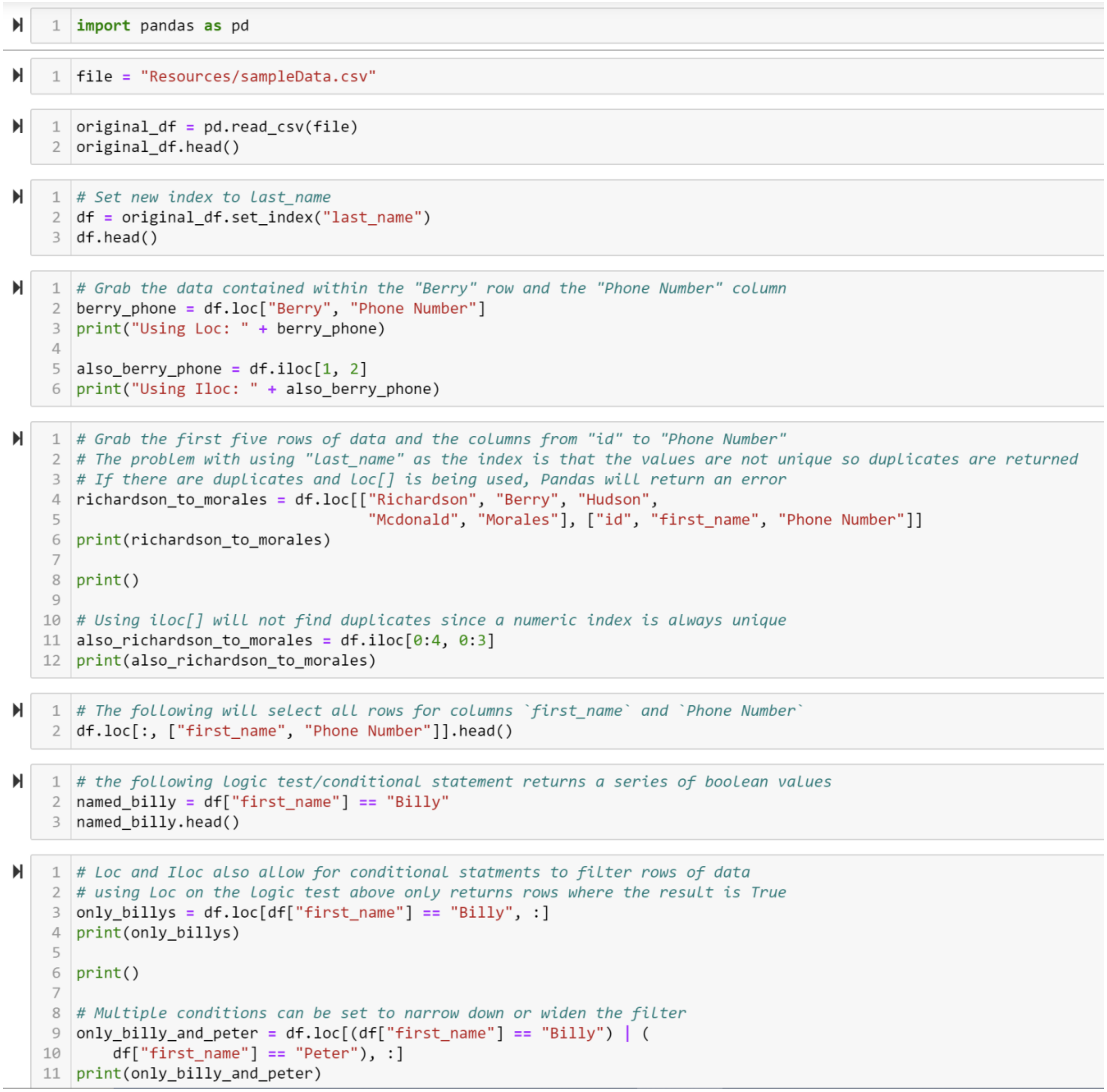
LESSON 4.2

SECTION-1: loc and iloc



```python
import pandas as pd
```

```python
file = "Resources/sampleData.csv"
```

```python
original_df = pd.read_csv(file)
original_df.head()
```

```python
# Set new index to last_name
df = original_df.set_index("last_name")
df.head()
```

```python
# Grab the data contained within the "Berry" row and the "Phone Number" column
berry_phone = df.loc["Berry", "Phone Number"]
print("Using Loc: " + berry_phone)

also_berry_phone = df.iloc[1, 2]
print("Using Iloc: " + also_berry_phone)
```

```python
# Grab the first five rows of data and the columns from "id" to "Phone Number"
# The problem with using "last_name" as the index is that the values are not unique so duplicates are returned
# If there are duplicates and loc[] is being used, Pandas will return an error
richardson_to_morales = df.loc[["Richardson", "Berry", "Hudson",
                                "Mcdonald", "Morales"], ["id", "first_name", "Phone Number"]]
print(richardson_to_morales)

print()

# Using iloc[] will not find duplicates since a numeric index is always unique
also_richardson_to_morales = df.iloc[0:4, 0:3]
print(also_richardson_to_morales)
```

```python
# The following will select all rows for columns `first_name` and `Phone Number`
df.loc[:, ["first_name", "Phone Number"]].head()
```

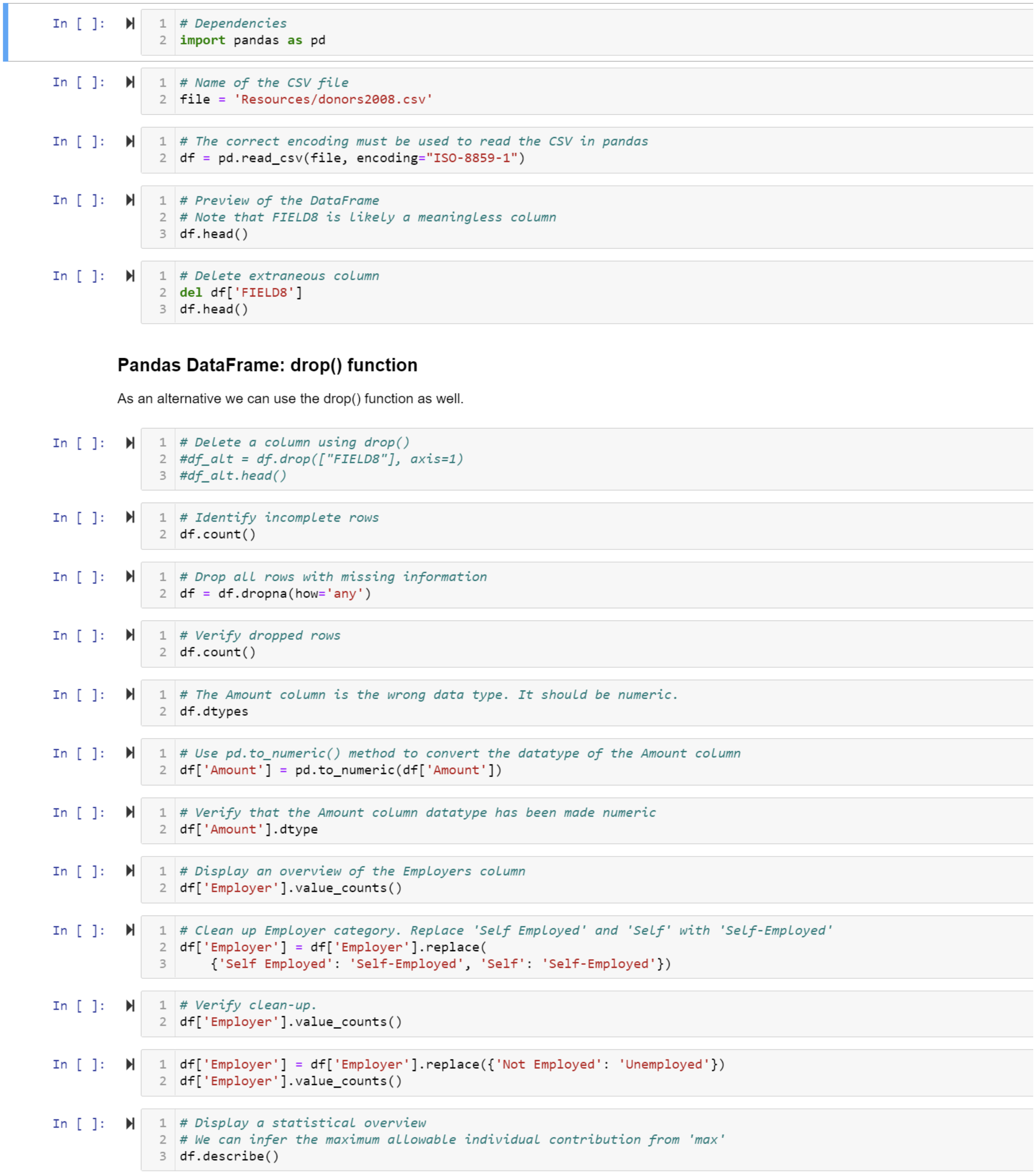
```python
# the following logic test/conditional statement returns a series of boolean values
named_billy = df["first_name"] == "Billy"
named_billy.head()
```

```python
# Loc and Iloc also allow for conditional statments to filter rows of data
# using Loc on the logic test above only returns rows where the result is True
only_billys = df.loc[df["first_name"] == "Billy", :]
print(only_billys)

print()

# Multiple conditions can be set to narrow down or widen the filter
only_billy_and_peter = df.loc[(df["first_name"] == "Billy") | (
    df["first_name"] == "Peter"), :]
print(only_billy_and_peter)
```

SECTION-2: DATA CLEANING



In [ ]:

```
# Dependencies
import pandas as pd
```

In [ ]:

```
# Name of the CSV file
file = 'Resources/donors2008.csv'
```

In [ ]:

```
# The correct encoding must be used to read the CSV in pandas
df = pd.read_csv(file, encoding="ISO-8859-1")
```

In [ ]:

```
# Preview of the DataFrame
# Note that FIELD8 is likely a meaningless column
df.head()
```

In [ ]:

```
# Delete extraneous column
del df['FIELD8']
df.head()
```

Pandas DataFrame: drop() function

As an alternative we can use the drop() function as well.

In [ ]:

```
# Delete a column using drop()
#df_alt = df.drop(["FIELD8"], axis=1)
#df_alt.head()
```

In [ ]:

```
# Identify incomplete rows
df.count()
```

In [ ]:

```
# Drop all rows with missing information
df = df.dropna(how='any')
```

In [ ]:

```
# Verify dropped rows
df.count()
```

In [ ]:

```
# The Amount column is the wrong data type. It should be numeric.
df.dtypes
```

In [ ]:

```
# Use pd.to_numeric() method to convert the datatype of the Amount column
df['Amount'] = pd.to_numeric(df['Amount'])
```

In [ ]:

```
# Verify that the Amount column datatype has been made numeric
df['Amount'].dtype
```

In [ ]:

```
# Display an overview of the Employers column
df['Employer'].value_counts()
```

In [ ]:

```
# Clean up Employer category. Replace 'Self Employed' and 'Self' with 'Self-Employed'
df['Employer'] = df['Employer'].replace(
    {'Self Employed': 'Self-Employed', 'Self': 'Self-Employed'})
```

In [ ]:

```
# Verify clean-up.
df['Employer'].value_counts()
```

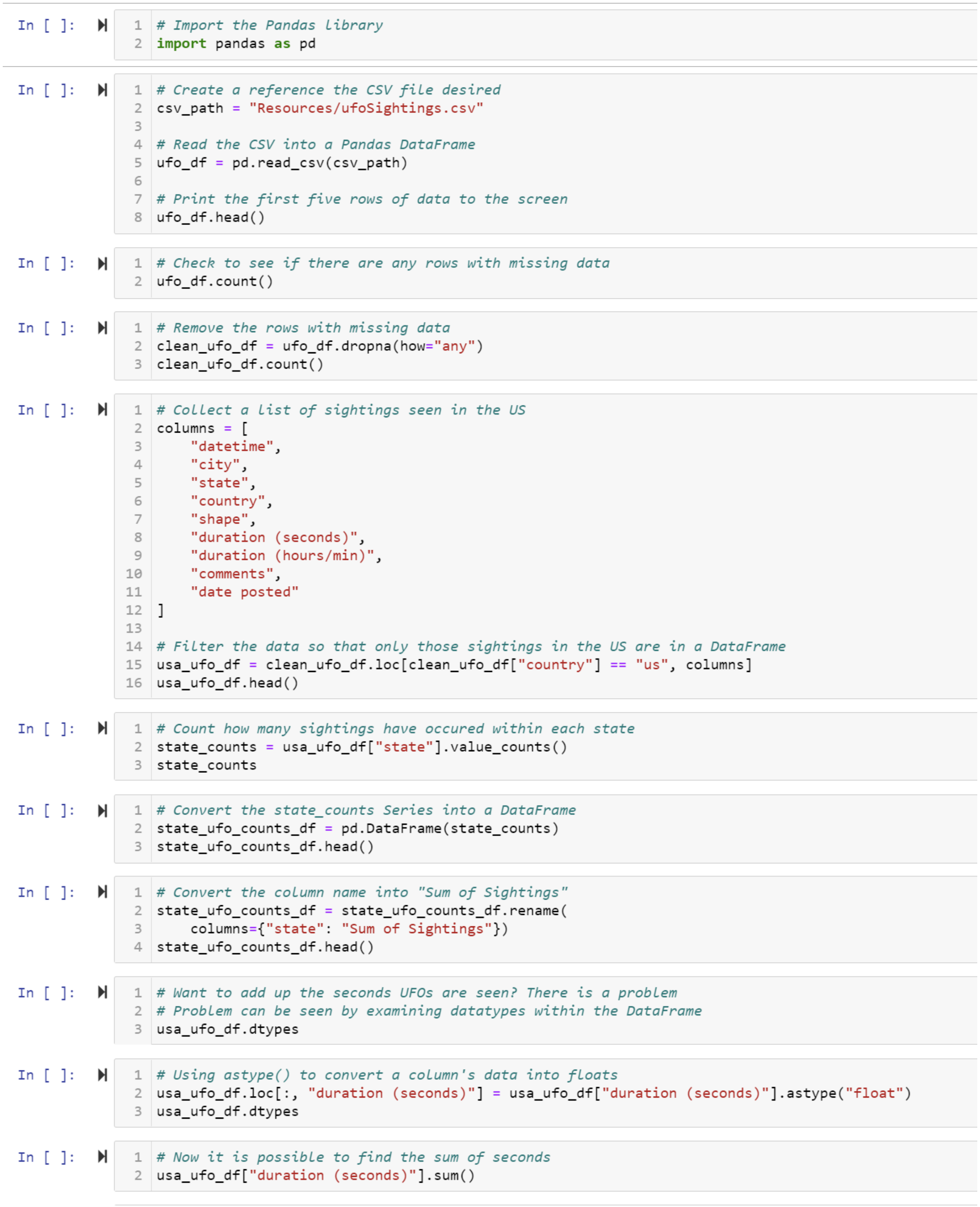
In [ ]:

```
df['Employer'] = df['Employer'].replace({'Not Employed': 'Unemployed'})
df['Employer'].value_counts()
```

In [ ]:

```
# Display a statistical overview
# We can infer the maximum allowable individual contribution from 'max'
df.describe()
```

PANDAS RECAP



In [ ]:

```python
# Import the Pandas library
import pandas as pd
```

In [ ]:

```python
# Create a reference the CSV file desired
csv_path = "Resources/ufoSightings.csv"

# Read the CSV into a Pandas DataFrame
ufo_df = pd.read_csv(csv_path)

# Print the first five rows of data to the screen
ufo_df.head()
```

In [ ]:

```python
# Check to see if there are any rows with missing data
ufo_df.count()
```

In [ ]:

```python
# Remove the rows with missing data
clean_ufo_df = ufo_df.dropna(how="any")
clean_ufo_df.count()
```

In [ ]:

```python
# Collect a list of sightings seen in the US
columns = [
    "datetime",
    "city",
    "state",
    "country",
    "shape",
    "duration (seconds)",
    "duration (hours/min)",
    "comments",
    "date posted"
]

# Filter the data so that only those sightings in the US are in a DataFrame
usa_ufo_df = clean_ufo_df.loc[clean_ufo_df["country"] == "us", columns]
usa_ufo_df.head()
```

In [ ]:

```python
# Count how many sightings have occured within each state
state_counts = usa_ufo_df["state"].value_counts()
state_counts
```

In [ ]:

```python
# Convert the state_counts Series into a DataFrame
state_ufo_counts_df = pd.DataFrame(state_counts)
state_ufo_counts_df.head()
```

In [ ]:

```python
# Convert the column name into "Sum of Sightings"
state_ufo_counts_df = state_ufo_counts_df.rename(
    columns={"state": "Sum of Sightings"})
state_ufo_counts_df.head()
```

In [ ]:

```python
# Want to add up the seconds UFOs are seen? There is a problem
# Problem can be seen by examining datatypes within the DataFrame
usa_ufo_df.dtypes
```

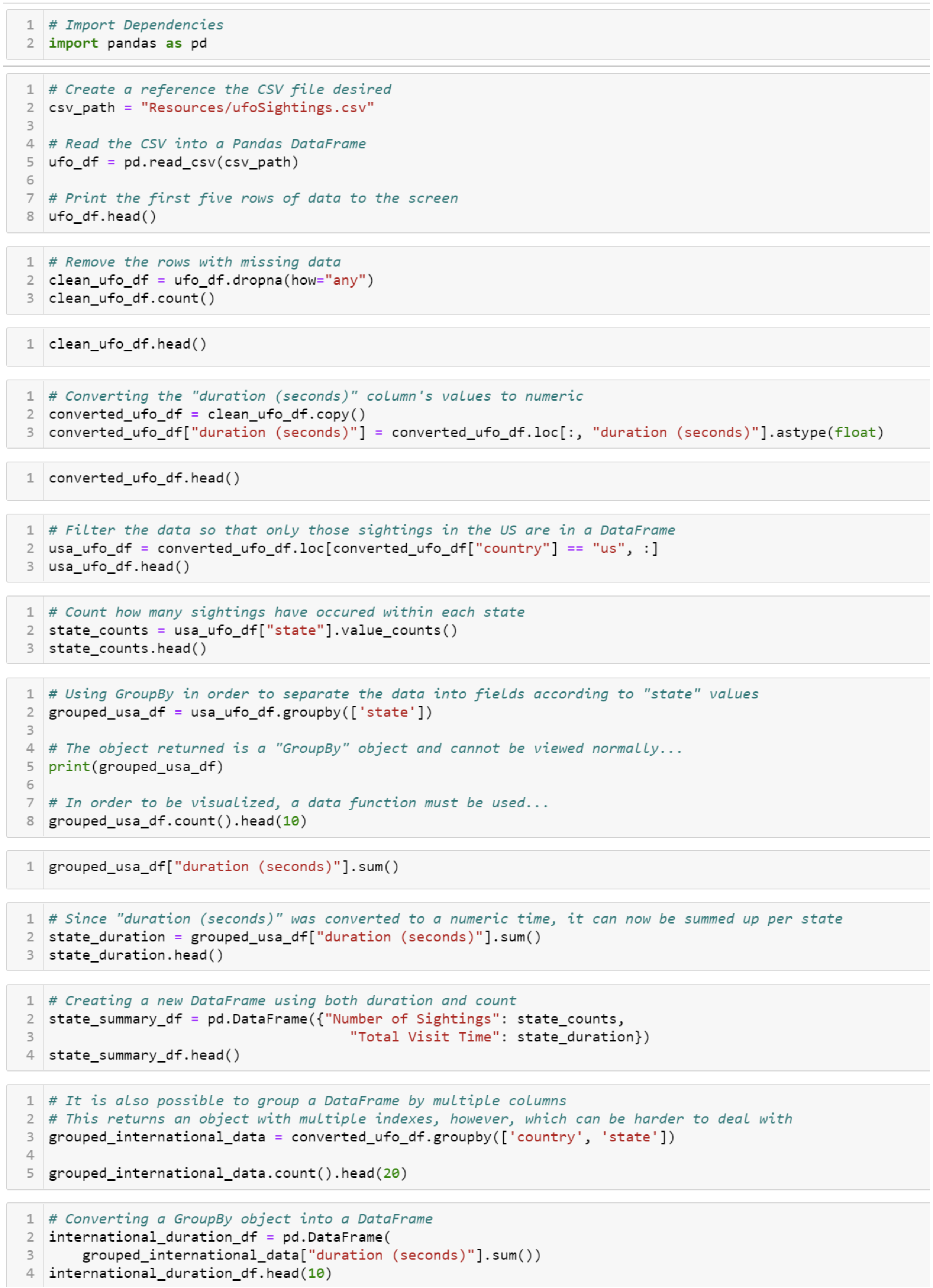
In [ ]:

```python
# Using astype() to convert a column's data into floats
usa_ufo_df.loc[:, "duration (seconds)"] = usa_ufo_df["duration (seconds)"].astype("float")
usa_ufo_df.dtypes
```

In [ ]:

```python
# Now it is possible to find the sum of seconds
usa_ufo_df["duration (seconds)"].sum()
```

SECTION-3: GROUPBY



```python
# Import Dependencies
import pandas as pd
```

```python
# Create a reference the CSV file desired
csv_path = "Resources/ufoSightings.csv"

# Read the CSV into a Pandas DataFrame
ufo_df = pd.read_csv(csv_path)

# Print the first five rows of data to the screen
ufo_df.head()
```

```python
# Remove the rows with missing data
clean_ufo_df = ufo_df.dropna(how="any")
clean_ufo_df.count()
```

```python
clean_ufo_df.head()
```

```python
# Converting the "duration (seconds)" column's values to numeric
converted_ufo_df = clean_ufo_df.copy()
converted_ufo_df["duration (seconds)"] = converted_ufo_df.loc[:, "duration (seconds)"].astype(float)
```

```python
converted_ufo_df.head()
```

```python
# Filter the data so that only those sightings in the US are in a DataFrame
usa_ufo_df = converted_ufo_df.loc[converted_ufo_df["country"] == "us", :]
usa_ufo_df.head()
```

```python
# Count how many sightings have occured within each state
state_counts = usa_ufo_df["state"].value_counts()
state_counts.head()
```

```python
# Using GroupBy in order to separate the data into fields according to "state" values
grouped_usa_df = usa_ufo_df.groupby(['state'])

# The object returned is a "GroupBy" object and cannot be viewed normally...
print(grouped_usa_df)

# In order to be visualized, a data function must be used...
grouped_usa_df.count().head(10)
```

```python
grouped_usa_df["duration (seconds)"].sum()
```

```python
# Since "duration (seconds)" was converted to a numeric time, it can now be summed up per state
state_duration = grouped_usa_df["duration (seconds)"].sum()
state_duration.head()
```

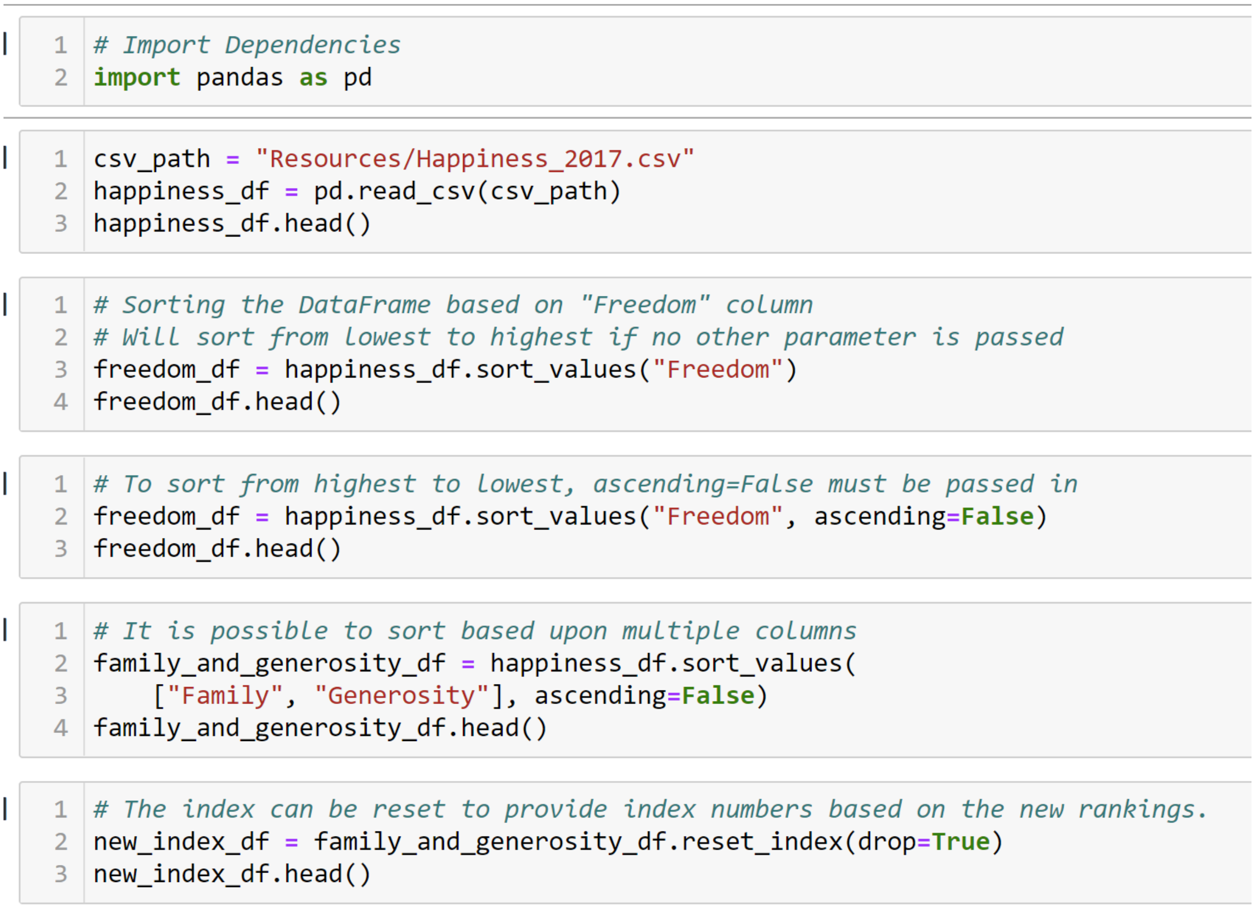
```python
# Creating a new DataFrame using both duration and count
state_summary_df = pd.DataFrame({"Number of Sightings": state_counts,
                                    "Total Visit Time": state_duration})
state_summary_df.head()
```

```python
# It is also possible to group a DataFrame by multiple columns
# This returns an object with multiple indexes, however, which can be harder to deal with
grouped_international_data = converted_ufo_df.groupby(['country', 'state'])

grouped_international_data.count().head(20)
```

```python
# Converting a GroupBy object into a DataFrame
international_duration_df = pd.DataFrame(
    grouped_international_data["duration (seconds)"].sum())
international_duration_df.head(10)
```

SECTION 4: SORTING



```python
# Import Dependencies
import pandas as pd
```

```python
csv_path = "Resources/Happiness_2017.csv"
happiness_df = pd.read_csv(csv_path)
happiness_df.head()
```

```python
# Sorting the DataFrame based on "Freedom" column
# Will sort from lowest to highest if no other parameter is passed
freedom_df = happiness_df.sort_values("Freedom")
freedom_df.head()
```

```python
# To sort from highest to lowest, ascending=False must be passed in
freedom_df = happiness_df.sort_values("Freedom", ascending=False)
freedom_df.head()
```

```python
# It is possible to sort based upon multiple columns
family_and_generosity_df = happiness_df.sort_values(
    ["Family", "Generosity"], ascending=False)
family_and_generosity_df.head()
```

```python
# The index can be reset to provide index numbers based on the new rankings.
new_index_df = family_and_generosity_df.reset_index(drop=True)
new_index_df.head()
```