Business Understanding

Since circa 150BC, when the Roman Empire implemented the use of concrete "opus caementicium" in the majority of their impressive construction projects, some still standing today, concrete is now ubuquitous.

Global construction work done is expected to increase to US13.9 trillion in 15years time, up from US$9.7 trillion in 2022. This will be pushed by superpower construction markets in China, the United States and India. Further, the Philippines, Vietnam, Malaysia and Indonesia are anticiptaed to be the fastest growing construction markets during that period Oxford Economics

Selecting the correct amounts of materials for making concrete of the required compressive strength not only has implications on the durability of projects, like the Roman Empire projects, but also on the cost of the projects, or indeed their feasiblity before construction has even begun.

In this project, Bekezela Bobbie Khabo, an AI Engineering student with IBM, aims to harness the power of Deep Learning to address this very challenge through use of the Keras Python API running on Tensorflow to perform Regression on the following dataset.

The data

Depending on the different quantities of the seven constituent materials below, the resulting concrete will have different compressive stregths. Results of compressive strengths depending different compositions of the seven materials are compiled in a CSV format referenced above. The seven materials, called predictors moving forward, are,

1. Cement: in cubic metres

2. Blast Furnace Slag: in cubic metres

3. Fly Ash: in cubic metres

4. Water: in cubic metres

5. Superplasticizer: in cubic metres

6. Coarse Aggregate: in cubic metres

7. Fine Aggregate: in cubic metres

An additonal predictor, which is not a material per se, is

8. Age: in days

The target variable is

Compressive strength: in megapascals

In [1]:

```python
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/framewo
rk/dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is de
precated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)typ
e'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/framewo
rk/dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is de
precated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)typ
e'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/framewo
rk/dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is de
precated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)typ
e'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/framewo
rk/dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is de
precated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)typ
e'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/framewo
rk/dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is de
precated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)typ
e'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/framewo
rk/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is de
precated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)typ
e'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/tensor
flow_stub/dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of typ
e is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/tensor
flow_stub/dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of typ
e is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/tensor
flow_stub/dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of typ
e is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/tensor
flow_stub/dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of typ
e is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/tensor
flow_stub/dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of typ
e is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/tensor
flow_stub/dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of typ
e is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
```

```
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Importing the data to a Pandas dataframe

In [3]:
```
concrete_data=pd.read_csv("https://cocl.us/concrete_data")

concrete_data
```

Out[3]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30
...	...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.28
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.18
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.70
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.77
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.40

1030 rows × 9 columns

In [7]:
```
#Checking shape of data in terms of rows and columns
concrete_data.shape
```

Out[7]:
```
(1030, 9)
```

As expected, the dataframe has 8 columns- eight predictor columns, and one target column.

It has 1030 different combinations of the materials and ages, each with its own compressive strength.

In [4]:
```
concrete_data.describe()
```

Out[4]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000

As noted under "count"- ech column has 1030 entries, a clean dataset. Ranges are as follows,

Cement from 102.0 to 540.0, with median 272.9

Blast Furnace Slag from 0 to 359.4, with median 22.0

Fly ash from 0 to 200.1, with median 0.

Water from 121.8 to 247.0 with median 185.0

Superplasticiser from 0.0 to 32.2, with median 6.4

Coarse Agreegate from 801.0 to 1145.0, with median 968.0

Fine Agreegate from 594.0 to 992.6, with median 779.5

Age from 1day to 365days, median duration 28days.

The range of Compressive Strengths are from 2.3 to 82.6, with median strength of 34.4

In [6]:
```
concrete_data.isnull().sum()
```

Out[6]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

In [8]:
```
#Defining the predictors as x, and target as y
concrete_data_columns=concrete_data.columns

x=concrete_data[concrete_data_columns[concrete_data_columns != 'Strength']]
y=concrete_data['Strength']
```

In [9]:
```
x.head()
```

Out[9]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360

In [10]:
```
y.head()
```

Out[10]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

In [12]:
```
# the eight independent variables
n_cols = x.shape[1]
```

In [13]:
```
#importing scikitlearn and keras before baseline model
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

