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`.

In [2]:
```
# 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.

In [44]:
```
df = pd.read_csv('titanic.csv', index_col = 0)
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    object 
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(4), object(6)
memory usage: 90.5+ KB
```

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!

In [45]:
```
no_first_class_df = df[(df['Pclass'] == '2') | (df['Pclass'] == '3')]
no_first_class_df.head()
```

Out[45]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	N
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	N

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.

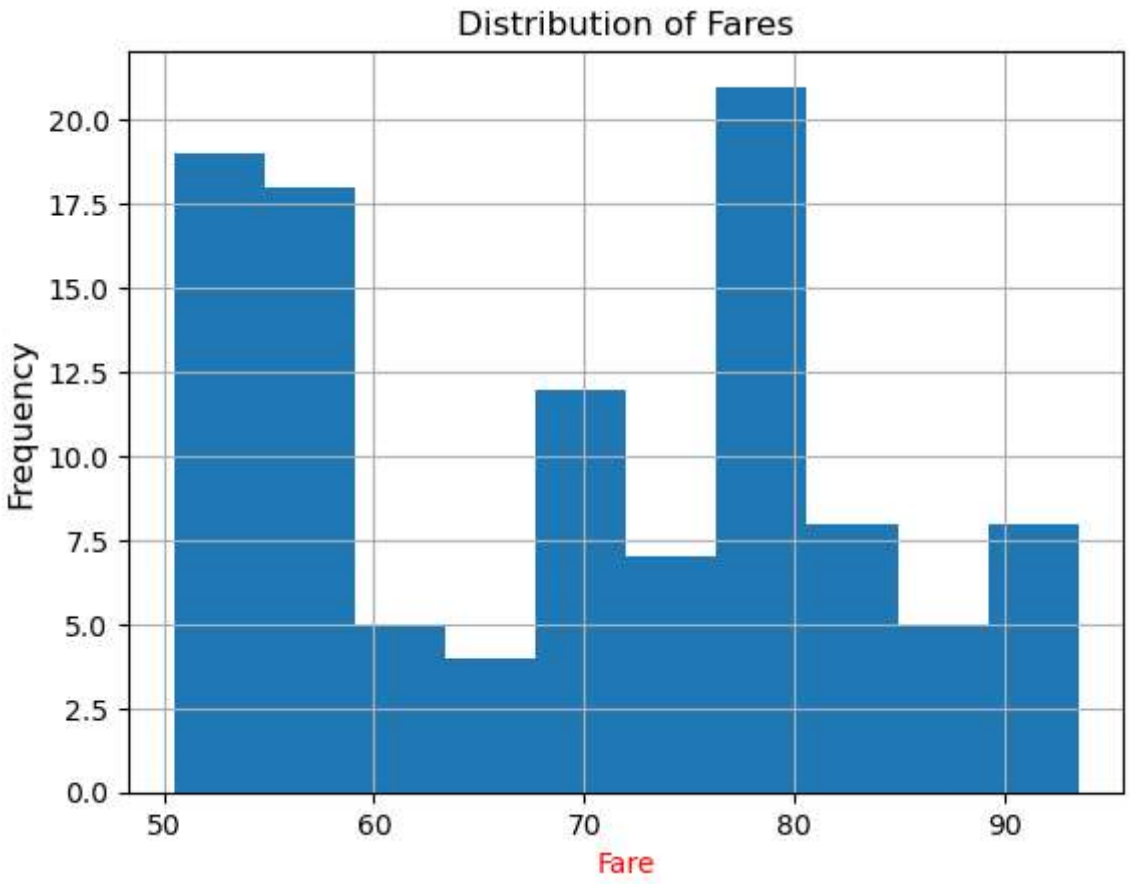
In [47]:
```
fares_50_to_100_df = df[(df['Fare'] < 100) & (df['Fare'] > 50)]
fares_50_to_100_df.head()
```

Out[47]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C8
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C12
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E4
34	35	0	1	Meyer, Mr. Edgar Joseph	male	28.0	1	0	PC 17604	82.1708	Na
35	36	0	1	Holverson, Mr. Alexander Oskar	male	42.0	1	0	113789	52.0000	Na

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

In [48]:
```
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.

In [49]:
```
# Checking column names for reference
df.columns
```

Out[49]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

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

In [64]:
```
poor_male_survivors_df = no_first_class_df[(no_first_class_df.Sex == 'male') &
poor_male_survivors_df.head()
```

Out[64]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN
21	22	1	2	Beesley, Mr. Lawrence	male	34.0	0	0	248698	13.0000	D56
36	37	1	3	Mamee, Mr. Hanna	male	NaN	0	0	2677	7.2292	NaN
65	66	1	3	Moubarek, Master. Gerios	male	NaN	1	1	2661	15.2458	NaN
74	75	1	3	Bing, Mr. Lee	male	32.0	0	0	1601	56.4958	NaN

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.

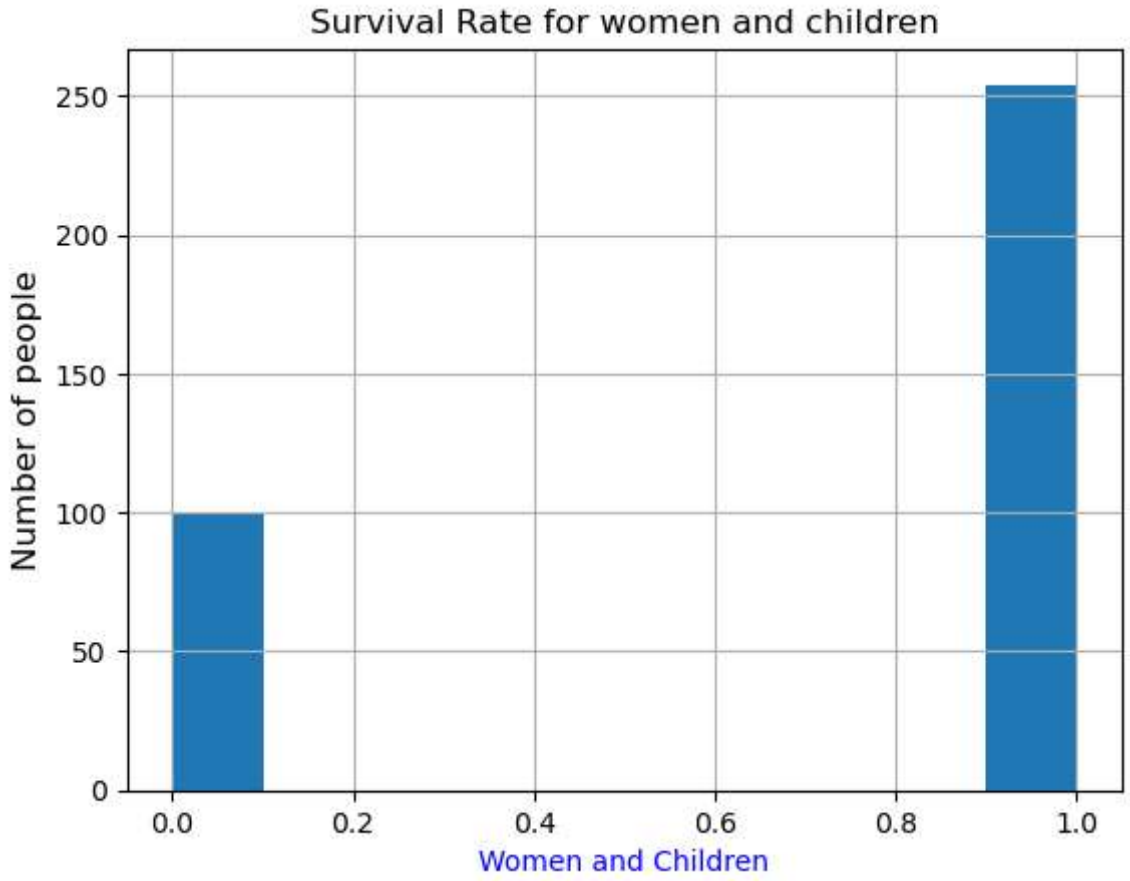
Additionally, create a DataFrame that contains only adult male passengers over the age of 15.

In [65]:
```
women_and_children_df = df[(df['Sex'] == 'female') | ((df['Sex'] == 'male') &
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.

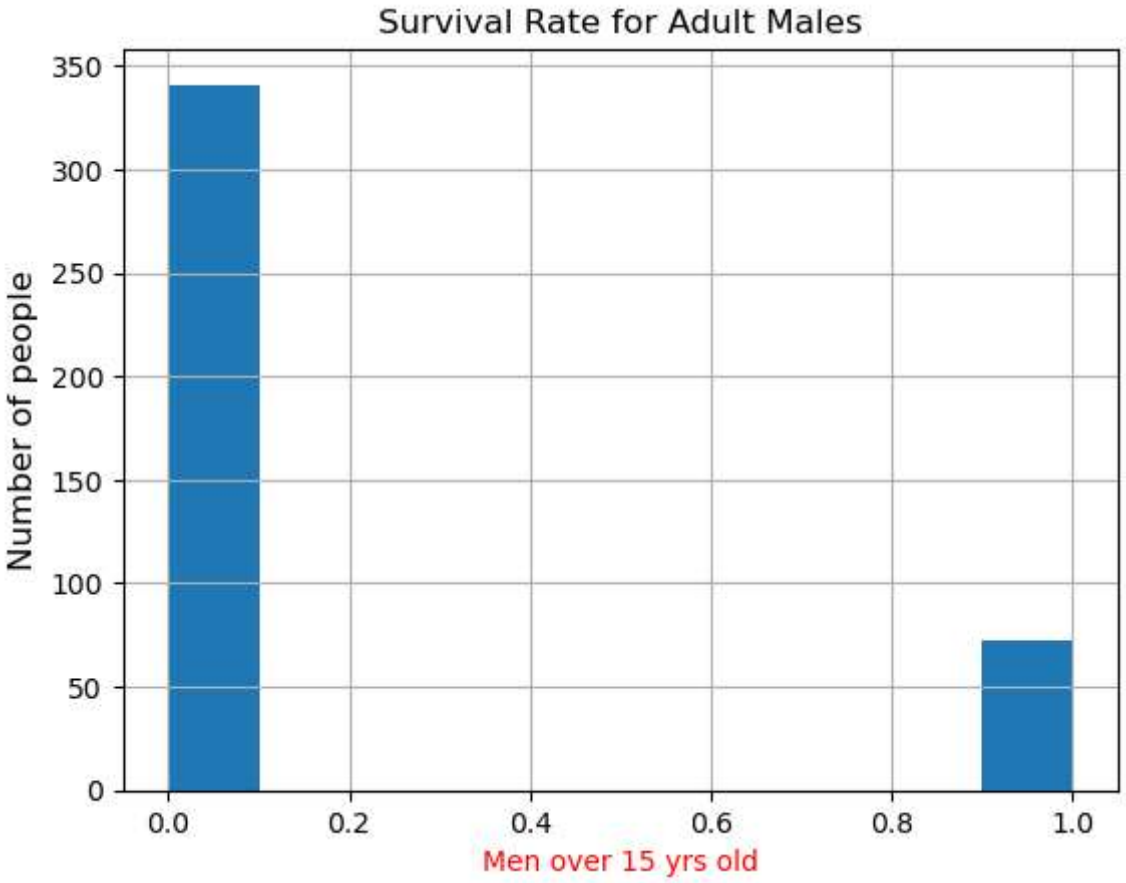
In the cell below, create histogram visualizations of the `Survived` column for both DataFrames. Bonus points if you use `plt.title()` to label them correctly and make it easy to tell them apart!

In [71]:
```
# Your code here
women_and_children_df['Survived'].hist()
plt.xlabel('Women and Children', color='blue')
plt.ylabel('Number of people', fontsize=12)
plt.title('Survival Rate for women and children');
```



In [72]:
```
adult_males_df['Survived'].hist()
plt.xlabel('Men over 15 yrs old', color='red')
plt.ylabel('Number of people', fontsize=12)
plt.title('Survival Rate for Adult Males')
```

Out[72]:
```
Text(0.5, 1.0, 'Survival Rate for Adult Males')
```



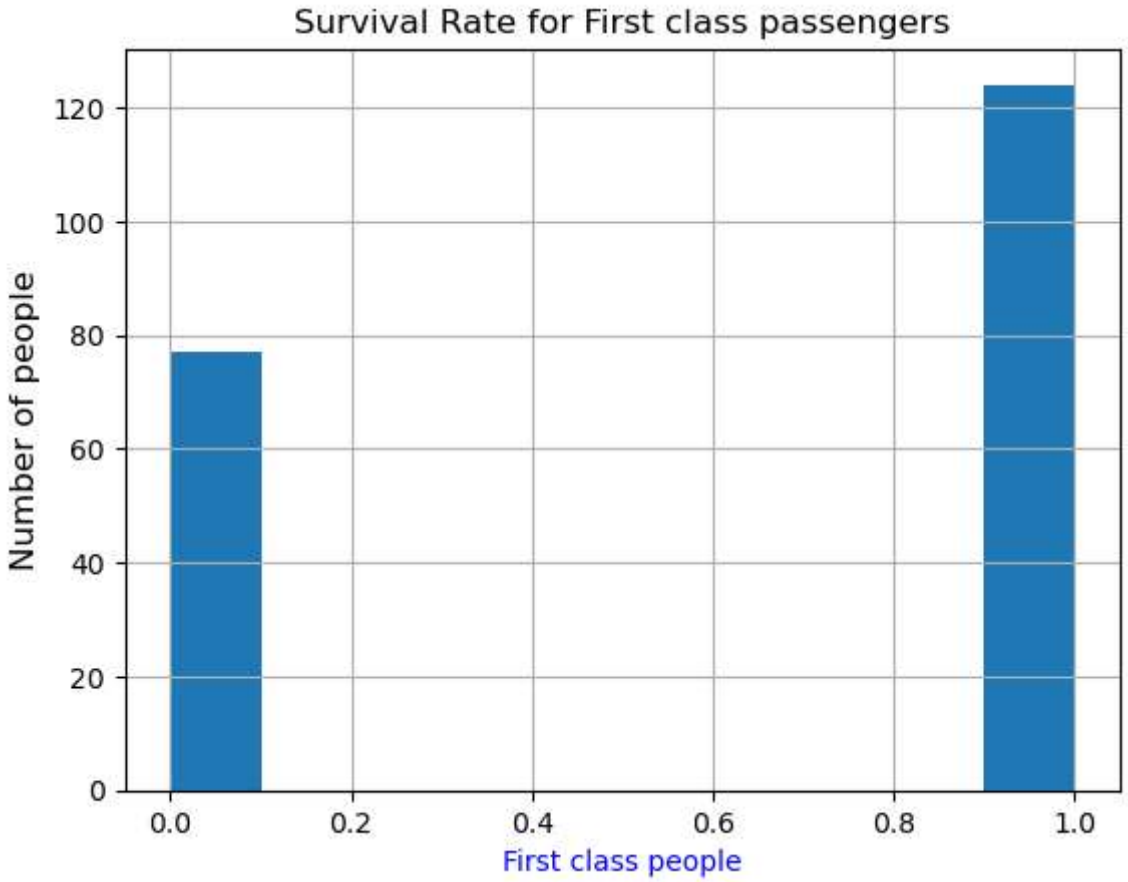
Well that seems like a pretty stark difference -- it seems that there was drastically different behavior between the groups! Now, let's repeat the same process, but separating rich and poor passengers.

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

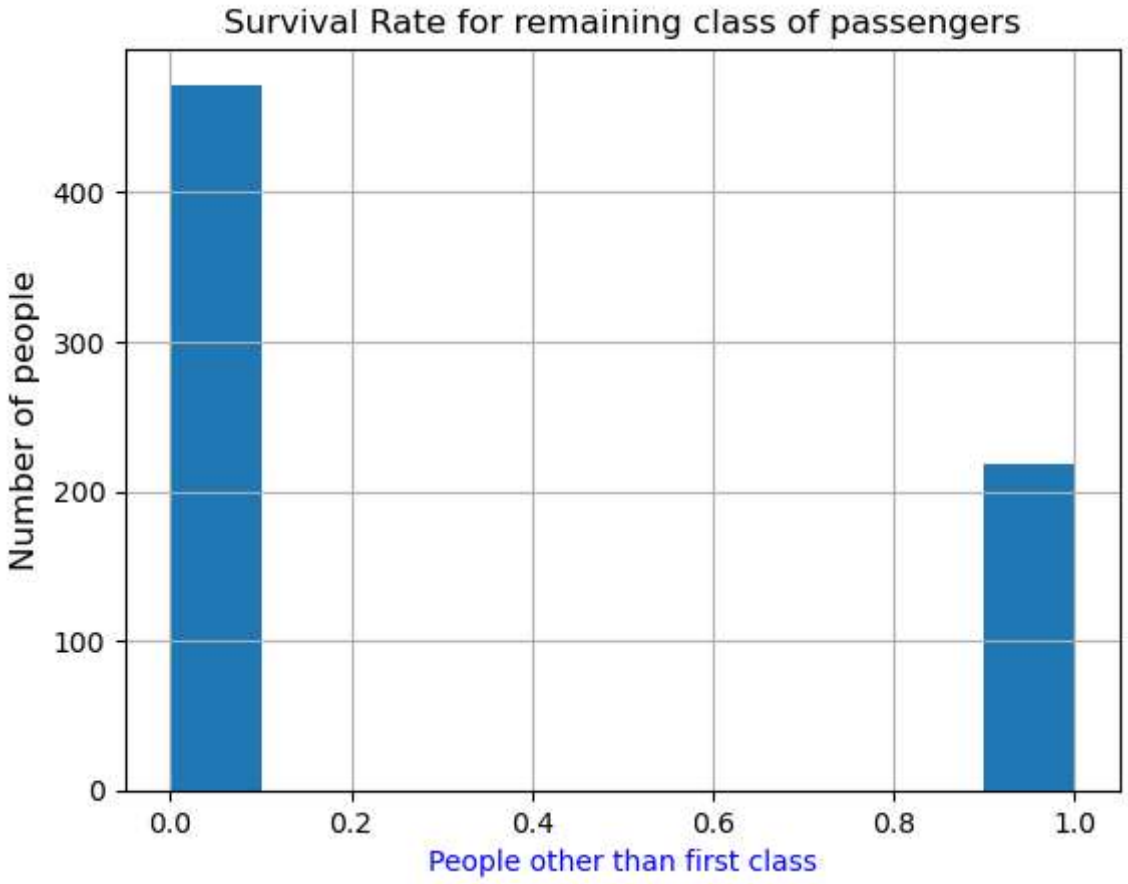
In [73]:
```
first_class_df = df[(df['Pclass'] == '1')]
second_third_class_df = df[(df['Pclass'] != '1')]
```

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

In [76]:
```
# Your code here
first_class_df['Survived'].hist()
plt.xlabel('First class people', color='blue')
plt.ylabel('Number of people', fontsize=12)
plt.title('Survival Rate for First class passengers');
```



In [77]:
```
second_third_class_df['Survived'].hist()
plt.xlabel('People other than first class', color='blue')
plt.ylabel('Number of people', fontsize=12)
plt.title('Survival Rate for remaining class of passengers');
```



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](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.query.html).

```
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.
```

In [82]:
```
query_string = 'PassengerId >= 500'
high_passenger_number_df = df.query(query_string)
high_passenger_number_df.head()
```

Out[82]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
499	500	0	3	Svensson, Mr. Olof	male	24.0	0	0	350035	7.7958	Na
500	501	0	3	Calic, Mr. Petar	male	17.0	0	0	315086	8.6625	Na
501	502	0	3	Canavan, Miss. Mary	female	21.0	0	0	364846	7.7500	Na
502	503	0	3	O'Sullivan, Miss. Bridget Mary	female	NaN	0	0	330909	7.6292	Na
503	504	0	3	Laitinen, Miss. Kristina Sofia	female	37.0	0	0	4135	9.5875	Na

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?*)

In [85]:
```
female_children_df = df.query("Sex == 'female' and Age < 15")
female_children_df.head()
```

Out[85]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	Na
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0	0	350406	7.8542	Na
24	25	0	3	Palsson, Miss. Torborg Danira	female	8.0	3	1	349909	21.0750	Na
39	40	1	3	Nicola-Yarred, Miss. Jamila	female	14.0	1	0	2651	11.2417	Na

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`.

In [86]:
```
df = df.eval('Age_x_Fare = Age * Fare')
df.head()
```

Out[86]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N

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 (https://github.com/yhat/pandasql) 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.

In [87]:
```
!pip install pandasql
```

```
Requirement already satisfied: pandasql in /opt/saturncloud/envs/saturn/lib/p
ython3.10/site-packages (0.7.3)
Requirement already satisfied: sqlalchemy in /opt/saturncloud/envs/saturn/li
b/python3.10/site-packages (from pandasql) (1.4.46)
Requirement already satisfied: pandas in /opt/saturncloud/envs/saturn/lib/pyt
hon3.10/site-packages (from pandasql) (1.5.2)
Requirement already satisfied: numpy in /opt/saturncloud/envs/saturn/lib/pyth
on3.10/site-packages (from pandasql) (1.24.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /opt/saturncloud/env
s/saturn/lib/python3.10/site-packages (from pandas->pandasql) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/saturncloud/envs/saturn/l
ib/python3.10/site-packages (from pandas->pandasql) (2022.7)
Requirement already satisfied: greenlet!=0.4.17 in /opt/saturncloud/envs/satu
rn/lib/python3.10/site-packages (from sqlalchemy->pandasql) (2.0.1)
Requirement already satisfied: six>=1.5 in /opt/saturncloud/envs/saturn/lib/p
ython3.10/site-packages (from python-dateutil>=2.8.1->pandas->pandasql) (1.1
6.0)
```

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`.

In [90]:
```
# 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 (https://github.com/yhat/pandasql).

In [91]:
```
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 [94]:
```
q = """
SELECT name
from df
LIMIT 10
"""

passenger_names = pysqldf(q)
passenger_names
```

Out[94]:

	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.

In [95]:
```
q2 = """
SELECT Name, Fare
FROM df
WHERE Sex == 'male' AND Survived == 1
LIMIT 30
"""

sql_surviving_males = pysqldf(q2)
sql_surviving_males
```

Out[95]:

	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).

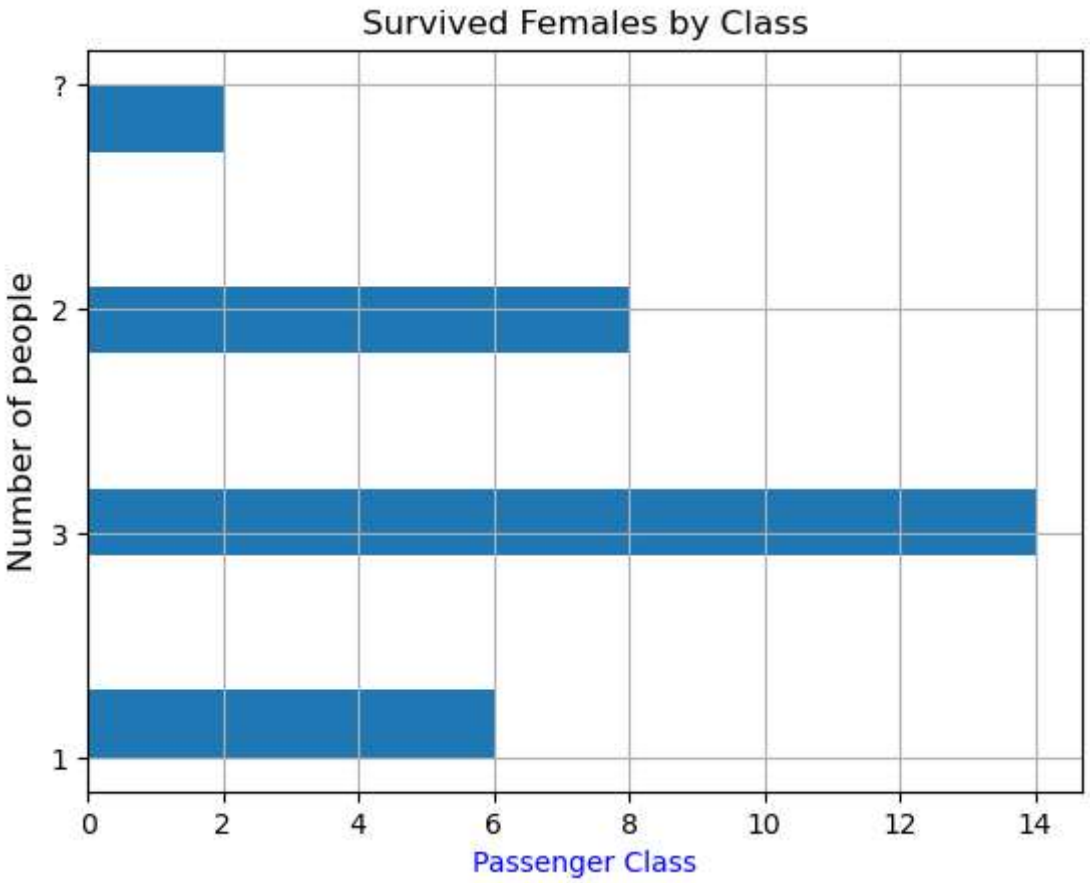
Although it's outside the scope of this lab, it's also worth noting that both `pandas` and `pandasql` provide built-in functionality for join operations, too!

Practical Example: SQL in Pandas

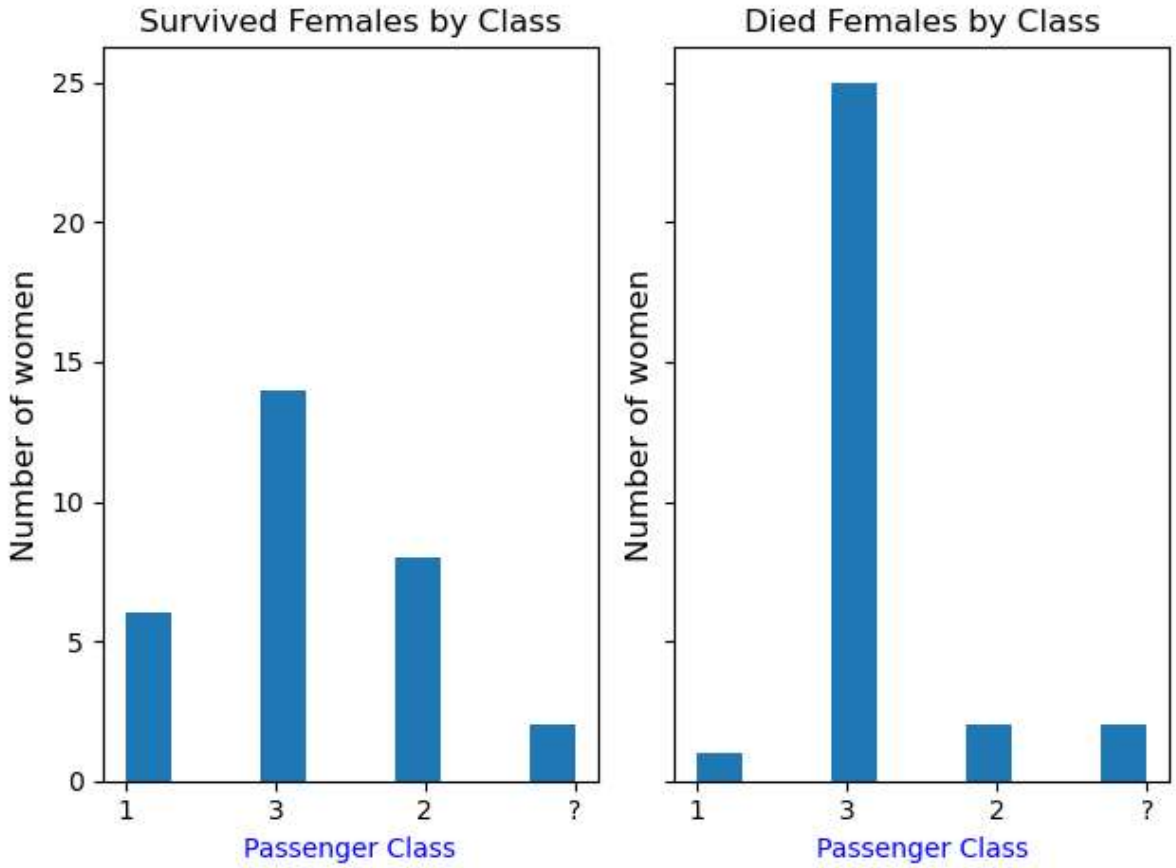
In the cell below, create 2 separate DataFrames using `pandasql`. One should contain the Pclass of all female passengers that survived, and the other should contain the Pclass of all female passengers that died.

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

In [102]:
```
# Write your queries in these variables to keep your code well-formatted and re
q3 ="""
SELECT Name, Pclass
FROM df
WHERE Sex == 'female' AND Survived == 1
LIMIT 30
"""
q4 = """
SELECT Name, Pclass
FROM df
WHERE Sex == 'female' AND Survived == 0
LIMIT 30
"""

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


survived_females_by_pclass_df['Pclass'].hist(orientation='horizontal')
plt.xlabel('Passenger Class', color='blue')
plt.ylabel('Number of people', fontsize=12)
plt.title('Survived Females by Class');
plt.show()
#died_females_by_pclass_df['Pclass'].hist(orientation='horizontal')
#plt.xlabel('Passenger Class', color='blue')
#plt.ylabel('Number of people', fontsize=12)
#plt.title('Died Females by Class');
#plt.show()
## Create and label the histograms for each below!
#survived_females_by_pclass_df.hist('Pclass', xlabel='Passenger Class', ylabel=
#plt.show()

# Plot the histogram for died females by class
#died_females_by_pclass_df.hist('Pclass', xlabel='Passenger Class', ylabel='Fre
#plt.show()
```


Survived Females by Class
Number of people
?
2
3
1
0
2
4
6
8
10
12
14
Passenger Class

In [109]:
```
q3 ="""
SELECT Name, Pclass
FROM df
WHERE Sex == 'female' AND Survived == 1
LIMIT 30
"""
q4 = """
SELECT Name, Pclass
FROM df
WHERE Sex == 'female' AND Survived == 0
LIMIT 30
"""

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


fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True, sharey=True)

# Plot the histogram for survived females by class on the first axes
ax1.hist(survived_females_by_pclass_df['Pclass'], orientation='vertical')
ax1.set_xlabel('Passenger Class', color='blue')
ax1.set_ylabel('Number of women', fontsize=12)
ax1.set_title('Survived Females by Class')

# Plot the histogram for died females by class on the second axes
ax2.hist(died_females_by_pclass_df['Pclass'], orientation='vertical')
ax2.set_xlabel('Passenger Class', color='blue')
ax2.set_ylabel('Number of women', fontsize=12)
ax2.set_title('Died Females by Class')

# Adjust the spacing between the plots
plt.tight_layout()

# Show the plots
plt.show()
```



Summary

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