Principles of Machine Learning

Lab 1 – Classification with Logistic Regression

Overview

In this lab, you will train and evaluate a two-class logistic regression classifier model. Classification is one of the fundamental machine learning methods used in data science. Classification models enable you to predict classes or categories of a label value. Classification algorithms can be two-class methods, where there are two possible categories, or multi-class methods. Like regression, classification is a supervised machine learning technique, wherein models are trained from labeled cases.

In this lab you will use the data set provided to categorize diabetes patients. The steps in this process include:

- Prepare the dataset for analysis
- Investigate relationships in the data set with visualization using custom R or Python code.
- Create a two-class logistic classification model.
- Evaluate the performance to the classification model.

What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A web browser and Internet connection
- The lab files for this lab

Note: To set up the required environment for the lab, follow the instructions in the Setup Guide for this course.

Preparing and Exploring the Data

In this lab you will work with a dataset that contains records of diabetes patients admitted to US hospitals. In this lab you will train and evaluate a classification model to predict which hospitalized diabetes patients will be readmitted for their condition at a later date. Readmission of patients is both a metric of potential poor care as well as a financial burden to patients, insurers, governments and health care providers.

Upload the Data Set

The diabetes patient readmissions dataset comes in two pieces. Following these steps to upload these two files:

1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. Create a new blank experiment, and give it the title **Diabetes Classification**.
3. With the **Diabetes Classification** experiment open, at the bottom left, click **NEW**. Then in the **NEW** dialog box, click the **DATASET** tab.
4. Click **FROM LOCAL FILE**. Then in the **Upload a new dataset** dialog box, browse to select the **diabetic_data.csv** file from the folder where you extracted the lab files on your local computer and enter the following details as shown in the image below, and then click the **OK** icon.
    - **This is a new version of an existing dataset**: Unselected
    - **Enter a name for the new dataset**: diabetic_data.csv
    - **Select a type for the new dataset**: Generic CSV file with a header (.csv)
    - **Provide an optional description**: Diabetes patient re-admissions data.
5. Wait for the upload of the dataset to be completed, and then on the experiment items pane, expand **Saved Datasets** and **My Datasets** to verify that the **diabetic_data.csv** dataset is listed.
6. Repeat the previous steps to upload the **admissions_mapping.csv** dataset with the following settings:
    - **This is a new version of an existing dataset**: Unselected
    - **Enter a name for the new dataset**: admissions_mapping.csv
    - **Select a type for the new dataset**: Generic CSV file with a header (.csv)
    - **Provide an optional description**: Admissions codes.
7. Wait for the upload of the dataset to be completed, and then on the experiment items pane, expand **Saved Datasets** and **My Datasets** to verify that the **diabetic_data.csv** and **admissions_mapping.csv** datasets are listed.
8. Drag the **diabetic_data.csv** and **admissions_mapping.csv** datasets and onto the canvas.
9. Right-click the output of the **diabetic_data.csv** dataset and click **Visualize** to view the data.

    Note that it contains a number of fields, including some IDs, some characteristics of the patient, diagnosis codes, indicators of whether the patient is being treated with a number of different drugs, and a column named **readmitted** that indicates whether the patient has been admitted, and if so if the number of days before readmissions is more than or less than 30.

    The number of diagnostic code categories in the dataset are too numerous (potentially several thousand) to be useful for analysis. These numeric codes should be reduced to the top level categories.

    Note also that the readmitted column is the label you will try to predict with your model – however it contains three possible values, which you must simplify to two values indicating whether or not the patient has been readmitted.

10. Visualize the output of the **admissions_mapping.csv** dataset, and note that it contains an admission type ID and a corresponding admission type description.

    Note that the admissions codes have ambiguous coding. Missing data are coded variously as *Not Available*, *NULL*, and *Not Mapped*.

Preparing the Dataset with R

Follow these steps to prepare the dataset using the tools in Azure ML and R.

Note: If you prefer to work with Python, complete the *Preparing the Dataset with Python* exercise below.

1. Search for the **Execute R Script** module, and add it to the experiment. Then connect the output of the **admissions_mapping.csv** dataset to its **Dataset1** (left) input.
2. Select the **Execute R Script** module, set the **R Version** to the latest available version of **Microsoft R Open**, and then replace the existing R code in the code editor window of the **Execute R Script** module with the following code. You can copy and paste this code from **CleanAdmissions.R** in the lab files folder for this lab.

   Tip: Do not copy and paste code from the lab document, as this can lead to errors caused by formatting. Instead, open the code file provided in the lab files, press CTRL+A to select all text, and press CTRL+C to copy it. Then place the cursor in the code editor pane for the Execute R Script module, press CTRL+A to select the existing code, and press CTRL+V to paste the copied code over the existing code.

   ```
   clean.admissions <- function(admissions){
     library(dplyr)
     admissions %>% mutate(admission_type_description =
                                   ifelse(admission_type_description %in%
                                          c('Not Available', 'NULL', 'Not
   Mapped'),
                                          'unknown',
   admission_type_description))
   }

   df <- maml.mapInputPort(1)
   df <- clean.admissions(df)
   maml.mapOutputPort('df')
   ```

   This code creates consistent coding for missing values by mapping any of the multiple missing values codes to *unknown*.

3. Add a **Join Data** module to the experiment and connect the output of the **diabetic_data.csv** dataset to its **Dataset1** (left hand) input; and the **Results Dataset** (left hand) output of the **Execute R Script** module to its **Dataset2** (right hand) input. Then configure its properties as follows:
   - **Join key columns for L Column names**: admission_type_id
   - **Join key columns for R Column names**: admission_type_id
   - **Match case**: checked
   - **Join type**: Left Outer Join
   - **Keep right key columns in joined table**: unchecked
4. Add another **Execute R Script** module to the experiment, and connect the output of the **Join Data** module to its **Dataset1** (left hand) input. Then replace the default R script with the following code, which you can copy and paste from the **SetDiagCodes.R** file in the lab files folder for this module:

   ```
   code.table <- function(){
     c(rep('infections', 139),
       rep('neoplasms', (239 - 139)),
       rep('endocrine', (279 - 239)),
   ```

```
    rep('blood', (289 - 279)),
    rep('mental', (319 - 289)),
    rep('nervous', (359 - 319)),
    rep('sense', (389 - 359)),
    rep('circulatory', (459-389)),
    rep('respiratory', (519-459)),
    rep('digestive', (579 - 519)),
    rep('genitourinary', (629 - 579)),
    rep('pregnancy', (679 - 629)),
    rep('skin', (709 - 679)),
    rep('musculoskeletal', (739 - 709)),
    rep('congenital', (759 - 739)),
    rep('perinatal', (779 - 759)),
    rep('ill-defined', (799 - 779)),
    rep('injury', (999-799))
  )
}

set.codes <- function(x, code){
  i <- 1
  print(str(code))
  for(num in x){
    if(num == 'unknown' | is.na(num) | num == '?') {
      x[i] <- 'unknown'
    }else{
      fchar <- toupper(substr(num, 1, 1))
      ifelse(fchar == 'E', x[i] <- 'injury',
             ifelse(fchar == 'V', x[i] <- 'suplemental',
                    x[i] <- codes[as.integer(num)]
             ))
    }
    i <- i + 1
  }
  x
}

df <- maml.mapInputPort(1)
codes <- code.table()
diagCols <- c("diag_1", "diag_2", "diag_3")
df[,diagCols] <- lapply(df[, diagCols], set.codes, codes)
maml.mapOutputPort('df')
```

This code deals with some non-numeric coding for special diagnostic codes, and transforms numeric codes by indexing a vector of the text codes.

5. Add another **Execute R Script** module to the experiment, and connect the **Results** dataset (left) output of the **Execute R Script** module that sets the diagnostic codes to its **Dataset1** (left-most) input. Then replace its default R script with the following code (which you can copy and paste from **SetReadmit.R**):

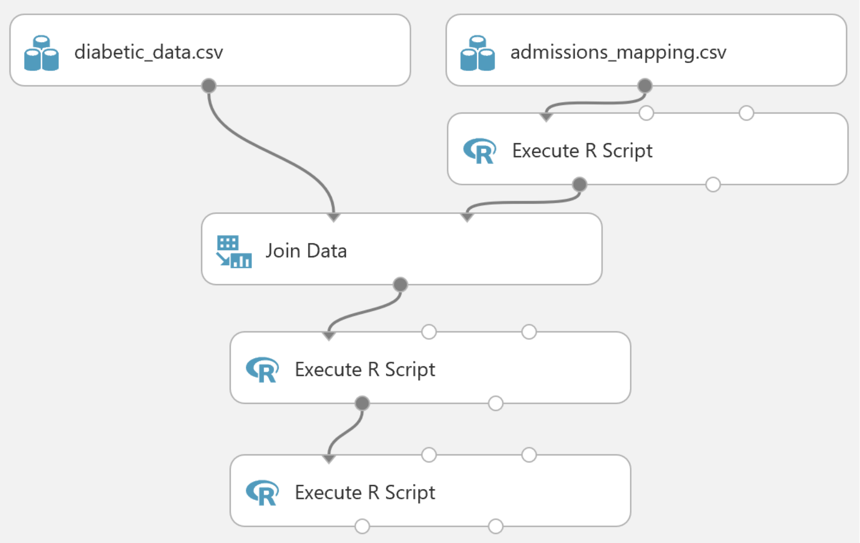
```
class.readmit <- function(x){
  out <- rep("NO", length(x))
  out[which(x != "NO")] <- "YES"
  out
}

df <- maml.mapInputPort(1)
df$readmitted <- class.readmit(df$readmitted)

maml.mapOutputPort('df')
```

This code transforms the three categories in the label column (NO, <30, >30) to two categories. This is required to support a two-class classification model.

6. Verify that your experiment looks like this:



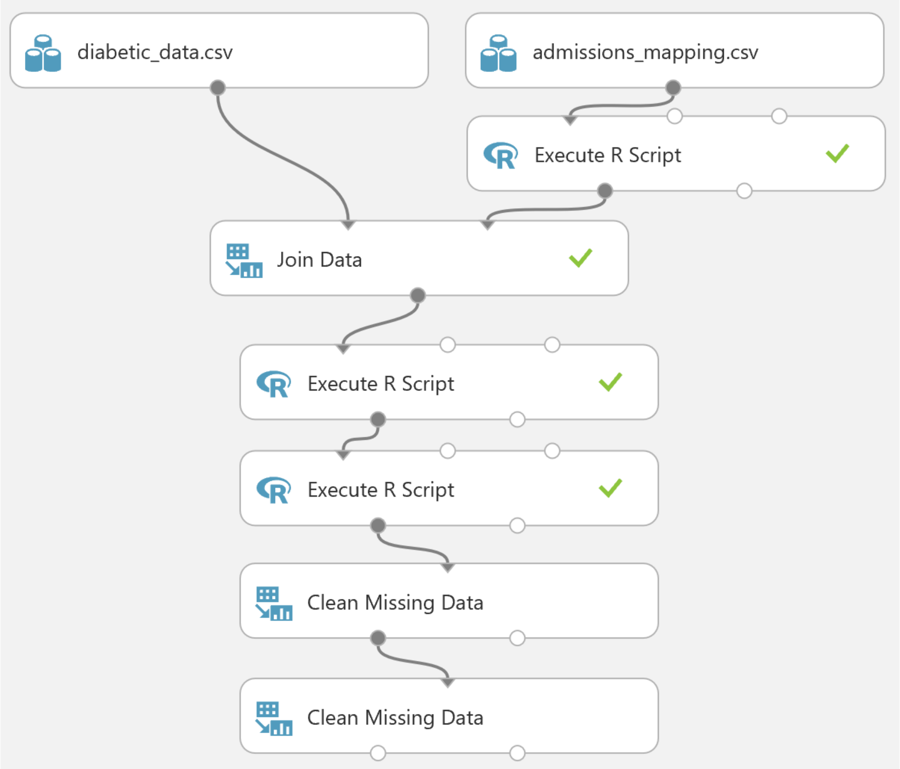
7. Save and run the experiment, and visualize the **Results Dataset** (left) output of the last **Execute R Script** module. Then select the **readmitted** column header and verify that the column contains only two values (**YES** and **NO**).
8. Add a **Clean Missing Data** module, connect the **Results Dataset** (left) output of the last **Execute R Script** module to its input, and set its properties as follows:
    - **Columns to be cleaned**: All string columns
    - **Minimum missing value ratio**: 0
    - **Maximum missing value ratio**: 1
    - **Cleaning mode**: Custom substitution value
    - **Replacement value**: *unknown*
    - **Generate missing value indicator column**: unchecked

   This replaces any missing string values with the text “unknown”.

9. Add a second **Clean Missing Data** to the experiment, connect the **Cleaned Dataset** (left) output of the first **Clean Missing Data** module to its input, and set its properties as follows:
    - **Columns to be cleaned**: All numeric columns
    - **Minimum missing value ratio**: 0
    - **Maximum missing value ratio**: 1
    - **Cleaning mode**: Custom substitution value
    - **Replacement value**: 0
    - **Generate missing value indicator column**: unchecked

This replaces any missing numeric values with 0.

10. Verify that your experiment looks this like:



11. Save and run the experiment, and visualize the output of the last **Clean Missing Data** module to verify that the data includes no missing values.
12. Add a **Select Columns in Dataset** module, and connect the **Cleaned Dataset** (left) output of the last **Clean Missing Data** module to its input. Then use the column selector to exclude the following columns:
    - admission_type_id
    - encounter_id
    - patient_nbr

    This removes columns that contain no useful information.

13. Add a **Normalize Data** module, connect the output of the **Select Columns in Dataset** module to its input, and set its properties as follows:
    - **Transformation method**: ZScore
    - **Use 0 for constant columns when checked**: checked
    - **Columns to transform**:
        - time_in_hospital
        - num_lab_procedures
        - num_procedures
        - num_medications
        - number_outpatient
        - number_emergency
        - number_inpatient

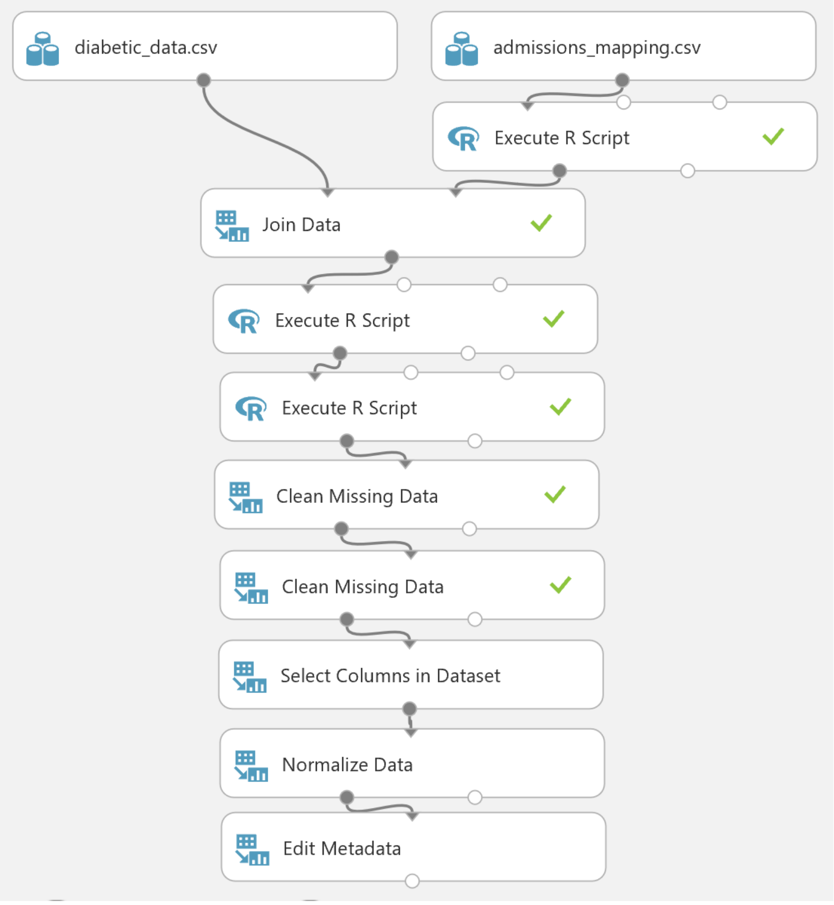
- number_diagnoses

This normalizes the continuous numeric columns in the dataset. Some other numeric columns are actually proxies for categorical variables, and should not be normalized.

14. Add an **Edit Metadata** module, and connect the **Transformed Data** (left) output of the **Normalize Data** module to its input. Then set its properties as follows:
    - **Column**: All string columns
    - **Data type**: Unchanged
    - **Categorical**: Make categorical
    - **Fields**: Unchanged
    - **New column named**: *leave blank*

    This converts string features to categorical variables.

15. Verify that your experiment looks like this:



16. Run the experiment and visualize the output of the last **Edit Metadata** module. Then select the column headings for the numeric columns you scaled to verify that values have been scaled, and select a string column to verify that it is now a categorical feature.

Preparing the Dataset with Python

Follow these steps to prepare the dataset using the tools in Azure ML and Python.

Note: If you prefer to work with R, complete the preceding exercise *Preparing the Dataset with R*.

1. Search for the **Execute Python Script** module, and add it to the experiment. Then connect the output of the **admissions_mapping.csv** dataset to its **Dataset1** (left) input.
2. Select the **Execute Python Script** module, set the **Python Version** to the latest available version of Python, and then replace the default Python script with the following code, which you can copy and paste from the **CleanAdmissions.py** file in the lab files folder for this module:

   Tip: Do not copy and paste code from the lab document, as this can lead to errors caused by formatting. Instead, open the code file provided in the lab files, press CTRL+A to select all text, and press CTRL+C to copy it. Then place the cursor in the code editor pane for the Execute R Script module, press CTRL+A to select the existing code, and press CTRL+V to paste the copied code over the existing code.

   ```
   def prep_admissions(admissions):
       import pandas as pd
       admissions['admission_type_description'] = ['unknown' if ((x in
   ['Not Available', 'Not Mapped', 'NULL']) | (pd.isnull(x))) else x
                                                    for x in
   admissions['admission_type_description']]
       return admissions
   def azureml_main(df):
       df = prep_admissions(df)
       return df
   ```

   This code creates consistent coding for missing values by mapping any of the multiple missing values codes to *unknown*.

3. Add a **Join Data** module to the experiment and connect the output of the **diabetic_data.csv** dataset to its **Dataset1** (left hand) input; and the **Results Dataset** (left hand) output of the **Execute Python Script** module to its **Dataset2** (right hand) input. Then configure its properties as follows:
   - **Join key columns for L Column names**: admissions_type_id
   - **Join key columns for R Column names**: admissions_type_id
   - **Match case**: checked
   - **Join type**: Inner Join
   - **Keep right key columns in joined table**: unchecked
4. Add a **Clean Missing Data** module, connect the output of the **Join Data** module to its input, and set its properties as follows:
   - **Columns to be cleaned**: String, All
   - **Minimum missing value ratio**: 0
   - **Maximum missing value ratio**: 1
   - **Cleaning mode**: Custom substitution value
   - **Replacement value**: unknown
   - **Generate missing value indicator column**: unchecked

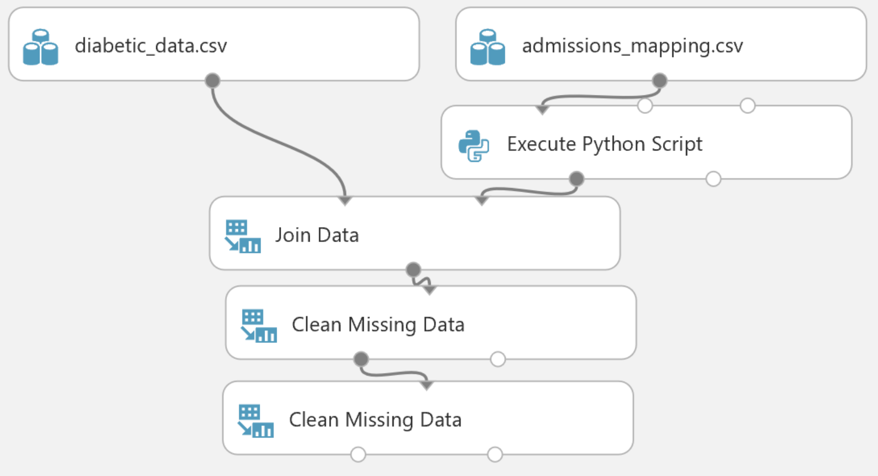
   This replaces any missing string values with the text “unknown”.

5. Add a **Clean Missing Data** to the experiment, connect the **Cleaned Dataset** (left) output of the first **Clean Missing Data** module to its input, and set its properties as follows:
   - **Columns to be cleaned**: Numeric, All
   - **Minimum missing value ratio**: 0

- **Maximum missing value ratio**: 1
- **Cleaning mode**: Custom substitution value
- **Replacement value**: 0
- **Generate missing value indicator column**: unchecked

This replaces any missing numeric values with 0.

6. Verify that your experiment looks like this:



7. Save and run the experiment, and visualize the output of the last **Clean Missing Data** module to verify that the data includes no missing values.
8. Add an **Execute Python Script** module to the experiment, set it to use the latest version of Python, and connect the **Cleaned Dataset** (left) output of the last **Clean Missing Data** module to its **Dataset1** (left hand) input. Then replace the default script with the following code, which you can copy and paste from the **SetDiagCodes.py** file in the lab files folder for this module:

```
def create_map():
    ## List of tuples with name and number of repititons.
    name_list = [('infections', 139),
                ('neoplasms', (239 - 139)),
                ('endocrine', (279 - 239)),
                ('blood', (289 - 279)),
                ('mental', (319 - 289)),
                ('nervous', (359 - 319)),
                ('sense', (389 - 359)),
                ('circulatory', (459-389)),
                ('respiratory', (519-459)),
                ('digestive', (579 - 519)),
                ('genitourinary', (629 - 579)),
                ('pregnancy', (679 - 629)),
                ('skin', (709 - 679)),
                ('musculoskeletal', (739 - 709)),
                ('congenital', (759 - 739)),
                ('perinatal', (779 - 759)),
                ('ill-defined', (799 - 779)),
                ('injury', (999 - 799))]
    ## Loop over the tuples to create a dictionary to map codes
    ## to the names.
    out_dict = {}
    count = 1
    for name, num in name_list:
        for i in range(num):
          out_dict.update({str(count): name})
          count += 1
    return out_dict


def map_codes(df, codes):
    import pandas as pd
    col_names = df.columns.tolist()
    for col in col_names:
        temp = []
        for num in df[col]:
            if ((num is None) | (num in ['unknown', '?']) |
(pd.isnull(num))): temp.append('unknown')
            elif(num.upper()[0] == 'V'): temp.append('supplemental')
            elif(num.upper()[0] == 'E'): temp.append('injury')
            else:
                lkup = num.split('.')[0]
                temp.append(codes[lkup])
        df.loc[:, col] = temp
    return df

def azureml_main(df):
    col_list = ['diag_1', 'diag_2', 'diag_3']
    codes = create_map()
    df[col_list] = map_codes(df[col_list], codes)
    return df
```

This code deals with some non-numeric coding for special diagnostic codes, and transforms numeric codes by indexing a vector of the text codes.

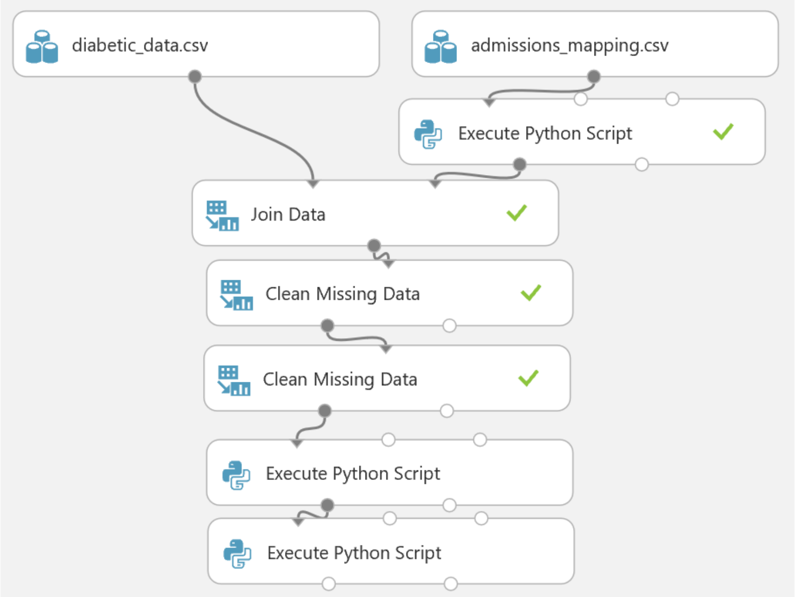
9. Add another **Execute Python Script** module to the experiment, and connect the **Results** dataset (left) output of the **Execute Python Script** module that sets the diagnostic codes to its **Dataset1** (left-most) input. Then replace its default script with the following code (which you can copy and paste from **SetReadmit.py**):

```
def set_readmit_class(x):
    return ['NO' if (y == 'NO') else 'YES' for y in x]

def azureml_main(df):
    df['readmitted'] = set_readmit_class(df['readmitted'])
    return df
```

This code transforms the three categories in the label column (No, <30, >30) to two categories. This is required to support a two-class classification model.

10. Verify that your experiment looks like this:



11. Save and run the experiment, and visualize the output of the last **Execute Python Script** module. Then select the **readmitted** column header and verify that the column contains only two values (**YES** and **NO**).
12. Add a **Select Columns in Dataset** module, and connect the **Result Dataset** (left) output of the last **Execute Python Script** module to its input. Then use the column selector to exclude the following columns as shown below:
    - admission_type_id
    - encounter_id
    - patient_nbr

    This removes columns that contain no useful information.

13. Add a **Normalize Data** module, connect the output of the **Select Columns in Dataset** module to its input, and set its properties as follows:
    - **Transformation method**: ZScore
    - **Use 0 for constant columns when checked**: checked
    - **Columns to transform**:
        - time_in_hospital
        - num_lab_procedures
        - num_procedures
        - num_medications
        - number_outpatient
        - number_emergency
        - number_inpatient
        - number_diagnoses

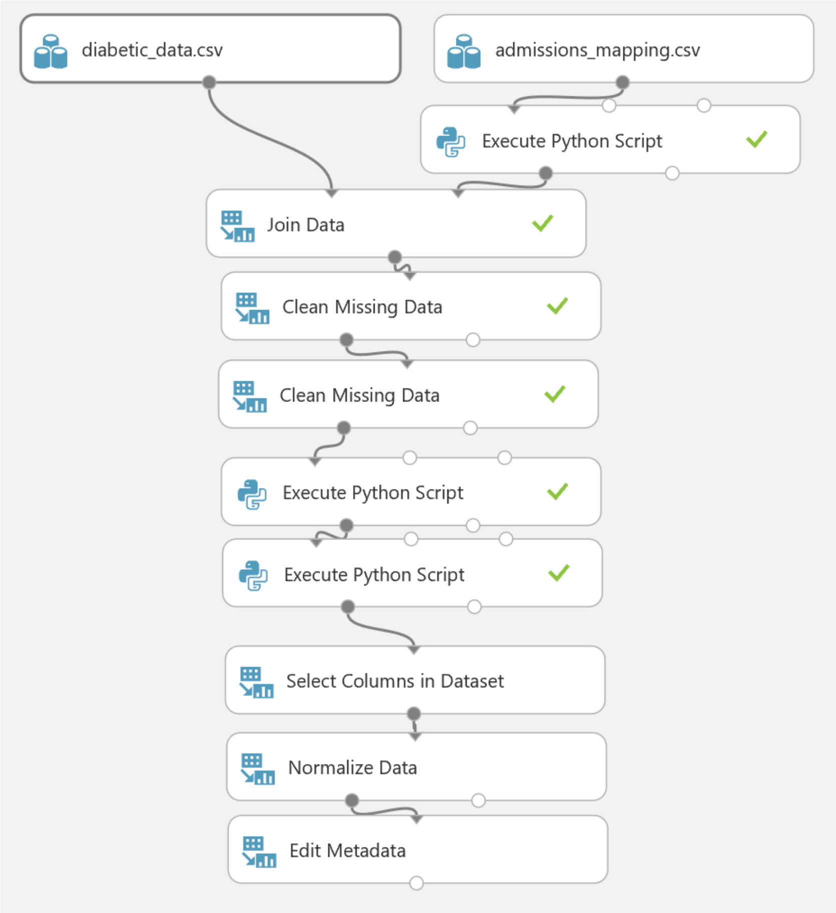
    This normalizes the continuous numeric columns in the dataset. Some other numeric columns are actually proxies for categorical variables, and should not be normalized.

17. Add an **Edit Metadata** module, and connect the **Transformed Data** (left) output of the **Normalize Data** module to its input. Then set its properties as follows:
    - **Column**: All string columns

- **Data type**: Unchanged
- **Categorical**: Make categorical
- **Fields**: Unchanged
- **New column named**: *leave blank*

This converts string features to categorical variables.

18. Verify that your experiment looks like this:



14. Run the experiment and visualize the output of the last **Edit Metadata** module. Then select the column headings for the numeric columns you scaled to verify that values have been scaled, and select a string column to verify that it is now a categorical feature.

Visualize the Data with R

In this exercise you will use custom R code to visualize the data set and examine the relationships. This data set has both numeric and categorical (string) features. The label column is named **readmitted**. The label column can have two values "YES" and "NO". Having two values in the label makes this a two-class or binary classification problem.

Visualization helps identify features which will help separate the two classes. These features will exhibit different values when conditioned by the two classes of the label. Other features will show little or no difference when conditioned by the label. Yet, other classes may be degenerate with nearly all values the same regardless of label value.

Note: If you prefer to work with Python, complete the *Visualize the Data with Python* exercise.

1. Add a **Convert to CSV** module to the experiment and connect the output of the **Edit Metadata** module to its input. Then run the experiment.
2. Right-click the output of the **Convert to CSV** module and in the **Open in a new workbook** submenu, click **R**.
3. In the new browser tab that opens, at the top of the page, rename the new workbook **Diabetes Classification Visualization**.
4. Review the code that has been generated automatically. The code in the first cell loads a data frame from your experiment. The code in the second cell uses the **head** command to display the first few rows of data.
5. On the **Cell** menu, click **Run All** to run all of the cells in the notebook, and then view the output.
6. On the **Insert** menu, click **Insert Cell Below** to add a new cell to the notebook.
7. In the new cell, enter the following code (which you can copy and paste from **VisualizeDiabetes.R**):

```
bar.plot <- function(x, cut = 200){
  require(ggplot2)
  if(is.factor(diabetes[, x]) | is.character(diabetes[, x]) & (x !=
'readmitted') & x != 'readmitted'){
    colList = c('readmitted', x)
    print(paste('*** The col name = ', x))
    diabetes[, colList] = lapply(diabetes[, colList], as.factor)
    sums <- summary(diabetes[, x], counts = n())
    msk <- names(sums[which(sums > cut)])
    tmp <- diabetes[diabetes[, x] %in% msk, colList]
    capture.output(
      if(strsplit(x, '[-]')[[1]][1] == x){
        g <- ggplot(tmp, aes_string(x)) +
          geom_bar() +
          facet_grid(. ~ readmitted) +
          ggtitle(paste('Readmissions by level of', x))
        print(g)
      }
    )
  }
}

box.plot <- function(x){
  require(ggplot2)
  if(is.numeric(diabetes[, x])){
    ggplot(diabetes, aes_string('readmitted', x)) +
      geom_boxplot() +
      ggtitle(paste('Readmissions by', x))
  }
}

hist.plot <- function(x){
  require(ggplot2)
  if(is.numeric(diabetes[, x])){
    capture.output(
      ggplot(diabetes, aes_string(x)) +
        geom_histogram() +
        facet_grid(readmitted ~ .) +
        ggtitle(paste('Readmissions by', x))
    )
  }
}
```

8. Ensure that the cursor is in the cell containing the function definitions above, and then on the **Cell** menu, click **Run and Select Below** (or click the ▶ button on the toolbar).
9. After the code has finished running, in the new empty cell at the bottom of the notebook, enter the following code, and then run the new cell and wait for the code to finish running:

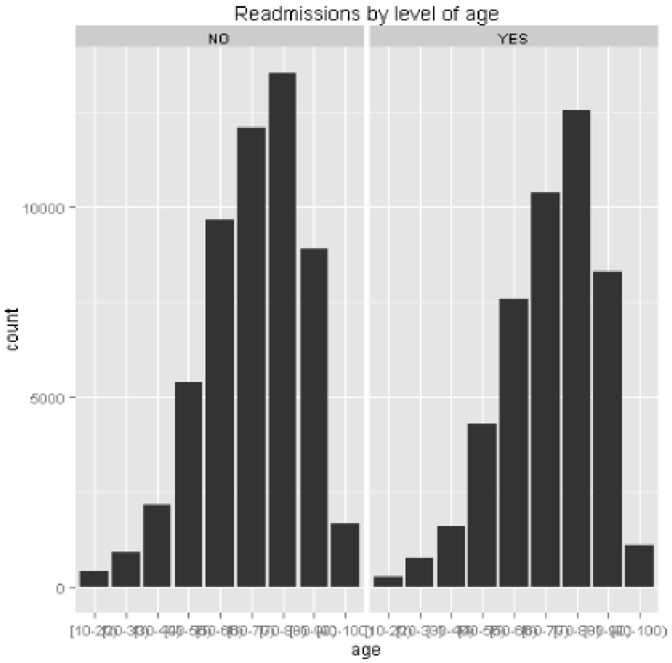
```
diabetes <- dat
col.names = names(diabetes)
col.names = c(col.names, names(diabetes))
lapply(col.names, bar.plot)
lapply(col.names, box.plot)
lapply(col.names, hist.plot)
```

Tip: The code may take a few minutes to run. While it is running, a ● symbol is displayed at the top right under the R logo, and when the code had finished running this changes to a O symbol

10. View the plots that are produced.

Tip: Click the grey bar on the left edge of the output pane containing the plots to see all of the output in the main notebook window.

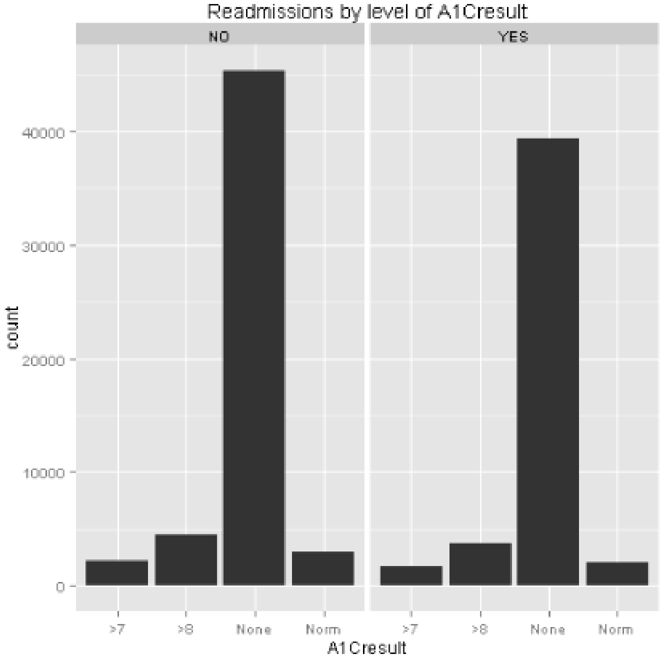
You will see a conditioned bar plot for each of the categorical features. Examine these bar plots to develop and understanding of the relationship between these categorical features and the label values. Note the bar plot of the **age** feature, as shown below:



Note, the vertical (frequency) scale is identical for each value of the label ('YES', 'NO'). There are more total cases not readmitted ('NO') than readmitted ('YES'). There are more total cases not readmitted than readmitted. Age is divided into bin by decade, e.g. 10-20, 20-30.

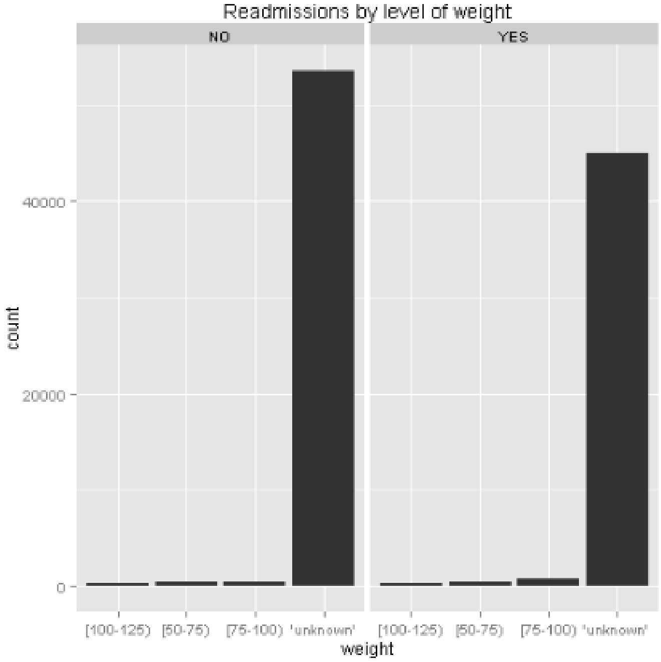
There is a slight skew toward older ages for patients who are readmitted. However, the effect is subtle, indicating that this feature contains only minimal information to separate the classes.

11. Locate the bar plot of the **A1 Cresult** feature, as shown below:



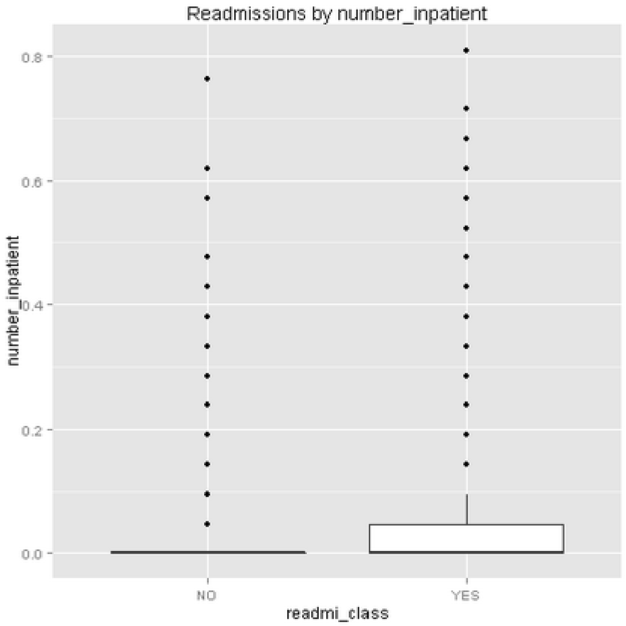
Examine this plot and note the reasons why this feature is unlikely to help predict the label cases. First, the relative frequencies of the four categories of the feature are nearly identical for the two label values. Second, all of the categories of the feature, except 'none', are fairly infrequent, meaning that even if there were significant difference these values would only separate a minority of cases.

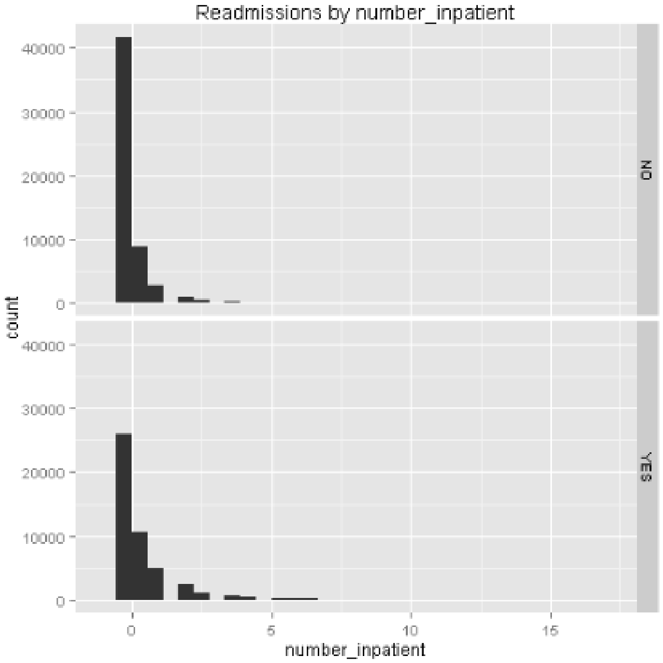
12. Scroll though the bar plots and locate some features where only one category is plotted, indicating there are less than 100 cases with any other value. In other cases, degenerate features will only have a single value for all cases. In either case, these features are unlikely to separate the label cases. An example is shown in the figure below for the **weight** feature:



Note that 'unknown' is the predominant category for this feature. Clearly, this feature is unlikely to contain significant information.

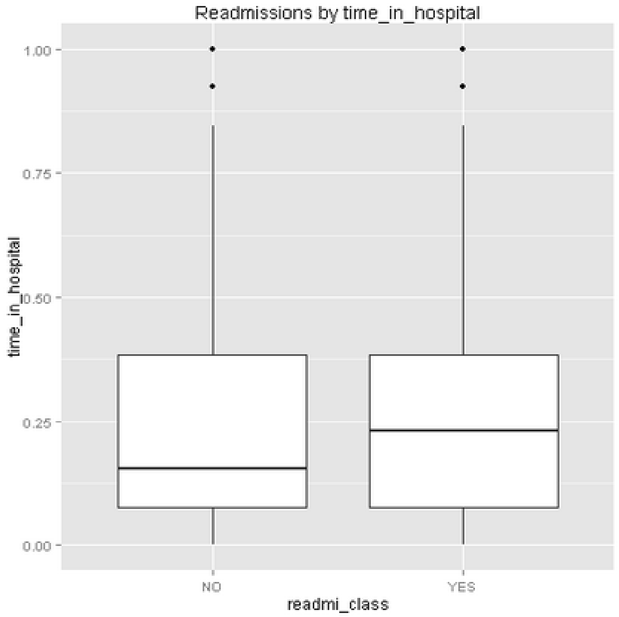
13. Locate the box plots and corresponding histograms of each numeric variable conditioned on the levels of the label. Examine these box plots to develop an understanding of the relationship between these features and the label values.
14. Locate the box plots and histograms for **number_inpatient** as shown below:

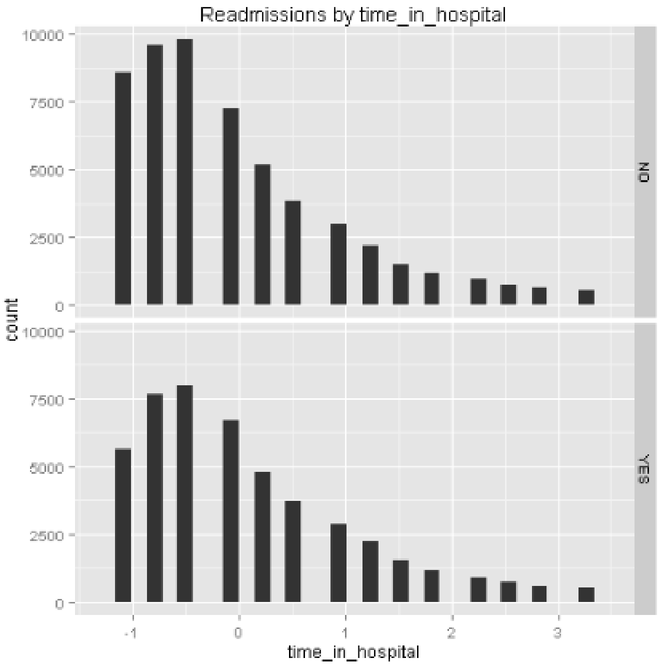




Examine these plots and note the differences between the values for patients who have been readmitted ('YES') and patients who have not been readmitted ('NO'). These values have been normalized or scaled. In both cases the median (black horizontal bar) in the box plot is near zero, indicating significant overlap between the cases. For readmitted patients ('YES'), the two upper quartiles (indicated by the box and the vertical line or whisker) shows little overlap with the other case (‘NO’). Examine the histograms, noticing that the distribution of readmitted patients has a slightly longer tail than for the patients never readmitted. Based on these observations of overlap and difference you can expect **number_inpatient** to separate only a few cases, and thus be a poor feature.

15. Next, examine the box plots and histograms for the **time_in_hospital** feature as shown below:





Examine these plots and note the differences between the values for patients who have been readmitted (‘YES’) and patients who are not readmitted (‘NO’). These values have been normalized or scaled. There is a bias or skew toward longer time in hospital for patients who are readmitted. However, given the overlap in values, this feature has only limited power to separate the two classes of the label.

16. Save the Jupyter notebook. Then on the **File** menu, click **Close and Halt** to close the tab, returning to the experiment in Azure ML Studio.

Visualize the Data with Python

In this exercise you will create custom Python code to visualize the data set and examine the relationships. This data set has both numeric and categorical (string) features. The label column is named **readmitted**. The label column can have two values "YES" and "NO". Having two values in the label makes this a two-class or binary classification problem.

Visualization helps identify features which will help separate the two classes. These features will exhibit different values when conditioned by the two classes of the label. Other features will show little or no difference when conditioned by the label. Yet, other classes may be degenerate with nearly all values the same regardless of label value.

Note: If you prefer to work with R, complete the proceeding exercise, *Visualize the Data with R*.

1. Add a **Convert to CSV** module to the experiment and connect the output of the **Edit Metadata** module to its input. Then run the experiment.
2. Right-click the output of the **Convert to CSV** module and in the **Open in a new workbook** submenu, click **Python 3**.
3. In the new browser tab that opens, at the top of the page, rename the new workbook **Diabetes Classification Visualization**.
4. Review the code that has been generated automatically. The code in the first cell loads a data frame from your experiment. The code in the second cell displays the data.
5. On the **Cell** menu, click **Run All** to run all of the cells in the notebook, and then view the output.
6. On the **Insert** menu, click **Insert Cell Below** to add a new cell to the notebook.
7. In the new cell, enter the following code (which you can copy and paste from **VisualizeDiabetes.py**):

```
def diabetes_bar(df):
    import matplotlib
    matplotlib.use('agg')  # Set backend
    import numpy as np
    import matplotlib.pyplot as plt

## Create a series of bar plots for the various levels of the
## string columns in the data frame by readmi_class.
    names = df.columns.tolist()
    for col in names:
        if(df[col].dtype not in [np.int64, np.int32, np.float64]):
            temp1 = df.ix[df.readmitted == 'YES', col].value_counts()
            temp0 = df.ix[df.readmitted == 'NO', col].value_counts()

            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax1 = fig.add_subplot(1, 2, 1)
            ax0 = fig.add_subplot(1, 2, 2)
            temp1.plot(kind = 'bar', ax = ax1)
            ax1.set_title('Values of ' + col + '\n for readmitted
patients')
            temp0.plot(kind = 'bar', ax = ax0)
            ax0.set_title('Values of ' + col + '\n for patients not
readmitted')
```

```
            fig.savefig('bar_' + col + '.png')

    return 'Done'

def diabetes_box(df):
    import matplotlib
    matplotlib.use('agg')  # Set backend
    import numpy as np
    import matplotlib.pyplot as plt


## Now make box plots of the columns with numerical values.
    names = df.columns.tolist()
    for col in names:
        if(df[col].dtype in [np.int64, np.int32, np.float64]):
            temp1 = df.ix[df.readmitted == 'YES', col]
            temp0 = df.ix[df.readmitted == 'NO', col]

            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax1 = fig.add_subplot(1, 2, 1)
            ax0 = fig.add_subplot(1, 2, 2)
            ax1.boxplot(temp1.as_matrix())
            ax1.set_title('Box plot of ' + col + '\n for readmitted
patients')
            ax0.boxplot(temp0.as_matrix())
            ax0.set_title('Box plot of ' + col + '\n for patients not
readmitted')
            fig.savefig('box_' + col + '.png')

    return 'Done'

def diabetes_hist(df):
    import matplotlib
    matplotlib.use('agg')  # Set backend
    import numpy as np
    import matplotlib.pyplot as plt

## Now make histograms of the columns with numerical values.
    names = df.columns.tolist()
    for col in names:
        if(df[col].dtype in [np.int64, np.int32, np.float64]):
            temp1 = df.ix[df.readmitted == 'YES', col]
            temp0 = df.ix[df.readmitted == 'NO', col]

            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax1 = fig.add_subplot(1, 2, 1)
            ax0 = fig.add_subplot(1, 2, 2)
            ax1.hist(temp1.as_matrix(), bins = 30)
            ax1.set_title('Histogram of ' + col + '\n for readmitted
patients')
            ax0.hist(temp0.as_matrix(), bins = 30)
            ax0.set_title('Histogram of ' + col + '\n for patients not
readmitted')
```

```
            fig.savefig('hist_' + col + '.png')

    return 'Done'
```

17. Ensure that the cursor is in the cell containing the function definitions above, and then on the **Cell** menu, click **Run and Select Below** (or click the ▶ button on the toolbar).
18. After the code has finished running, in the new empty cell at the bottom of the notebook, enter the following code, and then run the new cell and wait for the code to finish running:

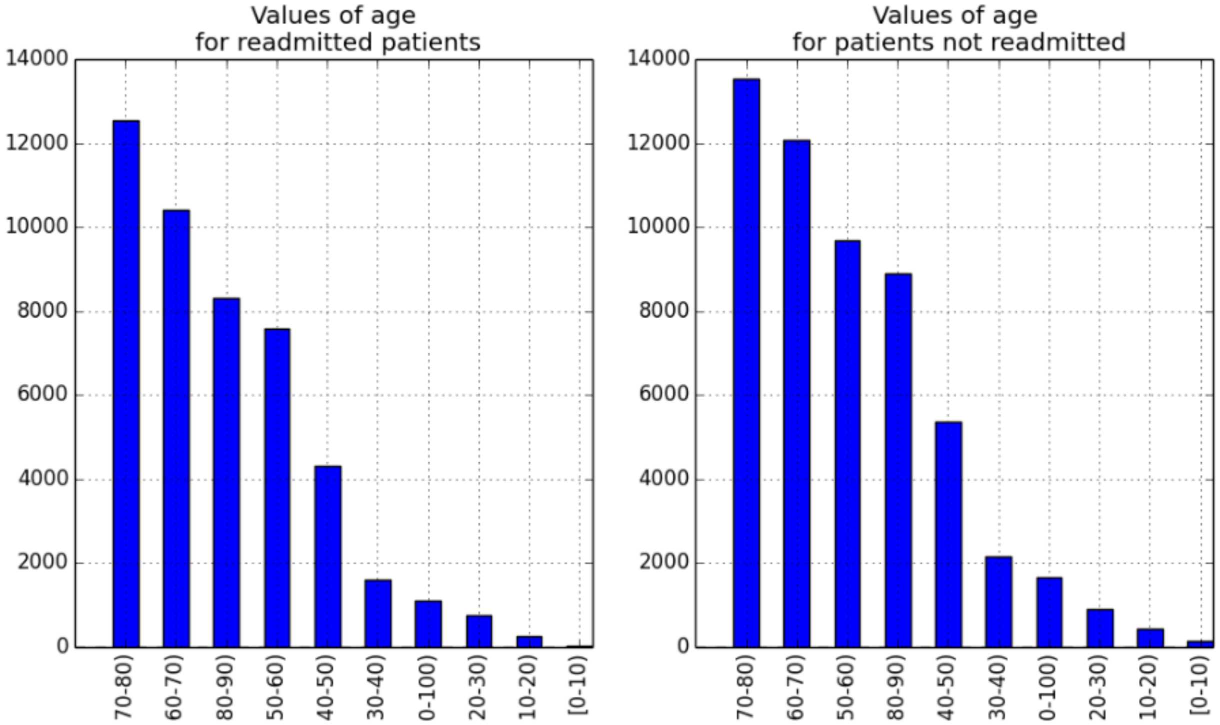
```
%matplotlib inline
diabetes_hist(frame)
diabetes_bar(frame)
diabetes_box(frame)
```

Tip: The code may take a few minutes to run. While it is running, a ● symbol is displayed at the top right under the Python logo, and when the code had finished running this changes to a ○ symbol

19. View the plots that are produced. You can ignore the error message that is displayed.

Tip: Click the grey bar on the left edge of the output pane containing the plots to see all of the output in the main notebook window.

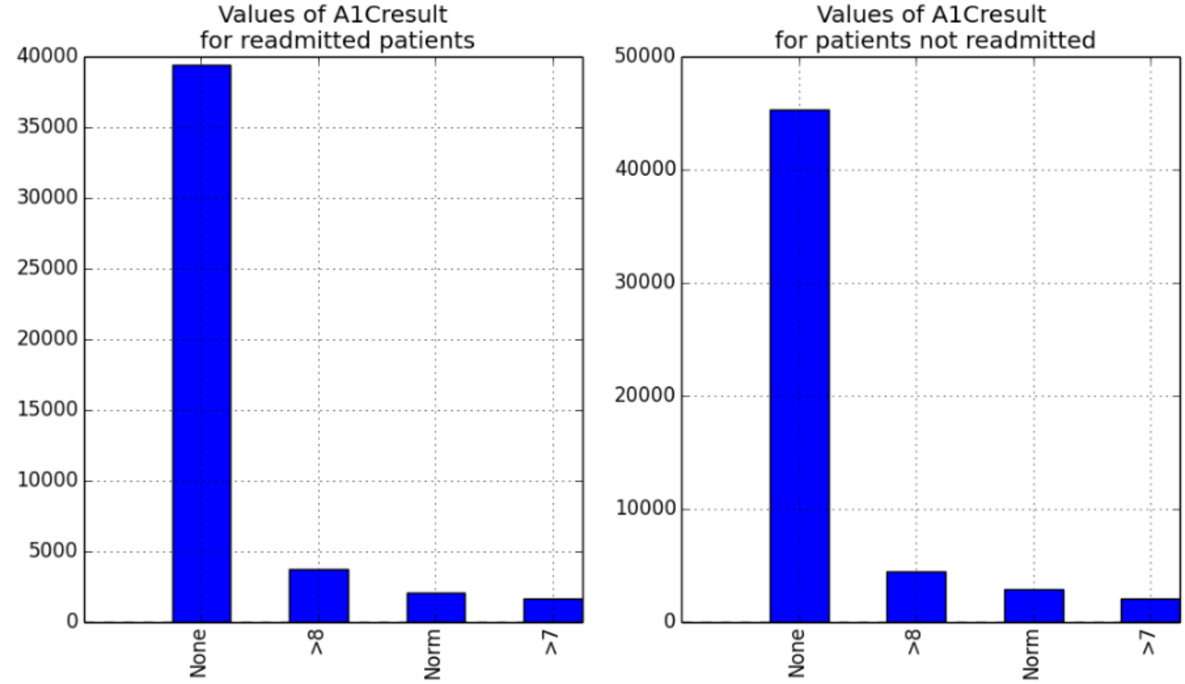
You will see a conditioned bar plot for each of the categorical features. Examine these bar plots to develop and understanding of the relationship between these categorical features and the label values. Note the bar plot of the **age** feature, as shown below:



Note, the vertical (frequency) scale is identical for each value of the label (‘readmitted’, ‘not readmitted’). There are more total cases not readmitted than readmitted. Age is divided into bin by decade, e.g. 10-20, 20-30.

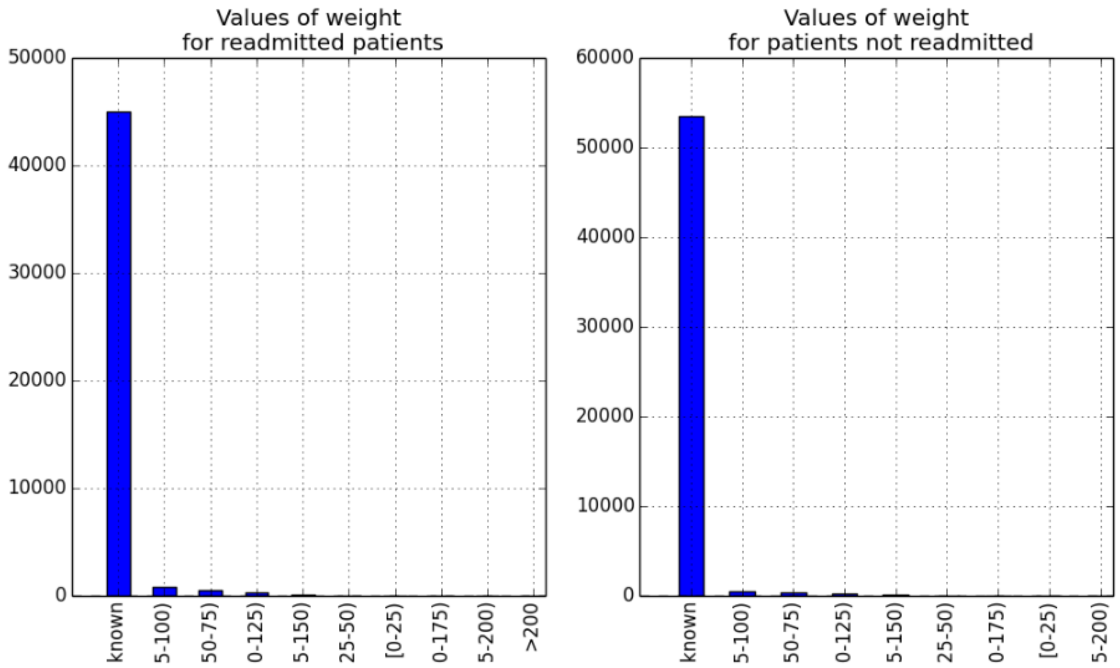
There is a slight skew toward older ages for patients who are readmitted. However, the effect is subtle, indicating that this feature contains only minimal information to separate the classes.

8. Locate the bar plot of the **A1 Cresult** feature, as shown below:



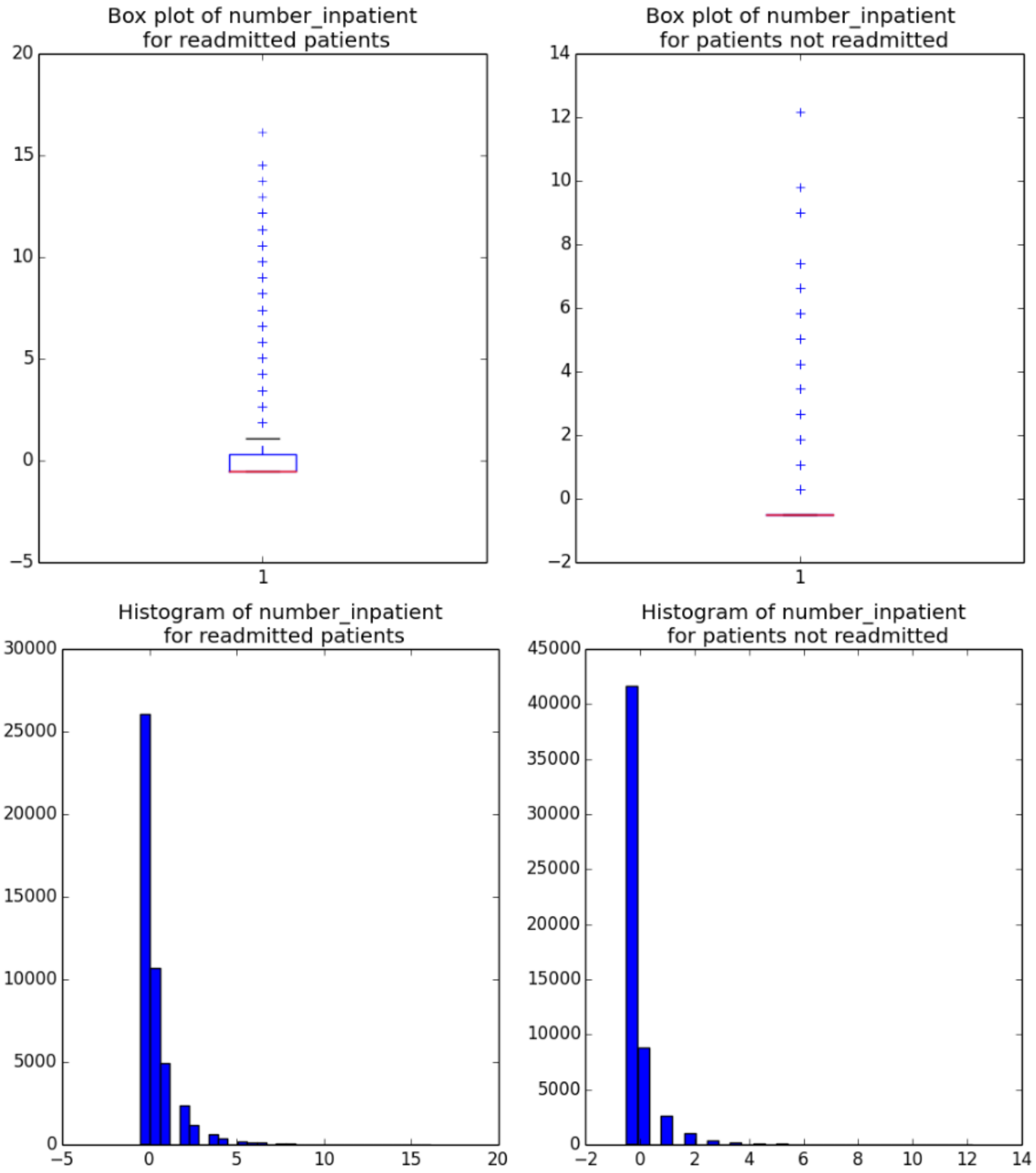
Examine this plot and note the reasons why this feature is unlikely to spate the label cases. First, the relative frequencies of the four categories of the feature are nearly identical for the two label values. Second, all categories of the feature, except none', are fairly infrequent, meaning that even if there were significant difference these values would only separate a minority of cases.

9. Scroll though the bar plots and locate some features where only one category is plotted, indicating there are less than 100 cases with any other value. In other cases, degenerate features will only have a single value for all cases. In either case, these features are unlikely to separate the label cases. An example is shown in the figure below for the **weight** feature:



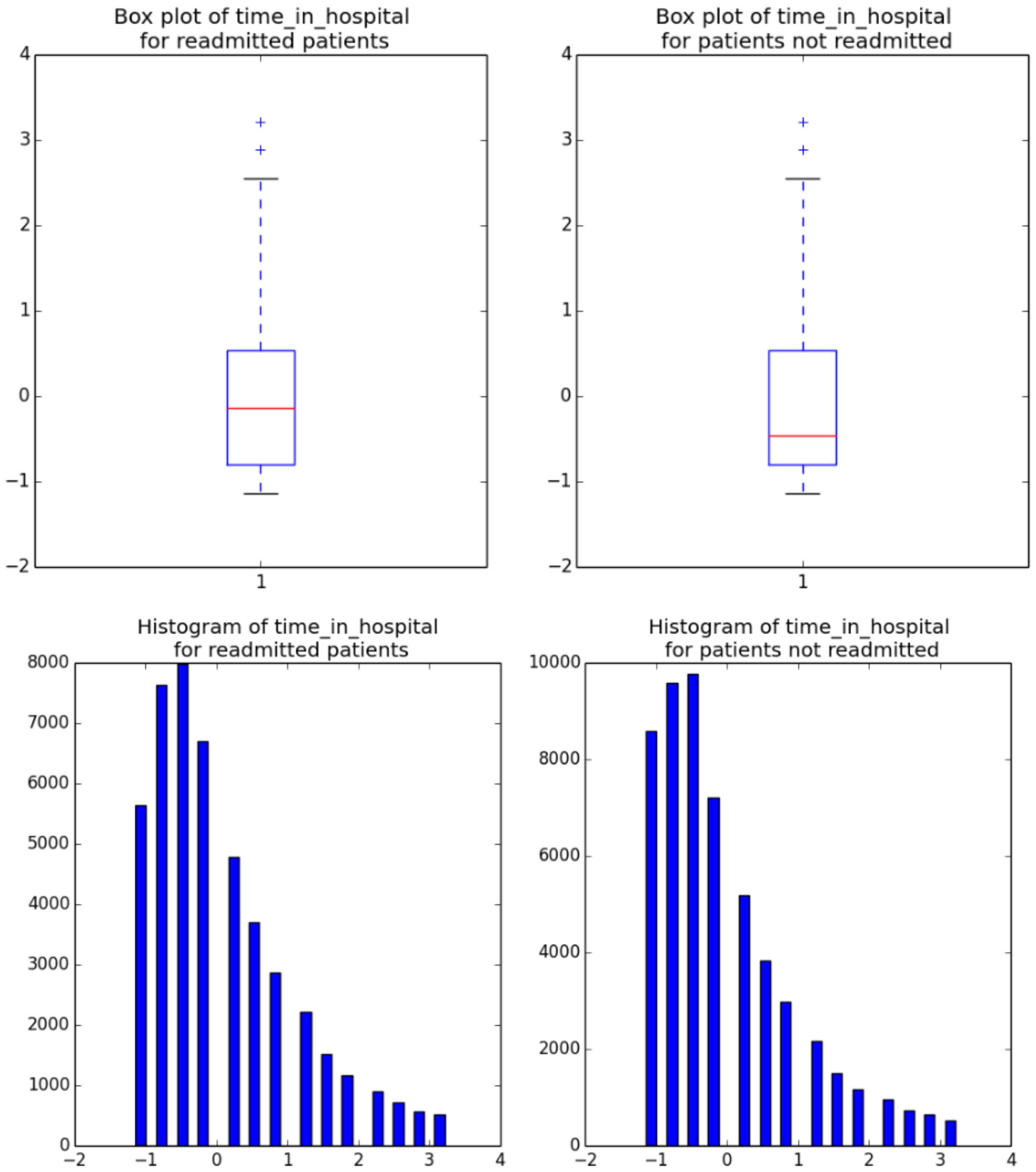
Note that ‘unknown’ is the predominant category for this feature. Clearly, this feature is unlikely to contain significant information.

10. Locate the box plots and corresponding histograms of each numeric variable conditioned on the levels of the label. Examine these box plots to develop and understanding of the relationship between these features and the label values.
11. Locate the box plots and histograms for **number_inpatient** as shown below:



Examine these plots and note the differences between the values for patients who have been readmitted and patients who have not been readmitted. These values have been normalized or scaled. In both cases the median (black horizontal bar) in the box plot is near zero, indicating significant overlap between the cases. For readmitted patients, the two upper quartiles (indicated by the box and the vertical line or whisker) shows little overlap with the other case. Examine the histograms, noticing that the distribution of readmitted patients has a slightly longer tail than for the patients never readmitted. Based on these observations of overlap and difference you can expect **number_inpatient** to separate only a few cases, and thus be a poor feature.

12. Next, examine the box plots and histograms for the **time_in_hospital** feature as shown below:



Examine these plots and note the differences between the values for patients who have been readmitted and patients who are not readmitted. These values have been normalized or scaled. There is a bias or skew toward longer time in hospital for patients who are readmitted. However, given the overlap in values, this feature has only limited power to separate the two classes of the label.

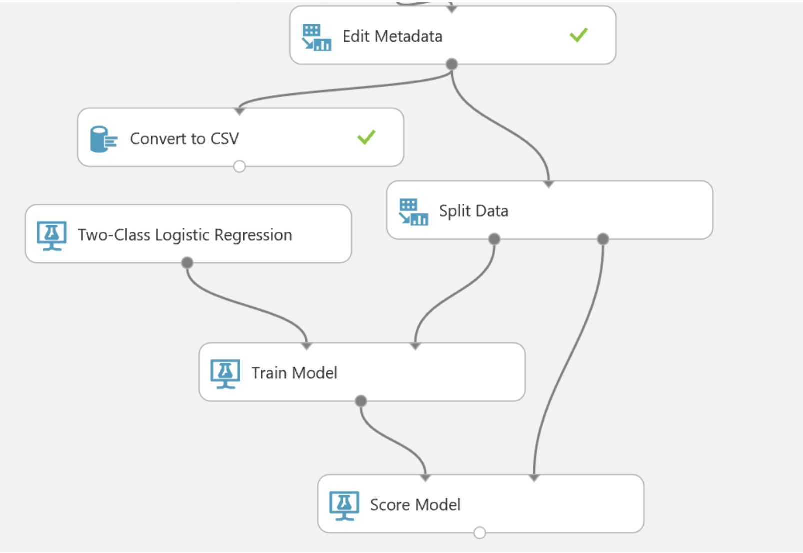
13. Save the Jupyter notebook. Then on the **File** menu, click **Close and Halt** to close the tab, returning to the experiment in Azure ML Studio.

Building a Classification Model

Now that you have investigated the relationships in the data you will build and evaluate a machine learning model. In this exercise you will use two-class logistic regression as the machine learning classifier.

Create a New Model

1. Add a **Split Data** module to the experiment, connect the output of the **Edit Metadata** module to its input port (bypassing the conversion to CSV, which was only required to load the data into a Jupyter notebook).
2. Set the **Properties** of the **Split Data** module as follows:
   - **Splitting mode:** Split Rows
   - **Fraction of rows in the first output:** 0.6
   - **Randomized split:** Checked
   - **Random seed:** 123
   - **Stratified split:** False
3. Add a **Two Class Logistic Regression** module to the experiment and set its properties as follows:
   - **Create trainer mode:** Single Parameter
   - **Optimization tolerance:** 1E-07
   - **L1 regularization weight:** 0.001
   - **L2 regularization weight:** 0.001
   - **Memory size for L-BFGS:** 20
   - **Random number seed:** 1234
   - **Allow unknown categorical levels:** checked
4. Add a **Train Model** module to the experiment, connect the **Untrained Model** output port of the **Two Class Logistic Regression** module to the **Untrained Model** (left) input port, and connect the **Results dataset1** (left) output port of the **Split Data** module to the **Dataset** (right) input port of the **Train model** module. Then configure its properties to select the **readmitted** column as the label column.
5. Add a **Score Model** module to the experiment. Then connect the output port of the of the **Train Model** module to its **Trained Model** (left) input port, and connect the **Results dataset2** (right) output port of the **Split Data** module to its **Dataset** (right) input port.
6. Verify that the bottom half of your experiment looks like this:



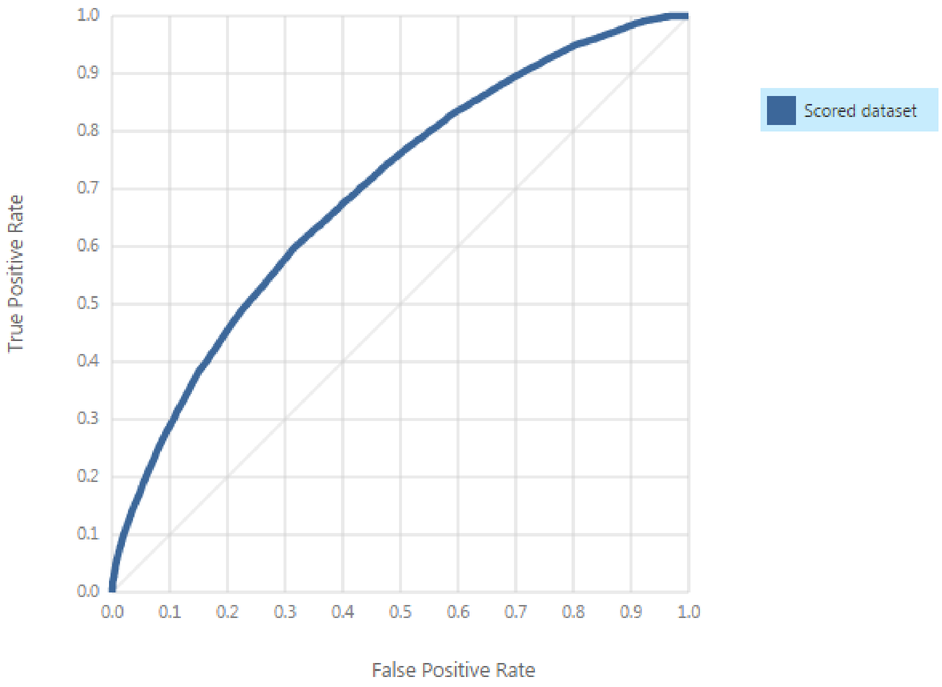
7. Save and run the experiment.

8. When the experiment has finished running, visualize the output of the **Score Model** module and view the **Scored Labels** column. Note that this contains the values predicted by the model – the actual values from the test data are in the **readmitted** column.

Evaluate the Model

1. Add an **Evaluate Model** module to the experiment. Then connect the output port of the **Score Model** module to the **Scored dataset** (left) input port of the **Evaluate Model** module.
14. Save and run the experiment.
15. When the experiment has finished, visualize the **Evaluation Result** port of the **Evaluate Model** module and scroll down to find the ROC curve as shown below.

Reminder: The goal of this classification problem is to prevent patients from requiring readmission to a hospital for their diabetic condition. Correctly identifying patients who are likely to require further treatment allows medical care providers to take actions which can prevent this situation. Patients with a false negative score will not be properly classified, and subsequently will require readmission to a hospital. Therefore, the recall statistic is important in this problem since maximizing recall minimizes the number of false negative scores.



Examine this ROC curve. Notice that the bold blue line is well above the diagonal grey line, indicating the model is performing better than random guessing.

20. Next, examine the statistics for the model evaluation, noting the area under the curve (AUC), the number of true positives, false positives, false negatives, and true negatives in the confusion matrix, and the accuracy, recall, precision, and F1 score.
    Notice the following:
    - Correctly classified values (**True Positive** + **True Negative**) values should outnumber the errors (**False Negative** + **False Positive**).
    - An **Accuracy** of greater than 0.5 indicates that the scores are correct more often than not.
    - There are nearly equal numbers of **True Positive** and **False Negative** scores.

- The relatively low **Recall** value results from a high number of **False Negative** values.
- The AUC metric quantifies the area under the curve – this should be greater than 0.5 in a model that performs better than random guessing.

16. Close the evaluation results.

Summary

In this lab, you have created a classification model using the Two-Class Logistic Regression algorithm. An initial evaluation of the model seems to indicate that it provides better results than random guessing, but more evaluation is needed, and it may be that the model could be improved. You will explore techniques for improving models later in this course.