Lecture 6 – Data Wrangling libraries in Python – Pandas

Introduction to Python

efl Data Science Courses

Anjana Cordes

Table of Contents

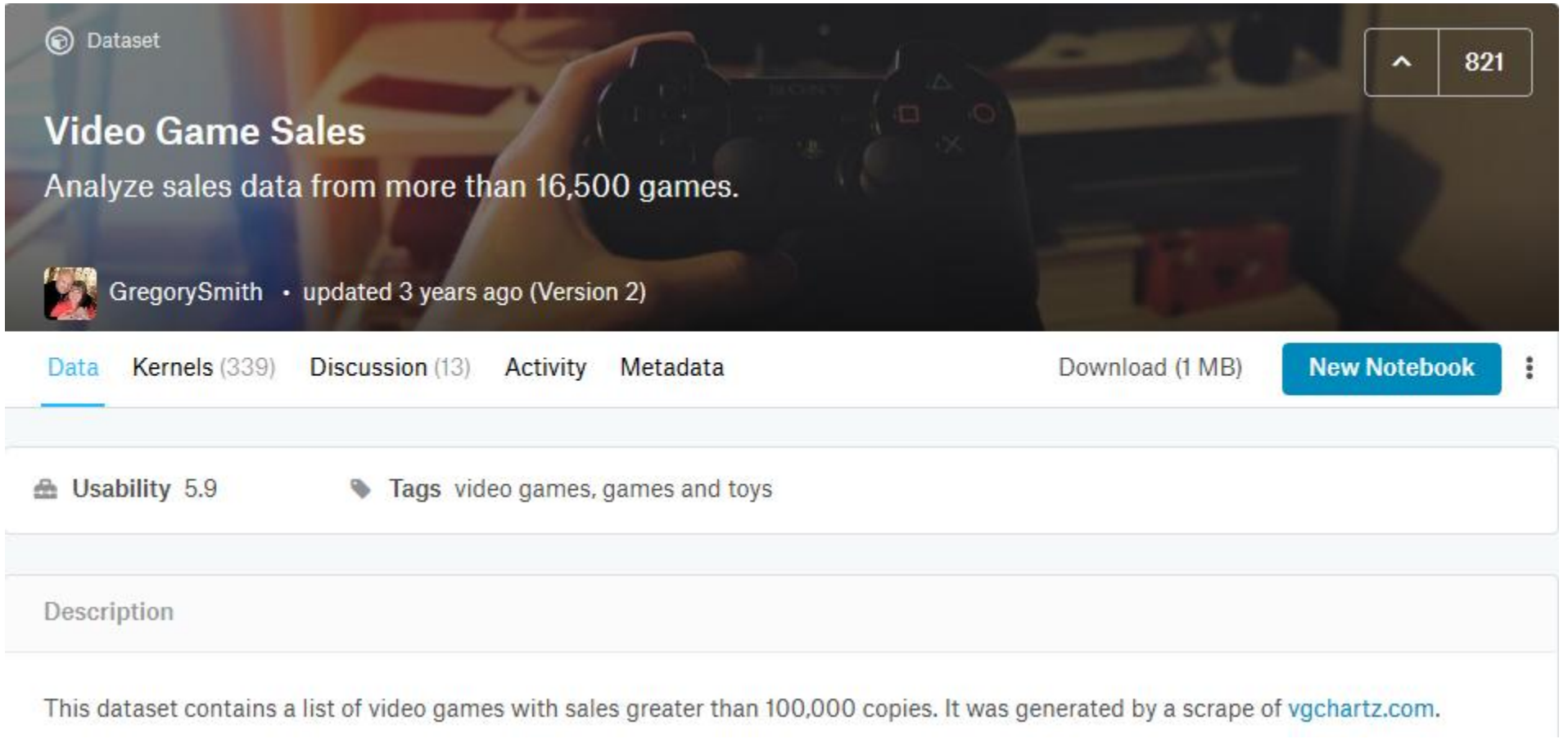
Using large Datasets for Data Analysis

- Until now, you have learned how to:
  - Assign single variables to certain datatypes (int, float, string)
  - Structure data through lists, sets, dictionaries
  - form control structures and loops
  - Construct functions
  - Apply the python standard libraries, especially importing them and reading in files
  - Got to know NumPy, an important numerical library for data analysis
- It is time that we take the next step: reading in large datasets

- For this cause, we will utilize a dataset on video game sales from kaggle.com

Using large Datasets for Data Analysis

- For this cause, we will utilize a dataset on video game sales from kaggle.com



- Do not worry about the scraping part. For now, the dataset is provided for you in your workspace.

NumPy: Lessons learned

- Numpy provides a plethora of fast, mathematical functions
- These functions are typically performed on ndArrays.
- ndArrays are computationally efficient and also referred to as scalars, vectors or matrices.
- The Pandas package and DataFrame inherit many of the positive features from NumPy!

File reading – the old fashioned way

- First use the os package to list all the files in your directory
- If you can not find it, change the directory via os commands or the directory browser on the top right.
- You should find a file called “vgsales.csv”
- Try to read this file in via the csv.reader and print the first 10 rows:

```
with open(“vgsales.csv", "r") as testFile:
    reader = csv.reader(testFile, delimiter=",")
    rows = list(reader)[]
    for i in range(#): #insert the number of rows to print
        print(rows[i])
```

File reading – the old fashioned way

- What now?
- For data science workflows, many statistical and mathematical operations need to be performed
- Simple operations are possible, but the file must stay opened and will be closed after the operations
- Could store the file in nested lists or dictionaries: complex and computationally inefficient

→It seems that the data structures and operations we learned until now are not sufficient for data science workflows on large datasets

→Pandas package provides a solution

Table of Contents

DataFrames – the Pandas way

- Pandas is a comprehensible and powerful library for data analysis
- Allows us to load files of various file types into a data structure called DataFrame
- DataFrame:
  - persistent data structure
  - represent data in a tabular way
  - consists of rows and columns
  - Columns are called 'Series' and are a subordinate

  data structure
  - Rows contain the actual Datapoint/ Rows

DataFrames – the Pandas way

- Take a look at a DataFrame

Column/ Series /Attribute/ Feature

Datapoint/ Row

Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
1	Wii Sports	Wii	2006	Sports	Nintendo	41,49	29.02	3.77	8.46	82.74
2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	33
5	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
6	Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
7	New Super Mario Bros.	DS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
8	Wii Play	Wii	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02
9	New Super Mario Bros. Wii	Wii	2009	Platform	Nintendo	14.59	7.06	4.7	2.26	28.62
10	Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31

DataFrame & Pandas – A first look

- Initialize a pandas dataframe. The general naming convention for dataframes is “df”.

```
df = pd.read_csv('vgsales.csv', sep=',')
```

- First, take a look at the dataframe in the variable explorer.
- Now, let’s see how many functions pandas provides. Then, how many functions Pandas provides for the df especially.

```
print(dir(pd))
print(dir(pd.DataFrame))
```

Remember: If we do not assign the values to a variable, please print your statement on the console to see the result.

- Lots of helpful functions!
- The functions we are going to go through will empower you to:
  - Get an overview over the dataset
  - Get descriptive statistics on the dataset
  - Find first indices for correlations in the dataset

DataFrames – the Pandas way

- Print the columns

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

Column/ Series /Attribute/ Feature

Datapoint/ Row

Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
1	Wii Sports	Wii	2006	Sports	Nintendo	41,49	29.02	3.77	8.46	82.74
2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	33
5	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
6	Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
7	New Super Mario Bros.	DS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
8	Wii Play	Wii	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02
9	New Super Mario Bros. Wii	Wii	2009	Platform	Nintendo	14.59	7.06	4.7	2.26	28.62
10	Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31

```
Columns['Rank', 'Name', 'Platform', 'Year', 'Genre', 'Publisher',
'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales'
```

DataFrames – the Pandas way

- Print the column length. Similar to python lists!

```
print(len(df.columns))
```

Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
1	Wii Sports	Wii	2006	Sports	Nintendo	41,49	29.02	3.77	8.46	82.74
2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	33
5	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
6	Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
7	New Super Mario Bros.	DS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
8	Wii Play	Wii	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02
9	New Super Mario Bros. Wii	Wii	2009	Platform	Nintendo	14.59	7.06	4.7	2.26	28.62
10	Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31

```
11
```

DataFrames – the Pandas way

- The shape may provide us with more information.

```
df_shape = df.shape
print(df_shape)
```

Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
1	Wii Sports	Wii	2006	Sports	Nintendo	41,49	29.02	3.77	8.46	82.74
2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	33
5	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
6	Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
7	New Super Mario Bros.	DS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
8	Wii Play	Wii	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02
9	New Super Mario Bros. Wii	Wii	2009	Platform	Nintendo	14.59	7.06	4.7	2.26	28.62
10	Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31

```
(16598, 11)
```

DataFrames – the Pandas way

- Let's get much more information! We want to see the datatypes, etc…

```
df_info = df.info()
print(df_info)
```

```
RangeIndex: 16598 entries, 0 to 16597
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Rank            16598 non-null  int64
 1   Name            16598 non-null  object
 2   Platform        16598 non-null  object
 3   Year            16327 non-null  float64
 4   Genre           16598 non-null  object
 5   Publisher       16540 non-null  object
 …    …               …               …          …
```

Column Names | No. of Entries for Column | Datatype of Column

DataFrame Datatypes

- In your very first python session, you learned about primitive datatypes, such as int, string, float.
- Complex datatypes are almost always represented as objects. Let's print the column 'Name' with the object datatype, to see if it is really complex and why it is labeled as object.
- Attention! There are two obvious ways to do this.

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

- As we can see, the object datatype actually contains strings.
- Pandas initially tries to assign datatypes to the various columns.
- Sometimes it does not get the correct type. See the Documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/basics.html#dtypes
- Citation: *Pandas uses the object dtype or StringDtype for storing strings.*

DataFrame Datatypes

- Problem solved. But what about Year? Should it not be int? Print the column.

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

- It should be int, yet it has been assigned float values.
- Again, we find our answer in the pandas documentation, see: https://pandas.pydata.org/docs/user_guide/integer_na.html
- Citation: *Because NaN (missing value) is a float, a column of integers with even one missing values is cast to floating-point dtype (see Support for integer NA for more).*

Missing values alter the datatype to float64!

- Missing values are reflected in the entries and datatypes. Let's see:

```
df_info = df.info()
print(df_info)
```

```
RangeIndex: 16598 entries, 0 to 16597
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Rank          16598 non-null  int64
 1   Name          16598 non-null  object
 2   Platform      16598 non-null  object
 3   Year          16327 non-null  float64
 4   Genre         16598 non-null  object
 5   Publisher     16540 non-null  object
 …    …             …               …          …
```

Column Names | No. of Entries for Column | Datatype of Column

DataFrame – Handling missing values

- This explains the issue. Let's check for missing values if it is true for the dataset.
- Try to find the function, which shows all null values. Then apply the .sum() function on top of it, to show the sum of null values per column.

```
df_null_values = df.isnull().sum()
print(df_null_values)
```

- Indeed! We have 271 missing values in the Year column, as well as some in the publisher column.
- We could now decide to drop all the rows, where information is missing, try to fill it with sample data (not very useful in our case) or drop the column (also not useful).

- Since we have a lot of data, we will drop the rows where year and publisher data is missing. Use **pd.DataFrame.dropna** on our **dataframe** to drop the rows. Then print the info on our dataset again.

```
df = df.dropna(axis=0)
df.info()
```

DataFrame – Handling missing values

- The 306 records have been dropped, such that we now only have 16291 records.
- It appears that Year is still a float. Let's convert it to int like this: **df[‘columnName’].astype(‘int64’)**
- See: https://pandas.pydata.org/pandas-docs/stable/getting_started/basics.html#astype
- Then print info again.

```
df['Year'] = df['Year'].astype('int64')
df.info()
```

- Great job! You have successfully cleaned your data from rows with missing values and converted a column to its correct datatype.
- Alternatively, we can load the dataset with **dtype_backend=´numpy_nullable´** to get the right datatype

```
df = pd.read_csv('test.csv', sep=',', dtype_backend="numpy_nullable")
```

DataFrame – Manipulation & Data Analysis

- Let's start with some manipulation and analysis.
- Let's see what the top rows and the bottom rows look like.

```
df.head()
df.tail()
```

- Get access to the 55 record. There are two options.

```
df.loc[54]
df.iloc[54]
```

- While loc searches the index by named labels (such as strings, but also int), iloc searches for row number.
- The differentiation of these two will become much clearer when performing slicing operations.

Remember: If we do not assign the values to a variable, please print your statement on the console to see the result.

DataFrame – loc & iloc

- Remember the slicing operations for strings and lists.
- In the pandas case, the first part of the slice before the comma refers to the records (x-axis), the second to the columns (y-axis): **df.iloc[:x,:y]**
- Let’s perform a slice on the rows until index 3, and columns until index 3.

```
df_slice_1 = df.iloc[:3, :3]
```

- This gives us a 3x3 snapshot of the dataframe, starting at the column 0 (index) and stopping before the column 4 (with index 3). Does it work the same way with loc?

```
df.loc[:3, :3]
```

- Nope. we learned that loc works with 'named' labels. Since the rows have numeric labels, we can change the first part according to the number of instances that we want to keep. Be careful, loc includes the last element of the slice. Let's change the second part to the column label.

```
df.loc[:2, :'Platform']
```

Remember: If we do not assign the values to a variable, please print your statement on the console to see the result.

DataFrame – loc & iloc

- Another example: find the last element 16597 with loc and iloc

```
df.loc[16597]
df.iloc[16597]
```

- Different story: know that iloc works with indices. We do have a first column that is named index, yet is not consistent with our actual dataset index. This is sometimes native to the dataset, sometimes caused by record drops.
- Remember, we dropped a few NA rows. To showcase this, get the element with index 180

```
df.loc[180]
df.iloc[180]
```

- We get a different record for iloc. Why? Due to dropping of NA values, the position of record with index 180 is shifted.

```
df.iloc[179]
```

Remember: If we do not assign the values to a variable, please print your statement on the console to see the result.

DataFrame – loc & iloc

- Another example: find the last element 16597 with loc and iloc

```
df.loc[16597]
df.iloc[16597]
```

- Different story: know that iloc works with indices. We do have a first column that is named index, yet is not consistent with our actual dataset index. This is sometimes native to the dataset, sometimes caused by record drops.
- We can clean up the index with this function.

```
df = df.reset_index(drop=True)
```

- 16597 would have been the last element. Let's try to get it another way. Remember how it works for lists!

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

Remember: If we do not assign the values to a variable, please print your statement on the console to see the result.

Data Analysis in Pandas

- Let’s do some advanced stuff now.
- Retain the a subset of the DataFrame containing the records 10-20 (including 10 and 20) and columns Platform - NA_Sales.
- do it with iloc.

```
df.iloc[10:21,2:7]
```

- Do it with loc.

```
df.loc[10:20,'Platform':'NA_Sales']
```

- write the contents of the last operation to a dict.

```
df_slice = df.loc[10:20,'Platform':'NA_Sales']
Dslice = df_slice.to_dict()
```

- Hmm... this does not look quite right. We want the rows to be the keys, not the columns. Let's see what a **df.transpose** can do. Assign the transposed df_slice to the appropriate variable. Then write it to the dict.

```
df_slice_transposed = df_slice.transpose()
Dslice_transposed = df_slice_transposed.to_dict()
```

Data Analysis in Pandas

- Perfect, this is what we wanted. What actually just happened?
- By transposing the dataframe, we changed the columns to rows and the rows to columns. This can sometimes be very helpful in data analysis, for example if you need to convert the data structure or the size of the data structure.
- Especially in Machine Learning and Deep Learning with Neural Nets, being able to transpose data structures is an invaluable feature.

Column/ Series /Attribute/ Feature

Datapoint/ Row

ID	Name	Salary
0	TestGuy	2000
1	James	2333
2	Alan	3000
3	Ada	3000

df.transpose()

ID	0	1	2	3
Name	Testguy	James	Alan	Ada
Salary	2000	2333	3000	3000

Table of Contents

Descriptive Stats – From np to pd

- Let‘s take a look at some functions for descriptive stats that numpy provides.

```
mymatrix.mean()
mymatrix.max()
mymatrix.min()
np.quantile(mymatrix,0.75)
np.quantile(mymatrix,0.50)
np.quantile(mymatrix,0.25)
np.std(mymatrix)
```

- Try to find these yourself in pandas! Apply them in the given order on the pandas df.

Descriptive Stats – Pandas

- Let‘s take a look at some functions for descriptive stats that numpy provides.

```
df_mean = df.mean(numeric_only=True)
df_max = df.max(numeric_only=True)
df_min = df.min(numeric_only=True)
df_q3 = df.quantile(q=0.75, numeric_only=True)
df_qmed = df.quantile(q=0.50, numeric_only=True)
df_q1 = df.quantile(q=0.25, numeric_only=True)
df_std = df.std(numeric_only=True)
df_count = df.count()
```

- Try to find these yourself in pandas! Apply them in the given order on the pandas df.

Descriptive Stats – Aggregate Functions

- There are also important aggregate functions that deliver the most interesting desc. stats at once.

```
df_description = df.describe()
```

- To find first hints on how features (another name for our attributes or columns that is used in data analysis) correlate, use the correlation function.

```
df_corr = df.corr(numeric_only=True)
```

Data Wrangling with Pandas: Lessons learned

- Pandas is probably the most important library for data analysis
- Numpy on the other hand is the most invaluable library for calculus, statistics and math in general
- Both are very important for python data scientists!
- DataFrames are flexible and useful datastructures for working on small and large datasets

References

- https://pandas.pydata.org/pandas-docs/stable/
- https://docs.scipy.org/doc/
- https://www.kaggle.com/gregorut/videogamesales/downloads/videogamesales.zip/2l