Working with Data with Python

Overview

- Entering your data [lists, dictionaries, series dataframes]
- Reading datasets and cleaning data
- Analyzing and visualizing data

Resources

- O'Reilly Learning Platform: https://databases.lib.wvu.edu/connect/1540334373

Sequence Types (Lists, Tupples, Ranges)

ordered sets that we can enter and retrieve information from.

- Python Documentation - Sequence Types

Lists

Lists are used to store multiple items in a single variable. Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

named_varible = ["item1", "item2", "item3"]

In [ ]:
```
#List integer

my_var = [1, 2, 3, 4]
print(my_var)
type(my_var)
```

In [ ]:
```
#List string

cities = ["Morgantown", "Charleston", "Reedsville", "Huntington"]

print(cities)
type(cities)
```

In [ ]:
```
#Indexing
#method returns the position at the first occurrence of the specified value.
#syntax []

cities [0] #numbered order starts at 0
#force an error messages with print(cities[4])

#cities [0:2] #range
```

In [ ]:
```
#Adding and removing items to a list .append and .remove

cities.append("Logan's Run")
cities

#remove
#cities.remove("Logan's Run")
#cities
```

In [ ]:
```
#What is "in" your list

len(cities)

#"Morgantown" in cities
```

In [ ]:
```
# Define a list of numbers
numbers = [1, 2, 3, 4, 5]

# Loop over each number in the list
for number in numbers:
    # Print each number
    print(number + 1)
```

Functions in Standard Python that are similiar to lists

- Tupples -> A tuple is a collection which is ordered and unchangeable. Tuples are written with round brackets().
  - (_)
- Sets -> A set is a collection which is unordered, unchangeable, and unindexed. Sets are written with curly brackets.
  - { _ }
- Dictionaries -> A dictionary is a collection which is ordered*, changeable and does not allow duplicates. Dictionaries are written with curly brackets, and have keys and values.
  - {key1: value1, key2: value2, key3:value3 }

In [ ]:
```
#dictionaires
#one-domensional key-value list

country_roads = {"Morgantown": 30847, "Charlestown": 45879, "Reedsville": 603, "Huntington": 449
print(country_roads)
#print(country_roads["Reedsville"])
```

In [ ]:
```
#add to dictionary

country_roads["Westover"]  = 4220
country_roads
```

Data science libraries in Python

Listed below are the major libraries that provide built-in functions, methods, and constants that are important for doing data science analysis. Each library has a website with documentation (remember the Python Standard Library) that is great for reference and tutorials.

Storage, Manipulations, Calculations

- Numpy
- Pandas
- Scipy
- StatsModels

Vizualization

- Matplotlib
- Bokeh

Machine Learning

- SciKit
- TensorFlow
- Keras

Pandas

In [ ]:
```
#calling a library

import pandas as pd

#Alias ->
#you can call libraries as aliases using "as". This will allow you to simplify your code and the
#need to do.
```

Series in Pandas

Like a dictionary in the standard library, a series allows you to store key-value pairs in python.

- Pandas Documentation - Series

In [ ]:
```
#series - A Pandas Series is like a column in a table. It is a one-dimensional array holding data

wvu_towns = pd.Series(country_roads)

wvu_towns
```

In [ ]:
```
series_example = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
series_example
```

Data Frames

Dataframes are 2 dimensional data structures containing key-value pairs.

In [ ]:
```python
import pandas as pd

# Creating individual Series for each column
year_series = pd.Series([1977, 1980, 1983], name="Year")
title_series = pd.Series(["Star Wars", "Empire Strikes Back", "Return of the Jedi"], name="Title
length_series = pd.Series([121, 124, 144], name="Length")
gross_series = pd.Series([787, 534, 572], name="Gross")


# Displaying the DataFrame
starwars_df
```

In [ ]:
```python
# Combining the Series into a DataFrame

starwars_df = pd.DataFrame({
    "Year": year_series,
    "Title": title_series,
    "Length": length_series,
    "Gross": gross_series
})
```

In [ ]:
```python
#subsetting

starwars_df['Title']
```

In [ ]:
```python
# data values

starwars_df.dtypes
```

In [ ]:
```python
# amending data values

starwars_df['Year'] = starwars_df['Year'].astype(str)

starwars_df.dtypes
```

In [ ]:
```python
# retrieve descriptive statistics

# starwars_reviews.mean()

starwars_df["Length"].mean()
```

Reading Data

Read Options

- read_csv -- Load delimited data from a file, URL, or file-like object; use comma as default delimiter
- read_fwf -- Read data in fixed-width column format (i.e., no delimiters)
- read_excel -- Read tabular data from an Excel XLS or XLSX file
- read_html -- Read all tables found in the given HTML document

- read_json -- Read data from a JSON (JavaScript Object Notation) string representation, file, URL, or file-like object
- read_sas -- Read a SAS dataset stored in one of the SAS system's custom storage formats
- read_spss -- Read a data file created by SPSS
- read_stata -- Read a dataset from Stata file format
- read_xml -- Read a table of data from an XML file

In [ ]:
```
# read spss file

import pandas as pd

demographics = pd.read_spss("demographics.sav")
demographics

# may need to install pyreadstat - pip install pyreadstat
```

In [ ]:
```
# read a csv file

import pandas as pd

reviews = pd.read_csv("customer_reviews.csv")
%whos
```

Exploring the Dataframe

In [ ]:
```
#Viewing dataframes .head() . tail()

reviews.head() #default is 5

#reviews.tail(10) #change the number displayed
```

In [ ]:
```
#viewing data values in dataframe

reviews.dtypes
```

In [ ]:
```
#look as basic information about the dataframe with info()

reviews.info()
```

In [ ]:
```
#viewing observations and variables numbers

reviews.shape
```

In [ ]:
```
#viewing the names of variables

reviews.columns
```

In [ ]:
```
# get basic descriptive statistics

reviews.describe()

#reviews.mean()
```

```
#reviews.median()
#reviews.mode()
```

Aggregation	Returns
count	Total number of items
first, last	First and last item
mean, median	Mean and median
min, max	Minimum and maximum
std, var	Standard deviation and variance
mad	Mean absolute deviation
prod	Product of all items
sum	Sum of all items

In [ ]:
```
# viewing the values counts of observations in a variable

reviews["Class_Name"].value_counts()
```

Cleaning Data

Why would you need to clean data

- Data in columns and rows are not ordered in the correct way
- Creating values or ignoring missing data
- Units are not correct or are wrong in some way
- Order of magnitude is off
- Outliers and skewing of the data

Missing Values

In [ ]:
```
#check for missing data

missing_values = reviews.isna()
missing_values
```

In [ ]:
```
# number of missing data

number_missing = reviews.isna().sum()
number_missing
```

In [ ]:
```
#Get a report

# Check for missing values
print("Missing values (True indicates missing):")
print(reviews.isna())

# Count missing values in each column
```

```
print("\nCount of missing values per column:")
print(reviews.isna().sum())

# Get a summary of the DataFrame
print("\nDataFrame info:")
reviews.info()

# Check if there are any missing values in the entire DataFrame
print("\nAre there any missing values in the entire DataFrame?")
print(reviews.isna().any().any())

# Check if there are any missing values in each column
print("\nAre there any missing values in each column?")
print(reviews.isna().any())

# Check for non-missing values
print("\nNon-missing values (True indicates non-missing):")
print(reviews.notna())
```

In [ ]:
```
# remove missing values

reviews.shape

#remove all observations with na

reviews_na = reviews.dropna()
reviews_na.shape

# remove
```

In [ ]:
```
# replace missing values

#replace na values

reviews["Title"] = reviews["Title"].fillna("None Given")
reviews["Title"]
```

In [ ]:
```
#get value counts for a variable

reviews["Title"].value_counts()
```

Change Data Values

In [ ]:
```
#view the dataframe

reviews
```

In [ ]:
```
# view types of data values in the dataframe

reviews.dtypes
```

In [ ]:
```
# change the values

# Clothing ID should be a string
reviews['Clothing_ID'] = reviews['Clothing_ID'].astype("str")
```

```
# Recommended_IND, Division_Name, Department_Name, and Class_Name should be a category
reviews[['Recommended_IND', 'Division_Name', 'Department_Name', 'Class_Name']] = reviews[['Recom

# view the values

reviews.dtypes
```

In [ ]:
```
reviews["Division_Name"].value_counts()
```

Datetime Values

In [ ]:
```
import pandas as pd

# load scotus approval dataframe

scotus = pd.read_csv("scotus_approval.csv")
scotus
```

In [ ]:
```
scotus["date"] = pd.to_datetime(scotus["end_date"])
scotus.dtypes
```

Boolean Operators

Use comparison operators to determine to filter observations in a variable.

- Equal (==)
- Not equal (!=)
- Greater than (>)
- Less than (<)
- Greater than or equal (>=)
- Less than or equal (<=)

Filter Variables

Filter out one or more observations from a variable using boolean operators to set criterias.

In [ ]:
```
# get mean of ratings

reviews["Rating"].mean()
```

In [ ]:
```
#filter observations using boolean operators

#find the mean of rating that people who bought from the General Department

reviews_filter = reviews[reviews["Division_Name"]== "General"]
reviews_filter["Division_Name"].value_counts()
```

In [ ]:
```
reviews_filter["Rating"].mean()
```

In [ ]:
```
reviews["Class_Name"].value_counts()
```

In [ ]:
```
#filter observations in mulitple variables

reviews_filter_2 = reviews[(reviews["Division_Name"] == "General") & (reviews["Class_Name"] != "
reviews_filter_2["Rating"].mean()
```

Select Variables

Select or remove variables from the dataframe

In [ ]:
```
scotus.columns
```

In [ ]:
```
# select columns from the dataframe

scotus = scotus[['poll_id',  'pollster', 'yes', 'no', 'alternate_answers', 'date']]
scotus
```

In [ ]:
```
scotus.columns
```

In [ ]:
```
# removed columns from the dataframe

scotus = scotus.drop(columns = ['alternate_answers'])
scotus
```

Sample

Filter random selections from the dataframe

In [ ]:
```
# number

reviews_sample = reviews.sample(n=500)
reviews_sample.describe()
```

In [ ]:
```
# fraction

reviews_sample = reviews.sample(frac=0.25, replace = True, random_state=1)
reviews_sample.describe()
```

Assign

Create new columns or modify existing ones in a dataframe.

In [ ]:
```
demographics
```

In [ ]:
```
# lets create a new column that divides the income variable by 1000

demographics = pd.read_spss("demographics.sav")

demographics = demographics.assign(income = (demographics["income"]/1000))
demographics
```

Recode Variables

Transform the values of a variable into new values based on specific criteria

In [ ]:
```
reviews["Recommended_IND"].value_counts()
```

In [ ]:
```
#overwrite the variable

reviews_recode = reviews["Recommended_IND"].replace([1, 0], ["Yes", "No"])
reviews_recode
```

In [ ]:
```
# create a new variable

reviews = reviews.assign(Recommended_recode = reviews["Recommended_IND"].map({0: "No", 1: "Yes"}
reviews

# assign creates and new variable and # map sets the new values
```

In [ ]:
```
reviews["Recommended_recode"].value_counts()
```

Rename Variables

Rename the column

In [ ]:
```
reviews = reviews.rename(columns={"Recommended_IND": "Recommended_num", "Recommended_recode":"Re
reviews
```

Relocate Variables

Move the column location in the dataframe

In [ ]:
```
reviews.columns
```

In [ ]:
```
new_order = ['Clothing_ID', 'Age', 'Title',
             'Review_Text', 'Rating', 'Recommended_num', 'Recommended_label',
             'Positive_Feedback_Count', 'Division_Name',
             'Department_Name', 'Class_Name']

reviews = reviews[new_order]
reviews
```

Sort Variables

Rearrange the observations in a column

In [ ]:
```
reviews.sort_values(by="Rating", ascending=False)
```

Bin Observations

Group observations into 'bins' or categories based on their values.

In [ ]:
```
#bin values
```

```
reviews = reviews.assign(Age_Category = lambda x: pd.cut(x['Age'], bins = [18, 35, 65, float('in
reviews["Age_Category"].value_counts()

# We use the assign method to add a new column "Age_Category" to the DataFrame.
# The lambda function passed to assign calculates the age category based on the "Age" column usi
# The pd.cut function bins the ages into categories defined by the specified bins and assigns co
```

Pivot Table

Allows you to summarize and analyze the data by aggregating values across different dimensions, such as rows and columns,

In [ ]:
```
reviews.dtypes
```

In [ ]:
```
reviews_pivot = reviews.pivot_table(
    values = "Rating",
    index = "Recommended_label",
    columns="Class_Name",
    aggfunc = "mean",
    fill_value = 0
)

reviews_pivot
```

Groupby

Groups your observations in one or more varaiables.

In [ ]:
```
#Groupby

class_rating = reviews[["Rating", "Class_Name"]].groupby("Class_Name", observed=True).mean()
class_rating


#class_rating = reviews[["Rating", "Class_Name"]].groupby("Class_Name", observed=True).describe(
#class_rating
```

Aggregate

Combine multiple data values into a single summary statistic. For example, finding the sum, mean, median, minimum, maximum, or standard deviation of a group

In [ ]:
```
result = reviews.groupby('Recommended_label', observed=True)[['Rating', 'Positive_Feedback_Count
result
```

Dummy Variables

Turns categories into numbers so we can use them in analyses.

In [ ]:
```
reviews_dummies = pd.get_dummies(reviews["Division_Name"], prefix = "Division")
```

```
reviews_dummies
```

In [ ]:
```
# Convert to 1 and 0

reviews_dummies = reviews_dummies.astype(int)
reviews_dummies.mean()
```

Concat Dataframes

Append dataframes with matching observations.

In [ ]:
```
reviews_combined = pd.concat([reviews, reviews_dummies], axis=1)
reviews_combined
```

Model Building with Dummies

- Use categorical data in machine learning and statistical models.
- Avoid implying ordinal relationships in categorical data.
- Capture categorical information accurately.
- Enhance flexibility in model building and analysis.

In [ ]:
```
reviews_combined.columns
```

In [ ]:
```
from sklearn.linear_model import LinearRegression

# Define the features (dummy variables) and target (Sales)
X = reviews_combined[['Division_General', 'Division_General Petite', 'Division_Initmates']]
y = reviews_combined["Rating"]

# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Print the coefficients
print("\nCoefficients of the regression model:")
print(model.coef_)
print("Intercept:", model.intercept_)
```

Export Dataframes

In [ ]:
```
#Write

reviews.to_csv("cleaned_reviews.csv")
reviews_pivot.to_csv("pivot_reviews.csv")
class_rating.to_csv("class_ratings.csv")
scotus.to_csv("scotus.csv")
```