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 Exploration**

The dataset comes from the [UCI Machine Learning repository](https://archive.ics.uci.edu/ml/index.php), 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](https://raw.githubusercontent.com/madmashup/targeted-marketing-predictive-engine/master/banking.csv).

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.cross\_validation 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.

data = pd.read\_csv('bank.csv', header=0)

data = data.dropna()

print(data.shape)

print(list(data.columns))

*(41188, 21)*

*['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']*

**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)

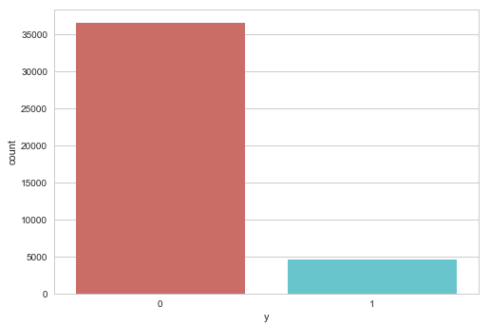
**Predict variable (desired target)**

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

**Barplot for the dependent variable**

sns.countplot(x='y',data=data, palette='hls')

plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-1.png)

**Check the missing values**

data.isnull().sum()

*age 0*

*job 0*

*marital 0*

*education 0*

*default 0*

*housing 0*

*loan 0*

*contact 0*

*month 0*

*day\_of\_week 0*

*duration 0*

*campaign 0*

*pdays 0*

*previous 0*

*poutcome 0*

*emp\_var\_rate 0*

*cons\_price\_idx 0*

*cons\_conf\_idx 0*

*euribor3m 0*

*nr\_employed 0*

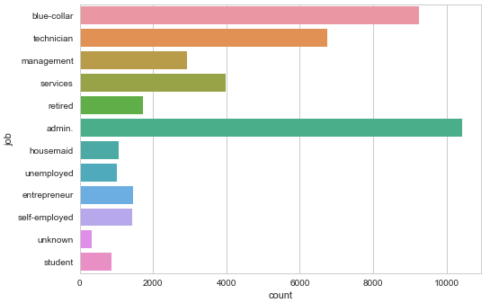
*y 0*

*dtype: int64*

**Customer job distribution**

sns.countplot(y="job", data=data)

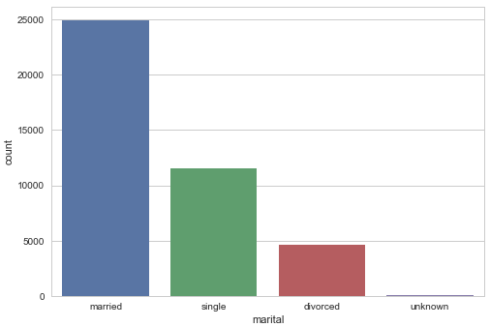
plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-3.png)

**Customer marital status distribution**

sns.countplot(x="marital", data=data)

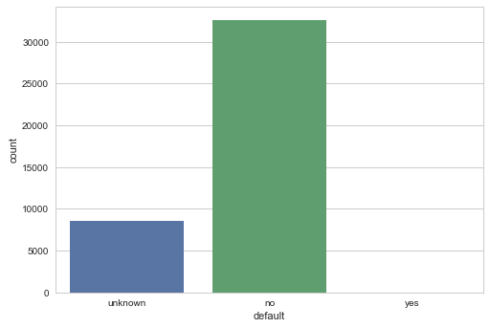
plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-4.png)

**Barplot for credit in default**

sns.countplot(x="default", data=data)

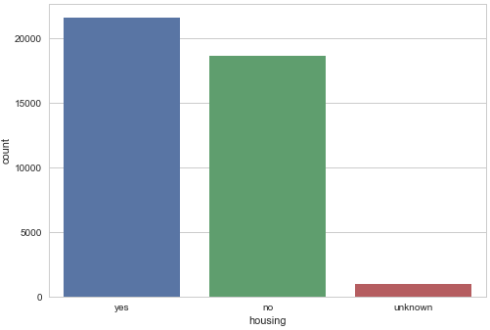
plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-5.png)

**Barplot for housing loan**

sns.countplot(x="housing", data=data)

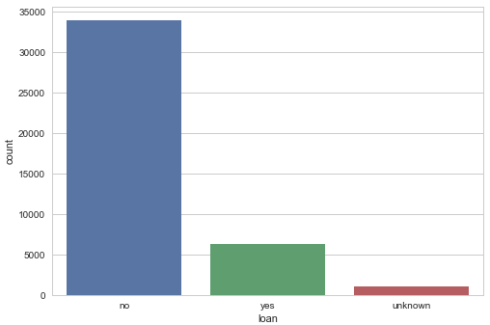
plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-6.png)

**Barplot for personal loan**

sns.countplot(x="loan", data=data)

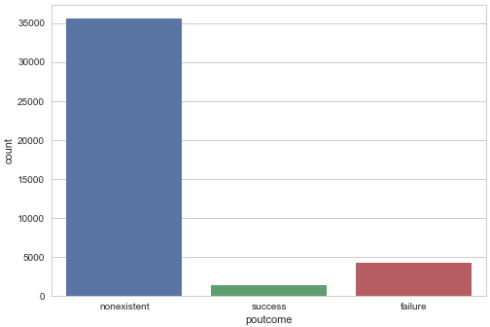
plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-7.png)

**Barplot for previous marketing campaign outcome**

sns.countplot(x="poutcome", data=data)

plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-8.png)

Our prediction will be based on the customer’s job, marital status, whether he(she) has credit in default, whether he(she) has a housing loan, whether he(she) has a personal loan, and the outcome of the previous marketing campaigns. So, we will drop the variables that we do not need.

data.drop(data.columns[[0, 3, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19]], axis=1, inplace=True)

**Data Preprocessing**

**Create dummy variables, that is variables with only two values, zero and one.**

In logistic regression models, encoding all of the independent variables as dummy variables allows easy interpretation and calculation of the odds ratios, and increases the stability and significance of the coefficients.

data2 = pd.get\_dummies(data, columns =['job', 'marital', 'default', 'housing', 'loan', 'poutcome'])

**Drop the unknown columns**

data2.drop(data2.columns[[12, 16, 18, 21, 24]], axis=1, inplace=True)

data2.columns

*Index(['y', 'job\_admin.', 'job\_blue-collar', 'job\_entrepreneur',*

*'job\_housemaid', 'job\_management', 'job\_retired', 'job\_self-employed',*

*'job\_services', 'job\_student', 'job\_technician', 'job\_unemployed',*

*'marital\_divorced', 'marital\_married', 'marital\_single', 'default\_no',*

*'default\_yes', 'housing\_no', 'housing\_yes', 'loan\_no', 'loan\_yes',*

*'poutcome\_failure', 'poutcome\_nonexistent', 'poutcome\_success'],*

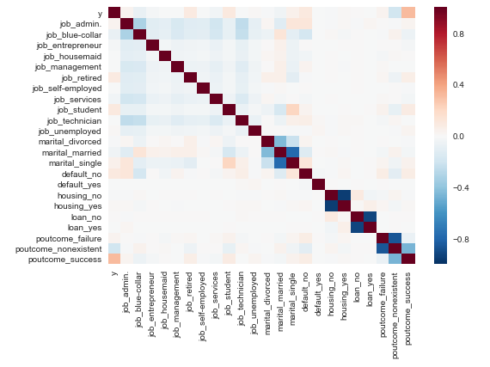
*dtype='object')*

Perfect! Exactly what we need for the next steps.

**Check the independence between the independent variables**

sns.heatmap(data2.corr())

plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-11.png)  
Looks good.

**Split the data into training and test sets**

X = data2.iloc[:,1:]

y = data2.iloc[:,0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

**Check out training data is sufficient**

X\_train.shape

*(30891, 23)*

Great! Now we can start building our logistic regression model.

**Logistic Regression Model**

**Fit logistic regression to the training set**

classifier = LogisticRegression(random\_state=0)

classifier.fit(X\_train, y\_train)

*LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,*

*intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,*

*penalty='l2', random\_state=0, solver='liblinear', tol=0.0001,*

*verbose=0, warm\_start=False)*

**Predicting the test set results and creating confusion matrix**

The confusion\_matrix() function will calculate a confusion matrix and return the result as an array.

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

confusion\_matrix = confusion\_matrix(y\_test, y\_pred)

print(confusion\_matrix)

*[[9046 110]*

*[ 912 229]]*

The result is telling us that we have 9046+229 correct predictions and 912+110 incorrect predictions.

**Accuracy**

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

*Accuracy of logistic regression classifier on test set: 0.90*

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

To quote from [Scikit Learn](http://scikit-learn.org/stable/index.html):  
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))

*precision recall f1-score support*

*0 0.91 0.99 0.95 9156*

*1 0.68 0.20 0.31 1141*

*avg / total 0.88 0.90 0.88 10297*

Interpretation: Of the entire test set, 88% of the promoted term deposit were the term deposit that the customers liked. Of the entire test set, 90% of the customer’s preferred term deposits that were promoted.

