Python Basics

Types of Objects:

Strings (more)
- Surrounded by single or double quotation marks
- Strings are often words or sentences but may be any type of character
- You can create empty strings or temporarily create one to be filled later (char = ‘’)

Integers (more)
- Whole numbers that may be used for mathematical calculations

Floats (more)
- Similar to integers, but include decimals

Lists (more)
- Items contained between brackets [] and separated by commas
- Can store any combination of strings, ints, floats, etc
- To create an empty list -> list = []
- Lists’ index starts at 0 (first element accessed list[0])

Tuple (more)
- Items contained between parentheses ()
- Used to store multiple items in a single variable

Dictionaries (more)
- key/value pairs stored between curly brackets {}
- Can store any combination of things

Python Basic Operations and Functions:

Print function (more)
- Will return any message in the output section

Basic Mathematical Operations (more)
- Addition (+), subtraction (-), multiplication (*), division (/)
- Floor (integer) division (//) rounds to whole number
- Truncated division (%) calculates the remainder

.append() function (more)
- Adds an item to the end of a list

.split() function (more)
- Takes a string and divides it into smaller strings in a list based on a dividing character
- You can choose dividing character but if left blank, it will automatically split at a space

Python Syntax and Features:
- No semicolons needed at the end of lines
- To initialize and declare a variable, use =
  - Ex. string = “hello world”, list = [1,2,3], tuple = (“hello”, “world”), dictionary = {key: value}
  - Don’t need to specify type when declaring

For Loops (more)
- Format: for *variable name* in *what you want to loop through*:
- Used to iterate through an object (string, list, etc) a specific number of times

While Loops (more)
- Used to repeat a task until a certain condition is met
- It’s important to update your relevant variables to avoid getting caught in a never-ending loop
-
```
while i <= 10:
    print(i)
    i = i + 1 #i += 1
```

If/Elif/Else Statements (more)
- Used to choose a snippet of code to run if a certain condition is met
- If there are 3 or more possible conditions, use elif for all the middle conditions
-
```
for num in list3:
    if num > 5:
        print(str(num) + " is greater than 5")
    elif num < 5:
        print(str(num) + " is less than 5")
    else:
        print(str(num) + " is equal to 5")
```

Functions (more)
- Blocks of code that must be called on in order to run them
- Opening line names the function (def ____)
- You must always return something when writing functions
-
```
list4 = ['baseball', 'football']
def add_to_our_list(our_list, new_word):
    our_list.append(new_word)
    return our_list
add_to_our_list(list4, 'soccer')
```

Slicing (more)
- Used to return a range of characters from a list, string, or tuple
- Format: list[pos1 : pos2]
  - Pos1 is inclusive and pos2 is exclusive
- Slice to the end or from the start by leaving one side of the colon empty

Libraries Used:

Import statistics (more)
- Allows us to do mathematical statistics on numerical values

IMPORTANT: remember to import ____ when you are using libraries

BeautifulSoup

Libraries Used:

Import requests (more)

- Simple library that allows us to request html data from a website

From bs4 import BeautifulSoup (more)

- Library that allows us to pull data from the html website

IMPORTANT: before you import requests and BeautifulSoup, make sure you pip install requests, pip install mysql-connector-python, and pip install bs4 in your terminal (conda install works too)

Functions Used:

Requests.get() (more)

- Sends a get request to the url provided in the function
- Url must be in quotes (it's a string)

BeautifulSoup() (more)

- Outputs all of the html tags and data from the url response created from requests.get()
- 2 parameters: url response and 'html.parser'

```
url = requests.get("https://www.basketball-reference.com/teams/GSW/2022.html")
soup = BeautifulSoup(url.text, 'html.parser')
#print(soup)
```

.prettify() (more)

- Makes our soup look much nicer by putting each tag and string on its own line and indexed when necessary
-

```
print(soup.prettify())
```

.find() and .find_all() (more)

- .find() will find and output the first iteration of the wanted tag
- .find_all() will find and output each iteration of the wanted tag. Each iteration will appear as its own item in a list
- The first parameter inside the function is the tag that we want to find
- We can have additional parameters such as id or class to find more specific tags within the html

```
rosterTable = soup.find("div", attrs = {'id' : 'div_roster'}).find('table')
#print(rosterTable)
```

```
tableRows = soup.find_all("tr")
#print(tableRows)
#print(tableRows[1])
```

Other Notes

Converting object types (more)

- In order to print multiple things on the same line, they must all be strings
- This means we may need to convert certain objects into a string
-

```
In [1]: x = 11
        print("The Red Wings have " + str(x) + " stanley cup championships.")

        The Red Wings have 11 stanley cup championships.
```

- We may also need to convert string representation of numbers into integers or floats
  - str() or float()
-

```
In [2]: num_string = "1000"
        int(num_string) / 10

Out[2]: 100.0

In [3]: num_string = "0.80"
        float(num_string) + 0.2

Out[3]: 1.0
```

- We cannot turn things into lists, tuples, or dictionaries but we can use type() to verify the type of object we are using
-

```
In [5]: college = {'name' : 'University of Michigan',
                   'city' : 'Ann Arbor',
                   'nickname' : 'Wolverines'}
        print(type(college))

        <class 'dict'>
```

ESPN Workaround

- When working with data from ESPN, we need to use a workaround by creating an headers object called User-Agent and adding that as a parameter to the requests.get() function as shown below

```
headers = {'User-Agent': '...'}
url = requests.get("insert link here", headers = headers)
soup = BeautifulSoup(url.text, 'html.parser')
```

SQL

More SQL Cheat Sheets : here
Libraries Used:
Import mysql.connector
- Used to connect Python to SQL database

Functions Used in Spyder:
Mysql.connector.connect (more)
- Creates connection to mySQL server
- Parameters are user ('wsa'), password ('LeBron>MJ!'), host ('34.68.250.121'), and database ('Tutorials-Winter2024')

```
cnx = mysql.connector.connect(user = "wsa",
                              host = "34.68.250.121",
                              database = "Tutorials-Fall2023",
                              password = "LeBron>MJ!")
```

.cursor() (more)
- Creates a cursor object in Python that is used to execute SQL commands
- One parameter (buffered = True)

SQL statement
- Basic statements that SQL can comprehend and subsequently execute actions that we want
- In Spyder, we use this to insert our data into our SQL table

.execute() (more)
- Used to execute sql statements
- We use it to execute the statement that will insert our data into the SQL table

.commit() (more)
- Verifies and completes all of the executions made my a cursor object

Navigating SQL and Creating Tables:
Creating Tables
- Schemas > Tutorials-Fall2023 > Tables (right click) > Create Table
- To update table attributes (columns & data types), hover over specified table and click on wrench
- To view contents of table, hover over table and click on the lighting bolt

Data types (more)
- SQL can use a variety of data types but we are only interested in a select few
- INT is used to store integer values
- VARCHAR(45) is used to store string values
  - 45 character limit

Table Constraints (more)
- When creating each column in the table, there are 8 constraints you can activate
- Primary Key (PK) uniquely identifies each row
- Not Null (NN) ensures a value cannot be null
- Auto Increment (AI) increases the value by 1 with each next row
- Unique (UQ) ensures each value is different
- Binary (B) means only binary values can be in the column (0s or 1s)
- The rest don't have importance to us

Default value
- If there is no value passed into the table, sql will automatically fill the slot with a default value
- Typically, we use NULL for these values

SQL Statements:
Select from (more)
- Allows us to select certain values from our table
- SELECT * FROM gives us all columns in each row
  - Format: SELECT [what you want] FROM [SQL schema].[specific table];
- We can select specific columns by replacing * with the column names, separated by commas

```
SELECT playerName, weight FROM `Tutorials-Winter2023`.NBA_rosters_justin_yang
```

Where clause (more)
- Allows us to only select certain data where a specific clause is met
- WHERE … added to end of SELECT statement

```
SELECT playerName, weight FROM `Tutorials-Winter2023`.NBA_rosters_justin_yang
WHERE weight >= 220;
```

Order by (more)
- Allows us to sort a table by values
- Automatically sorts from low value to high but adding DESC at the end switches the outcome

```
SELECT * FROM `Tutorials-Winter2023`.NBA_rosters_justin_yang
ORDER BY experience DESC;
```

Result Grid | Filter Rows: | Edit:

id	playerName	position	height	weight	experience
6	Andre Iguodala	SF	6-6	215	17
3	Stephen Curry	PG	6-2	185	12
5	Draymond Green	PF	6-6	230	9
13	Otto Porter Jr.	PF	6-8	198	8
14	Klay Thompson	SG	6-6	215	8
17	Andrew Wiggins	SF	6-7	197	7

Insert into (more)
- Allows us to insert data into the SQL table
- Format: INSERT INTO [table name] ([column names]) VALUES ([values]);

```
INSERT INTO `Tutorials-Winter2023`.NBA_rosters_justin_yang
(playerName, position, height, weight) VALUES ('Billy Bob', 'C', '6-5', 300);
```

Update (more)
- Allows us to update 1 or more values in the table
- Helpful when cleaning our data
- Format: UPDATE [table name] SET [variable = something] WHERE [condition is met];

```
UPDATE `Tutorials-Winter2023`.NBA_rosters_justin_yang SET playerName = 'Joe Schmo' WHERE id = 18;
```

Delete (more)
- Allows us to delete information from our tables
- Format: DELETE FROM [table name] WHERE [condition is met]

```
DELETE FROM `Tutorials-Winter2023`.NBA_rosters_justin_yang WHERE id = 18;
```

Pandas

Libraries Used:
Import pandas as pd (more)
- Used to compute statistical representations of our data as well as manipulate dataframes

Setup:
Extracting csv files from SQL
- To manipulate the data we have been using is recent weeks, we need to extract it from SQL
- Instructions are in intro slides and demo video
- Very important to save csv file in same folder as your Jupyter Notebook file

Pandas Functions:
.read_csv() (more)
- Reads the csv file into Pandas
- Make sure csv file is in same folder as jupyter notebook file

.head() (more)
- Returns the first n rows in a dataframe

.info() (more)
- Prints info about the dataframe

.shape (more)
- Prints number tuple containing number of rows and columns (row, column)

.append() (more)
- Adds the rows of one df to the end of another df
- [df appended onto].append([new rows/df])

.drop_duplicates() (more)
- Eliminates duplicate rows within a df

.columns (more)
- Prints list of columns within a df

.rename() (more)
- Renames the column names
- Parameters include dictionary of column name changes in 'old name' : 'new name' format and inplace = True

.isnull() (more)
- Returns True for all null values in df and false for non-null values

.describe() (more)
- Prints numerical summaries including count, mean, standard deviation, min, median, max, and quartiles for each column containing integers

.value_counts() (more)
- Counts the number of times a specific string/int appears in a column

.corr() (more)
- Prints correlation between each pair of columns that contain integers only

.groupby() (more)
- Splits the df into groups based on the parameter inside the parentheses
- Typically followed by second function such as .get_group(), .mean(), or .count() because it will not return anything on its own

Extracting data frame columns (more)
- Use single brackets [] to extract a column from the df but double brackets [[]] to set the column as its own df

```
opponent_col = df['opponent']
opponent_col
```

```
opponent_df = df[['opponent']]
opponent_df.head()
```

Accessing data frame rows (more)
- Use the .loc[] function or the .iloc[] function
  - loc requires label indexing (put the label of the row in the brackets) while iloc requires integer indexing (put the index of the row in the brackets)

```
row1_loc = alt_df.loc['2011-09-10']
row1_loc
row1_iloc = alt_df.iloc[1]
row1_iloc
```

- To access multiple rows, you can slice the df

```
first_100 = df.iloc[:100]
first_100
```

Accessing specific values within a data frame (more)
- Start by accessing the row and then access the column
- This can easily be done in one line

```
rush_yrds_natty = df.iloc[164]['rush yards']
rush_yrds_natty
```

Pandas/Matplotlib

Libraries Used:

Import matplotlib.pyplot as plt (more)

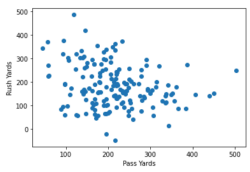
- Used to create graphical representations of our data and display trends

Matplotlib Plot Types:

.scatter() (more)

- Creates a scatter plot based on two parameters
- First parameter is values along x-axis, second is values along y-axis

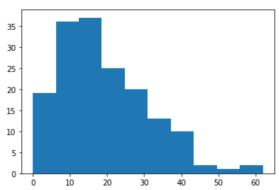
```
plt.scatter(df['pass yards'], df['rush yards'])
```



.hist() (more)

- Creates a histogram based on one parameter, the extracted df

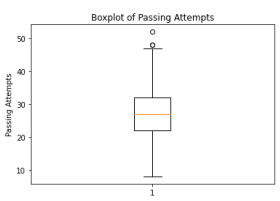
```
plt.hist(df["points against"])
```



.boxplot() (more)

- Creates a boxplot based on a series or dataframe similar to a histogram

```
plt.boxplot(pass_att_df)
```



.plot() (more)

- Used to make different graphs which can be specified in the parameters (kind = ___)
- Explore online to learn ab the types of graphs you can make
- Each kind of graph has its own additional parameters that can be used

Matplotlib Functions:

plt.show() (more)

- Must include this to actually show the graphs you create

plt.clf() (more)

- Clears the current graphical output for new graphs

plt.xlabel() (more)

- Adds a label to the x-axis

plt.ylabel() (more)

- Adds a label to the y-axis

plt.title() (more)

- Adds a label to the entire plot

plt.legend() (more)

- Adds a legend to the plot

Numpy

Libraries Used:
Import numpy as np (more)
- Great library when working with arrays, matrices, and numerical data

Numpy Functions:
np.array() (more)
- Converts a python list into an array
- Can be multi-dimensional if a list of lists is passed through

np.arange() (more)
- Creates a numpy array with evenly spaced values given the step size (default is 1)
- np.arange(start val, end val, step size)

np.linspace() (more)
- Creates a numpy array with evenly spaced values given the number of values in the array
- np.linspace(start val, end val, number of points)

np.empty() (more)
- Creates an empty array in the specified shape that is passed through using (rows, columns)

np.full() (more)
- Creates an array in the specified shape that is passed through with the value passed through
  - np.full((rows, columns), value)
- Alternatively, no value needs to be specified if the functions np.ones() or np.zeros() is used bc these arrays will be filled with ones or zeroes

.dtype (more)
- Creates a data type object

.shape (more)
- Returns the shape of an array

np.sum() (more)
- Returns the sum of an array

np.mean() (more)
- Returns the mean of an array

np.std() (more)
- Returns the standard deviation of an array

.T (more)
- Returns the transpose of a function (the rows are the columns and the columns are the rows)

np.linalg.norm() (more)
- Returns the magnitude of a vector which can be a full matrix or just one row

np.dot() (more)
- Returns the dot product of two arrays

np.matmul() (more)
- Multiplies two matrices and returns the outcome
- Important: you need the same number of rows in matrix 1 as the number of columns in matrix 2 in order to use this function

np.argmin() (more)
- Returns the index of the minimum value in the matrix
- If axis = 1 is passed through, a list containing the indices of the minimum value in each row will be returned

np.min() (more)
- Returns the minimum value in the matrix
- If axis = 1 is passed through, a list containing the minimum values in each row will be returned

np.reshape() (more)
- Reshapes an array to the shape that is specified when you run the function
- If you do not know the needed number of rows or columns, you can substitute the missing number with -1 and numpy will still be able to create the new array

Indexing and Slicing Arrays:
Indexing Arrays (more)
- We can index one-row arrays the same way as lists
- If an array has multiple rows, we can index it using a list
  - [1,3] finds the fourth element in the second row

Slicing Arrays (more)
- We can slice one-row arrays the same as lists
  - Start val is inclusive, End val is exclusive
- If an array has multiple rows, we can slice using lists
  - [:,1] gives us the second element in each row

More Numpy Features and Syntax:
Creating random arrays (more)
- We can use np.random.randint(start, end, size = (rows, columns)) to create an array with our choice of size and random integers in each cell

Mathematical operations and arrays (more)
- We can do mathematical operations on two matrices
- Operators: + (addition), - (subtraction), * (multiplication), / (division), // (integer division), ** (raising to the power of)

Scikit-Learn

Note:
There are a lot of functions and libraries used from previous tutorials including SQL, pandas, matplotlib, and numpy. Check those sections regarding how to use the functions from those libraries

Libraries Used:
Import sklearn (more)

- Specifically in this tutorial, we used the following:
  - From sklearn import preprocessing, svm
  - From sklearn.model_selection import train_test_split
  - From sklearn.linear_model import LinearRegression
  - From sklearn.preprocessing import PolynomialFeatures
  - From sklearn.metrics import mean_absolute_error, mean_squarred_error

Scikit-Learn Functions:
train_test_split() (more)

- Splits arrays into training and testing data
- First two parameters are the arrays containing our X-axis and Y-axis values
- Third parameter is test_size =0.x. Replace x to indicate what percentage of your data will be used for testing rather than training

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
```

LinearRegression() (more)

- Cretes a regression model (always paired with .fit() or .fit_transform())

.fit() (more)

- This trains/teaches our model given the X_train and Y_train data that we pass through the parameters

```
regr.fit(X_train, Y_train)
```

.score() (more)

- Returns the average accuracy of our model
- A score of 1 indicates perfect accuracy, 0 means complete mismatch

.predict() (more)

- Creates an equation that predicts the y-values given an x-value
  - $y = ax + b$ (linear)
  - $y = ax^2 + bx + c$ (polynomial)
- 1 parameter: X_test data

.intercept_

- Returns the intercept of the model prediction function

.coef_

- Returns the coefficient(s) of the model prediction function
- The last value in the array that is returned is the 'a' value, second to last is 'b', and so on

PolynomialFeatures() (more)

- Generates some polynomial features that we will use later on
- 1 parameter: degree = 2 for quadratic functions, otherwise degree = 1 indicates linear function

.fit_transform (more)

- Fits and transforms our X_train and X_test data to better fit the requirements of a polynomial regression model
- Include X_train and X_test in the parameters

mean_absolute_error() (more)

- Calculates the mean absolute error regression loss
  - This is the average magnitude of errors in our set of predictions without considering their direction

mean_squared_error() (more)

- Calculates the mean squared error regression loss
  - This is the mean squared error between the predicted and actual values
- Parameters include y_true = Y_test and y_pred = y_pred
- If you include a 3rd parameter (squared = False), it will calculate the root mean square error (RMSE)

Tensorflow

Libraries Used:

Import tensorflow as tf (more)

- Tensorflow is a python library that is commonly used in machine learning especially when dealing with neural networks

Non-Tensorflow Functions:

Pandas .sample() (more)

- Returns a random sample of a dataframe
- 1st parameter: frac = xx which determines what percentage of the df will be returned
- Optional 2nd parameter: random_state = xx which uses a previously randomly generated sequence so random return remains consistent

Pandas .drop() (more)

- Removes specific columns or rows from a df
- Specify columns or rows using axis = xx

Pandas .max() and .min() (more)

- Returns the maximum or minimum value in each column
- Can change to all rows or a specific row using axis= xx

Pandas .DataFrame() (more)

- Turns other object types, including lists, into a Pandas dataframe

Pandas .loc() and iloc() functions (more)

- Both are used to select particular data from a df
- .loc() allows you to pass through the name of a columns while .iloc() in more indexing-based so you must pass through the indices of the rows and columns that you want

Tensorflow Functions:

tf.keras.Sequential() (more)

- Creates our model by layering stacks of model layers
- We pass through a list with each item being a layer of our model (input -> hidden -> output)
- In our tutorial, we used Dense layers but there are other types
- Activation functions allow us to include non-linearity into our neural networks
    - We used one called relu (Rectified Linear Unit)
- Input_shape is the number of labels that we are passing through our model

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units = 64, activation = 'relu', input_shape = input_shape),
    tf.keras.layers.Dense(units = 64, activation = 'relu'),
    tf.keras.layers.Dense(units = 1)
])
```

.summary() (more)

- Returns a summary of our model and each of its individual layers including output shape

.compile() (more)

- Configures the model to be prepared for training
- This includes defining the optimizer and the loss type
- In our example, we used the adam optimizer (optimizer = 'adam') and MAE for loss (loss = 'mae')

.fit() (more)

- Trains our model in smaller batches that are easier to digest
- First two parameters and our X_train and Y_train data
- Batch_size refers to the sample count of each iteration (default is 32)
- Epochs is the number of iterations that we want our model to run through (each iteration goes through the X and Y data)

```
losses = model.fit(X_train, y_train, validation_data = (X_test, y_test), batch_size = 256, epochs = 15)
#batch_size refers to breaking our total training data into smaller groups and feeding it into the model
```

```
Epoch 1/15
581/581 [==============================] - 2s 2ms/step - loss: 0.0138 - val_loss: 0.0105
Epoch 2/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0107 - val_loss: 0.0104
Epoch 3/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0105 - val_loss: 0.0103
Epoch 4/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0104 - val_loss: 0.0103
Epoch 5/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0104 - val_loss: 0.0102
Epoch 6/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0104 - val_loss: 0.0102
Epoch 7/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0103 - val_loss: 0.0104
Epoch 8/15
581/581 [==============================] - 1s 1ms/step - loss: 0.0102 - val_loss: 0.0103
```

.predict() (more)

- Used to predict trends and outputs once the model is created and trained

.history (more)

- Stores both the training loss and validation loss from each epoch that ran when fitting the model