Correlation & Regression with Excel

Video Link : https://www.youtube.com/watch?v=IXHCyhO7DmY

Tool pack used: Data analysis Tool pack

> How to find the corrleation matrix:
>
> > Data → Data Analysis → Correlation &arrr; Select Input Variables → [Done]

> How to do regression analysis:
>
> > Data → Data Analysis → Regression → Select X & Y → [Done]

Data aggregation using pandas_profiling

Video Link : https://www.youtube.com/watch?v=CDwZPie29QQ
Notebook Link :
https://colab.research.google.com/drive/1GZdFIKmPONqrUDFuyducN0Z9zqBvuT4X

To generate pandas_profiling:

```
from pandas_profiling import ProfileReport
prof = ProfileReport(df)
prof.to_file(output_file='output.html')
`
```

To download the html report:

```
prof = ProfileReport(df)
prof.to_file('report.html')
files.download('report.html')
```

Data Cleaning using Openrefine

Video Link : https://www.youtube.com/watch?v=cX_2MkShlJk

Openrefine : OpenRefine is an open-source desktop application for data cleanup and transformation to other formats, an activity commonly known as data wrangling.

> How to cluster a column:
>
> > Drop down menu(Available in the Column) → Facet → Text Facet → Cluster

We can also edit the name of the new cluster formed in this way.

Some details about the Openrefine algorithm:

- **Key collision** is the default clustering algorithm. It is also the most stringent algorithm. It removes the special characters from the text then converts the whole string it into lowercase & then clusters it.
- **Nearest neighbors** [Levenshtein distance] is based on Levenshtein distance(*Number of edits that needs to be done between two strings*)
- **Nearest neighbors** [ppm] if any of the substring matches between 2 strings, it clusters those 2 strings into one.

Scraping using Geocoding API of Open Street Map

Video Link: https://www.youtube.com/watch?v=f0PZ-pphAXE
Notebook:
https://colab.research.google.com/drive/1cKOxgITK8aGoWMZfd2y5PoHo2DRXHF_z

---

Library used: geopy.geocoders
Function used: Nominatim

Some details about the syntax:

> `user_agent`: *An http request header that is sent with each request (defaultuser_agent= 'geopy/2.2.0'_)*

> **To geolocate a query to an address and coordinates**:
>
> > `from geopy.geocoders import Nominatim`
> > `geolocator = Nominatim(user_agent="custom_user_agent")`
> > `location = geolocator.geocode("place")`
> > `print(location.adress)` : *gives full address of the given location*
> > `print(location.latitutude, location.longitude)`: *gives lat & long data of the given location*
> > `print(location.raw)` : *outputs json*

> **To find the address corresponding to a set of coordinates**:
>
> > `from geopy.geocoders import Nominatim`
> > `geolocator =`
> > `Nominatim(user_agent="specify_your_app_name_here")`
> > `location = geolocator.reverse("52.509669, 13.376294")`
> > `print(location.address)`

Image classification using GCP

Video Link : https://www.youtube.com/watch?v=hC47vWauvfk

GCP &rarr, Google Cloud Platform

Tool used: Vison (Auto ML Vison)

> How to classify image using GCP:
>
> > New Dataset → Select the objective of the model →
> > Create a bucket & upload the data → Train the new model
> > We can also directly deploy the model from GCP itself.

Model the data using pycaret

Video Link : https://www.youtube.com/watch?v=WMUt7NOJGbo
Notebook :
https://colab.research.google.com/drive/1Tc70QuKMBCtH6h4fQv6B7HHR8dR8PCE9

Library used: Pycaret, Pandas

> How to setup a model:
>
> > `df = pd.read_csv("dataset")`
> > `model = setup(df, target = "target_variable")` target`
> > is not required for unsupervised models like: NLP, clustering

> How to compare models:
>
> > `best_model = compare_models()` # Returns top performing model

> How to create a model:
>
> > new_model = create_model(MODEL) # lr, lda etc

> How to tune the model:
>
> > `tuned_model = tune_model(new_model)`

Pdf Scraping

Video Link : https://www.youtube.com/watch?v=3Xw9YGh00aM
Notebook:
https://colab.research.google.com/drive/1mNhUTij7LdsjxgcfOKgfsmbFOI526y2t

Libraries used:

- requests: Scraping with Python Requests will allow us to **send HTTP/1.1 requests using Python**. With it, we can add content like headers, form data, multipart files, and parameters via simple Python libraries. https://docs.python-requests.org/en/latest/
- urllib.request: module defines functions and classes which help in opening URLs (mostly HTTP) in a complex world — basic and digest authentication, redirections, cookies and more.
- urllib.parse: This module defines a standard interface to break Uniform Resource Locator (URL) strings up in components (addressing scheme, network location, path etc.), to combine the components back into a URL string, and to convert a "relative URL" to an absolute URL given a "base URL."
- bs4: Scraping with Python Beautiful Soup is a Python library that is used for **web scraping purposes to pull the data out of HTML and XML files**. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner. https://beautiful-soup-4.readthedocs.io/en/latest/#
- Tabula: Tabula allows you to extract that data into a CSV or Microsoft Excel spreadsheet using a simple, easy-to-use interface. Tabula works on Mac, Windows and Linux. Tabula can read pdf files like pandas reads csv files.

> To read pdf file using Tabula:
>
> ```
> tabula.read_pdf(pdf_file_name,  pages='page_number')
> ```

> To convert pdf into csv file:

`from tabula import convert_into`
`tabula.io.convert_into(_input_path_, _output_path_, _output_format='csv'_, _java_options=None_, _**kwargs_)`
Output file will be saved into output_path

Scraping data from web using excel

Video link : https://www.youtube.com/watch?v=OCI6UdpmzRQ

Procedure:

> *Data → New Query → From other sources → From the web*

Some important points about excel scraping:

- Data from the web can be transformed before/after loading the data.
- All the steps performed during the transformation of the data gets listed in the `Applied Steps` box for future reference.
- The loaded data can be refreshed any time to get the updated values (How?)

> We can directly click on `Refresh` option available in the excel to get the latest data
> (or)
> We can *right click* on any of the entries of the table → `Refresh`

Scraping websites with Python

Video link : *https://www.youtube.com/watch?v=TTzcXj92zaw*
Notebook Link:
https://colab.research.google.com/drive/1Kwi14Twb6cnPPu850dKuo1VtTctoBqG5

Libraries Used:

- **BeautifulSoup**: Beautiful Soup is a Python library that is used for **web scraping purposes to pull the data out of HTML and XML files**. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner. https://beautiful-soup-4.readthedocs.io/en/latest/#
- **Requests**: Requests will allow us to **send HTTP/1.1 requests using Python**. With it, we can add content like headers, form data, multipart files, and parameters via simple Python libraries. https://docs.python-requests.org/en/latest/

Some important syntaxes:

To load the webpage:

```
r = requests.get("url")
```

To convert this `Response object` to beautifulsoup object:

```
Soup = BeautifulSoup(r.content)
```

Text sentiment analysis using Python

Video Link : https://www.youtube.com/watch?v=A9WX7HaS1eU
Notebook:
https://colab.research.google.com/drive/1NQ9EhpeJ0DYN8uXtyye1YLcG1YvmjKUA

Library used: TextBlob

> How to do subjectivity analysis:
>
> ```
> data['TextBlob_Subjectivity'] = df["col_name"].apply(lambda x: TextBlob(x).sentiment.subjectivity)
> ```

> How to do polarity analysis:
>
> ```
> data['TextBlob_Polarity'] = df['col_name'].apply(lambda x: TextBlob(x).sentiment.polarity)
> ```

WIkimedia: Wikipedia Scraping

Video Link : https://www.youtube.com/watch?v=b6puvm-QEY0
Notebook:
https://colab.research.google.com/drive/1UZky5JdOn2oMYIkls23WefTaT8VinYyg

---

Libraries Used:

- wikipedia : Wikipedia is a Python library that makes it easy to access and parse data from Wikipedia.

Some important syntax:

To do a Wikipedia search for query

```
wikipedia.search(_query_, _results=10_,
_suggestion=False_)
```

To generate a plain text summary of the page

```
wikipedia.summary(_query_, _sentences=0_, _chars=0_,
_auto_suggest=True_, _redirect=True_)
```

sentences - if set, return the first sentences sentences (can be no greater than 10).
chars - if set, return only the first chars characters

To get a WikipediaPage object for the page

```
wikipedia.page("query")
```

To get the full content

```
print(wikipedia.page("query").content)
```

To get all the URLs of the page

```
print(wikipedia.page("query").url
```

To get all the images

```
print(wikipedia.page("query").images
```