Testing the model

Import the relevant libraries

In [34]:
```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from scipy import stats
stats.chisqprob = lambda chisq, df: stats.chi2.sf(chisq, df)
```

Load the data

Load the 'Bank_data.csv' dataset.

In [35]:
```python
raw_data = pd.read_csv('Bank_data.csv')
raw_data
```

Out[35]:

	Unnamed: 0	interest_rate	credit	march	may	previous	duration	y
0	0	1.334	0.0	1.0	0.0	0.0	117.0	no
1	1	0.767	0.0	0.0	2.0	1.0	274.0	yes
2	2	4.858	0.0	1.0	0.0	0.0	167.0	no
3	3	4.120	0.0	0.0	0.0	0.0	686.0	yes
4	4	4.856	0.0	1.0	0.0	0.0	157.0	no
5	5	0.899	0.0	0.0	1.0	0.0	126.0	no
6	6	4.962	0.0	0.0	0.0	0.0	84.0	no
7	7	4.858	0.0	1.0	0.0	0.0	17.0	no
8	8	4.962	0.0	0.0	0.0	0.0	704.0	yes
9	9	4.865	0.0	0.0	0.0	0.0	185.0	no
10	10	1.365	0.0	0.0	1.0	1.0	374.0	no
11	11	0.773	0.0	0.0	0.0	0.0	91.0	yes
12	12	0.714	0.0	0.0	2.0	1.0	169.0	yes
13	13	4.864	0.0	0.0	0.0	0.0	249.0	no
14	14	4.966	0.0	0.0	0.0	0.0	215.0	no
15	15	0.904	0.0	0.0	0.0	0.0	324.0	yes
16	16	0.849	0.0	0.0	2.0	1.0	159.0	yes
17	17	1.811	1.0	0.0	0.0	0.0	120.0	yes
18	18	1.264	0.0	1.0	0.0	0.0	337.0	yes
19	19	4.076	0.0	0.0	0.0	0.0	640.0	no
20	20	1.262	0.0	0.0	0.0	0.0	663.0	yes
21	21	0.695	0.0	1.0	2.0	1.0	403.0	yes
22	22	4.960	0.0	0.0	0.0	0.0	300.0	no
23	23	4.963	0.0	0.0	0.0	0.0	255.0	yes
24	24	1.354	0.0	1.0	1.0	1.0	293.0	yes
25	25	1.244	0.0	1.0	0.0	0.0	68.0	no
26	26	0.748	0.0	0.0	1.0	0.0	266.0	yes
27	27	0.878	0.0	0.0	0.0	0.0	272.0	yes
28	28	0.644	0.0	0.0	0.0	0.0	398.0	yes
29	29	4.968	0.0	0.0	0.0	0.0	126.0	no
...	...	...	...	...	...	...	...	...
488	488	0.885	0.0	0.0	2.0	1.0	608.0	yes
489	489	4.153	0.0	0.0	0.0	0.0	250.0	no
490	490	0.735	0.0	0.0	0.0	0.0	107.0	yes
491	491	4.965	0.0	0.0	0.0	0.0	716.0	yes
492	492	4.859	0.0	1.0	0.0	0.0	619.0	yes
493	493	4.864	0.0	0.0	0.0	0.0	406.0	no
494	494	4.963	0.0	0.0	0.0	0.0	73.0	no
495	495	1.334	0.0	1.0	0.0	0.0	127.0	no
496	496	4.153	0.0	0.0	0.0	0.0	187.0	no
497	497	0.646	1.0	0.0	1.0	1.0	180.0	yes
498	498	1.266	0.0	1.0	0.0	0.0	326.0	yes
499	499	1.405	0.0	0.0	1.0	0.0	109.0	yes
500	500	0.900	0.0	0.0	2.0	1.0	470.0	yes
501	501	4.076	0.0	0.0	0.0	0.0	73.0	no
502	502	1.050	0.0	0.0	0.0	0.0	141.0	no
503	503	1.029	0.0	0.0	1.0	0.0	115.0	no
504	504	0.748	0.0	0.0	1.0	1.0	171.0	no
505	505	4.961	0.0	0.0	0.0	0.0	498.0	no
506	506	1.268	0.0	0.0	0.0	0.0	365.0	yes
507	507	4.959	0.0	0.0	0.0	0.0	10.0	no
508	508	4.021	0.0	0.0	1.0	0.0	796.0	yes
509	509	0.877	0.0	0.0	2.0	1.0	279.0	yes
510	510	1.327	0.0	1.0	0.0	0.0	476.0	yes
511	511	4.965	0.0	0.0	0.0	0.0	479.0	yes
512	512	1.266	0.0	1.0	1.0	0.0	225.0	no
513	513	1.334	0.0	1.0	0.0	0.0	204.0	no
514	514	0.861	0.0	0.0	2.0	1.0	806.0	yes
515	515	0.879	0.0	0.0	0.0	0.0	290.0	no
516	516	0.877	0.0	0.0	5.0	1.0	473.0	yes
517	517	4.965	0.0	0.0	0.0	0.0	142.0	no