**Classifier visualization playground**

The purpose of this section is to visualize logistic regression classsifiers’ decision boundaries. In order to better vizualize the decision boundaries, we’ll perform Principal Component Analysis (PCA) on the data to reduce the dimensionality to 2 dimensions.

from sklearn.decomposition import PCA

X = data2.iloc[:,1:]

y = data2.iloc[:,0]

pca = PCA(n\_components=2).fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pca, y, random\_state=0)

plt.figure(dpi=120)

plt.scatter(pca[y.values==0,0], pca[y.values==0,1], alpha=0.5, label='YES', s=2, color='navy')

plt.scatter(pca[y.values==1,0], pca[y.values==1,1], alpha=0.5, label='NO', s=2, color='darkorange')

plt.legend()

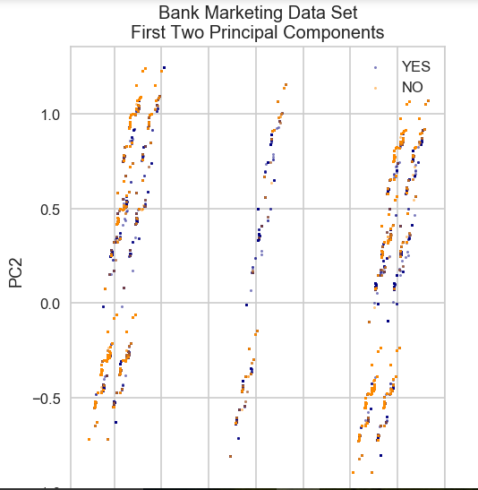
plt.title('Bank Marketing Data Set\nFirst Two Principal Components')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.gca().set\_aspect('equal')

plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-16.png)

def plot\_bank(X, y, fitted\_model):

plt.figure(figsize=(9.8,5), dpi=100)

for i, plot\_type in enumerate(['Decision Boundary', 'Decision Probabilities']):

plt.subplot(1,2,i+1)

mesh\_step\_size = 0.01 # step size in the mesh

x\_min, x\_max = X[:, 0].min() - .1, X[:, 0].max() + .1

y\_min, y\_max = X[:, 1].min() - .1, X[:, 1].max() + .1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, mesh\_step\_size), np.arange(y\_min, y\_max, mesh\_step\_size))

if i == 0:

Z = fitted\_model.predict(np.c\_[xx.ravel(), yy.ravel()])

else:

try:

Z = fitted\_model.predict\_proba(np.c\_[xx.ravel(), yy.ravel()])[:,1]

except:

plt.text(0.4, 0.5, 'Probabilities Unavailable', horizontalalignment='center',

verticalalignment='center', transform = plt.gca().transAxes, fontsize=12)

plt.axis('off')

break

Z = Z.reshape(xx.shape)

plt.scatter(X[y.values==0,0], X[y.values==0,1], alpha=0.8, label='YES', s=5, color='navy')

plt.scatter(X[y.values==1,0], X[y.values==1,1], alpha=0.8, label='NO', s=5, color='darkorange')

plt.imshow(Z, interpolation='nearest', cmap='RdYlBu\_r', alpha=0.15,

extent=(x\_min, x\_max, y\_min, y\_max), origin='lower')

plt.title(plot\_type + '\n' +

str(fitted\_model).split('(')[0]+ ' Test Accuracy: ' + str(np.round(fitted\_model.score(X, y), 5)))

plt.gca().set\_aspect('equal');

plt.tight\_layout()

plt.legend()

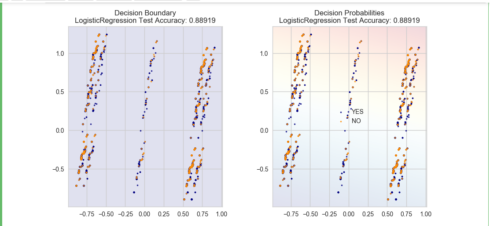
plt.subplots\_adjust(top=0.9, bottom=0.08, wspace=0.02)

model = LogisticRegression()

model.fit(X\_train,y\_train)

plot\_bank(X\_test, y\_test, model)

plt.show()

Gives this plot:  
[](https://datascienceplus.com/wp-content/uploads/2017/10/2017-09-28-17.png)

As you can see, the PCA has reduced the accuracy of our Logistic Regression model. This is because we use PCA to reduce the amount of the dimension, so we have removed information from our data.