index

February 25, 2022

1 Using SQL with Pandas - Lab

1.1 Introduction

In this lab, you will practice using SQL statements and the `.query()` method provided by Pandas to manipulate datasets.

1.2 Objectives

You will be able to:

- Compare accessing data in a DataFrame using query methods and conditional logic
- Query DataFrames with SQL using the `pandasql` library

1.3 The Dataset

In this lab, we will continue working with the *Titanic Survivors* dataset.

Begin by importing `pandas` as `pd`, `numpy` as `np`, and `matplotlib.pyplot` as `plt`, and set the appropriate alias for each. Additionally, set `%matplotlib inline`.

[1]:

```
# Your code here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Next, read in the data from `titanic.csv` and store it as a DataFrame in `df`. Display the `.head()` to ensure that everything loaded correctly.

[2]:

```
df = pd.read_csv("titanic.csv")
df.head()
```

[2]:

```
   Unnamed: 0  PassengerId  Survived Pclass  \
0           0            1         0      3
1           1            2         1      1
2           2            3         1      3
3           3            4         1      1
4           4            5         0      3
```

```
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

1.4 Slicing DataFrames Using Conditional Logic

One of the most common ways to query data with pandas is to simply slice the DataFrame so that the object returned contains only the data you're interested in.

In the cell below, slice the DataFrame so that it only contains passengers with 2nd or 3rd class tickets (denoted by the `Pclass` column).

Be sure to preview values first to ensure proper encoding when slicing

- ***Hint***: Remember, your conditional logic must be passed into the slicing operator to return a slice of the DataFrame–otherwise, it will just return a table of boolean values based on the conditional statement!

```
[3]: # Preview values first to ensure proper encoding when slicing
     df["Pclass"].unique()
```

```
[3]: array(['3', '1', '2', '?'], dtype=object)
```

```
[4]: no_first_class_df = df[(df["Pclass"] == "2") | (df["Pclass"] == "3")]
     no_first_class_df.head()
```

```
[4]:    Unnamed: 0  PassengerId  Survived Pclass                            Name  \
     0           0            1         0      3         Braund, Mr. Owen Harris
     2           2            3         1      3          Heikkinen, Miss. Laina
     4           4            5         0      3        Allen, Mr. William Henry
     5           5            6         0      3                Moran, Mr. James
     7           7            8         0      3  Palsson, Master. Gosta Leonard

           Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
     0    male  22.0      1      0         A/5 21171   7.2500   NaN        S
     2  female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
     4    male  35.0      0      0            373450   8.0500   NaN        S
     5    male   NaN      0      0            330877   8.4583   NaN        Q
     7    male   2.0      3      1            349909  21.0750   NaN        S
```

```
[51]: # From GitHub
      no_first_class_df = df[df["Pclass"].isin(["2","3"])]
      no_first_class_df
```

```
[51]:      Unnamed: 0  PassengerId  Survived Pclass  \
      0             0            1         0      3
      2             2            3         1      3
      4             4            5         0      3
      5             5            6         0      3
      7             7            8         0      3
      ..          …          …       …    …
      883         883          884         0      2
      884         884          885         0      3
      885         885          886         0      3
      886         886          887         0      2
      890         890          891         0      3

                                         Name     Sex   Age  SibSp  Parch  \
      0                 Braund, Mr. Owen Harris    male  22.0      1      0
      2                  Heikkinen, Miss. Laina  female  26.0      0      0
      4                Allen, Mr. William Henry    male  35.0      0      0
      5                        Moran, Mr. James    male   NaN      0      0
      7          Palsson, Master. Gosta Leonard    male   2.0      3      1
      ..                                    …     …   …    …    …
      883         Banfield, Mr. Frederick James    male  28.0      0      0
      884                Sutehall, Mr. Henry Jr    male  25.0      0      0
      885  Rice, Mrs. William (Margaret Norton)  female  39.0      0      5
      886                 Montvila, Rev. Juozas    male  27.0      0      0
      890                   Dooley, Mr. Patrick    male  32.0      0      0

                     Ticket     Fare Cabin Embarked
      0           A/5 21171   7.2500   NaN        S
      2    STON/O2. 3101282   7.9250   NaN        S
      4              373450   8.0500   NaN        S
      5              330877   8.4583   NaN        Q
      7              349909  21.0750   NaN        S
      ..                …      …   …      …
      883  C.A./SOTON 34068  10.5000   NaN        S
      884   SOTON/OQ 392076   7.0500   NaN        S
      885            382652  29.1250   NaN        Q
      886            211536  13.0000   NaN        S
      890            370376   7.7500   NaN        Q

      [641 rows x 13 columns]
```

We can also chain conditional statements together by wrapping them in parenthesis and making use of the & and | operators ('and' and 'or' operators, respectively).