def data_split_random (x,y,seed):
    x_train,x_test, y_train, y_test=train_test_split(x, y, test_size=0.3, random_state=0
    return x_train, x_test, y_train, y_test
```

In [21]:
```
#Defining a function to return one iteration of model
def build_baseline_model():
    baseline_model= Sequential()   #defining the model
    baseline_model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    baseline_model.add(Dense(10, activation='relu'))
    baseline_model.add(Dense(1))


    baseline_model.compile(optimizer='adam', loss='mean_squared_error')
    return baseline_model


#Repeating steps fifty times using a for loop
mse_list=[]
predicted_list={}

for i in range(50):
    if (i + 1) % 5 == 0:
        print ("Computing the {}th iteration".format(i +1))

    model=build_baseline_model()
    x_train, x_test, y_train, y_test=data_split_random(x, y, i)
    model.fit(x_train, y_train, epochs=50, verbose=0)
    y_hats=model.predict(x_test)
    mse=mean_squared_error(y_test, y_hats)
    mse_list.append(mse)
    predicted_list[i]={"y_test":y_test, "Y_hats": y_hats}



#Outputting the mean MSE, and its std dev
mse_mean=np.mean(mse_list)
mse_std=np.std(mse_list)
print("Mean MSE:{:.2f}, and Std Dev of MSE:{:.2f}".format(mse_mean, mse_std))
```

```
Computing the 5th iteration
Computing the 10th iteration
Computing the 15th iteration
Computing the 20th iteration
Computing the 25th iteration
Computing the 30th iteration
Computing the 35th iteration
Computing the 40th iteration
Computing the 45th iteration
Computing the 50th iteration
Mean MSE:174.99, and Std Dev of MSE:222.74
```

In [22]:
```
#Repeated steps but this time with normalised predictors

x_norm=(x-x.mean())/x.std()
```

```
x_norm.head()
```

Out[22]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	0.862735	-1.217079	-0.279597
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079	-0.279597
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	3.551340
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	5.055221
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569	4.976069

In [23]:

```
#Repeating steps fifty times using a for loop for normalised predictors
mse_list_norm=[]
predicted_list_norm={}

for i in range(50):
    if (i + 1) % 5 == 0:
        print ("Computing the {}th iteration".format(i +1))

    model=build_baseline_model()
    x_train, x_test, y_train, y_test=data_split_random(x_norm, y, i)
    model.fit(x_train, y_train, epochs=50, verbose=0)
    y_hats=model.predict(x_test)
    mse=mean_squared_error(y_test, y_hats)
    mse_list_norm.append(mse)
    predicted_list_norm[i]={"y_test":y_test, "Y_hats": y_hats}



#Outputting the mean MSE, and its std dev
mse_mean_norm=np.mean(mse_list_norm)
mse_std_norm=np.std(mse_list_norm)
print("Mean of Normaslised MSE:{:.2f}, and Std Dev of Normalised MSE:{:.2f}".format(mse_
```

```
Computing the 5th iteration
Computing the 10th iteration
Computing the 15th iteration
Computing the 20th iteration
Computing the 25th iteration
Computing the 30th iteration
Computing the 35th iteration
Computing the 40th iteration
Computing the 45th iteration
Computing the 50th iteration
Mean of Normaslised MSE:135.24, and Std Dev of Normalised MSE:6.36
```

In [24]:

```
#Repeating steps fifty times using a for loop for normalised predictors with hundred epo
mse_list_hundy=[]
predicted_list_hundy={}

for i in range(50):
    if (i + 1) % 5 == 0:
        print ("Computing the {}th iteration".format(i +1))

    model=build_baseline_model()
    x_train, x_test, y_train, y_test=data_split_random(x_norm, y, i)
    model.fit(x_train, y_train, epochs=100, verbose=0)
    y_hats=model.predict(x_test)
    mse=mean_squared_error(y_test, y_hats)
    mse_list_hundy.append(mse)
    predicted_list_hundy[i]={"y_test":y_test, "Y_hats": y_hats}
```

