Using SQL with Pandas - Lab

Introduction

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

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

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`.

```
# brian-answer
import pandas as pd
import pandasql as ps
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.

```
df = pd.read_csv('titanic.csv', index_col = 0)
df.head()

   PassengerId  Survived Pclass  \
0            1         0      3
1            2         1      1
2            3         1      3
3            4         1      1
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
```

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!

```
# Preview values first to ensure proper encoding when slicing
# df.Pclass.value_counts().to_frame()
df.Pclass.unique()
```

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

```
# brian-answer
no_first_class_df = df[df['Pclass'].isin(['2','3'])]
no_first_class_df.head()
```

```
   PassengerId  Survived Pclass                            Name
Sex   Age  \
0            1         0      3         Braund, Mr. Owen Harris
male  22.0
2            3         1      3          Heikkinen, Miss. Laina
female  26.0
4            5         0      3        Allen, Mr. William Henry
male  35.0
5            6         0      3                Moran, Mr. James
male   NaN
7            8         0      3  Palsson, Master. Gosta Leonard
male   2.0
```

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

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.

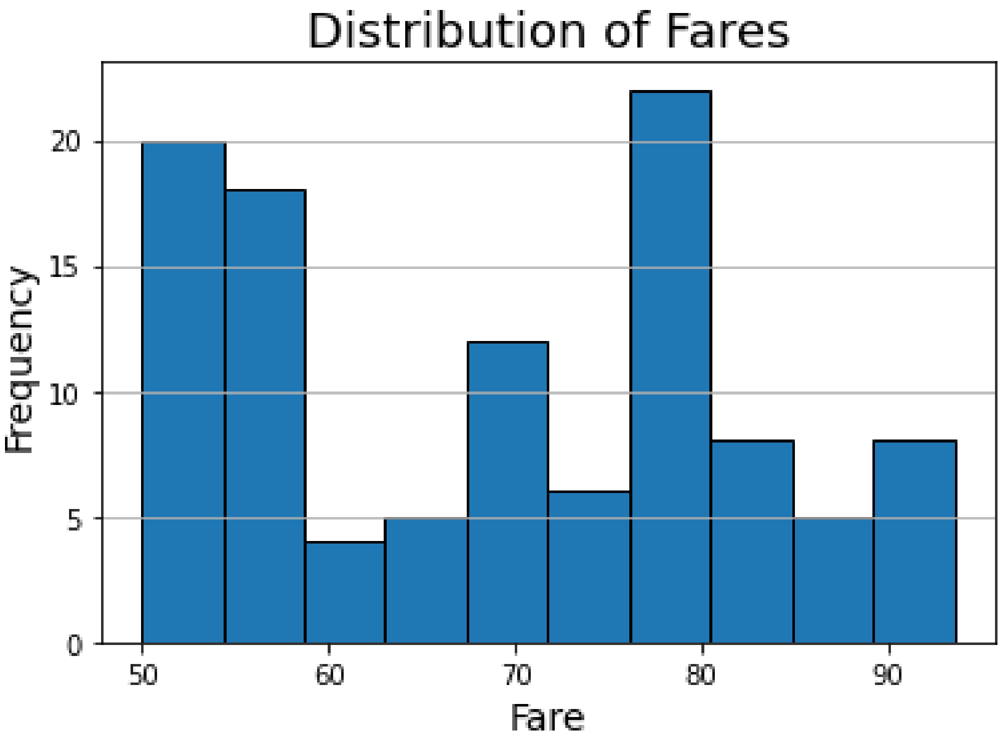
```
# brian-answer
fares_50_to_100_df = df[(df['Fare'] >= 50) & (df['Fare'] <= 100)]
fares_50_to_100_df.head()
```

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

```
plt.figure(figsize=(6, 4))
fares_50_to_100_df['Fare'].hist(edgecolor='black')
```

```
plt.title('Distribution of Fares', fontsize = 18)
plt.xlabel('Fare', fontsize = 14)
plt.ylabel('Frequency', fontsize = 14)
plt.grid(axis='x')
plt.show();
```



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.

```
# Checking column names for reference
list(df.columns)
```

```
['PassengerId',
 'Survived',
 'Pclass',
 'Name',
 'Sex',
 'Age',
 'SibSp',
 'Parch',
 'Ticket',
 'Fare',
 'Cabin',
 'Embarked']
```

```
# Checking column values to hardcode query below
df['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
poor_male_survivors_df = df[(df['Pclass'].isin(['2', '3'])) &
(df['Sex'] == 'male') & (df['Survived'] == 1)]
poor_male_survivors_df.head()
```

```
    PassengerId  Survived Pclass                          Name   Sex
Age  \
17           18         1      2  Williams, Mr. Charles Eugene  male
NaN
21           22         1      2          Beesley, Mr. Lawrence  male
34.0
36           37         1      3               Mamee, Mr. Hanna  male
NaN
65           66         1      3        Moubarek, Master. Gerios  male
NaN
74           75         1      3                  Bing, Mr. Lee  male
32.0

    SibSp  Parch  Ticket     Fare Cabin Embarked
17      0      0  244373  13.0000   NaN        S
21      0      0  248698  13.0000   D56        S
36      0      0    2677   7.2292   NaN        C
65      1      1    2661  15.2458   NaN        C
74      0      0    1601  56.4958   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.

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.

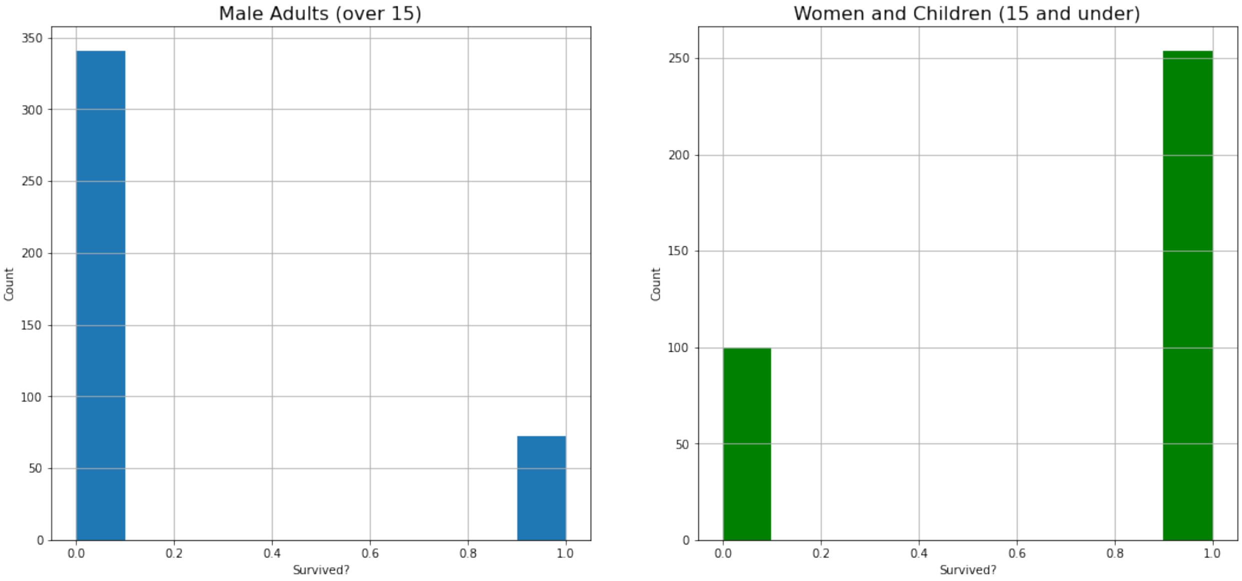
Additionally, create a DataFrame that contains only adult male passengers over the age of 15.

```
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!

```
# brian-answer
fig, axes = plt.subplots(ncols=2, nrows=1, figsize=(18, 8))
ax_0 = axes[0]
adult_males_df['Survived'].hist(ax = ax_0)
ax_0.set_title('Male Adults (over 15)', fontsize = 16)
ax_0.set_xlabel('Survived?')
ax_0.set_ylabel('Count')

ax_1 = axes[1]
women_and_children_df['Survived'].hist(ax = ax_1, color='green')
ax_1.set_title('Women and Children (15 and under)', fontsize = 16)
ax_1.set_xlabel('Survived?')
ax_1.set_ylabel('Count');
```



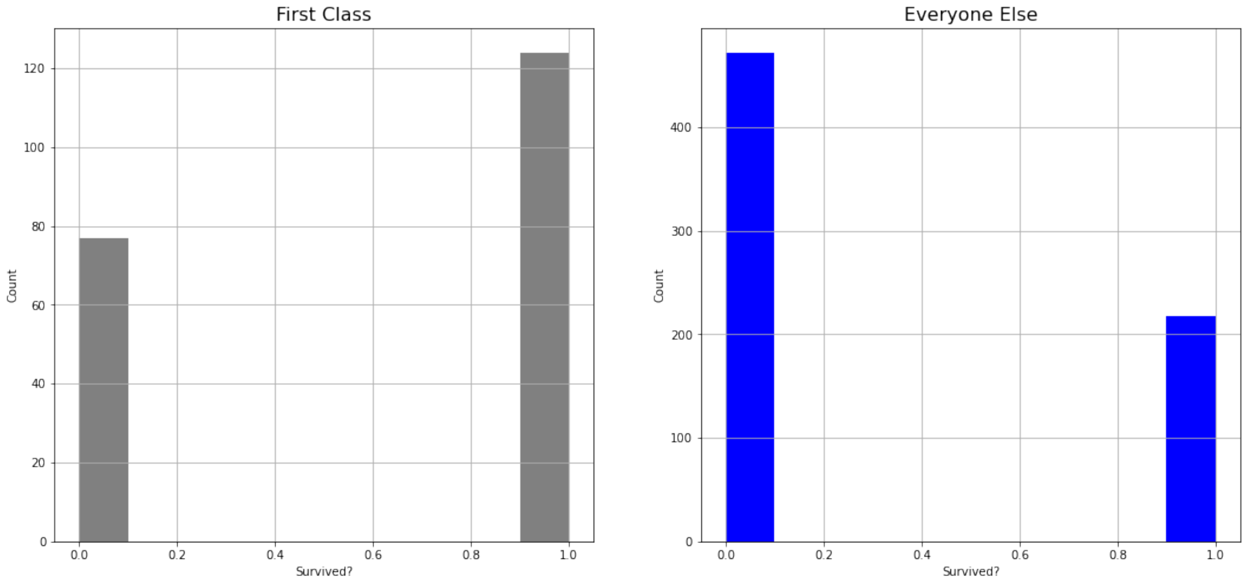
Well that seems like a pretty stark difference -- it seems that there was drastically different behavior between the groups! Now, let's repeat the same process, but separating rich and poor passengers.

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

```
first_class_df = df[df['Pclass'] == '1']
second_third_class_df = df[df['Pclass'] != '1']
```

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

```
# brian-answer
fig, axes = plt.subplots(ncols=2, nrows=1, figsize=(18, 8))
ax_0 = axes[0]
first_class_df['Survived'].hist(ax = ax_0, color='grey')
ax_0.set_title('First Class', fontsize = 16)
ax_0.set_xlabel('Survived?')
ax_0.set_ylabel('Count')

ax_1 = axes[1]
second_third_class_df['Survived'].hist(ax = ax_1, color='blue')
ax_1.set_title('Everyone Else', fontsize = 16)
ax_1.set_xlabel('Survived?')
ax_1.set_ylabel('Count');
```



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.

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.

```
query_string = 'PassengerId >= 500'
print(query_string)
high_passenger_number_df = df.query(query_string)
high_passenger_number_df.head()

PassengerId >= 500

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

      Age  SibSp  Parch  Ticket    Fare Cabin Embarked
499  24.0      0      0  350035  7.7958   NaN        S
500  17.0      0      0  315086  8.6625   NaN        S
501  21.0      0      0  364846  7.7500   NaN        Q
502   NaN      0      0  330909  7.6292   NaN        Q
503  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?*)

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

    PassengerId  Survived Pclass                                  Name
\
9            10         1      2   Nasser, Mrs. Nicholas (Adele Achem)

10           11         1      3       Sandstrom, Miss. Marguerite Rut

14           15         0      3  Vestrom, Miss. Hulda Amanda Adolfina

22           23         1      3           McGowan, Miss. Anna "Annie"
```

```
24           25         0      3         Palsson, Miss. Torborg Danira


       Sex   Age  SibSp  Parch   Ticket     Fare Cabin Embarked
9   female  14.0      1      0   237736  30.0708   NaN        C
10  female   4.0      1      1  PP 9549  16.7000    G6        S
14  female  14.0      0      0   350406   7.8542   NaN        S
22  female  15.0      0      0   330923   8.0292   NaN        Q
24  female   8.0      3      1   349909  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`.

```
df = df.eval('Age_x_Fare = Age*Fare')
df.head()

   PassengerId  Survived Pclass  \
0            1         0      3
1            2         1      1
2            3         1      3
3            4         1      1
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
```

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

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.

```
# !pip install pandasql
```

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`.

```
# brian-answer
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.

```
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.

```
q = """SELECT Name
       FROM df
       LIMIT 10;"""

passenger_names = pysqldf(q)
passenger_names

                                                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.

```
q2 = """SELECT Name, Fare
        FROM df
        WHERE Sex = 'male' AND Survived = 1
        LIMIT 30;"""

sql_surviving_males = pysqldf(q2)
sql_surviving_males

                                               Name     Fare
0                      Williams, Mr. Charles Eugene  13.0000
1                             Beesley, Mr. Lawrence  13.0000
2                      Sloper, Mr. William Thompson  35.5000
3                                  Mamee, Mr. Hanna   7.2292
4                                 Woolner, Mr. Hugh  35.5000
5                          Moubarek, Master. Gerios  15.2458
6                                     Bing, Mr. Lee  56.4958
7                     Caldwell, Master. Alden Gates  29.0000
8                       Sheerlinck, Mr. Jan Baptist   9.5000
```

```
9                   Greenfield, Mr. William Bertram  63.3583
10                           Moss, Mr. Albert Johan   7.7750
11                     Nicola-Yarred, Master. Elias  11.2417
12                        Madsen, Mr. Fridtjof Arne   7.1417
13     Andersson, Mr. August Edvard ("Wennerstrom")   7.7958
14  Goldsmith, Master. Frank John William "Frankie"  20.5250
15                        Becker, Master. Richard F  39.0000
16    Romaine, Mr. Charles Hallace ("Mr C Rolmane")  26.5500
17                       Navratil, Master. Michel M  26.0000
18                         Cohen, Mr. Gurshon "Gus"   8.0500
19                      Albimona, Mr. Nassef Cassem  18.7875
20                                 Blank, Mr. Henry  31.0000
21                   Sunderland, Mr. Victor Francis   8.0500
22                     Hoyt, Mr. Frederick Maxfield  90.0000
23                        Mellors, Mr. William John  10.5000
24                    Beckwith, Mr. Richard Leonard  52.5542
25                Asplund, Master. Edvin Rojj Felix  31.3875
26                         Persson, Mr. Ernst Ulrik   7.7750
27                     Tornquist, Mr. William Henry   0.0000
28                       Dorking, Mr. Edward Arthur   8.0500
29                          de Mulder, Mr. Theodore   9.5000
```

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).

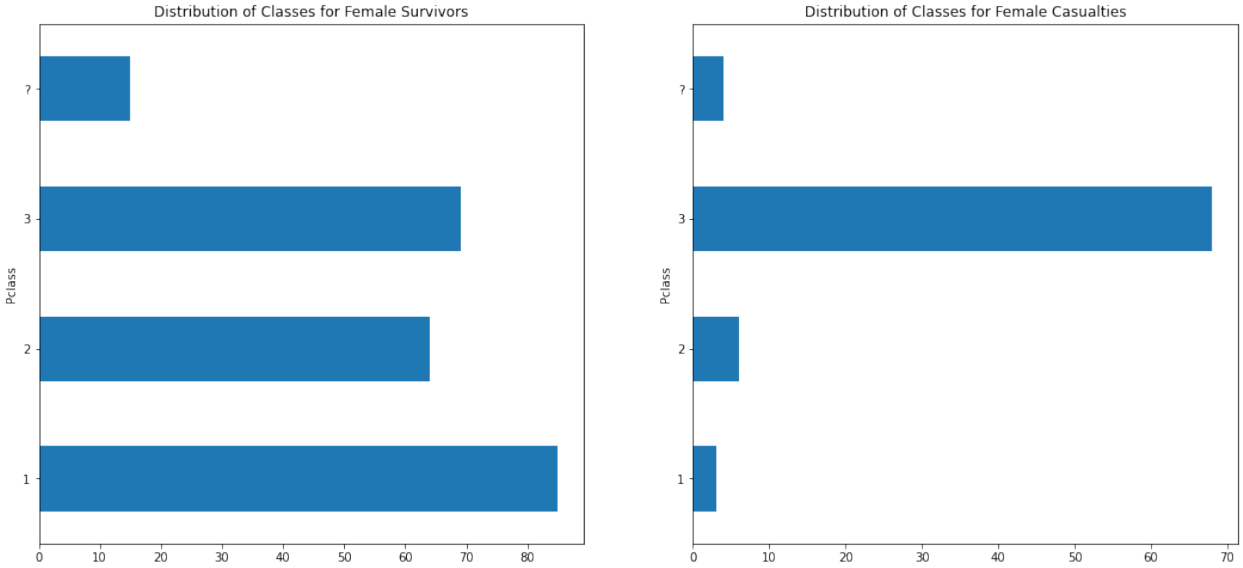
Although it's outside the scope of this lab, it's also worth noting that both `pandas` and `pandasql` provide built-in functionality for join operations, too!

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!

```
# 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');
```



Summary

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