In the cell below, slice the DataFrame so that it only contains passengers with a `Fare` value between 50 and 100, inclusive.

[52]:

```
fares_50_to_100_df = df[(df["Fare"] >= 50) & (df["Fare"] <= 100)]
fares_50_to_100_df.head()
```

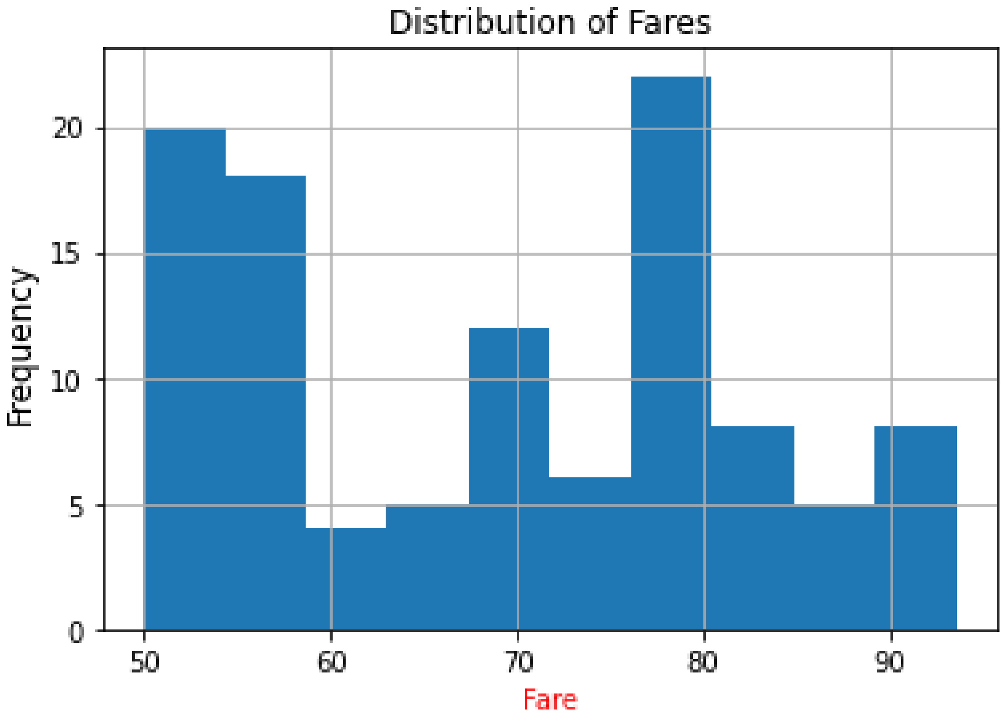
[52]:

```
    Unnamed: 0  PassengerId  Survived Pclass  \
1            1            2         1      1
3            3            4         1      1
6            6            7         0      1
34          34           35         0      1
35          35           36         0      1

                                                 Name     Sex   Age  SibSp  \
1   Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                             McCarthy, Mr. Timothy J    male  54.0      0
34                            Meyer, Mr. Edgar Joseph    male  28.0      1
35                     Holverson, Mr. Alexander Oskar    male  42.0      1

    Parch    Ticket     Fare Cabin Embarked
1       0  PC 17599  71.2833   C85        C
3       0    113803  53.1000  C123        S
6       0     17463  51.8625   E46        S
34      0  PC 17604  82.1708   NaN        C
35      0    113789  52.0000   NaN        S
```

We could go further and then preview the Fare column of this new subsetted DataFrame:

[53]:

```
fares_50_to_100_df['Fare'].hist()
plt.xlabel('Fare', color='red')
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Fares');
```



Remember that there are two syntactically correct ways to access a column in a DataFrame. For instance, `df['Name']` and `df.Name` return the same thing.

In the cell below, use the dot notation syntax and slice a DataFrame that contains male passengers that survived that also belong to Pclass 2 or 3. Be sure to preview the column names and content of the `Sex` column.

[54]:
```
# Checking column names for reference
df.columns
```

[54]:
```
Index(['Unnamed: 0', 'PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
       'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[55]:
```
# Checking column values to hardcode query below
```

[56]:
```
poor_male_survivors_df = df[(df["Pclass"].isin(["2","3"]))  & (df["Sex"] ==␣
 ↪"male")]
print(len(poor_male_survivors_df))
poor_male_survivors_df.head()
```

```
434
```

```
[56]:     Unnamed: 0  PassengerId  Survived Pclass                          Name  \
     0             0            1         0      3         Braund, Mr. Owen Harris
     4             4            5         0      3        Allen, Mr. William Henry
     5             5            6         0      3                Moran, Mr. James
     7             7            8         0      3  Palsson, Master. Gosta Leonard
     12           12           13         0      3  Saundercock, Mr. William Henry

          Sex   Age  SibSp  Parch      Ticket     Fare Cabin Embarked
     0   male  22.0      1      0   A/5 21171   7.2500   NaN        S
     4   male  35.0      0      0      373450   8.0500   NaN        S
     5   male   NaN      0      0      330877   8.4583   NaN        Q
     7   male   2.0      3      1      349909  21.0750   NaN        S
     12  male  20.0      0      0  A/5. 2151   8.0500   NaN        S
```

Great! Now that you've reviewed the methods for slicing a DataFrame for querying our data, let's explore a sample use case.

1.5 Practical Example: Slicing DataFrames

In this section, you're looking to investigate whether women and children survived more than men, or that rich passengers were more likely to survive than poor passengers. The easiest way to confirm this is to slice the data into DataFrames that contain each subgroup, and then quickly visualize the survival rate of each subgroup with histograms.

In the cell below, create a DataFrame that contains passengers that are female, as well as children (males included) ages 15 and under.

Additionally, create a DataFrame that contains only adult male passengers over the age of 15.

```
[ ]:
```

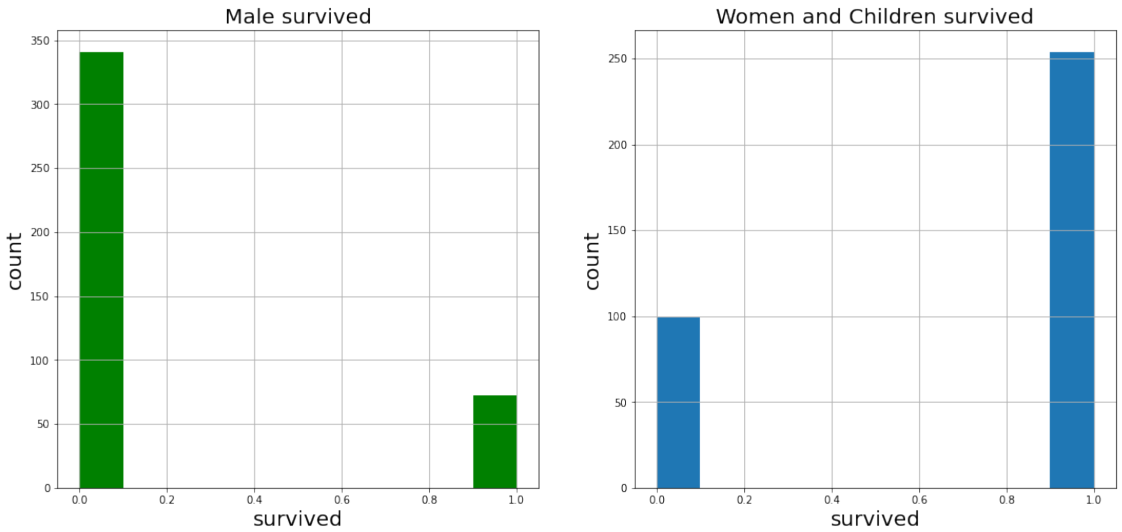
```
[57]: women_and_children_df = df[(df["Sex"] == "female") | (df["Age"] < 15)]
      adult_males_df = df[(df["Sex"] == "male") & (df["Age"] > 15)]
```

Great! Now, you can use the `matplotlib` functionality built into the DataFrame objects to quickly create visualizations of the `Survived` column for each DataFrame.

In the cell below, create histogram visualizations of the `Survived` column for both DataFrames. Bonus points if you use `plt.title()` to label them correctly and make it easy to tell them apart!

```
[58]: # Your code here
      fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (18,8))
      women_and_children_ax = axes[1]
      women_and_children_df["Survived"].hist(ax = women_and_children_ax)
      women_and_children_ax.set_title("Women and Children survived", fontsize = 20)
      women_and_children_ax.set_xlabel("survived", fontsize = 20)
      women_and_children_ax.set_ylabel("count", fontsize = 20)
```

```
adult_males_ax = axes[0]
adult_males_df["Survived"].hist(ax = adult_males_ax, color = "green")
adult_males_ax.set_title("Male survived", fontsize = 20)
adult_males_ax.set_xlabel("survived", fontsize = 20)
adult_males_ax.set_ylabel("count", fontsize = 20);
```



Well that seems like a pretty stark difference – it seems that there was drastically different behavior between the groups! Now, let's repeat the same process, but separating rich and poor passengers.

In the cell below, create one DataFrame containing First Class passengers (`Pclass == 1`), and another DataFrame containing everyone else.

[59]:
```
first_class_df = df[df["Pclass"]== "1"]
second_third_class_df = df[df["Pclass"].isin(["2","3"])]
first_class_df.head()
```

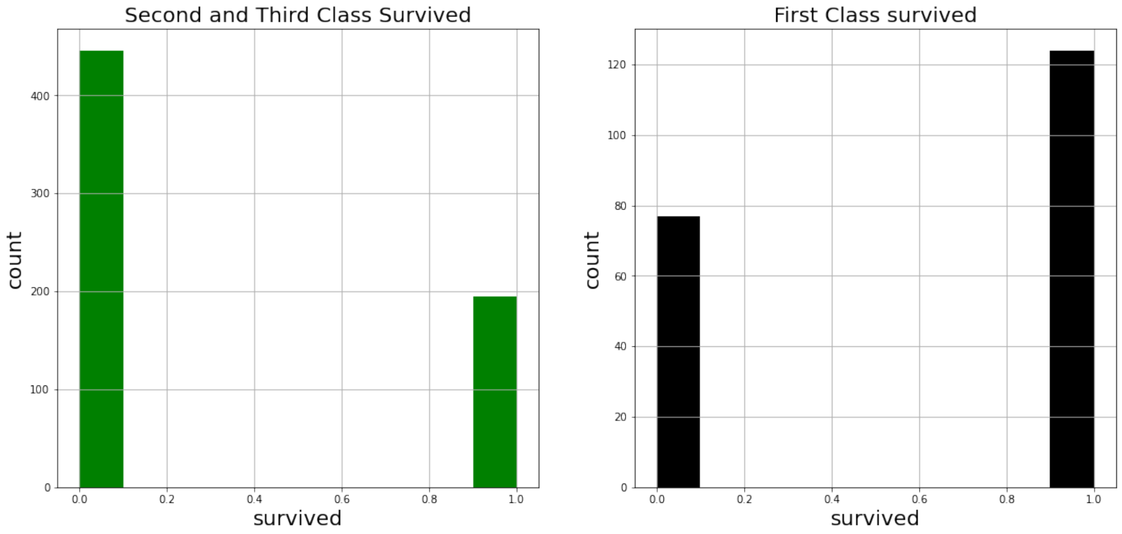
[59]:
```
    Unnamed: 0  PassengerId  Survived Pclass  \
1            1            2         1      1
3            3            4         1      1
6            6            7         0      1
11          11           12         1      1
23          23           24         1      1

                                                 Name     Sex   Age  SibSp  \
1   Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                             McCarthy, Mr. Timothy J    male  54.0      0
11                           Bonnell, Miss. Elizabeth  female  58.0      0
23                          Sloper, Mr. William Thompson    male  28.0      0
```

```
    Parch     Ticket     Fare Cabin Embarked
1       0   PC 17599  71.2833   C85        C
3       0     113803  53.1000  C123        S
6       0      17463  51.8625   E46        S
11      0     113783  26.5500  C103        S
23      0     113788  35.5000    A6        S
```

Now, create histograms of the surivival for each subgroup, just as you did above.

[60]:

```
# Your code here
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (18,8))
first_class_ax = axes[1]
first_class_df["Survived"].hist(ax = first_class_ax, color = "black")
first_class_ax.set_title("First Class survived", fontsize = 20)
first_class_ax.set_xlabel("survived", fontsize = 20)
first_class_ax.set_ylabel("count", fontsize = 20)




second_third_class_ax = axes[0]
second_third_class_df["Survived"].hist(ax = second_third_class_ax, color =␣
 ↪"green")
second_third_class_ax.set_title("Second and Third Class Survived", fontsize =␣
 ↪20)
second_third_class_ax.set_xlabel("survived", fontsize = 20)
second_third_class_ax.set_ylabel("count", fontsize = 20);
```



To the surprise of absolutely no one, it seems like First Class passengers were more likely to survive than not, while 2nd and 3rd class passengers were more likely to die than not. However, don't read too far into these graphs, as these aren't at the same scale, so they aren't fair comparisons.

Slicing is a useful method for quickly getting DataFrames that contain only the examples we're looking for. It's a quick, easy method that feels intuitive in Python, since we can rely on the same conditional logic that we would if we were just writing `if/else` statements.

1.6 Using the `.query()` method

Instead of slicing, you can also make use of the DataFrame's built-in `.query()` method. This method reads a bit more cleanly and allows us to pass in our arguments as a string. For more information or example code on how to use this method, see the pandas documentation.

In the cell below, use the `.query()` method to slice a DataFrame that contains only passengers who have a `PassengerId` greater than or equal to 500.

[61]:

```python
query_string = "PassengerId >= 500"
high_passenger_number_df = df.query(query_string)
high_passenger_number_df.head()
```

[61]:

```
     Unnamed: 0  PassengerId  Survived Pclass                            Name  \
499         499          500         0      3              Svensson, Mr. Olof
500         500          501         0      3                Calic, Mr. Petar
501         501          502         0      3             Canavan, Miss. Mary
502         502          503         0      3  O'Sullivan, Miss. Bridget Mary
503         503          504         0      3  Laitinen, Miss. Kristina Sofia

        Sex   Age  SibSp  Parch  Ticket    Fare Cabin Embarked
499    male  24.0      0      0  350035  7.7958   NaN        S
500    male  17.0      0      0  315086  8.6625   NaN        S
501  female  21.0      0      0  364846  7.7500   NaN        Q
502  female   NaN      0      0  330909  7.6292   NaN        Q
503  female  37.0      0      0    4135  9.5875   NaN        S
```

Just as with slicing, you can pass in queries with multiple conditions. One unique difference between using the `.query()` method and conditional slicing is that you can use `and` or `&` as well as `or` or `|` (for fun, try reading this last sentence out loud), while you are limited to the `&` and `|` symbols to denote and/or operations with conditional slicing.

In the cell below, use the `query()` method to return a DataFrame that contains only female passengers of ages 15 and under.

Hint: Although the entire query is a string, you'll still need to denote that `female` is also a string, within the string. (*String-Ception?*)

[65]:

```python
female_children_df = df.query("Sex == \"female\" and Age <= 15")
female_children_df.head()
```

[65]:

```
    Unnamed: 0  PassengerId  Survived Pclass  \
9            9           10         1      2
10          10           11         1      3
14          14           15         0      3
22          22           23         1      3
```

```
24          24           25          0      3

                                  Name     Sex   Age  SibSp  Parch    Ticket  \
9    Nasser, Mrs. Nicholas (Adele Achem)  female  14.0      1      0    237736
10       Sandstrom, Miss. Marguerite Rut  female   4.0      1      1   PP 9549
14  Vestrom, Miss. Hulda Amanda Adolfina  female  14.0      0      0    350406
22            McGowan, Miss. Anna "Annie"  female  15.0      0      0    330923
24          Palsson, Miss. Torborg Danira  female   8.0      3      1    349909

       Fare Cabin Embarked
9   30.0708   NaN        C
10  16.7000    G6        S
14   7.8542   NaN        S
22   8.0292   NaN        Q
24  21.0750   NaN        S
```

[66]:
```python
female_children_df1 = df[(df.Sex == "female") & (df.Age <=15)]
female_children_df1.head()
```

[66]:
```
    Unnamed: 0  PassengerId  Survived Pclass  \
9            9           10         1      2
10          10           11         1      3
14          14           15         0      3
22          22           23         1      3
24          24           25         0      3

                                  Name     Sex   Age  SibSp  Parch    Ticket  \
9    Nasser, Mrs. Nicholas (Adele Achem)  female  14.0      1      0    237736
10       Sandstrom, Miss. Marguerite Rut  female   4.0      1      1   PP 9549
14  Vestrom, Miss. Hulda Amanda Adolfina  female  14.0      0      0    350406
22            McGowan, Miss. Anna "Annie"  female  15.0      0      0    330923
24          Palsson, Miss. Torborg Danira  female   8.0      3      1    349909

       Fare Cabin Embarked
9   30.0708   NaN        C
10  16.7000    G6        S
14   7.8542   NaN        S
22   8.0292   NaN        Q
24  21.0750   NaN        S
```

A cousin of the `query()` method, `eval()` allows you to use the same string-filled syntax as querying for creating new columns. For instance:

```
some_df.eval('C = A + B')
```

would return a copy of the `some_df` dataframe, but will now include a column `C` where all values are equal to the sum of the `A` and `B` values for any given row. This method also allows the user to specify if the operation should be done in place or not, providing a quick, easy syntax for simple feature engineering.

In the cell below, use the DataFrame's `eval()` method in place to add a column called `Age_x_Fare`, and set it equal to `Age` multiplied by `Fare`.

[67]:
```
df = df.eval("Age_x_Fare = Age * Fare")
df.head()
```

[67]:
```
   Unnamed: 0  PassengerId  Survived Pclass  \
0           0            1         0      3
1           1            2         1      1
2           2            3         1      3
3           3            4         1      1
4           4            5         0      3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked  Age_x_Fare
0      0         A/5 21171   7.2500   NaN        S    159.5000
1      0          PC 17599  71.2833   C85        C   2708.7654
2      0  STON/O2. 3101282   7.9250   NaN        S    206.0500
3      0            113803  53.1000  C123        S   1858.5000
4      0            373450   8.0500   NaN        S    281.7500
```

[68]:
```
df["Age_X_Fare1"] = df.Age * df.Fare
df.head()
```

[68]:
```
   Unnamed: 0  PassengerId  Survived Pclass  \
0           0            1         0      3
1           1            2         1      1
2           2            3         1      3
3           3            4         1      1
4           4            5         0      3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked  Age_x_Fare  Age_X_Fare1
0      0         A/5 21171   7.2500   NaN        S    159.5000     159.5000
```

```
1       0           PC 17599  71.2833   C85        C   2708.7654    2708.7654
2       0  STON/O2. 3101282   7.9250   NaN        S    206.0500     206.0500
3       0             113803  53.1000  C123        S   1858.5000    1858.5000
4       0             373450   8.0500   NaN        S    281.7500     281.7500
```

Great! Now, let's move on the coolest part of this lab–querying DataFrames with SQL!

1.7 Querying DataFrames With SQL

For the final section of the lab, you'll make use of the `pandasql` library. Pandasql is a library designed to make it easy to query DataFrames directly with SQL syntax, which was open-sourced by the company, Yhat, in late 2016. It's very straightforward to use, but you are still encouraged to take a look at the documentation as needed.

If you're using the pre-built virtual environment, you should already have the package ready to import. If not, uncomment and run the cell below to `pip install pandasql` so that it is available to import.

[69]:
```
!pip install pandasql
```

```
Collecting pandasql
  Downloading pandasql-0.7.3.tar.gz (26 kB)
Requirement already satisfied: numpy in /opt/conda/lib/python3.9/site-packages
(from pandasql) (1.21.1)
Requirement already satisfied: pandas in /opt/conda/lib/python3.9/site-packages
(from pandasql) (1.3.1)
Requirement already satisfied: sqlalchemy in /opt/conda/lib/python3.9/site-
packages (from pandasql) (1.3.24)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.9/site-packages (from pandas->pandasql) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.9/site-
packages (from pandas->pandasql) (2021.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.9/site-
packages (from python-dateutil>=2.7.3->pandas->pandasql) (1.16.0)
Building wheels for collected packages: pandasql
  Building wheel for pandasql (setup.py) … done
  Created wheel for pandasql: filename=pandasql-0.7.3-py3-none-any.whl
size=26818
sha256=107b97d807e8f3c52f98093d970de0fe02d8a8316364aaa6a188693947695017
  Stored in directory: /home/jovyan/.cache/pip/wheels/63/e8/ec/75b1df467ecf57b6e
cecb32cb16f4e86697cbfe55cb0c51f07
Successfully built pandasql
Installing collected packages: pandasql
Successfully installed pandasql-0.7.3
```

That should have installed everything correctly. This library has a few dependencies, which you should already have installed. If you don't, just `pip install` them in your terminal and you'll be good to go!

In the cell below, import `sqldf` from `pandasql`.

[70]:
```
# Your code here
from pandasql import sqldf
```

Great! Now, it’s time to get some practice with this handy library.

`pandasql` allows you to pass in SQL queries in the form of a string to directly query your database. Each time you make a query, you need to pass an additional parameter that gives it access to the other variables in the session/environment. You can use a lambda function to pass `locals()` or `globals()` so that you don’t have to type this every time.

In the cell below, create a variable called `pysqldf` and set it equal to a lambda function `q` that returns `sqldf(q, globals())`. If you’re unsure of how to do this, see the example in the documentation.

[71]:
```
pysqldf = lambda q: sqldf(q, globals())
```

Great! That will save you from having to pass `globals()` as an argument every time you query, which can get a bit tedious.

Now write a basic query to get a list of passenger names from `df`, limit 10. If you would prefer to format your query on multiple lines and style it as canonical SQL, that’s fine – remember that multi-line strings in Python are denoted by `"""` – for example:

```
"""
This is a
Multi-Line String
"""
```

In the cell below, write a SQL query that returns the names of the first 10 passengers.

[72]:
```
q = """
SELECT Name
FROM df
LIMIT 10;

"""

passenger_names = pysqldf(q)
passenger_names
```

[72]:
```
                                                Name
0                            Braund, Mr. Owen Harris
1  Cumings, Mrs. John Bradley (Florence Briggs Th…
2                             Heikkinen, Miss. Laina
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                           Allen, Mr. William Henry
5                                   Moran, Mr. James
6                            McCarthy, Mr. Timothy J
7                     Palsson, Master. Gosta Leonard
8  Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
```

```
9                 Nasser, Mrs. Nicholas (Adele Achem)
```

Great! Now, for a harder one:

In the cell below, query the DataFrame for names and fares of any male passengers that survived, limit 30.

```
[77]: q2 = """
      SELECT Name, Fare
      FROM df
      WHERE Sex = "male and survived = 1
      LIMIT 30;
      """

      sql_surviving_males = pysqldf(q)
      sql_surviving_males
```

```
[77]:                                                 Name
      0                            Braund, Mr. Owen Harris
      1  Cumings, Mrs. John Bradley (Florence Briggs Th…
      2                             Heikkinen, Miss. Laina
      3       Futrelle, Mrs. Jacques Heath (Lily May Peel)
      4                           Allen, Mr. William Henry
      5                                   Moran, Mr. James
      6                            McCarthy, Mr. Timothy J
      7                     Palsson, Master. Gosta Leonard
      8  Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
      9                Nasser, Mrs. Nicholas (Adele Achem)
```

This library is really powerful! This makes it easy for us to leverage all of your SQL knowledge to quickly query any DataFrame, especially when you only want to select certain columns. This saves from having to slice/query the DataFrame and then slice the columns you want (or drop the ones you don't want).

Although it's outside the scope of this lab, it's also worth noting that both `pandas` and `pandasql` provide built-in functionality for join operations, too!

1.8 Practical Example: SQL in Pandas

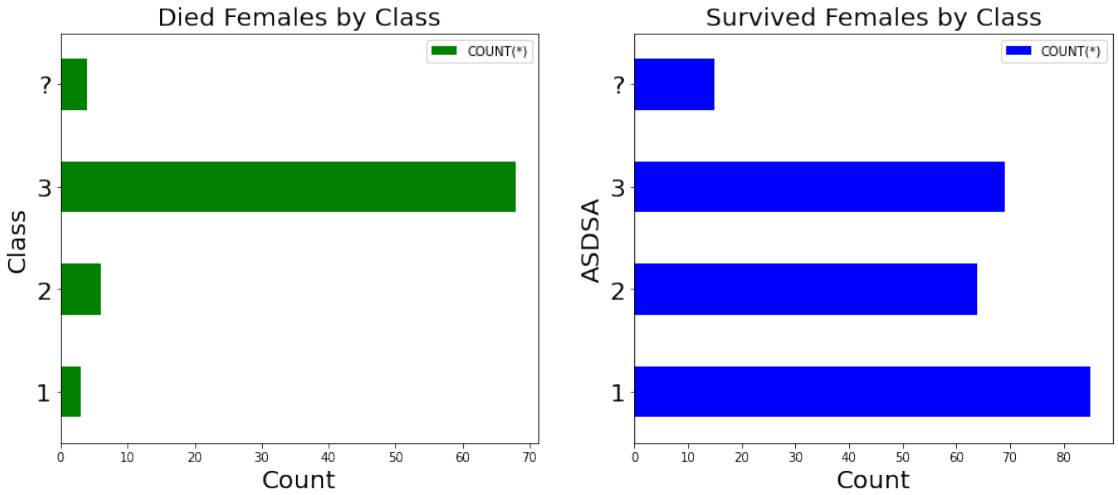
In the cell below, create 2 separate DataFrames using `pandasql`. One should contain the Pclass of all female passengers that survived, and the other should contain the Pclass of all female passengers that died.

Then, create a horizontal bar graph visualizations of the `Pclass` column for each DataFrame to compare the two. Bonus points for taking the time to make the graphs extra readable by adding titles, labeling each axis, and cleaning up the number of ticks on the X-axis!

```
[113]: # Write your queries in these variables to keep your code well-formatted and␣
        ↪readable
```

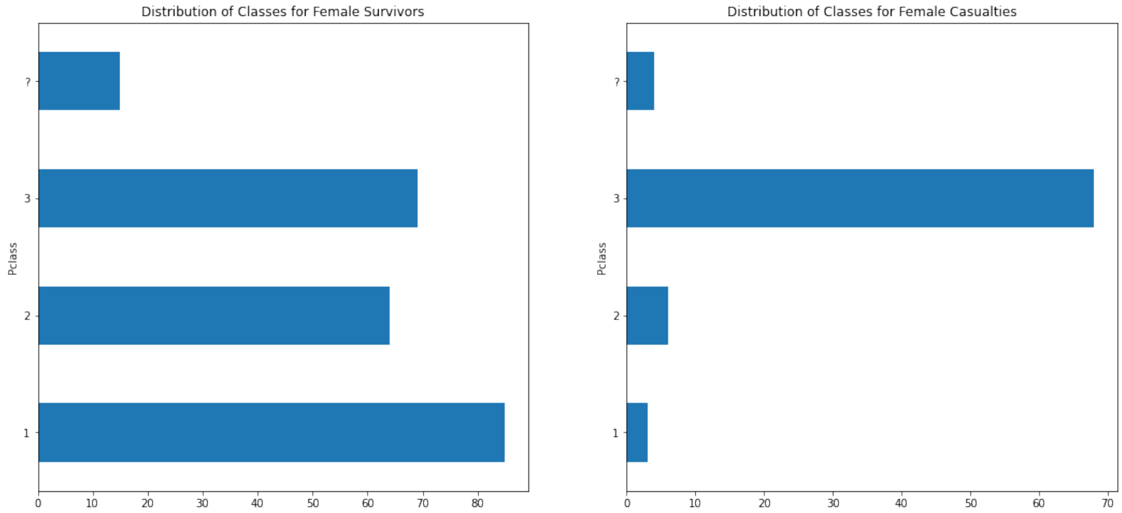
```
q3 = """ SELECT Pclass, COUNT(*) FROM df WHERE Survived = 1 and Sex = "female"␣
 ↪GROUP BY Pclass"""
q4 = """ SELECT Pclass, COUNT(*) FROM df WHERE Survived = 0 and Sex = "female"␣
 ↪GROUP BY Pclass  """

survived_females_by_pclass_df = pysqldf(q3)

died_females_by_pclass_df = pysqldf(q4)

print(died_females_by_pclass_df.info())

# Create and label the histograms for each below!
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (15, 6))



dfp_ax = axes[0]
lid = died_females_by_pclass_df["Pclass"]
died_females_by_pclass_df.plot(kind = "barh", ax = dfp_ax, color = "green")
dfp_ax.set_title("Died Females by Class", fontsize = 20)
dfp_ax.set_xlabel("Count", fontsize = 20)
dfp_ax.set_ylabel("Class", fontsize = 20)
dfp_ax.set_yticklabels(lid, fontsize = 20)





sfp_ax = axes[1]
lis = survived_females_by_pclass_df["Pclass"]
survived_females_by_pclass_df.plot(kind = "barh", ax = sfp_ax, color = "blue")
sfp_ax.set_title("Survived Females by Class", fontsize = 20)
sfp_ax.set_xlabel("Count", fontsize = 20)
sfp_ax.set_ylabel("ASDSA", fontsize = 20)
sfp_ax.set_yticklabels(lis, fontsize = 20);
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    4 non-null      object
 1   COUNT(*)  4 non-null      int64
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
None
```



[114]:

```
# From GitHub
# Write your queries in these variables to keep your code well-formatted and␣
↪readable
q3 = """SELECT Pclass, Count(*)
        FROM df
        WHERE Sex = 'female' AND Survived = 1
        GROUP BY Pclass;"""
q4 = """SELECT Pclass, Count(*)
        FROM df
        WHERE Sex = 'female' AND Survived = 0
        GROUP BY Pclass;"""

survived_females_by_pclass_df = pysqldf(q3)
died_females_by_pclass_df = pysqldf(q4)

# Create and label the histograms for each below!
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(18,8))

survived_females_by_pclass_df.set_index('Pclass')['Count(*)'].plot(kind='barh',␣
↪ax=axes[0])
axes[0].set_title('Distribution of Classes for Female Survivors')

died_females_by_pclass_df.set_index('Pclass')['Count(*)'].plot(kind='barh',␣
↪ax=axes[1])
axes[1].set_title('Distribution of Classes for Female Casualties');
```



1.9 Summary

In this lab, you practiced how to query Pandas DataFrames using SQL.

[ ]: