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']]
```

	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
```

```
          0
name    Bob
age      20
pet     cat

dtype: object
```

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

	2
name	Tim
age	40
pet	bird

dtype: object

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
2	bird
0	cat
1	dog

dtype: object

Use iloc to select the last row

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

	1
name	Jen
age	30
pet	dog

dtype: object

bold text# 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

`[ ]` ↳ 11 cells hidden

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()
```

Investigate the last 10 rows of data

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

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

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()
```

Get some summary statistics

To learn more about describe visit this link

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

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']
```

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']
```

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 evalatued
- 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()
```

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

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()
```

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()
```

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()
```

Create a DataFrame with groupby

```
new = data.groupby(['sex','pclass'])['survived','age'].mean()
new
```

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

[ ] ↳ 8 cells hidden

SUPPLEMENTARY ACTIVITY

Answer the following exercises using Pandas on your local device:

1. Given the following data structures:

sales = [100,130,119,92,35]

customer_account = ['B100','J101','X102','P103','R104']

city = ['BOS','LA','NYC','SF','CHI']

1.1. Create a DataFrame with the data above

```
given1 = pd.DataFrame({
    "sales": [100,130,119,92,35],
    "customer_account":['B100','J101','X102','P103','R104'],
    "city": ['BOS','LA','NYC','SF','CHI']})
```

```
given1
```

	sales	customer_account	city
0	100	B100	BOS
1	130	J101	LA
2	119	X102	NYC
3	92	P103	SF
4	35	R104	CHI

Next steps: View recommended plots | New interactive sheet

1.2. What is the name of the first column?

[Answer] **"Sales"** is the name of first column.

1.3. Sort the DataFrame by city in descending order (check the documentation for sort)

```
given1.sort_values('city',inplace=True, ascending=False)
```

```
given1
```

	sales	customer_account	city
3	92	P103	SF
2	119	X102	NYC
1	130	J101	LA
4	35	R104	CHI
0	100	B100	BOS

Next steps: View recommended plots | New interactive sheet

1.4. Which customer is in the last row of the DataFrame?

[Answer] **B100** is the last customer.

1.5 Reorder the columns with customer in the first column

```
given1[["customer_account", "sales", "city"]]
```

	customer_account	sales	city
3	P103	92	SF
2	X102	119	NYC
1	J101	130	LA
4	R104	35	CHI
0	B100	100	BOS

2. Load the Titanic Dataset Download Titanic Datasetand answer the following questions:

```
import pandas as pd
import numpy as np

given2 = pd.read_csv("Titanic-Dataset.csv")
```

2.1. What was the average age of the survivors?

```
mean_age = given2["Age"].mean()
print("Average age of survivors is", round(mean_age,2), ".")
```

```
Average age of survivors is 29.7 .
```

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

```
given2["Age"].isna().sum()
age_mean = mean_age
given2["Age"].fillna(age_mean, inplace=True)
```

```
<ipython-input-62-df29db39cc0a>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inp


  given2["Age"].fillna(age_mean, inplace=True)
```

```
num_seniors = len(given2[given2["Age"] > 60])
num_children = len(given2[given2["Age"] < 18])

senior_mean = round(given2[given2["Age"] > 60].Age.mean(), 2)
children_mean = round(given2[given2["Age"] < 18].Age.mean(), 2)

csr = children_mean / num_children
ssr = senior_mean / num_seniors

survival_rate = (csr + ssr) / 2

print(f"Number of seniors (above 60): {num_seniors}")
print(f"Number of children (below 18): {num_children}")
print(f"Survival rate (mean age of children and seniors): {round(survival_rate,2)}")
```

```
Number of seniors (above 60): 22
Number of children (below 18): 113
Survival rate (mean age of children and seniors): 1.54
```

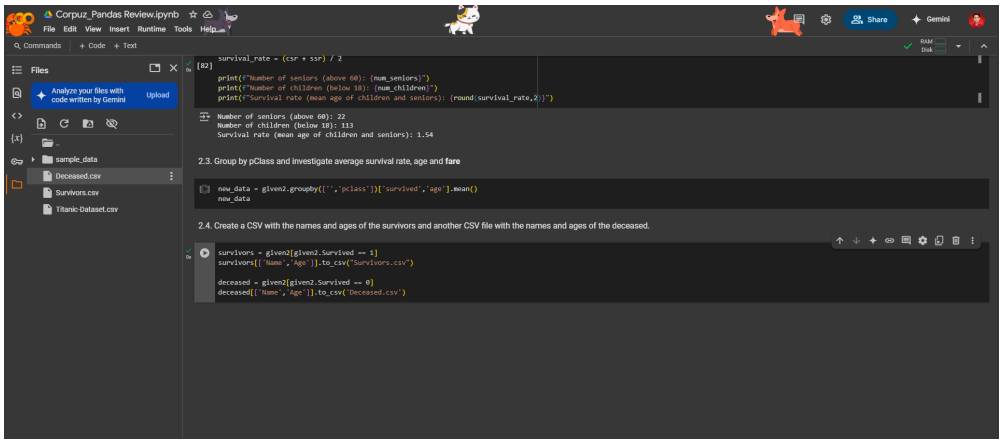
2.3. Group by pClass and investigate average survival rate, age and **fare**

```
grouped = given2.groupby('Pclass').agg(
    avg_survival_rate=('Survived', 'mean'),
    avg_age=('Age', 'mean'),
    avg_fare=('Fare', 'mean')
).reset_index()

print(grouped)
```

```
   Pclass  avg_survival_rate    avg_age   avg_fare
0       1           0.629630  37.048118  84.154687
1       2           0.472826  29.866958  20.662183
2       3           0.242363  26.403259  13.675550
```

2.4. Create a CSV with the names and ages of the survivors and another CSV file with the names and ages of the deceased.

```
survivors = given2[given2.Survived == 1]
survivors[['Name','Age']].to_csv("Survivors.csv")

deceased = given2[given2.Survived == 0]
deceased[['Name','Age']].to_csv('Deceased.csv')
```


Corpuz_Pandas Review.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
RAM
Disk
Files
Analyze your files with code written by Gemini
Upload
..
sample_data
Deceased.csv
Survivors.csv
Titanic-Dataset.csv
survival_rate = (csr + ssr) / 2
[82]
print(f"Number of seniors (above 60): {num_seniors}")
print(f"Number of children (below 18): {num_children}")
print(f"Survival rate (mean age of children and seniors): {round(survival_rate,2)}")
Number of seniors (above 60): 22
Number of children (below 18): 113
Survival rate (mean age of children and seniors): 1.54
2.3. Group by pClass and investigate average survival rate, age and fare
new_data = given2.groupby(['','pclass'])['survived','age'].mean()
new_data
2.4. Create a CSV with the names and ages of the survivors and another CSV file with the names and ages of the deceased.
survivors = given2[given2.Survived == 1]
survivors[['Name','Age']].to_csv("Survivors.csv")
deceased = given2[given2.Survived == 0]
deceased[['Name','Age']].to_csv('Deceased.csv')