# **pandas installation in Jupyter notebook**

* pip install pandas

**PIP is a package management system used to install and manage software packages/libraries written in Python**. These files are stored in a large “**online repository**” termed as **Python Package Index (PyPI).** Pandas can be installed using PIP by the use of the following command: pip install pandas.

**Python Data Analysis Library**

**Pandas stand for “Python Data Analysis Library”.** According to the Wikipedia page on Pandas, “the name is derived from the term “panel data”, an econometrics term for multidimensional structured data sets.” But I think it's just a cute name to a super-useful Python library!

**NumPy** is a library for Python that adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. **Pandas** is a high-level data manipulation tool that is built on the NumPy package.

* pip install pandasql

**pandasql** **allows you to query pandas DataFrames using SQL syntax**. It works similarly to sqldf in R. pandasql seeks to provide a more familiar way of manipulating and cleaning data for people new to Python or pandas.

* import pandas as pd
* import pandasql as ps

# **pandas version check in python**

* print(pd.\_\_version\_\_)

1.4.2

# **Shortcuts**

Crtl + Enter- Run and Execute

Click A - Insert a cell above the selected cell

Click B - Insert a cell below the selected cell

Click d d –To delete a cell

For this module we will be working with Titanic dataset. It is a famous dataset which has the data of Titanic Disaster that happened in 1912. It records Name, Age, Gender, Ticket etc. of each passenger and also has a flag saying if a person survived the disaster or not. To know more about this dataset [click here](https://www.kaggle.com/c/titanic).  
We will start with importing this data in pandas and then execute the SQL queries in pandasql. Do not worry if you do not know pandas. All you need to focus on is the SQL code.

* titanic\_disaster\_dataset = **pd.read\_csv**("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

print(titanic\_disaster\_dataset.shape, titanic\_disaster\_dataset[['PassengerId']].**drop\_duplicates().shape**)

titanic\_disaster\_dataset.**head()**

* #Giving an alias name to the dataset and using it in "SELECT" statement too

ps.sqldf(""" **SELECT**

df.\*

FROM titanic\_disaster\_dataset AS **df**

""")

* #Taking only Passenger ID and name from the dataset and also renaming the two chosen columns

ps.sqldf("""

SELECT

df.PassengerId AS ID\_of\_Passenger,

df.Name AS Name\_of\_Passenger

FROM titanic\_disaster\_dataset AS df

**LIMIT** 5 """)

\*\*\* **LIMIT** is used in SQL, but **TOP** is used in Teradata (example: select Top 3 \* from <tablename>)

* ps.sqldf("""SELECT

df.\*

FROM titanic\_disaster\_dataset AS df

**WHERE** df.age IS NULL""")

* ps.sqldf("""

SELECT

df.\*

FROM titanic\_disaster\_dataset AS df

WHERE df.age **BETWEEN** 5 **AND** 10 """)

\*\*\***Between and is Inclusive of 5 and 10**

* #You can get non-null rows by using "IS NOT NULL"

ps.sqldf("""

SELECT

df.\*

FROM titanic\_disaster\_dataset AS df

WHERE df.age **IS NOT NULL**""")

* #Selecting everything from the dataset for passengers where (age is below 1 year and sex is female) OR

#(age is above 70 year and Pclass is 1)

ps.sqldf("""

SELECT

df.\*

FROM titanic\_disaster\_dataset AS df

**WHERE** (df.age < 1 **AND** df.Sex = "female")

**OR** (df.age > 70 **AND** df.Pclass = 1)""")

* #Selecting everything from the dataset for passengers where Age is one of 65, 70, 71

ps.sqldf("""SELECT

df.\*

FROM titanic\_disaster\_dataset AS df WHERE df.Age **IN** (65, 70, 71)""")

For text/string columns, a powerful way of filtering is using "LIKE" keyword. It works by the specifying the substring you want in a string and surrounding the substring with '%'s. Here is how it looks with examples below -

* For Names starting with "ABC" - Name LIKE "ABC%"
* For Names ending with "ABC" - Name LIKE "%ABC"
* For Names which contain "ABC" - Name LIKE "%ABC%"
* As you can see **the "LIKE" condition is not case sensitive** i.e. both 'mrs.' and 'MRS.' will be found with the same search string
* ps.sqldf("""

SELECT

df.\*

FROM titanic\_disaster\_dataset AS df

WHERE df.Name **LIKE** "%MRS.%"

LIMIT 5""")

**SUBSTR()** function can be used to take out a part of a string from the text columns in a dataset. SUBSTR() function requires three inputs - **SUBSTR(*column\_name, starting\_point, number\_of\_characters*).** Here is an example where we output 4 characters, starting from the second character from "Name" column .

* #Using SUBSTR() to take out a substring

ps.sqldf(""" SELECT

df.Name,

**SUBSTR(df.Name,2, 4)** AS out

FROM titanic\_disaster\_dataset AS df

LIMIT 5

""")

**LOWER() and UPPER()** can be used to convert text columns to lower and upper case, respectively.

* #Using SUBSTR() to take out a substring

ps.sqldf("""SELECT

df.Name,

**LOWER**(df.Name) AS lcase\_name,

**UPPER**(df.Name) AS ucase\_name

FROM titanic\_disaster\_dataset AS df

LIMIT 5""")

You can get all the unique values in a column using "**DISTINCT**" keyword

* #Getting all distinct values of "Embarked" column

ps.sqldf("""

SELECT

**DISTINCT** df.Embarked

FROM titanic\_disaster\_dataset AS df""")