In [ ]:
```
version = "REPLACE_PACKAGE_VERSION"
```

Assignment 2 Part 2: Time Series Similarities (50 pts)

In this assignment, we're going to explore several techniques for measuring similarity between two time series.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

# Suppress all warnings
import warnings
warnings.filterwarnings("ignore")
```

Question 1: Load data (5 pts)

We will continue to explore the data we used in Part 1, `assets/time_series_covid19_confirmed_global.csv` from the [Johns Hopkins University CSSE COVID-19 dataset (https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series)](https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series). This time, we are interested in the number of daily new cases **exclusively from the top 5 countries that have the most cumulative cases as of August 21, 2020**.

Create a function called `load_data` that reads in the csv file and produces a `pd.DataFrame` that looks like:

	?	?	?	?	?
2020-01-23	0.0	0.0	0.0	0.0	0.0
2020-01-24	1.0	0.0	0.0	0.0	0.0
2020-01-25	0.0	0.0	0.0	0.0	0.0
2020-01-26	3.0	0.0	0.0	0.0	0.0
2020-01-27	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...
2020-08-17	35112.0	19373.0	55018.0	4839.0	2541.0
2020-08-18	44091.0	47784.0	64572.0	4718.0	2258.0
2020-08-19	47408.0	49298.0	69672.0	4790.0	3916.0
2020-08-20	44023.0	45323.0	68900.0	4767.0	3880.0
2020-08-21	48693.0	30355.0	69876.0	4838.0	3398.0

where

- the index of the DataFrame is a `pd.DatetimeIndex` ;
- the column names "?" are the top 5 countries with the most cumulative cases as of August 21, 2020, sorted in descending order from left to right;
- the values of the DataFrame are daily new cases; and
- the DataFrame doesn't contain any `NaN` values.

This function should return a `pd.DataFrame` of shape `(212, 5)` , whose index is a `pd.DatetimeIndex` and whose column labels are the top 5 countries.

In [ ]:
```
def load_data():
    daily_new_cases = None

    # YOUR CODE HERE
    raise NotImplementedError()

    return daily_new_cases
```

In [ ]:
```
# Autograder tests

stu_ans = load_data()

assert isinstance(stu_ans, pd.DataFrame), "Q1: Your function should retu
assert stu_ans.shape == (212, 5), "Q1: The shape of your pd.DataFrame ret
assert isinstance(stu_ans.index, pd.DatetimeIndex), "Q1: The index of you
assert (("2020-01-23" <= stu_ans.index) & (stu_ans.index <= "2020-08-21"
assert not stu_ans.isna().any(axis=None), "Q1: Your pd.DataFrame contain
assert stu_ans.dtypes.apply(lambda x: np.issubdtype(x, np.floating)).all


# Some hidden tests


del stu_ans
```

In [ ]:
```
# Let's plot and see the time series
axes = load_data().plot(figsize=(10, 6), title="Daily New COVID-19 Cases

del axes
```

Question 2: Extract Seasonal Components (5 pts)

Recall from the lectures that an additive Seasonal Decomposition decomposes a time series into the following components:

$$Y(t) = T(t) + S(t) + R(t)$$

where $T(t)$ represents trends, $S(t)$ represents seasonal patterns and $R(t)$ represents residuals. In the rest of the assignment, we will work with the seasonal component $S(t)$ to understand the similarities among the seasonal patterns of the five time series we have, so let's write a function that extracts this very seasonal component.

Complete the function below that accepts a `pd.DataFrame` and returns another `pd.DataFrame` of the same shape that looks like:

	?	?	?	?	?
2020-01-23	2431.761670	3380.626554	441.179428	-54.886371	322.986535
2020-01-24	3446.796153	3457.641332	621.396176	23.689984	362.434811
2020-01-25	578.564626	586.665963	594.066127	55.034811	391.346141
2020-01-26	-2728.454422	-6031.950950	46.655454	137.908703	76.880131
2020-01-27	-3293.854422	-7144.674760	-1234.673118	1.842036	-507.496059
...	...	...	...	...	...
2020-08-17	-3293.854422	-7144.674760	-1234.673118	1.842036	-507.496059
2020-08-18	-719.521088	1549.577621	-544.749308	-28.929392	-662.877011
2020-08-19	284.707483	4202.114239	76.125240	-134.659770	16.725452
2020-08-20	2431.761670	3380.626554	441.179428	-54.886371	322.986535
2020-08-21	3446.796153	3457.641332	621.396176	23.689984	362.434811

where

- the index of the DataFrame is a `pd.DatetimeIndex` ;
- the column names "?" are the top 5 countries with the most cumulative cases as of August 21, 2020, sorted in descending order from left to right;
- the values of the DataFrame are the seasonal components $S(t)$ as returned by the `seasonal_decompose` function from `statsmodels` ; and
- the DataFrame doesn't contain any `NaN` values.

This function should return a `pd.DataFrame` of shape `(len(df), 5)` , whose index is a `pd.DatetimeIndex` and whose column labels are the top 5 countries.

In [ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

def sea_decomp(df, model="additive"):
    """
    Takes in a DataFrame and extracts the seasonal components
    """
    sea_df = None

    # YOUR CODE HERE
    raise NotImplementedError()

    return sea_df
```

In [ ]:
```
# Autograder tests

stu_df = load_data()
stu_ans = sea_decomp(stu_df, "additive")

assert isinstance(stu_ans, pd.DataFrame), "Q2: Your function should retu
assert stu_ans.shape == (len(stu_df), 5), "Q2: The shape of your pd.Data
assert isinstance(stu_ans.index, pd.DatetimeIndex), "Q2: The index of yo
assert (("2020-01-23" <= stu_ans.index) & (stu_ans.index <= "2020-08-21"
assert not stu_ans.isna().any(axis=None), "Q2: Your pd.DataFrame contain
assert stu_ans.dtypes.apply(lambda x: np.issubdtype(x, np.floating)).all

# Some hidden tests


del stu_df, stu_ans
```

In [ ]:
```
# Let's plot and see the seasonal components

df = load_data()
axes = sea_decomp(df).plot(figsize=(10, 6), title="Seasonal Component of

del df, axes
```

Question 3: Calculate Euclidean Distance (10 pts)

Now, we may start to ask questions like, "which country in the top 5 countries are the most similar to Country A in terms of seasonal patterns?". In addition to the seasonal components that reflect seasonal patterns, we also need a measure of similarity between two time series in order to answer questions like this. One of such measures is the good old Euclidean Distance.

Recall that the Euclidean Distance between two vectors x and y is the length of the vector $x - y$:

$$\text{EucDist}(x, y) = \|x - y\|_2 = \sqrt{(x - y)^T\,(x - y)} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Complete the function below that accepts a `pd.DataFrame` , whose columns are time series for each country, and that returns all pairwise Euclidean Distance among these time series, similar to the following:

	?	**?**	**?**	**?**	**?**
?	0.000000	233760.757213			
?	233760.757213	0.000000			
?			0.000000		
?				0.000000	
?					0.000000

where

- the index and the column names "?" are the top 5 countries with the most cumulative cases as of August 21, 2020, sorted in descending order from top to bottom and from left to right; and
- the values of the DataFrame are pairwise Euclidean Distance, for example, `233760.757213` is the Euclidean Distance between the time series of the Rank 1 country and the Rank 2 country

This function should return a `pd.DataFrame` of shape `(5, 5)` , whose index and column labels are the top 5 countries.

In [ ]:
```
def calc_euclidean_dist(df):
    """
    Takes in a DataFrame and computes all pairwise Euclidean Distance
    """
    euclidean_dist_df = None

    # YOUR CODE HERE
    raise NotImplementedError()

    return euclidean_dist_df
```

In [ ]:
```
# Autograder tests

stu_df = load_data()
stu_ans = calc_euclidean_dist(stu_df)

assert isinstance(stu_ans, pd.DataFrame), "Q3: Your function should retu
assert stu_ans.shape == (5, 5), "Q3: The shape of your pd.DataFrame is no
assert (stu_ans.index == stu_ans.columns).all(), "Q3: Your pd.DataFrame s
assert stu_ans.dtypes.apply(lambda x: np.issubdtype(x, np.floating)).all

# Some hidden tests


del stu_df, stu_ans
```

Now let's calculate the pairwise Euclidean Distance between seasonal patterns. What can you say about the similarities among these seasonal patterns?

In [ ]:
```
# Let's show the pairwise Euclidean Distance matrix

df = load_data()
calc_euclidean_dist(sea_decomp(df))
```

Question 4: Calculate Cosine Similarity (10 pts)

Another commonly used similarity measure is the Cosine Similarity. Recall that the Cosine Similarity between two vectors x and y is the cosine of the angle between x and y:

$$\text{CosSim}(x, y) = \frac{x^T y}{\|x\|_2 \|y\|_2} = \left(\frac{x}{\|x\|_2}\right)^T \left(\frac{y}{\|y\|_2}\right)$$

Complete the function below that accepts a `pd.DataFrame` , whose columns are the time series for each country, and that returns all pairwise Cosine Similarity among these time series, similar to the following:

	?	**?**	**?**	**?**	**?**
?	1.000000	0.898664			
?	0.898664	1.000000			
?			1.000000		
?				1.000000	
?					1.000000

where

- the index and the column names "?" are the top 5 countries with the most cumulative cases as of August 21, 2020, sorted in descending order from top to bottom and from left to right; and
- the values of the DataFrame are pairwise Cosine Similarity, for example, `0.898664` is the Cosine Similarity between the time series of the Rank 1 country and the Rank 2 country

This function should return a `pd.DataFrame` of shape `(5, 5)` , whose index and column labels are the top 5 countries.

In [ ]:
```
def calc_cos_sim(df):
    """
    Takes in a DataFrame and computes all pairwise Cosine Similarity
    """
    cos_sim_df = None

    # YOUR CODE HERE
    raise NotImplementedError()

    return cos_sim_df
```

In [ ]:
```
# Autograder tests

stu_df = load_data()
stu_ans = calc_cos_sim(stu_df)

assert isinstance(stu_ans, pd.DataFrame), "Q4: Your function should retu
assert stu_ans.shape == (5, 5), "Q4: The shape of your pd.DataFrame is no
assert (stu_ans.index == stu_ans.columns).all(), "Q4: Your pd.DataFrame s
assert stu_ans.dtypes.apply(lambda x: np.issubdtype(x, np.floating)).all

# Some hidden tests


del stu_df, stu_ans
```

Now let's calculate the pairwise Cosine Similarity between seasonal patterns. What can you say about the similarities among these seasonal patterns?

In [ ]:
```
# Let's show the pairwise Cosine Similarity matrix

df = load_data()
calc_cos_sim(sea_decomp(df))
```

Question 5: Calculate Dynamic Time Warping (DTW) Cost (20 pts)

Last but not least, the cost of aligning two time series can also be used as a similarity measure. Two time series are more similar if it incurs less cost to align them. One of the commonly used alignment costs is the Dynamic Time Warping (DTW) cost, which we will explore in this problem.

Question 5a (10 pts)

Recall from the lectures that the DTW cost is defined by the following recursive relations:

$$\mathrm{DTW}(1,1) = d(x_1, y_1)$$

$$\mathrm{DTW}(i,j) = d(x_i, y_j) + \min \begin{cases} \mathrm{DTW}(i, j-1) & \text{Repeat } x_i \\ \mathrm{DTW}(i-1, j) & \text{Repeat } y_j \\ \mathrm{DTW}(i-1, j-1) & \text{Both proceed} \end{cases}$$

where we define $d(x_i, y_j) = (x_i - y_j)^2$ as in the lectures. With reference to the demo of the DTW algorithm in the lecture slides, implement the function below that computes the DTW cost for two time series. **We don't take the square root of the results just yet, until later when we compare the DTW costs with the Euclidean Distance.**

This function should EITHER return a `np.ndarray` of shape `(len(y), len(x))` which represents the DTW cost matrix, OR a single `float` that represents the overall DTW cost, depending whether the parameter `ret_matrix=True` .

In [ ]:

```
import math

def calc_pairwise_dtw_cost(x, y, ret_matrix=False):
    """
    Takes in two series. If ret_matrix=True, returns the full DTW cost ma
    otherwise, returns only the overall DTW cost
    """

    cost_matrix = np.zeros((len(y), len(x)))
    dtw_cost = None

    dist_fn = lambda a, b: (a - b) ** 2  # Optional helper function

    # YOUR CODE HERE
    raise NotImplementedError()

    return cost_matrix if ret_matrix else dtw_cost
```

In [ ]:
```
# Autograder tests

stu_df = load_data()

# First test with ret_matrix=False
stu_ans = calc_pairwise_dtw_cost(stu_df.iloc[:, 0], stu_df.iloc[:, 1], re
assert isinstance(stu_ans, float), "Q5a: Your function should return a fl
assert np.isclose(stu_ans, 9575974038.0), "Q5a: The DTW cost between Rank

# Then test with ret_matrix=True
stu_ans = calc_pairwise_dtw_cost(stu_df.iloc[:, 0], stu_df.iloc[:, 1], re
assert isinstance(stu_ans, np.ndarray), "Q5a: Your function should return
assert stu_ans.shape == (len(stu_df), len(stu_df)), "Q5a: The shape of yo
assert np.issubdtype(stu_ans.dtype, np.floating), "Q5a: Your np.ndarray s

# Also test with reversing the order of the inputs
stu_ans_T = calc_pairwise_dtw_cost(stu_df.iloc[:, 1], stu_df.iloc[:, 0],
assert np.isclose(stu_ans.T, stu_ans_T).all(), "Q5a: When the order of th

# Some hidden tests - for Rank 1 and Rank 2 countries


del stu_df, stu_ans
```

Question 5b (10 pts)

Now let's compute all pairwise DTW costs for our five time series. Complete the function below that accepts a `pd.DataFrame` , whose columns are the time series for each country, and that returns all pairwise DTW costs among these time series, similar to the following:

	?	?	?	?	?
?	0.000000e+00	9.575974e+09			
?	9.575974e+09	0.000000e+00			
?			0.000000e+00		
?				0.000000e+00	
?					0.000000e+00

where

- the index and the column names "?" are the top 5 countries with the most cumulative cases as of August 21, 2020, sorted in descending order from top to bottom and from left to right; and
- the values of the DataFrame are pairwise DTW costs, for example, `9.575974e+09` is the DTW cost between the time series of the Rank 1 country and the Rank 2 country

This function should return a `pd.DataFrame` of shape `(5, 5)` , whose index and column labels are the top 5 countries.

In [ ]:

```
def calc_dtw_cost(df):
    """
    Takes in a DataFrame and computes all pairwise DTW costs
    """

    dtw_cost_df = None

    # YOUR CODE HERE
    raise NotImplementedError()

    return dtw_cost_df
```

In [ ]:
```
# Autograder tests - takes some time

stu_df = load_data()
stu_ans = calc_dtw_cost(stu_df)

assert isinstance(stu_ans, pd.DataFrame), "Q5b: Your function should retu
assert stu_ans.shape == (5, 5), "Q5b: The shape of your pd.DataFrame is n
assert (stu_ans.index == stu_ans.columns).all(), "Q5b: Your pd.DataFrame 
assert stu_ans.dtypes.apply(lambda x: np.issubdtype(x, np.floating)).all

# Some hidden tests


del stu_df, stu_ans
```

Now let's calculate the pairwise DTW costs between seasonal patterns. **Take the sqaure root so that we can compare it with the Euclidean Distance**. What can you say about the similarities among these seasonal patterns?

In [ ]:
```
# Let's show the pairwise DTW costs matrix

df = load_data()
np.sqrt(calc_dtw_cost(sea_decomp(df)))
```