

✓ Basic Pandas Concepts

Some very basic Pandas and python concepts to review.

✓ Import the pandas package

```
import pandas as pd
```

✓ Create a simple DataFrame

- syntax: `pd.DataFrame({'column1': value1, 'column2': value2, 'column3': value3})`

You can have anything as column names and anything as values.

The only requirement is to have all value lists being of equal length (all are of length 3 in this example)

There are many ways to create a data frame and you will see some more during the course. All of them can be seen documented [here](#).

```
df = pd.DataFrame({'name': ['Bob', 'Jen', 'Tim'],  
                  'age': [20, 30, 40],  
                  'pet': ['cat', 'dog', 'bird']})
```

df



	name	age	pet
0	Bob	20	cat
1	Jen	30	dog
2	Tim	40	bird

Next steps: [View recommended plots](#) [New interactive sheet](#)

✓ View the column names and index values

The index is one of the most important concepts in pandas.

Each dataframe has only a single index which is always available as `df.index` and if you do not supply one (as we did not for this dataframe) a new one is made automatically.

Indexes define how to access rows of the dataframe.

The simplest index is the range index but there are more complex ones like interval index, datetime index and multi index.

We will explore indexes more in depth during the course of this lecture.

```
print(df.columns)  
print(df.index)
```

```
Index(['name', 'age', 'pet'], dtype='object')
RangeIndex(start=0, stop=3, step=1)
```

✓ Select a column by name in 2 different ways

These two ways are equivalent and can be used interchangeably almost always.

The primary exception is when the name of the column contains spaces. If for example we had a column called "weekly sales" we have to use `df['weekly sales']` because `df.weekly sales` is a syntactic error.

```
print(df['name'])
print(df.name)
```

```
0    Bob
1    Jen
2    Tim
Name: name, dtype: object
0    Bob
1    Jen
2    Tim
Name: name, dtype: object
```

✓ Select multiple columns

To select multiple columns we use `df[columns_to_select]` where `columns_to_select` are the columns we are interested in given as a simple python list. As the result we will get another data frame.

This is the equivalent of listing columns names in `SELECT` part of a sql query.

```
df[['name', 'pet']]
```

```
0    Bob  cat
1    Jen  dog
2    Tim  bird
```

✓ Select a row by index

Regular selection of rows goes via its index. When using range indices we can access rows using integer indices but this will not work when using datetime index for example.

We can always access any row in the dataframe using `.iloc[i]` for some integer `i`.

The result is a series object from which we can access values by using column indexing.

```
df.iloc[0]
```



	0
name	Bob
age	20
pet	cat

dtype: object

✓ Sort Function

- pandas.pydata.org
- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html

✓ Sort the data by pet

There are two ways to sort.

- By index
- By value

By value means sorting according to value in a column.

In this example we sort the rows of the dataframe based on values in 'pet' column.

The parameter `ascending = True` means that we want the rows sorted in ascending order. This is the same as sql 'ASC'. To get descending order use `ascending = False`.

`inplace` is very important and you should always remember it. When `inplace=True` the dataframe is modified in place which means that no copies are made and your previous data stored in the dataframe is lost. By default `inplace` is always `False`. When it is `false` a copy is made of your data and that copy is sorted and returned as output.

The output of `sort_values` is always a dataframe returned but the behaviour depends strongly on the `inplace` parameter.

```
df.sort_values('pet',inplace=True, ascending=True)
```

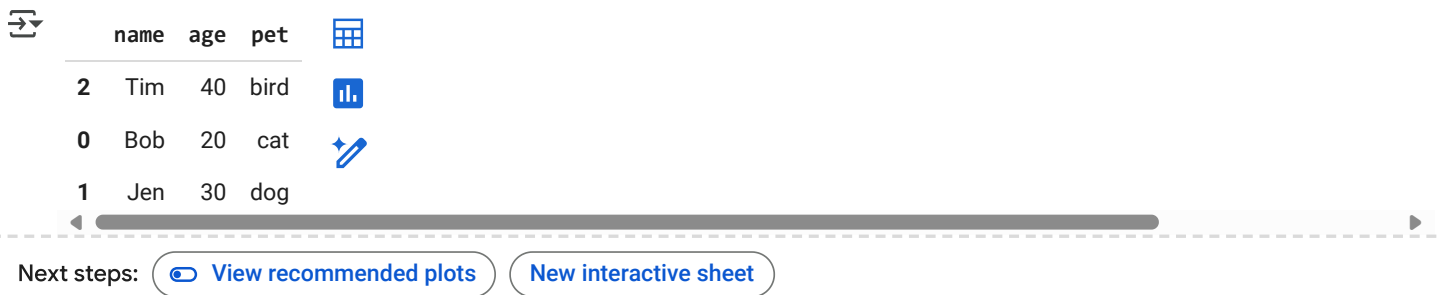
✓ Indexing with DataFrames

Everything we discussed about indexing in numpy arrays applies to dataframes as well.

