Data Analyst in Python

**Course: Introduction to Data Science in Python**

**Chapter 1**

**Section 1 - Dive Into Python**

Remember to import the appropriate modules

Alias modules – *import pandas as pd*

**Section 2 - Creating Variables**

Must start with a letter (usually lowercase), then letters, numbers, underscores, but no special characters.

Case-sensitive

**Section 3 - Functions**

Section of code (action) that converts some input to a given output

plt.plot(x-value, y-value, label=’label’)

Common errors: missing commas (syntax), missing parenthesis (syntax)

**Chapter 2**

**Section 1 - pandas**

pandas is a module for working with tabular data

load from multiple sources

search for particular rows or columns

calculate aggregate data

combine data from different sources

DataFrame loading is the first step

Easiest way to create DataFrame is using a .csv file

Always add import pandas as pd

head() function prints first 5 rows

info() method gives a summary of the data you’ve imported

**Section 2 - Selecting Columns**

Calculate data from column… (sum of numbers, etc)

Plot data from columns…

**Section 3 - Selecting Rows with Logic**

Return true or false when checking = = or > or < or >= or !=

= = checks for equality while = sets a value

credit\_records[credit\_records.price > 20.00] >>> selecting rows where the column price is greater than 20.00

Booleans (only two, true and false)

**Chapter 3 - Plotting Data with matplotlib**

**Section 1 - Creating line plots**

Import matplotlib\

from matplotlib import pyplot as plt

plt.plot(dataframe\_x\_values.column\_name, dataframe\_y\_values.column\_name)  
plt.show()

multiple lines use a 2nd, or 3rd, etc. plt.plot statement before plt.show()

**Section 2 - Adding text to plots**

plt.xlabel(“Label for X”)

plt.ylabel(“Label for Y”)

plt.title(“Plot Title”, fontsize=xx) #fontsize is optional

plt.plot(dataframe\_x\_values.column\_name, dataframe\_y\_values.column\_name, label=”Label for Legend”)

plt.legend(color=”green”) #color is optional

plt.text(x\_coord, y\_coord, “Text Message”)

plt.show()

**Section 3 - Styling graphs**

**Chapter 4**

Section 1

Section 2 - Bar Charts

plt.bar(arg1, arg2, yerr=dataframe.column\_for\_error)  
plt.ylabel(“Label”)  
plt.show()

plt.barh() # plot horizontal bars

Stacked Bar chart

plt.bar(x\_column, y\_column1, label=’Label’)  
plt.bar(x\_column, y\_column2, bottom=y\_column1, label=’Label’)  
plt.legend()  
plt.show()

Section 3 - Histograms

plt.hist(dataframe.column, bins=[number of bins], range=(xmin, xmax))

Normalizing

Compare differing sample sizes on portion of total sample (normalized to 1)

Add keyword: density=true

Section 4

**Course: Intermediate Python**

Chapter 1

Section 1

Section 2

Section 3

Section 4

Chapter 2

Section 1

Section 2

Section 3

Section 4

Chapter 3

Section 1

Section 2

Section 3

Section 4

Chapter 4

Section 1

Section 2

Section 3

Section 4

**Assessment: Python Programming**

**Course: Data Manipulation with pandas**

Chapter 1

Section 1

Section 2

**Section 3 - New Columns**

Add new columns when existing data doesn’t meet your needs

Mutate, transform, feature engineering

dataframe[“new\_column”] = dataframe[“existing\_column”] {math}

dataframe[“new\_column”] = dataframe[“column1”] \* dataframe[“column2”]

**Chapter 2 - Aggregating Data**

**Section 1 - Summary Statistics**

df[“column”].mean()

.median(), .mode(), min(), max(), var(), std()

.agg() allows for the use of custom calculations for analysis

df[“column”].agg(function\_name) for single column

df[[“column1”, “column2”]].agg(function\_name) for multiple columns

use .agg() to get multiple summaries: df[“column”].agg([function1\_name, function2\_name])

.cumsum() summation of data moving down the row also: .cummax(), .cummin(),.cumprod()

**Section 2 - Counting**

Dropping duplicates: df.drop\_duplicates(subset=”column”) for a single column

Multiple columns: df.drop\_duplicates(subset=[“column1”, “column2”])

Get counts of unique items: df[“column”].value\_counts()

Sort: df[“column”].value\_counts(sort=True)

Normalize to distribution: df[“column”].value\_counts(normalize=True)

**Section 3 - Grouped Summary Statistics**

df[df[“column”] == “criteria”][“column2”].mean() find average of column2

Group: df.groupby(“grouping\_column”)[“investigative\_column”].mean()

Using agg() for multiple statistics: df.groupby(“group\_column”)[“investigative\_column”].agg([min, max, sum])

Group by multiple columns: df.groupby([“column1”, “column2”])[“investigate\_column”].mean()

Group & aggregate multiples: df. groupby([“column1”, “column2”])[[“investigate\_column”, “i\_column2”]].mean()

**Section 4 - Pivot Tables**

Pivot tables help summarize and group data: df.pivot\_table(values=”column”, index=”column”)

Group by index, values are columns to be summarized

Multiple stats: df.pivot\_table(values=”column”, index=”column”, aggfunc=[np.mean, np.median])

Multiple columns: df.pivot\_table(values=”column”, index=”column”, columns=”column2”)

Replace null/NaN with value: df.pivot\_table(values=”column”, index=”column”, columns=”column2”, fill\_value=0)

Mean: Multiple columns: df.pivot\_table(values=”column”, index=”column”, columns=”column2”, margins=True)

**Chapter 3**

**Section 1 - Explicit Indexes**

df\_ind = df.set\_index(“name”) >>> can accept a list as well > [“index1”, “index2”]

df\_ind.reset\_index() >>> can accept drop=True argument to remove the index altogether

indexes make sub-setting with .loc easier

If multiple indexes, subset outer level using .loc >>> df\_ind.loc[[“index1”, “index2”]]

If multiple levels, pass list of tuples to .loc >>> df\_ind.loc[[(“index1”, “index2”), (“index3”, “index4”)]]

Sort by index using df\_ind.sort\_index()

Control order and level by passing lists >>> df\_ind.sort\_index(level=[“index1”, “index2”], ascending=[True, False])

**Section 2 - Slicing and subsetting with .loc and .iloc**

Slice >>> list[x:y] includes x but not y

Slice dataset: first sort by index, then use .loc >>> use index values >>> list\_sort.loc[“index1”:“index2”]

Slicing inner index levels: list\_sort.loc[(“index1”, “index2”):(“index3”, “index4”)]

Slicing dataset columns: list\_sort[:, “column1”:”column2”]

Slice on rows and columns >>> list\_sort.loc[(“index1”, “index2”):(“index3”, “index4”), “column1”:”column2”]

Slicing dates, pass dates as strings, but can pass partial dates like “2014”

Slicing with iloc uses row/column numbers >>> list.iloc[row\_x:row\_y, column\_a:column\_b]

**Section 3 - Working with pivot tables**

Can use .loc and slicing on pivot tables

Calculate mean by axis: pivot\_table.mean(axis=”index”)

Calculate across columns using pivot\_table.mean(axis=”columns”) >>> every column has the same data type

Section 4

Chapter 4

Section 1

Section 2

Section 3

Section 4

**Course: Joining Data with pandas**

**Assessment: Data Manipulation with Python**

**Course: Introduction to Data Visualization with Matplotlib**