APPLIED MACHINE LEARNING
LAB ACTIVITIES (LAB 2)

10/10/19

PREPARING DATA

This workbook is designed to guide you through the activities proposed for today's lab. As you will be working independently, feel free to proceed through the text at your own pace, spending more time on the parts that are less familiar to you. The workbook contains both hands-on tasks and links to learning materials such as tutorials, articles and videos. When you are unsure about something, feel free to ask our teaching assistants or use Internet resources to look for a solution. At the end of each section, there will be questions and exercises to verify your understanding of the presented information. You may need to do some research to answer the questions.

1. Visualisation

You must understand your data in order to get the best results from machine learning algorithms. The fastest way to learn more about your data is to use data visualisation. In this lab you will discover exactly how you can visualise your machine learning data in Python using Pandas. We will continue use the Pima Indians onset of diabetes dataset introduced in the previous lab.

Univariate Plots

We will look at three techniques that you can use to understand each attribute of your dataset independently.

- Histograms
- Density Plots
- Box and Whisker Plots

Histograms. A fast way to get an idea of the distribution of each attribute is to look at histograms. Histograms group data into bins and provide you a count of the number of observations in each bin. From the shape of the bins you can quickly get a feeling for whether an attribute is Gaussian, skewed or even has an exponential distribution. It can also help you see possible outliers.

We will do the same way as you used a Jupyter notebook during your previous lab session.

You will need to create a Jupyter notebook and upload the Pima Indians dataset.

Creating a new Jupyter Notebook is easy. Just use the New dropdown menu and select option Python 3 to open a new Jupyter Notebook for Python.

To upload the dataset, just use the Upload button on the main Jupyter notebook page.

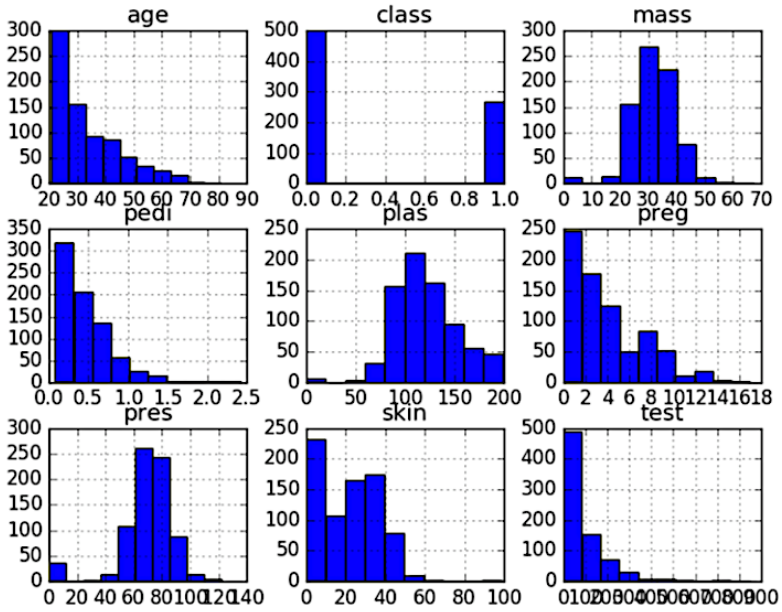
The notebook itself consists of cells. A first empty cell is already available after having created the new notebook.

Copy the below code block and past on to the first cell.

```
# Univariate Histograms
from matplotlib import pyplot
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
```

```
data.hist()
pyplot.show()
```

Executing code in this cell can be done by either clicking on the run cell button or hitting Shift + Return keys.



Histograms of each attributes

More details on $pandas.DataFrame.hist$ function are provided below.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.hist.html

Verify your understanding:

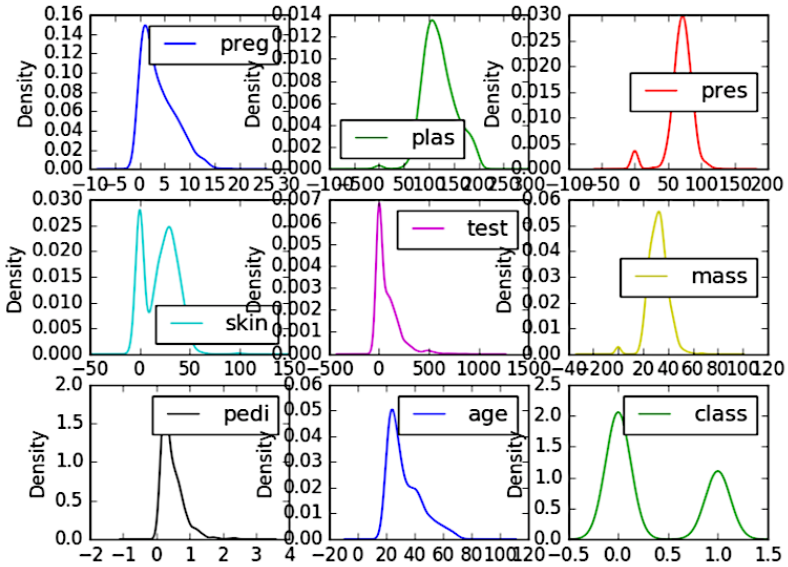
(a) Referring to the information on the above webpage, adjust figure size.

(b) *What is the default number of bins?*

Density Plots. Density plots are another way of getting a quick idea of the distribution of each attribute. The plots look like an abstracted histogram with a smooth curve drawn through the top of each bin, much like your eye tried to do with the histograms.

```
# Univariate Density Plots
from matplotlib import pyplot
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
pyplot.show()
```

Run the above code block and you will be able to see the distribution for each attribute is clearer than the histograms.



Density plots of each attribute

More details on $pandas.DataFrame.plot$ function are provided below.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

Verify your understanding:

(c) Referring to the information on the above webpage, adjust figure size.

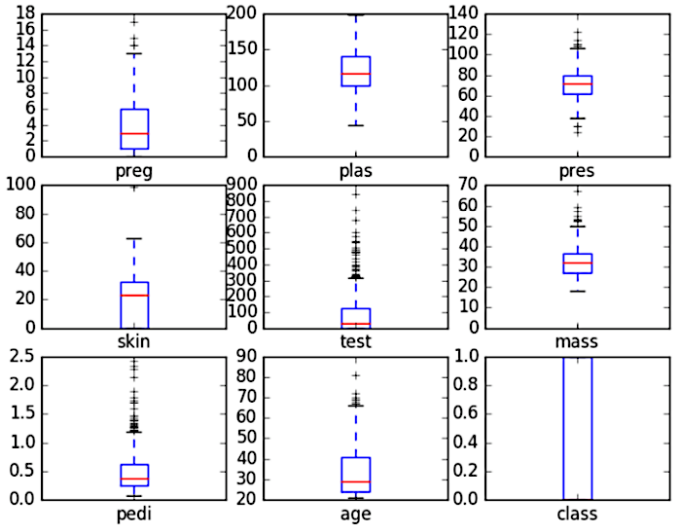
(d) *How many different types of plots the function provides?*

(e) *What argument should be changed to display Box and Whisker plot?*

Box and Whisker Plots. Another useful way to review the distribution of each attribute is to use Box and Whisker Plots or boxplots for short. Boxplots summarise the distribution of each attribute, drawing a line for the median (middle value) and a box around the 25th and 75th percentiles (the middle 50% of the data). The whiskers give an idea of the spread of the data and dots outside of the whiskers show candidate outlier values (values that are 1.5 times greater than the size of spread of the middle 50% of the data).

```
# Box and Whisker Plots
from matplotlib import pyplot
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False)
pyplot.show()
```

Run the above code block and you will be able to see that the spread of attributes is quite different. Some like age, test and skin appear quite skewed towards smaller values.



Verify your understanding:

(f) *Analyse the three univariate plots you generated and discuss what attributes should be used for the classification task.*

Multivariate Plots

We will look at the examples of two plots that show the interactions between multiple variables in your dataset.

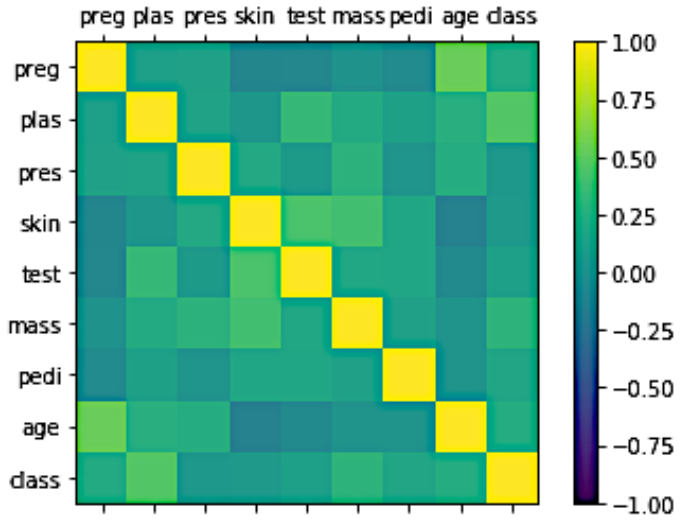
- Correlation Matrix Plot.
- Scatter Plot Matrix.

Correlation Matrix Plot. Correlation gives an indication of how related the changes are between two variables. If two variables change in the same direction they are positively correlated. If they change in opposite directions together (one goes up, one goes down), then they are negatively correlated. You can calculate the correlation between each pair of attributes. This is called a correlation matrix. You can then plot the correlation matrix and get an idea of which variables have a high correlation with each other. This is useful to know, because some machine learning algorithms like linear and logistic regression can have poor performance if there are highly correlated input variables in your data.

```
# Correction Matrix Plot
from matplotlib import pyplot
from pandas import read_csv
import numpy
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```

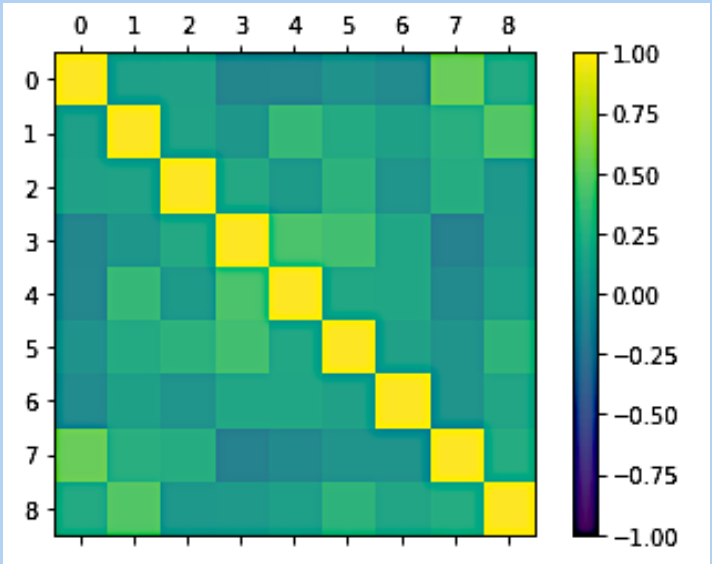
Ticks are the values used to show specific points on the coordinate axis. It can be a number or a string. Whenever we plot a graph, the axes adjust and take the default ticks. Matplotlib's default ticks are generally sufficient in common situations but are in no way optimal for every plot.

Run the above code block and you will be able to see that the matrix is symmetrical, i.e. the bottom left of the matrix is the same as the top right. This is useful as we can see two different views on the same data in one plot. We can also see that each variable is perfectly positively correlated with each other (as you would have expected) in the diagonal line from top left to bottom right.



Verify your understanding:

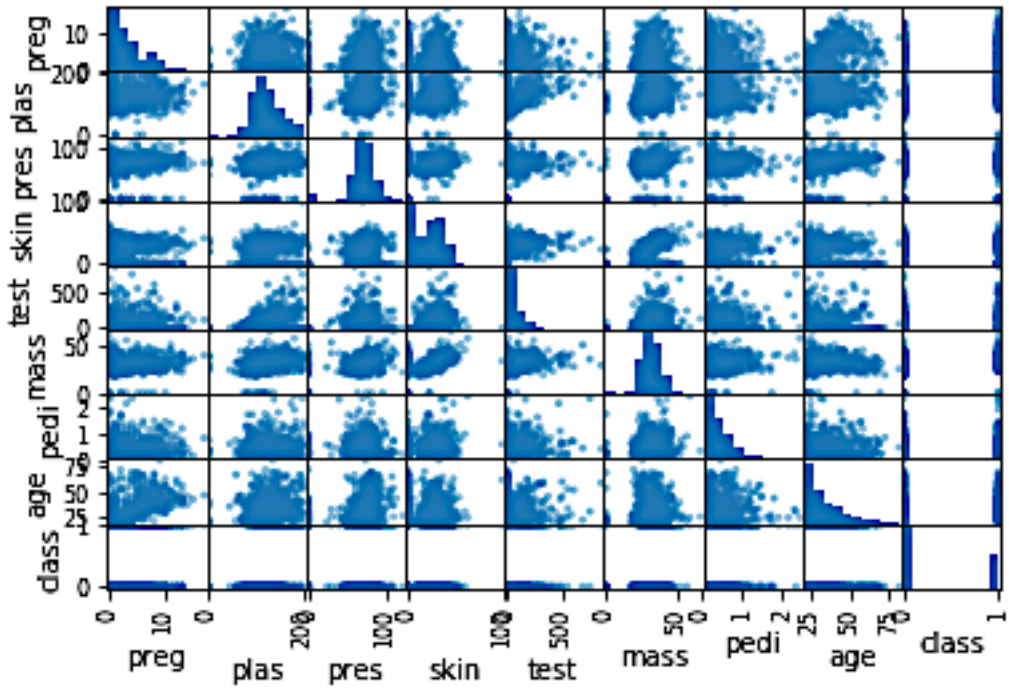
(g) *The above example is not generic in that it specifies the names for the attributes along the axes as well as the number of ticks. Make the plot more generic by removing the names like the one below.*



Scatter Plot Matrix. A scatter plot shows the relationship between two variables as dots in two dimensions, one axis for each attribute. You can create a scatter plot for each pair of attributes in your data. Drawing all these scatter plots together is called a scatter plot matrix. Scatter plots are useful for spotting structured relationships between variables, like whether you could summarise the relationship between two variables with a line. Attributes with structured relationships may also be correlated and good candidates for removal from your dataset.

```
# Scatterplot Matrix
from matplotlib import pyplot
from pandas import read_csv
from pandas.plotting import scatter_matrix
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
scatter_matrix(data, figsize=[20, 20])
pyplot.show()
```

Like the Correlation Matrix Plot above, the scatter plot matrix is symmetrical. This is useful to look at the pairwise relationships from different perspectives. Because there is little point of drawing a scatter plot of each variable with itself, the diagonal shows histograms of each attribute.



Verify your understanding:

(h) *Analyse the three multivariate plots you generated and discuss if the selected attributes remain the same.*

2. Data Preparation

You almost always need to pre-process your data. It is a required step. A difficulty is that different algorithms make different assumptions about your data and may require different transforms. Furthermore, when you follow all of the rules and prepare your data, sometimes algorithms can deliver better results without pre-processing.

Generally, I would recommend creating many different views and transforms of your data, then exercise a handful of algorithms on each view of your dataset. This will help you to flush out which data transforms might be better at exposing the structure of your problem in general.

Data Transforms

The scikit-learn library provides two standard idioms for transforming data. Each are useful in different circumstances. The transforms are calculated in such a way that they can be applied to your training data and any samples of data you may have in the future. The scikit-learn documentation has some information on how to use various different pre-processing methods:

- Fit and Multiple Transform.
- Combined Fit-And-Transform.

The Fit and Multiple Transform method is the preferred approach. You call the $fit()$ function to prepare the parameters of the transform once on your data. Then later you can use the $transform()$ function on the same data to prepare it for modelling and again on the test or validation dataset or new data that you may see in the future. The Combined Fit-And-Transform is a convenience that you can use for one off tasks. This might be useful if you are interested in plotting or summarising the transformed data. You can review the preprocess API in scikit-learn here.
https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

Rescale Data. When your data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale. Often this is referred to as normalisation and attributes are often rescaled into the range between 0 and 1. This is useful for optimisation algorithms used in the core of machine learning algorithms like gradient descent. It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like k-Nearest Neighbors. You can rescale your data using scikit-learn using the $MinMaxScaler$ class. Details are available at

http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Verify your understanding:

(i) *Run the above code block and you should be able to see that all of the values are in the range between 0 and 1.*

Standardise Data. Standardisation is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression and linear discriminate analysis. You can standardise data using scikit-learn with the $StandardScaler$ class. More details are available at

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

```
# Standardise data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
```

```
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarise transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Verify your understanding:

(j) *Run the above code block and you will be able to see that the values for each attribute now have a mean value of 0 and a standard deviation of 1.*

(k) *Write a code block (using pandas.DataFrame.plot) that plots original data.*

(l) *Write a code block (using* `seaborn.distplot`*) that plots each attribute and see if any changes.* https://seaborn.pydata.org/generated/seaborn.distplot.html

Normalise Data. Normalising in scikit-learn refers to rescaling each observation (row) to have a length of 1 (called a unit norm or a vector with the length of 1 in linear algebra). This pre-processing method can be useful for sparse datasets (lots of zeros) with attributes of varying scales when using algorithms that weight input values such as neural networks and algorithms that use distance measures such as k-Nearest Neighbors. You can normalise data in Python with scikit-learn using the *Normalizer* class. Details are available at

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html

```
# Normalise data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(normalizedX[0:5,:])
```

Verify your understanding:

(m) *Run the above code block and you will be able to see that the rows are normalised to length 1.*

Binarise Data. You can transform your data using a binary threshold. All values above the threshold are marked 1 and all equal to or below are marked as 0. This is called binarising your data or thresholding your data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful. You can create new binary attributes in Python using scikit-learn with the *Binarizer* class. Details are available at

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Binarizer.html

```
# binarisation
from sklearn.preprocessing import Binarizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
```

```
X = array[:,0:8]
Y = array[:,8]
binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(binaryX[0:5,:])
```

Verify your understanding:

(n) *Run the above code block and you will be able to see that all values equal or less than 0 are marked 0 and all of those above 0 are marked 1.*

(o) [Challenge!] *Run a Decision Tree algorithm on both the raw and the normalised data and compare their results. The output should look like the below.*

```
Normalised Samples
 [[0.034 0.828 0.403 0.196 0.    0.188 0.004 0.28 ]
 [0.008 0.716 0.556 0.244 0.    0.224 0.003 0.261]
 [0.04  0.924 0.323 0.    0.    0.118 0.003 0.162]
 [0.007 0.588 0.436 0.152 0.622 0.186 0.001 0.139]
 [0.    0.596 0.174 0.152 0.731 0.188 0.01  0.144]]
Mean estimated accuracy
 0.6874572795625428
Mean estimated accuracy on normalised data
 0.6378332194121668
```

You may use the below codes.

```
# Decision tree classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
kfold = KFold(n_splits=10, random_state=7)
model = DecisionTreeClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print("Mean estimated accuracy \n",results.mean())
```

3. References

[1]. Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

[2]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.

[3]. Grus, J., 2019. Data science from scratch: first principles with python. O'Reilly Media.

[4]. Müller, A.C. and Guido, S., 2016. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".

[5]. Brownlee, J., 2014. Machine learning mastery.

[6]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.

[7]. SAS, 2018, Advanced Predictive Modelling using SAS