DataFrames are very similar to 2d-arrays with the main exception being that in DataFrames you can index using strings (column names).

✓ View the index after the sort

```
df
```



	name	age	pet
2	Tim	40	bird
0	Bob	20	cat
1	Jen	30	dog

Next steps: [View recommended plots](#) [New interactive sheet](#)

▼ Difference between loc and iloc

- `.loc` selection is based on the value of the index. For example if the index was categorical we could index via some category.
- `.iloc` selection is **always** based on integer positions. When using `iloc` we are treating the dataframe as 2d-array with no special structure compared to the case of `.loc`

```
df.loc[0] #index based
```

```
df.iloc[0] #relative position based indexing
```

▼ Use iloc to select all rows of a column

This will select all rows of the second column.

Remember : = ::1

First index is always row and second is always column when dealing with dataframes.

```
df.iloc[:,2]
```



	pet
0	cat
1	dog
2	bird

dtype: object

▼ Use iloc to select the last row

```
df.iloc[-1,:]
```



	2
name	Tim
age	40
pet	bird

dtype: object

✓ Basic Pandas Functionality

Before we learn about what Pandas can do, we need to first import some data

Importing Data

Python allows you to connect to any type of database. To make this easy for newbies, we've create a notebook to help you connect to the Strata Scratch platform and pull data. Use the notebook below to pull data from our database.

[Connect to Strata Scratch with Python](#)

✓ Install the Database Module

The code below installs a postgres database module to allow our notebook to connect to the Strata Scratch database

```
!pip install psycopg2
```

➡ Requirement already satisfied: psycopg2 in /usr/local/lib/python3.11/dist-packages (2.9.10)

✓ Import Required Modules

Import a few required modules that enables us to query data and perform analytics

```
import numpy as np
import pandas as pd
import psycopg2 as ps
```

✓ Connect to Strata Scratch

Make sure to enter your username and database password. Your database password is not the same as your login password. You can find your database password in the Profile tab once logged into Strata Scratch.

```
host_name = 'db-strata.stratascratch.com'
dbname = 'db_strata'
port = '5432'
user_name = '' #enter username
pwd = '' #enter your database password found in the profile tab in Strata Scratch

try:
    conn = ps.connect(host=host_name,database=dbname,user=user_name,password=pwd,port=port)
except ps.OperationalError as e:
    raise e
else:
    print('Connected!')
```

```
-----
OperationalError                                Traceback (most recent call last)
<ipython-input-17-2a2be5c6e876> in <cell line: 0>()
      8     conn = ps.connect(host=host_name,database=dbname,user=user_name,password=pwd,port=port)
      9 except ps.OperationalError as e:
--> 10     raise e
     11 else:
     12     print('Connected!')
```

1 frames

```
/usr/local/lib/python3.11/dist-packages/psycopg2/_init_.py in connect(dsn, connection_factory,
cursor_factory, **kwargs)
    120
    121     dsn = _ext.make_dsn(dsn, **kwargs)
--> 122     conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
    123     if cursor_factory is not None:
    124         conn.cursor_factory = cursor_factory

OperationalError: could not translate host name "db-strata.stratascratch.com" to address: Name or service
not known
```

✓ Pull the Titanic Dataset From Strata Scratch

✓ Enter SQL code below to pull the dataset you're interested in

If you get an error, it likely means that the connection timed out. Try connecting to Strata Scratch again before executing the code below.

A list of datasets is found in SQL LAB in Strata Scratch.

```
#Write SQL below to pull datasets
cur = conn.cursor()
cur.execute("""
            SELECT * FROM titanic;
            """)
data = cur.fetchall()
colnames = [desc[0] for desc in cur.description]
conn.commit()

#create the pandas dataframe
data = pd.DataFrame(data)
data.columns = colnames

#close the connection
cur.close()
```


```
-----
NameError                                Traceback (most recent call last)
<ipython-input-18-2a4c24da3de6> in <cell line: 0>()
      1 #Write SQL below to pull datasets
----> 2 cur = conn.cursor()
      3 cur.execute("""
      4         SELECT * FROM titanic;
      5         """)

NameError: name 'conn' is not defined
```

✓ Check To See If Your Pulled The Dataset

The Titanic dataset should be in a pandas dataframe named `data`


```
from google.colab import files
uploaded =files.upload()
```

 Choose Files Titanic-Dataset.csv

- **Titanic-Dataset.csv**(text/csv) - 61194 bytes, last modified: 3/18/2025 - 100% done

Saving Titanic-Dataset.csv to Titanic-Dataset (1).csv

```
data=pd.read_csv('Titanic-Dataset.csv')
data.head()
```



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
				Futrelle, Mrs.								

Next steps: [View recommended plots](#) [New interactive sheet](#)

✓ Basic Pandas Functionality

Now that we imported some data, let's take a look at what Pandas can do

✓ Investigate the first few rows of data

The `head` method by default prints the first 5 rows of your dataframe.

If you pass it a parameter `n` it will print first `n` rows.

The docs are [here](#)

```
data.head()
```



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
				Futrelle, Mrs.								

Next steps:

 [View recommended plots](#)

[New interactive sheet](#)

Investigate the last 10 rows of data

tail is similar to head except it prints the last `n` rows.

```
data.tail(10)
```


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958	NaN	S
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	S
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	C
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr.	male	32.0	0	0	370376	7.7500	NaN	C

▼ Investigate the data types in the DataFrame

This method will tell you the types of columns.

Types are automatically inferred by pandas and usually you do not have to worry about them.

[docs](#)

```
data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 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    int64

```

```

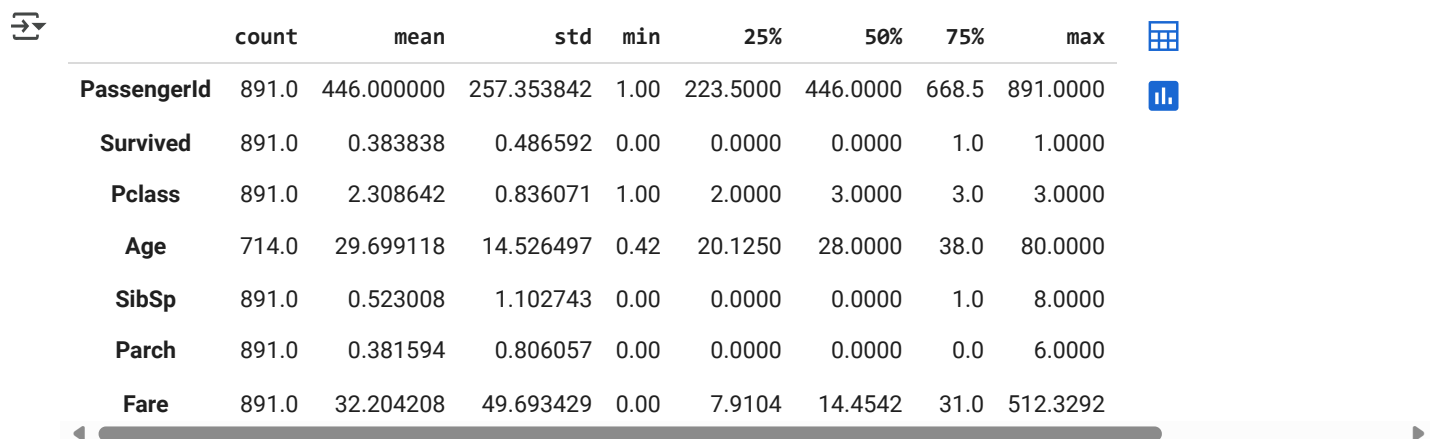
3  Name      891 non-null  object
4  Sex       891 non-null  object
5  Age       714 non-null  float64
6  SibSp     891 non-null  int64
7  Parch     891 non-null  int64
8  Ticket   891 non-null  object
9  Fare      891 non-null  float64
10 Cabin    204 non-null  object
11 Embarked  889 non-null  object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

```

▼ Get some summary statistics

To learn more about describe visit [this link](#)

```
data.describe().T
```



The table displays summary statistics for the Titanic dataset. It includes a header row with columns: count, mean, std, min, 25%, 50%, 75%, and max. The rows represent different variables: PassengerId, Survived, Pclass, Age, SibSp, Parch, and Fare. To the right of the table, there are two icons: a grid icon and a bar chart icon. A horizontal scrollbar is located at the bottom of the table.

	count	mean	std	min	25%	50%	75%	max
PassengerId	891.0	446.000000	257.353842	1.00	223.5000	446.0000	668.5	891.0000
Survived	891.0	0.383838	0.486592	0.00	0.0000	0.0000	1.0	1.0000
Pclass	891.0	2.308642	0.836071	1.00	2.0000	3.0000	3.0	3.0000
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

▼ Filtering Dataframes

You can filter data based on the columns and values in the dataframe


▼ Filter the data for men

There are two pieces of the puzzle here:

- `data.sex=='male'` will give a boolean array where True means that row has a column called sex which has value 'male'. This numpy array is called the predicate.
- `data[data.sex=='male']` will give back all rows for which the predicate holds true.

The result of this filter is a dataframe with same columns as the input dataframe.

```
data[data.Sex=='male']
```



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
...
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
889	890	1	1	Behr, Mr. Karl	male	26.0	0	0	111369	30.0000	C148	C

Filter the ages for the men

Again there are two important parts:

- `data.sex=='male'` is the predicate as before
- `data.age` means taking the values for the age column, and `data.age[data.sex=='male']` means taking all ages which are related to male rows.

The result of this is pandas series **not** a dataframe.

```
data.Age[data.Sex=='male']
```



	Age
0	22.0
4	35.0
5	NaN
6	54.0
7	2.0
...	...
883	28.0
884	25.0
886	27.0
889	26.0
890	32.0

577 rows × 1 columns

dtype: float64

✓ Adding methods to filters

A method is a function and is used frequently when analyzing data in Pandas. There are countless Pandas methods. We'll go over a few of the basic ones to show how you can use methods to quickly analyze your data.


✓ How many men and women were on the Titanic?

The pipeline always goes the same way

- Predicate is evaluated
- Data is filtered according to a predicate
- An aggregate value is computed after the filtering.


The count method simply counts the number of frames in the dataframe.

```
data.Sex[data.Sex=='male'].count()
```



```
np.int64(577)
```

```
data.Sex[data.Sex=='female'].count()
```



```
np.int64(314)
```

✓ What was the survival rate for adult men (age>=18)

Here we combine predicates using the and operator (&).

This operator applies the logical and operation between elements at matching positions.

For example:

- `x = np.array([True, False, True, True])`
- `y = np.array([False, True, False, True])`
- will give `x & y = np.array([True & False, False & True, True & False, True & True])`.

In the following example we use the or combiner (`|`).

You can combine any two boolean numpy arrays as long as they have the same shape using the `&` and `|` operators.

Combining regular python lists this way does not work.

```
data.Survived[(data.Sex=='male') & (data.Age>=18)].mean()
```

```
np.float64(0.17721518987341772)
```

✓ What was the survival rate for women and children?

The mean method is the same as AVERAGE in SQL.

```
data.Survived[(data.Sex=='female') | (data.Age<18)].mean()
```

```
np.float64(0.6881720430107527)
```

✓ Use groupby to compare the survival rates of men and women

The `groupby` method is one of the most important tools you will use in your day to day work.

It's main input parameter is either a string denoting a column name or a list of strings denoting a list of column names.

It's output is a GroupBy object which is very similar to a dataframe.

The operation of `groupby` is the same as SQL GROUPBY.

For more info see the [docs](#).

```
data.groupby('Sex')['Survived'].mean()
```

```

Survived
Sex
female  0.742038
male    0.188908


dtvne: float64

```

✓ Create a DataFrame with groupby

```
new = data.groupby(['Sex', 'Pclass'])[['Survived', 'Age']].mean()
```

```
new
```

		Survived	Age	
Sex	Pclass			
female	1	0.968085	34.611765	
	2	0.921053	28.722973	
	3	0.500000	21.750000	
male	1	0.368852	41.281386	
	2	0.157407	30.740707	
	3	0.135447	26.507589	

Next steps: [View recommended plots](#) [New interactive sheet](#)

Importing and Exporting Data with Pandas

Pandas has easy to use functions for importing and exporting different data types:


- CSV Files
- Excel Worksheets
- Queries from Databases

Strata Scratch notebooks will exclusively be import data from our platform so we will not be covering other import techniques.

✓ More Basic Pandas Exercises


✓ What was the average age of the survivors?

```
data.Age[data.Survived].mean()
```

```
 np.float64(28.141414141414142)
```

✓ What was the combined survival rate of both children (age less than 18) and seniors (age greater than 60)?

```
data.Survived[(data.Age<=18)|(data.Age>=60)].mean()
```

```
 np.float64(0.4666666666666667)
```

```
new = data.groupby(['Survived'])[['Age']].mean()
new
```



ΔΡΕ

