**Understanding Logistic Regression in Python**

Classification techniques are an essential part of machine learning and data mining applications. Approximately 70% of problems in Data Science are classification problems. There are lots of classification problems that are available, but the logistics regression is common and is a useful regression method for solving the binary classification problem. Another category of classification is Multinomial classification, which handles the issues where multiple classes are present in the target variable. For example, IRIS dataset a very famous example of multi-class classification. Other examples are classifying article/blog/document category.

Logistic Regression can be used for various classification problems such as spam detection. Diabetes prediction, if a given customer will purchase a particular product or will they churn another competitor, whether the user will click on a given advertisement link or not, and many more examples are in the bucket.

Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Its basic fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.

### Logistic Regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence.

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

### Linear Regression Vs. Logistic Regression

Linear regression gives you a continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. Example's of the discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn. Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.



### Maximum Likelihood Estimation Vs. Least Square Method

The MLE is a "likelihood" maximization method, while OLS is a distance-minimizing approximation method. Maximizing the likelihood function determines the parameters that are most likely to produce the observed data. From a statistical point of view, MLE sets the mean and variance as parameters in determining the specific parametric values for a given model. This set of parameters can be used for predicting the data needed in a normal distribution.

Ordinary Least squares estimates are computed by fitting a regression line on given data points that has the minimum sum of the squared deviations (least square error). Both are used to estimate the parameters of a linear regression model. MLE assumes a joint probability mass function, while OLS doesn't require any stochastic assumptions for minimizing distance.

**Logistic Regression** is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

### Logistic Regression Assumptions

* Binary logistic regression requires the dependent variable to be binary.
* For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
* Only the meaningful variables should be included.
* The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
* The independent variables are linearly related to the log odds.
* Logistic regression requires quite large sample sizes.

Keeping the above assumptions in mind, let’s look at our dataset.

### Data

The dataset comes from the UCI Machine Learning repository, and it is related to direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y). The dataset can be downloaded from here.

import pandas as pd  
import numpy as np  
from sklearn import preprocessing  
import matplotlib.pyplot as plt   
plt.rc("font", size=14)  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
import seaborn as sns  
sns.set(style="white")  
sns.set(style="whitegrid", color\_codes=True)

The dataset provides the bank customers’ information. It includes 41,188 records and 21 fields.

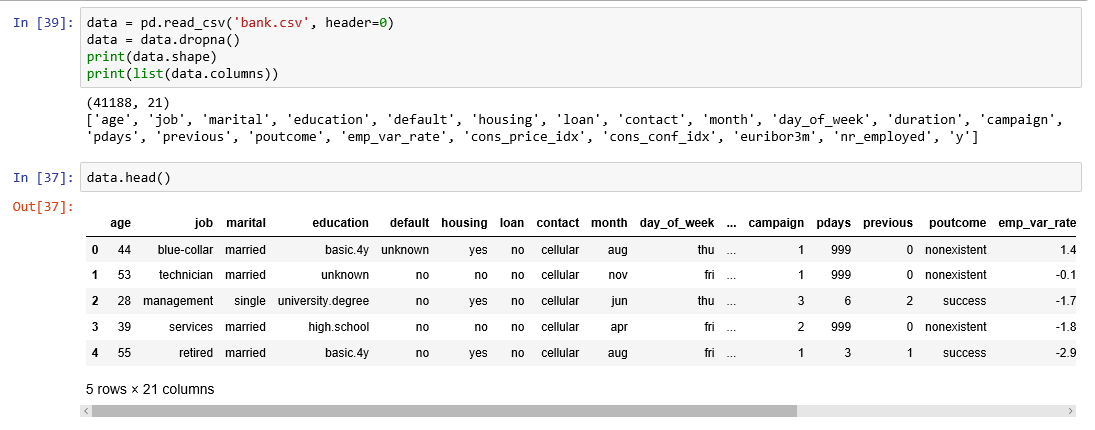


Figure 1

**Input variables**

1. age (numeric)
2. job : type of job (categorical: “admin”, “blue-collar”, “entrepreneur”, “housemaid”, “management”, “retired”, “self-employed”, “services”, “student”, “technician”, “unemployed”, “unknown”)
3. marital : marital status (categorical: “divorced”, “married”, “single”, “unknown”)
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: has housing loan? (categorical: “no”, “yes”, “unknown”)
7. loan: has personal loan? (categorical: “no”, “yes”, “unknown”)
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 the week (categorical: “mon”, “tue”, “wed”, “thu”, “fri”)
11. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y=’no’). The duration is not known before a call is performed, also, after the end of the call, y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model
12. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. previous: number of contacts performed before this campaign and for this client (numeric)
15. poutcome: outcome of the previous marketing campaign (categorical: “failure”, “nonexistent”, “success”)
16. emp.var.rate: employment variation rate — (numeric)
17. cons.price.idx: consumer price index — (numeric)
18. cons.conf.idx: consumer confidence index — (numeric)
19. euribor3m: euribor 3 month rate — (numeric)
20. nr.employed: number of employees — (numeric)

**Predict variable (desired target):**

y — has the client subscribed a term deposit? (binary: “1”, means “Yes”, “0” means “No”)

The education column of the dataset has many categories and we need to reduce the categories for a better modelling. The education column has the following categories:

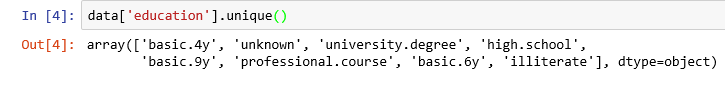


Figure 2

Let us group “basic.4y”, “basic.9y” and “basic.6y” together and call them “basic”.

data['education']=np.where(data['education'] =='basic.9y', 'Basic', data['education'])  
data['education']=np.where(data['education'] =='basic.6y', 'Basic', data['education'])  
data['education']=np.where(data['education'] =='basic.4y', 'Basic', data['education'])

After grouping, this is the columns:

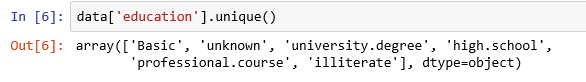


Figure 3

### Data exploration

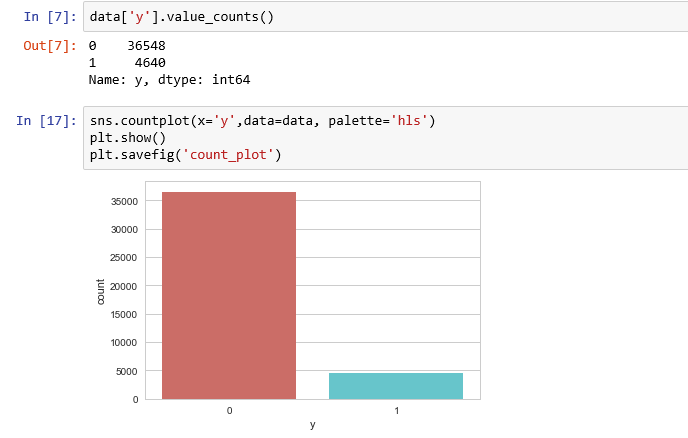


Figure 4

count\_no\_sub = len(data[data['y']==0])  
count\_sub = len(data[data['y']==1])  
pct\_of\_no\_sub = count\_no\_sub/(count\_no\_sub+count\_sub)  
print("percentage of no subscription is", pct\_of\_no\_sub\*100)  
pct\_of\_sub = count\_sub/(count\_no\_sub+count\_sub)  
print("percentage of subscription", pct\_of\_sub\*100)

**percentage of no subscription is 88.73458288821988**

**percentage of subscription 11.265417111780131**

Our classes are imbalanced, and the ratio of no-subscription to subscription instances is 89:11. Before we go ahead to balance the classes, let’s do some more exploration.

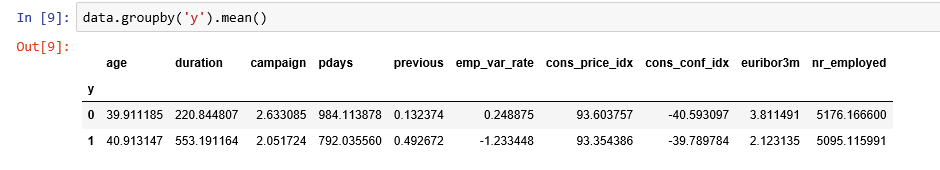


Figure 5

**Observations**:

* The average age of customers who bought the term deposit is higher than that of the customers who didn’t.
* The pdays (days since the customer was last contacted) is understandably lower for the customers who bought it. The lower the pdays, the better the memory of the last call and hence the better chances of a sale.
* Surprisingly, campaigns (number of contacts or calls made during the current campaign) are lower for customers who bought the term deposit.