518 rows × 8 columns

Note that *interest rate* indicates the 3-month interest rate between banks and *duration* indicates the time since the last contact was made with a given consumer. The *previous* variable shows whether the last marketing campaign was successful with this customer. The *march* and *may* are Boolean variables that account for when the call was made to the specific customer and *credit* shows if the customer has enough credit to avoid defaulting.

We want to know whether the bank marketing strategy was successful, so we need to transform the outcome variable into Boolean values in order to run regressions.

In [36]:
```python
# We make sure to create a copy of the data before we start altering it. Note that we don't change the original
data = raw_data.copy()
# Removes the index column thata comes with the data
data = data.drop(['Unnamed: 0'], axis = 1)
# We use the map function to change any 'yes' values to 1 and 'no'values to 0.
data['y'] = data['y'].map({'yes':1, 'no':0})
data
```

Out[36]:

	interest_rate	credit	march	may	previous	duration	y
0	1.334	0.0	1.0	0.0	0.0	117.0	0
1	0.767	0.0	0.0	2.0	1.0	274.0	1
2	4.858	0.0	1.0	0.0	0.0	167.0	0
3	4.120	0.0	0.0	0.0	0.0	686.0	1
4	4.856	0.0	1.0	0.0	0.0	157.0	0
5	0.899	0.0	0.0	1.0	0.0	126.0	0
6	4.962	0.0	0.0	0.0	0.0	84.0	0
7	4.858	0.0	1.0	0.0	0.0	17.0	0
8	4.962	0.0	0.0	0.0	0.0	704.0	1
9	4.865	0.0	0.0	0.0	0.0	185.0	0
10	1.365	0.0	0.0	1.0	1.0	374.0	0
11	0.773	0.0	0.0	0.0	0.0	91.0	1
12	0.714	0.0	0.0	2.0	1.0	169.0	1
13	4.864	0.0	0.0	0.0	0.0	249.0	0
14	4.966	0.0	0.0	0.0	0.0	215.0	0
15	0.904	0.0	0.0	0.0	0.0	324.0	1
16	0.849	0.0	0.0	2.0	1.0	159.0	1
17	1.811	1.0	0.0	0.0	0.0	120.0	1
18	1.264	0.0	1.0	0.0	0.0	337.0	1
19	4.076	0.0	0.0	0.0	0.0	640.0	0
20	1.262	0.0	0.0	0.0	0.0	663.0	1
21	0.695	0.0	1.0	2.0	1.0	403.0	1
22	4.960	0.0	0.0	0.0	0.0	300.0	0
23	4.963	0.0	0.0	0.0	0.0	255.0	1
24	1.354	0.0	1.0	1.0	1.0	293.0	1
25	1.244	0.0	1.0	0.0	0.0	68.0	0
26	0.748	0.0	0.0	1.0	0.0	266.0	1
27	0.878	0.0	0.0	0.0	0.0	272.0	1
28	0.644	0.0	0.0	0.0	0.0	398.0	1
29	4.968	0.0	0.0	0.0	0.0	126.0	0
...	...	...	...	...	...	...	...
488	0.885	0.0	0.0	2.0	1.0	608.0	1
489	4.153	0.0	0.0	0.0	0.0	250.0	0
490	0.735	0.0	0.0	0.0	0.0	107.0	1
491	4.965	0.0	0.0	0.0	0.0	716.0	1
492	4.859	0.0	1.0	0.0	0.0	619.0	1
493	4.864	0.0	0.0	0.0	0.0	406.0	0
494	4.963	0.0	0.0	0.0	0.0	73.0	0
495	1.334	0.0	1.0	0.0	0.0	127.0	0
496	4.153	0.0	0.0	0.0	0.0	187.0	0
497	0.646	1.0	0.0	1.0	1.0	180.0	1
498	1.266	0.0	1.0	0.0	0.0	326.0	1
499	1.405	0.0	0.0	1.0	0.0	109.0	1
500	0.900	0.0	0.0	2.0	1.0	470.0	1
501	4.076	0.0	0.0	0.0	0.0	73.0	0
502	1.050	0.0	0.0	0.0	0.0	141.0	0
503	1.029	0.0	0.0	1.0	0.0	115.0	0
504	0.748	0.0	0.0	1.0	1.0	171.0	0
505	4.961	0.0	0.0	0.0	0.0	498.0	0
506	1.268	0.0	0.0	0.0	0.0	365.0	1
507	4.959	0.0	0.0	0.0	0.0	10.0	0
508	4.021	0.0	0.0	1.0	0.0	796.0	1
509	0.877	0.0	0.0	2.0	1.0	279.0	1
510	1.327	0.0	1.0	0.0	0.0	476.0	1
511	4.965	0.0	0.0	0.0	0.0	479.0	1
512	1.266	0.0	1.0	1.0	0.0	225.0	0
513	1.334	0.0	1.0	0.0	0.0	204.0	0
514	0.861	0.0	0.0	2.0	1.0	806.0	1
515	0.879	0.0	0.0	0.0	0.0	290.0	0
516	0.877	0.0	0.0	5.0	1.0	473.0	1
517	4.965	0.0	0.0	0.0	0.0	142.0	0

518 rows × 7 columns

In [37]:
```python
data.describe()
```

Out[37]:

	interest_rate	credit	march	may	previous	duration	y
count	518.000000	518.000000	518.000000	518.000000	518.000000	518.000000	518.000000
mean	2.835776	0.034749	0.266409	0.388031	0.127413	382.177606	0.500000
std	1.876903	0.183321	0.442508	0.814527	0.333758	344.295990	0.500483
min	0.635000	0.000000	0.000000	0.000000	0.000000	9.000000	0.000000
25%	1.042750	0.000000	0.000000	0.000000	0.000000	155.000000	0.000000
50%	1.466000	0.000000	0.000000	0.000000	0.000000	266.500000	0.500000
75%	4.956500	0.000000	1.000000	0.000000	0.000000	482.750000	1.000000
max	4.970000	1.000000	1.000000	5.000000	1.000000	2653.000000	1.000000

Declare the dependent and independent variables

Use 'duration' as the independet variable.

In [38]:
```python
y = data['y']
x1 = data['duration']
```

Simple Logistic Regression

Run the regression and graph the scatter plot.

In [39]:
```python
x = sm.add_constant(x1)
reg_log = sm.Logit(y,x)
results_log = reg_log.fit()
# Get the regression summary
results_log.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.546118
         Iterations 7
```

Out[39]:

Logit Regression Results

Dep. Variable:	y	No. Observations:	518
Model:	Logit	Df Residuals:	516
Method:	MLE	Df Model:	1
Date:	Wed, 30 Jan 2019	Pseudo R-squ.:	0.2121
Time:	17:12:17	Log-Likelihood:	-282.89
converged:	True	LL-Null:	-359.05
		LLR p-value:	5.387e-35

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.7001	0.192	-8.863	0.000	-2.076	-1.324
duration	0.0051	0.001	9.159	0.000	0.004	0.006

In [40]:
```python
# Create a scatter plot of x1 (Duration, no constant) and y (Subscribed)
plt.scatter(x1,y,color = 'C0')
# Don't forget to label your axes!
plt.xlabel('Duration', fontsize = 20)
plt.ylabel('Subscription', fontsize = 20)
plt.show()
```

Expand the model

We can be omitting many causal factors in our simple logistic model, so we instead switch to a multivariate logistic regression model. Add the 'interest_rate', 'march', 'credit' and 'previous' estimators to our model and run the regression again.

Declare the independent variable(s)

In [45]:
```python
# To avoid writing them out every time, we save the names of the estimators of our model in a list.
estimators=['interest_rate','credit','march','previous','duration']

X1_all = data[estimators]
y = data['y']
```

In [47]:
```python
X_all = sm.add_constant(X1_all)
reg_logit = sm.Logit(y,X_all)
results_logit = reg_logit.fit()
results_logit.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.336664
         Iterations 7
```

Out[47]:

Model:	Logit	Pseudo R-squared:	0.514
Dependent Variable:	y	AIC:	360.7836
Date:	2019-01-30 17:13	BIC:	386.2834
No. Observations:	518	Log-Likelihood:	-174.39
Df Model:	5	LL-Null:	-359.05
Df Residuals:	512	LLR p-value:	1.2114e-77
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-0.0211	0.3113	-0.0677	0.9460	-0.6313	0.5891
interest_rate	-0.8001	0.0895	-8.9434	0.0000	-0.9755	-0.6248
credit	2.3585	1.0875	2.1688	0.0301	0.2271	4.4900
march	-1.8322	0.3297	-5.5563	0.0000	-2.4785	-1.1859
previous	1.5363	0.5010	3.0666	0.0022	0.5544	2.5182
duration	0.0070	0.0007	9.3810	0.0000	0.0055	0.0084

Confusion Matrix

Find the confusion matrix of the model and estimate its accuracy.

In [48]:
```python
def confusion_matrix(data,actual_values,model):

        # Confusion matrix

        # Parameters
        # ----------
        # data: data frame or array
            # data is a data frame formatted in the same way as your input data (without the actual values)
            # e.g. const, var1, var2, etc. Order is very important!
        # actual_values: data frame or array
            # These are the actual values from the test_data
            # In the case of a logistic regression, it should be a single column with 0s and 1s

        # model: a LogitResults object
            # this is the variable where you have the fitted model
            # e.g. results_log in this course
        # ----------

        #Predict the values using the Logit model
        pred_values = model.predict(data)
        # Specify the bins
        bins=np.array([0,0.5,1])
        # Create a histogram, where if values are between 0 and 0.5 tell will be considered 0
        # if they are between 0.5 and 1, they will be considered 1
        cm = np.histogram2d(actual_values, pred_values, bins=bins)[0]
        # Calculate the accuracy
        accuracy = (cm[0,0]+cm[1,1])/cm.sum()
        # Return the confusion matrix and
        return cm, accuracy
```

In [50]:
```python
confusion_matrix(X_all,y,results_logit)
```

Out[50]:
```
(array([[218.,  41.],
        [ 30., 229.]]), 0.862934362934363)
```

Test the model

Load the test data from the 'Bank_data_testing.csv' file provided. (Remember to convert the outcome variable 'y' into Boolean).

Load new data

In [51]:
```python
# We have to load data our model has never seen before.
raw_data2 = pd.read_csv('Bank-data-testing.csv')
data_test = raw_data2.copy()
# Removes the index column thata comes with the data
data_test = data_test.drop(['Unnamed: 0'], axis = 1)
```

In [52]:
```python
# Coverting the outcome variable into 1s and 0s again.
data_test['y'] = data_test['y'].map({'yes':1, 'no':0})
data_test
```

Out[52]:

	interest_rate	credit	march	may	previous	duration	y
0	1.313	0.0	1.0	0.0	0.0	487.0	0
1	4.961	0.0	0.0	0.0	0.0	132.0	0
2	4.856	0.0	1.0	0.0	0.0	92.0	0
3	4.120	0.0	0.0	0.0	0.0	1468.0	1
4	4.963	0.0	0.0	0.0	0.0	36.0	0
5	0.697	0.0	1.0	4.0	0.0	131.0	0
6	0.639	1.0	0.0	0.0	0.0	215.0	1
7	4.120	0.0	0.0	0.0	0.0	499.0	0
8	1.281	0.0	1.0	1.0	0.0	809.0	1
9	4.966	0.0	0.0	0.0	0.0	389.0	0
10	4.965	0.0	0.0	0.0	0.0	168.0	0
11	4.968	0.0	0.0	0.0	0.0	766.0	1
12	0.714	0.0	0.0	1.0	0.0	192.0	1
13	1.410	0.0	0.0	1.0	0.0	48.0	0
14	4.964	0.0	0.0	0.0	0.0	181.0	0
15	4.120	0.0	0.0	0.0	0.0	288.0	0
16	1.260	0.0	0.0	2.0	0.0	431.0	1
17	0.900	0.0	0.0	1.0	1.0	153.0	1
18	1.354	0.0	1.0	0.0	0.0	617.0	1
19	4.857	0.0	1.0	0.0	0.0	293.0	0
20	4.857	0.0	1.0	0.0	0.0	157.0	0
21	0.883	0.0	0.0	4.0	0.0	167.0	1
22	1.405	0.0	0.0	0.0	0.0	180.0	0
23	0.702	0.0	0.0	0.0	0.0	7.0	0
24	4.021	0.0	0.0	0.0	0.0	1074.0	1
25	1.264	0.0	1.0	0.0	0.0	377.0	1
26	4.962	0.0	0.0	0.0	0.0	1441.0	1
27	4.120	0.0	0.0	0.0	0.0	30.0	0
28	4.857	0.0	1.0	0.0	0.0	154.0	0
29	1.260	0.0	0.0	0.0	0.0	121.0	1
...	...	...	...	...	...	...	...
192	0.709	0.0	0.0	1.0	0.0	367.0	1
193	1.299	0.0	1.0	0.0	0.0	123.0	0
194	0.788	0.0	0.0	1.0	1.0	261.0	1
195	4.856	0.0	1.0	0.0	0.0	130.0	0
196	0.702	0.0	0.0	1.0	1.0	347.0	1
197	0.900	0.0	0.0	3.0	1.0	288.0	1
198	4.963	0.0	0.0	0.0	0.0	119.0	0
199	0.851	0.0	0.0	1.0	1.0	63.0	1
200	0.720	0.0	0.0	0.0	0.0	485.0	1
201	1.327	0.0	1.0	0.0	0.0	263.0	0
202	1.281	0.0	1.0	0.0	0.0	77.0	0
203	4.961	0.0	0.0	0.0	0.0	135.0	0
204	1.327	0.0	1.0	0.0	0.0	1046.0	1
205	1.291	0.0	1.0	0.0	0.0	725.0	1
206	1.498	0.0	0.0	0.0	0.0	351.0	1
207	4.120	0.0	0.0	0.0	0.0	91.0	0
208	1.410	0.0	0.0	1.0	0.0	291.0	0
209	4.968	0.0	0.0	0.0	0.0	81.0	0
210	1.266	0.0	1.0	0.0	0.0	533.0	1
211	4.961	0.0	0.0	0.0	0.0	952.0	1
212	1.281	0.0	1.0	0.0	0.0	6.0	0
213	1.260	0.0	0.0	0.0	0.0	372.0	1
214	4.858	0.0	1.0	0.0	0.0	607.0	1
215	1.778	1.0	0.0	0.0	0.0	377.0	1
216	1.268	0.0	0.0	0.0	0.0	166.0	1
217	4.963	0.0	0.0	0.0	0.0	458.0	1
218	1.264	0.0	1.0	1.0	0.0	397.0	1
219	1.281	0.0	1.0	0.0	0.0	34.0	0
220	0.739	0.0	0.0	2.0	0.0	233.0	0
221	1.046	0.0	0.0	0.0	0.0	238.0	1

222 rows × 7 columns

Declare the dependent and the independent variables

In [53]:
```python
y_test = data_test['y']
# We already declared a list called 'estimators' that holds all relevant estimators for our model.
X1_test = data_test[estimators]
X_test = sm.add_constant(X1_test)
```

Determine the test confusion matrix and the test accuracy and compare them with the train confusion matrix and the train accuracy.

In [54]:
```python
# Determine the Confusion Matrix and the accuracy of the model with the new data. Note that the model itself st
# test accuracy
confusion_matrix(X_test, y_test, results_logit)
```

Out[54]:
```
(array([[93., 18.],
        [13., 98.]]), 0.8603603603603603)
```

In [56]:
```python
# Compare these values to the Confusion Matrix and the accuracy of the model with the old data.
# train accuracy
confusion_matrix(X_all,y, results_logit)
```

Out[56]:
```
(array([[218.,  41.],
        [ 30., 229.]]), 0.862934362934363)
```

Looking at the test acccuracy we see a number which is a tiny but lower: 86.04%, compared to 86.29% for train accuracy.

In general, we always expect the test accuracy to be lower than the train one. If the test accuracy is higher, this is just due to luck.

Note that when you run the regression, you may get different numbers than us!