```
#Outputting the mean MSE, and its std dev
mse_mean_hundy=np.mean(mse_list_hundy)
mse_std_hundy=np.std(mse_list_hundy)
print("Mean of Normaslised MSE with Hundred Epochs :{:.2f}, and Std Dev of Normalised MS
```

```
Computing the 5th iteration
Computing the 10th iteration
Computing the 15th iteration
Computing the 20th iteration
Computing the 25th iteration
Computing the 30th iteration
Computing the 35th iteration
Computing the 40th iteration
Computing the 45th iteration
Computing the 50th iteration
Mean of Normaslised MSE with Hundred Epochs :102.78, and Std Dev of Normalised MSE with
Hundred Epochs:9.20
```

In [25]:

```
#Repeat with normalised predictors but with three hidden nodes
def build_three_layer_model():
    baseline_model= Sequential()   #defining the model
    baseline_model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    baseline_model.add(Dense(10, activation='relu'))
    baseline_model.add(Dense(10, activation='relu'))
    baseline_model.add(Dense(10, activation='relu'))
    baseline_model.add(Dense(1))


    baseline_model.compile(optimizer='adam', loss='mean_squared_error')
    return baseline_model


#Repeating steps fifty times using a for loop
mse_list_three_layer=[]
predicted_list_three_layer={}

for i in range(50):
    if (i + 1) % 5 == 0:
        print ("Computing the {}th iteration".format(i +1))

    model=build_three_layer_model()
    x_train, x_test, y_train, y_test=data_split_random(x_norm, y, i)
    model.fit(x_train, y_train, epochs=50, verbose=0)
    y_hats=model.predict(x_test)
    mse=mean_squared_error(y_test, y_hats)
    mse_list_three_layer.append(mse)
    predicted_list_three_layer[i]={"y_test":y_test, "Y_hats": y_hats}



#Outputting the mean MSE, and its std dev
mse_mean_three_layer=np.mean(mse_list_three_layer)
mse_std=np.std(mse_list_three_layer)
print("Mean MSE of Three Layer Model:{:.2f}, and Std Dev of MSE of Three Layer Model:{:.
```

```
Computing the 5th iteration
Computing the 10th iteration
Computing the 15th iteration
Computing the 20th iteration
Computing the 25th iteration
Computing the 30th iteration
Computing the 35th iteration
Computing the 40th iteration
```

```
Computing the 45th iteration
Computing the 50th iteration
Mean MSE of Three Layer Model:174.99, and Std Dev of MSE of Three Layer Model:16.67
```

Results before Normalisation of predictor variables

Mean MSE:174.99, and Std Dev of MSE:222.74

There is so much variation in the 50 computed MSEs before normalising the data emphasizing the need to normalize data prior to compiling, running, and evaluating the model

Results after Normalisation of predictor variables

Mean of Normalised MSE:135.24, and Std Dev of Normalised MSE:6.36

After the data has been normalised, the MSE has decreased, which is ideal. More importantly, the variation in the computed 50MSEs decreased dramatically, further attesting to the need to normalise predictor variables so that they carry equal weight before modeling using neural networks. This version of the regression model is better in all aspects than the previous.

Results after running 100 epochs for each iteration

Mean of Normalised MSE with Hundred Epochs :102.78, and Std Dev of Normalised MSE with Hundred Epochs:9.20

Afterrunning 100 epochs each iteration, the MSE has decreased even further, again, preferable. However, variation in the MSEs has increased. There is more variation in the MSEs. Its is more accurate, but becoming less precise.

Results after adding hidden layers to three total

Mean MSE of Three Layer Model:174.99, and Std Dev of MSE of Three Layer Model:16.67

After adding some hidden layers to a total of three, the error in the model has increased to almost the level prior to normalisation, albeit with less variation in the error. I attribute this to the regression model being overtrained on the training data, resulting in overfitting. This means it performs less well on "unseen" testing portion of the data. Training of the

regression model to this point is less than ideal, and should have stopped when the error of the model on "unseen" test data began to get worse.

By Bekezela B Khabo email: drkhabo.b@gmail.com

In [3]:
```
!pip install -U notebook-as-pdf
!pyppeteer-install
```

```
Requirement already satisfied: notebook-as-pdf in ./anaconda3/lib/python3.10/site-packag
es (0.5.0)
Requirement already satisfied: pyppeteer in ./anaconda3/lib/python3.10/site-packages (fr
om notebook-as-pdf) (1.0.2)
Requirement already satisfied: PyPDF2 in ./anaconda3/lib/python3.10/site-packages (from
notebook-as-pdf) (3.0.1)
Requirement already satisfied: nbconvert in ./anaconda3/lib/python3.10/site-packages (fr
om notebook-as-pdf) (6.5.4)
Requirement already satisfied: MarkupSafe>=2.0 in ./anaconda3/lib/python3.10/site-packag
es (from nbconvert->notebook-as-pdf) (2.1.1)
Requirement already satisfied: traitlets>=5.0 in ./anaconda3/lib/python3.10/site-package
s (from nbconvert->notebook-as-pdf) (5.7.1)
Requirement already satisfied: jinja2>=3.0 in ./anaconda3/lib/python3.10/site-packages
(from nbconvert->notebook-as-pdf) (3.1.2)
Requirement already satisfied: packaging in ./anaconda3/lib/python3.10/site-packages (fr
om nbconvert->notebook-as-pdf) (22.0)
Requirement already satisfied: entrypoints>=0.2.2 in ./anaconda3/lib/python3.10/site-pac
kages (from nbconvert->notebook-as-pdf) (0.4)
Requirement already satisfied: mistune<2,>=0.8.1 in ./anaconda3/lib/python3.10/site-pack
ages (from nbconvert->notebook-as-pdf) (0.8.4)
Requirement already satisfied: pandocfilters>=1.4.1 in ./anaconda3/lib/python3.10/site-p
ackages (from nbconvert->notebook-as-pdf) (1.5.0)
Requirement already satisfied: jupyter-core>=4.7 in ./anaconda3/lib/python3.10/site-pack
ages (from nbconvert->notebook-as-pdf) (5.2.0)
Requirement already satisfied: beautifulsoup4 in ./anaconda3/lib/python3.10/site-package
s (from nbconvert->notebook-as-pdf) (4.11.1)
Requirement already satisfied: defusedxml in ./anaconda3/lib/python3.10/site-packages (f
rom nbconvert->notebook-as-pdf) (0.7.1)
Requirement already satisfied: nbclient>=0.5.0 in ./anaconda3/lib/python3.10/site-packag
es (from nbconvert->notebook-as-pdf) (0.5.13)
Requirement already satisfied: nbformat>=5.1 in ./anaconda3/lib/python3.10/site-packages
(from nbconvert->notebook-as-pdf) (5.7.0)
Requirement already satisfied: pygments>=2.4.1 in ./anaconda3/lib/python3.10/site-packag
es (from nbconvert->notebook-as-pdf) (2.11.2)
Requirement already satisfied: bleach in ./anaconda3/lib/python3.10/site-packages (from
nbconvert->notebook-as-pdf) (4.1.0)
Requirement already satisfied: jupyterlab-pygments in ./anaconda3/lib/python3.10/site-pa
ckages (from nbconvert->notebook-as-pdf) (0.1.2)
Requirement already satisfied: tinycss2 in ./anaconda3/lib/python3.10/site-packages (fro
m nbconvert->notebook-as-pdf) (1.2.1)
Requirement already satisfied: lxml in ./anaconda3/lib/python3.10/site-packages (from nb
convert->notebook-as-pdf) (4.9.1)
Requirement already satisfied: importlib-metadata>=1.4 in ./anaconda3/lib/python3.10/sit
e-packages (from pyppeteer->notebook-as-pdf) (4.11.3)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in ./anaconda3/lib/python3.10/site-
packages (from pyppeteer->notebook-as-pdf) (1.4.4)
Requirement already satisfied: pyee<9.0.0,>=8.1.0 in ./anaconda3/lib/python3.10/site-pac
kages (from pyppeteer->notebook-as-pdf) (8.2.2)
Requirement already satisfied: certifi>=2021 in ./anaconda3/lib/python3.10/site-packages
(from pyppeteer->notebook-as-pdf) (2023.5.7)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in ./anaconda3/lib/python3.10/site-pa
ckages (from pyppeteer->notebook-as-pdf) (4.64.1)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in ./anaconda3/lib/python3.10/site
-packages (from pyppeteer->notebook-as-pdf) (1.26.14)
Requirement already satisfied: websockets<11.0,>=10.0 in ./anaconda3/lib/python3.10/site
-packages (from pyppeteer->notebook-as-pdf) (10.4)
```