We can calculate categorical means for other categorical variables such as education and marital status to get a more detailed sense of our data.

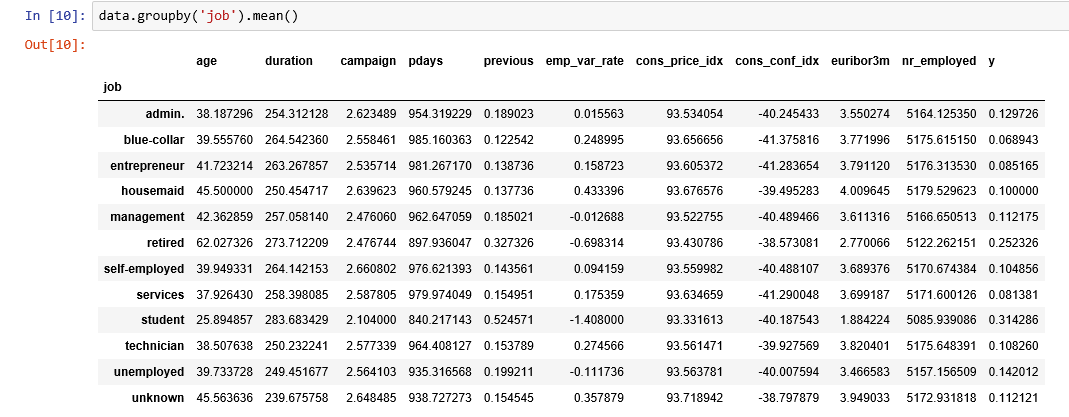


Figure 6

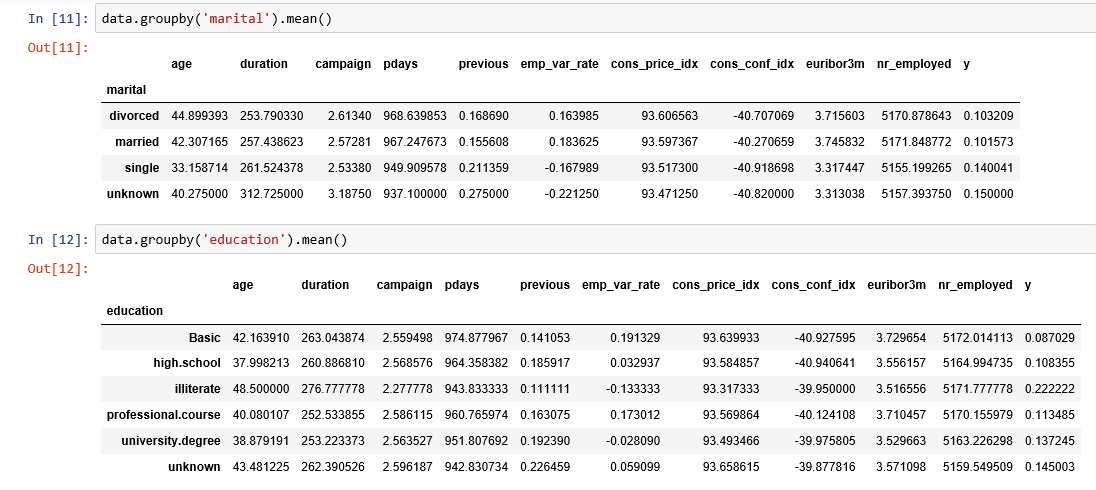


Figure 7

### Visualizations

%matplotlib inline  
pd.crosstab(data.job,data.y).plot(kind='bar')  
plt.title('Purchase Frequency for Job Title')  
plt.xlabel('Job')  
plt.ylabel('Frequency of Purchase')  
plt.savefig('purchase\_fre\_job')

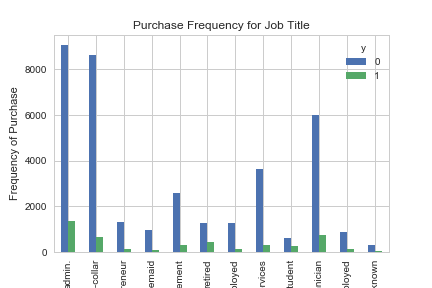


Figure 8

The frequency of purchase of the deposit depends a great deal on the job title. Thus, the job title can be a good predictor of the outcome variable.

