Interactive exercise week #9b

Liping Wu 300-958-061 11/9/2020

In this exercise we will do the following:

- Explore a dataset
- Process & clean up and visualize the dataset
- Build a logistic regression model

Pre-requisites:

1- Install Anoconda
2- We will be using a lot of Public datasets these datasets are available at https://goo.gl/zjS4C6 under a folder named "Datasets for Predictive Modelling with Python", the datasets are organized in the order of the text book chapters: Python: Advanced Predictive Analytics, chapter # 6 files are required

Steps for exploring and building a logistic regression model:

1- Open your spider IDE
2- Load the 'Bank.csv' file into a dataframe name *the dataframe data_firstname_b where first name is your first name* carry out the following activities:
   a. Display the column names
   b. Display the shape of the data frame i.e number of rows and number of columns
   c. Display the main statistics of the data
   d. Display the types of columns
   e. Display the first five records

Following is the code, *make sure you update the path to the correct path where you placed the files and update the data frame name correctly*:

```
import pandas as pd
import os
path = " D:/CentennialWu/2020Fall/COMP309Data/Assignments/Lab07Wk09"
filename = 'Bank.csv'
fullpath = os.path.join(path,filename)
data_liping_b = pd.read_csv(fullpath,sep=';')

print(data_liping_b.columns.values)
```

```
print(data_liping_b.shape)
print(data_liping_b.describe())
print(data_liping_b.dtypes)
print(data_liping_b.head(5))
```

```
   ...: print(data_liping_b.head(5))
['age' 'job' 'marital' 'education' 'default' 'housing' 'loan' 'contact'
 'month' 'day_of_week' 'duration' 'campaign' 'pdays' 'previous' 'poutcome'
 'emp.var.rate' 'cons.price.idx' 'cons.conf.idx' 'euribor3m' 'nr.employed'
 'y']
(4119, 21)
               age     duration  ...    euribor3m  nr.employed
count  4119.000000  4119.000000  ...  4119.000000  4119.000000
mean     40.113620   256.788055  ...     3.621356  5166.481695
std      10.313362   254.703736  ...     1.733591    73.667904
min      18.000000     0.000000  ...     0.635000  4963.600000
25%      32.000000   103.000000  ...     1.334000  5099.100000
50%      38.000000   181.000000  ...     4.857000  5191.000000
75%      47.000000   317.000000  ...     4.961000  5228.100000
max      88.000000  3643.000000  ...     5.045000  5228.100000

[8 rows x 10 columns]
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
   age          job  marital  ... euribor3m nr.employed   y
0   30  blue-collar  married  ...     1.313      5099.1  no
1   39     services   single  ...     4.855      5191.0  no
2   25     services  married  ...     4.962      5228.1  no
3   38     services  married  ...     4.959      5228.1  no
4   47       admin.  married  ...     4.191      5195.8  no
```

3- Explore, Check & Process the data, we need to carry out several processing steps (clean up steps) before building the model as follows:

   a. 1-change the y column from object to integer, this is the class or label column
   b. Reduce the categories of the education column
   c. Check the values of who purchased the deposit account
   d. Check the average of all the numeric columns
   e. Check the mean of all numeric columns grouped by education
   f. Plot a histogram showing purchase by education category

g. Draw a stacked bar chart of the marital status and the purchase of term deposit to see whether this can be a good predictor of the outcome
h. Plot the bar chart for the Frequency of Purchase against each day of the week to see whether this can be a good predictor of the outcome.
i. Repeat step h for the month
j. Plot a histogram of the age distribution

Following is the code:

Explore, Check & Process the data

1-change the y column from object to integer

```
print(data_liping_b)
data_liping_b['y']=(data_liping_b['y']=='yes').astype(int)
print(data_liping_b)
```

```
In [6]: print(data_liping_b)
   ...: data_liping_b['y']=(data_liping_b['y']=='yes').astype(int)
   ...: print(data_liping_b)
      age          job  marital  ... euribor3m nr.employed   y
0      30  blue-collar  married  ...     1.313      5099.1  no
1      39     services   single  ...     4.855      5191.0  no
2      25     services  married  ...     4.962      5228.1  no
3      38     services  married  ...     4.959      5228.1  no
4      47       admin.  married  ...     4.191      5195.8  no
...   ...          ...      ...  ...       ...         ...  ..
4114   30       admin.  married  ...     4.958      5228.1  no
4115   39       admin.  married  ...     4.959      5228.1  no
4116   27      student   single  ...     1.354      5099.1  no
4117   58       admin.  married  ...     4.966      5228.1  no
4118   34   management   single  ...     4.120      5195.8  no

[4119 rows x 21 columns]
      age          job  marital  ... euribor3m nr.employed  y
0      30  blue-collar  married  ...     1.313      5099.1  0
1      39     services   single  ...     4.855      5191.0  0
2      25     services  married  ...     4.962      5228.1  0
3      38     services  married  ...     4.959      5228.1  0
4      47       admin.  married  ...     4.191      5195.8  0
...   ...          ...      ...  ...       ...         ... ..
4114   30       admin.  married  ...     4.958      5228.1  0
4115   39       admin.  married  ...     4.959      5228.1  0
4116   27      student   single  ...     1.354      5099.1  0
4117   58       admin.  married  ...     4.966      5228.1  0
4118   34   management   single  ...     4.120      5195.8  0

[4119 rows x 21 columns]
```

#2- Reduce the categories of the education column

```
print(data_liping_b['education'].unique())
```

```
In [14]: print(data_liping_b['education'].unique())
['basic.9y' 'high.school' 'university.degree' 'professional.course'
 'basic.6y' 'basic.4y' 'unknown' 'illiterate']
```

```
import numpy as np
data_liping_b['education']=np.where(data_liping_b['education'] =='basic.9y', 'Basic', data_liping_b['education'])
data_liping_b['education']=np.where(data_liping_b['education'] =='basic.6y', 'Basic', data_liping_b['education'])
```

data_liping_b['education']=np.where(data_liping_b['education'] =='basic.4y', 'Basic', data_liping_b['education'])
data_liping_b['education']=np.where(data_liping_b['education'] =='university.degree', 'University Degree', data_liping_b['education'])
data_liping_b['education']=np.where(data_liping_b['education'] =='professional.course', 'Professional Course', data_liping_b['education'])
data_liping_b['education']=np.where(data_liping_b['education'] =='high.school', 'High School', data_liping_b['education'])
data_liping_b['education']=np.where(data_liping_b['education'] =='illiterate', 'Illiterate', data_liping_b['education'])
data_liping_b['education']=np.where(data_liping_b['education'] =='unknown', 'Unknown', data_liping_b['education'])
# After cleaning
print(data_liping_b['education'].unique())

```
   ...: print(data_liping_b['education'].unique())
['Basic' 'High School' 'University Degree' 'Professional Course' 'Unknown'
 'Illiterate']
```

#Check the values of who purchased the deposit account
print(data_liping_b['y'].value_counts())

```
In [16]: print(data_liping_b['y'].value_counts())
0    3668
1     451
Name: y, dtype: int64
```

