THE UNIVERSITY OF SYDNEY

Introduction to Python Course

Level 1B

Centre for Continuing Education

Version 1.1

April 2019

Contents

1 Installing Python

We will be working with the Anaconda version (distribution) of Python 3. Download the appropriate version for your operating system from https://www.anaconda.com/download/.

If you already have Anaconda Python installed on your device, you may want to create a new environment for this course:

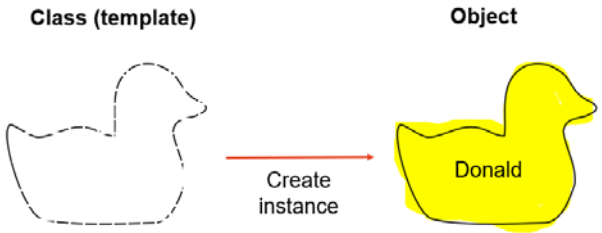
1. Open the Anaconda Navigator application and select **Environments**.
2. Select **Create**.
3. Choose a name, e.g., 'CCE_Course', choose Python 3, then click **Create**.
4. Clock on **Home** and install Jupyter Notebook.

2 Classes and objects

Python is an object-oriented programming language.

Almost everything in Python is an object, with attributes (variables) and methods (functions).

A class can be thought of as a template for creating objects. I.e., an object is an instance of a class.



Attributes	Methods
Sound = 'Quack!'	Swim
Colour	Gain weight
Weight	

Attributes	Methods
Sound = 'Quack!'	Swim
Colour = Yellow	Gain weight
Weight = 5kg	

- Create a 'duck' class:

```
class duck:

    # Set default attribute(s).
    sound = 'Quack!'

    # Define method(s).
    def swim(self, speed):
        print('Swims {0}.'.format(speed))
```

- Create a new instance of duck called 'donald'.

```
donald = duck()
```

- Get an object attribute

```
print(donald.sound)
```

- Execute a method.

```
donald.swim('fast')
```

The 'self' parameter is a reference to the class itself and is used to access variables that belong to the class.

It does not have to be named 'self' - you can call it whatever you like, but it must be the first parameter of any method in the class.

Classes have a special method called __init__(), which is always executed when a new object is created.

- Use the __init__() method to assign values to object properties.

```
class duck:

    # Default attribute.
    sound = 'Quack!'

    # Methods.
    def swim(self, speed):
        print('Swims {0}.'.format(speed))

    def __init__(self, colour, weight):
        self.colour = colour
        self.weight = weight

    def add_weight(self, kg):
        self.weight = self.weight + kg

# Create a new instance of duck called 'donald'.
donald = duck('yellow', 5)

print('Donald is a {0}, {1}kg duck who says
"{2}".'.format(donald.colour, donald.weight,
donald.sound))

donald.add_weight(2)

print('A {0}, {1}kg duck who says
"{2}".'.format(donald.colour, donald.weight,
donald.sound))
```

- Another special method is __str__(), which is called when the object is printed.

```
class duck:

    # Default attribute.
    sound = 'Quack!'
```

```
        # Methods.
        def swim(self, speed):
            print('Swims {0}.'.format(speed))

        def __init__(self, colour, weight):
            self.colour = colour
            self.weight = weight

        def add_weight(self, kg):
            self.weight = self.weight + kg

        def __str__(self):
            return 'A {0}, {1}kg duck who says
    "{2}".'.format(self.colour, self.weight, self.sound)

    # Create a new instance of duck called 'donald'.
    donald = duck('yellow', 5)
    print(donald)
```

- To change an object attribute:

```
donald.sound = 'Moo!'
print(donald)
```

- To delete an object:

```
del donald
```

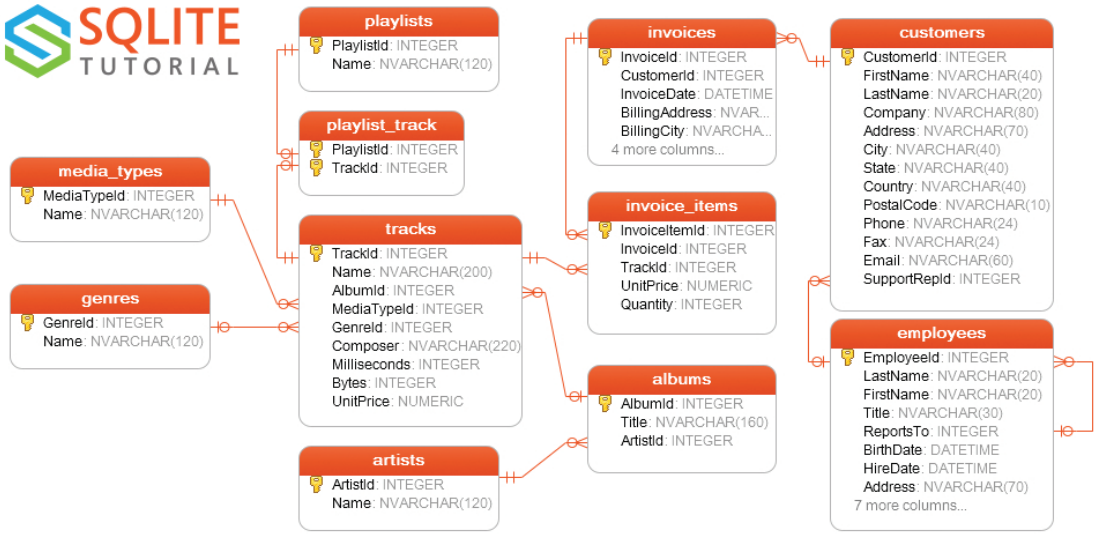
Exercise 2.1:

Create a Python class named 'rectangle' with attributes length and width, and a method for calculating the area.

3 Database access and queries

Until now, we have been working with datasets stored in Excel files. But Excel has a limit of 1,048,576 rows. Large datasets are often stored in relational databases, e.g., Oracle, MySQL, SQLite.

To see how a database can be queried from Python, we will use the 'chinook' sample database. This is an SQLite database for a fictional music download service. It is available from http://www.sqlitetutorial.net/sqlite-sample-database/.



1. Create a new Python 3 notebook and rename it 'database'.
2. Import the sqlite3 module:

```
import sqlite3
```

3. Connect to the chinook database and create a handle:

```
db = sqlite3.connect('chinook.db')
```

4. Create a cursor:

```
cursor = db.cursor()
```

5. Obtain information about individual tables:

```
query = 'SELECT * FROM albums'
cursor.execute(query)
cursor.description

query = 'SELECT * FROM artists'
cursor.execute(query)
cursor.description
```

6. Count rows in the 'albums' table:

```
query = 'SELECT COUNT(*) FROM albums'
cursor.execute(query)
count = cursor.fetchone()
print(count)
```

7. Point to the contents of the 'albums' table:

```
query = 'SELECT * FROM albums'
cursor.execute(query)
```

8. Retrieve the first row:

```
album1 = cursor.fetchone()
print(album1)
```

9. Retrieve all rows:

```
allAlbums = cursor.fetchall()

for album in allAlbums:
    print(album)
```

Exercise 3.1:

Retrieve all rows from the 'artists' table.

10. Join two tables and retrieve specified columns:

```
query = """
        SELECT AlbumId, Title, Name
        FROM albums JOIN artists
        ON albums.ArtistId = artists.ArtistId
        """
cursor.execute(query)
allRows = cursor.fetchall()

for row in allRows:
    print(row)
```

11. Select rows from join based on a given condition:

```
query = """
        SELECT AlbumId, Title, Name
        FROM albums JOIN artists
        ON albums.ArtistId = artists.ArtistId
        WHERE Name = 'Miles Davis'
        """
cursor.execute(query)
allRows = cursor.fetchall()

for row in allRows:
    print(row)
```

12. Format output:

```
colNames = ['AlbumId', 'Album', 'Artist']
print('{0:<10}{1:<40}{2}'.format(colNames[0],colNames
[1],colNames[2]))

for row in allRows:

print('{0:<10}{1:<40}{2}'.format(row[0],row[1],row[2]
))
```

13. Read a table into a pandas DataFrame:

```
import pandas as pd

query = 'SELECT * FROM albums'

albumsDF = pd.read_sql(query, db)

albumsDF.head(10)
```

14. Join three tables and read selected output into a pandas DataFrame:

```
query = """
        SELECT tracks.Name, albums.Title,
artists.Name
        FROM tracks
        JOIN albums
        ON tracks.AlbumId = albums.AlbumId
        JOIN artists
        ON albums.ArtistId = artists.ArtistId
        WHERE artists.Name = "Miles Davis"
        """

milesDavisTracksDF = pd.read_sql(query, db)

milesDavisTracksDF
```

Exercise 3.2:

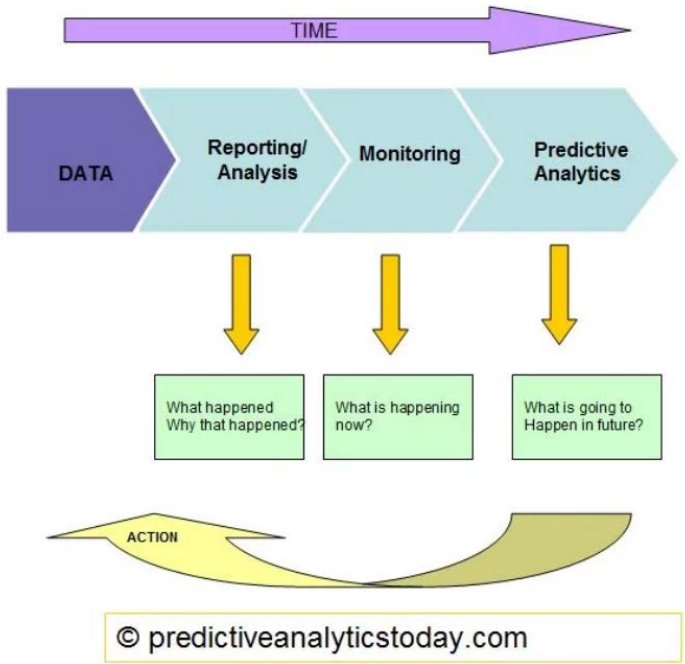
Find all the tracks in the chinook database that have the genre 'Latin'.

15. Disconnect from database:

```
db.close()
```

4 Predictive analytics

Predictive analytics encompasses a variety of statistical techniques from data mining, predictive modelling and machine learning, which analyse current and historical data to make predictions about future events.



Predictive analytics is an enabler of big data: businesses collect vast amounts of customer data and predictive analytics uses this historical data, combined with customer insight, to predict future events. Predictive analytics enable organisations to use big data (both stored and real-time) to move from a historical view to a forward-looking perspective of the customer.

For example, stores that use data from loyalty programs can analyse past buying behaviour to predict the coupons or promotions a customer is most likely to participate in or buy in the future. Predictive analytics could also be applied to customer website browsing behaviours to deliver a personalised website experience for the customer.

4.1 Labels and targets

A *label* is a variable we are trying to predict, e.g., house price, temperature or even the brand of car that someone will choose. Usually we use the letter y to denote the label. It is also known as the dependant variable.

Features are variables we believe might have some effect on the label, e.g., house size, number of bedrooms, distance to a school, etc. might affect house price. We use the letter x to denote feature variables. Features are also known as independent variables.

4.2 Training and inference

Once we have determined what the target variable and feature variables are, our next step is to build and test a model. There are two main stages to this process:

- Training is learning process. We give the model historical training data and the model corrects itself to model or learn the relationships between the features and label.
- Inference is the process of applying our model to new data. You use the trained model to make predictions.

4.3 Prediction of a continuous variable

A linear regression model is used to predict a label that is continuous, e.g., house price and temperature (but not brand of car).

The relationship between a continuous label y and features $x_1, x_2, \ldots, x_3$ can be written as an equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

where β_0 is a constant and β_1, β_2, etc. are parameters (coefficients), all of which need to be estimated during the training process.

Let's look at an example using house price data collected in Baton Rouge, Louisiana (US).

The following variables are included in the file 'BatonRouge.xls':

- Price: The price the house sold for
- SQFT: Size of the house in square feet
- Bedrooms: Number of bedrooms
- Baths: Number of bathrooms
- Age: Age of the house in years
- Occupancy: Number of occupants
- Pool: Does the house have a pool (yes/no)
- Style: Architectural style of the house
- Fireplace: Does the house have a fireplace (yes/no)
- Waterfront: Is the house on the waterfront (yes/no)
- DOM: Number of days the house spent on the market before it was sold

Given these data, we can do many regressions depending on which variable we select as the target or features. However, there are two guiding questions you can ask yourself:

- Which variable can you estimate from the others? (I.e., what is the dependency order of variables?)
- What is your goal?

In this example, our goal is to estimate the house price. House price is the most obvious dependant variable. All others have only weak dependency on each other. For example, we expect that a bigger house is more expensive. But a more expensive house is not always bigger (due to other factors, such as location).

Label: Price

Features: All other variables

1. Start by importing the dataset into a pandas DataFrame.

```
import pandas as pd

batonRouge = pd.read_excel('BatonRouge.xls')

batonRouge.head()
```

Next, we need to do some exploratory data analysis, to see if there are any relationships between price and the other variables.

2. One approach is to calculate the correlation between price and each of the other variables:

```
batonRouge.corr()
```

3. It looks like the area (SQFT) has the largest linear correlation with price. We can confirm this with a scatterplot:

```
%matplotlib inline
import matplotlib.pyplot as plt

y = batonRouge.Price
x = batonRouge.SQFT

fig = plt.figure()
plt.scatter(x, y, alpha=0.25)
plt.xlabel('Area (sq.ft.)')
plt.ylabel('Price ($)')
plt.show()
```

Note that relying on correlation scores is somewhat risky, because they reflect only the linear correlation between variables. If there is a strong non-linear relationship, it might be missed – but would be seen on a scatterplot.

4. We can train a model using statsmodels:

```
import statsmodels.formula.api as smf
```

5. Define the formula (using ~ in place of an equal sign):

```
formula = 'Price ~ SQFT'
```

6. Initialise model:

```
lin_reg = smf.ols(formula = formula, data =
batonRouge)
```

7. Fit the model and output results:

```
model_1 = lin_reg.fit()

model_1.summary()
```

The results include the following outputs that indicate how good our model is:

- **R-squared:** The R-squared value is a measure of the amount of variance that the model explains. A value of 1 means the model explains all variance. However, in most cases this would be considered overfitting.
- **P>[t]:** The p-value gives the statistical significance of the parameter value being non-zero. A value less than 0.05 means we can be 95% confident the parameter is non-zero; a value less than 0.01 means we can be 99% confident the parameter is non-zero; etc. This

does not necessarily mean the variable should be included in the model. You must also check the coefficient value.

- **coef:** Coefficient value: the magnitude of the coefficient/parameter value (β). The larger this value, the more it contributes to shifting the value of the dependant variable. A large coefficient value means the variable is practically significant.

8. Plot regression line:

```
import numpy as np

min_x = np.min(x)
max_x = np.max(x)

x_points = np.linspace(min_x, max_x, 100)

y_points = model_1.params[0] + model_1.params[1] *
x_points

fig = plt.figure()

plt.scatter(x, y, alpha = 0.1)

plt.plot(x_points, y_points, color = "red")

plt.xlabel("SQFT")
plt.ylabel("Price")

plt.show()
```

9. Now the model has been trained, we can use it to make predictions using new data. The easiest way to do this for a simple case is with a dictionary:

```
new_house = {'SQFT': 1500}

model_1.predict(new_house)
```

10. To improve the model, we can add more features:

```
formula = 'Price ~ SQFT + Bedrooms + Baths + Age +
Occupancy + Pool + Style + Fireplace + Waterfront +
DOM'

lin_reg = smf.ols(formula = formula, data =
batonRouge)

model_2 = lin_reg.fit()

model_2.summary()
```

11. Drop problematic features:

```
formula = 'Price ~ SQFT + Baths + Age + Waterfront'

lin_reg = smf.ols(formula = formula, data =
batonRouge)

model_3 = lin_reg.fit()

model_3.summary()
```

Exercise 4.1:

Predict the price of a house with the following features:

- 1500 sq.ft.
- 1 bathroom
- 20 years old
- Waterfront location

How and why does the predicted price differ from the previous prediction?

Exercise 4.2:

Using the DirectMarketing.csv dataset, predict how much a customer will spend, given a set of attributes.

1. Investigate which features you think would be helpful to explain the target variable 'AmountSpent'. It will be helpful to aggregate and visualise the data.

2. Build a simple linear regression model to predict 'AmountSpent'. Use only one feature that you think will best predict the 'AmountSpent'.

3. Test your model by predicting 'AmountSpent' for customers with differing values of the feature.

4. Build a multiple linear regression model with 'Salary', 'Children', 'Gender', 'Catalogs'.

5. Estimate 'AmountSpent' for two people with the following attributes:

- $50,000 salary, 0 children, 10 catalogs and female.

- $10,000 salary, 1 child, 0 catalogs and male.

Tip: Use the Pandas get_dummies() function to convert string/categorical variables to dummy (binary) variables.

4.4 Prediction of a categorical variable

Let's look at an example using historical data on bank loan defaults.

The following variables are included in the file 'loan_defaults.xlsx':

- default: Whether the customer defaulted on the loan (1) or not (0)
- student: whether the customer is a student or not (Yes/No)
- balance: size the of the loan
- income: customer's annual income

In this example, our goal is to predict whether a customer will default on a loan or not. But we cannot use linear regression because 'default' is not a continuous variable (what would a default value of -1, 0.4 or 2 mean?).

Instead, we can use logistic regression, where we predict the *probability* (between 0 and 1) of the customer defaulting.

The probability y given a single feature x_1 is estimated using the logistic function:

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1)}}$$

where β_0 is a constant and β_1 is a parameter (coefficient). These need to be estimated during the training process.

Label: default

Features: All other variables

1. Start by importing the dataset into a pandas DataFrame:

```
import pandas as pd

data = pd.read_csv('loan_defaults.csv')

data.head()
```

Next, we should investigate the data.

2. Check how many observations and variables we have:

```
data.shape
```

The label ('default') has two possible classes: 0 (didn't default) and 1 (did default). We suspect that very few customers defaulted on their loans – meaning there are very few observations for customers who did default. If one class dominates the dataset, this could bias our model.

3. Count the number of customers in each class:

```
data['default'].value_counts()
```

Regression works on numerical data, but the 'student' variable contains text data ('Yes'/'No').

4. Convert text data to numerical:

```
data = pd.get_dummies(data, drop_first=True)

data.head()
```

Next, we need to explore any relationships between 'default' and the other variables.

5. Group the data by default class and calculate the average feature values for each class:

```
data.groupby('default').mean()
```

6. We can also calculate the correlation between 'default' and each of the other variables:

```
data.corr()
```

7. Visualise the distributions of balance for the two classes:

```
%matplotlib inline
import matplotlib.pyplot as plt

defaultN = data.query('default == 0')
defaultY = data.query('default == 1')

defaultN.head()
defaultY.head()

plt.hist(defaultN.balance, alpha = 0.5)
plt.hist(defaultY.balance, alpha = 0.5)

plt.ylabel('Frequency')
plt.xlabel('Loan balance ($)')
plt.legend(['No', 'Yes'])
plt.title('Histogram of default by loan balance')

plt.show()
```

8. Set density = True to ignore any scaling effects:

```
plt.hist(defaultN.balance, alpha = 0.5, density =
True)
plt.hist(defaultY.balance, alpha = 0.5, density =
True)

plt.xlabel('Loan balance ($)')
plt.legend(['No', 'Yes'])
plt.title('Histogram of default vs loan balance')

plt.show()
```

We can train a logistic regression model using sklearn:

9. To begin with, it's good practice to reserve a subset of the dataset for final model testing:

```
from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest =
train_test_split(data.balance, data.default,
random_state = 0)

print(Xtrain)
```

10. Xtrain and Xtest are pandas Series, but sklearn needs them to be in an array with a single column:

```
Xtrain = Xtrain.values.reshape(-1, 1)
Xtest = Xtest.values.reshape(-1, 1)
```

The first argument of the reshape() function is -1. This means the first dimension (number of rows) of the array should be inferred from the original Series. The second argument of the reshape() function is 1. This means the second dimension (number of columns) of the array should be 1.

11. Initialise model. Because the dataset is not balanced (very few defaulters), set the 'class_weight' parameter to 'balanced':

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(class_weight =
"balanced")
```

12. Fit the model:

```
log_reg.fit(Xtrain, ytrain)
```

13. Get the constant and parameter values (β_0 and β_1):

```
log_reg.intercept_
log_reg.coef_
```

14. Test the model with different values of our feature ('balance'):

```
import numpy as np
balances = np.array([1000])

log_reg.predict_proba(balances.reshape(-1, 1))

log_reg.predict(balances.reshape(-1, 1))
```

15. Evaluate the model:

```
log_reg.score(Xtrain, ytrain)

log_reg.score(Xtest, ytest)

from sklearn.metrics import confusion_matrix

ytest_preds = log_reg.predict(Xtest)
```

```
tn, fp, fn, tp = confusion_matrix(ytest,
ytest_preds).ravel()
print(tn, fp, fn, tp)
```

16. Create a heatmap of the confusion matrix:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

confusion = confusion_matrix(ytest, ytest_preds)

plt.figure(figsize = (10,7))

sns.heatmap(confusion, annot = True, fmt = 'd' , cmap
= 'Blues',
            xticklabels = ['No', 'Yes'],
            yticklabels = ['No', 'Yes'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
```

17. Obtain some summary statistics

```
from sklearn.metrics import classification_report

print(classification_report(ytest, ytest_preds))
```

Exercise 4.3:

See if adding more features improves the model.

Tip: Slice the last three columns of the DataFrame to grab all features:

```
Xtrain, Xtest, ytrain, ytest = train_test_split(
data.iloc[:, 1:4], data.default, random_state = 0)
```

This stores Xtrain and Xtest in two-dimensional data structures (pandas DataFrames), so they do not need to be reshaped this time.

5 Text analytics

We will look at some Tweets by Barack Obama and Donald Trump, visualise them using Word Cloud, then build a model to predict the author of other Tweets made by them.

1. To install WordCloud using Anaconda Navigator:
   - Go to Environments.
   - Click on **Channels**.
   - Click on **Add…**.
   - Type 'https://anaconda.org/conda-forge' and press Enter.
   - Click on Update channels.

- Search packages for 'wordcloud' and install it.

2. Create a new Python 3 notebook and rename it.
3. Load modules:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
from wordcloud import WordCloud, STOPWORDS
```

4. Get Obama's Tweets.

```
obama = pd.read_csv('BarackObamaTweets.csv')
obama.head()

obamaTweets = obama.text
```

5. Use Word Cloud and pyplot to visualise Tweets.

```
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(obamaTweets))

fig = plt.figure(
    figsize = (40, 30),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Exercise 5.1:

Create a Word Cloud of Trump's Tweets (from the file 'DonaldTrumpTweets.csv').

Make the following adjustments:

1. Change the background colour to white.

2. Set the minimum font size to 10.

3. Set the maximum number of words to 50.

Documentation for Word Cloud:

https://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html.

For a regression analysis, the data need to be in one DataFrame.

6. Add a column 'Trump' to both DataFrames to indicate whether a Tweet was made my Trump (1) or not (0).

```
trump['Trump'] = 1
obama['Trump'] = 0
```

7. Concatenate data into one DataFrame:

```
data = pd.concat([trump[['text', 'Trump']],
obama[['text', 'Trump']]], axis = 0)

data.head(10)

corpus = data['text']
```

8. Split the corpus into words (i.e., tokenise) and check for any words that may cause problems.

```
from sklearn.feature_extraction.text import
CountVectorizer

vect = CountVectorizer()

vect.fit_transform(corpus)

print(vect.get_feature_names())

len(vect.get_feature_names())
```

There are many strange words, e.g., '0rtsmxdnfc'.

This is because the tweets include many website and picture links.

We can find these problematic words using regular expressions, then delete them.

First, let's find the website links. These start with 'http'. A suitable regular expression is 'http\S+', which means find the string 'http' followed by any sequence of non-whitespace characters.

Regular expressions can be tested using https://www.regexpal.com.

9. Test the regular expression 'http\S+' with the Tweet:

'Tell your friends you fist bumped President Obama enter now for your chance. http://ofa.bo/fAEqÂ pic.twitter.com/zJEelmele6'.

10. To remove all website links from the corpus, use the sub() function to replace each matched string with an empty string (''):

```
import re

corpusCleaned = corpus.apply(
    lambda x: re.sub(
        r'http\S+',
        '',
        x,
        flags=re.IGNORECASE))
```

11. See how many words have been removed.

```
vect = CountVectorizer()
vect.fit_transform(corpusCleaned)

print(vect.get_feature_names())

len(vect.get_feature_names())
```

Exercise 5.2:

Remove the URLs and picture links (strings beginning with 'pic.twitter.com') from the corpus.

Tip: To combine regular expression, use the OR operator "|" (pipe symbol), e.g.,

'http\S+|<add another regular expression here>'.

How many words remain?

12. Vectorise the cleaned corpus, removing stop words:

```
vect = CountVectorizer(stop_words = 'english')

X = vect.fit_transform(corpusCleaned)
```

13. Split off some test data to test our model later:

```
from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(X,
data['Trump'], random_state = 0)
```

14. Train a logistic regression model using the training data:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
```

```
log_reg.fit(Xtrain, ytrain)
```

15. Find out which words have the most predictive power:

```
featureNames = vect.get_feature_names()

sortedCoefIndex = log_reg.coef_[0].argsort()

for i in sortedCoefIndex:
    print(featureNames[i])
```

16. A concise way to generate a list from another list is to use list comprehension:

```
[featureNames[i] for i in sortedCoefIndex[0:30]]

[featureNames[i] for i in sortedCoefIndex[-30:]]
```

17. Test the model on some Tweets:

```
testTweets = vect.transform(['mexican border wall',
                             'crime',
                             'peace',
                             'health'])

print(testTweets)

log_reg.predict_proba(testTweets)

log_reg.predict(testTweets)
```

Exercise 5.3:

Try this with your own Tweets.

18. Evaluate the model:

```
log_reg.score(Xtrain, ytrain)

log_reg.score(Xtest, ytest)

from sklearn.metrics import confusion_matrix
ytest_preds = log_reg.predict(Xtest)
confusion = confusion_matrix(ytest, ytest_preds)
print(confusion)
```

19. Create a heatmap of the confusion matrix:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize = (10,7))

sns.heatmap(confusion, annot=True, fmt = 'd' , cmap =
'Blues',
            xticklabels = ['Obama', 'Trump'],
            yticklabels = ['Obama', 'Trump'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
```

20. Obtain some summary statistics:

```
from sklearn.metrics import classification_report

print(classification_report(ytest, ytest_preds))
```

6 Data visualisation with Tableau

6.1 Set up the TabPy API on your own device

1. Open Anaconda Prompt (Windows) or Terminal (Mac).
2. Create a new environment named 'Tableau-Python-Server'.

```
conda create -n Tableau-Python-Server python=3.x
```

3. Activate environment.

```
conda activate Tableau-Python-Server
```

4. Install the TabPy API.

```
conda install tabpy-client
conda install tabpy-server
```

5. Search your computer for 'tabpy_server' and change directory to its location, e.g.:

```
cd Anaconda3\envs\Tableau-Python-Server\Lib\site-
packages\tabpy_server
```

6. Type 'startup' and press enter.

6.2 Connect to TabPy API on your own device

1. Open Tableau Desktop.
2. Select Help > Settings and Performance > Manage External Service Connection.
3. Under Select an External Service choose TabPy/External API.
4. For the Server, select localhost.
5. For the port, enter '9004'.
6. Test the connection then click on OK.

6.3 Connect to TabPy API on a server

1. Open Tableau Desktop.
2. Select Help > Settings and Performance > Manage External Service Connection.
3. Under Select an External Service choose TabPy/External API.
4. For the Server, enter 'http://10.83.65.199'.
5. For the port, enter '9004'.
6. Test the connection then click on OK.

6.4 Using the Python API

1. Connect to the Microsoft Excel file 'BatonRouge.xls'.
2. Go to the Worksheet.
3. Create a scatterplot of 'Price' vs 'Sqft'.
4. Convert 'Waterfront' and 'Pool' to dimensions.
5. Drag 'Waterfront' to **Columns**.

Say we want to show the correlation between 'Price' and 'Sqft' for each pane. We can calculate these in Python.

To let Tableau know a calculation needs to go to Python, it must be passed through one of the four Tableau functions, depending on the data type that we want to be returned:

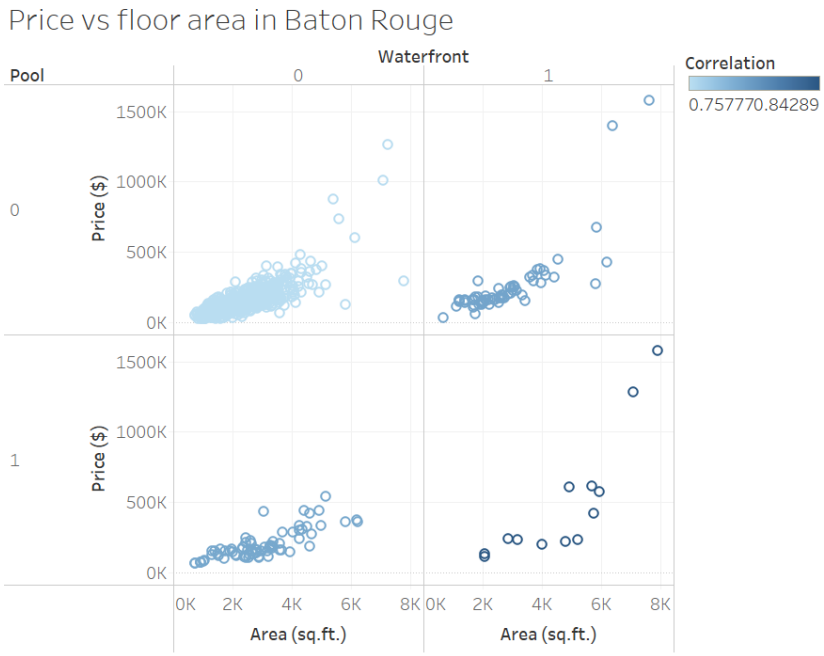
- SCRIPT_BOOL() – returns a Boolean
- SCRIPT_INT() – returns an integer
- SCRIPT_REAL()– returns a real number
- SCRIPT_STR() – returns a string

Python functions are computed as table calculations in Tableau, so all the variables being passed to them must be aggregated, e.g., SUM([Price]), MIN([Price]), AVG([Sqft]), etc.

6. Create a new calculated field PyPriceAreaCorr

```
SCRIPT_REAL('
  import numpy as np
  return np.corrcoef(_arg1,_arg2)[0,1]
',
SUM([Price]), SUM([Sqft]))
```

7. Drag 'PySalesProfitCorrelation' to **Color**, click the dropdown arrow and select **Compute Using > Cell**.



Area (sq.ft.) vs. Price broken down by Waterfront vs. Pool. Color shows PyPriceAreaCorr.

For more information about using Python with Tableau, see **https://www.tableau.com/about/blog/2016/11/leverage-power-python-tableau-tabpy-62077.**