Logistic Regression: Solutions #5

David Gerbing
The School of Business
Portland State University
gerbing@pdx.edu

Table of Contents

Preliminaries

```
from datetime import datetime as dt
now = dt.now()
print ("Analysis on", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M"))

    Analysis on 2021-07-26 at 00:37
```

```
import os
os.getcwd()

    '/content'
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(solver='lbfgs', max_iter=500)
```

Read and Prepare data

The following data file contains information on over 7000 customers of a telecom service, including former customers who left the service plan within the last 30 days the data was collected. Build a model to predict customer churn (customer exists the service plan).

Data: http://web.pdx.edu/~gerbing/data/CustomerChurn.csv

Read and Verify Data

a.

- *Read the data into a data frame.*
- *Display the number of rows and columns in the data file, and the first five lines of the data file, including the variable names.*
- *Display all variable names and corresponding data values by transposing the output table.*
- *Display the data type for each variable.*

```
d = pd.read_csv('http://web.pdx.edu/~gerbing/data/CustomerChurn.csv')
#d = pd.read_csv('data/CustomerChurn.csv')
```

```
d.shape

    (7043, 21)
```

```
d.head().transpose()
```

	0	1	2	3	4
customerID	7590-VHVEG	5575-GNVDE	3668-QPYBK	7795-CFOCW	9237-HQITU
gender	Female	Male	Male	Male	Female
SeniorCitizen	0	0	0	0	0
Partner	Yes	No	No	No	No
Dependents	No	No	No	No	No
tenure	1	34	2	45	2
PhoneService	No	Yes	Yes	No	Yes
MultipleLines	No phone service	No	No	No phone service	No
InternetService	DSL	DSL	DSL	DSL	Fiber optic
OnlineSecurity	No	Yes	Yes	Yes	No
OnlineBackup	Yes	No	Yes	No	No
DeviceProtection	No	Yes	No	Yes	No
TechSupport	No	No	No	Yes	No
StreamingTV	No	No	No	No	No
StreamingMovies	No	No	No	No	No
Contract	Month-to-month	One year	Month-to-month	One year	Month-to-month
PaperlessBilling	Yes	No	Yes	No	Yes
PaymentMethod	Electronic check	Mailed check	Mailed check	Bank transfer (automatic)	Electronic check
MonthlyCharges	29.85	56.95	53.85	42.3	70.7

```
d.dtypes

    customerID           object
    gender               object
    SeniorCitizen         int64
    Partner              object
    Dependents           object
    tenure                int64
    PhoneService         object
    MultipleLines        object
    InternetService      object
    OnlineSecurity       object
    OnlineBackup         object
    DeviceProtection     object
    TechSupport          object
    StreamingTV          object
    StreamingMovies      object
    Contract             object
    PaperlessBilling     object
    PaymentMethod        object
    MonthlyCharges      float64
    TotalCharges         object
    Churn                object
    dtype: object
```

b. The variable TotalCharges is conceptually a numeric variable but is read into the data frame as an object variable, i.e., non-numeric. Convert to numeric. As always, audit (verify) any change to the data table. This is one reason why the way in which the variables are read into the system is important to understand before continuing the analysis.

Use the following code for the `to_numeric` function to convert (where *d* is the data frame name, but could be any valid Python name).

`d['TotalCharges'] = pd.to_numeric(d['TotalCharges'], errors='coerce')`

The `errors` parameter set to `'coerce'` instructs to convert to a `NaN` any data value that cannot be converted to a legitimate number.

```
d['TotalCharges'] = pd.to_numeric(d['TotalCharges'], errors='coerce')
d.TotalCharges.dtypes

    dtype('float64')
```

Pre-Process Data

c. Drop the customerID variable.

Hint: Illustrated in several previous notebooks, including 02Wrangle.

```
d.drop('customerID', axis='columns', inplace=True)
```

d. Most of the variables are categorical. Pre-process each categorical variable to become a dummy variable, a type of indicator variable. Retain all k dummy variables for each categorical variable with k levels (to be able to pick and choose the dummy variables to analyze.

Hint: You do not need to list each variable, though could, just the data frame name.

```
d = pd.get_dummies(d)
d.head().transpose()
```

	0	1	2	3	4
SeniorCitizen	0.00	0.00	0.00	0.00	0.00
tenure	1.00	34.00	2.00	45.00	2.00
MonthlyCharges	29.85	56.95	53.85	42.30	70.70
TotalCharges	29.85	1889.50	108.15	1840.75	151.65
gender_Female	1.00	0.00	0.00	0.00	1.00
gender_Male	0.00	1.00	1.00	1.00	0.00
Partner_No	0.00	1.00	1.00	1.00	1.00
Partner_Yes	1.00	0.00	0.00	0.00	0.00
Dependents_No	1.00	1.00	1.00	1.00	1.00
Dependents_Yes	0.00	0.00	0.00	0.00	0.00
PhoneService_No	1.00	0.00	0.00	1.00	0.00
PhoneService_Yes	0.00	1.00	1.00	0.00	1.00
MultipleLines_No	0.00	1.00	1.00	0.00	1.00
MultipleLines_No phone service	1.00	0.00	0.00	1.00	0.00
MultipleLines_Yes	0.00	0.00	0.00	0.00	0.00
InternetService_DSL	1.00	1.00	1.00	1.00	0.00
InternetService_Fiber optic	0.00	0.00	0.00	0.00	1.00
InternetService_No	0.00	0.00	0.00	0.00	0.00
OnlineSecurity_No	1.00	0.00	0.00	0.00	1.00
OnlineSecurity_No internet service	0.00	0.00	0.00	0.00	0.00
OnlineSecurity_Yes	0.00	1.00	1.00	1.00	0.00
OnlineBackup_No	0.00	1.00	0.00	1.00	1.00
OnlineBackup_No internet service	0.00	0.00	0.00	0.00	0.00
OnlineBackup_Yes	1.00	0.00	1.00	0.00	0.00
DeviceProtection_No	1.00	0.00	1.00	0.00	1.00
DeviceProtection_No internet service	0.00	0.00	0.00	0.00	0.00
DeviceProtection_Yes	0.00	1.00	0.00	1.00	0.00
TechSupport_No	1.00	1.00	1.00	0.00	1.00
TechSupport_No internet service	0.00	0.00	0.00	0.00	0.00
TechSupport_Yes	0.00	0.00	0.00	1.00	0.00
StreamingTV_No	1.00	1.00	1.00	1.00	1.00
StreamingTV_No internet service	0.00	0.00	0.00	0.00	0.00
StreamingTV_Yes	0.00	0.00	0.00	0.00	0.00
StreamingMovies_No	1.00	1.00	1.00	1.00	1.00
StreamingMovies_No internet service	0.00	0.00	0.00	0.00	0.00
StreamingMovies_Yes	0.00	0.00	0.00	0.00	0.00
Contract_Month-to-month	1.00	0.00	1.00	0.00	1.00
Contract_One year	0.00	1.00	0.00	1.00	0.00
Contract_Two year	0.00	0.00	0.00	0.00	0.00
PaperlessBilling_No	0.00	1.00	0.00	1.00	0.00
PaperlessBilling_Yes	1.00	0.00	1.00	0.00	1.00
PaymentMethod_Bank transfer (automatic)	0.00	0.00	0.00	1.00	0.00
PaymentMethod_Credit card (automatic)	0.00	0.00	0.00	0.00	0.00
PaymentMethod_Electronic check	1.00	0.00	0.00	0.00	1.00
PaymentMethod_Mailed check	0.00	1.00	1.00	0.00	0.00
Churn_No	1.00	1.00	0.00	1.00	0.00
Churn_Yes	0.00	0.00	1.00	0.00	1.00

e. To keep the analysis simpler, and to drop excess dummy variables retain just the following (mostly indicator) variables for analysis.

'MonthlyCharges', 'TotalCharges', 'Contract_Month-to-month', 'PaperlessBilling_Yes', 'PaymentMethod_Mailed check', 'PhoneService_Yes', 'tenure', 'Dependents_Yes', 'InternetService_No', 'Churn_Yes'

Hint: See subsetting in 02Wrangling.

```
d = d.loc[:, ['MonthlyCharges', 'TotalCharges', 'Contract_Month-to-month',
            'PaperlessBilling_Yes', 'PaymentMethod_Mailed check',
            'PhoneService_Yes', 'tenure', 'Dependents_Yes',
            'InternetService_No', 'Churn_Yes']].copy()
```

f. Simplify the variable names. Rename as follows. Audit.

