ABDELGHAFOR'S VIRTUAL INTERNSHIP

DATA ANALYSIS PROGRAM

SESSION (1)

PREPARED BY : MARK KOSTANTINE

Hello!

Warm greetings to all present. As we gather here today, I am excited to be with you in the

Abdelghafor's Virtual Internships - Data Analysis Program

Congratulation to you all for being selected

Agenda Overview

PANDAS INTRODUCTION

What is Pandas ?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas ?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

PANDAS INTRODUCTION

What Can Pandas Do?

- Pandas gives you answers about the data. Like :
  - Is there a correlation between two or more columns?
  - What is average value?
  - Max value?
  - Min value?
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Installation and Import of Pandas

```
pip install pandas
```

Once Pandas is installed, import it in your applications by adding the **import** keyword

```
import pandas
```

PANDAS INTRODUCTION

Pandas as pd

Pandas is usually imported under the **pd** alias.

```
import pandas as pd
```

Now the Pandas package can be referred to as **pd** instead of pandas.

```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

SERIES

What is a Series ?

- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.

Labels

- If nothing else is specified, the values are labeled with their index number.
- First value has index 0, second value has index 1 etc.
- This label can be used to access a specified value.

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar[0])
```

SERIES

Create Labels

With the **index** argument, you can name your own labels

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

When you have created labels, you can access an item by referring to the label

```
print(myvar["y"])
```

SERIES

Key/Value Objects as Series

You can also use a key/value object, like a dictionary, when creating a Series.

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

Note : The keys of the dictionary become the labels.

To select only some of the items in the dictionary, use the index argument and specify only the items you want

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

SERIES EXERCISE

Insert the correct Pandas method to create a Series

DATAFRAMES

What is a DataFrame ?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

DATAFRAMES

Locate Row

- As you can see from the precious result, the DataFrame is like a table with rows and columns.
- Pandas use the loc attribute to return one or more specified row(s)

```
#refer to the row index:
print(df.loc[0])
```

returns a Pandas Series

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

returns a Pandas DataFrame

DATAFRAMES

Named Indexes

- With the index argument, you can name your own indexes

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

- Use the named index in the **loc** attribute to return the specified row(s)

```
#refer to the named index:
print(df.loc["day2"])
```

DATAFRAMES EXERCISE

Insert the correct Pandas method to create a DataFrame

READING CSV FILES

- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

Tip : use **to_string(**) to print the entire DataFrame.

If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

READING CSV FILES

Max_rows

- The number of rows returned is defined in Pandas option settings.
- You can check your system's maximum rows with the **pd.options.display.max_rows** statement.

```
import pandas as pd

print(pd.options.display.max_rows)
```

You can change the maximum rows number with the same statement

```
import pandas as pd

pd.options.display.max_rows = 9999

df = pd.read_csv('data.csv')

print(df)
```

ANALYZING DATAFRAMES

Viewing the Data

- One of the most used method for getting a quick overview of the DataFrame, is the **head( )** method.
- The **head( )** method returns the headers and a specified number of rows, starting from the top.

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

Note : if the number of rows is not specified, the head() method will return the top 5 rows

- There is also a **tail( )** method for viewing the last rows of the DataFrame.
- The **tail( )** method returns the headers and a specified number of rows, starting from the bottom.

```
print(df.tail())
```

The DataFrames object has a method called **info( )** , that gives you more information about the data set.

```
print(df.info())
```

CLEANING DATA

Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

- **Empty cells**
- **Data in wrong format**
- **Wrong data**
- **Duplicates**

EMPTY CELLS

Remove Rows

- One way to deal with empty cells is to remove rows that contain empty cells.
- This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

Note : By default, the **dropna( )** method returns a new DataFrame, and will not change the original

If you want to change the original DataFrame, use the **inplace = True** argument

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

EMPTY CELLS

Replace Empty Values

- Another way of dealing with empty cells is to insert a new value instead.
- This way you do not have to delete entire rows just because of some empty cells.
- The fillna( ) method allows us to replace empty cells with a value:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

Replace Only For Specified Columns

To only replace empty values for one column, specify the column name for the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)

print(df.to_string())
```

EMPTY CELLS

Replace Using Mean, Median, or Mode

- A common way to replace empty cells, is to calculate the mean, median or mode value of the column.
- Pandas uses the **mean( ) median( )** and **mode( )** methods to calculate the respective values for a specified column

```python
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

df["Calories"].fillna(x, inplace = True)

print(df.to_string())
```

Mean = the average value (the sum of all values divided by number of values)

```python
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].median()

df["Calories"].fillna(x, inplace = True)

print(df.to_string())
```

Median = the value in the middle, after you have sorted all values ascending

```python
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode()[0]

df["Calories"].fillna(x, inplace = True)

print(df.to_string())
```

Mode = the value that appears most frequently.

WRONG FORMAT

Convert Into a Correct Format

- Cells with data of wrong format can make it difficult, or even impossible, to analyze data.
- To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.
- Pandas has a **to_datetime( )** method for this

```
import pandas as pd

df = pd.read_csv('data.csv')

df['Date'] = pd.to_datetime(df['Date'])

print(df.to_string())
```

WRONG FORMAT

Fixing Wrong Data

- "Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".
- Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.
- One way to fix wrong values is to replace them with something else.

```
import pandas as pd

df = pd.read_csv('data.csv')

df.loc[7,'Duration'] = 45

print(df.to_string())
```

WRONG FORMAT

Replacing Values

- For small data sets you might be able to replace the wrong data one by one, but not for big data sets.
- To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

```
import pandas as pd

df = pd.read_csv('data.csv')

for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120

print(df.to_string())
```

REMOVING DUPLICATES

Discovering Duplicates

- Duplicate rows are rows that have been registered more than one time.
- To discover duplicates, we can use the **duplicated( )** method.
- The **duplicated( )** method returns a Boolean values for each row

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.duplicated())
```

Removing Duplicates

- To remove duplicates, use the **drop_duplicates( )** method.

```
import pandas as pd

df = pd.read_csv('data.csv')

df.drop_duplicates(inplace = True)

print(df.to_string())
```

CLEANING DATA EXERCISE

Insert the correct syntax for removing rows with empty cells

TASKS

Choose one of the following Datasets to use what you learnt on it

DATASET 1

https://www.kaggle.com/datasets/prmohanty/pandas-movie-dataset

DATASET 2

https://www.kaggle.com/datasets/melihkanbay/police

DATASET 3

https://www.kaggle.com/datasets/themrityunjaypathak/pandas-practice-dataset

ANY QUESTIONS ?

DATA ANALYSIS PROGRAM

THANK YOU

UPCOMING NEXT WEEK : SESSION (2)