table=pd.crosstab(data.marital,data.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')  
plt.savefig('mariral\_vs\_pur\_stack')

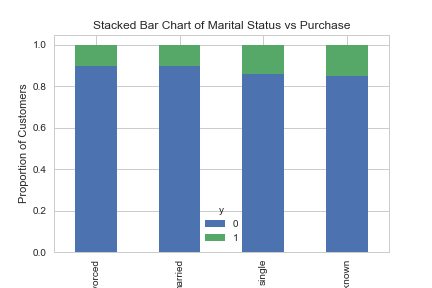


Figure 9

The marital status does not seem a strong predictor for the outcome variable.

table=pd.crosstab(data.education,data.y)  
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)  
plt.title('Stacked Bar Chart of Education vs Purchase')  
plt.xlabel('Education')  
plt.ylabel('Proportion of Customers')  
plt.savefig('edu\_vs\_pur\_stack')

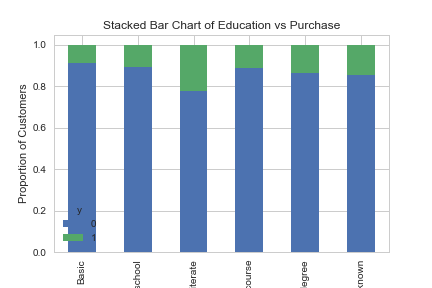


Figure 10

Education seems a good predictor of the outcome variable.

pd.crosstab(data.day\_of\_week,data.y).plot(kind='bar')  
plt.title('Purchase Frequency for Day of Week')  
plt.xlabel('Day of Week')  
plt.ylabel('Frequency of Purchase')  
plt.savefig('pur\_dayofweek\_bar')

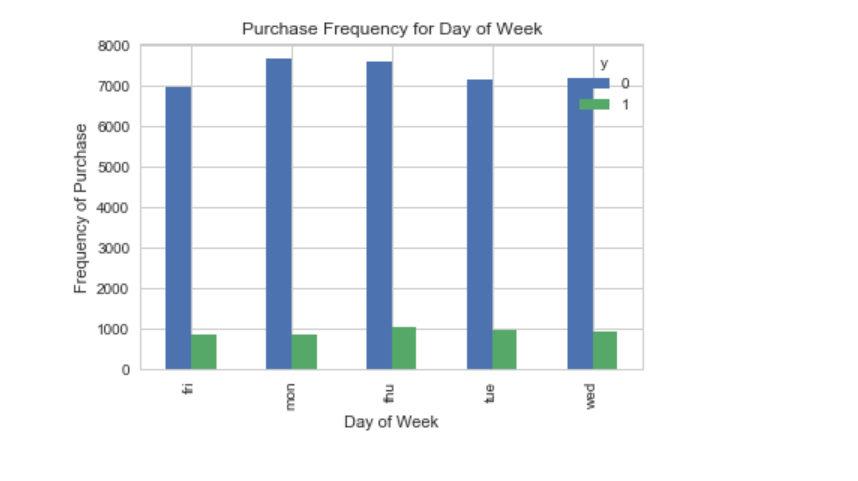


Figure 11

Day of week may not be a good predictor of the outcome.

pd.crosstab(data.month,data.y).plot(kind='bar')  
plt.title('Purchase Frequency for Month')  
plt.xlabel('Month')  
plt.ylabel('Frequency of Purchase')  
plt.savefig('pur\_fre\_month\_bar')

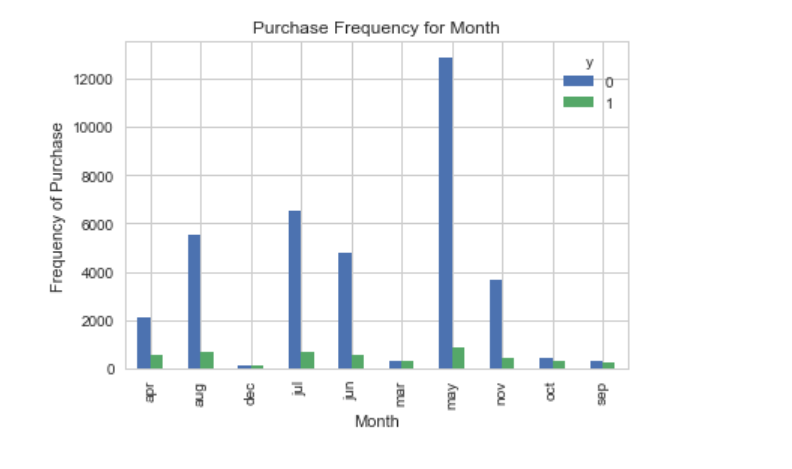


Figure 12

Month might be a good predictor of the outcome variable.

data.age.hist()  
plt.title('Histogram of Age')  
plt.xlabel('Age')  
plt.ylabel('Frequency')  
plt.savefig('hist\_age')

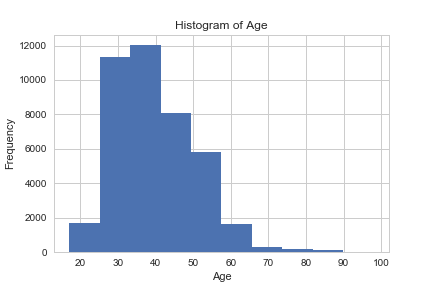


Figure 13

Most of the customers of the bank in this dataset are in the age range of 30–40.

pd.crosstab(data.poutcome,data.y).plot(kind='bar')  
plt.title('Purchase Frequency for Poutcome')  
plt.xlabel('Poutcome')  
plt.ylabel('Frequency of Purchase')  
plt.savefig('pur\_fre\_pout\_bar')

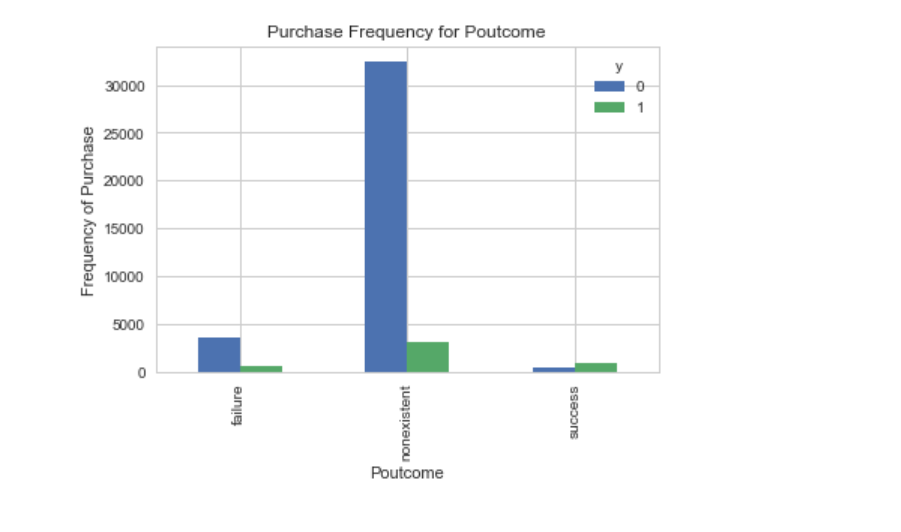


Figure 14

Poutcome seems to be a good predictor of the outcome variable.

### ****Create dummy variables****

That is variables with only two values, zero and one.

cat\_vars=['job','marital','education','default','housing','loan','contact','month','day\_of\_week','poutcome']  
for var in cat\_vars:  
 cat\_list='var'+'\_'+var  
 cat\_list = pd.get\_dummies(data[var], prefix=var)  
 data1=data.join(cat\_list)  
 data=data1

cat\_vars=['job','marital','education','default','housing','loan','contact','month','day\_of\_week','poutcome']  
data\_vars=data.columns.values.tolist()  
to\_keep=[i for i in data\_vars if i not in cat\_vars]

Our final data columns will be:

data\_final=data[to\_keep]  
data\_final.columns.values

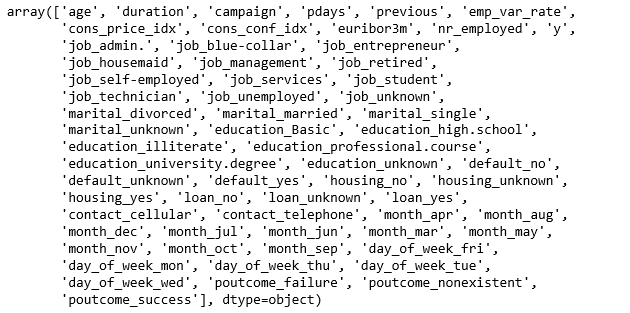


Figure 15

### ****Over-sampling using SMOTE****

With our training data created, I’ll up-sample the no-subscription using the SMOTE algorithm(Synthetic Minority Oversampling Technique). At a high level, SMOTE:

1. Works by creating synthetic samples from the minor class (no-subscription) instead of creating copies.
2. Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observations.

We are going to implement SMOTE in Python.

X = data\_final.loc[:, data\_final.columns != 'y']  
y = data\_final.loc[:, data\_final.columns == 'y']

from imblearn.over\_sampling import SMOTE

os = SMOTE(random\_state=0)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)  
columns = X\_train.columns

os\_data\_X,os\_data\_y=os.fit\_sample(X\_train, y\_train)  
os\_data\_X = pd.DataFrame(data=os\_data\_X,columns=columns )  
os\_data\_y= pd.DataFrame(data=os\_data\_y,columns=['y'])  
# we can Check the numbers of our data  
print("length of oversampled data is ",len(os\_data\_X))  
print("Number of no subscription in oversampled data",len(os\_data\_y[os\_data\_y['y']==0]))  
print("Number of subscription",len(os\_data\_y[os\_data\_y['y']==1]))  
print("Proportion of no subscription data in oversampled data is ",len(os\_data\_y[os\_data\_y['y']==0])/len(os\_data\_X))  
print("Proportion of subscription data in oversampled data is ",len(os\_data\_y[os\_data\_y['y']==1])/len(os\_data\_X))

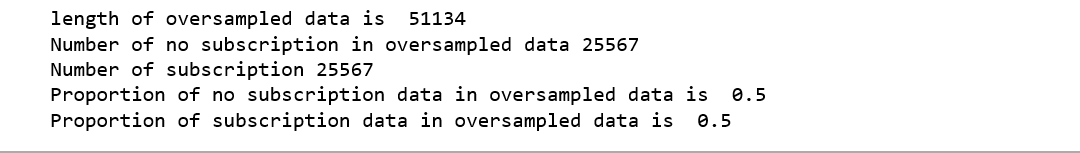


Figure 16

Now we have a perfect balanced data! You may have noticed that I over-sampled only on the training data, because by oversampling only on the training data, none of the information in the test data is being used to create synthetic observations, therefore, no information will bleed from test data into the model training.

### ****Recursive Feature Elimination****

Recursive Feature Elimination (RFE) is based on the idea to repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. The goal of RFE is to select features by recursively considering smaller and smaller sets of features.

data\_final\_vars=data\_final.columns.values.tolist()  
y=['y']  
X=[i for i in data\_final\_vars if i not in y]

from sklearn.feature\_selection import RFE  
from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression()

rfe = RFE(logreg, 20)  
rfe = rfe.fit(os\_data\_X, os\_data\_y.values.ravel())  
print(rfe.support\_)  
print(rfe.ranking\_)

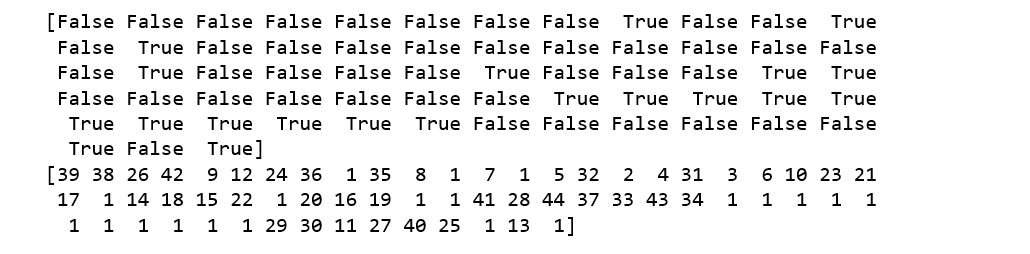


Figure 16

The RFE has helped us select the following features: “euribor3m”, “job\_blue-collar”, “job\_housemaid”, “marital\_unknown”, “education\_illiterate”, “default\_no”, “default\_unknown”, “contact\_cellular”, “contact\_telephone”, “month\_apr”, “month\_aug”, “month\_dec”, “month\_jul”, “month\_jun”, “month\_mar”, “month\_may”, “month\_nov”, “month\_oct”, “poutcome\_failure”, “poutcome\_success”.

cols=['euribor3m', 'job\_blue-collar', 'job\_housemaid', 'marital\_unknown', 'education\_illiterate', 'default\_no', 'default\_unknown',   
 'contact\_cellular', 'contact\_telephone', 'month\_apr', 'month\_aug', 'month\_dec', 'month\_jul', 'month\_jun', 'month\_mar',   
 'month\_may', 'month\_nov', 'month\_oct', "poutcome\_failure", "poutcome\_success"]   
X=os\_data\_X[cols]  
y=os\_data\_y['y']

### Implementing the model

import statsmodels.api as sm  
logit\_model=sm.Logit(y,X)  
result=logit\_model.fit()  
print(result.summary2())

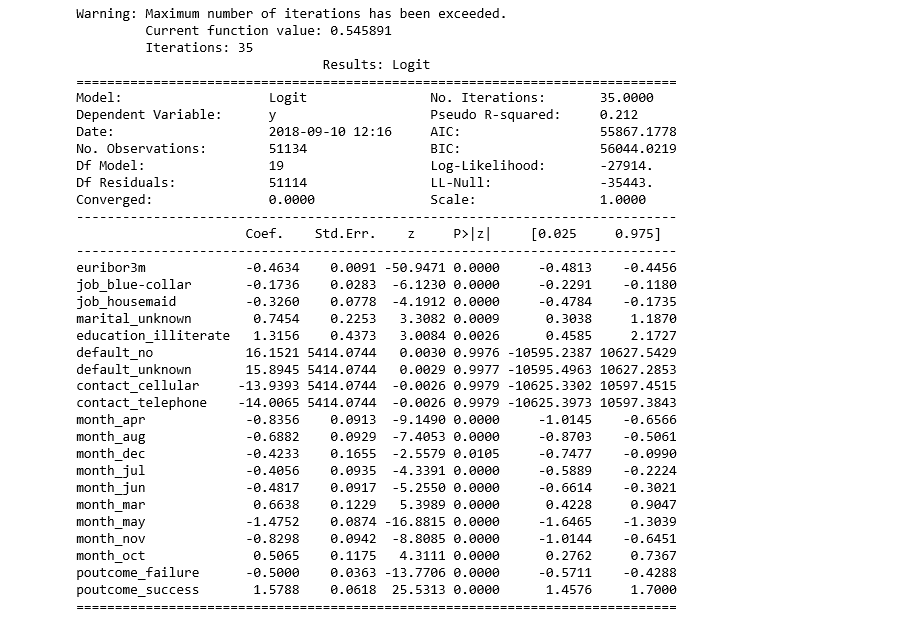


Figure 17

The p-values for most of the variables are smaller than 0.05, except four variables, therefore, we will remove them.

cols=['euribor3m', 'job\_blue-collar', 'job\_housemaid', 'marital\_unknown', 'education\_illiterate',   
 'month\_apr', 'month\_aug', 'month\_dec', 'month\_jul', 'month\_jun', 'month\_mar',   
 'month\_may', 'month\_nov', 'month\_oct', "poutcome\_failure", "poutcome\_success"]   
X=os\_data\_X[cols]  
y=os\_data\_y['y']

logit\_model=sm.Logit(y,X)  
result=logit\_model.fit()  
print(result.summary2())