- MonthlyCharges --> Charges,
- Contract_Month-to-month --> Contract_MtoM,
- PaperlessBilling_Yes --> Paperless,
- PaymentMethod_Mailed check --> Check,
- PhoneService_Yes --> Phone,
- tenure --> Tenure,
- Dependents_Yes --> Dependents,
- InternetService_No --> InternetNo,
- Churn_Yes --> Churn

Hint: Several previous examples, including 02Wrangle.

```
d = d.rename(columns= {'MonthlyCharges': 'Charges',
                       'Contract_Month-to-month': 'MtoM',
                       'PaperlessBilling_Yes': 'Paperless',
                       'PaymentMethod_Mailed check': 'Check',
                       'PhoneService_Yes': 'Phone',
                       'tenure': 'Tenure',
                       'Dependents_Yes': 'Dependents',
                       'InternetService_No': 'InternetNo',
                       'Churn_Yes': 'Churn'})
d.head()
```

	Charges	TotalCharges	MtoM	Paperless	Check	Phone	Tenure	Dependents	Inte
0	29.85	29.85	1	1	0	0	1	0	
1	56.95	1889.50	0	0	1	1	34	0	
2	53.85	108.15	1	1	1	1	2	0	
3	42.30	1840.75	0	0	0	0	45	0	
4	70.70	151.65	1	1	0	1	2	0	

To review the syntax, everything inside { } is called a Python `dictionary`, a core Python data structure. The `dictionary` lists keyword-value pairs.

g. Check for missing data. If not too much, delete the offenders. If severe, impute the missing values. Audit.

Hint: Done in 02PreProcess.

```
print (d.isna().sum())
print ('\nTotal Missing:', d.isna().sum().sum())

    Charges          0
    TotalCharges    11
    MtoM             0
    Paperless        0
    Check            0
    Phone            0
    Tenure           0
    Dependents       0
    InternetNo       0
    Churn            0
    dtype: int64

    Total Missing: 11
```

Very little missing data. Of over 7000 rows of data, just 11 with missing data on only one variable. Delete.

```
print('shape before delete:', d.shape)
d = pd.DataFrame(d.dropna(axis='rows'))
print('shape after delete:', d.shape)

    shape before delete: (7043, 10)
    shape after delete: (7032, 10)
```

Pre-Analysis Understanding and Feature Selection

Target Distribution

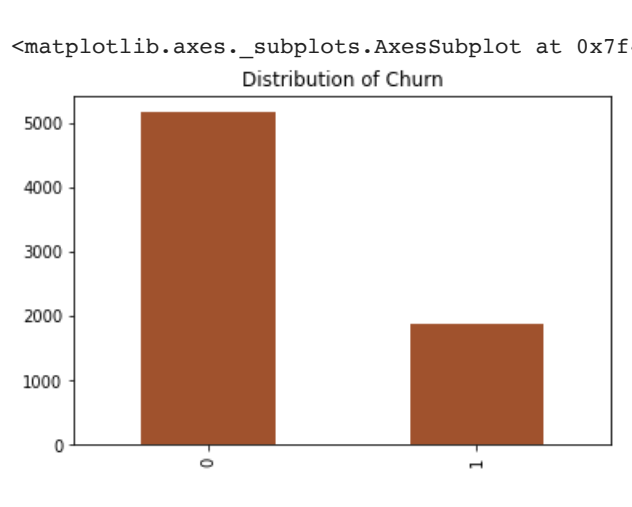
h. Check out the distribution of the target, with a frequency distribution and then the corresponding bar chart.

```
freq = d['Churn'].value_counts()
freq

    0    5163
    1    1869
    Name: Churn, dtype: int64
```

```
plt.title('Distribution of Churn', fontsize=12)
freq.plot(kind='bar', color="sienna")

    <matplotlib.axes._subplots.AxesSubplot at 0x7f4115620fd0>
```



If going to run a model focused on forecasting the target, should also understand the nature of the target variable. The distribution is unbalanced (certainly the company would *not* want a balanced distribution). Over 5000 customers remain and under 2000 customers left (churned). One implication is that the base *null model* accuracy will not be 0.5, but more than 5/7.

Feature Relevance

i. Are all the features relevant? Examine the difference in means of Churn across the features.

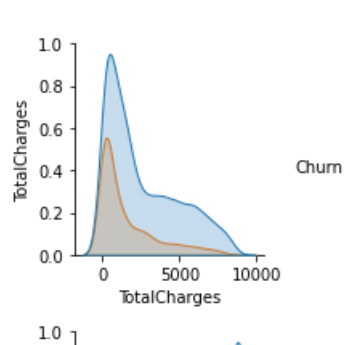
```
d.groupby('Churn').mean()
```

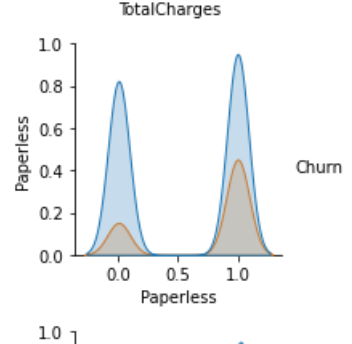
	Charges	TotalCharges	MtoM	Paperless	Check	Phone	Tenure	De
Churn								
0	61.307408	2555.344141	0.429983	0.536122	0.251017	0.901220	37.650010	
1	74.441332	1531.796094	0.885500	0.749064	0.164794	0.909042	17.979133	

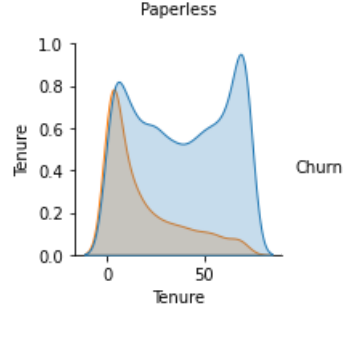
All the means appear to differ depending on Gender, so forecasting accuracy should at least be better than the null model. (Thougn not we are just looking here at the means as descriptive statistics, would be better to do the inferential test, t-test of the mean difference, for each numerical variable.)

j. Examine the overlap in the distributions of Churn for numerical features TotalCharges, Paperless, and tenure. Which variable is likely the best predictor of churn?

```
check = ['TotalCharges', 'Paperless', 'Tenure']
for column in d[check]:
    sns.pairplot(d, vars=[column], hue='Churn')
```







The variable *tenure* (months of being a customer) differentiates the most between those customers who churned and those who stayed. That makes sense, those customers with the most tenure (customers the longest) stayed (apparently the blue distribution), while those with little tenure tended to churn (orange distribution).

Feature Redundancy

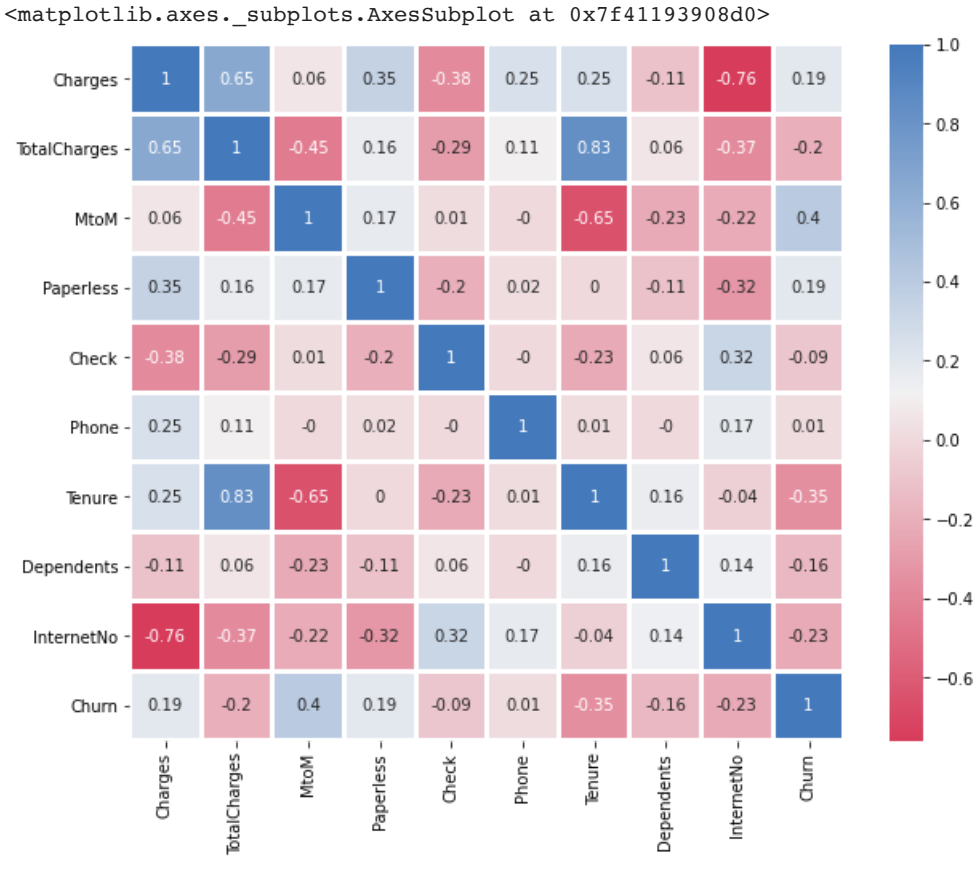
k. Check for collinearity. Comment.

Even not having the need to drop features before model estimation as CPU time is not an issue, it is useful to explore relations of the features with each other, and with the target, to understand more about how the model will perform and not analyze with no understanding of the data.

Because correlations span from negative to positive, use a diverging color palette, with blue indicating positive correlations and red indicating very small positive to negative correlations.

```
plt.figure(figsize=(10,8))
sns.heatmap(d.corr().round(2), linewidths=2.0, annot=True,
           cmap=sns.diverging_palette(5, 250, as_cmap=True))

    <matplotlib.axes._subplots.AxesSubplot at 0x7f41193908d0>
```



In general, collinearity is not bad, feature inter-correlations are generally not high. However, there are some high feature correlations. *Charges* correlations -0.76 with *InternetNo* customers, which makes sense because not having Internet reduces the bill. *MtoM*, that is, a month-to-month customer instead of a longer contract customer, correlates -0.65 with *tenure*, which also logically follows. At least one of these variables for each pair can likely be dropped from the final model without losing much, if any, forecasting efficiency.

Create Feature and Target Data Structures

l. Define all feature variables in a data structure X. Define the target variable as a data structure y, a column of 0's and 1's.

```
y = d['Churn']
pred_vars = ['TotalCharges', 'Paperless', 'Check',
            'Phone', 'Tenure', 'Dependents', 'InternetNo']
X = d[pred_vars]
X.head()
```

	TotalCharges	Paperless	Check	Phone	Tenure	Dependents	InternetNo
0	29.85	1	0	0	1	0	0
1	1889.50	0	1	1	34	0	0
2	108.15	1	1	1	2	0	0
3	1840.75	0	0	0	45	0	0
4	151.65	1	0	1	2	0	0

Not necessary, but see how many features are defined.

```
n_pred = len(pred_vars)
print("Number of predictor variables:", n_pred)

    Number of predictor variables: 7
```

Not necessary, but see how many features are defined with code.

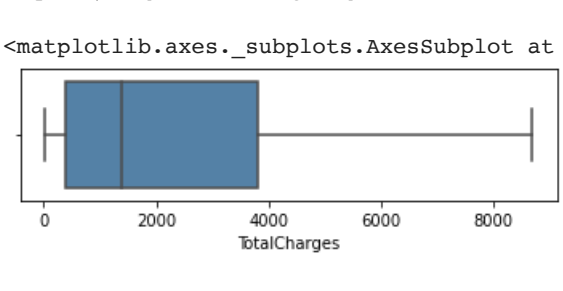
```
n_pred = len(pred_vars)
print("Number of predictor variables:", n_pred)

    Number of predictor variables: 7
```

m. All dummy variables consist of values of only 0 or 1. The numerical variables TotalCharges and Tenure range much more than 0 to 1. Generate a box plot for these variables to examine their range and check for outliers. Discuss.

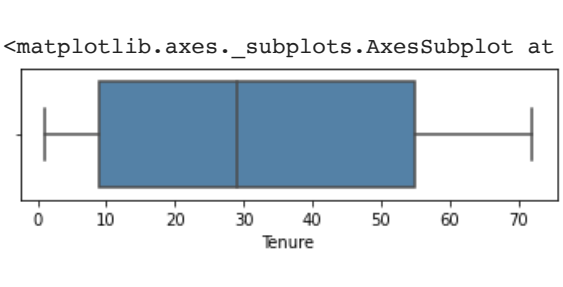
```
plt.figure(figsize=(6,1.5))
sns.boxplot(x=d['TotalCharges'], color='steelblue')

    <matplotlib.axes._subplots.AxesSubplot at 0x7f41099dfe10>
```




```
plt.figure(figsize=(6,1.5))
sns.boxplot(x=d['Tenure'], color='steelblue')

    <matplotlib.axes._subplots.AxesSubplot at 0x7f4109987e90>
```



TotalCharges ranges from 0 to over 8000 USD. No outliers (no points beyond the whiskers). *Tenure* ranges from just over 0 to over 70 months. Again, no outliers. Do not want to covert to a 0 - 1 range with extreme outliers as that would drastically affect all the transformed data values over a much compressed range except for the extreme outliers transformed at or near 0 or 1.

n. Convert TotalCharges and Tenure these variables to a 0 to 1 range so that all feature variables are on the same scale. As always, verify any change in the data.

Hint: Re-scaling done in 02PreProcess.

Data Leak Warning: Better to do this analysis with the re-scaling only done on test data, then done again, anew, on the testing data separately. Otherwise there is *data leakage*, where the testing data is confounded with the training data because at this point in the analysis, the training and testing data are together. Characteristics of the training data will impact the way that later test data is tested.

If doing just one train/test split, separate re-scaling of training and testing data can easily be accomplished with what we know. Just rescale separately the two data sets after forming the split. For the preferred k-fold cross-validation, however, the testing data in each fold needs to be re-scaled separately. To do so we need to introduce the concept of a `pipeline`, which starts to be too much after introducing everything else. To be pure, if on the job for example, should do the separate re-scaling after the train/test split and not do k-fold. Or, even better, learn about constructing a `pipeline`, such as here and here. Another straightforward step, not that hard, but enough for now and not included in this course.

Preferably, we would estimate the re-scaling parameters and then the model itself on all of the data only after successful model validation. This version of the model would then be used to forecast from new data.

Fortunately, with such a large data set, the re-scaling parameters should be reasonably robust. If not constructing a `pipeline`, this step of doing one data rescaling before validation is better than not doing any rescaling at all given the large discrepancies of scales for *TotalCharges* and *Tenure*.

```
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = preprocessing.MinMaxScaler()
X = mm_scaler.fit_transform(X)
X = pd.DataFrame(X, columns=['Charges', 'Paperless', 'Check',
                    'Phone', 'Tenure', 'Dependents', 'InternetNo'])
X.head()
```

	Charges	Paperless	Check	Phone	Tenure	Dependents	InternetNo
0	0.001275	1.0	0.0	0.0	0.000000	0.0	0.0
1	0.215867	0.0	1.0	1.0	0.464789	0.0	0.0
2	0.010310	1.0	1.0	1.0	0.014085	0.0	0.0
3	0.210241	0.0	0.0	0.0	0.619718	0.0	0.0
4	0.015330	1.0	0.0	1.0	0.014085	0.0	0.0

X now has re-scaled variables.

Fit Model and Evaluate with *One* Hold-Out Sample

o. Do a 70% training data and 30% testing data split of X and y data structures. Show the dimensions of the output data structures.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=9)
```

```
print("size of X data structures: ", X_train.shape, X_test.shape)
print("size of y data structures: ", y_train.shape, y_test.shape)

    size of X data structures:  (4922, 7) (2110, 7)
    size of y data structures:  (4922,) (2110,)
```

p. Fit the model to the training data.

```
logistic_model.fit(X_train, y_train)

    LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                       intercept_scaling=1, l1_ratio=None, max_iter=500,
                       multi_class='auto', n_jobs=None, penalty='l2',
                       random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                       warm_start=False)
```

q. Calculate the baseline probability for prediction in the absence of all information regarding X, the null model, the group with the largest proportion.

```
my = y.mean()
max_my = np.max([y.mean(), 1-y.mean()])
print('Proportion of 0\'s (do not churn): %.3f' % (1-my))
print('Proportion of 1\'s (churn): %.3f' % my)
print('Null model accuracy: %.3f' % max_my)

    Proportion of 0's (do not churn): 0.734
    Proportion of 1's (churn): 0.266
    Null model accuracy: 0.734
```

If all customers are predicted to *not* churn, accuracy is 73.4%. The question: How much better can the logistic regression model do for accuracy greater than 73.4%?

r. As a basis for evaluating forecasting accuracy, get the values fit by the model from the corresponding X values, for training and testing data.

```
y_fit = logistic_model.predict(X_train)
y_pred = logistic_model.predict(X_test)
```

s. For the testing data, calculate the probability of a churn for all the rows of testing data from the values of the features, the predictor variables.

```
probs = [i[1] for i in logistic_model.predict_proba(X_test)]
probs[0:5]

    [0.2780896592421815,
     0.15770858618339492,
     0.25350047972517026,
     0.0023574929364177837,
     0.4631213430138494]
```

t. To understand more of what is happening here (for pedagogy), view the true values, forecasted values, and the estimated probability of churning for about 10 or so rows of data. Best display is as a data frame, so convert just to show a more readable display.

```
pred_df = pd.DataFrame({'true_values': y_test,
                        'pred_values': y_pred,
                        'pred_probs':probs})
pred_df.head(15).transpose().style.format("{:.3}")
```

	1888	5398	2622	531	1472	2035	4838	5505	3528	3279	3287	3423	1434	6508
true_values	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0
pred_values	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
pred_probs	0.278	0.158	0.254	0.00236	0.463	0.153	0.179	0.293	0.118	0.216	0.437	0.339	0.463	0.242

u. Assess the accuracy of the model on training and testing data. Any overfitting?

```
from sklearn.metrics import accuracy_score
print ('Accuracy for training data:  %.3f' % accuracy_score(y_train, y_fit))
print ('Accuracy for testing data:  %.3f' % accuracy_score(y_test, y_pred))

    Accuracy for training data:  0.779
    Accuracy for testing data:  0.793
```

By chance, the testing data actually has a higher accuracy count by 1.4%. Clearly no overfitting here.

v. Show the confusion matrix and explicitly identify the True Negatives, True Positives, False Negatives, and False Positives.

```
from sklearn.metrics import confusion_matrix
dc = pd.DataFrame(confusion_matrix(y_test, y_pred))
dc
```

	0	1
0	1413	155
1	282	260

```
print("True Negatives: ", dc.iloc[0,0])
print("True Positives: ", dc.iloc[1,1])
print("False Negatives: ", dc.iloc[1,0])
print("False Positives: ", dc.iloc[0,1])

    True Negatives:  1413
    True Positives:  260
    False Negatives:  282
    False Positives:  155
```

w. Calculate the recall, precision, and, F1 metrics. Comment on the meaning of each from the perspective of management.

```
from sklearn.metrics import recall_score, precision_score, f1_score
print ('Recall for testing data:  %.3f' % recall_score(y_test, y_pred))
print ('Precision for testing data:  %.3f' % precision_score(y_test, y_pred))
print ('F1 for testing data:  %.3f' % f1_score(y_test, y_pred))

    Recall for testing data:  0.480
    Precision for testing data:  0.627
    F1 for testing data:  0.543
```

The lowest fit index, *recall* (sensitivity) is 48%. That means that the model correctly forecasts 48% of customers who churned (true positive). So the model mislabels about 52% of actual customers as churned who did not churn. This 52% comes from the 282 false negatives from the confusion matrix. Clearly a failure from management's point of view, missing so many customers who ended up leaving the company's service plan.

Conclusion: The model is not sensitive to detecting those who will churn.

The *precision* is somewhat higher, which means that of those the model forecasted as churned, 62.7% actually did churn. Of those forecasted to churn, are, in fact, did not churn, a false positive. As seen in the confusion matrix, only 155 false positives.

The *F1* statistic is between recall and precision, their harmonic average, at 54.3%.

Fit Model, then Predict, Evaluate with *Multiple* Hold-Out Samples

x. Do a 5-fold cross-validation an report individual fold and average values of accuracy, recall, and precision.

Access and instanstiate.

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

Cross-validate with k-fold. Request training data scores also so can evaluate for overfitting by comparing to the corresponding test data scores.

```
from sklearn.model_selection import cross_validate
scores = cross_validate(logistic_model, X, y, cv=skf,
                        scoring=('accuracy', 'recall', 'precision'),
                        return_train_score=True)
```

Print fit metrics for each fold.

```
ds = pd.DataFrame(scores).round(3)
print(ds)

       fit_time  score_time  ...  test_precision  train_precision
    0     0.063       0.032  ...           0.629            0.614
    1     0.049       0.006  ...           0.600            0.626
    2     0.032       0.006  ...           0.581            0.629
    3     0.033       0.006  ...           0.624            0.627
    4     0.052       0.006  ...           0.656            0.618

    [5 rows x 8 columns]
```

Print averages on test data across the folds for the three primary fit indices.

```
print('Mean of test accuracy: %.3f' % ds['test_accuracy'].mean())
print('Mean of test recall: %.3f' % ds['test_recall'].mean())
print('Mean of test precision: %.3f' % ds['test_precision'].mean())

    Mean of test accuracy: 0.781
    Mean of test recall: 0.466
    Mean of test precision: 0.618
```

Not far off from the single training/test split, but these averages are the best estimates of the validity of the model.

y. Comment on the worth of the model.

Baseline accuracy is 73.4%, so only a slight improvement by the model, to 78.1%. The primary utility of the model is to identify customers who will churn before they do so. The model does poorly at this task, only identifying 46.6% of the customers who actually churn. Of those forecasted to churn, only 61.8% actually did. Predictions in either direction are not particularly good.

As it exists, the model is not ready for application.

Automated Feature Selection

The model is not working well, so in practice there is less concern for reducing the number of features to see if not much fit is sacrificed when inputting less information into the model. Still, feature selection shows us which features are most useful, providing a core for constructing future models with different information added to what already is somewhat working.

z. Do a univariate feature selection of the top 4 features. Identify these features.

Univariate Selection

```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(f_classif, k=4).fit(X,y)
selected = selector.get_support()
selected

    array([ True,  True, False, False,  True, False,  True])
```

```
X2 = X.iloc[:, selected]
X2.head()
```

	Charges	Paperless	Tenure	InternetNo
0	0.001275	1.0	0.000000	0.0
1	0.215867	0.0	0.464789	0.0
2	0.010310	1.0	0.014085	0.0
3	0.210241	0.0	0.619718	0.0
4	0.015330	1.0	0.014085	0.0

Multivariate Selection

aa. Do a multivariate feature selection. Identify the selected features.

The LogisticRegression module has already been instantiated as *model*.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import RFE
selector = RFE(logistic_model, n_features_to_select=4, step=1).fit(X,y)
```

The features are selected, but now the X data frame of feature data must be pared down to just include the selected features. For this there are two variables that `RFE()` created. The `support_` vector indicates by True or False the selected variables. The `ranking_` vector ranks the features, with all the selected variables ranked at 1.

```
print(selector.support_)
print(selector.ranking_)

    [ True  True False False  True False  True]
    [1 1 2 4 1 3 1]
```

Use the `support_` created data variable with the `iloc()` function to redefine the feature data frame. Here we return to the full data set of the features, the X data frame.

```
X_reduced = X.iloc[:, selector.support_]
X_reduced.head()
```

	Charges	Paperless	Tenure	InternetNo
0	0.001275	1.0	0.000000	0.0
1	0.215867	0.0	0.464789	0.0
2	0.010310	1.0	0.014085	0.0
3	0.210241	0.0	0.619718	0.0
4	0.015330	1.0	0.014085	0.0

ab. Rank the features in importance.

```
rnk = pd.DataFrame()
rnk['Feature'] = X.columns
rnk['Rank']= selector.ranking_
rnk.sort_values('Rank').transpose()
```

	0	1	4	6	2	5	3
Feature	Charges	Paperless	Tenure	InternetNo	Check	Dependents	Phone
Rank	1	1	1	1	2	3	4

No validation of the reduced model as the full model did not validate. Knowing the most important features serves as a building block to future models, not to serve as model on its own.

Estimate Validated Model on All Data

The `LogisticRegression` classifier has already been instantiated as *logistic_model*. Here fit to all of the data, then retrieve the model coefficients.

```
X.head()
```

	Charges	Paperless	Check	Phone	Tenure	Dependents	InternetNo
0	0.001275	1.0	0.0	0.0	0.000000	0.0	0.0
1	0.215867	0.0	1.0	1.0	0.464789	0.0	0.0
2	0.010310	1.0	1.0	1.0	0.014085	0.0	0.0
3	0.210241	0.0	0.0	0.0	0.619718	0.0	0.0
4	0.015330	1.0	0.0	1.0	0.014085	0.0	0.0

This model is to be fit with seven features, the variables in the X data frame.

```
logistic_model.fit(X, y)
print("intercept %.3f" % logistic_model.intercept_, "\n")
cf = pd.DataFrame()
cf['Feature'] = X.columns
print(X.columns)
cf['Coef']= np.transpose(logistic_model.coef_).round(3)
cf.transpose()

    intercept 0.171

    Index(['Charges', 'Paperless', 'Check', 'Phone', 'Tenure', 'Dependents',
           'InternetNo'],
          dtype='object')
```

	0	1	2	3	4	5	6
Feature	Charges	Paperless	Check	Phone	Tenure	Dependents	InternetNo
Coef	3.455	0.62	-0.553	0.019	-5.581	-0.406	-1.291

Model:

$$\hat{y}_{Churn} = 0.171 + 3.455(x_{Charges}) + 0.62(x_{Paperless}) - 0.553(x_{Check} + 0.019(x_{Phone}) - 5.581(x_{Tenure}) - 0.406(x_{Dependents}) - 1.291(x_{Internet})$$

Apply the Model

ac. Forecast if the customer churns from new data.

Customer data:

- Charges: 200
- Paperless: 1
- Check: 1
- Phone: 1
- Tenure: 12
- Dependents: 0
- Internet: 0

Create a list of these data values, making sure to enter in the same order that the variables appear in the X data frame.

Also, because the data was re-scaled, any new data from which to make a prediction also needs to be re-scaled. I do not believe there was an example of this, so the re-scaling transformation is provided here. Basically, take the *mm_scaler* construct previously defined from the original transformation, and then apply the `transform()` function by itself, without the `fit()` function.

```
print('Multiplier:', mm_scaler.scale_.round(3))
print('Additive:', mm_scaler.min_.round(3))

    Multiplier: [0.     1.     1.     1.     0.014 1.     1.    ]
    Additive: [-0.002  0.      0.      0.     -0.014  0.      0.    ]
```

```
X_new = [[200, 1, 1, 1, 12, 0, 0]]
X_new

    [[200, 1, 1, 1, 12, 0, 0]]
```

```
X_new = mm_scaler.transform(X_new)
X_new

    array([[0.0209093 , 1.        , 1.        , 1.        , 0.15492958,
            0.        , 0.        ]])
```

Now from this re-scaled list, create the forecast, Group 0 (not-churn) or Group 1 (churn) and the associated probability.

```
y_new = logistic_model.predict(X_new)
print("Predicted group membership:", y_new)
y_prob = logistic_model.predict_proba(X_new)
print(round(y_prob[0,1], 3))

    Predicted group membership: [0]
    0.369
```