All Seasons - 6 different weather conditions(long time frame - Second-Order Markov Chain)

Import libraries and dataset

In [ ]:
```
import pandas as pd
import numpy as np
from datetime import datetime
date_format = "%Y-%m-%d"
```

In [ ]:
```
all_seasons = pd.read_csv('Datasets/all_seasons.csv')
all_seasons = all_seasons[['datetime', 'conditions']]
```

In [ ]:
```
all_seasons.head()
```

Out[ ]:

	datetime	conditions
0	2000-01-01	Partially cloudy
1	2000-01-02	Clear
2	2000-01-03	Clear
3	2000-01-04	Clear
4	2000-01-05	Clear

Classify and separate data

In [ ]:
```
classifier = {'Overcast':'overcast', 'Partially cloudy':'partially_cloudy'

all_seasons['condition'] = all_seasons['conditions'].map(classifier)
```

In [ ]:
```
all_seasons.head()
```

Out[ ]:

	datetime	conditions	condition
0	2000-01-01	Partially cloudy	partially_cloudy
1	2000-01-02	Clear	clear
2	2000-01-03	Clear	clear
3	2000-01-04	Clear	clear
4	2000-01-05	Clear	clear

In [ ]:
```
all_seasons = all_seasons[['datetime', 'condition']]
```

In [ ]:
```
all_seasons.head()
```

Out[ ]:

	datetime	condition
0	2000-01-01	partially_cloudy
1	2000-01-02	clear
2	2000-01-03	clear
3	2000-01-04	clear
4	2000-01-05	clear

In [ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
all_seasons_train = all_seasons.loc[all_seasons['datetime'].between(train_s
all_seasons_train = all_seasons_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
all_seasons_test = all_seasons.loc[all_seasons['datetime'].between(test_sta
all_seasons_test = all_seasons_test.reset_index()
```

Calculate proportions of conditions & Create transition matrix

In [ ]:

```
# Initialize count variables

# 0: 'clear' - C
# 1: 'partially_cloudy' - PC
# 2: 'overcast' - OV
# 3: 'rain' - R
# 4: 'rain_partially_cloudy' - RPC
# 5: 'rain_overcast' - ROV

conditions = ['clear', 'partially_cloudy', 'overcast', 'rain', 'rain_partia
prev_conditions = [f"{state_0}->{state_1}" for state_0 in conditions for st
prev_conditions
```

Out[ ]:

```
['clear->clear',
 'clear->partially_cloudy',
 'clear->overcast',
 'clear->rain',
 'clear->rain_partially_cloudy',
 'clear->rain_overcast',
 'partially_cloudy->clear',
 'partially_cloudy->partially_cloudy',
 'partially_cloudy->overcast',
 'partially_cloudy->rain',
 'partially_cloudy->rain_partially_cloudy',
 'partially_cloudy->rain_overcast',
 'overcast->clear',
 'overcast->partially_cloudy',
 'overcast->overcast',
 'overcast->rain',
 'overcast->rain_partially_cloudy',
 'overcast->rain_overcast',
 'rain->clear',
 'rain->partially_cloudy',
 'rain->overcast',
 'rain->rain',
 'rain->rain_partially_cloudy',
 'rain->rain_overcast',
 'rain_partially_cloudy->clear',
 'rain_partially_cloudy->partially_cloudy',
 'rain_partially_cloudy->overcast',
 'rain_partially_cloudy->rain',
 'rain_partially_cloudy->rain_partially_cloudy',
 'rain_partially_cloudy->rain_overcast',
 'rain_overcast->clear',
 'rain_overcast->partially_cloudy',
 'rain_overcast->overcast',
 'rain_overcast->rain',
 'rain_overcast->rain_partially_cloudy',
 'rain_overcast->rain_overcast']
```

In [ ]:
```
# Adding a column to identify past two states

for i in range(2, len(all_seasons_train)):
    state_0 = all_seasons_train.loc[i-2, 'condition']
    state_1 = all_seasons_train.loc[i-1, 'condition']
    all_seasons_train.loc[i, 'prev_states'] = f"{state_0}->{state_1}"

all_seasons_train
```

Out[ ]:

	index	datetime	condition	prev_states
0	731	2002-01-01	partially_cloudy	NaN
1	732	2002-01-02	rain_partially_cloudy	NaN
2	733	2002-01-03	rain_partially_cloudy	partially_cloudy->rain_partially_cloudy
3	734	2002-01-04	partially_cloudy	rain_partially_cloudy->rain_partially_cloudy
4	735	2002-01-05	partially_cloudy	rain_partially_cloudy->partially_cloudy
...	...	...	...	...
5839	6570	2017-12-27	clear	clear->clear
5840	6571	2017-12-28	clear	clear->clear
5841	6572	2017-12-29	clear	clear->clear
5842	6573	2017-12-30	partially_cloudy	clear->clear
5843	6574	2017-12-31	partially_cloudy	clear->partially_cloudy

5844 rows × 4 columns

In [ ]:
```
# Creating a count matrix
# transition_counts = prev_conditions x conditions matrix

transition_counts = np.zeros((len(prev_conditions), len(conditions)))

for i in range(len(transition_counts)):
    for j in range(len(transition_counts[0])):
        transition_counts[i][j] = len(all_seasons_train[(all_seasons_train

transition_counts
```

Out[ ]:
```
array([[1.174e+03, 4.200e+02, 2.000e+00, 3.200e+01, 7.900e+01, 6.000e+00],
       [1.400e+02, 3.510e+02, 1.200e+01, 5.000e+00, 6.700e+01, 1.500e+01],
       [1.000e+00, 2.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [3.300e+01, 8.000e+00, 0.000e+00, 8.000e+00, 4.000e+00, 0.000e+00],
       [3.000e+01, 2.700e+01, 0.000e+00, 1.100e+01, 4.000e+01, 1.000e+01],
       [2.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 4.000e+00, 1.000e+00],
       [3.440e+02, 1.170e+02, 1.000e+00, 1.200e+01, 2.100e+01, 0.000e+00],
       [2.900e+02, 1.027e+03, 5.600e+01, 1.500e+01, 9.600e+01, 2.300e+01],
       [1.000e+00, 5.300e+01, 1.100e+01, 0.000e+00, 1.000e+01, 6.000e+00],
       [1.800e+01, 5.000e+00, 0.000e+00, 3.000e+00, 5.000e+00, 0.000e+00],
       [5.800e+01, 5.600e+01, 3.000e+00, 5.000e+00, 6.800e+01, 1.000e+01],
       [6.000e+00, 9.000e+00, 0.000e+00, 2.000e+00, 2.200e+01, 4.000e+00],
       [2.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [1.500e+01, 4.100e+01, 1.000e+00, 1.000e+00, 7.000e+00, 2.000e+00],
       [0.000e+00, 9.000e+00, 4.000e+00, 0.000e+00, 2.000e+00, 2.000e+00],
       [0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [7.000e+00, 3.000e+00, 0.000e+00, 0.000e+00, 3.000e+00, 2.000e+00],
       [0.000e+00, 1.000e+00, 1.000e+00, 0.000e+00, 3.000e+00, 3.000e+00],
       [5.900e+01, 1.500e+01, 0.000e+00, 3.000e+00, 6.000e+00, 1.000e+00],
       [8.000e+00, 9.000e+00, 1.000e+00, 0.000e+00, 3.000e+00, 0.000e+00],
       [0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [1.000e+01, 3.000e+00, 0.000e+00, 4.000e+00, 4.000e+00, 0.000e+00],
       [3.000e+00, 6.000e+00, 0.000e+00, 0.000e+00, 8.000e+00, 1.000e+00],
       [0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00],
       [1.280e+02, 3.600e+01, 0.000e+00, 5.000e+00, 1.200e+01, 0.000e+00],
       [3.900e+01, 7.400e+01, 1.000e+01, 8.000e+00, 2.400e+01, 2.000e+00],
       [0.000e+00, 2.000e+00, 1.000e+00, 0.000e+00, 2.000e+00, 0.000e+00],
       [1.900e+01, 5.000e+00, 0.000e+00, 6.000e+00, 5.000e+00, 1.000e+00],
       [6.500e+01, 4.800e+01, 2.000e+00, 1.500e+01, 7.100e+01, 7.000e+00],
       [1.000e+00, 1.000e+00, 2.000e+00, 2.000e+00, 1.200e+01, 1.500e+01],
       [6.000e+00, 2.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00],
       [3.000e+00, 6.000e+00, 1.000e+00, 2.000e+00, 2.000e+00, 1.000e+00],
       [0.000e+00, 1.000e+00, 1.000e+00, 0.000e+00, 1.000e+00, 0.000e+00],
       [4.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [1.800e+01, 1.700e+01, 0.000e+00, 5.000e+00, 1.800e+01, 3.000e+00],
       [0.000e+00, 4.000e+00, 0.000e+00, 0.000e+00, 1.900e+01, 1.200e+01]])
```

In [ ]:
```
# Turning count matrix into proportions by normalizing across rows

def normalize(arr):
    total = sum(arr)
    if total == 0:
        return arr
    return arr / total

transition_prob = np.apply_along_axis(normalize, 1, transition_counts)
transition_prob
```

Out[ ]:
```
array([[0.68534734, 0.24518389, 0.00116754, 0.01868068, 0.04611792,
        0.00350263],
       [0.23728814, 0.59491525, 0.02033898, 0.00847458, 0.11355932,
        0.02542373],
       [0.33333333, 0.66666667, 0.        , 0.        , 0.        ,
        0.        ],
       [0.62264151, 0.1509434 , 0.        , 0.1509434 , 0.0754717 ,
        0.        ],
```

```
[0.25423729, 0.22881356, 0.        , 0.09322034, 0.33898305,
 0.08474576],
[0.28571429, 0.        , 0.        , 0.        , 0.57142857,
 0.14285714],
[0.69494949, 0.23636364, 0.0020202 , 0.02424242, 0.04242424,
 0.        ],
[0.1924353 , 0.6814864 , 0.03715992, 0.00995355, 0.06370272,
 0.01526211],
[0.01234568, 0.65432099, 0.13580247, 0.        , 0.12345679,
 0.07407407],
[0.58064516, 0.16129032, 0.        , 0.09677419, 0.16129032,
 0.        ],
[0.29      , 0.28      , 0.015     , 0.025     , 0.34      ,
 0.05      ],
[0.13953488, 0.20930233, 0.        , 0.04651163, 0.51162791,
 0.09302326],
[1.        , 0.        , 0.        , 0.        , 0.        ,
 0.        ],
[0.2238806 , 0.6119403 , 0.01492537, 0.01492537, 0.10447761,
 0.02985075],
[0.        , 0.52941176, 0.23529412, 0.        , 0.11764706,
 0.11764706],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        ],
[0.46666667, 0.2       , 0.        , 0.        , 0.2       ,
 0.13333333],
[0.        , 0.125     , 0.125     , 0.        , 0.375     ,
 0.375     ],
[0.70238095, 0.17857143, 0.        , 0.03571429, 0.07142857,
 0.01190476],
[0.38095238, 0.42857143, 0.04761905, 0.        , 0.14285714,
 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        ],
[0.47619048, 0.14285714, 0.        , 0.19047619, 0.19047619,
 0.        ],
[0.16666667, 0.33333333, 0.        , 0.        , 0.44444444,
 0.05555556],
[0.        , 0.        , 0.        , 0.        , 1.        ,
 0.        ],
[0.70718232, 0.19889503, 0.        , 0.02762431, 0.06629834,
 0.        ],
[0.24840764, 0.47133758, 0.06369427, 0.05095541, 0.15286624,
 0.01273885],
[0.        , 0.4       , 0.2       , 0.        , 0.4       ,
 0.        ],
[0.52777778, 0.13888889, 0.        , 0.16666667, 0.13888889,
 0.02777778],
[0.3125    , 0.23076923, 0.00961538, 0.07211538, 0.34134615,
 0.03365385],
[0.03030303, 0.03030303, 0.06060606, 0.06060606, 0.36363636,
 0.45454545],
[0.66666667, 0.22222222, 0.        , 0.11111111, 0.        ,
 0.        ],
[0.2       , 0.4       , 0.06666667, 0.13333333, 0.13333333,
 0.06666667],
[0.        , 0.33333333, 0.33333333, 0.        , 0.33333333,
```

```
        0.        ],
       [1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        ],
       [0.29508197, 0.27868852, 0.        , 0.08196721, 0.29508197,
        0.04918033],
       [0.        , 0.11428571, 0.        , 0.        , 0.54285714,
        0.34285714]])
```

In [ ]:
```
# Verifying rows sum to 1
np.apply_along_axis(sum, 1, transition_prob)
```

Out[ ]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1.])
```

In [ ]:
```
all_seasons_test.head(1)
```

Out[ ]:

	index	datetime	condition
0	6575	2018-01-01	clear

First day of 2018: clear

Below cells are commented out to avoid errors when running

def predict_weather_six_conditions(test_data): state = {0:'clear', 1:'partially_cloudy', 2:'overcast', 3:'rain', 4:'rain_partially_cloudy', 5:'rain_overcast'} n = len(test_data) # how many steps to test start_state = 0 # 0 = clear test_result = test_data.copy() prev_state = start_state result = [] while n-1: curr_state = np.random.choice([0,1,2,3,4,5], p=t_array[prev_state]) #taking the probability from the transition matrix result.append(state[curr_state]) prev_state = curr_state n -= 1 curr_state = np.random.choice([0,1,2,3,4,5], p=t_array[prev_state]) #taking the probability from the transition matrix result.append(state[curr_state]) test_result['predicted_condition'] = result return test_result def find_accuracy(predicted_result): correct_count = 0.0 for i in range(len(predicted_result)): if predicted_result.loc[i, 'condition'] == predicted_result.loc[i, 'predicted_condition']: correct_count += 1 correct_prop = correct_count / len(predicted_result) return correct_prop def run_predictions_return_avg_accuracy(test_data, trial_count): accuracy_sum = 0.0 for i in range(trial_count): predicted_result = predict_weather_six_conditions(test_data) accuracy = find_accuracy(predicted_result) accuracy_sum += accuracy avg_accuracy = accuracy_sum / trial_count return avg_accuracy# Sample prediction (for table graphic) sample_prediction = predict_weather_six_conditions(all_seasons_test) sample_accuracy = find_accuracy(sample_prediction) print(sample_prediction.head()) print(sample_accuracy)run_predictions_return_avg_accuracy(all_seasons_test, 100)

In [ ]: