NEWBank-telemarketing-campaign

January 28, 2022

[2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import random
from sklearn.model_selection import GridSearchCV
```

[3]:
```
df= pd.read_csv("train.csv")
```

[4]:
```
df.head()
```

[4]:
```
      ID  age         job   marital education default  balance housing loan  \
0  26110   56      admin.   married   unknown      no     1933      no   no
1  40576   31     unknown   married secondary      no        3      no   no
2  15320   27    services   married secondary      no      891     yes   no
3  43962   57  management  divorced  tertiary      no     3287      no   no
4  29842   31  technician   married secondary      no      119     yes   no

     contact  day month  duration  campaign  pdays  previous poutcome  \
0  telephone   19   nov        44         2     -1         0  unknown
1   cellular   20   jul        91         2     -1         0  unknown
2   cellular   18   jul       240         1     -1         0  unknown
3   cellular   22   jun       867         1     84         3  success
4   cellular    4   feb       380         1     -1         0  unknown

  subscribed
0         no
1         no
2         no
3        yes
4         no
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 31647 entries, 0 to 31646
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          31647 non-null  int64
 1   age         31647 non-null  int64
 2   job         31647 non-null  object
 3   marital     31647 non-null  object
 4   education   31647 non-null  object
 5   default     31647 non-null  object
 6   balance     31647 non-null  int64
 7   housing     31647 non-null  object
 8   loan        31647 non-null  object
 9   contact     31647 non-null  object
 10  day         31647 non-null  int64
 11  month       31647 non-null  object
 12  duration    31647 non-null  int64
 13  campaign    31647 non-null  int64
 14  pdays       31647 non-null  int64
 15  previous    31647 non-null  int64
 16  poutcome    31647 non-null  object
 17  subscribed  31647 non-null  object
dtypes: int64(8), object(10)
memory usage: 4.3+ MB
```

[6]:
```
df.shape
```

[6]:
```
(31647, 18)
```

[ ]:
```

```

[7]:
```
df.describe()
```

[7]:
```
                 ID           age        balance           day      duration  \
count  31647.000000  31647.000000   31647.000000  31647.000000  31647.000000
mean   22563.972162     40.957247    1363.890258     15.835466    258.113534
std    13075.936990     10.625134    3028.304293      8.337097    257.118973
min        2.000000     18.000000   -8019.000000      1.000000      0.000000
25%    11218.000000     33.000000      73.000000      8.000000    104.000000
50%    22519.000000     39.000000     450.000000     16.000000    180.000000
75%    33879.500000     48.000000    1431.000000     21.000000    318.500000
max    45211.000000     95.000000  102127.000000     31.000000   4918.000000

           campaign         pdays      previous
count  31647.000000  31647.000000  31647.000000
mean       2.765697     39.576042      0.574272
std        3.113830     99.317592      2.422529
min        1.000000     -1.000000      0.000000
```

```
25%        1.000000     -1.000000      0.000000
50%        2.000000     -1.000000      0.000000
75%        3.000000     -1.000000      0.000000
max       63.000000    871.000000    275.000000
```

0.1 Categorical feature

0.1.1 Exploring the unique values

[8]:
```
## Exploring the unique values

for col in df.select_dtypes(include='object').columns:
    print(col)
    print(df[col].unique())
```

```
job
['admin.' 'unknown' 'services' 'management' 'technician' 'retired'
 'blue-collar' 'housemaid' 'self-employed' 'student' 'entrepreneur'
 'unemployed']
marital
['married' 'divorced' 'single']
education
['unknown' 'secondary' 'tertiary' 'primary']
default
['no' 'yes']
housing
['no' 'yes']
loan
['no' 'yes']
contact
['telephone' 'cellular' 'unknown']
month
['nov' 'jul' 'jun' 'feb' 'sep' 'jan' 'may' 'aug' 'apr' 'oct' 'mar' 'dec']
poutcome
['unknown' 'success' 'failure' 'other']
subscribed
['no' 'yes']
```

[9]:
```
categorical_features=[feature for feature in df.columns if ((df[feature].
 ↪dtypes=='O') & (feature not in ['subscribed']))]
categorical_features
```

[9]:
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
```

```
 'contact',
 'month',
 'poutcome']
```

[10]:
```
### categorical values in each feature

for feature in categorical_features:
    print('The feature is {} and number of categories are {}'.
 ↪format(feature,len(df[feature].unique())))
```

```
The feature is job and number of categories are 12
The feature is marital and number of categories are 3
The feature is education and number of categories are 4
The feature is default and number of categories are 2
The feature is housing and number of categories are 2
The feature is loan and number of categories are 2
The feature is contact and number of categories are 3
The feature is month and number of categories are 12
The feature is poutcome and number of categories are 4
```

There are 9 categorical features.

feature job and month has highest number of categorical values

[11]:
```
# find missing values



features_na = [features for features in df.columns if df[features].isnull().
 ↪sum() > 0]
for feature in features_na:
    print(feature, np.round(df[feature].isnull().mean(), 4),  ' % missing␣
 ↪values')
else:
    print("No missing value found")
```

```
No missing value found
```

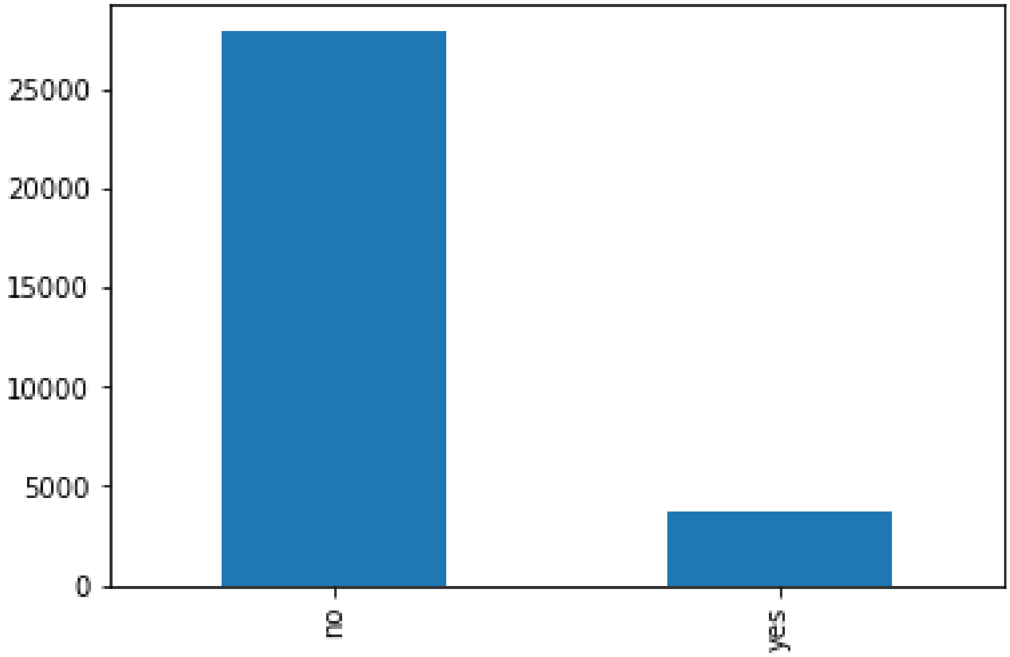
We first did the exploratory analysis of the categorical variables and saw what are there categories and were there any missing values for these categories. Here, Now we will use the seaborn package to create the bar graphs below.

Count of dependent variable

[12]:
```
subs = df['subscribed'].value_counts()
```

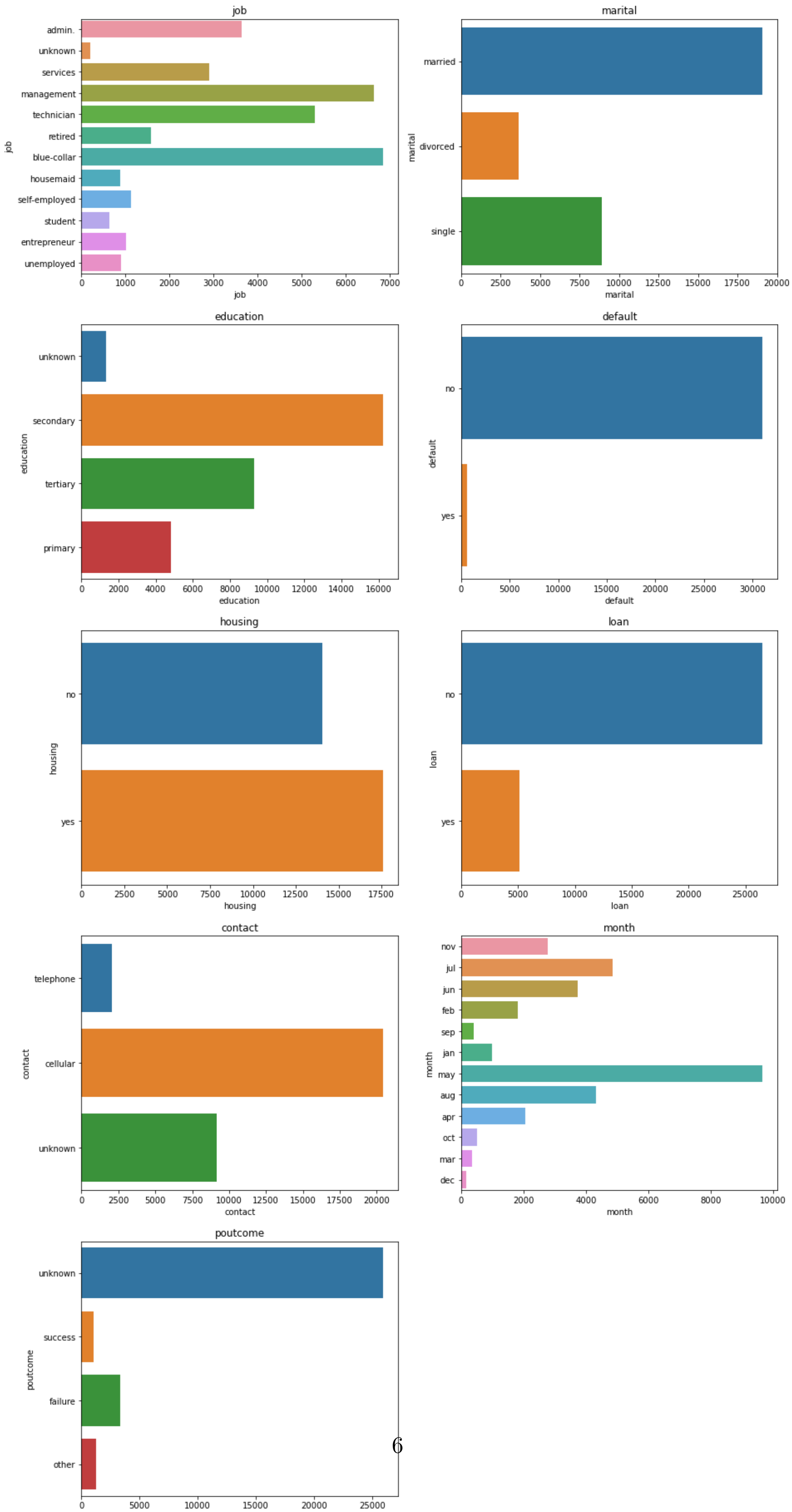
[13]:
```
subs.plot.bar()
```

[13]: <AxesSubplot:>



Data is highly imbalanced.The number of people saying no to term deposit is more than the people saying yes.

[14]:
```
plt.figure(figsize=(15,80), facecolor='white')
plotnumber =1
for categorical_feature in categorical_features:
    ax = plt.subplot(12,2,plotnumber)
    sns.countplot(y=categorical_feature,data=df)
    plt.xlabel(categorical_feature)
    plt.title(categorical_feature)
    plotnumber+=1
plt.show()
```


job
admin.
unknown
services
management
technician
retired
blue-collar
housemaid
self-employed
student
entrepreneur
unemployed
0 1000 2000 3000 4000 5000 6000 7000
marital
married
divorced
single
0 2500 5000 7500 10000 12500 15000 17500 20000
education
unknown
secondary
tertiary
primary
0 2000 4000 6000 8000 10000 12000 14000 16000
default
no
yes
0 5000 10000 15000 20000 25000 30000
housing
no
yes
0 2500 5000 7500 10000 12500 15000 17500
loan
no
yes
0 5000 10000 15000 20000 25000
contact
telephone
cellular
unknown
0 2500 5000 7500 10000 12500 15000 17500 20000
month
nov
jul
jun
feb
sep
jan
may
aug
apr
oct
mar
dec
0 2000 4000 6000 8000 10000
poutcome
unknown
success
failure
other
0 5000 10000 15000 20000 25000

client with job type as management records and blue collor are high in given dataset and housemaid are very less.

client who married are high in records in given dataset and divorced are less.

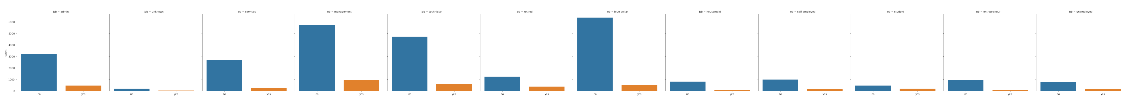
client whoes education background is secondary are in high numbers in given dataset.

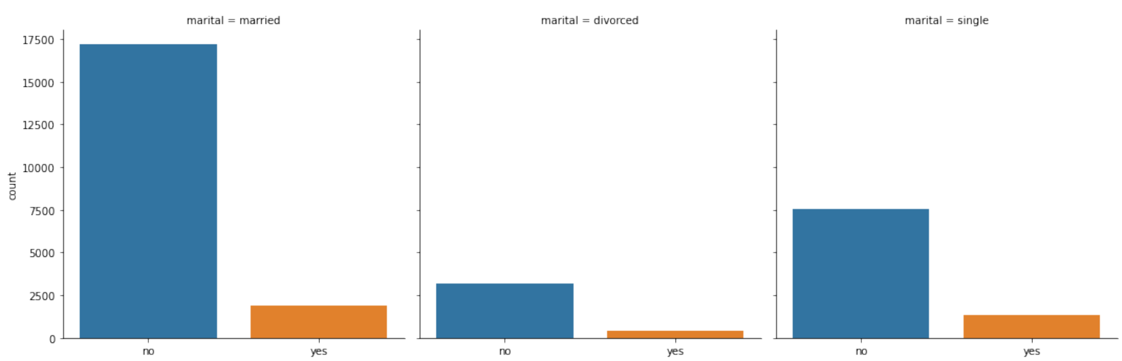
defualt feature seems to be does not play importand role as it has value of no at high ratio to value yes which can drop.

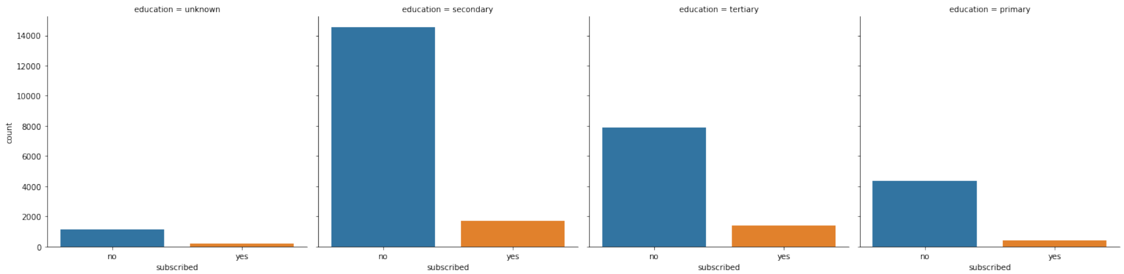
data in month of may is high and less in dec.

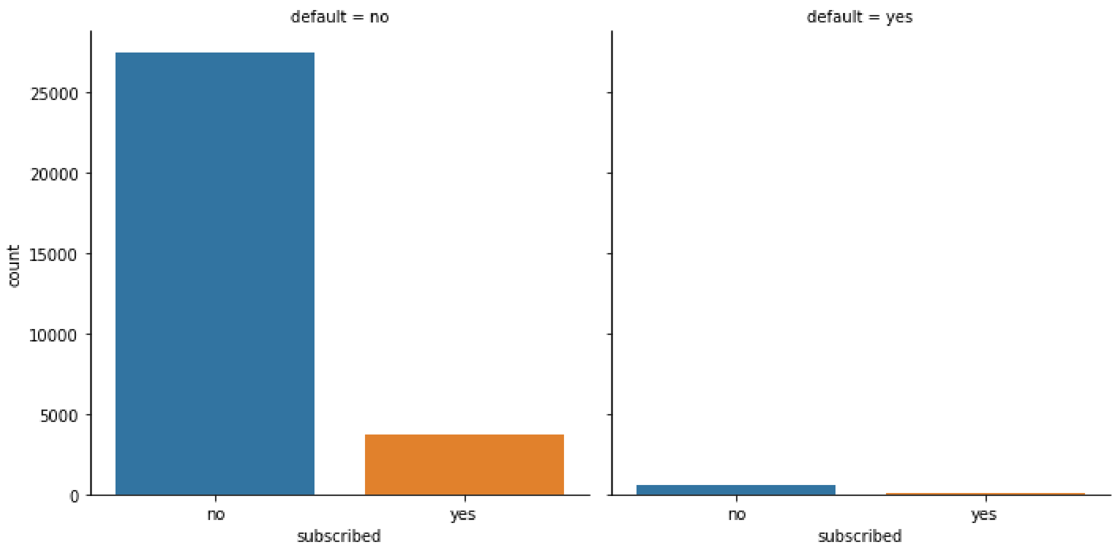
1 Visualize the relation between feature category vs depended variable 'subscribed'.

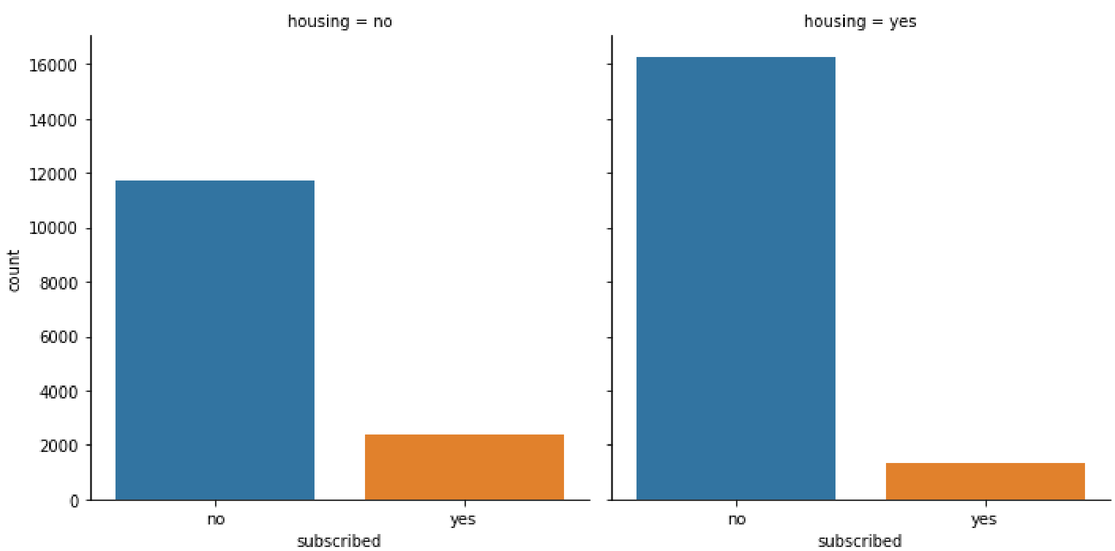
[15]:
```
## Find out the relationship between categorical variable and dependent variable


for categorical_feature in categorical_features:
    sns.catplot(x='subscribed', col=categorical_feature, kind='count', data= df)
plt.show()
```

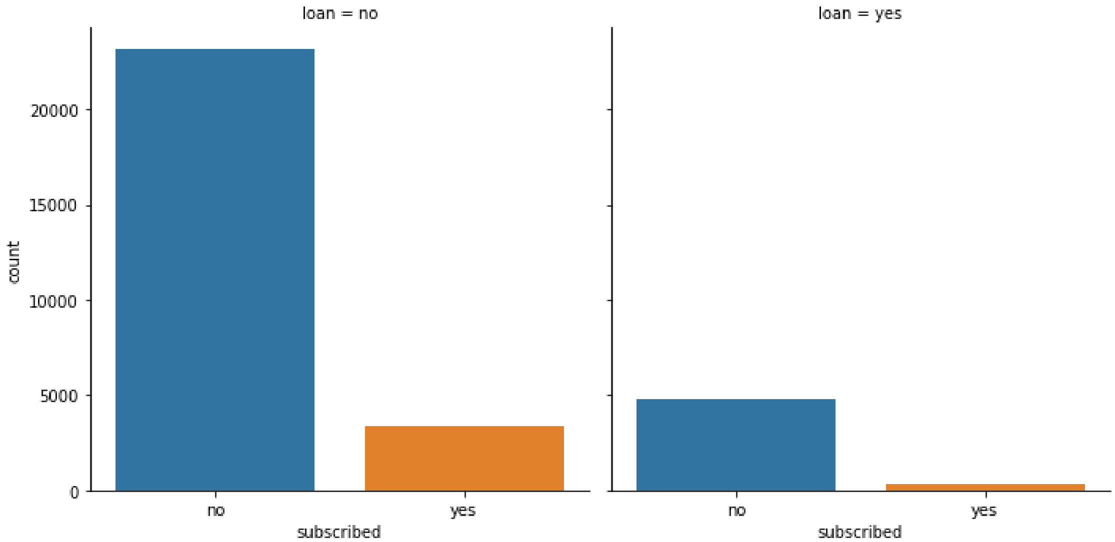


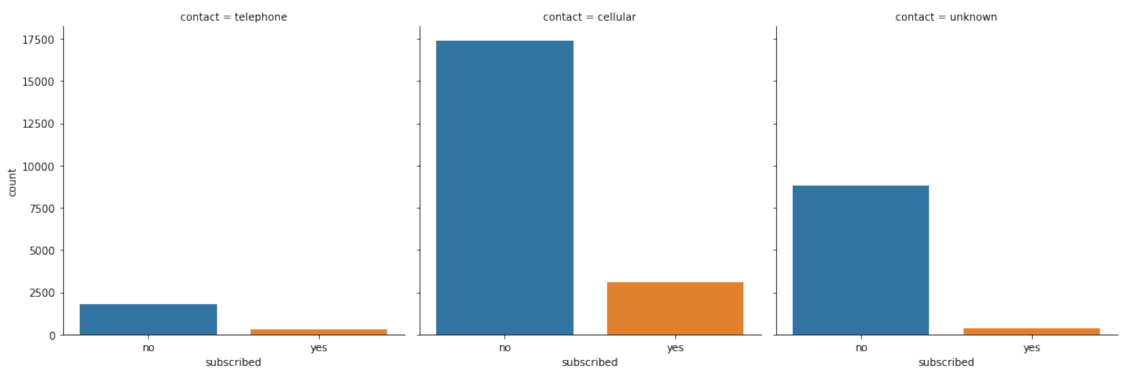


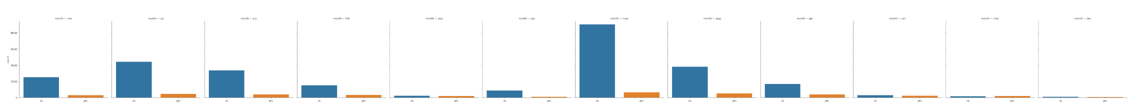

education = unknown
education = secondary
education = tertiary
education = primary
count
subscribed
no
yes

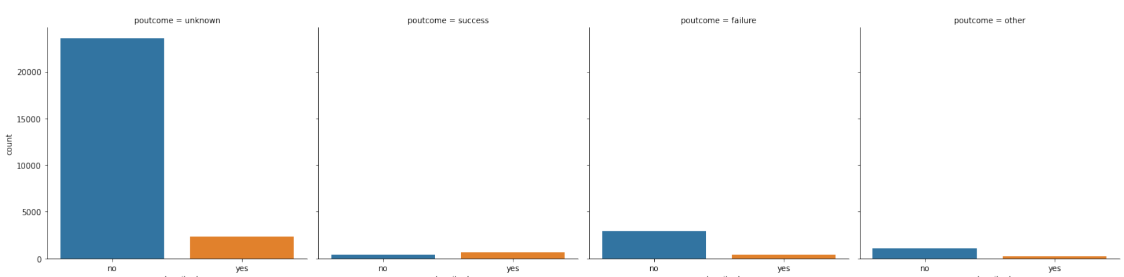

default = no
default = yes
25000
20000
15000
10000
5000
0
count
no
yes
subscribed


housing = no
housing = yes
16000
14000
12000
10000
8000
6000
4000
2000
0
count
no
yes
subscribed


loan = no
loan = yes
20000
15000
10000
5000
0
count
no
yes
subscribed
no
yes
subscribed


contact = telephone
contact = cellular
contact = unknown
17500
15000
12500
10000
7500
5000
2500
0
count
no
yes
subscribed
no
yes
subscribed
no
yes
subscribed




poutcome = unknown
poutcome = success
poutcome = failure
poutcome = other
20000
15000
10000
5000
0
count
no
yes
subscribed
no
yes
subscribed
no
yes
subscribed
no
yes
subscribed

[16]:
```python
###Checking the target label split over categorical features and find the count
## subscribed columns against categorical feature


for categorical_feature in categorical_features:
    print(df.groupby(['subscribed',categorical_feature]).size())
```

```
subscribed  job
no          admin.           3179
            blue-collar      6353
            entrepreneur      923
            housemaid         795
            management       5716
            retired          1212
            self-employed     983
            services         2649
            student           453
            technician       4713
            unemployed        776
            unknown           180
yes         admin.            452
            blue-collar       489
            entrepreneur       85
            housemaid          79
            management        923
            retired           362
            self-employed     140
            services          254
            student           182
            technician        594
            unemployed        129
            unknown            26
dtype: int64
subscribed  marital
no          divorced     3185
            married     17176
            single       7571
yes         divorced      445
            married      1919
            single       1351
dtype: int64
subscribed  education
no          primary       4381
            secondary    14527
            tertiary      7886
            unknown       1138
yes         primary        427
            secondary     1697
```

```
            tertiary      1415
            unknown        176
dtype: int64
subscribed  default
no          no         27388
            yes          544
yes         no          3674
            yes           41
dtype: int64
subscribed  housing
no          no         11698
            yes        16234
yes         no          2365
            yes         1350
dtype: int64
subscribed  loan
no          no      23132
            yes      4800
yes         no       3384
            yes       331
dtype: int64
subscribed  contact
no          cellular     17352
            telephone     1779
            unknown       8801
yes         cellular      3071
            telephone      268
            unknown        376
dtype: int64
subscribed  month
no          apr      1671
            aug      3813
            dec        85
            feb      1522
            jan       880
            jul      4403
            jun      3355
            mar       168
            may      9020
            nov      2508
            oct       288
            sep       219
yes         apr       384
            aug       520
            dec        72
            feb       305
            jan        97
            jul       441
```

```
             jun        383
             mar        174
             may        649
             nov        275
             oct        224
             sep        191
dtype: int64
subscribed  poutcome
no          failure      2931
            other        1071
            success       374
            unknown     23556
yes         failure       431
            other         217
            success       694
            unknown      2373
dtype: int64
```

Retired client has high interest on deposit.

Client who has housing loan seems to be not interested much on deposit.

In month of may, records are high but client interst ratio is very less.

1.0.1 Explore numerical features

[17]:
```
### list of numerical variables

numerical_features = [feature for feature in df.columns if ((df[feature].dtypes
↪!= 'O') & (feature not in ['subsribed']))]
print('Number of numerical variables: ', len(numerical_features))


# visualise the numerical variables
df[numerical_features].head()
```

```
Number of numerical variables:  8
```

[17]:

	ID	age	balance	day	duration	campaign	pdays	previous
0	26110	56	1933	19	44	2	-1	0
1	40576	31	3	20	91	2	-1	0
2	15320	27	891	18	240	1	-1	0
3	43962	57	3287	22	867	1	84	3
4	29842	31	119	4	380	1	-1	0

1.0.2 Find Discrete numerical feature

[18]:
```
discrete_feature=[feature for feature in numerical_features if len(df[feature].
 ↪unique())<25]
print("Discrete Variables Count: {}".format(len(discrete_feature)))
```

```
Discrete Variables Count: 0
```

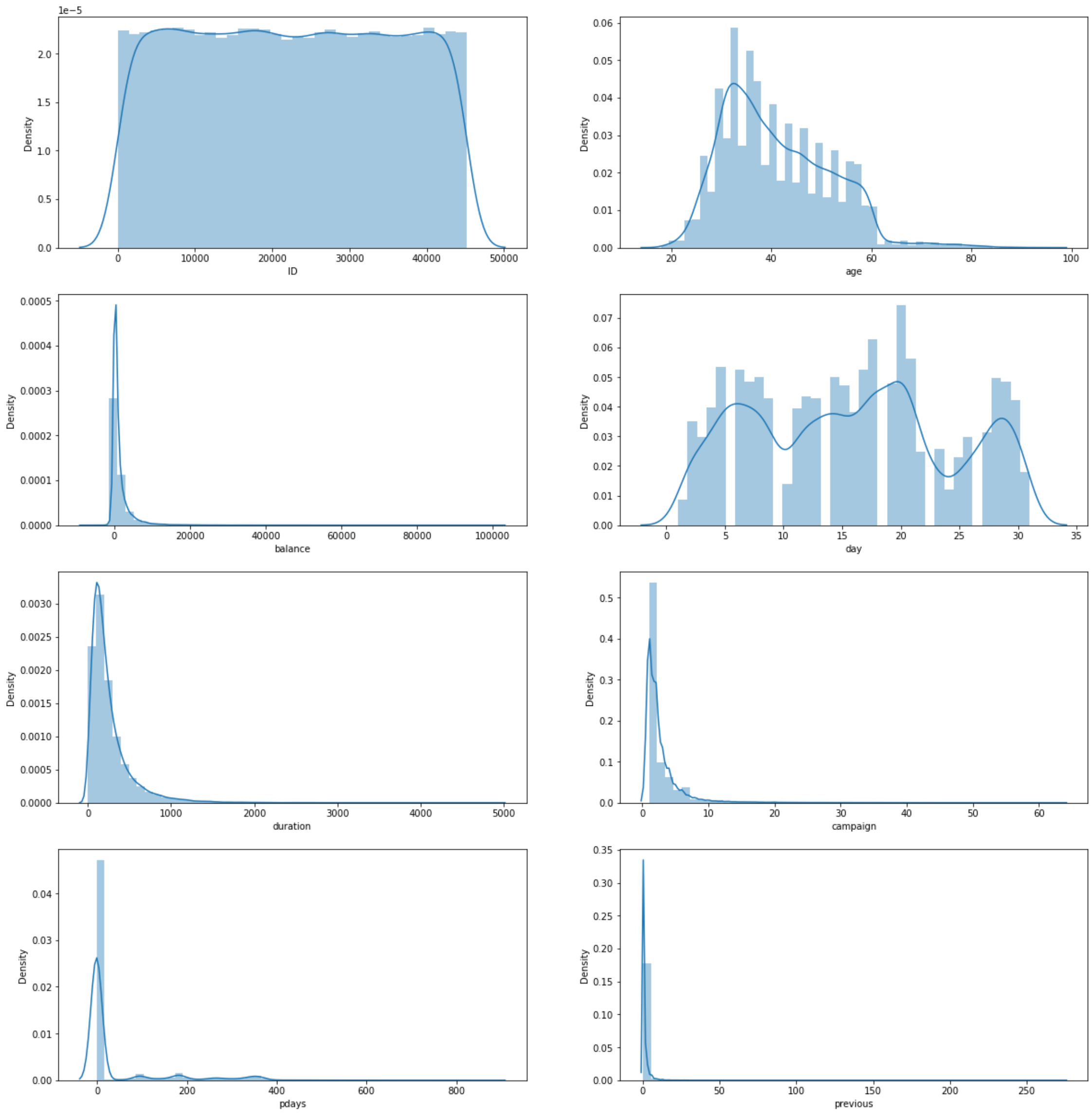
1.0.3 Find continous numerical feature

[19]:
```
continuous_features=[feature for feature in numerical_features if feature not␣
 ↪in discrete_feature+['subscribed']]
print("Continuous feature Count {}".format(len(continuous_features)))
```

```
Continuous feature Count 8
```

1.0.4 Distribution of continous numerical features

[20]:
```
### plot a univariate distribution of continues observations



plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for continuous_feature in continuous_features:
    ax = plt.subplot(11,2,plotnumber)
    sns.distplot(df[continuous_feature])
    plt.xlabel(continuous_feature)
    plotnumber+=1
plt.show()
```



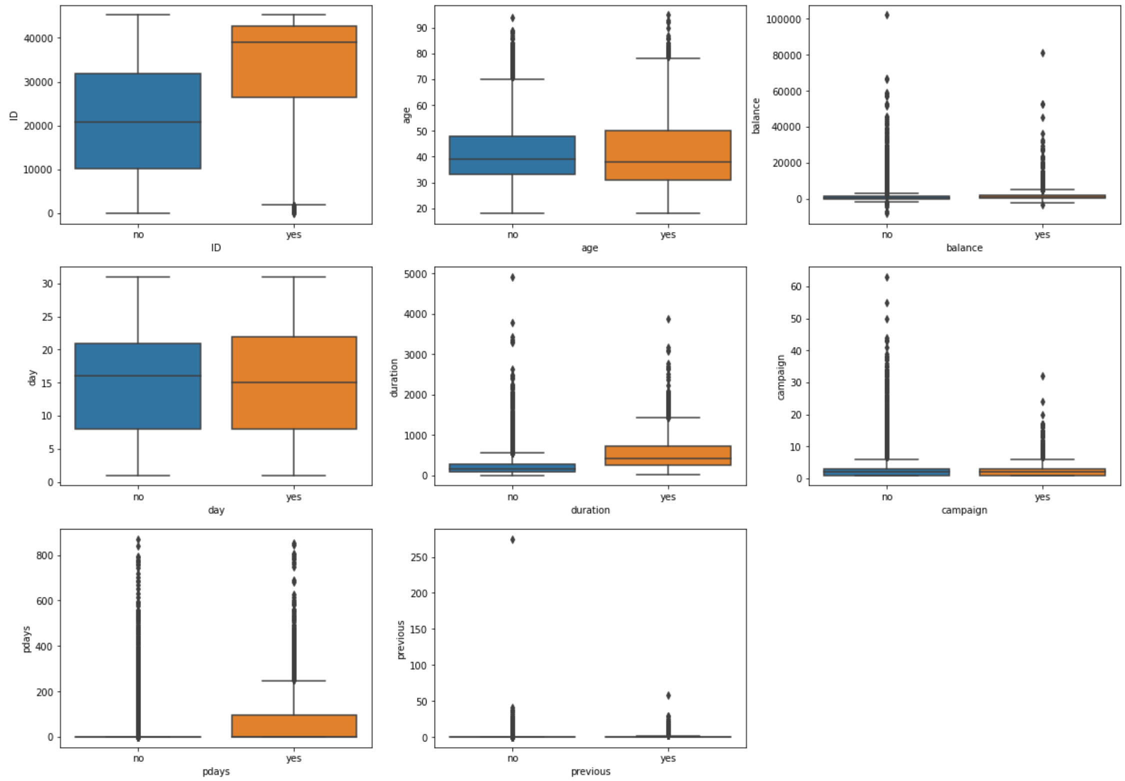
It seems age & days distributed normally.

balance, duration, compaign, pdays and previous heavely skewed towards left and seems to be have some outliers.

1.1 Relation between continous numerical feature and labels

[21]:
```
#boxplot to show target distribution with respect numerical features

plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for feature in continuous_features:
    ax = plt.subplot(12,3,plotnumber)
```

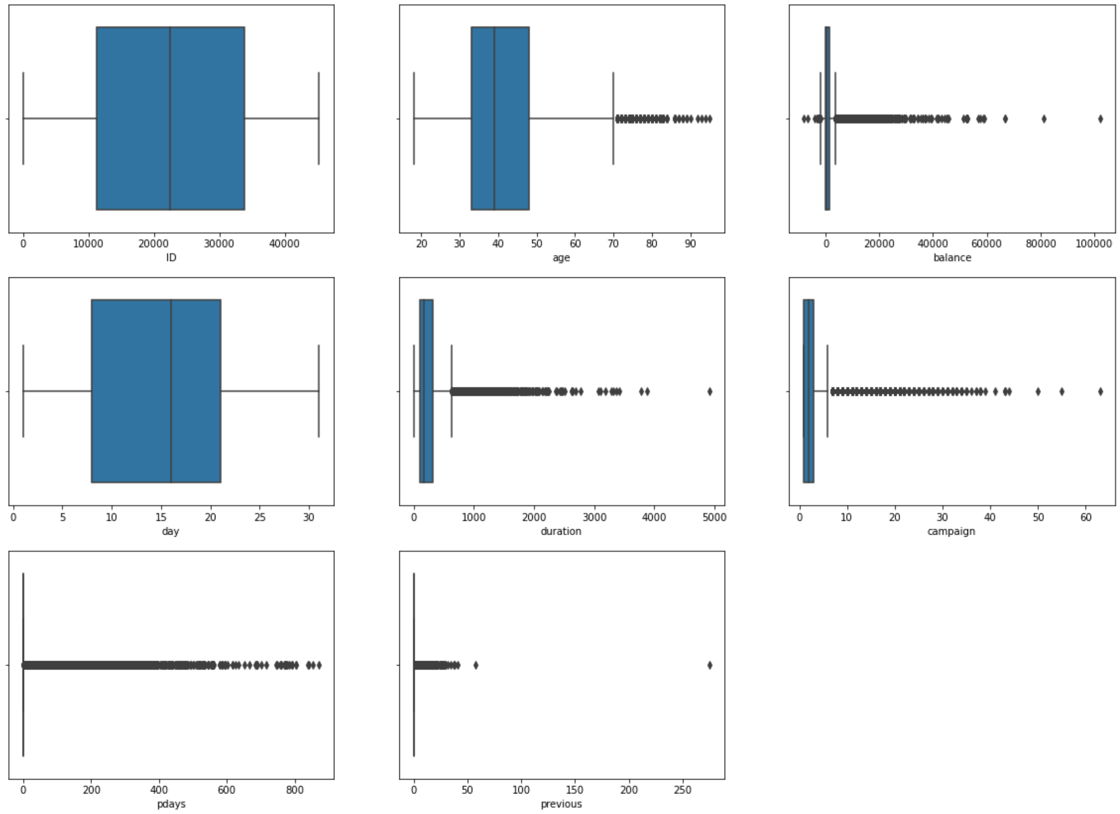
```
    sns.boxplot(x="subscribed", y= df[feature], data=df)
    plt.xlabel(feature)
    plotnumber+=1
plt.show()
```



client shows interest on deposit who had discussion for longer duration

1.1.1 Find outlier in numerical feature

[22]:
```
#boxplot on numerical features to find outliers

plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for numerical_feature in numerical_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.boxplot(df[numerical_feature])
    plt.xlabel(numerical_feature)
    plotnumber+=1
plt.show()
```

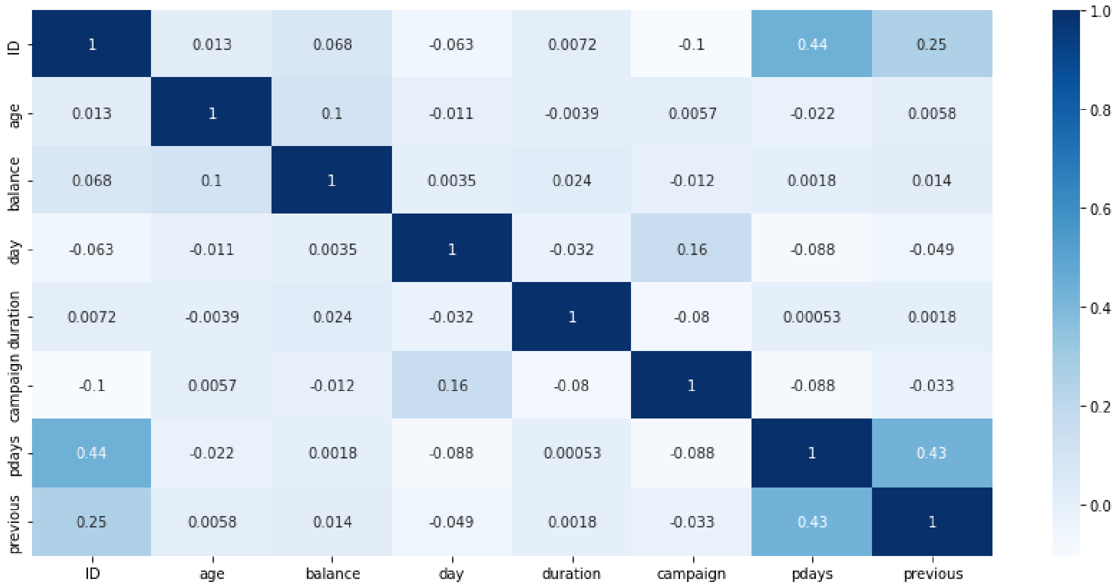


age, balance, duration, compaign, pdays and previous has some outliers

1.1.2 Correlattion between numerical feature

[27]:
```
## Checking for correlation

cor_mat=df.corr()
fig = plt.figure(figsize=(15,7))
sns.heatmap(cor_mat,annot=True ,cmap="Blues")
```

[27]: <AxesSubplot:>



We can infer that duration of the call is highly correlated with the target variable. As the duration of the call is more, there are higher chances that the client is showing interest in the term deposit and hence there are higher chances that the client will subscribe to term deposit.

1.1.3 Bivariate analysis

[28]:
```
#job vs subscribed
print(pd.crosstab(df['job'],df['subscribed']))
```

```
subscribed       no   yes
job
admin.         3179   452
blue-collar    6353   489
entrepreneur    923    85
housemaid       795    79
management     5716   923
retired        1212   362
self-employed   983   140
services       2649   254
student         453   182
technician     4713   594
unemployed      776   129
unknown         180    26
```

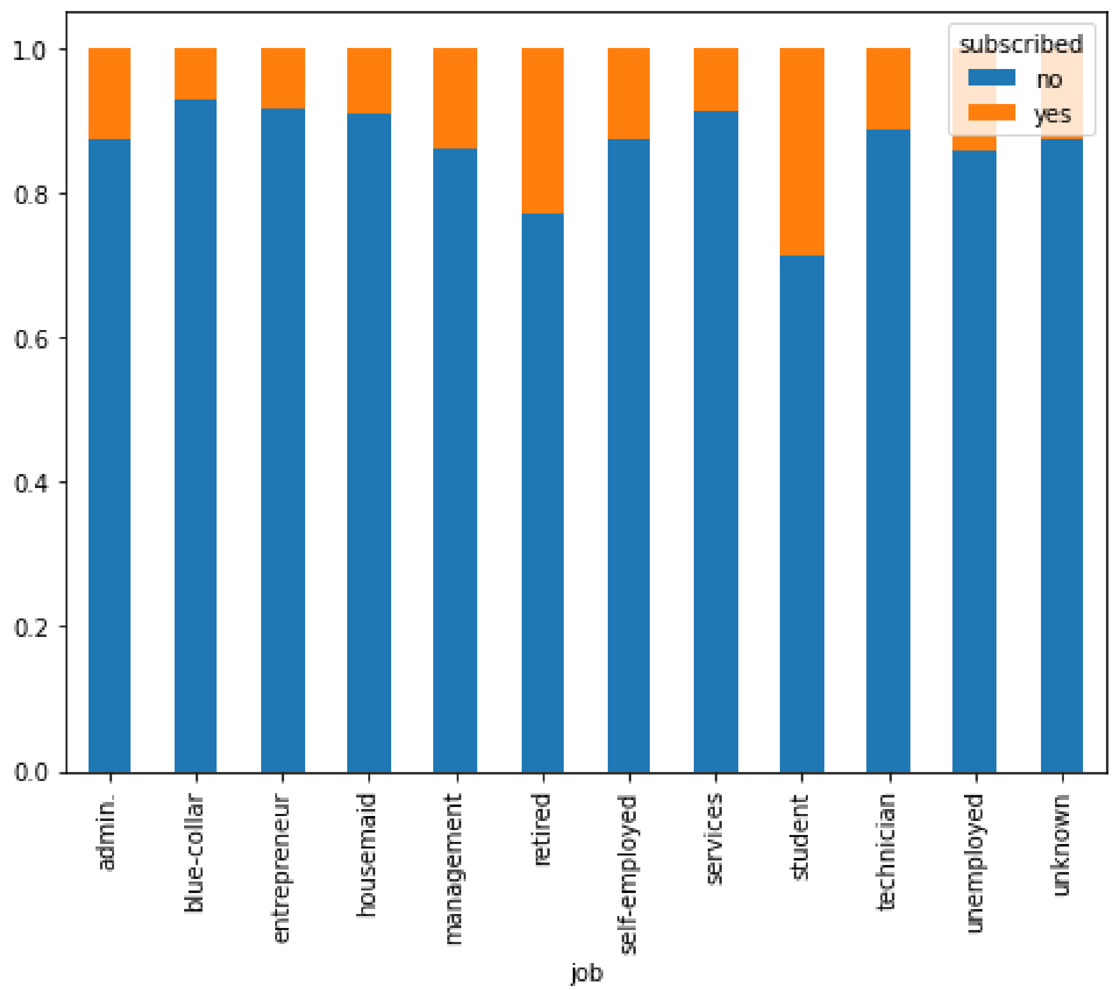
[29]:
```
job = pd.crosstab(df['job'],df['subscribed'])
job_norm = job.div(job.sum(1).astype(float), axis=0)
job_norm
```

[29]:

```
subscribed           no       yes
job
admin.         0.875516  0.124484
blue-collar    0.928530  0.071470
entrepreneur   0.915675  0.084325
housemaid      0.909611  0.090389
management     0.860973  0.139027
retired        0.770013  0.229987
self-employed  0.875334  0.124666
services       0.912504  0.087496
student        0.713386  0.286614
technician     0.888072  0.111928
unemployed     0.857459  0.142541
unknown        0.873786  0.126214
```

[30]:

```
job_norm.plot.bar(stacked=True,figsize=(8,6));
```



From the above graph we can infer that students and retired people have higher chances of subscribing to a term deposit, which is surprising as students generally do not subscribe to a term deposit. The possible reason is that the number of students in the dataset is less and comparatively to other job types, more students have subscribed to a term deposit.

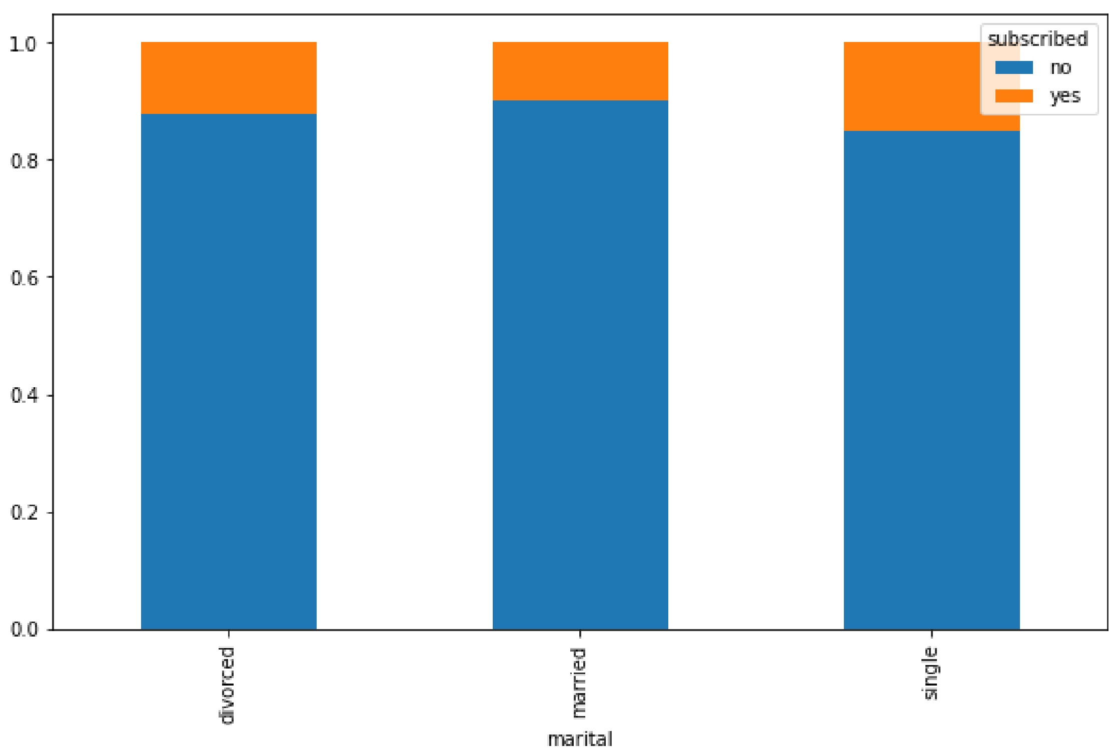
[31]:
```
#Marital status vs subscribed

pd.crosstab(df['marital'], df['subscribed'])
```

[31]:
```
subscribed     no   yes
marital
divorced     3185   445
married     17176  1919
single       7571  1351
```

[32]:
```
marital = pd.crosstab(df['marital'], df['subscribed'])
marital_norm = marital.div(marital.sum(1).astype(float), axis=0)
marital_norm
```

[32]:
```
subscribed        no       yes
marital
divorced    0.877410  0.122590
married     0.899502  0.100498
single      0.848577  0.151423
```

[33]:
```
marital_norm.plot.bar(stacked=True, figsize=(10,6));
```



From the above analysis we can infer that marital status doesn't have a major impact on the subscription to term deposits.

[34]:
```
#default vs subscription

pd.crosstab(df['default'], df['subscribed'])
```

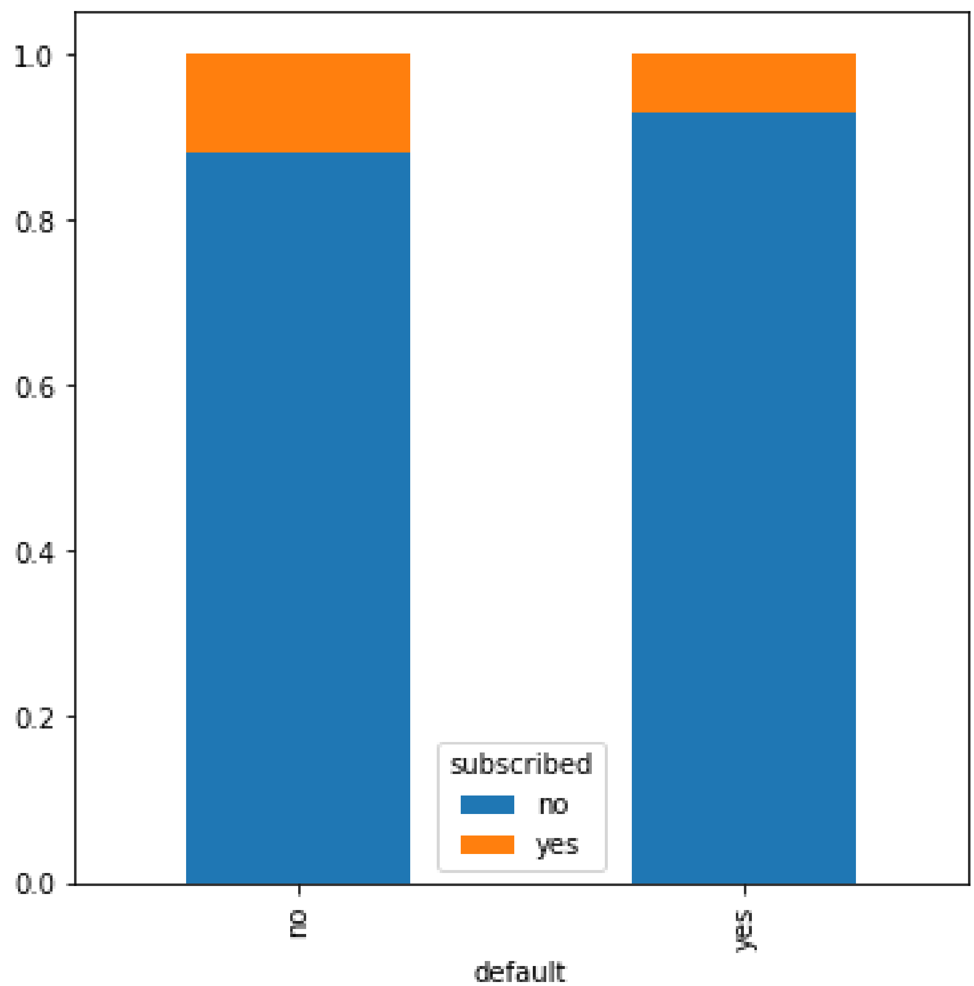
[34]:
```
subscribed     no   yes
default
no          27388  3674
yes           544    41
```

[35]:
```
dflt = pd.crosstab(df['default'], df['subscribed'])
dflt_norm = dflt.div(dflt.sum(1).astype(float), axis=0)
dflt_norm
```

[35]:
```
subscribed        no       yes
default
no          0.881720  0.118280
yes         0.929915  0.070085
```

[36]:
```
dflt_norm.plot.bar(stacked=True, figsize=(6,6))
```

[36]:
```
<AxesSubplot:xlabel='default'>
```



We can infer that clients having no previous default have slightly higher chances of subscribing to a term loan as compared to the clients who have previous default history.

[37]:
```
# Converting the target variables into 0s and 1s
df['subscribed'].replace('no', 0,inplace=True)
df['subscribed'].replace('yes', 1,inplace=True)
```

[38]:
```
df['subscribed']
```

[38]:
```
0        0
1        0
2        0
3        1
4        0
        ..
31642    0
```

```
31643    1
31644    0
31645    0
31646    1
Name: subscribed, Length: 31647, dtype: int64
```

[ ]:

```python

```

[ ]:

```python

```

2 Model Building

[40]:

```python
df.columns
```

[40]:

```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

[82]:

```python
y = df['subscribed']
X = df[[ 'job','age','education','contact','loan','housing','duration',␣
 ↪'pdays','previous']]
```

[83]:

```python
X = pd.get_dummies(X)
X.head()
```

[83]:

```
   age  duration  pdays  previous  job_admin.  job_blue-collar  \
0   56        44     -1         0           1                0
1   31        91     -1         0           0                0
2   27       240     -1         0           0                0
3   57       867     84         3           0                0
4   31       380     -1         0           0                0

   job_entrepreneur  job_housemaid  job_management  job_retired  …  \
0                 0              0               0            0  …
1                 0              0               0            0  …
2                 0              0               0            0  …
3                 0              0               1            0  …
4                 0              0               0            0  …

   education_secondary  education_tertiary  education_unknown  \
0                    0                   0                  1
1                    1                   0                  0
2                    1                   0                  0
3                    0                   1                  0
4                    1                   0                  0
```

```
   contact_cellular  contact_telephone  contact_unknown  loan_no  loan_yes  \
0                 0                  1                0        1         0
1                 1                  0                0        1         0
2                 1                  0                0        1         0
3                 1                  0                0        1         0
4                 1                  0                0        1         0

   housing_no  housing_yes
0           1            0
1           1            0
2           0            1
3           1            0
4           0            1

[5 rows x 27 columns]
```

2.1 Split the Dataset in training set and test set

Splitting the data into train(X) and validation set such as to validate the results of our model on the validation set. keeping 20% of the dataset as our validation set and the rest as our training set.

[84]:
```
# import the required module
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

[85]:
```
# Generate the dataset

X, y= make_classification(
    n_samples=100,
    n_features=1,
    n_classes=2,
    n_clusters_per_class=1,
    flip_y=0.03,
    n_informative=1,
    n_redundant=0,
    n_repeated=0
)
print(y)
```

```
[0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0
 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0
 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0]
```

[86]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 100)
```

Now our data is ready and it's time to build our model and check its performance. Since it's a classification problem, I'll be using Logistic Regression model for this problem.

[87]:
```
len(X_train)
```

[87]: 70

[88]:
```
len(X_test)
```

[88]: 30

[89]:
```
from sklearn.linear_model import LogisticRegression
```

[90]:
```
log_reg = LogisticRegression()
log_reg.fit(X_train,y_train)
```

[90]: LogisticRegression()

[91]:
```
print(log_reg.coef_)
print(log_reg.intercept_)
```

```
[[2.84923506]]
[-0.01758398]
```

[92]:
```
y_pred = log_reg.predict(X_test)
```

[93]:
```
# display the confusion matrix
confusion_matrix(y_test, y_pred)
```

[93]:
```
array([[20,  0],
       [ 0, 10]], dtype=int64)
```

Checking the accuracy of our model

[94]:
```
from sklearn.metrics import classification_report
```

[95]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

We got an accuracy score of the validation dataset. Logistic regression has a linear decision boundary. What if our data have non linearity? We need a model that can capture this non linearity.

[96]:
```python
# visualize the data (scatter plot)
plt.scatter(X, y, c=y, cmap='rainbow')
plt.title('Scatter plot of Logistic Regression')
plt.show()
```

[ ]:
```python

```

2.1.1 Using Decision Tree algorithm for dealing with non-linearity

[126]:
```python
from sklearn.tree import DecisionTreeClassifier
```

[127]:
```python
clf = DecisionTreeClassifier()
```

[128]:
```python
clf.fit(X_train, y_train)
```

[128]:
```
DecisionTreeClassifier()
```

[129]:
```python
y_pred = clf.predict(X_test)
```

[130]:
```python
from sklearn.metrics import accuracy_score
```

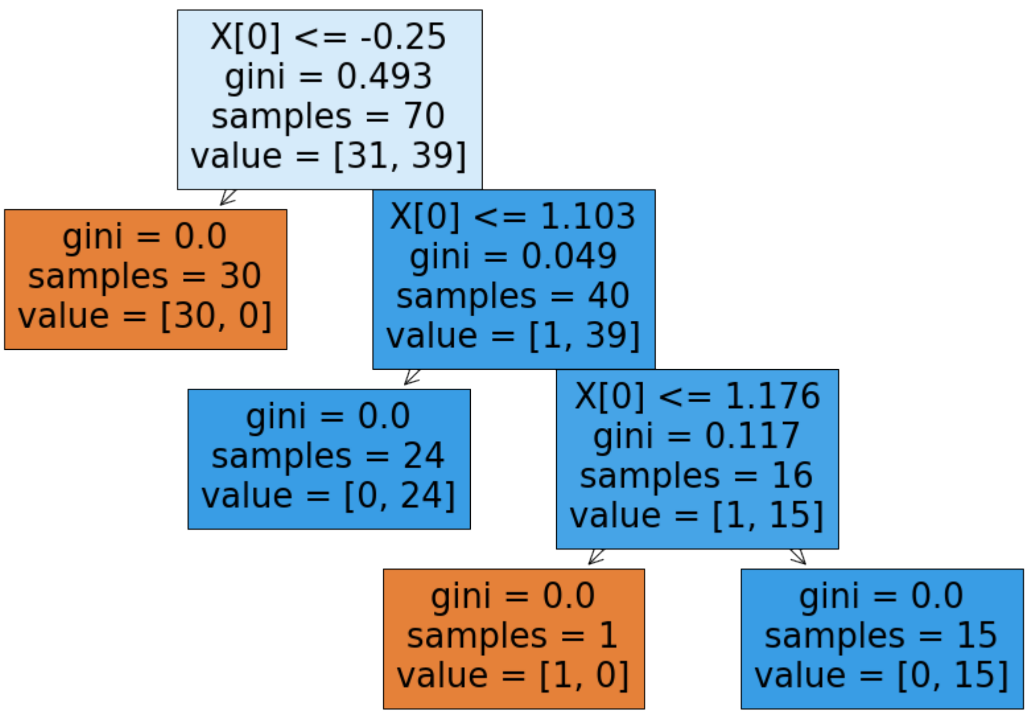
[131]:
```python
print("Training Accuracy: ", accuracy_score(y_train, clf.predict(X_train)))
print("Test Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Training Accuracy:  1.0
Test Accuracy:  0.9333333333333333
```

[132]:
```python
from sklearn import tree
```

[133]:
```python
plt.figure(figsize=(15,10))
tree.plot_tree(clf,filled=True)
```

[133]:
```
[Text(279.0, 475.65000000000003, 'X[0] <= -0.25\ngini = 0.493\nsamples =
70\nvalue = [31, 39]'),
 Text(139.5, 339.75, 'gini = 0.0\nsamples = 30\nvalue = [30, 0]'),
 Text(418.5, 339.75, 'X[0] <= 1.103\ngini = 0.049\nsamples = 40\nvalue = [1,
39]'),
 Text(279.0, 203.85000000000002, 'gini = 0.0\nsamples = 24\nvalue = [0, 24]'),
 Text(558.0, 203.85000000000002, 'X[0] <= 1.176\ngini = 0.117\nsamples =
16\nvalue = [1, 15]'),
 Text(418.5, 67.94999999999999, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(697.5, 67.94999999999999, 'gini = 0.0\nsamples = 15\nvalue = [0, 15]')]
```



[134]:
```python
print(tree.export_text(clf))
```

```
|--- feature_0 <= -0.25
|   |--- class: 0
|--- feature_0 >  -0.25
|   |--- feature_0 <= 1.10
```

```
|   |   |--- class: 1
|   |--- feature_0 >  1.10
|   |   |--- feature_0 <= 1.18
|   |   |   |--- class: 0
|   |   |--- feature_0 >  1.18
|   |   |   |--- class: 1
```

[135]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.95      0.90      0.92        30
weighted avg       0.94      0.93      0.93        30
```

[ ]:
```

```

2.2 Random forest

[106]:
```
from sklearn.ensemble import RandomForestClassifier


model = RandomForestClassifier(n_estimators=40)
model.fit(X_train, y_train)
```

[106]:
```
RandomForestClassifier(n_estimators=40)
```

[107]:
```
model.score(X_test,y_test)
```

[107]:
```
0.9333333333333333
```

[108]:
```
y_pred = model.predict(X_test)
```

[109]:
```
print(classification_report(y_test,y_pred))
```

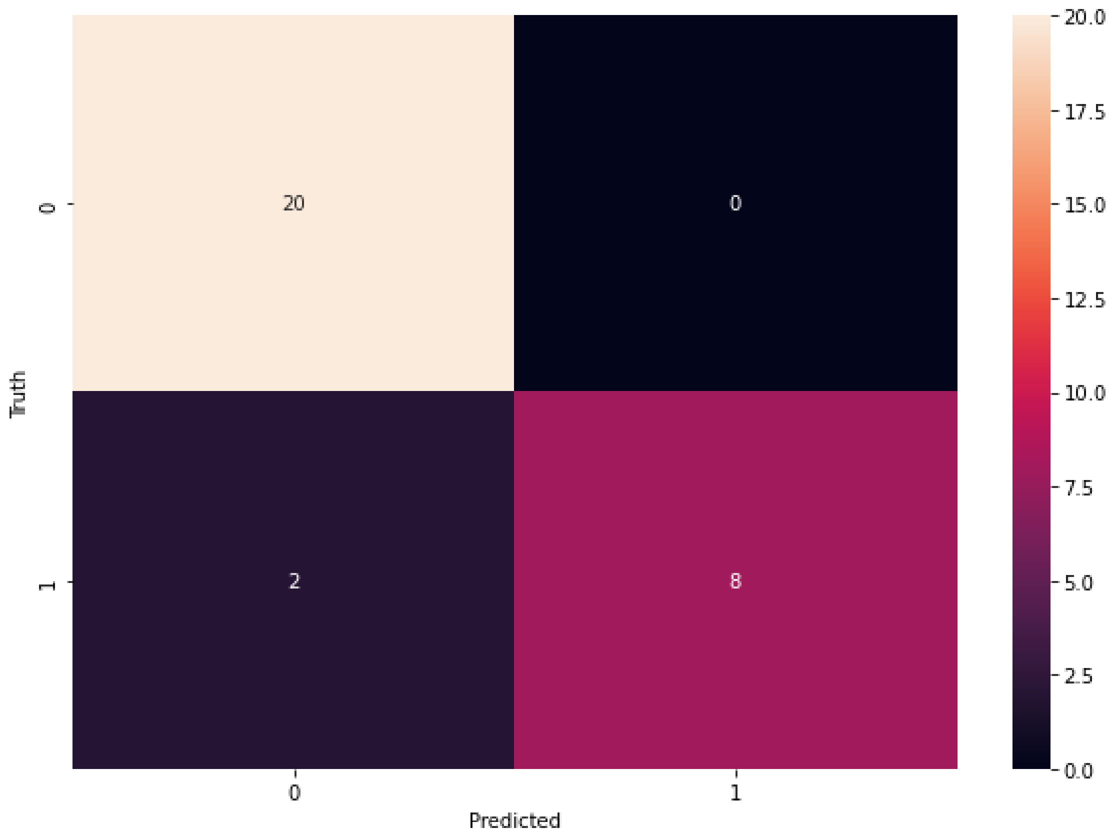
```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.95      0.90      0.92        30
weighted avg       0.94      0.93      0.93        30
```

[110]:
```
# Confusion Matrix

cm = confusion_matrix(y_test,y_pred)
cm
```

[110]:
```
array([[20,  0],
       [ 2,  8]], dtype=int64)
```

[111]:
```
%matplotlib inline

plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[111]:
```
Text(69.0, 0.5, 'Truth')
```



20 times truth is 0 and predicted also 0

8 times truth is 1 and predicted 1

2 times truth is 1 but predicted 0

3 Using KNN

[112]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[113]:
```
knn = KNeighborsClassifier(n_neighbors=1)
```

[114]:
```
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

[115]:
```
# Prediction and Evalutions

from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
```

[116]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[20  0]
 [ 2  8]]
```

[117]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.95      0.90      0.92        30
weighted avg       0.94      0.93      0.93        30
```

3.0.1 Choosing a K value

used elbow method tp pick a good K value

[118]:
```
accuracy_rate = []

#will take some time and run loop to 1 to 40
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,X,y,cv=10)
    accuracy_rate.append(score.mean())
```

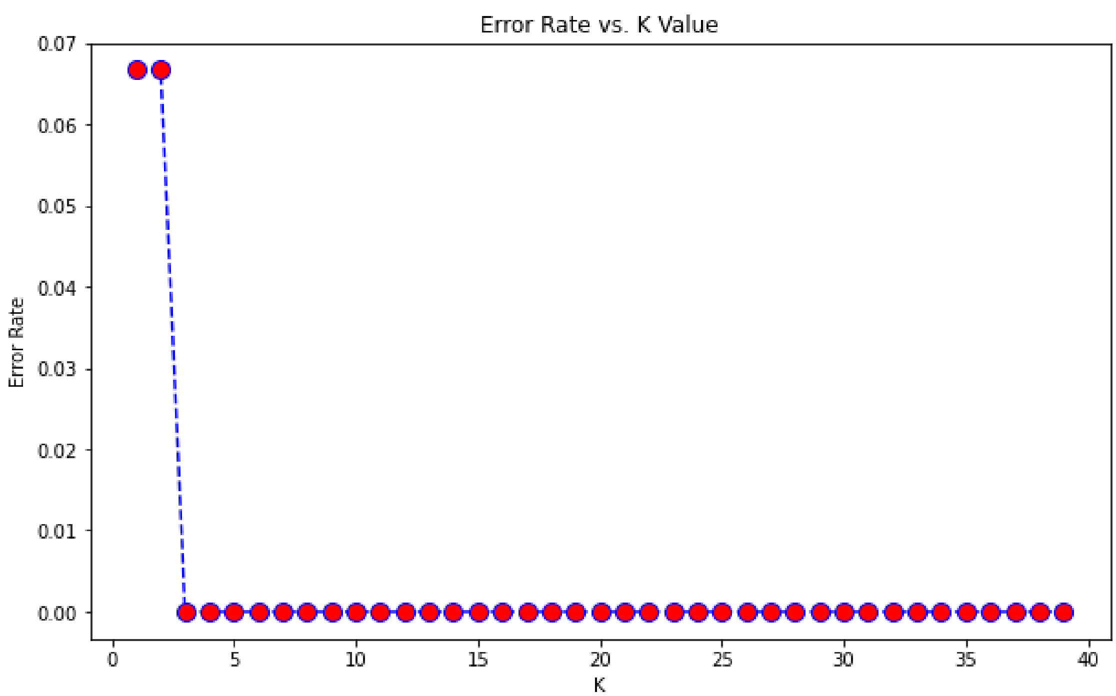
[119]:
```
error_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
```

```
    score=cross_val_score(knn,X,y,cv=10)
    error_rate.append(1-score.mean())
```

[120]:
```
error_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

[121]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate, color='blue', linestyle='dashed',␣
↪marker='o',markerfacecolor='red', markersize=10)
#plt.plot(range(1,40),accuracy_rate, color='blue', linestyle='dashed',␣
↪marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[121]:
```
Text(0, 0.5, 'Error Rate')
```



[122]:
```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=1


[[20  0]
 [ 2  8]]


              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.95      0.90      0.92        30
weighted avg       0.94      0.93      0.93        30
```

[123]:
```
# NOW WITH K=10
knn = KNeighborsClassifier(n_neighbors=10)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=10')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=10


[[20  0]
 [ 0 10]]
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[124]:

```

```

[ ]:

```

```