y	0.010261
education_basic.6y	0.008843
job_self-employed	0.007332

	importance
job_entrepreneur	0.006973
job_retired	0.006697
month_oct	0.006247
job_student	0.005393
job_unemployed	0.005204
job_housemaid	0.005193
month_apr	0.003560
month_mar	0.003041
month_jun	0.002442
month_jul	0.002277
month_aug	0.001993
month_nov	0.001632
month_sep	0.001029
marital_unknown	0.000739
month_dec	0.000404
education_illiterate	0.000266

SVM Classifier

In [0]:

```
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

In [0]:

```

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

In [0]:

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

In [144]:

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

0.7313915857605178
[[9640 1328]
 [7496 3616]]
0.6003623188405797
0.7313915857605178
0.3254139668826494
f1 score 0.45042351768809163
```

In [145]:

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

```
The best model is:  {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
This model produces a mean cross-validated score (precision) of 0.8724364983
829902
```

Testing

In [146]:

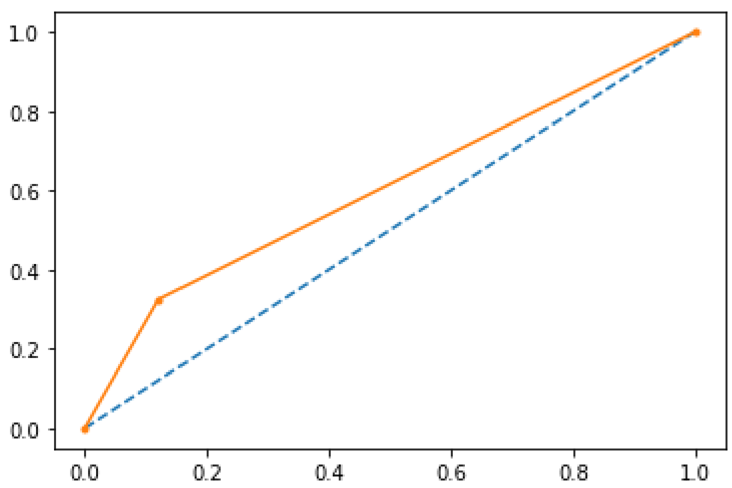
```
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
y_true, y_pred = y_test, finalModel_gb.predict(X_test)
pre1 = precision_score(y_true, y_pred)
rec1 = recall_score(y_true, y_pred)
acc1 = accuracy_score(y_true, y_pred)
f1_1 = f1_score(y_true, y_pred)
print('precision 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:  0.7313915857605178
recall on the evaluation set:  0.3254139668826494
accuracy on the evaluation set:  0.6003623188405797
F1 on the evaluation set 0.45042351768809163
```

In [147]:

```
# 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()
```

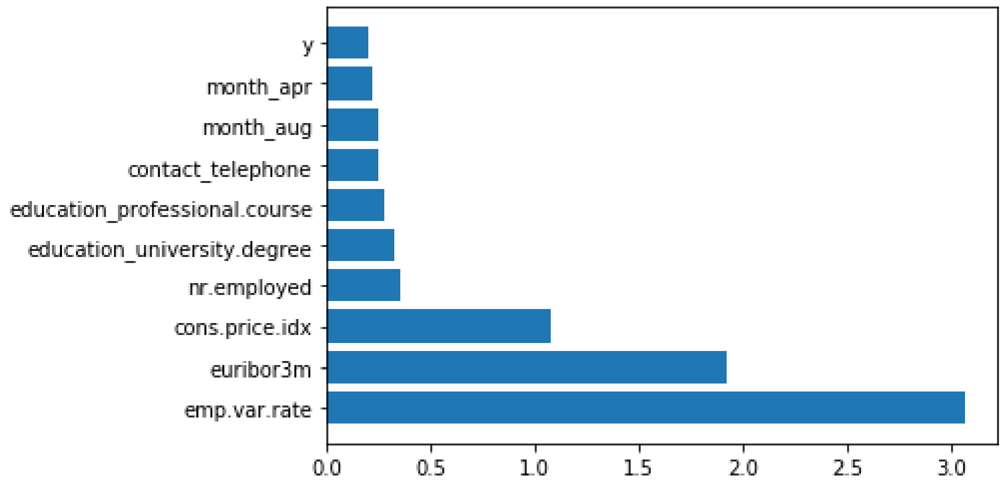
```
AUC: 0.602
```



In [148]:

```
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/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
ConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



Reducing Features using PCA

Classify the model using XGBClassifier

In [0]:

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

In [150]:

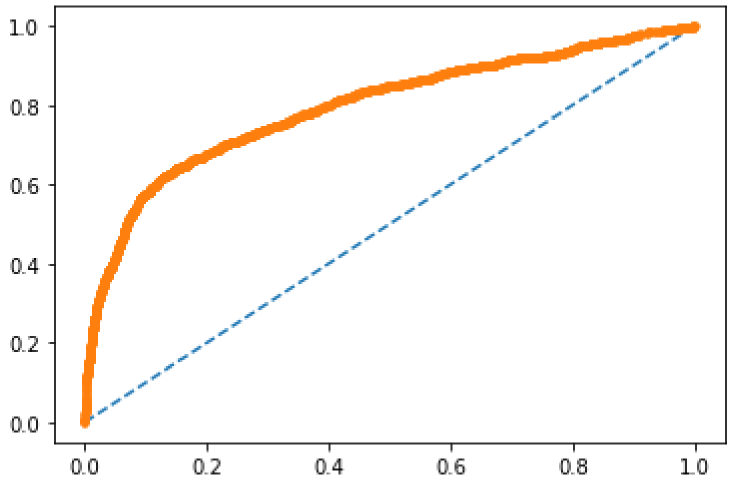
```
# fit model no training data
model = XGBClassifier()
finalModel_XGB = Convert_Model(X_train,y_train,X_test,y_test,model)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:219: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:252: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)

0.7442934782608696
[[9330 1638]
 [4008 7104]]
0.7442934782608696
0.8126286890871655
0.6393088552915767
f1 score 0.7156240556059232
```

In [151]:

```
#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: 0.797
```



In [152]:

```
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=[
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:219: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:252: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

In [153]:

```
feature_importances
```

Out[153]:

	importance
nr.employed	0.575482
month_oct	0.051430
cons.conf.idx	0.049530
pdays	0.033252
cons.price.idx	0.033085
euribor3m	0.028439
default	0.016820
contact_telephone	0.015261
poutcome	0.011918
previous	0.011453
month_apr	0.010338
age	0.009056
job_services	0.009027
month_mar	0.008550
emp.var.rate	0.008050
day_of_week_tue	0.007238
education_basic.4y	0.006695
month_may	0.006264
marital_divorced	0.006192
month_nov	0.006008
education_professional.course	0.005805
month_aug	0.005701
education_university.degree	0.005622
education_basic.9y	0.005564
education_basic.6y	0.005544
job_unemployed	0.005374
marital_married	0.005251
day_of_week_wed	0.005194
day_of_week_mon	0.004796
campaign	0.004566
job_student	0.004439
job_entrepreneur	0.004310
day_of_week_thu	0.004272
loan	0.004083
housing	0.003945

	importance
job_blue-collar	0.003919
job_technician	0.003517
job_housemaid	0.003488
job_retired	0.003431
month_jul	0.002492
education_illiterate	0.002200
job_management	0.001210
job_self-employed	0.001191
job_admin.	0.000000
month_dec	0.000000
month_jun	0.000000
pdays2	0.000000
education_high.school	0.000000
marital_unknown	0.000000
month_sep	0.000000
day_of_week_fri	0.000000

MLP Classifier with 3 layer

In [154]:

```
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/python3.6/dist-packages/sklearn/neural_network/multilayer_per
ceptron.py:921: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples, ), for exa
mple using ravel().
  y = column_or_1d(y, warn=True)
```

Out[154]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.
9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(13, 13, 13), learning_rate='constant',
              learning_rate_init=0.001, max_iter=500, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

In [155]:

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

```
[[8373 2595]
 [3952 7160]]
```

In [156]:

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

```
              precision    recall  f1-score   support

           0       0.68      0.76      0.72     10968
           1       0.73      0.64      0.69     11112

    accuracy                           0.70     22080
   macro avg       0.71      0.70      0.70     22080
weighted avg       0.71      0.70      0.70     22080
```

In [157]:

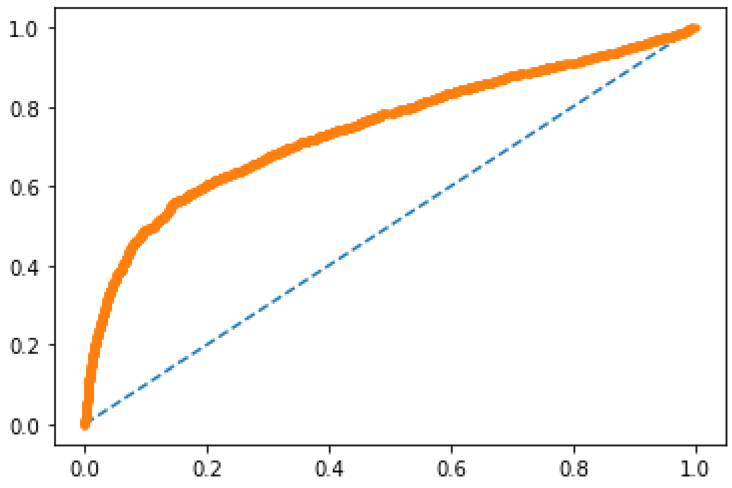
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score


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/python3.6/dist-packages/sklearn/neural_network/multilayer_per
ceptron.py:921: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples, ), for exa
mple using ravel().
  y = column_or_1d(y, warn=True)

0.7008152173913044
[[8618 2350]
 [4256 6856]]
0.7008152173913044
0.7447316967195308
0.6169906407487401
f1 score 0.6748695737769466
```