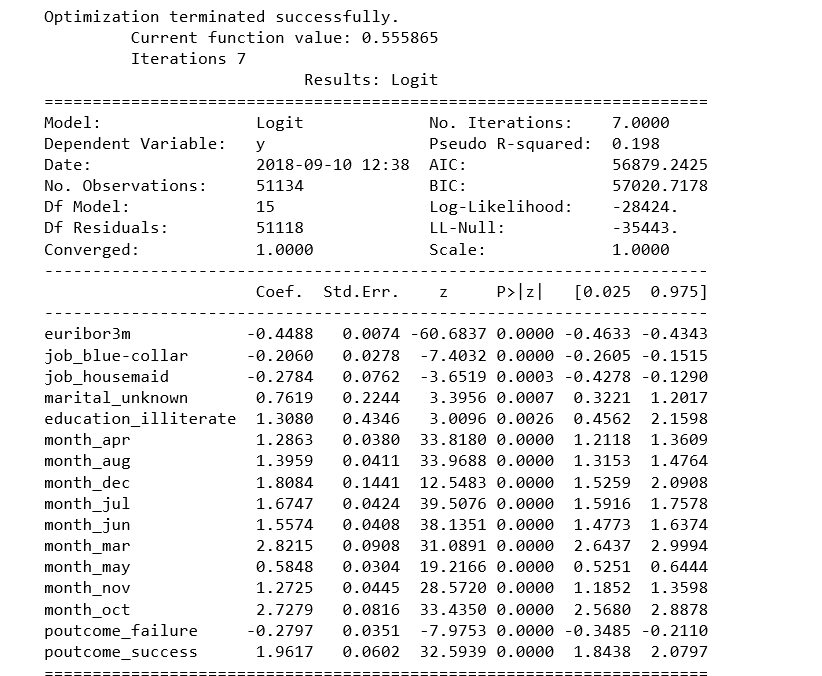


Figure 18

### Logistic Regression Model Fitting

from sklearn.linear\_model import LogisticRegression  
from sklearn import metrics

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)  
logreg = LogisticRegression()  
logreg.fit(X\_train, y\_train)

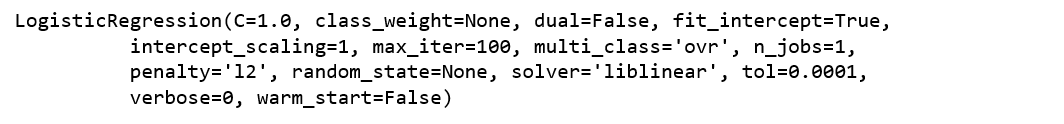


Figure 19

**Predicting the test set results and calculating the accuracy**

y\_pred = logreg.predict(X\_test)  
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X\_test, y\_test)))

**Accuracy of logistic regression classifier on test set: 0.74**

### Confusion Matrix

from sklearn.metrics import confusion\_matrix  
confusion\_matrix = confusion\_matrix(y\_test, y\_pred)  
print(confusion\_matrix)

**[[6124 1542]**

**[2505 5170]]**

The result is telling us that we have **6124+5170** correct predictions and **2505+1542** incorrect predictions.

### ****Compute precision, recall, F-measure and support****

To quote from Scikit Learn:

The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative.

The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights the recall more than the precision by a factor of beta. beta = 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in y\_test.

from sklearn.metrics import classification\_report  
print(classification\_report(y\_test, y\_pred))

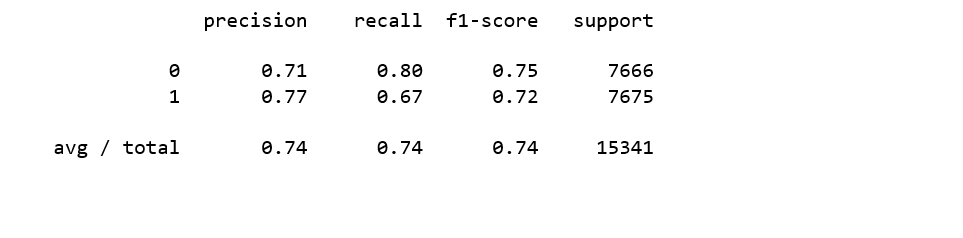


Figure 20

**Interpretation**: Of the entire test set, 74% of the promoted term deposit were the term deposit that the customers liked. Of the entire test set, 74% of the customer’s preferred term deposits that were promoted.

### ROC Curve

from sklearn.metrics import roc\_auc\_score  
from sklearn.metrics import roc\_curve  
logit\_roc\_auc = roc\_auc\_score(y\_test, logreg.predict(X\_test))  
fpr, tpr, thresholds = roc\_curve(y\_test, logreg.predict\_proba(X\_test)[:,1])  
plt.figure()  
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit\_roc\_auc)  
plt.plot([0, 1], [0, 1],'r--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver operating characteristic')  
plt.legend(loc="lower right")  
plt.savefig('Log\_ROC')  
plt.show()



Figure 21

The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner)