**Importing Packages**

import seaborn as sb

import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
%matplotlib inline

import random

import statsmodels.api as sm

from pandas import Series, DataFrame

from matplotlib import rcParams

**Pandas Data Frames** (f is a dataframe)

**Creating Data Frames**

np.zeros(5) <- create a vector of 5 zeros(any number can be used instead of zeros ie np.ones)

np.zeros((5,5)) <- create a 5x5 matrix of zeros

np.array([[list1],[list2],etc.]) <- list 1 = row 1; list 2 = row 2

f[:,1:3] <- select the first two columns in an array

f[1,:] or f[1] <- select the first row of an array

Series(np.arrange(number of terms), index = [‘index names’])

f = DataFrame({‘column a ‘:[list of column a data]

‘column b’:[list of column b data]})

**Uploading Data**

f = pd.read\_excel(r'C:\Users\harrison\Documents\SoccerCareer\Newcastle.2018.2019.Season.xlsx')

f = pd.read\_csv( (r'C:\Users\harrison\Downloads\filename.csv'))

f = pd.DataFrame(data = datavariable, columns = columnnamesvariable)

**First Glances**

f.dtypes <- shows what type of data is in each column

f.info <- look for non-null values

a.head(n)<- show n rows of the data array

a.tail(n) <- shows n rows from the end of the array

f.transpose() <- switches rows and columns and vice versa

f.shape <- shows (rows, columns)

f.size <- total number of elements

f[‘column name’].unique() <- shows unique values for that column

f[‘column name’].describe() <- shows basic statistical summaries

f.reshape(10,10) <- reshapes data that isn’t in the format you want 10x10 matrix in this case

**Missing Data**

amissing = a[‘column name’].isna() <- displans Nan values

series\_obj.isnull() <- returns Booleans for null values

f.loc[amissing, :] <- shows row of missing value

f[‘column name’].isna().sum() <- returns total number of Nan values

f.isna().sum() <- shows the whole dataframe nans by index

**Removing Filling and Duplicating Missing Data in Rows**

f.dropna() <- drops all rows with missing values

f[‘column name’].fillna(0) <- fills Nan in with 0

f[‘column name’].fillna({index: newnumber, rowindex: new number}) change nulls for row or column

f[‘column name’].fillna(method=’bfill’) <- fills in Nan with next value in column

f[‘column name’].fillna(method=’ffill’) <- fills in Nan with previous value in column

f[‘column name’].interpolate(method=’linear’) <- fills in Nan with average of previous and next

f[‘row name’] = 8 <- changes the values into an 8

f.duplicated() <- returns duplicate rows

f.drop\_duplicates() <- drops duplicate rows

f.drop\_duplicates([‘column name’]) <- drops duplicates from a specific column

**Concatenation, Transformation, and Grouping**

f[‘column name’].name = ‘co’

f.append(f, ignore\_index=True) <- adds a dataframe to itself with an index that continues where f left off

f.sort\_values(by=(5), ascending=False) <- a sorted by column 5 in descending order

f.groupby(a[‘column name’])

**Slicing Data**

f[[‘column name’,’column name’]] <- list of subselected colum(s)

f[‘column name’] <- looks for a key in the data frame, returns a series

f[‘column name’][x:x+3] <- slicing a series

f.loc[:, ‘column name’] <- sliced series

f.loc[:, [‘column name’]] <- sliced non-series

f.iloc[rowindex:rowindex] <- slice by row

ff.loc[ff['Goals For']>=2,['Place']] <- example of advanced slice

**Filtering Data**

f.groupby(by=[‘column name’]) <- groups column by the data in the column

f[‘column name’].value\_counts() <- how many of each category there are

*ffiltered = f[‘column name’] == ‘filter category’*

*f[ffiltered] <- Use with above to get a dataframe*

a.loc[ffiltered, :] <- possibly same as above?

f = ffiltered <- reset new filtered data to original name

**Adding and Renaming Columns**

f[‘new column name’] = variable <- create a new column

f = f.rename(columns={‘column name’: ‘new column name’,

‘column name’: ‘new column name’}) <- renames some columns

f.columns = [‘column 1 new name’, ‘column 2 new name’, etc.]

**Deleting Columns**

f.drop(1) <- can be used to drop row or columns by index

del f[‘column name]

**Converting Dataframe**

f.to\_numpy() <- converting to a numpy array

f.values <- converting to a numpy array

f.to\_dict() <- converting to a dictionary

**Exporting Dataframe**

a.to\_csv(path\_or\_buf=’folderpath/filename.csv’, index=False) - ???

a.to\_excel(excel\_writer=’folderpath/filename.xlsx, index=False) - ???

**Chart Types** (matplotlib)

**Bar Chart**

plt.bar(‘name of the bar’, measure variable)

f[column name].plot(kind=’bar’) <- use barh for a horizontal bar chart

**Heatmaps**

sb.heatmap(f, cmap = ‘Blues’);

**Histograms**

a[‘column name’].hist()

a[‘column name’].hist(bins = 30, edgecolor = ‘black’) <- sets bin size

plt.hist(a, normed=True) <- y axis become proportions not totals

plt.hist(a, bins=np.linspace(1,10,10) <- bins go one to ten with **9** total bins

plt.hist(a, cumulative = True) <- cumulative histogram

plt.hist(a, histtype = “step’) <- creates a histogram with no fill

sb.distplot(dataname)

**Line Chart**

plt.plot(xvariable, yvariable)

plt.plot(x, y, marker = ‘.’, markersize = 10, linewidth = 10) <- changes the look of the line

plt.plot(x, y, c=’b’) <- changes the color of a line to blue

plt.plot(x, y, c = (0, 0, 1)) <- color tuple

plt.grid()

plt.grid(c = ‘g’, alpha = .9, linestyle = ‘-‘) <- added flair to grid

Keyword Arguments = linewidth, markersize, “gs-“ <- green, squares, solid line

**Scatterplot**

plt.scatter(x, y, alpha = ‘’) <- scatterplot

m, b = np.polyfit(x, y, 1) <- line of best fit part 1

plt.plot(x, m\*x + b) <- line of best fit part 2

sb.regplotI(x=’x variable’, y=’yvariable’, data=DataFrame, scatter=True)

sb.pairplot(DataFrame) <- scatter plot matrix

**Pie Chart**

plt.pie(x, labels=list of labels)

**Boxplots**

plt.boxplot([‘numpyarray1’, ‘numpyarray2’], labels = [‘arraylabel1’, ‘arraylabel2])

sb.boxplot(x=’column name’, y=’column name’, data = f, palette = ‘hls’) <- Seaborn method

**Tukey Boxplot**

f.boxplot(return\_type=’dict’) <- finds which variable has outliers

variable\_in\_question = X[:,1]

outliers = (variable\_in\_question > 4)

f[outlier] <- prints out a table of outliers

*pd.options.display.float\_format = ‘{:.1f}’.format <- helps one find the interquartile range*

*print(a.describe())*

**Subplots**

*plt.subplot() <- make a new subplot for each plot you want*

*plt.hist(x)*

plt.subplot(1, 5, 1) <- (rows, columns, index(1 for first image))

Matplot.lib gallery <- website for more info

**Object orientated plotting**

fig = plt.figure()

ax = fig.add\_axes([.1,.1,1,1])

ax.set\_xlim([1,9])

same for y but with a y instead (ylim)

ax.set\_xticks([list])

ax.grid()

**Object Oreintated Subplot**

Fig, (ax1, ax2) = plt.subplots(1,2)

Ax1.plot(x)

Ax2.plot(x, y)

**Plot Formatting**

Marker styles

Line styles

*Wide = [.5,.5,.5,1,.5] <- must be same length as there are variables, different widths*

*Color = [‘salmon’]*

*Plt.bar(x, y, width = wide, color = color, align=’center’)*

Color\_theme = [‘color1’, ‘color2’ etc.] <- can also use RGB code: #A9A9A9

Plt.plot(x, y, ds=’steps’, lw=5, ls=’- -') <- ls is linestyle, lw is line width, and ds is design?

mew = marker width

marker = change marker style.

Variable = a.columnname

Variable.idxmax <- returns the index of the maximum value

**Statistics**

.count(), .sum(), .max(), .min(), .mean(), .median(), .std(), .var(), .value\_counts() <- the count of unique values

**Linear Regression Model**

x = f[[‘x1’, ‘x2’, etc.]]

y = f['y variable']

x = sm.add\_constant(x) <- creates an intercept for the output

model = sm.OLS(y, x).fit() <- creates model

predictions = model.predict(x) <- creates predictions

model.summary() <- shows the regression output

**Summarizing Categorical Data**

f.index = f.variablename <- can index by a specific variable

f[‘group’] = pd.Series(f.variable, dtype=’category) <- create a categorical data column

**Cross Tabulation Table**

pd.crosstab(f[‘column’], f[‘column’])

**Correlation**

From scipy.stats.stats import pearsonr

Plt.style.use(‘seaborn style’)

Sb.pairplot(f)

Pearsonr\_coefficient, p\_value = pearsonr(variable1, variable2)

f.corr <- shows a correlation matrix

sb.heatmap(corr, xticklabels = corr.columns.values, yticklabels = corr.colukns.values)

**Correlation for Non-Linear Variables**

**Spearman Rank correlation method – works on ordinal variables**

**Assumes variables are ordinal**

**Also that data is non normally distributed**

**They are related non-linearly**

Spearmanr\_coefficient, p\_value = spearmanr(variable1, variable2)

Print(‘string %0.3f’ % (spearman\_coefficient))

**Chi Square Test**

table = pd.crosstab(variable1, variable2)

from scipy.stats import chi2\_contingency

chi2, p, dof, expected = chi2\_contingency(table.values)

**Transform dataset distributions**

***Import sklearn***

***From sklearn import preprocessing***

***From sklearn.preprocessing import scale***

f[‘column’].values.reshape(-1,1) <- creates one colum matrix

**Scaling**

Scaled = preprocessing.MinMaxScalar() <- can add feature\_range = (0,10) in the ()

Scaled\_variable = scaled.fit\_transform(single column matrix)

Plt.plot(scaled\_variable)

*Scale remains the same, but values are between 0 and 1*

Variable = scale(mpg, axis=0, with\_mean = False, with std=False) <- returns a scaled variable to normal

Variable = scale(mpg) <- creates a standardized plot

Plt.plot(variable)

**Outlier Analysis**

***From pylab import rcParams***

X = df.iloc[:,0:4].values

Y = df.iloc[:,4].values

**Multivariate Outlier Detection – finds outliers within variable combinations**

From pylab import rcParams

Data = f.iloc[:,0:4] <- selects columns 0 through 4

target = f.iloc[:,4]

sb.boxplot(x = ‘species’, y = ‘sepal length’, data=df, palette=’hls’)

sb.pairplot(df, hue=’species’, palette=’hls’)

**Random Module**

**Random**

Import random

random.choice([“H”,”T”]) <- randomizes the choices in the list can also use ([0,1])

np.random.random(10) <- create 10 random numbers between 0 and 1

np.random.normal(size =100) <- 0 is the mean 1 is the variance, generating from normal dist.

np.random.seed(n) <- generates the same random number for the same seed

**Numpy Random Module**

np.random.random(5,6) <- array of 5x6 uniform distribution 0 to 1

np.random.normal(0,1,(2,2)) <- 0 is mean, 1 is standard deviation, 2x2 array

np.random.randint(1,10) <- random integer between 1 and 9

np.sum(array, axis=0) <- sums rows of the array. Axis=1 sums columns

**Timing**

*Import* time

Start\_time = time.clock() <- gives the current time

Start\_time-End\_time = how long a program took to run

DataFrame(np.random.rand(36).reshape((6,6)), index=[‘row names’], columns=[‘column names’] <- creates a 6 by b dataframe of random numbers

**Matplotlib**

**Styles**

plt.style.available <- lists all style options

plt.style.use(‘stylename’) <- use a new style

**Creating Variables**

Variablea = f.loc[:, ‘column name’].values <- set a variable as a numpy array

a.loc[f[‘column name’] == ‘category’, ‘y value’].values <- same as above, more advanced

*nums = [i for i in range(24,47) if i%2!=0]*

*newarray = f.iloc[:,nums]*

index = [list] f[index] <- a way to index multiple rows at once (MAKE edits this way too)

f > 6 <- returns booleans

f[f>6] <- returns a list of values that satisfy the condition

win = f > 6 f[win] <- another way to get non-booleans returned

**Plot Titles, Labels, Limits, and Legends**

plt.xlim(left = 1, right = 100)

plt.ylim(bottom = 0, top = 100)

plt.xlabel(‘Label Name’)

plt.ylabel(‘Label Name’, fontsize = 20)

plt.title(‘Title Name’) or plt.title(‘’) removes title

plt.xticks(fontsize = 20)

plt.xticks(rotation = 90) <- rotates X axis labels

plt.legend() <- charts need to have labels for this plt.plot(xvariable, yvariable, label=”name”)

plt.legend(loc = “center right”) <- can also do loc = (1.1,0) for more specificity

plt.axis([xmin, xmax, ymin, ymax) <- change axes

np.linspace(0, 100, 10) <- creates an array from 0 to 100 with ten linearly spaced elements

.annotate(xy, xytext, arrowprop) <- where to annotate, what to say, arrow to the point

**Logarithmic Axis**

plt.semilogx() <- x on log scale and y on original scale

plt.semiology() <- y on log scale and x on original scale

plt.loglog() <- both on log scale

np.log10(250) <- take log based 10 of 250

**Save plots to Files**

plt.savefig(‘filepath/filename.png’, dpi = 300)

plt.tight\_layout() <- makes sure none of the image is cut off

**Plotly**

! pip install Plotly

! pip install cufflinks

Import pandas as pd

Import numpy as np

Import cufflinks as cf

Import plotly.plotly as py

Import plotly.tools as tls

Import plotly.graph\_objs as go

tls.set\_credentials\_file(username=’’, api\_key=’find this number in plotly’)

*py.iplot(data, filename=’file\_name’) <- have no spaces in the file name* <- line chart

*layout = dict(title=’Title of chart’, xaxis = dict(title=’x-axis), yaxis = dict(title=’y-axis’))*

data = [go.Bar(x=[1,2,3,4,5,6,7,8,9,10], y=[1,2,3,4,5,6,7,8,9,10])] <- bar chart

layout = (same as above)

*fig = {‘data’:[{‘labels’: [‘bicycle’, ‘motorcycle’, ‘car’, ‘van’],*

*‘values’: [1,2,3,4], ‘type’: ‘pie’}], ‘layout’: {‘title’: ‘simple Pie Chart’}}*

*Py.iplot(fig)* <- pie chart

variable.iplot(kind=’histogram’, filename=’’) <- create a histogram

variable.iplot(kind=’box’, filename=’filename’) <- create a boxplot

**Web Scraping**

conda install -c anaconda beautifulsoup4 <- do this in anaconda, not jupityr notebook

import sys

print(sys.version)

from bs4 import BeautifulSoup

html <- create an html document(the code was not discussed)

our\_soup\_object = BeautifulSoup(html, ‘html.parser’)

print(out\_soup\_object.prettify()[0:300]) <- prints the first 300 characters of html code

**Tag Objects**

soup\_object = BeautifulSoup(‘h1 attribute\_1 = “Heading Level 1””>Future Trends for IOT in 2018</h1>’, “lxml”)

tag = soup\_object.h1

print(tag)

tag.name = h1 <- can be used to change the name of a tag

**Tag Attributes**

soup\_object <- see above for this

tag = soup\_object.h1

tag[‘attribute\_1’]

tag.attrs <- shows a dictionary of all attributes in a tag

tag[‘attribute\_2’] = “Heading Level 1\*”

del tag[‘attribute\_2’] <- deletes an attribute from the tag

**Navigating Parse Tree**

our\_soup\_object.head <- returns the head tag

our\_soup\_object.title <- returns the title tag

our\_soup\_object.body.b <- returns bold text in the body of html code

our\_soup\_object.li <- retrieve list objects

our\_soup\_object.a <- returns links

**Navigatable String Object**

Soup\_object = BeautifulSoup(‘<h1 attribute\_1 = “Heading Level 1””>Future Trends in IoT in 2018</h1>’, ‘lxml’)

Tag = soup\_object.h1

Type(tag) <- verifies that you created a tag object or not

Tag.name <- shows you the name of the tag

Tag.string <- prints the string from within the tag object

TagString = tag.string

TagString.replace\_with(‘NaN’) <- replaces the text in a string

*For string in our\_soup\_object.stripped\_strings:*

*Print(repr(string))* <- this prints an html document without tags, just the text

First\_link = our\_soup\_object.a <- finds the first link in the document

First\_link.parent <- shows the parent tag of the link

First\_link.string <- shows the string the link is embedded into

First\_link.string.parent <- shows the parent of the link’s string

**Data Parsing**

from bs4 import BeautifulSoup

import urllib

import urllib.request

import re

*with urllib.request.urlopen(‘web address’) as response:* <- put in a web URL

*html= response.read*

soup = BeautifulSoup(html, ‘lxml’)

soup.pretify()[:100] <- first 100 characters prettified

text\_only = soup.get\_text <- prints only text from html code

soup.find\_all(“li”) <- shows all the li tags in a html document

soup.find\_all(id = ‘link 7’) <- filters with keyword arguments

soup.find\_all(‘ol’) <- filters with string arguments

soup.find\_all([‘ol’, ‘b’])<- filtering with list objects

*t = re.compile(‘t’)*

*for tag in soup.find\_all(t):*

*print tag.name* <- filtering with regular expressions

*for tag in soup.find\_all(True):*

*print tag.name()* <- filtering with Boolean values

*for link in soup.find\_all(‘a’):*

*print(link.get(‘href’))* <- prints all links in a document

soup.find\_all(string=re.compile(“data”)) <- finds phrases with the word Data in it

**Scraping from Web**

*From bs4 import BeautifulSoup*

*Import urllib.request*

*From IPython.display import HTML*

*Import re*

R = urllib.request.urlopen(“https:…”).read()

Soup = BeautifulSoup(R, ‘lxml’)

*For link in soup.findAll(‘a’, attrs={‘href’: re.compile(‘^http’)})*

*Print(link)* <- finds all links with an href and an http

File = open(‘parsed\_data.txt’, ‘w’) <- open a text file that can be written on

*For link in soup.findAll(‘a’, attrs={‘href’: re.compile(‘^http’)})*

*Soup\_link = str(link)*

*Print(soup\_link)*

*File.write(soup\_link)*

*File.flush()*

*File.close()*

*%pwd <- present working directory (where any files you create can be found)*

**Natural Language Processing**

*conda install -c anaconda nltk <- anaconda environment*

*import nltk*

text = sentence(any sentence as a string works)

nltk.download(punkt) <- downloads the tokenizer

*from nltk.tokenize import sent\_tokenize*

*sent\_tk = sent\_tokenize(text) <- creates a list with sentences as variables*

*from nltk.tokenize import word\_tokenize*

*word\_tk = word\_tokenize(text) <- creates a list with words as variables*

**Removing stop words**

nltk.download(‘stopwords’)

from nltk.corpus import stopwords

sw = set(stopwords.words(‘English’) <- creates stop words

filtered\_words = [w for w in word\_tk if not w in sw] <- prints words after removing stopwords

**Stemming**

from nltk.stem import PorterStemmer

from nltk.tokenize import sent\_tokenize, word\_tokenize

port\_stem = PorterStemmer()

stemmed words = []

*for w in filtered\_words:* <- returns the stems of the filtered words from above

*stemmed\_words.append(port\_stem, stem(w))*

**Lemmatizing**

nltk.download(‘wordnet’)

from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer

stem = PorterStemmer()

lemm\_words = []

for i in range(len(filtered\_words)):

lemm\_words.append(lem.lemmatize(filtered\_words[i]))

**Part of Speech tagging**

nltk.download(‘averaged\_perceptron\_tagger’)

from nltk import pos\_tag

pos\_tagged\_words = pos\_tag(word\_tk) <- creates a variable that has each word and its part of speech in the form of a list

**Frequency Distribution Plots for Words**

from nltk.probability import FreqDist

fd = FreqDist(word\_tk)

import matplotlib.pyplot

fd.plot(30, cumulative=False)

plt.show()

**Useful Functions**

[i for i in range(24,47) if i%2!=0]

help(a.functionname())

dir(variable) <- shows the directory for all available functions for that data type

15 // 7 = 2 <- double slash is integer division (16 // 15 = 0)

\_ <- same as ANS on a calculator list.append(n) <- add n to end of the list

list.reverse() <- reverse order of the list

sorted(list) <- creates a new list that is sorted

list.sort() <- sorts alphabetically or numerically

(x,y) = tuple <- unpack a tuple of length 2

range(1,10,2) <- starts at 1 goes up until 10 and lists every 2nd item

set([list]) <- creates a set, a list-like object that does not contain repeats

thing.copy() thing.deepcopy() <- creates copy of an object

open(filename, “a” “w”) <- allows a file to be read or written on; close(filename)<- closes a file

Choose “a” to append to file or “w” to write over file but not both

rstrip() <- returns a copy of a string with trailing characters removed, default is spaces

class ClassName(): <- this and the next line are an example of a class

self.remove(min(self))

%pwd <- see what file you are working in currently

for index, element in enumerate(my\_list):

print(index, element)

f = zip(t, n) <- creates a list/tuple combining the two groups, can be used for lists

t, n = zip(\*f) <- unzips

np.set\_printoptions(precision = 2)

**Python Stuff**

Boolean operations – “or” “and” “not”

Comparisons – “>” “<” “>=” “<=” “==” “!=” “is” “is not”