In [158]:

```
# 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: 0.745
```



MLP Classifier with 2 layer

In [162]:

```
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)
```

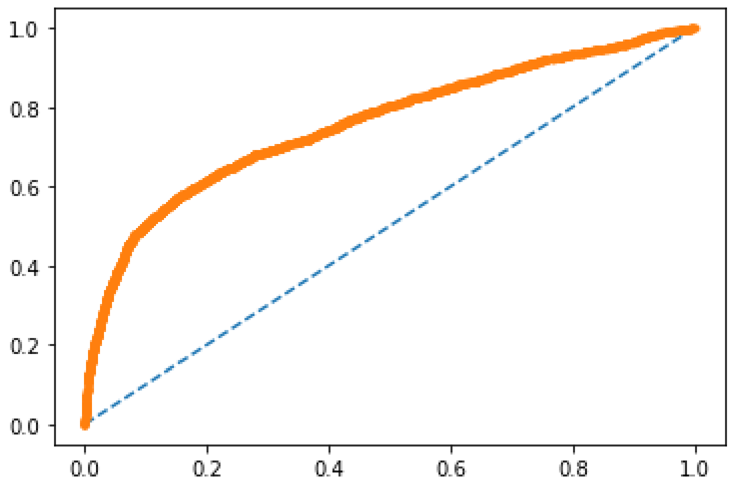
```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_per
ceptron.py:921: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples, ), for exa
mple using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.7070199275362319
[[8723 2245]
 [4224 6888]]
0.7070199275362319
0.754188109055075
0.6198704103671706
f1 score 0.6804643121758459
```

In [163]:

```
# 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: 0.760
```



MLP Classifier with 1 layer

In [164]:

```
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)
```

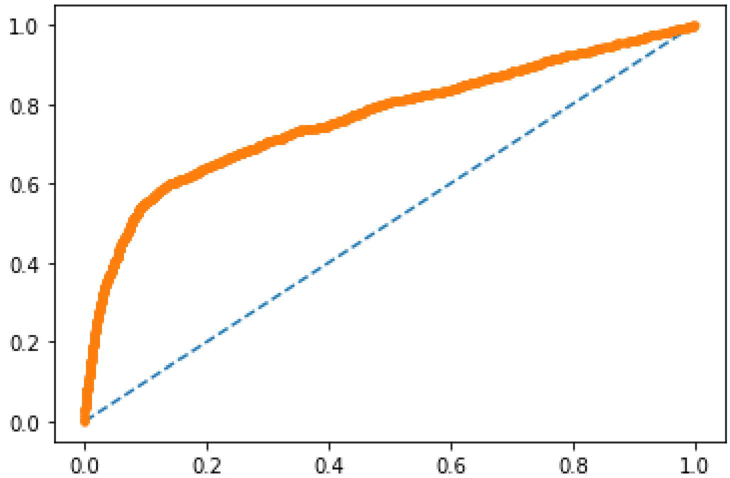
```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_per
ceptron.py:921: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples, ), for exa
mple using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.7223278985507247
[[9021 1947]
 [4184 6928]]
0.7223278985507247
0.7806197183098591
0.6234701223902088
f1 score 0.693250612898384
```

In [165]:

```
# 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: 0.766
```



In [166]:

```
#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       |      0.6
2      |   0.7    |  0.78 |
|         Random Forest          |      0.65      |       0.85      |      0.3
8      |   0.52   | 0.766 |
|         SVM classifier         |      0.73      |       0.82      |      0.6
|   0.69   |  0.73 |
|           XGB boost            |      0.74      |       0.81      |      0.6
3      |   0.71   | 0.798 |
| MLP  classifier with 3 layers |      0.7       |       0.74      |      0.6
1      |   0.67   | 0.745 |
|  MLP classifier with 2 layers |      0.7       |       0.75      |      0.6
1      |   0.68   |  0.76 |
|    MLP classifier 1 layers    |      0.72      |       0.78      |      0.6
2      |  0.693   | 0.766 |
+--------------------------------+-----------------+-----------------+--------
------+----------+-------+
```

In [0]:

```
# for EDA part i have take refrance from Chaitra Hegde github assigment
# for differnt model implemnetation  google as source
```