```
Requirement already satisfied: zipp>=0.5 in ./anaconda3/lib/python3.10/site-packages (fr
om importlib-metadata>=1.4->pyppeteer->notebook-as-pdf) (3.11.0)
Requirement already satisfied: platformdirs>=2.5 in ./anaconda3/lib/python3.10/site-pack
ages (from jupyter-core>=4.7->nbconvert->notebook-as-pdf) (2.5.2)
Requirement already satisfied: jupyter-client>=6.1.5 in ./anaconda3/lib/python3.10/site-
packages (from nbclient>=0.5.0->nbconvert->notebook-as-pdf) (7.3.4)
Requirement already satisfied: nest-asyncio in ./anaconda3/lib/python3.10/site-packages
(from nbclient>=0.5.0->nbconvert->notebook-as-pdf) (1.5.6)
Requirement already satisfied: jsonschema>=2.6 in ./anaconda3/lib/python3.10/site-packag
es (from nbformat>=5.1->nbconvert->notebook-as-pdf) (4.17.3)
Requirement already satisfied: fastjsonschema in ./anaconda3/lib/python3.10/site-package
s (from nbformat>=5.1->nbconvert->notebook-as-pdf) (2.16.2)
Requirement already satisfied: soupsieve>1.2 in ./anaconda3/lib/python3.10/site-packages
(from beautifulsoup4->nbconvert->notebook-as-pdf) (2.3.2.post1)
Requirement already satisfied: six>=1.9.0 in ./anaconda3/lib/python3.10/site-packages (f
rom bleach->nbconvert->notebook-as-pdf) (1.16.0)
Requirement already satisfied: webencodings in ./anaconda3/lib/python3.10/site-packages
(from bleach->nbconvert->notebook-as-pdf) (0.5.1)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in ./anacon
da3/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->notebo
ok-as-pdf) (0.18.0)
Requirement already satisfied: attrs>=17.4.0 in ./anaconda3/lib/python3.10/site-packages
(from jsonschema>=2.6->nbformat>=5.1->nbconvert->notebook-as-pdf) (22.1.0)
Requirement already satisfied: tornado>=6.0 in ./anaconda3/lib/python3.10/site-packages
(from jupyter-client>=6.1.5->nbclient>=0.5.0->nbconvert->notebook-as-pdf) (6.1)
Requirement already satisfied: python-dateutil>=2.8.2 in ./anaconda3/lib/python3.10/site
-packages (from jupyter-client>=6.1.5->nbclient>=0.5.0->nbconvert->notebook-as-pdf) (2.
8.2)
Requirement already satisfied: pyzmq>=23.0 in ./anaconda3/lib/python3.10/site-packages
(from jupyter-client>=6.1.5->nbclient>=0.5.0->nbconvert->notebook-as-pdf) (23.2.0)
[INFO] Starting Chromium download.
100%|██████████████████████████████████████| 86.8M/86.8M [00:24<00:00, 3.57Mb/s]
[INFO] Beginning extraction
[INFO] Chromium extracted to: /Users/imac/Library/Application Support/pyppeteer/local-ch
romium/588429
```

In [ ]: