Emily Lai
Magnestar Coding Challenge

Challenge

Given a supplied data file containing gathered satellite information and telemetry for 333 active space satellites, we ask that you produce a script that:

1. Reads in the satellite data CSV file
2. Optionally filters the data by either Operator and/or Frequency Band
3. Output subset of columns for the filtered results

Instructions to Run:

The NumPy and Pandas libraries must be installed to run this code. The satellites.csv file must also be in the same directory as the code.

This coding challenge is submitted as both a Jupyter Notebook .ipynb file and a Python .py file. Each cell of the Jupyter Notebook should be run in order, by pressing either Ctrl-Enter to run an individual cell, or Shift-Enter to run a cell and select the one below it. User input will be required for filtering of the operator and frequency bands, however they can also be left blank for no filter.

The Python .py file can be run on any Python-compatible IDE.

Approach

A short demo of the code can be found here: https://youtu.be/NytHLgfUcfk
This challenge was completed in the Python coding language.

```
import numpy as np
import pandas as pd
```

Due to the nature of this challenge in handling data, the numpy and pandas libraries are imported.

Requirement 1: Reads in satellite data CSV file

```
satellites = pd.read_csv("satellites.csv")
```

The provided satellites.csv file is then read into a Pandas dataframe using the pandas pd.read_csv() function.

Requirement 2: Filters Data by Operator and/or Frequency Band

As according to user input:

```
# Parse user input
# Operators: will search for user input as substring within each Operator, case sensitive
# Frequency Band: will find satellites with user input frequency bands in any order, case sensitive
# Case sensitive
user_op = input("Please enter operator or leave blank for none: ")
user_freq_band = input("Please enter frequency bands separated by / or leave blank for none: ").split("/")
```

The next section of code takes in the user input for the operator and the frequency bands.

The user's input for the operator will be stored in "user_op". When searching, the program will search for the user's input as a substring within each operator. It is also case sensitive.
For instance, the user input "RASC" would return rows with operators that include the string "RASC", such as "RASC (Radio Amatuer Satellite Corporation)" seen in the dataframe.

The user's input for the frequency band is stored in "user_freq_band". The program will find the satellites with the user frequency bands in any order. This input is also case-sensitive; for instance, the frequency band "Ku" will not be retrieved if the user enters "KU". This is a consideration for potential business requirements, in case different capitalizations correlate to different frequency bands. However, in the event that the program should be case-insensitive, this is possible by passing all entries through the .upper() or .lower() Python function. The value inputted is split on the forward slash "/". The entered values are stored in an array.

Sample Output:

```
In [214]: # Parse user input
          # Operators: will search for user input as substring within each Operator, case sensitive
          # Frequency Band: will find satellites with user input frequency bands in any order, case sensitive
          # Case sensitive
          user_op = input("Please enter operator or leave blank for none: ")
          user_freq_band = input("Please enter frequency bands separated by / or leave blank for none: ").spl

          Please enter operator or leave blank for none: RASC
          Please enter frequency bands separated by / or leave blank for none: VHF
```

Filter on Operator Input:

```
# Filter if user has entered an operator, otherwise no filter
if user_op != "":
      satellites = satellites[satellites.apply(lambda x: user_op in str(x["Operator"]), axis=1)]
```

The following code filters on the operator input. If the user input is blank, then no filter will be applied and all operators will be considered. If the user has entered an operator, then a lambda

function is applied on the satellites dataframe.

A lambda function is applied to check for the user input “user_op” in the “Operator” column, converted to string value. The reason for the string conversion is that null values appear as NaN, which is considered a float value. The axis value is set to 1 instead of the default 0, so that the program will look through column titles instead of row titles.

This code will replace the satellites dataframe with a dataframe containing only the relevant operators.

Function to filter on Frequency Band Input:

```
def can_create(user_freq_band, df_freq_band):
      if type(df_freq_band) is float:
      return False

      df_freq_band = df_freq_band.split('/')
      # Convert both to sets to remove edge case of duplicates
      user_set = set(user_freq_band)
      df_set = set(df_freq_band)


      for band in user_set:
      if band not in df_set:
            return False
      return True
```

The can_create function is defined here. This function is used in the next section of code to filter the frequency bands. If this function returns True, the row with the frequency band(s) in question will remain in the dataframe. If the function returns False, the row will be removed from the dataframe.

The function takes in two inputs, “user_freq_band” as the frequency bands previously inputted by the user, and “df_freq_band” as the frequency bands in the row of the dataframe that is being searched.

The first section of the function:

```
      if type(df_freq_band) is float:
      return False
```

This statement checks if the datatype of the value in the Frequency Band column of the dataframe is a float. If it is a float, that means it is a null value (NaN) and will return False, therefore removing this row from the dataframe.

The second section of the function:

```
        df_freq_band = df_freq_band.split('/')
        # Convert both to sets to remove edge case of duplicates
        user_set = set(user_freq_band)
        df_set = set(df_freq_band)
```

This statement splits the value in the Frequency Band column of the dataframe (“df_freq_band”) on the forward slash “/”. Thus, the frequency band values will be stored in an array. This was previously done with the “user_freq_band” input, thus allowing these values to be separated and compared in any order.

To remove any duplicate values entered by the user, both arrays “user_freq_band” and “df_freq_band” are converted to a set. Since the set data structure does not include duplicates, any repeated values will be filtered out. The frequency bands inputted by the user are now “user_set” and the frequency bands in the dataframe are “df_set”.

The final part of this function:

```
        for band in user_set:
            if band not in df_set:
                return False
        return True
```

For each band listed in the “user_set” set, this statement checks whether the band is in the “df_set” set. If the band is not in the “df_set”, that means it does not meet the filter requirements, and False will be returned, removing this row from the dataframe.

If a row has passed all parts of this function, then the filters have all been applied and True is returned. The row will remain in the dataframe and will be outputted in the final output.

Filter on Frequency Band Input:

```
# Filter if user has entered a value for frequency band, otherwise no
filter
if user_freq_band != ['']:
        satellites = satellites[satellites.apply(lambda x:
can_create(user_freq_band, x["Frequency Band"]), axis=1)]
```

The following code filters on the frequency band input. If the user input is blank, then no filter will be applied and all frequency bands will be considered. If the user has entered a frequency band, then a lambda function is applied on the satellites dataframe.

A lambda function is applied to check for the “user_freq_band” values in the “Frequency Band” column. The can_create function defined above is employed to filter each row of the dataframe. The axis value is set to 1 instead of the default 0, so that the program will look through column titles instead of row titles.

This code will replace the satellites dataframe with a dataframe now containing only the rows with both the relevant operators and frequency bands.

Requirement 3: Output subset of columns for the filtered results

```
satellites[['sat name', 'NORAD', 'Operator', 'Frequency Band']]
```

This statement outputs only the required columns: “sat name”, “NORAD”, “Operator”, and “Frequency Band”.

Sample Output:

```
In [218]: satellites[['sat name', 'NORAD', 'Operator', 'Frequency Band']]
```

Out[218]:

	sat name	NORAD	Operator	Frequency Band
11	OSCAR 7 (AO-7)	7530	RASC (Radio Amatuer Satellite Corporation)	HF/VHF
15	UOSAT 2 (UO-11)	14781	RASC (Radio Amatuer Satellite Corporation)	VHF/UHF
21	LUSAT (LO-19)	20442	RASC (Radio Amatuer Satellite Corporation)	VHF/UHF

(Filtered on Operator “RASC” and Frequency Band “VHF”)

A short demo of the code can be found here: https://youtu.be/NytHLgfUcfk