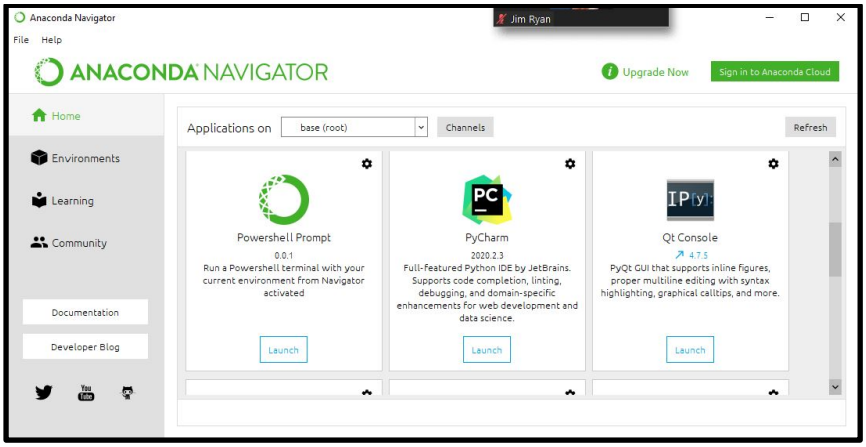
Mining & Data Storing

Data mining continued…

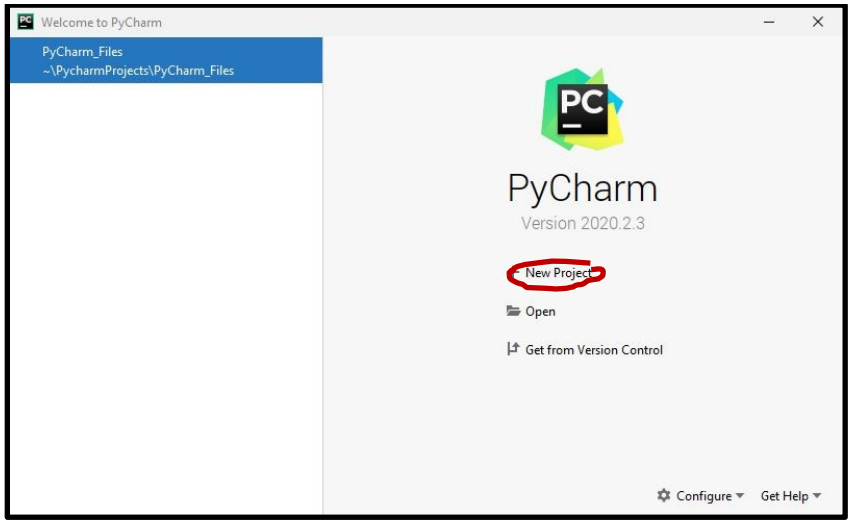
Data mining is a combination of several different processes, tools, or methodologies used to extract relevant data from a larger set of raw data. As we saw in Tutorial Eight-A, data mining implies analyzing data patterns in large batches using one or more software applications. With respect to this tutorial, we will continue our focus on Python data mining processes for text mining and network mining, as well as introducing array and matrix math along with data input and output (reading, writing, and appending) via SQL scripts, NumPy, and Pandas. Tutorial Ten will explore more text mining via Natural Language Processing (NLP).

Create a MiningStoring Project in PyCharm

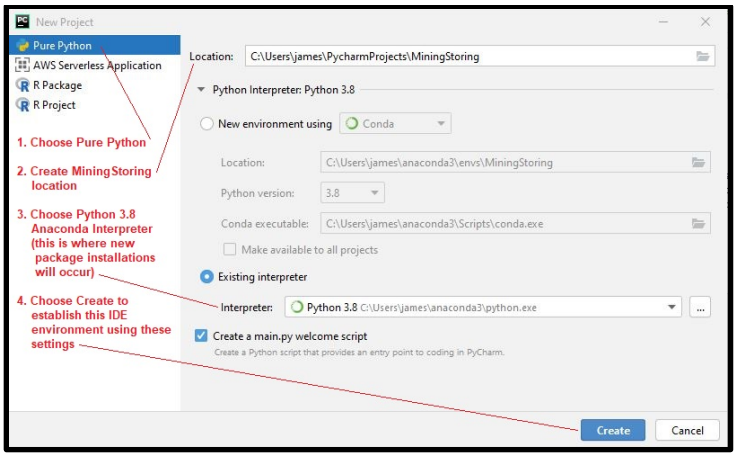
0) To ensure your various Python package libraries are available to import, be sure to use Anaconda Navigator to launch PyCharm.



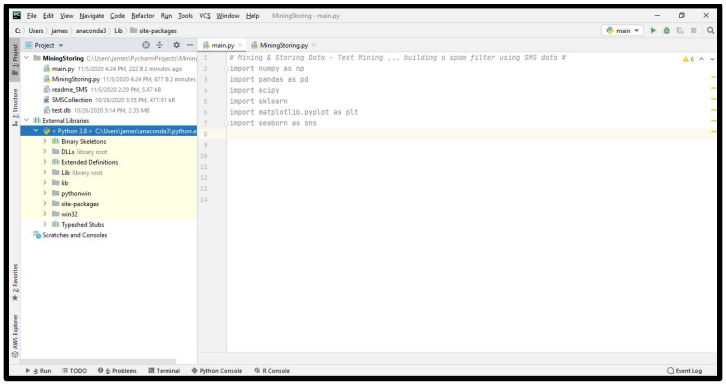
1) Launch PyCharm and choose a New Project.



1) Choose a Pure Python project,
2) Change your project location to create a MiningStoring folder
3) Be sure to choose the Python interpreter as Python 3.8 associated with Anaconda.
4) Choose Create to open PyCharm with these settings.

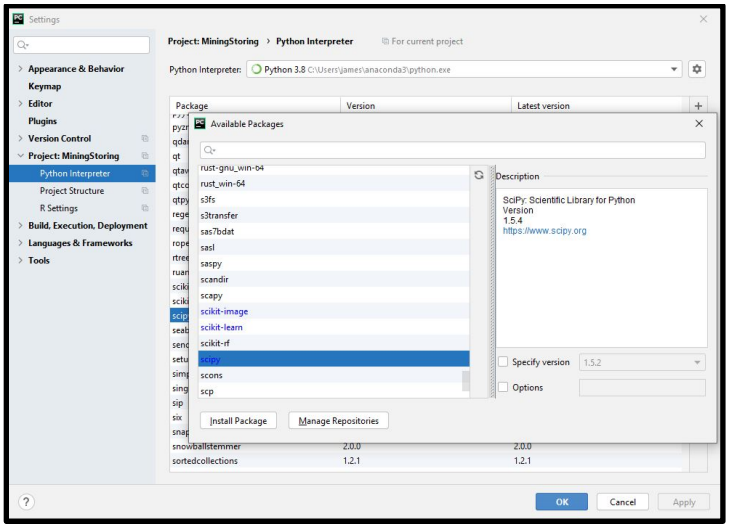


5) *Download* the SMSCollection, readme_SMS, and test.db files from the Week 11 Canvas Module to the MiningStoring directory created in the last step. The readme_SMS file explains the data in the SMSCollection file. The test.db file is an SQLite database.



All visualizations may be viewed at 400% magnification for clarity.

6) *When importing a new package library, check the Python Interpreter to ensure the package is installed via Files->Settings->Python Interpreter or left-click Python3.8 in the lower right screen corner choosing Interpreter Settings...* Once in the Python Interpreter, pan down to find the particular Python package library and double click on package version. In the resulting window, all installed packages are highlighted in blue.



Reading a text file into a Pandas DataFrame:

We learned about Pandas in Tutorials Seven and Eight-A. Again in this tutorial, we will look at DataFrame. For more specific details about Pandas not covered in this tutorial, please refer to the following URL:

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

1) Anaconda will load the libraries in PyCharm. To access the libraries we are using for text mining in Tutorial Eight-B (e.g., numpy, pandas, scipy, seaborn, sklearn, and matplotlib.pyplot), enter the following code into Main.py in the MiningStoring project directory of PyCharmProjects:

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

2) To format the run window to handle the size of the csv data file we will be data mining in Tutorial Eight, enter the following Python code:

```
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
```

3) To read the SMSCollection file into a pandas dataframe, type the following Python code into main.py via PyCharm in the line below the previous code above.

```
data_file = 'SMSCollection'   # assigns file name variable data_file
sms_raw = pd.read_table(data_file, header=None)     # reads SMS file
sms_raw.columns = ('spam', 'message')          # assigns column names
```

4) Once the SMSCollection file is loaded into a dataframe table, we can begin text mining. Any of the following Python functions may be used with the dataframe sms_raw, once it has been assigned.

```
print(sms_raw.head(2))     # returns top two rows
print(sms_raw.tail(5))     # returns bottom five rows
print(type(sms_raw))       # returns type of dataframe
print(sms_raw.shape)       # returns row & column count
print(sms_raw.describe()) # returns data profiling metrics
print(sms_raw.info())      # returns column attributes
```

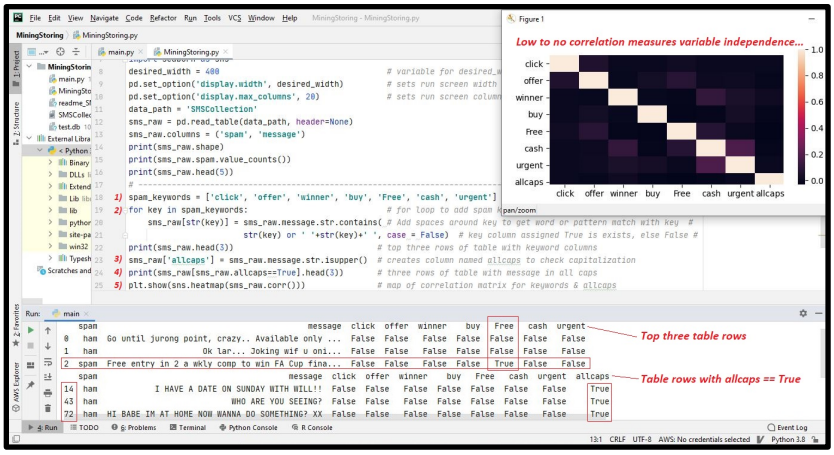
5) The MiningStoring.py file has the Python code for all screenshots in Tutorial Eight-B.

Text Mining:

Text mining, also referred to as text analytics, transforms the free (unstructured) text in documents and databases into normalized, structured data suitable for analysis or to drive machine learning (ML) algorithms. Widely used in knowledge-driven organizations, text mining is the process of examining large collections of documents to discover new information or help answer specific research questions. Text mining identifies facts, relationships and assertions that would otherwise remain buried in the mass of textual big data. Tutorial Ten explains Natural Language Processing (NLP), which is similar to text mining. Text mining and NLP have difference in the end goal and difference in methods. Text mining techniques are usually shallow and do not consider the text structure, while using bag-of-words, n-grams, and possibly stemming. NLP methods usually involve

the text structure, sentence splitting, part-of-speech tagging, and parse tree construction. For this exercise in Tutorial Eight-B, we will demonstrate general text mining methods via Python code towards developing a spam filter.

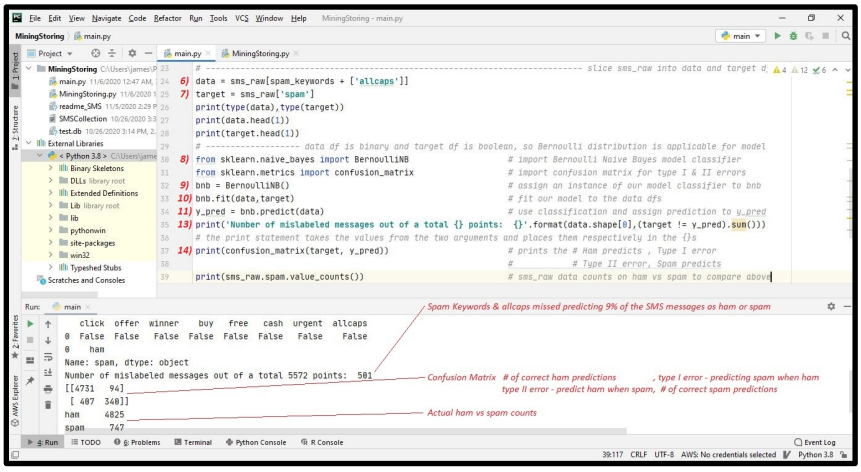
1) We imported the five libraries and added Python code from the prior reading df section. In the PyCharm screenshot below, you will see how to (1) assign spam keywords to a list, (2) use a 'for loop' to check each message column row to see if a spam keyword is in it, (3) add an allcaps column in sms_raw to check if message is all upper case, (4) print sms_raw rows with allcaps == True, and (5) plot a heat map to identify correlation between spam keywords and allcaps. The resulting heatmap plot shows little to no correlation and suggests strong independence between spam keyword and 'allcaps' variables.



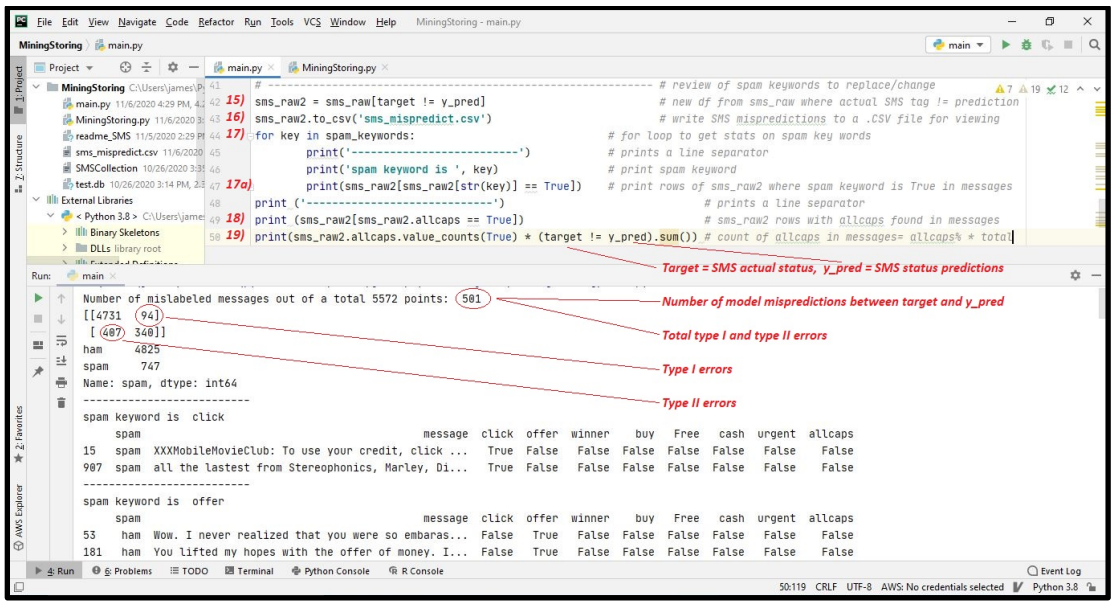
All screenshots may be enlarged 400x for clarity

2) In the following PyCharm screenshot, we (6) slice the sms_raw df columns spam_keywords and 'allcaps' into data df, (7) slice the sms_raw df column 'spam' into target df, (8) import a Bernoulli Naïve Bayesian classifier model and a confusion matrix prediction checker, (9) assign a variable to the classifier model, (10) fit the classifier model to data and target, (11) assign the classifier model prediction to y_pred, (12) count the number of total data points vs. mislabeled predictions (type I & II errors), (13) calculate the prediction check confusion matrix, and (14) print the sms_raw df column 'spam' value counts. More information on the Bernoulli Naïve Bayes classifier is available via the URL:

https://scikit-learn.org/stable/modules/naive_bayes.html#bernoulli-naive-bayes



3) In the two prior PyCharm screenshots, the spam keywords and allcaps column attributes are used as rules to detect spam SMS messages from regular (ham) SMS messages. The mispredictions of type I and type II errors are calculated. In the following PyCharm screenshot, we (15) create a data frame sms_raw2 which holds all the mispredictions of type I and type II errors, (16) write the sms_raw2 data frame to a .CSV file for review in Notepad or Excel, (17) use a 'for loop' to list all the type I and type II errors where a spam keyword was found in the SMS message, (18) listing of type I and type II errors where allcaps were true, and (19) a count of rows where type I and type II errors showed allcaps were true.



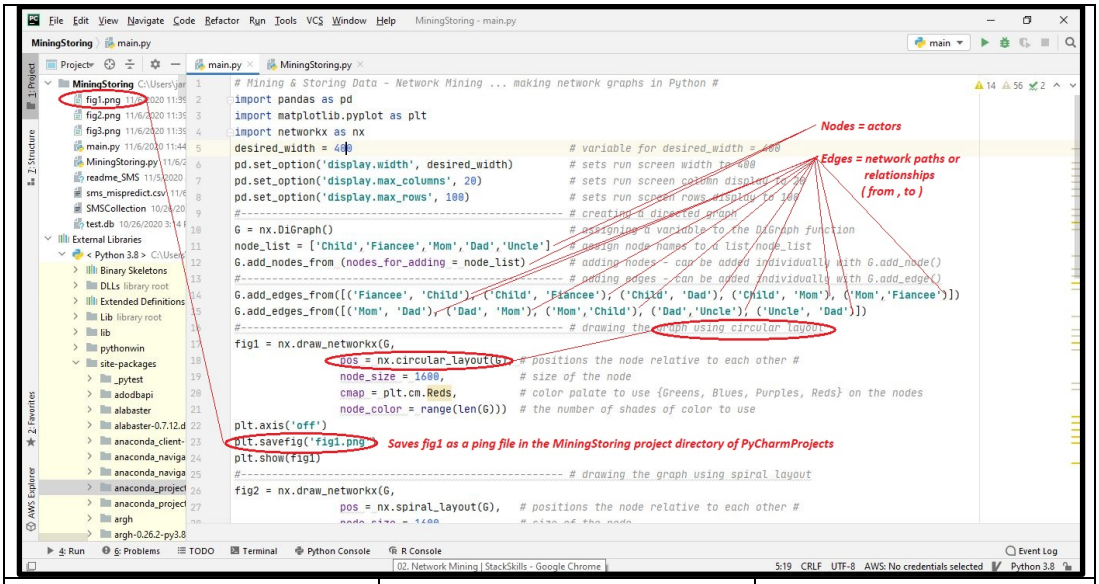
4) **Exercise deliverable:** Review the sms_mispredic.csv file using Notepad, Excel, or another text editor. Using the Python code in this text mining example (steps 1 to 14), add or change spam_keywords to improve the model fit and minimize type II errors. If you prefer to duplicate this text mining example from your project data set, then please do. Capture a PyCharm screenshot showing the number of mislabeled predictions as well as the confusion matrix for use later in this tutorial.

Network Mining:

Graph and network mining has leaped to the forefront of data mining research, spurred by an avalanche of structured data from applications such as bioinformatics, cheminformatics, online social networks, and sensor networks. This structured data is often best represented either as a set of independent graphs or as a large network of interconnected nodes. The proliferation of graph and network data is both an opportunity and a challenge. Graph mining is playing an increasingly important role in the analysis of highly structured data such as chemical compounds, proteins, very large scale integration (VLSI) designs, and program execution traces. Examples of graph mining applications include predicting protein function (graph classification), searching for compounds with certain substructures (graph similarity search), and detecting software bugs in programs

(graph anomaly detection). Network mining offers the opportunity for analyzing large collections of inter-related objects such as Web graphs, social networks, and biological networks. Common applications are assessing the spread of epidemics (influence maximization), predicting future collaboration between authors (link prediction), and finding authoritative web pages (link-based node ranking). Traditional algorithms, which typically assume that objects are independent and identically distributed, are not appropriate for mining such data. Furthermore, some of the data mining tasks of interest (particularly in network data) have no counterparts in record-based data (e.g., influence maximization and link prediction) .For more information on network mining, refer to the URL: https://networkx.org/ .

1) **Network Graphs:** The following Python screenshot uses Python code and the networkx package library to show relationships within a family unit {Fiancée, Child, Mom, Dad, Uncle}. Several different graph layouts are used to plot a network graph, specifically circular, spiral, and spring layouts. For more information on the numerous networkz graph layouts, refer to the URL: https://networkx.org/documentation/stable//_modules/networkx/drawing/layout.html



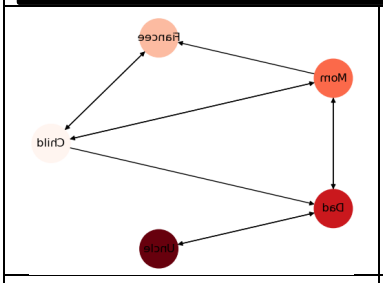


Figure 1 – Circular Layout

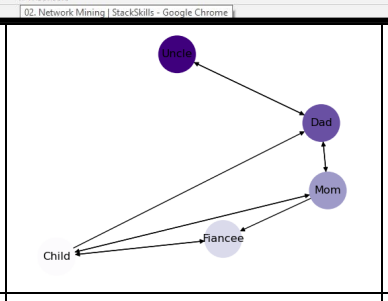


Figure 2 – Spiral Layout

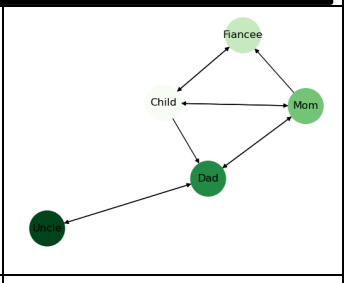
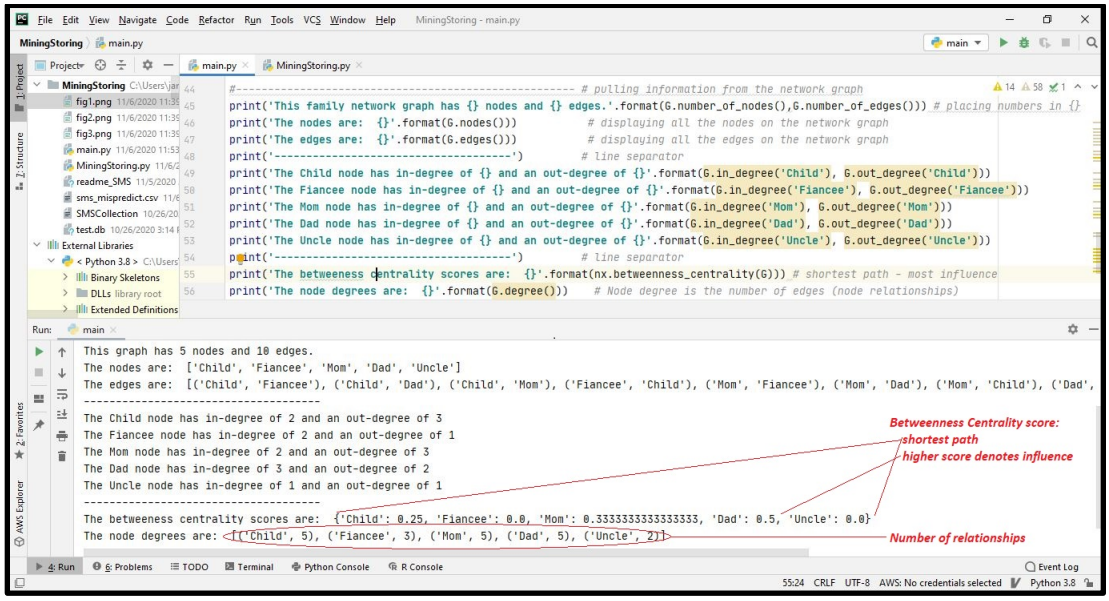


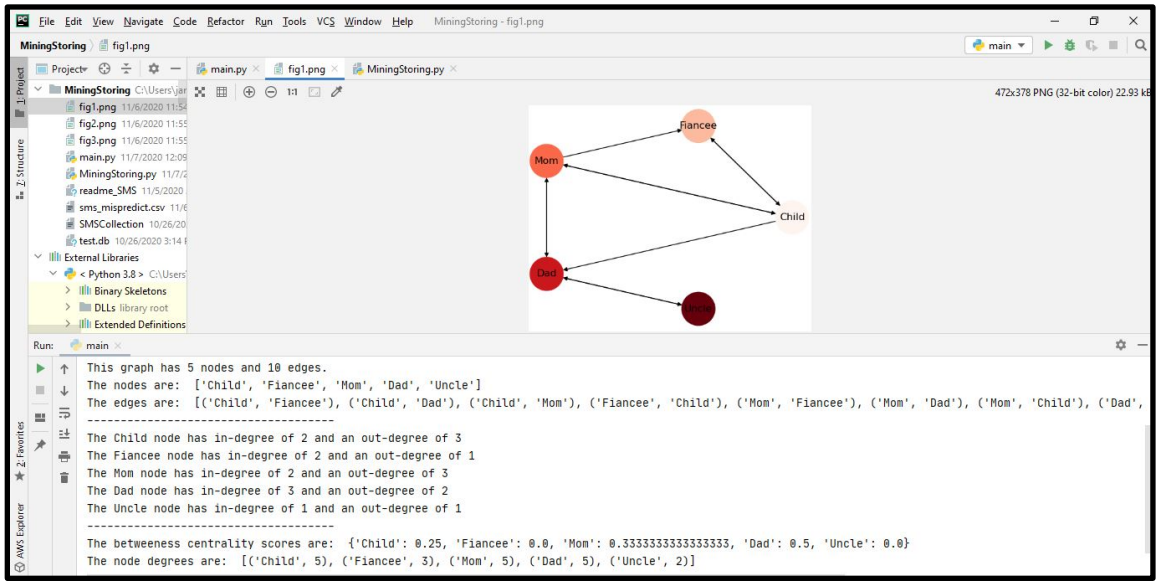
Figure 3 – Spring Layout

2) ***Network Graph Information:*** Once the network graph is plotted, Python code allows you to summarize the nodes, edges (i.e., network paths or relationships), betweenness centrality to identify the shortest network path, and number of degrees (i.e., relationships) for each node.

The following PyCharm screenshot illustrates the Python code to retrieve network graph information.



3) **Exercise deliverable:** Construct a network graph from your project data set, if applicable. If a network graph is not applicable, then choose four friends from Facebook or four colleagues from your LinkedIn network. (0) Code the names of your friends with their initials and the number of friends or colleagues in their networks. (1) Identify if the any of the four have relationships with each other on the social platform. (2) Build a network graph and try a few different graph layouts and choose the one you like. (3) Save the network graph as a ping file. (4) Gather the same network graph information as detailed in the PyCharm screenshot above. (5) Capture a PyCharm screenshot displaying the ping file above the run window that displays the information. An example of this PyCharm screenshot configuration is below.



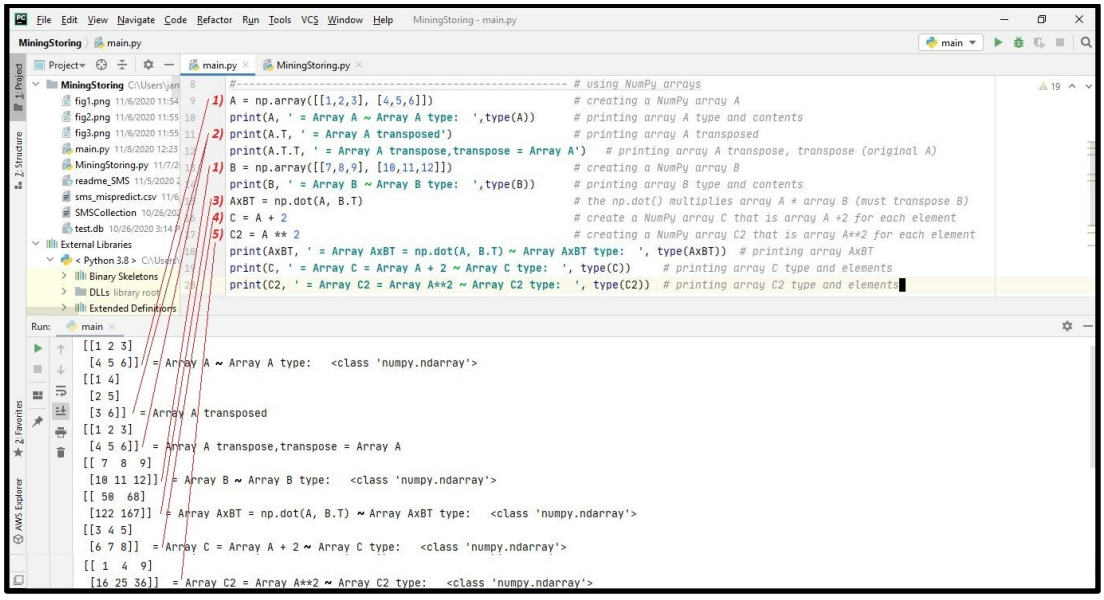
Matrix:

Should you take the red pill or blue pill? In Python, we can use the Numpy or Pandas libraries to help with performing matrix math. NumPy is a powerful python library that expands Python's functionality by allowing users to create multi-dimensional array objects (ndarray). In addition to the creation of ndarray objects, NumPy provides a large set of mathematical functions that can operate quickly on the entries of the ndarray without the need of for loops. The Pandas (i.e., PANel DAta) Python library allows for easy and fast data analysis and manipulation tools by providing numerical tables and time series data structures called DataFrame and Series, respectively. Pandas was created to (1) provide data structures that can handle both time and non-time series data, (2) allow mathematical operations on the data structures while ignoring the metadata of the data structures, (3) use relational operations like those found in SQL (join, group by, etc.), and (4) handle missing data. Pandas is built on top of NumPy, relying on ndarray and its mathematical functions. Likewise, Pandas relies heavily on the NumPy array for the implementation of pandas data objects and shares many of its features. In addition, pandas builds upon functionality provided by NumPy. For more information on matrix math under NumPy and Pandas in Python, refer to the following URLs:

https://www.python-course.eu/matrix_arithmetic.php (NumPy)

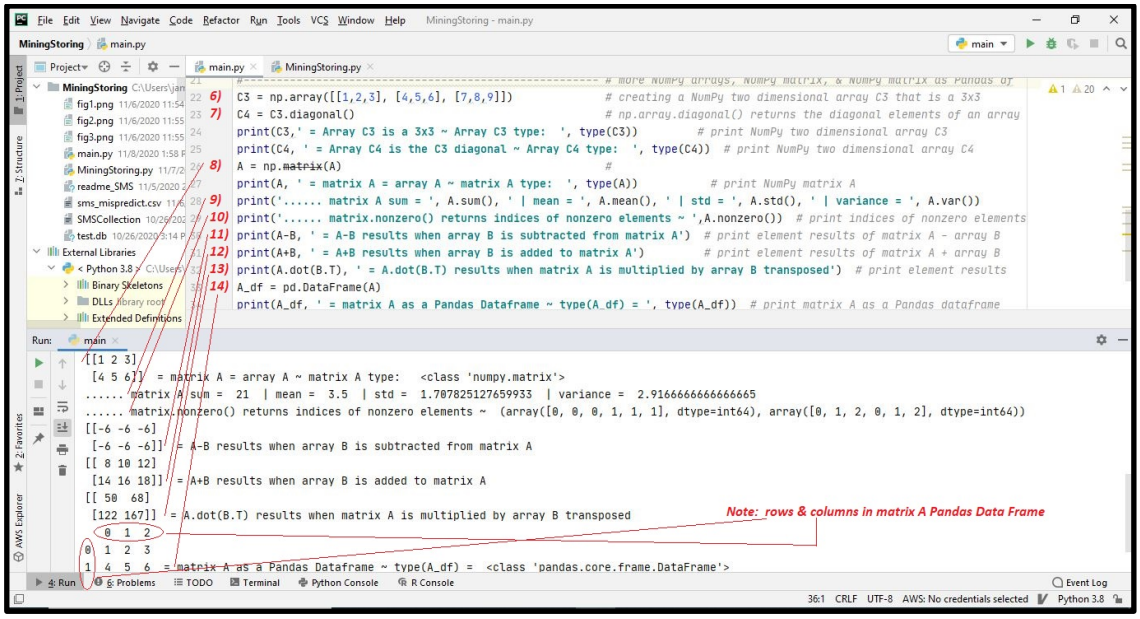
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dot.html (Pandas)

1) The Python code in the following PyCharm screenshot reviews some basic matrix math using NumPy two dimensional arrays. Specifically, we execute (1) np.array to create a NumPy array A and array B, (2) A.T to transpose array A (i.e., array A.T.T is the same as array A), (3) np.dot() that multiplies two arrays, (4) adding 2 to each element in array C, and (5) raising each element in array C2 to the power of 2 (i.e., squaring each item).



2) For more examples of arrays, matrices, and data frames, the Python code in the following PyCharm screenshot reviews more functionality using NumPy arrays, a NumPy matrix, and a NumPy matrix converted to a Pandas data frame. For precision in math calculations, a NumPy

array or matrix is recommended. The following PyCharm screenshot specifically demonstrates the execution of (6) np.array to create a 3x3 NumPy two dimensional array C3; (7) use of the array.diagonal() function to display NumPy array C4 as the diagonal of C3; (8) use of np.matrix() to convert array A into matrix A; (9) use of the matrix.sum(), matrix.mean(), matrix.std(), and matrix.var() functionality; (10) use of the matrix.nonzero() function; (11) subtracting array B from matrix A; (12) adding array B to matrix A; (13) use of the matrix.dot() function to multiply matrix A by array B transposed; and (14) converting a NumPy matrix to a Pandas data frame.

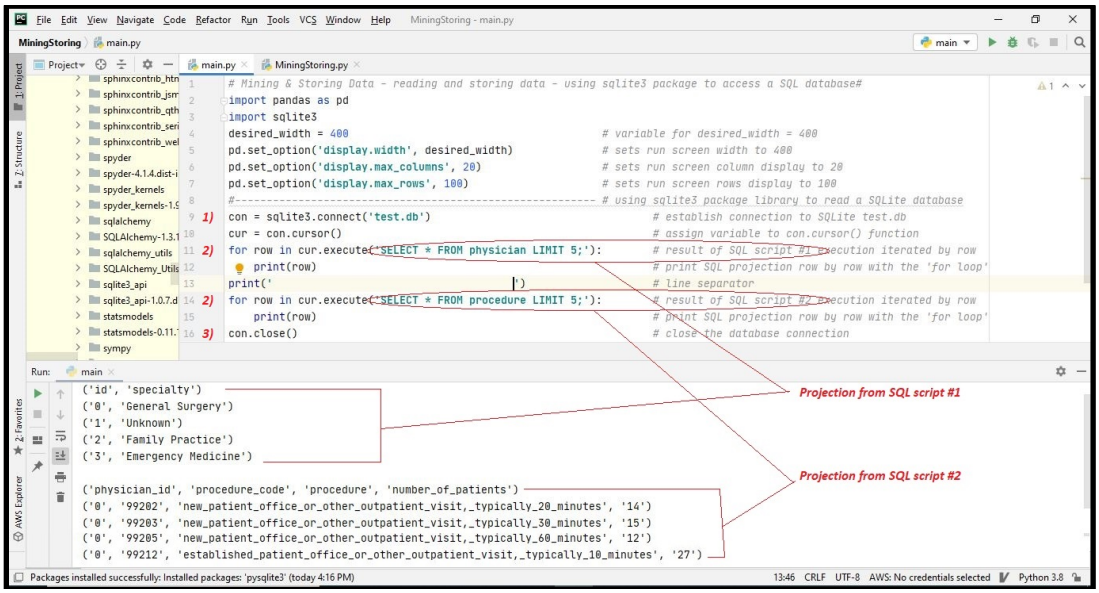


3) **Exercise deliverable:** Use your project data set or a data set you derive to illustrate array, matrix, and Pandas data frame functionality. In a PyCharm screenshot, show the array/matrix and data frame elements, mean(), sum(), std(), var(), and if applicable diagonal(). Note that an array or matrix applies aggregate functions to all elements. The data frame will apply the aggregate functions to a column (i.e., in the screenshot above print(A_df.mean()) would print 2.5 for column 0, 3.5 for column 1, and 4.5 for column 2). The diagonal() function is not applicable to data frames as the matrix is segregated by rows and columns. Save the PyCharm screenshot for use later in this tutorial.

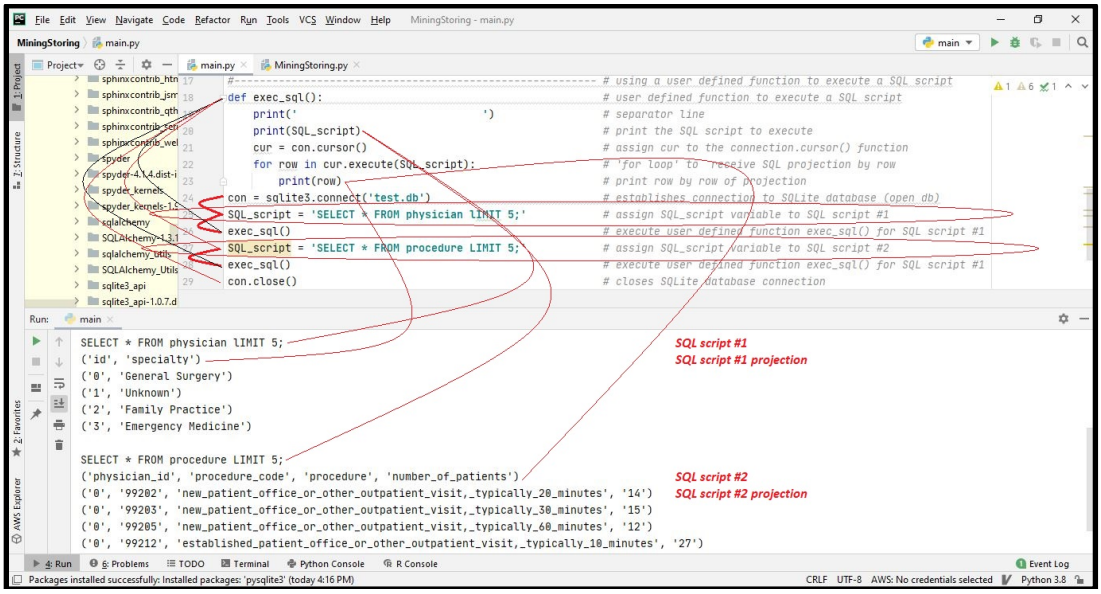
Storing and Reading Data:

In prior Tutorials, we have looked at reading CSV or flat text into Panda data frames. This section of the tutorial will look at reading as well as storing data using Python code, specifically in SQLite databases and CSV files. For reading and writing data to SQLite databases, we will import and use the sqlite3 package library. For more information on the sqlite3 package library not covered in this tutorial, please refer to the URl: https://docs.python.org/3/library/sqlite3.html

1) ***SQL Script Execution:*** The following PyCharm screenshot demonstrates how to (1) open a SQLite database, (2) execute SQL scripts, and (3) close the SQLite database connection. In this particular example, two SQL scripts are executed from the SQLite database test.db located in the …PyCharmProject/MiningStoring. The SQLite database should always be in the subdirectory where the Python program is executed or the path to the database should be specified in the .connect function (i.e., sqlite.connect(path/database.db).

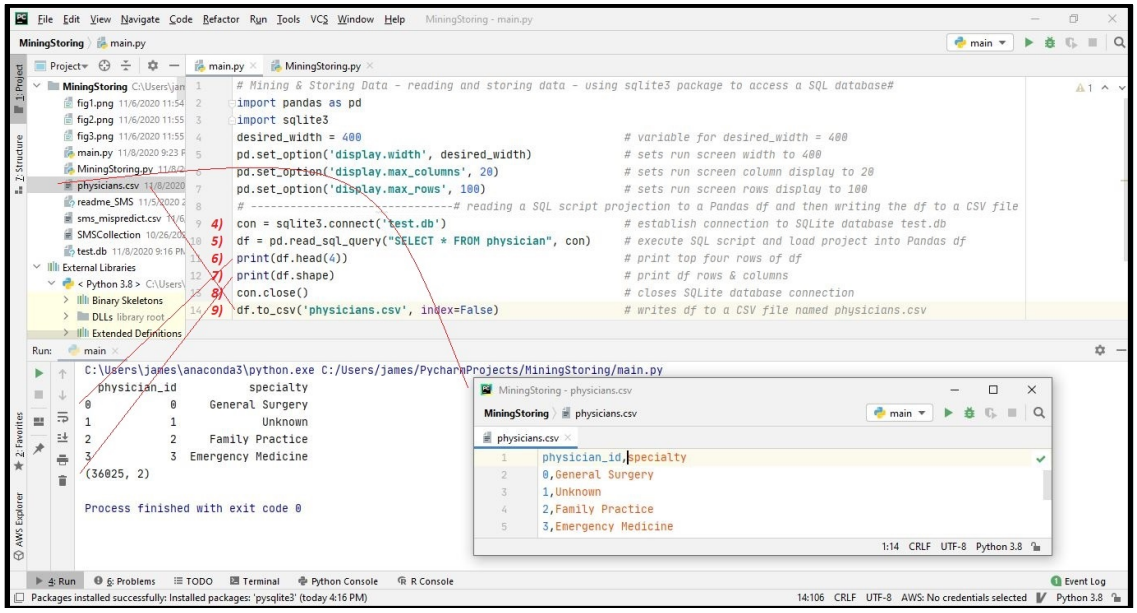


The following PyCharm screenshot displays Python code using a user-defined function to execute the same two SQL scripts from the previous example. It should be noted SQL scripts may be data manipulation language to project or insert data, data definition language to create database objects (i.e., tables, views, indexes, etc.), or data control language to assign rights and security.

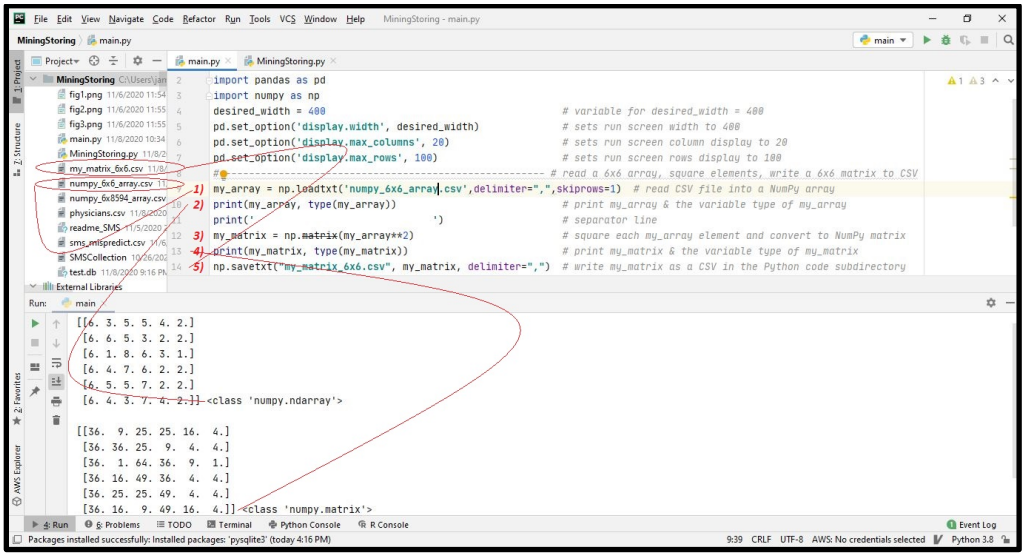


2) ***Reading a SQL Script Projection into a Pandas Data File and Saving the df as a CSV:*** The following PyCharm screenshot demonstrates how to (4) open a SQLite database, (5) read an SQL script SELECT statement into a Pandas df, (6) check the first four rows of the df, (7) list the df's number of columns and rows, (8) close the SQLite database connection, and (9) write the df to a CSV file. In this particular example, the SQL script projection is read directly into a

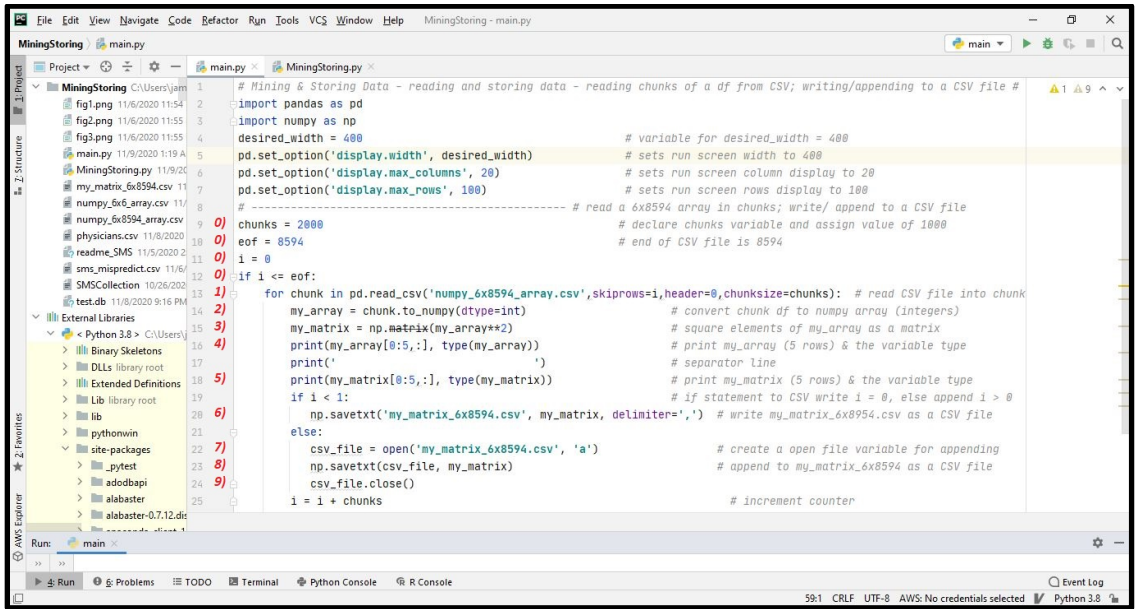
Pandas df and the resulting df is written to a CSV file in the …PyCharmProjects/MiningStoring directory.



3) ***Reading a CSV file into NumPy array; Writing manipulated matrix to a CSV file.*** In this example, we read a CSV called numpy_6x6_array.csv that is stored in the Python code subdirectory …PycharmProjects/MiningStoring. As a general rule, using the Pandas import method is a more 'forgiving', so if you have trouble reading directly into a NumPy array, try loading in a Pandas data frame and then converting it to a NumPy array. NumPy's loadtxt method reads delimited text. We specify the separator as a comma. The data we are loading also has a text header, so we use skiprows=1 to skip the header row, which would cause problems for NumPy. We use the savetxt method to save the resulting matrix to a CSV file. In the following PyCharm screenshot, we specifically (1) load a CSV file into a NumPy array, (2) print the array and variable type, (3) square the elements of the array and convert the array to a matrix, (4) print the matrix and the variable type, and (4) save the matrix to a CSV file.



4) ***Reading a CSV file into Pandas data frame in chunks; Writing/appending to a CSV file.*** In this example, we read a CSV file numpy_6x8594_array.csv that is stored in the Python code subdirectory …PycharmProjects/MiningStoring. The CSV file has 8,594 rows, yet let us imagine the file is too large to load all the array elements. We can load a data frame in chunks and then convert the chunk data frame into a NumPy array. We use the savetxt method to save the resulting matrix to a CSV file, but after the first chunk write we need to append the latter chunk writes. In the following PyCharm screenshot, we specifically (0) establish chunks size, i counter, and eof_size variables to use in an 'if statement' (1) load a CSV file into a chunk data frame using a 'for loop', (2) pass the chunk data frame to a NumPy array, (3) square the NumPy array and save as a NumPy matrix, (4) print the array first five rows and variable type, (5) print the matrix first five rows and variable type, (6) write the NumPy matrix to a CSV file for the first pass chunk, (7) create a file handle to open the CSV file for appending, (8) append successive chunk matrices to the CSV file, and (9) close the file handle.



All Python code in PyCharm screenshots Is available in MiningStoring.py

5) **Exercise deliverable:** (1) Practice reading and writing CSV files from your project data set. Choose a SQLite database file from Tutorials One or Four and (2) execute a SQL script as well as read an SQL SELECT statement into a Pandas dataframe. Save PyCharm screenshots from both deliverables and save for use later in the Tutorial.

Deliverables from Exercises Above:

Take the noted deliverables (i.e., PyCharm screenshots) from each exercise above (i.e., total of 8 screenshots), insert the screenshots into a MS Word document as you label each screenshot deliverable from the corresponding exercise, save the document as

one PDF, and submit the PDF to the Tutorial Eight-B Canvas Assignment uplink having the following qualities:

a. Your screenshots for each of the four exercise deliverables above is included.
b. Each screenshot is labeled appropriately for each exercise.
    i. Text Mining …….. 1 screenshot
    ii. Network Mining …….. 1 screenshot
    iii. Matrix …….. 1 screenshot
    iv. SQL, NumPy, & Pandas Read/Write CSV …….. 2 screenshots
    Total PyCharm screenshots to submit …… 5 screenshots
c. All screenshots are legible output for each PyCharm exercise deliverable.