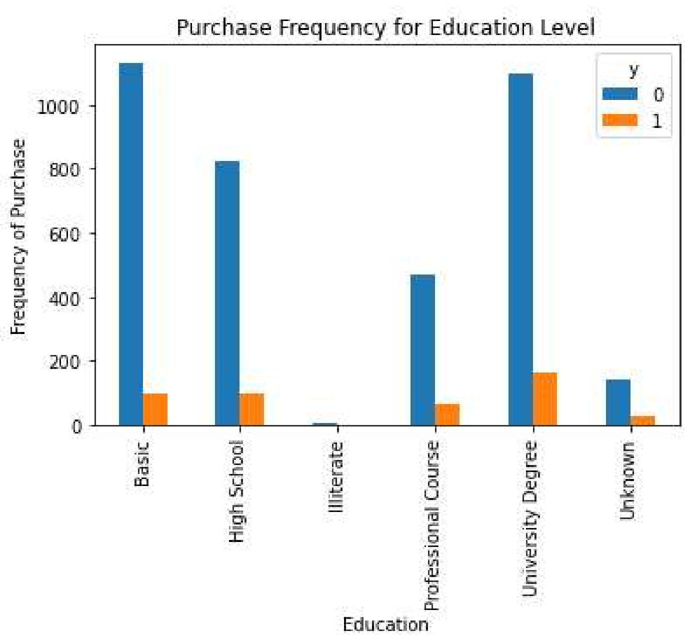
#Check the average of all the numeric columns
pd.set_option('display.max_columns',100)
print(data_liping_b.groupby('y').mean())

```
In [18]: pd.set_option('display.max_columns',100)
    ...: print(data_liping_b.groupby('y').mean())
         age   duration  campaign       pdays  previous  emp.var.rate  \
y
0  39.895311  219.40976  2.605780  982.763086  0.141767      0.240185
1  41.889135  560.78714  1.980044  778.722838  0.585366     -1.177384

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed
y
0       93.599677     -40.586723   3.802826  5175.502072
1       93.417268     -39.786475   2.145448  5093.118625
```

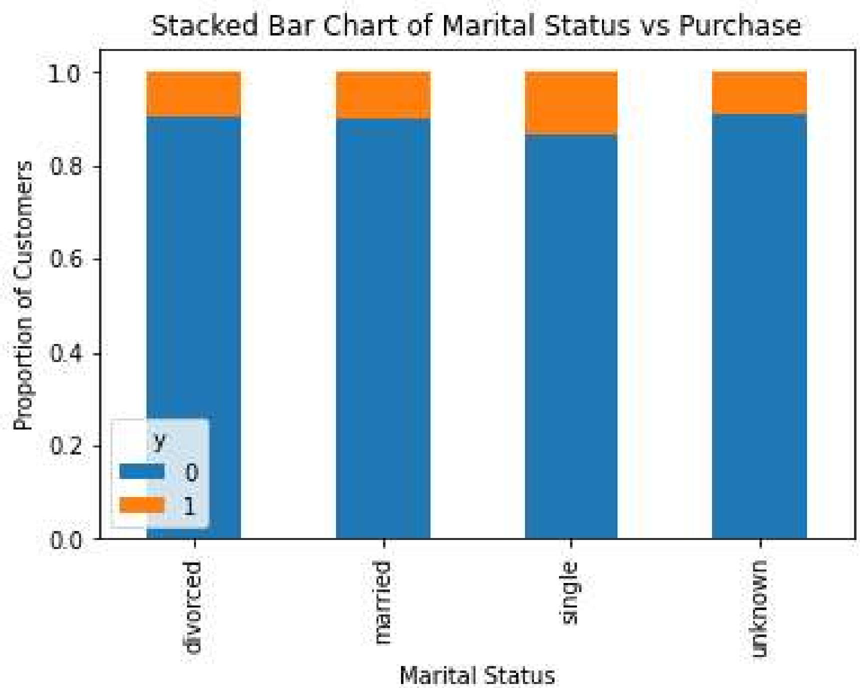
#Check the mean of all numeric columns grouped by education
print(data_liping_b.groupby('education').mean())

```
In [19]: print(data_liping_b.groupby('education').mean())
                          age    duration  campaign       pdays  previous  \
education
Basic               42.337124  253.898457  2.429732  978.815597  0.149472
High School         38.097720  258.534202  2.630836  958.022801  0.206298
Illiterate          42.000000  146.000000  4.000000  999.000000  0.000000
Professional Course 40.207477  278.816822  2.512150  958.211215  0.194393
University Degree   39.017405  247.707278  2.583070  947.900316  0.207278
Unknown             42.826347  267.281437  2.538922  939.700599  0.263473

                    emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
education
Basic                   0.237368       93.658600     -41.120552   3.775701
High School            -0.002497       93.564314     -40.995765   3.511732
Illiterate             -2.900000       92.201000     -31.400000   0.834000
Professional Course     0.163925       93.599630     -40.127664   3.701426
University Degree      -0.009731       93.499109     -39.830063   3.547132
Unknown                -0.074251       93.637455     -39.487425   3.410174

                    nr.employed         y
education
Basic               5174.133144  0.079610
High School         5163.212595  0.105320
Illiterate          5076.200000  0.000000
Professional Course 5167.595140  0.121495
University Degree   5163.023180  0.130538
Unknown             5151.260479  0.155689
```

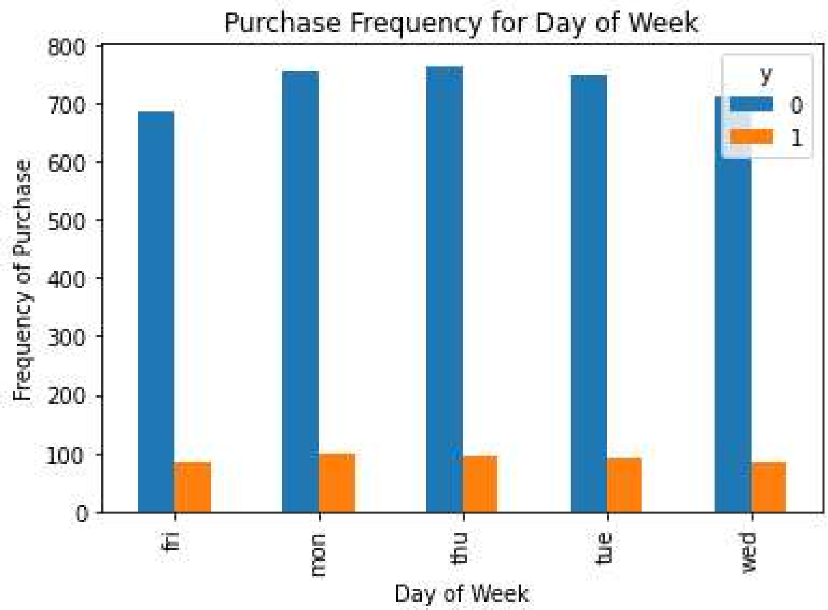
#Plot a histogram showing purchase by education category
import matplotlib.pyplot as plt
pd.crosstab(data_liping_b.education,data_liping_b.y)
pd.crosstab(data_liping_b.education,data_liping_b.y).plot(kind='bar')
plt.title('Purchase Frequency for Education Level')
plt.xlabel('Education')
plt.ylabel('Frequency of Purchase')



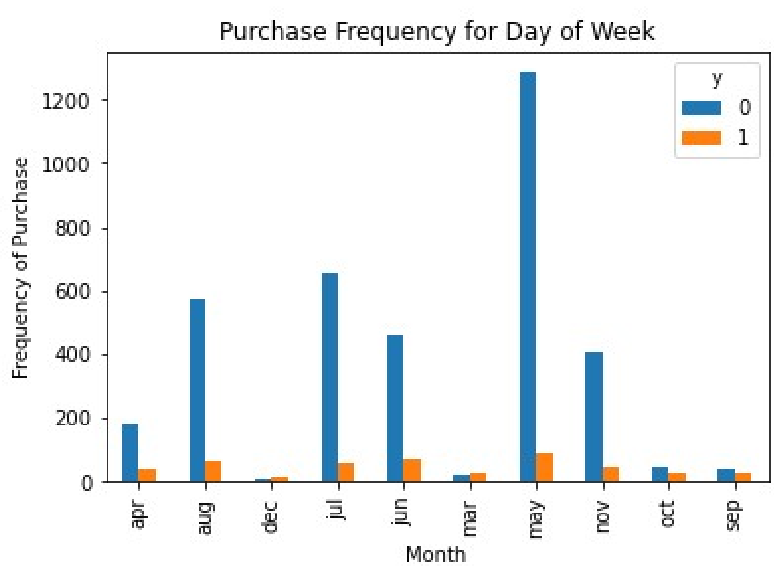
#draw a stacked bar chart of the marital status and the purchase of term deposit to see whether this can be a good predictor of the outcome
table=pd.crosstab(data_liping_b.marital,data_liping_b.y)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Marital Status vs Purchase')
plt.xlabel('Marital Status')
plt.ylabel('Proportion of Customers')



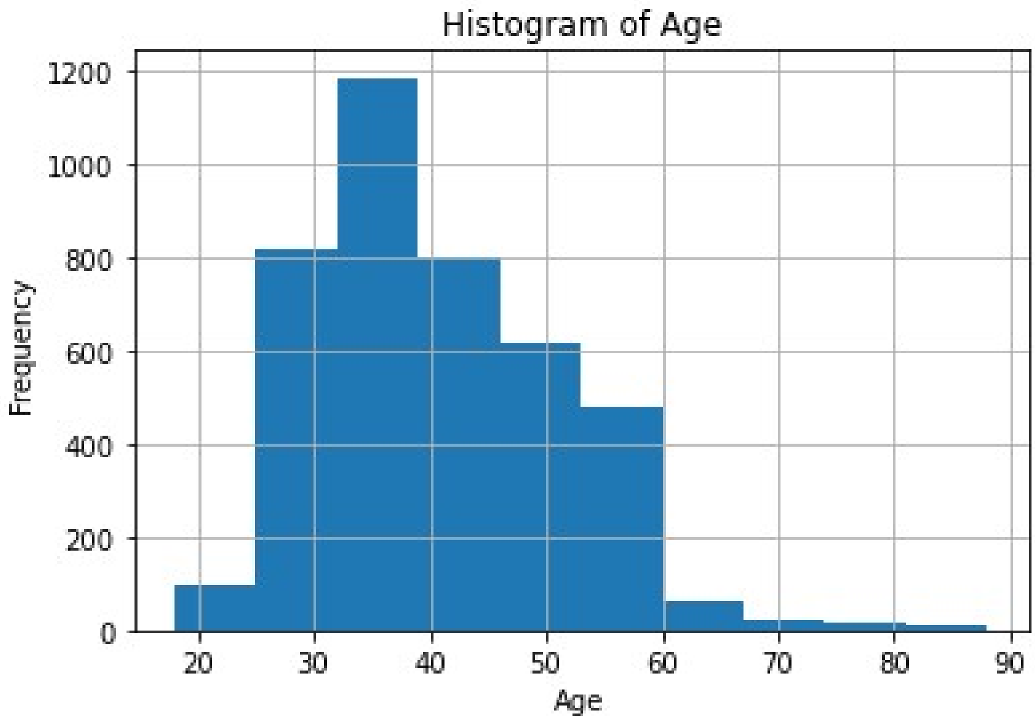
#plot the bar chart for the Frequency of Purchase against each day of the week to see whether this can be a good predictor of the outcome
pd.crosstab(data_liping_b.day_of_week,data_liping_b.y).plot(kind='bar')
plt.title('Purchase Frequency for Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Frequency of Purchase')




```
#Repeat for the month
pd.crosstab(data_liping_b.month,data_liping_b.y).plot(kind='bar')
plt.title('Purchase Frequency for Day of Week')
plt.xlabel('Month')
plt.ylabel('Frequency of Purchase')
```




```
#Plot a histogram of the age distribution
data_liping_b.age.hist()
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
```



4- Deal with the categorical variables, as follows:

   a. Create the dummy variables , use a loop
   b. Remove the original columns
   c. Prepare the data for the model build as X (inputs, predictor) and Y(output, predicted)

Following is the code.

```
cat_vars=['job','marital','education','default','housing','loan','contact','month','day_of_week','pou
tcome']
for var in cat_vars:
  cat_list='var'+'_'+var
  print(cat_list)
  cat_list = pd.get_dummies(data_liping_b[var], prefix=var)
  print(cat_list)
```

cat_list.describe

```
In [131]: cat_list.describe
Out[131]:
<bound method NDFrame.describe of       poutcome_failure  poutcome_nonexistent  poutcome_success
0                    0                     1                 0
1                    0                     1                 0
2                    0                     1                 0
3                    0                     1                 0
4                    0                     1                 0
...                ...                   ...               ...
4114                 0                     1                 0
4115                 0                     1                 0
4116                 1                     0                 0
4117                 0                     1                 0
4118                 0                     1                 0

[4119 rows x 3 columns]>
```

print(data_liping_b)

```
         poutcome  emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
0     nonexistent          -1.8          92.893          -46.2      1.313
1     nonexistent           1.1          93.994          -36.4      4.855
2     nonexistent           1.4          94.465          -41.8      4.962
3     nonexistent           1.4          94.465          -41.8      4.959
4     nonexistent          -0.1          93.200          -42.0      4.191
...           ...           ...             ...            ...        ...
4114  nonexistent           1.4          93.918          -42.7      4.958
4115  nonexistent           1.4          93.918          -42.7      4.959
4116      failure          -1.8          92.893          -46.2      1.354
4117  nonexistent           1.4          93.444          -36.1      4.966
4118  nonexistent          -0.1          93.200          -42.0      4.120

      nr.employed  y
0          5099.1  0
1          5191.0  0
2          5228.1  0
3          5228.1  0
4          5195.8  0
...           ... ..
4114       5228.1  0
4115       5228.1  0
4116       5099.1  0
4117       5228.1  0
4118       5195.8  0

[4119 rows x 21 columns]
```

#data_liping_b1=data_liping_b.join(cat_list,how= 'inner') need to fix by join

data_liping_b = pd.merge(data_liping_b, cat_list,left_index=True, right_index=True, how='outer')

to observe the datachange

data_liping_b.columns.values

data_liping_b.describe()

data_liping_b.head()

data_liping_b.describe

```
      nr.employed  y  poutcome_failure  poutcome_nonexistent  poutcome_success
0          5099.1  0                 0                     1                 0
1          5191.0  0                 0                     1                 0
2          5228.1  0                 0                     1                 0
3          5228.1  0                 0                     1                 0
4          5195.8  0                 0                     1                 0
...           ... ..               ...                   ...               ...
4114       5228.1  0                 0                     1                 0
4115       5228.1  0                 0                     1                 0
4116       5099.1  0                 1                     0                 0
4117       5228.1  0                 0                     1                 0
4118       5195.8  0                 0                     1                 0

[4119 rows x 24 columns]>
```

```
# 2- Remove the original columns
cat_vars=['job','marital','education','default','housing','loan','contact','month','day_of_week','pou
tcome']
data_liping_b_vars=data_liping_b.columns.values.tolist()
to_keep=[i for i in data_liping_b_vars if i not in cat_vars]
data_liping_b_final=data_liping_b[to_keep]
data_liping_b_final.columns.values
```

```
In [135]: data_liping_b_final.columns.values
Out[135]:
array(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y',
       'poutcome_failure', 'poutcome_nonexistent', 'poutcome_success'],
      dtype=object)
```

```
data_liping_b_final.describe
```

```
In [136]: data_liping_b_final.describe
Out[136]:
<bound method NDFrame.describe of       age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0      30       487         2    999         0          -1.8          92.893
1      39       346         4    999         0           1.1          93.994
2      25       227         1    999         0           1.4          94.465
3      38        17         3    999         0           1.4          94.465
4      47        58         1    999         0          -0.1          93.200
...   ...       ...       ...    ...       ...           ...             ...
4114   30        53         1    999         0           1.4          93.918
4115   39       219         1    999         0           1.4          93.918
4116   27        64         2    999         1          -1.8          92.893
4117   58       528         1    999         0           1.4          93.444
4118   34       175         1    999         0          -0.1          93.200

      cons.conf.idx  euribor3m  nr.employed  y  poutcome_failure  \
0             -46.2      1.313       5099.1  0                 0
1             -36.4      4.855       5191.0  0                 0
2             -41.8      4.962       5228.1  0                 0
3             -41.8      4.959       5228.1  0                 0
4             -42.0      4.191       5195.8  0                 0
...             ...        ...          ... ..               ...
4114          -42.7      4.958       5228.1  0                 0
4115          -42.7      4.959       5228.1  0                 0
4116          -46.2      1.354       5099.1  0                 1
4117          -36.1      4.966       5228.1  0                 0
4118          -42.0      4.120       5195.8  0                 0

      poutcome_nonexistent  poutcome_success
0                        1                 0
1                        1                 0
2                        1                 0
3                        1                 0
4                        1                 0
...                    ...               ...
4114                     1                 0
4115                     1                 0
4116                     0                 0
4117                     1                 0
4118                     1                 0

[4119 rows x 14 columns]>
```

3- Prepare the data for the model build as X (inputs, predictor) and Y(output, predicted)

data_liping_b_final_vars=data_liping_b_final.columns.values.tolist()

Y=['y']

X=[i for i in data_liping_b_final_vars if i not in Y]

type(Y)

type(X)

```
In [137]: data_liping_b_final_vars=data_liping_b_final.columns.values.tolist()

In [138]: Y=['y']
     ...: X=[i for i in data_liping_b_final_vars if i not in Y ]

In [139]: type(Y)
     ...: type(X)
Out[139]: list

In [140]: data_liping_b_final_vars=data_liping_b_final.columns.values.tolist()
     ...: Y=['y']
     ...: X=[i for i in data_liping_b_final_vars if i not in Y ]
     ...: type(Y)
     ...: type(X)
Out[140]: list
```

5- Carryout feature selection and update the data, as follows:

a. Carry out feature selection using the REF module from sklearn.model_selection to select only 12 feature

b. Update X and Y to reflect only 12 features

Following is the code:

```
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
rfe = RFE(model, 12)
rfe = rfe.fit(data_liping_b_final[X],data_liping_b_final[Y] )
print(rfe.support_)
print(rfe.ranking_)
for r in zip(rfe.support_, rfe.ranking_ ):
   print(r)
```

```
In [144]: print(rfe.support_)
[ True  True  True False  True  True  True  True  True  True  True  True
  True]

In [145]: print(rfe.ranking_)
[1 1 1 2 1 1 1 1 1 1 1 1 1]

In [146]: for r in zip(rfe.support_, rfe.ranking_ ):
     ...:     print(r)
(True, 1)
(True, 1)
(True, 1)
(False, 2)
(True, 1)
(True, 1)
(True, 1)
(True, 1)
(True, 1)
(True, 1)
(True, 1)
(True, 1)
(True, 1)
```

```
#2- Update X and Y with selected features
cols=['previous', 'euribor3m',  'poutcome_success', 'poutcome_failure']
data_liping_b_final.columns.values
X=data_liping_b_final[cols]
Y=data_liping_b_final['y']
type(Y)
type(X)
```

```
In [154]: cols=['previous', 'euribor3m',  'poutcome_success', 'poutcome_failure']
     ...: data_liping_b_final.columns.values
Out[154]:
array(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y',
       'poutcome_failure', 'poutcome_nonexistent', 'poutcome_success'],
      dtype=object)

In [155]: X=data_liping_b_final[cols]

In [156]: Y=data_liping_b_final['y']

In [157]: type(Y)
     ...: type(X)
Out[157]: pandas.core.frame.DataFrame
```

6- Build the logistic regression model as follows:

a. Split the data into 70%training and 30% for testing
b. Build the model using “sklearn linear_model.LogisticRegression”
c. Fit the training data
d. Validate the parameters and check model accuracy

Following is the code*:*

```
#1- split the data into 70%training and 30% for testing, note  added the solver to avoid warnings
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
X_train.describe     #  [2883 rows x 4 columns]>
Y_train.describe   # Name: y, Length: 2883, dtype: int32>
X_test.describe
Y_test.describe
```

```
In [166]: X_train.describe
Out[166]:
<bound method NDFrame.describe of       previous  euribor3m  poutcome_success  poutcome_failure
4070         0      4.021                 0                 0
1240         0      4.856                 0                 0
349          0      4.963                 0                 0
3706         0      4.866                 0                 0
4043         0      4.857                 0                 0
...        ...        ...               ...               ...
1033         0      4.961                 0                 0
3264         0      4.961                 0                 0
1653         0      0.879                 0                 0
2607         0      4.962                 0                 0
2732         0      4.966                 0                 0

[2883 rows x 4 columns]>

In [167]: Y_train.describe
Out[167]:
<bound method NDFrame.describe of 4070    0
1240    0
349     0
3706    0
4043    0
       ..
1033    0
3264    0
1653    1
2607    0
2732    1
Name: y, Length: 2883, dtype: int32>
```

```
In [168]: X_test.describe
Out[168]:
<bound method NDFrame.describe of       previous  euribor3m  poutcome_success  poutcome_failure
3754         0      4.961                 0                 0
45           0      1.327                 0                 0
2774         0      4.865                 0                 0
1170         0      4.958                 0                 0
4042         0      4.120                 0                 0
...        ...        ...               ...               ...
1728         0      4.958                 0                 0
1460         0      4.960                 0                 0
1363         0      0.884                 0                 0
1898         0      4.857                 0                 0
3519         1      1.726                 1                 0

[1236 rows x 4 columns]>

In [169]: Y_test.describe
Out[169]:
<bound method NDFrame.describe of 3754    0
45      0
2774    0
1170    0
4042    0
       ..
1728    0
1460    0
1363    0
1898    0
3519    1
Name: y, Length: 1236, dtype: int32>
```

```
# 2-Let us build the model and validate the parameters
from sklearn import linear_model
from sklearn import metrics
clf1 = linear_model.LogisticRegression(solver='lbfgs')
clf1.fit(X_train, Y_train)
#3- Run the test data against the new model
probs = clf1.predict_proba(X_test)
print(probs)
predicted = clf1.predict(X_test)
print (predicted)
```

```
In [175]: print(probs)
[[0.95653402 0.04346598]
 [0.8027594  0.1972406 ]
 [0.95464216 0.04535784]
 ...
 [0.76814958 0.23185042]
 [0.95448101 0.04551899]
 [0.47625222 0.52374778]]
```

```
#4-Check model accuracy
print (metrics.accuracy_score(Y_test, predicted))
```

```
In [177]: print (metrics.accuracy_score(Y_test, predicted))
0.9037216828478964
```

7- To avoid sampling bias run cross validation for 10 times, as follows:
   a. Use the cross_val_score module from sklearn.model_selection and set the parameters
   b. Save the results of each run in scores
   c. Produce the mean

Following is the code:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(linear_model.LogisticRegression(solver='lbfgs'), X, Y,
scoring='accuracy', cv=10)
print (scores)
print (scores.mean())
```

```
In [178]: from sklearn.model_selection import cross_val_score
     ...: scores = cross_val_score(linear_model.LogisticRegression(solver='lbfgs'), X, Y,
scoring='accuracy', cv=10)
     ...: print (scores)
     ...: print (scores.mean())
[0.91990291 0.90048544 0.8907767  0.89805825 0.90533981 0.90533981
 0.89320388 0.90048544 0.89320388 0.90754258]
0.901433869558028
```

8- Generate the confusion matrix as follows:

a. Prepare two arrays one for the predicted values Y_P and one for actual values Y_A of the test. For the predicted use a threshold of 0.05, this means if the probability is higher than 0.05 the model will classify the instance as 1 and if it is lower than 0.05 it will be classified as 0.

b. Use the confusion_matrix option from the sklearn.metrics module to generate the matrix

Following is the code:

```
prob=probs[:,1]
prob_df=pd.DataFrame(prob)
prob_df['predict']=np.where(prob_df[0]>=0.05,1,0)
import numpy as np
Y_A =Y_test.values
Y_P = np.array(prob_df['predict'])
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(Y_A, Y_P)
print (confusion_matrix)
```

```
In [179]: prob=probs[:,1]
     ...: prob_df=pd.DataFrame(prob)
     ...: prob_df['predict']=np.where(prob_df[0]>=0.05,1,0)
     ...: import numpy as np
     ...: Y_A =Y_test.values
     ...: Y_P = np.array(prob_df['predict'])
     ...: from sklearn.metrics import confusion_matrix
     ...: confusion_matrix = confusion_matrix(Y_A, Y_P)
     ...: print (confusion_matrix)
[[715 399]
 [ 35  87]]
```