Weather forecasting with Markov chains

Math 42 Final Project
Department of Mathematics, UCLA

Junwon Choi
(UID: 605 317 213 / Email: cjunwon@g.ucla.edu)
Brian Chau
(UID: 109 548 540 / Email: bchau3648@g.ucla.edu)
Yifan Jiang
(UID: 706 037 292 / Email: jiangyifan0421@g.ucla.edu)
Mingyeong Kim
(UID: 905 974 280 / Email lunamkbruin@g.ucla.edu)
Nalin Chopra
(UID: 405 746 993 / Email: nalinchopra@g.ucla.edu)

December 7, 2022

Contents

Abstract

Weather forecasting typically relies on analyzing physical patterns and environmental conditions. This paper attempts to predict future weather conditions without relying on current physical/environmental conditions. This process was modeled purely based on the Markov chain and prediction results were coded and generated with Python. The Markov chain methodology was applied across a total of 20 years worth of training and testing weather data mainly in the campus of the University of California, Los Angeles. Predictions were tested across multiple subsections of our dataset. A simplified model categorized each day as 'Rain' and 'No Rain', and predictions using a simplified model produced an average accuracy of 77% across different datasets. A more complex model that categorized each day as one of six different weather conditions had predictions producing an average accuracy no greater than 35%.

All code and datasets used in this report can be found here: `https://github.com/cjunwon/Math_42_Final_Project`.

1 Problem description

Weather plays a crucial role in our lives, influencing clothing choices, travel plans, choice of transportation, etc. Naturally, weather forecasting is an essential tool that helps individuals make decisions on a daily basis. Hence most people are usually interested in short-term forecasts, which are predictions of the weather for up to a week in the future. [1] According to the American Geosciences Institute, "In earlier times, before the telegraph and the telephone were invented, weather observations from faraway places could not be collected in one place soon after they were made. In those times, the only way of predicting the weather was to use your local experience." [1] Analyzing weather patterns became feasible in the 1800's, with more advanced technology being introduced in the 1900's, allowing for more accurate weather forecasting methods.

Based on such information, we were interested in predicting the weather from the perspective of people before advanced technologies were developed.

This research paper attempts to forecast weather conditions purely through mathematical modeling that does not depend on the physical conditions of our current environment or any other tools developed for weather forecasting. The paper addresses the following questions:

- Which mathematical model is most appropriate for weather prediction?
- How can we fit the selected model to predict weather in a particular location?
- How accurate/significant is the model? How does it perform when fitted to different locations?

Thus we would need to develop and apply a mathematical model that only required past weather data to predict future weather conditions. The model could be tested by applying it to predict in the same location as the origin of the dataset and further validated by applying the same model to different locations that are distant from the initial dataset.

This method would simulate a similar thought process that humans have. For example, a Los Angeles native that is used to weather patterns of Southern California might lack necessary information and experience about weather patterns of New York, and thus their assumptions or predictions about New York's weather would be based on Southern California's patterns, not New York's.

2 Simplifications

2.1 Data collection and train/test Splitting

Weather data was collected from "https://www.visualcrossing.com" with location restricted to the University of California, Los Angeles campus (zip code - 90024) [2]. Weather data from the year 2000 and 2022 were downloaded.

We used two different time frame schemes to find the most efficient and accurate model of our data. The Markov chain discrete models were broken into two: one with only five years' worth of data and one with the entire data, that can assist in predicting if it is more accurate to predict using short term or long term datasets. We broke annual data into quarters (as defined by UCLA's quarter system), so we collected data for Winter, Spring, Summer, and Fall quarters, allowing us to cater more for each specific quarter. For example, winter would contain the months January, February, and March, which parallels UCLA's winter quarter. We can compare it with models built on seasonal data only vs annual data to see which is more accurate.

For a *long time frame* analysis, data collected daily from 2002 to 2017 were used as a training dataset to create the transition matrix that represents a Markov chain. Data collected from 2018 to 2021 were used as a testing dataset to fit the transition matrix obtained with the training data. Data from 2022 were omitted as it did not contain a full year's worth of data. Data was therefore split into 80% training data and 20% testing data.

For a *short time frame* analysis, data collected daily from 2017 to 2020 were used as a training dataset to create the transition matrix that represents a Markov chain. Data collected from 2021 was used as a testing dataset to fit the transition matrix obtained with the training data. Entries from 2022 were omitted as it did not contain a full year's worth of data. Data was therefore split into 80% training data and 20% testing data.

2.2 Analysis steps

The analysis of our problem was split into 6 main steps:

1. All seasons + Simplified (long time frame)

   This takes into account every day of each year, and categorizes each day as 'Rain' or 'No Rain'. A total of 20 years worth of training and testing data were used.

2. All seasons + Simplified (short time frame)

   This takes into account every day of each year, and categorizes each day as 'Rain' or 'No Rain'. A total of 5 years worth of training and testing data were used.

3. All seasons + 6 different weather conditions (long time frame)

   This takes into account every day of each year, and categorizes each day as 'Clear', 'Partially cloudy', 'Overcast', 'Rain', 'Rain, Partially cloudy', 'Rain, Overcast'. A total of 20 years worth of training and testing data were used.

4. All seasons + 6 different weather conditions (short time frame)

   This takes into account every day of each year, and categorizes each day as 'Clear', 'Partially cloudy', 'Overcast', 'Rain', 'Rain, Partially cloudy', 'Rain, Overcast'. A total of 5 years worth of training and testing data were used.

5. By season + 6 different weather conditions (long time frame)

   This separates each year's data into four different seasons, and categorizes each day as 'Clear', 'Partially cloudy', 'Overcast', 'Rain', 'Rain, Partially cloudy', 'Rain, Overcast'. A total of 20 years worth of training and testing data were used.

6. All seasons + Simplified applied to University of California, San Diego (UCSD)/New York University (NYU) (long time frame)

   This takes into account every day of each year, and categorizes each day as 'Rain' or 'No Rain'. A model created using the University of California, Los Angeles' (UCLA) weather data is applied to a nearby location (UCSD) and a more distant location (NYU) for validation purposes. A total of 20 years worth of training and testing data were used.

2.3 Assumptions and limitations

Our model will assume that the weather condition for a certain day depends on the weather conditions of the day before. The model is simplified by categorizing each day as one of the weather conditions mentioned above. In reality, weather is significantly more complex and has various factors affecting it, such as temperature, humidity, wind speed, etc. Therefore, solely using the probability of categorical conditions without taking into account other natural variables that affect weather conditions can be a limiting factor that hinders the accuracy of the model.

3 Mathematical Model

3.1 Explanation of Markov chains

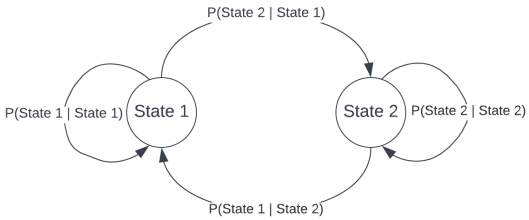


Figure 1: Visualization of General Markov Chain with Two States

A Markov chain is a mathematical system that changes from one state to another based on statistical criteria. The defining feature of a Markov chain is that the possible future states are fixed regardless of how the process got to its current state. In other words, the chance of transitioning to any particular condition is purely determined by the current state and the amount of time elapsed. As we can see from Figure 1, the probabilities from transitioning from one state to another are conditional probabilities that only depend on the information about the previous state. While Markov chains with any size of state space can be discussed, the initial theory and most implementations are focused on scenarios with a finite (or countably infinite) number of states [5].

There are three properties which identify a state model as being a Markov model. We considered these properties to build up our own Markov chain model and equations [3].

- The Markov assumption: the probability of one's moving from state i to state j is independent of what happened before moving to state j and of how one got to state i.
- Conservation: the sum of the probabilities out of a state must be one.
- The vector $X(t)$ is a probability distribution vector which describes the probability of the system's being in each of the states at time n.

One of the Markov chain properties that we focused on is that knowledge of the previous state is all that is necessary to determine the probability distribution of the current state [2]. We concentrate on finding the transition matrix of the Markov chain, whose entries are $P(s_i = X|s_{i-1} = Y)$, where s_i is the state of the chain at the current time step, s_{i-1} is the state at the previous time step, and X, Y are contained in the respective state space of the model. The transition matrix for a Markov chain is then a matrix of probabilities of moving from one state to another. Thus, if T is the transition matrix with n states,

$$T = \begin{pmatrix} p_{11} & p_{12} & \dots & p_{1n} \\ p_{21} & p_{22} & \dots & p_{2n} \\ \dots & \dots & \ddots & \dots \\ p_{n1} & p_{n2} & \dots & p_{nn} \end{pmatrix}$$

such that the rows would represent a state at time t and the columns would represent the state at time $t + 1$.

3.2 Application of Markov chains to weather forecasting

As we have different combinations of models, we have two different types of state spaces. We define the universal state space to be S = {Clear; Partially cloudy; Overcast; Rain; Rain, Partially cloudy; Rain, Overcast}, which is used to create the more complicated model. We also define the state space A = {Rain, No Rain}. This is the simplest version we use to predict the daily weather conditions. From the state space S, {Clear; Partially cloudy; Overcast} are conditions that fall under {No Rain} in state space A whereas the other three conditions are considered under {Rain} conditions. We utilize these state spaces to generate probabilities as well as the transition matrix and predict the sequence of states of future weather conditions.

Focusing on the simplest model with the state space A = {Rain, No Rain}, the transition matrix T would be of the form

$$\mathrm{T}_{\text{simplified}} = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} P(R|R) & P(NR|R) \\ P(R|NR) & P(NR|NR) \end{pmatrix} \end{array}$$

where P(A_i, A_j) is the conditional probability of the current condition being A_i given that the previous day had a condition of A_j and $A_i, A_j \in A$. In the transition matrix, for the rows i and columns j, $A_{i,j}$ represents the probability of the weather condition on day $t+1$ to be the weather condition assigned to column j, where row i is the weather assigned on day t.

Similarly, for the model with six conditions, the transition matrix would be

$$\mathrm{T}_{\text{six cond.}} = \begin{array}{c} \\ \text{C} \\ \text{PC} \\ \text{OV} \\ \text{R} \\ \text{RPC} \\ \text{ROV} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{C} & \text{PC} & \text{OV} & \text{R} & \text{RPC} & \text{ROV} \end{array} \\ \begin{pmatrix} P(C|C) & P(PC|C) & P(OV|C) & P(R|C) & P(RPC|C) & P(ROV|C) \\ P(C|PC) & P(PC|PC) & P(OV|PC) & P(R|PC) & P(RPC|PC) & P(ROV|PC) \\ P(C|OV) & P(PC|OV) & P(OV|OV) & P(R|OV) & P(RPC|OV) & P(ROV|OV) \\ P(C|R) & P(PC|R) & P(OV|R) & P(R|R) & P(RPC|R) & P(ROV|R) \\ P(C|RPC) & P(PC|RPC) & P(OV|RPC) & P(R|RPC) & P(RPC|RPC) & P(ROV|RPC) \\ P(C|ROV) & P(PC|ROV) & P(OV|ROV) & P(R|ROV) & P(RPC|ROV) & P(ROV|ROV) \end{pmatrix} \end{array}$$

Where: Clear - (C), Partially Cloudy - (PC), Overcast - (OV), Rain - (R), Rain, Partially Cloudy - (RPC), Rain, Overcast - (ROV)

4 Solution of Mathematical Problem

To create our Markov chain model, the parameters inside the transition matrix were first calculated. The parameters inside the transition matrix need to be the probabilities of a particular weather condition shall succeed a particular weather condition the following day. For this process, we utilize the standard method of calculating conditional probabilities for possible events. We counted the number of times a state on one day would succeed a state on the next day, the states representing all the combinations of weather conditions. Then these counts were organized in a matrix. After normalizing the rows so that the sum of each row would be equal to one, we get our transition matrix.

Following this process, we get the following matrices for our models:

1. All seasons + Simplified (long time frame):

$$\mathrm{T} = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.467 & 0.533 \\ 0.096 & 0.904 \end{pmatrix} \end{array}$$

2. All seasons + Simplified (short time frame):

$$\mathrm{T} = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.495 & 0.505 \\ 0.085 & 0.915 \end{pmatrix} \end{array}$$

3. All seasons + 6 different weather conditions (long time frame):

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.690 & 0.199 & 0.001 & 0.034 & 0.073 & 0.004 \\ 0.250 & 0.640 & 0.028 & 0.009 & 0.067 & 0.006 \\ 0.028 & 0.743 & 0.156 & 0.0 & 0.046 & 0.028 \\ 0.366 & 0.214 & 0.0 & 0.145 & 0.248 & 0.028 \\ 0.190 & 0.323 & 0.0242 & 0.029 & 0.335 & 0.098 \\ 0.055 & 0.339 & 0.0630 & 0.008 & 0.260 & 0.276 \end{pmatrix} \end{array}$$

4. All seasons + 6 different weather conditions (short time frame):

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.734 & 0.173 & 0.0 & 0.013 & 0.079 & 0.0 \\ 0.249 & 0.636 & 0.037 & 0.004 & 0.064 & 0.011 \\ 0.0 & 0.765 & 0.206 & 0.0 & 0.029 & 0.0 \\ 0.5 & 0.056 & 0.0 & 0.222 & 0.222 & 0.0 \\ 0.183 & 0.254 & 0.0296 & 0.018 & 0.420 & 0.095 \\ 0.087 & 0.565 & 0.087 & 0. & 0.217 & 0.043 \end{pmatrix} \end{array}$$

5. By season + 6 different weather conditions (long time frame):

   - Fall:

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.713 & 0.139 & 0.0 & 0.048 & 0.090 & 0.010 \\ 0.346 & 0.488 & 0.016 & 0.026 & 0.111 & 0.013 \\ 0.0 & 0.588 & 0.176 & 0.0 & 0.176 & 0.059 \\ 0.288 & 0.205 & 0.0 & 0.233 & 0.247 & 0.027 \\ 0.213 & 0.269 & 0.028 & 0.046 & 0.347 & 0.097 \\ 0.107 & 0.268 & 0.036 & 0.018 & 0.214 & 0.357 \end{pmatrix} \end{array}$$

   - Winter:

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.677 & 0.158 & 0.0 & 0.043 & 0.116 & 0.002 \\ 0.343 & 0.491 & 0.019 & 0.009 & 0.123 & 0.012 \\ 0.076 & 0.846 & 0.076 & 0.0 & 0.0 & 0.0 \\ 0.375 & 0.15 & 0.0 & 0.05 & 0.375 & 0.05 \\ 0.210 & 0.261 & 0.012 & 0.019 & 0.377 & 0.121 \\ 0.038 & 0.396 & 0.019 & 0.0 & 0.302 & 0.245 \end{pmatrix} \end{array}$$

   - Spring:

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.649 & 0.272 & 0.004 & 0.016 & 0.057 & 0.0 \\ 0.196 & 0.693 & 0.044 & 0.002 & 0.056 & 0.006 \\ 0.017 & 0.736 & 0.192 & 0.0 & 0.035 & 0.017 \\ 0.454 & 0.363 & 0.0 & 0.0 & 0.181 & 0.0 \\ 0.115 & 0.519 & 0.038 & 0.009 & 0.240 & 0.076 \\ 0.0 & 0.375 & 0.312 & 0.0 & 0.187 & 0.125 \end{pmatrix} \end{array}$$

   - Summer:

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.693 & 0.262 & 0.001 & 0.021 & 0.021 & 0.0 \\ 0.211 & 0.734 & 0.021 & 0.006 & 0.025 & 0.0 \\ 0.136 & 0.727 & 0.090 & 0.0 & 0.0 & 0.045 \\ 0.619 & 0.285 & 0.0 & 0.047 & 0.047 & 0.0 \\ 0.162 & 0.534 & 0.046 & 0.046 & 0.186 & 0.023 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.5 & 0.0 \end{pmatrix} \end{array}$$

6. All seasons + Simplified applied to UCSD/NYU (long time frame):

   - UCSD:

$$T = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.467 & 0.532 \\ 0.095 & 0.904 \end{pmatrix} \end{array}$$

   - NYU:

$$T = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.467 & 0.532 \\ 0.095 & 0.904 \end{pmatrix} \end{array}$$

A visual representation of these transition matrices can be found in the appendix 8.

Alongside this matrix, an initial condition is needed. We take the first day of the testing data and make a column vector whose entries are all 0 except for the entry that corresponds to the weather outlook condition of the first day, which would be equal to 1. The reasoning behind this is that given the information about the first day (the initial condition), we know that the probability of it happening will be 100%. This will allow our model to be specific to the test data, as opposed to making a column vector of the proportion of days that are of each condition.

With the transition matrix and the initial condition, we are now able to make predictions. Since we know the condition of the first day, we take the row of the transition matrix corresponding to that condition. We then use the probabilities in that row as weights to randomly select the condition of the next day. This process can be repeated for any number of days.

For example, with an all season simplified model (long time frame):
Let Day 1 of the model be a 'No Rain' day, which would give a vector $V = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. We take transition matrix $T = \begin{pmatrix} 0.467 & 0.533 \\ 0.096 & 0.904 \end{pmatrix}$.

We multiply V and the transpose of T using a Python function. This outputs a vector with two elements containing probabilities of 'Rain' or 'No Rain' given the past day is 'No Rain'. Since the Markov chain model's key feature is randomness, we use a Python function to use the two probabilities of the transposed second row (which add to 1.0) as inputs to make a random choice. This choice will determine whether Day 2 will be 'Rain' or 'No Rain'. It is safe to assume that the random function has a higher probability of choosing 'No Rain' given the previous day was 'No Rain', since that has a probability of 90.4%. We repeat this step by a desired amount.

The prediction outputs would be appended to the testing dataset containing actual values to later calculate the accuracy of the predictions.

As an example, the first five entries of the 'All Seasons 6 Weather Conditions (long time frame)' and its accuracy at one iteration is shown below:

```
   index    datetime         condition predicted_condition
0   6575  2018-01-01             clear               clear
1   6576  2018-01-02             clear               clear
2   6577  2018-01-03             clear               clear
3   6578  2018-01-04  partially_cloudy    partially_cloudy
4   6579  2018-01-05  partially_cloudy                rain
0.3750855578370979
```

Figure 2: First five entries of a predictive iteration for 'All Seasons 6 Weather Conditions' and its accuracy

This is different from the algorithm used in class, which repeatedly multiplied the transition matrix by the column vector to get a distribution of the population. Because we are trying to find an exact solution for each day, and not a distribution or probability vector, we decided on the method detailed above. In addition, due to the long testing period, the algorithm used in class would tend towards a stationary distribution. Our method ensures that this does not happen and allows our model to be more dynamic, which would fit the problem better since weather is constantly changing.

5 Results

In order to compare the models against each other, we needed to quantify the accuracy of our model. We considered statistical tests, such as the chi-squared test, as well as other options, like calculating the distribution of days with different weather outlooks predicted by the model and seeing how much the distribution differs from reality. However, this only took into account the proportion of the types of conditions and ignored their order. We believe that the sequential arrangement of days is an important aspect of the Markov chain. As a result, we decided to directly calculate the proportion of days where the results from our model matched the test data, although the other approaches could possibly have lead to higher accuracy. It is important to note that because our process involved some randomness, we generated 100 predictions and averaged their accuracy.

We list out the accuracy for all 10 of our models

1. All seasons + Simplified (long time frame): 77%
2. All seasons + Simplified (short time frame): 79%
3. All seasons + 6 different weather conditions (long time frame): 37%
4. All seasons + 6 different weather conditions (short time frame): 39%
5. By season + 6 different weather conditions (long time frame)
    - Fall: 37%
    - Winter: 35%
    - Spring: 40%
    - Summer: 45%
6. All seasons + Simplified applied to UCSD/NYU (long time frame)
    - UCSD: 73%
    - NYU: 43%

Concentrating on models numbered one to four, it is clear that the simplified model performed much better than the model with six conditions and would be more useful for applications to real world scenarios. The low accuracy of the more complicated model is expected, considering the simplicity of the Markov chain. But, it is unexpected that the models trained on a shorter time frame

had about a 2% higher accuracy than the ones trained on a longer period. This could be due to the fact that the models trained on a shorter time frame only considered years that were closer to the testing data. On the other hand, the models trained with a longer time frame included data that was irrelevant to the testing data. As a result, the shorter time frame could feasibly be more accurate. This highlights one of our model's advantages – our model does not need an extensive amount of data to be accurate, at least for the simplified one.

Now we focus on model number five, where we make a distinction between seasons. From the results, we can see that most of the uncertainty from the model lies in the fall and winter seasons. This is reasonable because according to the New York Times, California "typically gets 75 percent of its annual rainfall between November and March" [4]. As a result, it is to be expected that fall and winter would have more of a mix between rainy and clear days while spring and summer would have more consistent weather.

For model number six, we apply our simplified model that spans all seasons (model number 1) to UCSD and NYU. Although we get slightly worse performance at UCSD, there is a drastic decrease in accuracy with NYU. This is expected because California has a relatively temperate climate and the weather would not vary widely when comparing UCLA's weather to that of UCSD. However, NYU is on the other side of the coast and also includes snowy days. Consequently, we get significantly worse accuracy, which explains our results.

6 Improvement

6.1 Implementation of time-inhomogeneous Markov chains

The transition matrix of our model stays constant throughout the whole process (i.e. at each time step, the same initial transition matrix is applied to the current step). A possible improvement is to use time-inhomogeneous Markov Chains. Its definition is broader, as it allows for non-stationary transition probabilities, that is, as time goes on, the probability of moving from one state to another may change. With time-inhomogeneous Markov chains, the transition matrix is generated again at each time step. Since this method adjusts the probabilities at each timestep, we assume that it provides a more accurate result.

6.2 Extending time frame of analysis with second-order Markov chains

To improve the accuracy of our model, we attempted to implement a second-order Markov chain for both the simplified and six conditions models, which would use the past two states to predict the next state instead of one past state. However, during this implementation, we realized that there are still limitations that prevent us from doing so. For example, as we were trying to generate the transition matrix, we would get a row of all zeros because there aren't any observations where the two previous states were 'clear' $\rightarrow$ 'overcast'. Therefore, extending the time frame of our analysis when attempting to use the second-order Markov chains may resolve the problem that we faced, as it will have more instances that have all pairs of weather conditions.

7 Conclusions

7.1 Project achievements

Through this project, we were able to make a prediction model for weather that is independent of climate measurements and only relies on the condition of the previous day. By utilizing Markov chains, we were able to predict years worth of weather conditions with relatively high accuracy for a simplified weather model.
Additionally, we successfully expanded our model's scope by applying the transition matrices derived from UCLA-based data to predicting weather at UCSD (about 120 miles away from Los Angeles) with a relatively close accuracy to UCLA.

7.2 Project takeaways

This project highlighted a real world application of the Markov chain model. We learned that Markov chains are able to reflect our common thinking process in a more accurate and systematic manner.

References

[1] How has weather forecasting changed over the past two hundred years? *The American Geosciences Institute*, 2022.

[2] Visual Crossing Corporation, 2022.

[3] Randall Swift Douglas D. Mooney. *A Course in Mathematical Modeling*, page 122. The Mathematical Association of America, 1999.

[4] Soumya Karlamangla. California is expected to enter a fourth straight year of drought. *The New York Times*, October.

[5] Henry Maltby, et al. Markov chains. *Brilliant.org*, December 2022.

8 Appendix

8.1 Visual Representation of Transition Matrices

1. All seasons + Simplified (long time frame):

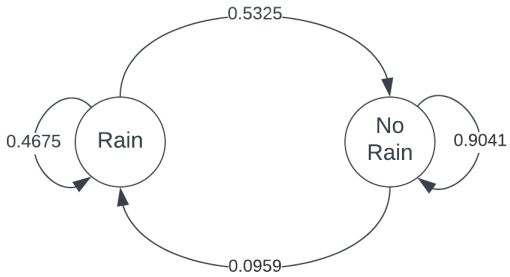


Figure 3: Visualization of Simplified Markov Model Trained Over Long Time Frame with All Seasons

2. All seasons + Simplified (short time frame):

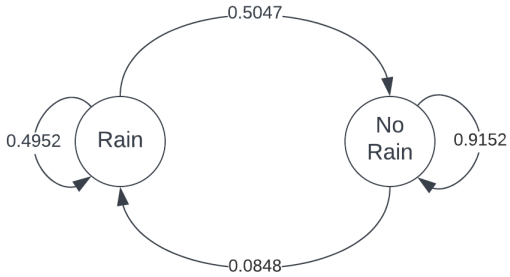


Figure 4: Visualization of Simplified Markov Model Trained Over Short Time Frame with All Seasons

3. All seasons + 6 different weather conditions (long time frame):

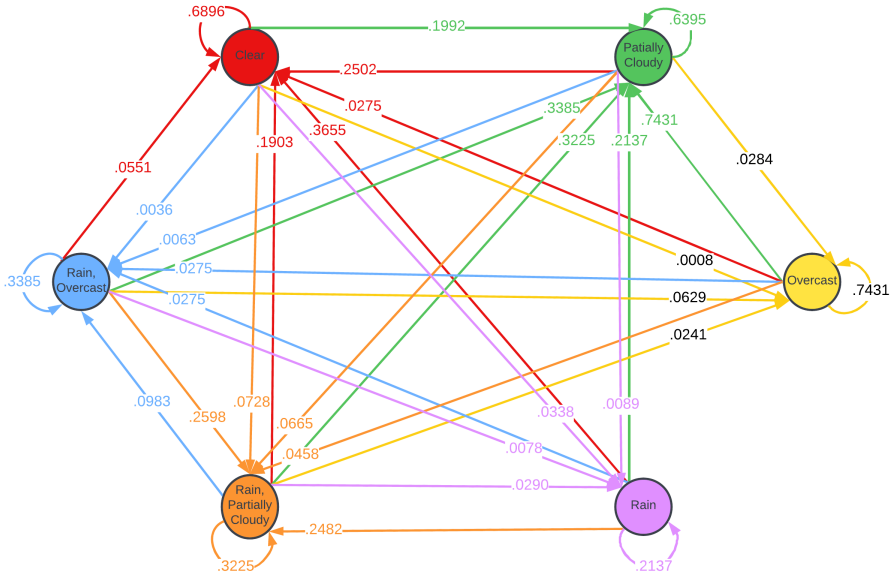


Figure 5: Visualization of Six States Markov Model Trained Over Long Time Frame with All Seasons

4. All seasons + 6 different weather conditions (short time frame):

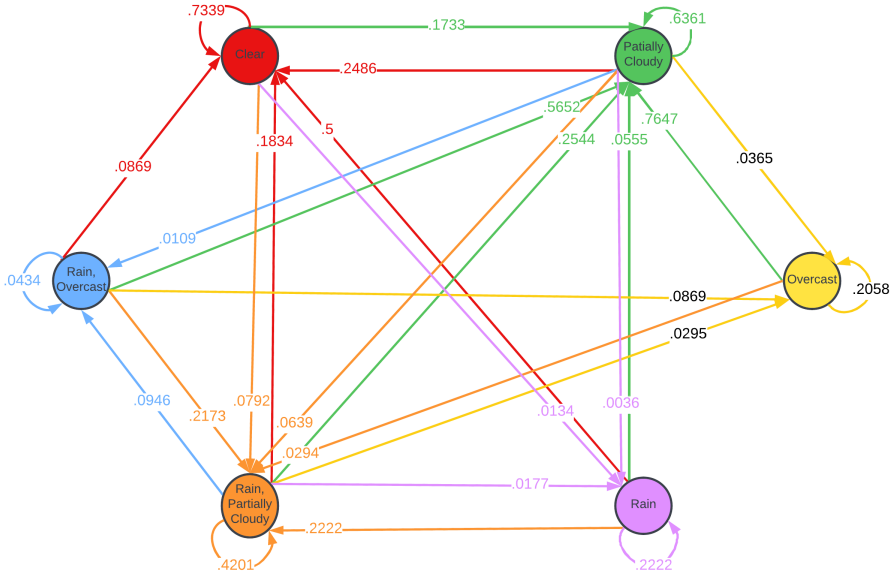


Figure 6: Visualization of Six States Markov Model Trained Over Short Time Frame with All Seasons

5. By season + 6 different weather conditions (long time frame)
   - Fall:

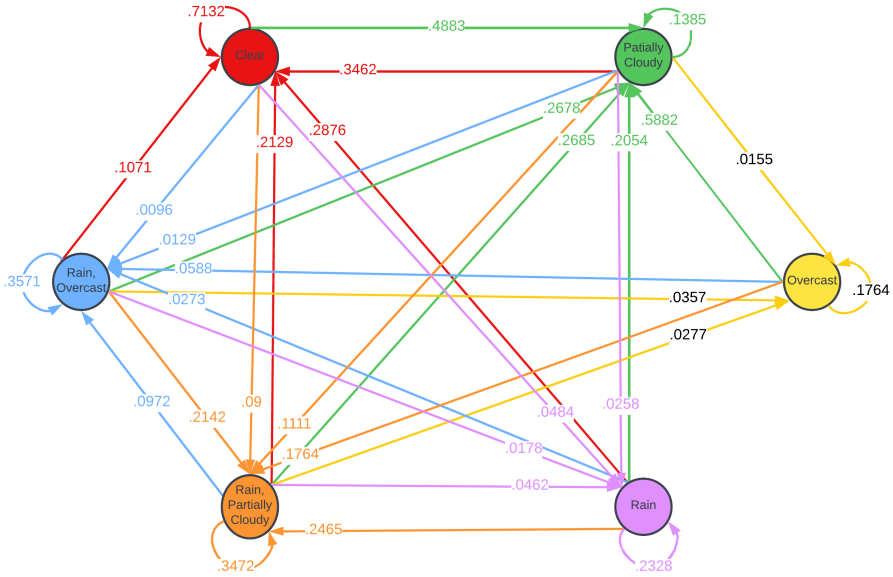


Figure 7: Visualization of Six States Markov Model Trained Over Long Time Frame for Fall

- Winter:

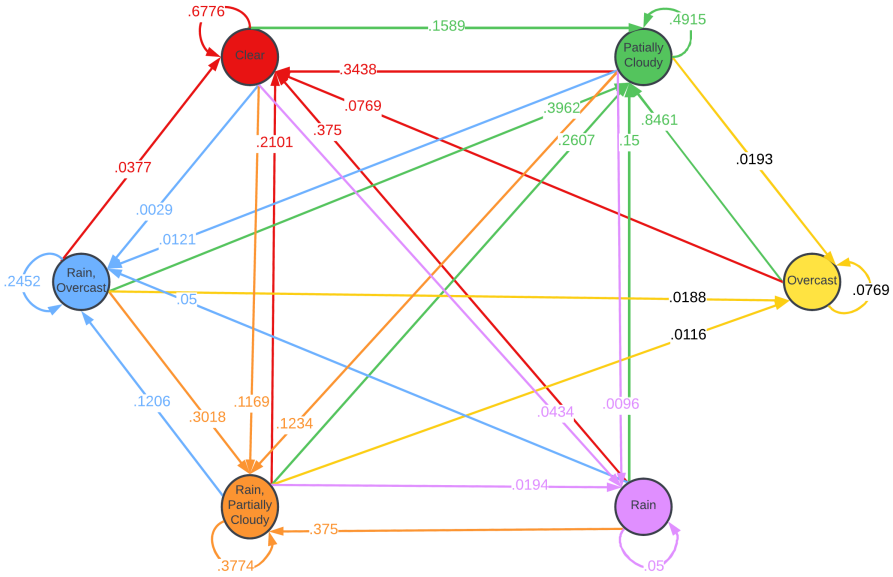


Figure 8: Visualization of Six States Markov Model Trained Over Long Time Frame for Winter

- Spring:

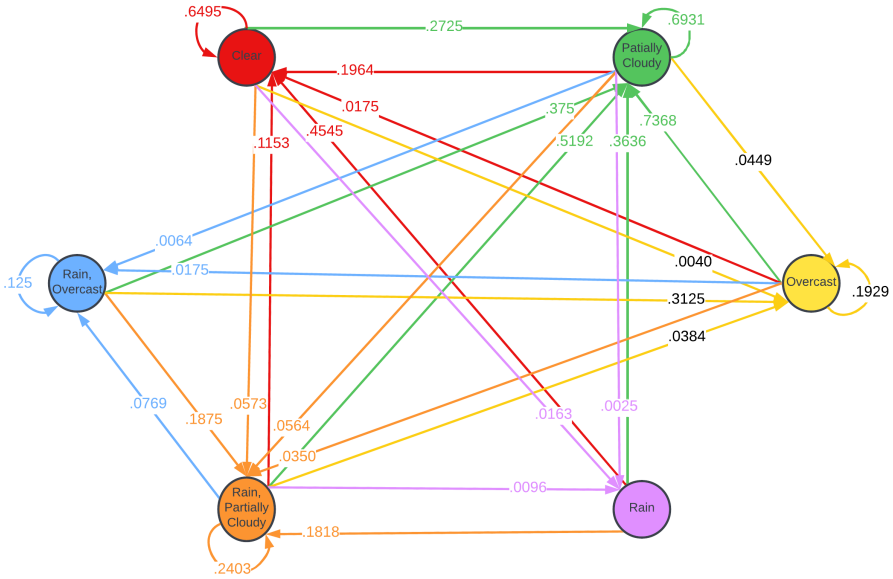


Figure 9: Visualization of Six States Markov Model Trained Over Long Time Frame for Spring

- Summer:

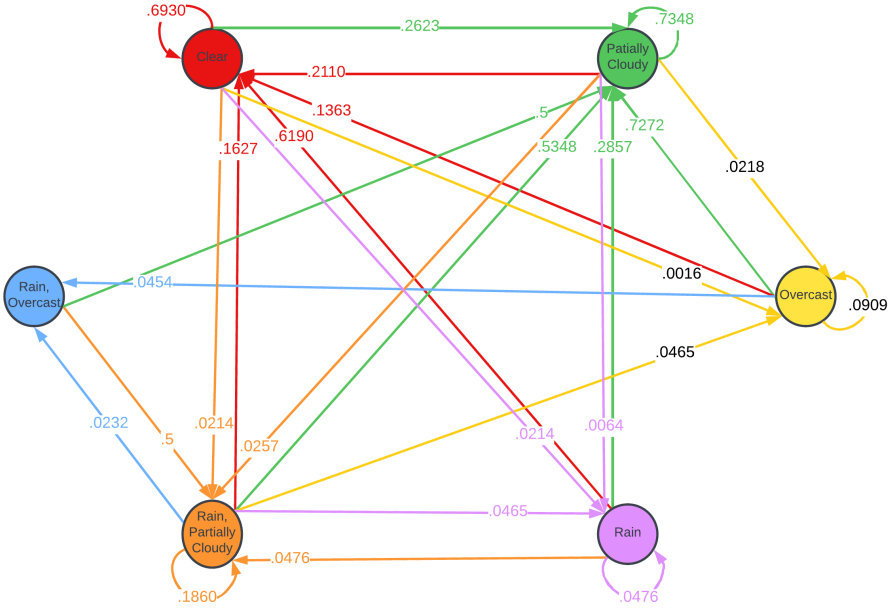


Figure 10: Visualization of Six States Markov Model Trained Over Long Time Frame for Summer

6. All seasons + Simplified applied to UCSD/NYU: Note that the same transition matrix as Figure 3 was applied to UCSD and NYU

Dataset Link:
https://github.com/cjunwon/Math_42_Final_Project/tree/main/Datasets

datacleaning_UCLA

December 7, 2022

1 Data cleaning for UCLA

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

[ ]:
```
raw_data = pd.read_csv('Datasets/2000_01_01_2022_11_01_UCLA.csv')
```

[ ]:
```
raw_data.head()
```

[ ]:
```
    name    datetime  tempmax  tempmin  temp  feelslikemax  feelslikemin  \
0  90024  2000-01-01     13.9      8.1  10.7          13.9           7.7
1  90024  2000-01-02     15.7      8.1  12.4          15.7           6.6
2  90024  2000-01-03     18.6      6.3  11.9          18.6           5.5
3  90024  2000-01-04     19.6      8.2  13.3          19.6           8.2
4  90024  2000-01-05     21.2      7.3  13.9          21.2           7.3

   feelslike  dew  humidity  …  solarenergy  uvindex  severerisk  \
0       10.6  5.5      70.8  …          NaN      NaN         NaN
1       12.2 -0.4      44.4  …          NaN      NaN         NaN
2       11.8 -0.2      46.1  …          NaN      NaN         NaN
3       13.2  1.7      48.3  …          NaN      NaN         NaN
4       13.8  2.2      47.9  …          NaN      NaN         NaN

                sunrise               sunset  moonphase        conditions  \
0  2000-01-01T06:59:26  2000-01-01T16:55:04       0.89  Partially cloudy
1  2000-01-02T06:59:37  2000-01-02T16:55:49       0.93             Clear
2  2000-01-03T06:59:47  2000-01-03T16:56:36       0.96             Clear
3  2000-01-04T06:59:55  2000-01-04T16:57:23       0.98             Clear
4  2000-01-05T07:00:01  2000-01-05T16:58:12       1.00             Clear

                           description               icon  \
0    Partly cloudy throughout the day.  partly-cloudy-day
1  Clear conditions throughout the day.          clear-day
2  Clear conditions throughout the day.          clear-day
3  Clear conditions throughout the day.          clear-day
```

```
4  Clear conditions throughout the day.          clear-day

                              stations
0  72295023174,72287493134,72297023129
1  72295023174,72287493134,72297023129
2  72295023174,72287493134,72297023129
3  72295023174,72287493134,72297023129
4  72295023174,72287493134,72297023129

[5 rows x 33 columns]
```

[ ]:
```
print(raw_data.conditions.unique())
```

```
['Partially cloudy' 'Clear' 'Rain, Partially cloudy' 'Rain'
 'Rain, Overcast' 'Overcast']
```

[ ]:
```
print(raw_data.icon.unique())
```

```
['partly-cloudy-day' 'clear-day' 'rain' 'wind' 'cloudy']
```

[ ]:
```
all_seasons = raw_data[['datetime', 'conditions']]
```

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

[ ]:

	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
5	2000-01-06	Clear
6	2000-01-07	Partially cloudy
7	2000-01-08	Partially cloudy
8	2000-01-09	Partially cloudy
9	2000-01-10	Partially cloudy
10	2000-01-11	Partially cloudy
11	2000-01-12	Partially cloudy
12	2000-01-13	Rain, Partially cloudy
13	2000-01-14	Partially cloudy
14	2000-01-15	Partially cloudy
15	2000-01-16	Rain, Partially cloudy
16	2000-01-17	Rain, Partially cloudy
17	2000-01-18	Partially cloudy
18	2000-01-19	Partially cloudy
19	2000-01-20	Partially cloudy

[ ]:
```
all_seasons['datetime'] = [datetime.strptime(dt, date_format) for dt in␣
 ↪all_seasons['datetime']]
all_seasons['quarter'] = [dt.quarter for dt in all_seasons['datetime']]
```

```
/var/folders/6m/88dwrhnx7m3cybxwl1p0rtq40000gn/T/ipykernel_80665/3833035098.py:1
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  all_seasons['datetime'] = [datetime.strptime(dt, date_format) for dt in
all_seasons['datetime']]
/var/folders/6m/88dwrhnx7m3cybxwl1p0rtq40000gn/T/ipykernel_80665/3833035098.py:2
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  all_seasons['quarter'] = [dt.quarter for dt in all_seasons['datetime']]
```

[ ]:
```
all_seasons.quarter.unique()
```

[ ]:
```
array([1, 2, 3, 4])
```

[ ]:
```
winter = all_seasons[all_seasons.quarter == 1]
spring = all_seasons[all_seasons.quarter == 2]
summer = all_seasons[all_seasons.quarter == 3]
fall = all_seasons[all_seasons.quarter == 4]
```

[ ]:
```
all_seasons.to_csv('all_seasons.csv', encoding='utf-8', index=False)
winter.to_csv('winter.csv', encoding='utf-8', index=False)
spring.to_csv('spring.csv', encoding='utf-8', index=False)
summer.to_csv('summer.csv', encoding='utf-8', index=False)
fall.to_csv('fall.csv', encoding='utf-8', index=False)
```

datacleaning_UCSD

December 7, 2022

1 Data Cleaning for UCSD

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

[ ]:
```
raw_data = pd.read_csv('Datasets/2000_01_01_2022_11_01_UCSD.csv')
```

[ ]:
```
raw_data.head()
```

[ ]:

	name	datetime	tempmax	tempmin	temp	feelslikemax	feelslikemin	\
0	92093	2000-01-01	14.7	10.1	12.4	14.7	10.1	
1	92093	2000-01-02	15.1	9.6	12.9	15.1	9.6	
2	92093	2000-01-03	18.9	6.6	12.1	18.9	6.6	
3	92093	2000-01-04	18.5	7.8	13.1	18.5	6.8	
4	92093	2000-01-05	17.1	7.3	11.7	17.1	7.3	

	feelslike	dew	humidity	…	solarenergy	uvindex	severerisk	\
0	12.4	8.1	75.6	…	NaN	NaN	NaN	
1	12.9	5.8	63.1	…	NaN	NaN	NaN	
2	12.1	2.5	54.9	…	NaN	NaN	NaN	
3	12.9	3.6	54.8	…	NaN	NaN	NaN	
4	11.7	6.0	70.0	…	NaN	NaN	NaN	

	sunrise	sunset	moonphase	\
0	2000-01-01T06:50:28	2000-01-01T16:51:18	0.89	
1	2000-01-02T06:50:40	2000-01-02T16:52:02	0.93	
2	2000-01-03T06:50:51	2000-01-03T16:52:48	0.96	
3	2000-01-04T06:51:00	2000-01-04T16:53:35	0.98	
4	2000-01-05T06:51:07	2000-01-05T16:54:22	1.00	

	conditions	description	\
0	Rain, Partially cloudy	Partly cloudy throughout the day with late aft…	
1	Partially cloudy	Partly cloudy throughout the day.	
2	Clear	Clear conditions throughout the day.	
3	Clear	Clear conditions throughout the day.	

```
4                   Clear              Clear conditions throughout the day.

                icon                  stations
0               rain  72290693112,72290023188
1  partly-cloudy-day  72290693112,72290023188
2          clear-day  72290693112,72290023188
3          clear-day  72290693112,72290023188
4          clear-day  72290693112,72290023188

[5 rows x 33 columns]
```

[ ]:
```
print(raw_data.conditions.unique())
```

```
['Rain, Partially cloudy' 'Partially cloudy' 'Clear' 'Overcast' 'Rain'
 'Rain, Overcast' 'Snow, Rain' 'Snow, Rain, Partially cloudy'
 'Snow, Rain, Overcast' 'Snow']
```

[ ]:
```
print(raw_data.icon.unique())
```

```
['rain' 'partly-cloudy-day' 'clear-day' 'cloudy' 'wind' 'snow']
```

[ ]:
```
all_seasons = raw_data[['datetime', 'conditions']]
```

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

[ ]:
```
     datetime              conditions
0  2000-01-01  Rain, Partially cloudy
1  2000-01-02       Partially cloudy
2  2000-01-03                  Clear
3  2000-01-04                  Clear
4  2000-01-05                  Clear
```

[ ]:
```
# all_seasons['datetime'] = [datetime.strptime(dt, date_format) for dt in␣
 ↪all_seasons['datetime']]
# all_seasons['quarter'] = [dt.quarter for dt in all_seasons['datetime']]
```

```
/var/folders/6m/88dwrhnx7m3cybxwl1p0rtq40000gn/T/ipykernel_69670/3833035098.py:1
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  all_seasons['datetime'] = [datetime.strptime(dt, date_format) for dt in
all_seasons['datetime']]
/var/folders/6m/88dwrhnx7m3cybxwl1p0rtq40000gn/T/ipykernel_69670/3833035098.py:2
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  all_seasons['quarter'] = [dt.quarter for dt in all_seasons['datetime']]
```

[ ]:
```
# all_seasons.quarter.unique()
```

[ ]:
```
array([1, 2, 3, 4])
```

[ ]:
```
# winter = all_seasons[all_seasons.quarter == 1]
# spring = all_seasons[all_seasons.quarter == 2]
# summer = all_seasons[all_seasons.quarter == 3]
# fall = all_seasons[all_seasons.quarter == 4]
```

[ ]:
```
all_seasons.to_csv('Datasets/all_seasons_UCSD.csv', encoding='utf-8',␣
↪index=False)
# winter.to_csv('winter.csv', encoding='utf-8', index=False)
# spring.to_csv('spring.csv', encoding='utf-8', index=False)
# summer.to_csv('summer.csv', encoding='utf-8', index=False)
# fall.to_csv('fall.csv', encoding='utf-8', index=False)
```

datacleaning_NYU

December 7, 2022

1 Data Cleaning for NYU

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

[ ]:
```
raw_data = pd.read_csv('Datasets/2000_01_01_2022_11_01_NYU.csv')
```

[ ]:
```
raw_data.head()
```

[ ]:
```
     name    datetime  tempmax  tempmin  temp  feelslikemax  feelslikemin  \
0  10012  2000-01-01      9.5      1.3   4.6           7.0          -1.9
1  10012  2000-01-02     14.1      4.3   9.6          14.1           4.0
2  10012  2000-01-03     16.5      9.5  13.3          16.5           8.2
3  10012  2000-01-04     18.9      9.1  12.4          18.9           7.5
4  10012  2000-01-05      8.5     -1.0   3.2           5.4          -5.8

   feelslike   dew  humidity  …  solarenergy  uvindex  severerisk  \
0        2.0   3.7      94.1  …          NaN      NaN         NaN
1        9.3   7.9      89.2  …          NaN      NaN         NaN
2       13.2  11.0      86.7  …          NaN      NaN         NaN
3       12.1  11.1      91.7  …          NaN      NaN         NaN
4       -2.0  -5.1      55.5  …          NaN      NaN         NaN

                sunrise               sunset  moonphase  \
0  2000-01-01T07:20:08  2000-01-01T16:38:45       0.89
1  2000-01-02T07:20:15  2000-01-02T16:39:35       0.93
2  2000-01-03T07:20:20  2000-01-03T16:40:27       0.96
3  2000-01-04T07:20:22  2000-01-04T16:41:21       0.98
4  2000-01-05T07:20:22  2000-01-05T16:42:16       1.00

              conditions                                        description  \
0       Partially cloudy                Partly cloudy throughout the day.
1               Overcast                 Cloudy skies throughout the day.
2               Overcast                 Cloudy skies throughout the day.
3         Rain, Overcast        Cloudy skies throughout the day with rain.
```

```
4  Rain, Partially cloudy  Partly cloudy throughout the day with early mo…

                icon                                          stations
0  partly-cloudy-day  72503794745,72502014734,74486094789,72503014732
1             cloudy  72503794745,72502014734,74486094789,72503014732
2             cloudy  72503794745,72502014734,74486094789,72503014732
3               rain  72503794745,72502014734,74486094789,72503014732
4               rain  72503794745,72502014734,74486094789,72503014732

[5 rows x 33 columns]
```

[ ]:
```
print(raw_data.conditions.unique())
```

```
['Partially cloudy' 'Overcast' 'Rain, Overcast' 'Rain, Partially cloudy'
 'Snow, Rain, Partially cloudy' 'Clear' 'Snow, Rain, Overcast'
 'Snow, Partially cloudy' 'Snow, Overcast' 'Rain' 'Snow' 'Snow, Rain'
 'Snow, Rain, Freezing Drizzle/Freezing Rain, Overcast'
 'Rain, Freezing Drizzle/Freezing Rain, Ice, Partially cloudy'
 'Snow, Rain, Ice, Overcast']
```

[ ]:
```
print(raw_data.icon.unique())
```

```
['partly-cloudy-day' 'cloudy' 'rain' 'wind' 'snow' 'clear-day']
```

[ ]:
```
all_seasons = raw_data[['datetime', 'conditions']]
```

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

[ ]:
```
     datetime              conditions
0  2000-01-01        Partially cloudy
1  2000-01-02                Overcast
2  2000-01-03                Overcast
3  2000-01-04          Rain, Overcast
4  2000-01-05  Rain, Partially cloudy
```

[ ]:
```
# all_seasons['datetime'] = [datetime.strptime(dt, date_format) for dt in␣
↪all_seasons['datetime']]
# all_seasons['quarter'] = [dt.quarter for dt in all_seasons['datetime']]
```

[ ]:
```
# all_seasons.quarter.unique()
```

[ ]:
```
# winter = all_seasons[all_seasons.quarter == 1]
# spring = all_seasons[all_seasons.quarter == 2]
# summer = all_seasons[all_seasons.quarter == 3]
# fall = all_seasons[all_seasons.quarter == 4]
```

[ ]:
```
all_seasons.to_csv('Datasets/all_seasons_NYU.csv', encoding='utf-8',␣
↪index=False)
# winter.to_csv('winter.csv', encoding='utf-8', index=False)
# spring.to_csv('spring.csv', encoding='utf-8', index=False)
# summer.to_csv('summer.csv', encoding='utf-8', index=False)
# fall.to_csv('fall.csv', encoding='utf-8', index=False)
```

all_seasons_simplified

December 7, 2022

1 All Seasons - Simplified(long time frame)

1.1 Import libraries and dataset

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

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

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

[ ]:

	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

1.2 Classify and separate data

[ ]:
```
simplifier = {'Overcast':'no_rain', 'Partially cloudy':'no_rain', 'Clear':
 ↪'no_rain', 'Rain, Partially cloudy':'rain', 'Rain':'rain', 'Rain, Overcast':
 ↪'rain'}

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

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

[ ]:

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

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

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

[ ]:
```
     datetime condition
0  2000-01-01   no_rain
1  2000-01-02   no_rain
2  2000-01-03   no_rain
3  2000-01-04   no_rain
4  2000-01-05   no_rain
```

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
all_seasons_train = all_seasons.loc[all_seasons['datetime'].
 ↪between(train_start_date, train_end_date)]
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_start_date, test_end_date)]
all_seasons_test = all_seasons_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

We will refer to rain is 'R' and no rain as 'N'

[ ]:
```
# Initialize count variables
R_after_R_count = 0.0
N_after_R_count = 0.0

R_after_N_count = 0.0
N_after_N_count = 0.0
```

[ ]:
```
all_seasons_train
```

[ ]:
```
      index    datetime condition
0       731  2002-01-01   no_rain
1       732  2002-01-02      rain
2       733  2002-01-03      rain
3       734  2002-01-04   no_rain
4       735  2002-01-05   no_rain
…     …          …       …
5839   6570  2017-12-27   no_rain
5840   6571  2017-12-28   no_rain
5841   6572  2017-12-29   no_rain
5842   6573  2017-12-30   no_rain
```

```
5843   6574  2017-12-31   no_rain

[5844 rows x 3 columns]
```

[ ]:
```
# Count conditions

all_seasons_train['condition_shift'] = all_seasons_train['condition'].shift(-1)

for i in range(len(all_seasons_train)):
    if all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_train.
 ↪loc[i, 'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_train.loc[i, 'condition_shift'] == 'rain':
        N_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_train.
 ↪loc[i, 'condition_shift'] == 'no_rain':
        R_after_N_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_train.loc[i, 'condition_shift'] == 'no_rain':
        N_after_N_count += 1
```

[ ]:
```
current_R_total = R_after_R_count + N_after_R_count
current_N_total = R_after_N_count + N_after_N_count
```

[ ]:
```
R_after_R_prob = R_after_R_count / current_R_total
N_after_R_prob = N_after_R_count / current_R_total

R_after_N_prob = R_after_N_count / current_N_total
N_after_N_prob = N_after_N_count / current_N_total
```

[ ]:
```
# Printing our probabilities for 2x2 transition matrix:
print(R_after_R_prob)
print(N_after_R_prob)
print(R_after_N_prob)
print(N_after_N_prob)
```

```
0.4674887892376682
0.5325112107623319
0.09594021409816199
0.904059785901838
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(R_after_R_prob + N_after_R_prob)
print(R_after_N_prob + N_after_N_prob)
```

```
1.0
```

```
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_name = [['RR', 'RN'], ['RN', 'NN']]
transition_matrix = [[R_after_R_prob, N_after_R_prob], [R_after_N_prob,␣
 ↪N_after_N_prob]]
print(transition_matrix)
```

```
[[0.4674887892376682, 0.5325112107623319], [0.09594021409816199,
0.904059785901838]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.46748879 0.53251121]
 [0.09594021 0.90405979]]
```

First Day of 2018: No Rain

[ ]:
```
def predict_weather_simplified(test_data):
    state = {0:'rain', 1:'no_rain'}
    n = len(test_data) #how many steps to test
    start_state = 1 #1 = No Rain
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1], 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], 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_simplified(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_simplified(all_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   6575  2018-01-01   no_rain             no_rain
1   6576  2018-01-02   no_rain             no_rain
2   6577  2018-01-03   no_rain             no_rain
3   6578  2018-01-04   no_rain             no_rain
4   6579  2018-01-05   no_rain             no_rain
0.7652292950034223
```

[ ]:
```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

[ ]:
```
0.7703011635865848
```

all_seasons_simplified_short

December 7, 2022

1 All Seasons - Simplified(short time frame)

1.1 Import libraries and dataset

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

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

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

[ ]:

	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

1.2 Classify and separate data

[ ]:
```
simplifier = {'Overcast':'no_rain', 'Partially cloudy':'no_rain', 'Clear':
 ↪'no_rain', 'Rain, Partially cloudy':'rain', 'Rain':'rain', 'Rain, Overcast':
 ↪'rain'}

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

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

[ ]:

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

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

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

[ ]:
```
     datetime condition
0  2000-01-01   no_rain
1  2000-01-02   no_rain
2  2000-01-03   no_rain
3  2000-01-04   no_rain
4  2000-01-05   no_rain
```

[ ]:
```
train_start_date = '2017-01-01'
train_end_date = '2020-12-31'
all_seasons_train = all_seasons.loc[all_seasons['datetime'].
 ↪between(train_start_date, train_end_date)]
all_seasons_train = all_seasons_train.reset_index()

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

1.3 Calculate proportions of conditions & Create transition matrix

We will refer to rain is 'R' and no rain as 'N'

[ ]:
```
# Initialize count variables
R_after_R_count = 0.0
N_after_R_count = 0.0

R_after_N_count = 0.0
N_after_N_count = 0.0
```

[ ]:
```
all_seasons_train
```

[ ]:
```
      index    datetime condition
0      5844  2016-01-01   no_rain
1      5845  2016-01-02   no_rain
2      5846  2016-01-03   no_rain
3      5847  2016-01-04      rain
4      5848  2016-01-05      rain
…       …          …         …
1456   7300  2019-12-27   no_rain
1457   7301  2019-12-28   no_rain
1458   7302  2019-12-29   no_rain
1459   7303  2019-12-30   no_rain
```

```
1460   7304  2019-12-31   no_rain

[1461 rows x 3 columns]
```

[ ]:
```
# Count conditions

all_seasons_train['condition_shift'] = all_seasons_train['condition'].shift(-1)

for i in range(len(all_seasons_train)):
    if all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_train.
 ↪loc[i, 'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_train.loc[i, 'condition_shift'] == 'rain':
        N_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_train.
 ↪loc[i, 'condition_shift'] == 'no_rain':
        R_after_N_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_train.loc[i, 'condition_shift'] == 'no_rain':
        N_after_N_count += 1
```

[ ]:
```
current_R_total = R_after_R_count + N_after_R_count
current_N_total = R_after_N_count + N_after_N_count
```

[ ]:
```
R_after_R_prob = R_after_R_count / current_R_total
N_after_R_prob = N_after_R_count / current_R_total

R_after_N_prob = R_after_N_count / current_N_total
N_after_N_prob = N_after_N_count / current_N_total
```

[ ]:
```
# Printing our probabilities for 2x2 transition matrix:
print(R_after_R_prob)
print(N_after_R_prob)
print(R_after_N_prob)
print(N_after_N_prob)
```

```
0.49523809523809526
0.5047619047619047
0.0848
0.9152
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(R_after_R_prob + N_after_R_prob)
print(R_after_N_prob + N_after_N_prob)
```

```
1.0
```

```
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_name = [['RR', 'NR'], ['RN', 'NN']]
transition_matrix = [[R_after_R_prob, N_after_R_prob], [R_after_N_prob,␣
 ↪N_after_N_prob]]
print(transition_matrix)
```

```
[[0.49523809523809526, 0.5047619047619047], [0.0848, 0.9152]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.4952381 0.5047619]
 [0.0848    0.9152   ]]
```

First Day of 2018: No Rain

[ ]:
```
def predict_weather_simplified(test_data):
    state = {0:'rain', 1:'no_rain'}
    n = len(test_data) #how many steps to test
    start_state = 1 #1 = No Rain
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1], 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], 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_simplified(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_simplified(all_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   7305  2020-01-01   no_rain             no_rain
1   7306  2020-01-02   no_rain             no_rain
2   7307  2020-01-03   no_rain             no_rain
3   7308  2020-01-04   no_rain             no_rain
4   7309  2020-01-05   no_rain             no_rain
0.819672131147541
```

[ ]:
```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

[ ]: 0.7933879781420766

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 [ ]:

All Seasons - 6 different weather conditions(long time frame)

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

We will refer to rain is 'R' and no rain as 'N'

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

C_after_C_count = 0.0
PC_after_C_count = 0.0
OV_after_C_count = 0.0
R_after_C_count = 0.0
RPC_after_C_count = 0.0
ROV_after_C_count = 0.0

C_after_PC_count = 0.0
PC_after_PC_count = 0.0
OV_after_PC_count = 0.0
R_after_PC_count = 0.0
RPC_after_PC_count = 0.0
ROV_after_PC_count = 0.0

C_after_OV_count = 0.0
PC_after_OV_count = 0.0
OV_after_OV_count = 0.0
R_after_OV_count = 0.0
RPC_after_OV_count = 0.0
ROV_after_OV_count = 0.0

C_after_R_count = 0.0
PC_after_R_count = 0.0
OV_after_R_count = 0.0
R_after_R_count = 0.0
RPC_after_R_count = 0.0
ROV_after_R_count = 0.0

C_after_RPC_count = 0.0
PC_after_RPC_count = 0.0
OV_after_RPC_count = 0.0
R_after_RPC_count = 0.0
RPC_after_RPC_count = 0.0
ROV_after_RPC_count = 0.0

C_after_ROV_count = 0.0
PC_after_ROV_count = 0.0
OV_after_ROV_count = 0.0
R_after_ROV_count = 0.0
RPC_after_ROV_count = 0.0
ROV_after_ROV_count = 0.0
```

In [ ]:

```
all_seasons_train
```

Out[ ]:

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

5844 rows × 3 columns

In [ ]:

```
# Count conditions

all_seasons_train['condition_shift'] = all_seasons_train['condition'].shift

for i in range(len(all_seasons_train)):
    # Current 'clear'
    if all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_tra
        C_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_ti
        R_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_C_count += 1
    # Current 'partially_cloudy'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_ti
        R_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_PC_count += 1
    # Current 'overcast'
```

```
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_OV_count += 1
    # Current 'rain'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_R_count += 1
    # Current 'rain_partially_cloudy'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_RPC_count += 1
    # Current 'rain_overcast'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_ROV_count += 1
```

In [ ]:

```
current_C_total = C_after_C_count + PC_after_C_count + OV_after_C_count + R
current_PC_total = C_after_PC_count + PC_after_PC_count + OV_after_PC_count
current_OV_total = C_after_OV_count + PC_after_OV_count + OV_after_OV_count
current_R_total =C_after_R_count + PC_after_R_count + OV_after_R_count + R_
current_RPC_total = C_after_RPC_count + PC_after_RPC_count + OV_after_RPC_c
current_ROV_total = C_after_ROV_count + PC_after_ROV_count + OV_after_ROV_c
```

In [ ]:

```
C_after_C_prob = C_after_C_count / current_C_total
PC_after_C_prob = PC_after_C_count / current_C_total
OV_after_C_prob = OV_after_C_count / current_C_total
R_after_C_prob = R_after_C_count / current_C_total
RPC_after_C_prob = RPC_after_C_count / current_C_total
ROV_after_C_prob = ROV_after_C_count / current_C_total

C_after_PC_prob = C_after_PC_count / current_PC_total
PC_after_PC_prob = PC_after_PC_count / current_PC_total
OV_after_PC_prob = OV_after_PC_count / current_PC_total
R_after_PC_prob = R_after_PC_count / current_PC_total
RPC_after_PC_prob = RPC_after_PC_count / current_PC_total
ROV_after_PC_prob = ROV_after_PC_count / current_PC_total

C_after_OV_prob = C_after_OV_count / current_OV_total
PC_after_OV_prob = PC_after_OV_count / current_OV_total
OV_after_OV_prob = OV_after_OV_count / current_OV_total
R_after_OV_prob = R_after_OV_count / current_OV_total
RPC_after_OV_prob = RPC_after_OV_count / current_OV_total
ROV_after_OV_prob = ROV_after_OV_count / current_OV_total

C_after_R_prob = C_after_R_count / current_R_total
PC_after_R_prob = PC_after_R_count / current_R_total
OV_after_R_prob = OV_after_R_count / current_R_total
R_after_R_prob = R_after_R_count / current_R_total
RPC_after_R_prob = RPC_after_R_count / current_R_total
ROV_after_R_prob = ROV_after_R_count / current_R_total

C_after_RPC_prob = C_after_RPC_count / current_RPC_total
PC_after_RPC_prob = PC_after_RPC_count / current_RPC_total
OV_after_RPC_prob = OV_after_RPC_count / current_RPC_total
R_after_RPC_prob = R_after_RPC_count / current_RPC_total
RPC_after_RPC_prob = RPC_after_RPC_count / current_RPC_total
ROV_after_RPC_prob = ROV_after_RPC_count / current_RPC_total

C_after_ROV_prob = C_after_ROV_count / current_ROV_total
PC_after_ROV_prob = PC_after_ROV_count / current_ROV_total
OV_after_ROV_prob = OV_after_ROV_count / current_ROV_total
R_after_ROV_prob = R_after_ROV_count / current_ROV_total
RPC_after_ROV_prob = RPC_after_ROV_count / current_ROV_total
ROV_after_ROV_prob = ROV_after_ROV_count / current_ROV_total
```

In [ ]:

```
# Printing our probabilities for 6x6 transition matrix:
print(C_after_C_prob)
print(PC_after_C_prob)
print(OV_after_C_prob)
print(R_after_C_prob)
print(RPC_after_C_prob)
print(ROV_after_C_prob)

print(C_after_PC_prob)
print(PC_after_PC_prob)
print(OV_after_PC_prob)
print(R_after_PC_prob)
print(RPC_after_PC_prob)
print(ROV_after_PC_prob)

print(C_after_OV_prob)
print(PC_after_OV_prob)
print(OV_after_OV_prob)
print(R_after_OV_prob)
print(RPC_after_OV_prob)
print(ROV_after_OV_prob)

print(C_after_R_prob)
print(PC_after_R_prob)
print(OV_after_R_prob)
print(R_after_R_prob)
print(RPC_after_R_prob)
print(ROV_after_R_prob)

print(C_after_RPC_prob)
print(PC_after_RPC_prob)
print(OV_after_RPC_prob)
print(R_after_RPC_prob)
print(RPC_after_RPC_prob)
print(ROV_after_RPC_prob)

print(C_after_ROV_prob)
print(PC_after_ROV_prob)
print(OV_after_ROV_prob)
print(R_after_ROV_prob)
print(RPC_after_ROV_prob)
print(ROV_after_ROV_prob)
```

```
0.6896135265700483
0.19927536231884058
0.0008051529790660225
0.033816425120772944
0.07286634460547504
0.0036231884057971015
0.2502120441051739
0.6395250212044106
0.028413910093299407
0.008905852417302799
0.06658184902459711
0.006361323155216285
0.027522935779816515
0.7431192660550459
0.1559633027522936
0.0
0.045871559633027525
0.027522935779816515
0.36551724137931035
0.21379310344827587
0.0
0.14482758620689656
0.2482758620689655
0.027586206896551724
0.19032258064516128
0.3225806451612903
0.024193548387096774
0.02903225806451613
0.33548387096774196
0.09838709677419355
0.05511811023622047
0.33858267716535434
0.06299212598425197
0.007874015748031496
0.25984251968503935
0.2755905511811024
```

In [ ]:

```
# Checking that each row in the transition matrix adds up to 1:
print(C_after_C_prob + PC_after_C_prob + OV_after_C_prob + R_after_C_prob +
print(C_after_PC_prob + PC_after_PC_prob + OV_after_PC_prob + R_after_PC_pr
print(C_after_OV_prob + PC_after_OV_prob + OV_after_OV_prob + R_after_OV_pr
print(C_after_R_prob + PC_after_R_prob + OV_after_R_prob + R_after_R_prob +
print(C_after_RPC_prob + PC_after_RPC_prob + OV_after_RPC_prob + R_after_RP
print(C_after_ROV_prob + PC_after_ROV_prob + OV_after_ROV_prob + R_after_RO
```

```
1.0
1.0
1.0
1.0000000000000002
1.0
1.0
```

In [ ]:
```
# Creating the transition matrix:
transition_matrix = [[C_after_C_prob, PC_after_C_prob, OV_after_C_prob, R_a
                     [C_after_PC_prob, PC_after_PC_prob, OV_after_PC_prob, R
                     [C_after_OV_prob, PC_after_OV_prob, OV_after_OV_prob, R
                     [C_after_R_prob, PC_after_R_prob, OV_after_R_prob, R_a
                     [C_after_RPC_prob, PC_after_RPC_prob, OV_after_RPC_prob
                     [C_after_ROV_prob, PC_after_ROV_prob, OV_after_ROV_prob
print(transition_matrix)
```

```
[[0.6896135265700483, 0.19927536231884058, 0.0008051529790660225, 0.0338164
25120772944, 0.07286634460547504, 0.003623188405797101 5], [0.25021204410517
39, 0.6395250212044106, 0.028413910093299407, 0.008905852417302799, 0.06658
184902459711, 0.006361323155216285], [0.027522935779816515, 0.7431192660550
459, 0.1559633027522936, 0.0, 0.045871559633027525, 0.027522935779816515],
[0.36551724137931035, 0.21379310344827587, 0.0, 0.14482758620689656, 0.2482
758620689655, 0.027586206896551724], [0.19032258064516128, 0.32258064516129
03, 0.024193548387096774, 0.02903225806451613, 0.33548387096774196, 0.09838
709677419355], [0.05511811023622047, 0.33858267716535434, 0.062992125984251
97, 0.007874015748031496, 0.25984251968503935, 0.2755905511811024]]
```

In [ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.68961353 0.19927536 0.00080515 0.03381643 0.07286634 0.00362319]
 [0.25021204 0.63952502 0.02841391 0.00890585 0.06658185 0.00636132]
 [0.02752294 0.74311927 0.1559633  0.         0.04587156 0.02752294]
 [0.36551724 0.2137931  0.         0.14482759 0.24827586 0.02758621]
 [0.19032258 0.32258065 0.02419355 0.02903226 0.33548387 0.0983871 ]
 [0.05511811 0.33858268 0.06299213 0.00787402 0.25984252 0.27559055]]
```

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

Out[ ]:

	index	datetime	condition
0	6575	2018-01-01	clear

First Day of 2018: clear

In [ ]:

```
def predict_weather_six_conditions(test_data):
    state = {0:'clear', 1:'partially_cloudy', 2:'overcast', 3:'rain', 4:'ra
    n = len(test_data) # how many steps to test
    start_state = 0 # 0 = clear
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1,2,3,4,5], p=t_array[prev_state]
        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]) #
    # 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,
            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
```

In [ ]:

```
# 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)
```

```
   index    datetime         condition predicted_condition
0   6575  2018-01-01             clear               clear
1   6576  2018-01-02             clear               clear
2   6577  2018-01-03             clear               clear
3   6578  2018-01-04  partially_cloudy    partially_cloudy
4   6579  2018-01-05  partially_cloudy                rain
0.3750855578370979
```

In [ ]:
```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

Out[ ]:
```
0.36932238193018485
```

All Seaons - 6 different weather conditions(short time frame)

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 = '2017-01-01'
train_end_date = '2020-12-31'
all_seasons_train = all_seasons.loc[all_seasons['datetime'].between(train_s
all_seasons_train = all_seasons_train.reset_index()

test_start_date = '2021-01-01'
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

We will refer to rain is 'R' and no rain as 'N'

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

C_after_C_count = 0.0
PC_after_C_count = 0.0
OV_after_C_count = 0.0
R_after_C_count = 0.0
RPC_after_C_count = 0.0
ROV_after_C_count = 0.0

C_after_PC_count = 0.0
PC_after_PC_count = 0.0
OV_after_PC_count = 0.0
R_after_PC_count = 0.0
RPC_after_PC_count = 0.0
ROV_after_PC_count = 0.0

C_after_OV_count = 0.0
PC_after_OV_count = 0.0
OV_after_OV_count = 0.0
R_after_OV_count = 0.0
RPC_after_OV_count = 0.0
ROV_after_OV_count = 0.0

C_after_R_count = 0.0
PC_after_R_count = 0.0
OV_after_R_count = 0.0
R_after_R_count = 0.0
RPC_after_R_count = 0.0
ROV_after_R_count = 0.0

C_after_RPC_count = 0.0
PC_after_RPC_count = 0.0
OV_after_RPC_count = 0.0
R_after_RPC_count = 0.0
RPC_after_RPC_count = 0.0
ROV_after_RPC_count = 0.0

C_after_ROV_count = 0.0
PC_after_ROV_count = 0.0
OV_after_ROV_count = 0.0
R_after_ROV_count = 0.0
RPC_after_ROV_count = 0.0
ROV_after_ROV_count = 0.0
```

In [ ]:

```
all_seasons_train
```

Out[ ]:

	index	datetime	condition
0	6210	2017-01-01	partially_cloudy
1	6211	2017-01-02	partially_cloudy
2	6212	2017-01-03	partially_cloudy
3	6213	2017-01-04	rain_overcast
4	6214	2017-01-05	rain_partially_cloudy
...	...	...	...
1456	7666	2020-12-27	partially_cloudy
1457	7667	2020-12-28	rain_partially_cloudy
1458	7668	2020-12-29	clear
1459	7669	2020-12-30	clear
1460	7670	2020-12-31	clear

1461 rows × 3 columns

In [ ]:

```
# Count conditions

all_seasons_train['condition_shift'] = all_seasons_train['condition'].shift

for i in range(len(all_seasons_train)):
    # Current 'clear'
    if all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_tra
        C_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_ti
        R_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_C_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_C_count += 1
    # Current 'partially_cloudy'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_ti
        R_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_PC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_PC_count += 1
    # Current 'overcast'
```

```
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_OV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_OV_count += 1
    # Current 'rain'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_R_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_R_count += 1
    # Current 'rain_partially_cloudy'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_RPC_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_RPC_count += 1
    # Current 'rain_overcast'
    elif all_seasons_train.loc[i, 'condition'] == 'clear' and all_seasons_t
        C_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and al
        PC_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'overcast' and all_seasor
        OV_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain' and all_seasons_tr
        R_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy' a
        RPC_after_ROV_count += 1
    elif all_seasons_train.loc[i, 'condition'] == 'rain_overcast' and all_s
        ROV_after_ROV_count += 1
```

In [ ]:

```
current_C_total = C_after_C_count + PC_after_C_count + OV_after_C_count + R
current_PC_total = C_after_PC_count + PC_after_PC_count + OV_after_PC_count
current_OV_total = C_after_OV_count + PC_after_OV_count + OV_after_OV_count
current_R_total =C_after_R_count + PC_after_R_count + OV_after_R_count + R_
current_RPC_total = C_after_RPC_count + PC_after_RPC_count + OV_after_RPC_c
current_ROV_total = C_after_ROV_count + PC_after_ROV_count + OV_after_ROV_c
```

In [ ]:

```
C_after_C_prob = C_after_C_count / current_C_total
PC_after_C_prob = PC_after_C_count / current_C_total
OV_after_C_prob = OV_after_C_count / current_C_total
R_after_C_prob = R_after_C_count / current_C_total
RPC_after_C_prob = RPC_after_C_count / current_C_total
ROV_after_C_prob = ROV_after_C_count / current_C_total

C_after_PC_prob = C_after_PC_count / current_PC_total
PC_after_PC_prob = PC_after_PC_count / current_PC_total
OV_after_PC_prob = OV_after_PC_count / current_PC_total
R_after_PC_prob = R_after_PC_count / current_PC_total
RPC_after_PC_prob = RPC_after_PC_count / current_PC_total
ROV_after_PC_prob = ROV_after_PC_count / current_PC_total

C_after_OV_prob = C_after_OV_count / current_OV_total
PC_after_OV_prob = PC_after_OV_count / current_OV_total
OV_after_OV_prob = OV_after_OV_count / current_OV_total
R_after_OV_prob = R_after_OV_count / current_OV_total
RPC_after_OV_prob = RPC_after_OV_count / current_OV_total
ROV_after_OV_prob = ROV_after_OV_count / current_OV_total

C_after_R_prob = C_after_R_count / current_R_total
PC_after_R_prob = PC_after_R_count / current_R_total
OV_after_R_prob = OV_after_R_count / current_R_total
R_after_R_prob = R_after_R_count / current_R_total
RPC_after_R_prob = RPC_after_R_count / current_R_total
ROV_after_R_prob = ROV_after_R_count / current_R_total

C_after_RPC_prob = C_after_RPC_count / current_RPC_total
PC_after_RPC_prob = PC_after_RPC_count / current_RPC_total
OV_after_RPC_prob = OV_after_RPC_count / current_RPC_total
R_after_RPC_prob = R_after_RPC_count / current_RPC_total
RPC_after_RPC_prob = RPC_after_RPC_count / current_RPC_total
ROV_after_RPC_prob = ROV_after_RPC_count / current_RPC_total

C_after_ROV_prob = C_after_ROV_count / current_ROV_total
PC_after_ROV_prob = PC_after_ROV_count / current_ROV_total
OV_after_ROV_prob = OV_after_ROV_count / current_ROV_total
R_after_ROV_prob = R_after_ROV_count / current_ROV_total
RPC_after_ROV_prob = RPC_after_ROV_count / current_ROV_total
ROV_after_ROV_prob = ROV_after_ROV_count / current_ROV_total
```

In [ ]:

```
# Printing our probabilities for 6x6 transition matrix:
print(C_after_C_prob)
print(PC_after_C_prob)
print(OV_after_C_prob)
print(R_after_C_prob)
print(RPC_after_C_prob)
print(ROV_after_C_prob)

print(C_after_PC_prob)
print(PC_after_PC_prob)
print(OV_after_PC_prob)
print(R_after_PC_prob)
print(RPC_after_PC_prob)
print(ROV_after_PC_prob)

print(C_after_OV_prob)
print(PC_after_OV_prob)
print(OV_after_OV_prob)
print(R_after_OV_prob)
print(RPC_after_OV_prob)
print(ROV_after_OV_prob)

print(C_after_R_prob)
print(PC_after_R_prob)
print(OV_after_R_prob)
print(R_after_R_prob)
print(RPC_after_R_prob)
print(ROV_after_R_prob)

print(C_after_RPC_prob)
print(PC_after_RPC_prob)
print(OV_after_RPC_prob)
print(R_after_RPC_prob)
print(RPC_after_RPC_prob)
print(ROV_after_RPC_prob)

print(C_after_ROV_prob)
print(PC_after_ROV_prob)
print(OV_after_ROV_prob)
print(R_after_ROV_prob)
print(RPC_after_ROV_prob)
print(ROV_after_ROV_prob)
```

```
0.7429775280898876
0.1853932584269663
0.0
0.008426966292134831
0.06320224719101124
0.0
0.27706422018348625
0.6220183486238532
0.03669724770642202
0.0
0.05871559633027523
0.005504587155963303
0.03225806451612903
0.7741935483870968
0.1935483870967742
0.0
0.0
0.0
0.2857142857142857
0.2857142857142857
0.0
0.0
0.42857142857142855
0.0
0.18
0.26666666666666666
0.02
0.006666666666666667
0.44666666666666666
0.08
0.06666666666666667
0.6
0.13333333333333333
0.0
0.2
0.0
```

In [ ]:

```
# Checking that each row in the transition matrix adds up to 1:
print(C_after_C_prob + PC_after_C_prob + OV_after_C_prob + R_after_C_prob +
print(C_after_PC_prob + PC_after_PC_prob + OV_after_PC_prob + R_after_PC_pr
print(C_after_OV_prob + PC_after_OV_prob + OV_after_OV_prob + R_after_OV_pr
print(C_after_R_prob + PC_after_R_prob + OV_after_R_prob + R_after_R_prob +
print(C_after_RPC_prob + PC_after_RPC_prob + OV_after_RPC_prob + R_after_RP
print(C_after_ROV_prob + PC_after_ROV_prob + OV_after_ROV_prob + R_after_RO
```

```
0.9999999999999999
1.0
1.0
1.0
0.9999999999999999
1.0
```

In [ ]:
```
# Creating the transition matrix:
transition_matrix = [[C_after_C_prob, PC_after_C_prob, OV_after_C_prob, R_a
                     [C_after_PC_prob, PC_after_PC_prob, OV_after_PC_prob, R
                     [C_after_OV_prob, PC_after_OV_prob, OV_after_OV_prob, R
                     [C_after_R_prob, PC_after_R_prob, OV_after_R_prob, R_af
                     [C_after_RPC_prob, PC_after_RPC_prob, OV_after_RPC_prob
                     [C_after_ROV_prob, PC_after_ROV_prob, OV_after_ROV_prob
print(transition_matrix)
```

```
[[0.7429775280898876, 0.1853932584269663, 0.0, 0.008426966292134831, 0.0632
0224719101124, 0.0], [0.27706422018348625, 0.6220183486238532, 0.0366972477
0642202, 0.0, 0.05871559633027523, 0.005504587155963303], [0.03225806451612
903, 0.7741935483870968, 0.1935483870967742, 0.0, 0.0, 0.0], [0.28571428571
42857, 0.2857142857142857, 0.0, 0.0, 0.42857142857142855, 0.0], [0.18, 0.26
666666666666666, 0.02, 0.006666666666666667, 0.4666666666666666, 0.08], [0
.06666666666666667, 0.6, 0.13333333333333333, 0.0, 0.2, 0.0]]
```

In [ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.74297753 0.18539326 0.         0.00842697 0.06320225 0.        ]
 [0.27706422 0.62201835 0.03669725 0.         0.0587156  0.00550459]
 [0.03225806 0.77419355 0.19354839 0.         0.         0.        ]
 [0.28571429 0.28571429 0.         0.         0.42857143 0.        ]
 [0.18       0.26666667 0.02       0.00666667 0.44666667 0.08      ]
 [0.06666667 0.6        0.13333333 0.         0.2        0.        ]]
```

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

Out[ ]:

	index	datetime	condition
0	7671	2021-01-01	clear

First Day of 2018: clear

In [ ]:

```
def predict_weather_six_conditions(test_data):
    state = {0:'clear', 1:'partially_cloudy', 2:'overcast', 3:'rain', 4:'ra
    n = len(test_data) # how many steps to test
    start_state = 0 # 0 = clear
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1,2,3,4,5], p=t_array[prev_state]
        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]) #
    # 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,
            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
```

In [ ]:

```
# 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)
```

```
   index    datetime         condition    predicted_condition
0   7671  2021-01-01             clear                  clear
1   7672  2021-01-02             clear  rain_partially_cloudy
2   7673  2021-01-03  partially_cloudy  rain_partially_cloudy
3   7674  2021-01-04  partially_cloudy               overcast
4   7675  2021-01-05  partially_cloudy       partially_cloudy
0.3698630136986301
```

In [ ]:
```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

Out[ ]:
```
0.38753424657534247
```

all_seasons_simplified_second_order

December 7, 2022

1 All Seasons - Simplified(long time frame - Second-Order Markov Chain)

1.1 Import libraries and dataset

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

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

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

[ ]:

	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

1.2 Classify and separate data

[ ]:
```
simplifier = {'Overcast':'no_rain', 'Partially cloudy':'no_rain', 'Clear':
 ↪'no_rain', 'Rain, Partially cloudy':'rain', 'Rain':'rain', 'Rain, Overcast':
 ↪'rain'}

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

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

[ ]:

	datetime	conditions	condition
0	2000-01-01	Partially cloudy	no_rain
1	2000-01-02	Clear	no_rain
2	2000-01-03	Clear	no_rain
3	2000-01-04	Clear	no_rain

```
4  2000-01-05            Clear   no_rain
```

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

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

[ ]:
```
     datetime condition
0  2000-01-01   no_rain
1  2000-01-02   no_rain
2  2000-01-03   no_rain
3  2000-01-04   no_rain
4  2000-01-05   no_rain
```

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
all_seasons_train = all_seasons.loc[all_seasons['datetime'].
 ↪between(train_start_date, train_end_date)]
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_start_date, test_end_date)]
all_seasons_test = all_seasons_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

[ ]:
```
# 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 = ['rain', 'no_rain']
prev_conditions = [f"{state_0}->{state_1}" for state_0 in conditions for␣
 ↪state_1 in conditions]
prev_conditions
```

[ ]:
```
['rain->rain', 'rain->no_rain', 'no_rain->rain', 'no_rain->no_rain']
```

[ ]:
```
# 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
```

[ ]:

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

[5844 rows x 4 columns]

[ ]:
```
# 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.
↪condition == conditions[j]) & (all_seasons_train.prev_states ==␣
↪prev_conditions[i])])

transition_counts
```

[ ]:
```
array([[ 209.,  208.],
       [  73.,  402.],
       [ 208.,  267.],
       [ 401., 4074.]])
```

[ ]:
```
# 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
```

[ ]:
```
array([[0.50119904, 0.49880096],
       [0.15368421, 0.84631579],
       [0.43789474, 0.56210526],
       [0.08960894, 0.91039106]])
```

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

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

[ ]:
```
all_seasons_test.head(2)
```

[ ]:
```
   index    datetime condition
0   6575  2018-01-01   no_rain
1   6576  2018-01-02   no_rain
```

First day of 2018: no_rain->no_rain

[ ]:
```
tuple_to_row = {('rain','rain'):0, ('rain','no_rain'):1, ('no_rain','rain'):2,␣
↪('no_rain','no_rain'):3}
tuple_to_row[('rain','rain')]
```

[ ]:
```
0
```

[ ]:
```
def predict_weather_simplified(test_data):
    state = {0:'rain', 1:'no_rain'}
    prev_conditions = {0: 'rain->rain', 1:'rain->no_rain', 2:'no_rain->rain', 3:
↪'no_rain->no_rain'}
    tuple_to_row = {('rain','rain'):0, ('rain','no_rain'):1, ('no_rain','rain'):
↪2, ('no_rain','no_rain'):3}

    n = len(test_data) - 2 #how many steps to test
    start_state = (test_data.condition[0], test_data.condition[1])
    test_result = test_data.copy()

    prev_state = start_state
    result = [test_data.condition[0], test_data.condition[1]]
    while n-1:
        curr_state = np.random.choice([0,1],␣
↪p=transition_prob[tuple_to_row[prev_state]]) #taking the probability from␣
↪the transition matrix
        result.append(state[curr_state])
        prev_state = (prev_state[1], state[curr_state])
        n -= 1
```

```
    curr_state = np.random.choice([0,1],␣
↪p=transition_prob[tuple_to_row[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_simplified(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_simplified(all_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   6575  2018-01-01   no_rain            no_rain
1   6576  2018-01-02   no_rain            no_rain
2   6577  2018-01-03   no_rain               rain
3   6578  2018-01-04   no_rain            no_rain
4   6579  2018-01-05   no_rain            no_rain
0.7741273100616016
```

[ ]:
```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

[ ]:
```
0.7696988364134153
```

fall_six_conditions

December 7, 2022

1 Fall Season - 6 different weather conditions(long time frame)

1.1 Import libraries and dataset

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

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

[ ]:
```
fall.head()
```

[ ]:

	datetime	conditions
0	2000-10-01	Partially cloudy
1	2000-10-02	Partially cloudy
2	2000-10-03	Partially cloudy
3	2000-10-04	Rain, Partially cloudy
4	2000-10-05	Partially cloudy

1.2 Classify and separate data

[ ]:
```
classifier = {'Overcast':'overcast', 'Partially cloudy':'partially_cloudy',␣
↪'Clear':'clear', 'Rain, Partially cloudy':'rain_partially_cloudy', 'Rain':
↪'rain', 'Rain, Overcast':'rain_overcast'}

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

[ ]:
```
fall.head()
```

[ ]:

	datetime	conditions	condition
0	2000-10-01	Partially cloudy	partially_cloudy
1	2000-10-02	Partially cloudy	partially_cloudy
2	2000-10-03	Partially cloudy	partially_cloudy
3	2000-10-04	Rain, Partially cloudy	rain_partially_cloudy
4	2000-10-05	Partially cloudy	partially_cloudy

[ ]:
```
fall = fall[['datetime', 'condition']]
```

[ ]:
```
fall.head()
```

[ ]:
```
     datetime              condition
0  2000-10-01       partially_cloudy
1  2000-10-02       partially_cloudy
2  2000-10-03       partially_cloudy
3  2000-10-04  rain_partially_cloudy
4  2000-10-05       partially_cloudy
```

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
fall_train = fall.loc[fall['datetime'].between(train_start_date,␣
 ↪train_end_date)]
fall_train = fall_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
fall_test = fall.loc[fall['datetime'].between(test_start_date, test_end_date)]
fall_test = fall_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

[ ]:
```
# 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

C_after_C_count = 0.0
PC_after_C_count = 0.0
OV_after_C_count = 0.0
R_after_C_count = 0.0
RPC_after_C_count = 0.0
ROV_after_C_count = 0.0

C_after_PC_count = 0.0
PC_after_PC_count = 0.0
OV_after_PC_count = 0.0
R_after_PC_count = 0.0
RPC_after_PC_count = 0.0
ROV_after_PC_count = 0.0
```

```
C_after_OV_count = 0.0
PC_after_OV_count = 0.0
OV_after_OV_count = 0.0
R_after_OV_count = 0.0
RPC_after_OV_count = 0.0
ROV_after_OV_count = 0.0

C_after_R_count = 0.0
PC_after_R_count = 0.0
OV_after_R_count = 0.0
R_after_R_count = 0.0
RPC_after_R_count = 0.0
ROV_after_R_count = 0.0

C_after_RPC_count = 0.0
PC_after_RPC_count = 0.0
OV_after_RPC_count = 0.0
R_after_RPC_count = 0.0
RPC_after_RPC_count = 0.0
ROV_after_RPC_count = 0.0

C_after_ROV_count = 0.0
PC_after_ROV_count = 0.0
OV_after_ROV_count = 0.0
R_after_ROV_count = 0.0
RPC_after_ROV_count = 0.0
ROV_after_ROV_count = 0.0
```

[ ]:
```
fall_train
```

[ ]:
```
      index    datetime         condition
0       184  2002-10-01  partially_cloudy
1       185  2002-10-02             clear
2       186  2002-10-03             clear
3       187  2002-10-04             clear
4       188  2002-10-05             clear
…     …          …                …
1467   1651  2017-12-27             clear
1468   1652  2017-12-28             clear
1469   1653  2017-12-29             clear
1470   1654  2017-12-30  partially_cloudy
1471   1655  2017-12-31  partially_cloudy

[1472 rows x 3 columns]
```

[ ]:
```
# Count conditions
```

```
fall_train['condition_shift'] = fall_train['condition'].shift(-1)

for i in range(len(fall_train)):
    # Current 'clear'
    if fall_train.loc[i, 'condition'] == 'clear' and fall_train.loc[i,␣
↪'condition_shift'] == 'clear':
        C_after_C_count += 1
    elif fall_train.loc[i, 'condition'] == 'partially_cloudy' and fall_train.
↪loc[i, 'condition_shift'] == 'clear':
        PC_after_C_count += 1
    elif fall_train.loc[i, 'condition'] == 'overcast' and fall_train.loc[i,␣
↪'condition_shift'] == 'clear':
        OV_after_C_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain' and fall_train.loc[i,␣
↪'condition_shift'] == 'clear':
        R_after_C_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪fall_train.loc[i, 'condition_shift'] == 'clear':
        RPC_after_C_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_overcast' and fall_train.
↪loc[i, 'condition_shift'] == 'clear':
        ROV_after_C_count += 1
    # Current 'partially_cloudy'
    elif fall_train.loc[i, 'condition'] == 'clear' and fall_train.loc[i,␣
↪'condition_shift'] == 'partially_cloudy':
        C_after_PC_count += 1
    elif fall_train.loc[i, 'condition'] == 'partially_cloudy' and fall_train.
↪loc[i, 'condition_shift'] == 'partially_cloudy':
        PC_after_PC_count += 1
    elif fall_train.loc[i, 'condition'] == 'overcast' and fall_train.loc[i,␣
↪'condition_shift'] == 'partially_cloudy':
        OV_after_PC_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain' and fall_train.loc[i,␣
↪'condition_shift'] == 'partially_cloudy':
        R_after_PC_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪fall_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        RPC_after_PC_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_overcast' and fall_train.
↪loc[i, 'condition_shift'] == 'partially_cloudy':
        ROV_after_PC_count += 1
    # Current 'overcast'
    elif fall_train.loc[i, 'condition'] == 'clear' and fall_train.loc[i,␣
↪'condition_shift'] == 'overcast':
        C_after_OV_count += 1
```

```
    elif fall_train.loc[i, 'condition'] == 'partially_cloudy' and fall_train.
↪loc[i, 'condition_shift'] == 'overcast':
        PC_after_OV_count += 1
    elif fall_train.loc[i, 'condition'] == 'overcast' and fall_train.loc[i,␣
↪'condition_shift'] == 'overcast':
        OV_after_OV_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain' and fall_train.loc[i,␣
↪'condition_shift'] == 'overcast':
        R_after_OV_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪fall_train.loc[i, 'condition_shift'] == 'overcast':
        RPC_after_OV_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_overcast' and fall_train.
↪loc[i, 'condition_shift'] == 'overcast':
        ROV_after_OV_count += 1
    # Current 'rain'
    elif fall_train.loc[i, 'condition'] == 'clear' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain':
        C_after_R_count += 1
    elif fall_train.loc[i, 'condition'] == 'partially_cloudy' and fall_train.
↪loc[i, 'condition_shift'] == 'rain':
        PC_after_R_count += 1
    elif fall_train.loc[i, 'condition'] == 'overcast' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain':
        OV_after_R_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪fall_train.loc[i, 'condition_shift'] == 'rain':
        RPC_after_R_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_overcast' and fall_train.
↪loc[i, 'condition_shift'] == 'rain':
        ROV_after_R_count += 1
    # Current 'rain_partially_cloudy'
    elif fall_train.loc[i, 'condition'] == 'clear' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain_partially_cloudy':
        C_after_RPC_count += 1
    elif fall_train.loc[i, 'condition'] == 'partially_cloudy' and fall_train.
↪loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        PC_after_RPC_count += 1
    elif fall_train.loc[i, 'condition'] == 'overcast' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain_partially_cloudy':
        OV_after_RPC_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain_partially_cloudy':
```

```
        R_after_RPC_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪fall_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        RPC_after_RPC_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_overcast' and fall_train.
↪loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        ROV_after_RPC_count += 1
    # Current 'rain_overcast'
    elif fall_train.loc[i, 'condition'] == 'clear' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain_overcast':
        C_after_ROV_count += 1
    elif fall_train.loc[i, 'condition'] == 'partially_cloudy' and fall_train.
↪loc[i, 'condition_shift'] == 'rain_overcast':
        PC_after_ROV_count += 1
    elif fall_train.loc[i, 'condition'] == 'overcast' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain_overcast':
        OV_after_ROV_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain' and fall_train.loc[i,␣
↪'condition_shift'] == 'rain_overcast':
        R_after_ROV_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪fall_train.loc[i, 'condition_shift'] == 'rain_overcast':
        RPC_after_ROV_count += 1
    elif fall_train.loc[i, 'condition'] == 'rain_overcast' and fall_train.
↪loc[i, 'condition_shift'] == 'rain_overcast':
        ROV_after_ROV_count += 1
```

[ ]:
```
current_C_total = C_after_C_count + PC_after_C_count + OV_after_C_count +␣
↪R_after_C_count + RPC_after_C_count + ROV_after_C_count
current_PC_total = C_after_PC_count + PC_after_PC_count + OV_after_PC_count +␣
↪R_after_PC_count + RPC_after_PC_count + ROV_after_PC_count
current_OV_total = C_after_OV_count + PC_after_OV_count + OV_after_OV_count +␣
↪R_after_OV_count + RPC_after_OV_count + ROV_after_OV_count
current_R_total =C_after_R_count + PC_after_R_count + OV_after_R_count +␣
↪R_after_R_count + RPC_after_R_count + ROV_after_R_count
current_RPC_total = C_after_RPC_count + PC_after_RPC_count + OV_after_RPC_count␣
↪+ R_after_RPC_count + RPC_after_RPC_count + ROV_after_RPC_count
current_ROV_total = C_after_ROV_count + PC_after_ROV_count + OV_after_ROV_count␣
↪+ R_after_ROV_count + RPC_after_ROV_count + ROV_after_ROV_count
```

[ ]:
```
C_after_C_prob = C_after_C_count / current_C_total
PC_after_C_prob = PC_after_C_count / current_C_total
OV_after_C_prob = OV_after_C_count / current_C_total
R_after_C_prob = R_after_C_count / current_C_total
RPC_after_C_prob = RPC_after_C_count / current_C_total
ROV_after_C_prob = ROV_after_C_count / current_C_total
```

```
C_after_PC_prob = C_after_PC_count / current_PC_total
PC_after_PC_prob = PC_after_PC_count / current_PC_total
OV_after_PC_prob = OV_after_PC_count / current_PC_total
R_after_PC_prob = R_after_PC_count / current_PC_total
RPC_after_PC_prob = RPC_after_PC_count / current_PC_total
ROV_after_PC_prob = ROV_after_PC_count / current_PC_total

C_after_OV_prob = C_after_OV_count / current_OV_total
PC_after_OV_prob = PC_after_OV_count / current_OV_total
OV_after_OV_prob = OV_after_OV_count / current_OV_total
R_after_OV_prob = R_after_OV_count / current_OV_total
RPC_after_OV_prob = RPC_after_OV_count / current_OV_total
ROV_after_OV_prob = ROV_after_OV_count / current_OV_total

C_after_R_prob = C_after_R_count / current_R_total
PC_after_R_prob = PC_after_R_count / current_R_total
OV_after_R_prob = OV_after_R_count / current_R_total
R_after_R_prob = R_after_R_count / current_R_total
RPC_after_R_prob = RPC_after_R_count / current_R_total
ROV_after_R_prob = ROV_after_R_count / current_R_total

C_after_RPC_prob = C_after_RPC_count / current_RPC_total
PC_after_RPC_prob = PC_after_RPC_count / current_RPC_total
OV_after_RPC_prob = OV_after_RPC_count / current_RPC_total
R_after_RPC_prob = R_after_RPC_count / current_RPC_total
RPC_after_RPC_prob = RPC_after_RPC_count / current_RPC_total
ROV_after_RPC_prob = ROV_after_RPC_count / current_RPC_total

C_after_ROV_prob = C_after_ROV_count / current_ROV_total
PC_after_ROV_prob = PC_after_ROV_count / current_ROV_total
OV_after_ROV_prob = OV_after_ROV_count / current_ROV_total
R_after_ROV_prob = R_after_ROV_count / current_ROV_total
RPC_after_ROV_prob = RPC_after_ROV_count / current_ROV_total
ROV_after_ROV_prob = ROV_after_ROV_count / current_ROV_total
```

[ ]:
```
# Printing our probabilities for 6x6 transition matrix:
print(C_after_C_prob)
print(PC_after_C_prob)
print(OV_after_C_prob)
print(R_after_C_prob)
print(RPC_after_C_prob)
print(ROV_after_C_prob)

print(C_after_PC_prob)
print(PC_after_PC_prob)
print(OV_after_PC_prob)
```

```
print(R_after_PC_prob)
print(RPC_after_PC_prob)
print(ROV_after_PC_prob)

print(C_after_OV_prob)
print(PC_after_OV_prob)
print(OV_after_OV_prob)
print(R_after_OV_prob)
print(RPC_after_OV_prob)
print(ROV_after_OV_prob)

print(C_after_R_prob)
print(PC_after_R_prob)
print(OV_after_R_prob)
print(R_after_R_prob)
print(RPC_after_R_prob)
print(ROV_after_R_prob)

print(C_after_RPC_prob)
print(PC_after_RPC_prob)
print(OV_after_RPC_prob)
print(R_after_RPC_prob)
print(RPC_after_RPC_prob)
print(ROV_after_RPC_prob)

print(C_after_ROV_prob)
print(PC_after_ROV_prob)
print(OV_after_ROV_prob)
print(R_after_ROV_prob)
print(RPC_after_ROV_prob)
print(ROV_after_ROV_prob)
```

```
0.7132963988919667
0.13850415512465375
0.0
0.04847645429362881
0.09002770083102493
0.009695290858725761
0.3462532299741602
0.4883720930232558
0.015503875968992248
0.025839793281653745
0.1111111111111111
0.012919896640826873
0.0
0.5882352941176471
0.17647058823529413
```

```
0.0
0.17647058823529413
0.058823529411764705
0.2876712328767123
0.2054794520547945
0.0
0.2328767123287671
0.2465753424657534
0.0273972602739726
0.21296296296296297
0.26851851851851855
0.027777777777777776
0.046296296296296294
0.3472222222222222
0.09722222222222222
0.10714285714285714
0.26785714285714285
0.03571428571428571
0.017857142857142856
0.21428571428571427
0.35714285714285715
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(C_after_C_prob + PC_after_C_prob + OV_after_C_prob + R_after_C_prob +␣
 ↪RPC_after_C_prob + ROV_after_C_prob)
print(C_after_PC_prob + PC_after_PC_prob + OV_after_PC_prob + R_after_PC_prob +␣
 ↪RPC_after_PC_prob + ROV_after_PC_prob)
print(C_after_OV_prob + PC_after_OV_prob + OV_after_OV_prob + R_after_OV_prob +␣
 ↪RPC_after_OV_prob + ROV_after_OV_prob)
print(C_after_R_prob + PC_after_R_prob + OV_after_R_prob + R_after_R_prob +␣
 ↪RPC_after_R_prob + ROV_after_R_prob)
print(C_after_RPC_prob + PC_after_RPC_prob + OV_after_RPC_prob +␣
 ↪R_after_RPC_prob + RPC_after_RPC_prob + ROV_after_RPC_prob)
print(C_after_ROV_prob + PC_after_ROV_prob + OV_after_ROV_prob +␣
 ↪R_after_ROV_prob + RPC_after_ROV_prob + ROV_after_ROV_prob)
```

```
1.0
1.0
1.0
1.0
1.0
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_matrix = [[C_after_C_prob, PC_after_C_prob, OV_after_C_prob,␣
 ↪R_after_C_prob, RPC_after_C_prob, ROV_after_C_prob],
```

```
                    [C_after_PC_prob, PC_after_PC_prob, OV_after_PC_prob,␣
 ↪R_after_PC_prob, RPC_after_PC_prob, ROV_after_PC_prob],
                    [C_after_OV_prob, PC_after_OV_prob, OV_after_OV_prob,␣
 ↪R_after_OV_prob, RPC_after_OV_prob, ROV_after_OV_prob],
                    [C_after_R_prob, PC_after_R_prob, OV_after_R_prob,␣
 ↪R_after_R_prob, RPC_after_R_prob, ROV_after_R_prob],
                    [C_after_RPC_prob, PC_after_RPC_prob, OV_after_RPC_prob,␣
 ↪R_after_RPC_prob, RPC_after_RPC_prob, ROV_after_RPC_prob],
                    [C_after_ROV_prob, PC_after_ROV_prob, OV_after_ROV_prob,␣
 ↪R_after_ROV_prob, RPC_after_ROV_prob, ROV_after_ROV_prob]]
print(transition_matrix)
```

```
[[0.7132963988919667, 0.13850415512465375, 0.0, 0.04847645429362881,
0.09002770083102493, 0.009695290858725761], [0.3462532299741602,
0.4883720930232558, 0.015503875968992248, 0.025839793281653745,
0.1111111111111111, 0.012919896640826873], [0.0, 0.5882352941176471,
0.17647058823529413, 0.0, 0.17647058823529413, 0.05882352941176471],
[0.2876712328767123, 0.2054794520547945, 0.0, 0.2328767123287671,
0.2465753424657534, 0.0273972602739726], [0.21296296296296297,
0.26851851851851855, 0.027777777777777776, 0.046296296296296294,
0.3472222222222222, 0.09722222222222222], [0.10714285714285714,
0.26785714285714285, 0.03571428571428571, 0.017857142857142856,
0.21428571428571427, 0.35714285714285715]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.7132964  0.13850416 0.         0.04847645 0.0900277  0.00969529]
 [0.34625323 0.48837209 0.01550388 0.02583979 0.11111111 0.0129199 ]
 [0.         0.58823529 0.17647059 0.         0.17647059 0.05882353]
 [0.28767123 0.20547945 0.         0.23287671 0.24657534 0.02739726]
 [0.21296296 0.26851852 0.02777778 0.0462963  0.34722222 0.09722222]
 [0.10714286 0.26785714 0.03571429 0.01785714 0.21428571 0.35714286]]
```

[ ]:
```
fall_test.head(1)
```

[ ]:

	index	datetime	condition
0	1656	2018-10-01	clear

First day of fall 2018: clear

[ ]:
```
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 = [state[start_state]]
    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(fall_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime              condition predicted_condition
0   1656  2018-10-01                  clear               clear
1   1657  2018-10-02        partially_cloudy               clear
2   1658  2018-10-03  rain_partially_cloudy               clear
3   1659  2018-10-04  rain_partially_cloudy    partially_cloudy
4   1660  2018-10-05        partially_cloudy               clear
0.38858695652173914
```

[ ]:
```
run_predictions_return_avg_accuracy(fall_test, 100)
```

[ ]:
```
0.3698097826086957
```

winter_six_conditions

December 7, 2022

1 Winter Season - 6 different weather conditions(long time frame)

1.1 Import libraries and dataset

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

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

[ ]:
```
winter_seasons.head(200)
```

[ ]:

	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
..	…	…
195	2002-01-15	Partially cloudy
196	2002-01-16	Partially cloudy
197	2002-01-17	Clear
198	2002-01-18	Partially cloudy
199	2002-01-19	Partially cloudy

[200 rows x 2 columns]

1.2 Classify and separate data

[ ]:
```
classifier = {'Overcast':'overcast', 'Partially cloudy':'partially_cloudy',␣
↪'Clear':'clear', 'Rain, Partially cloudy':'rain_partially_cloudy', 'Rain':
↪'rain', 'Rain, Overcast':'rain_overcast'}

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

[ ]:
```
winter_seasons.head()
```

[ ]:
```
     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
```

[ ]:
```
winter_seasons = winter_seasons[['datetime', 'condition']]
```

[ ]:
```
winter_seasons.head()
```

[ ]:
```
     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
```

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
winter_seasons_train = winter_seasons.loc[winter_seasons['datetime'].
 ↪between(train_start_date, train_end_date)]
winter_seasons_train = winter_seasons_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
winter_seasons_test = winter_seasons.loc[winter_seasons['datetime'].
 ↪between(test_start_date, test_end_date)]
winter_seasons_test = winter_seasons_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

[ ]:
```
# 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

C_after_C_count = 0.0
PC_after_C_count = 0.0
OV_after_C_count = 0.0
R_after_C_count = 0.0
```

```
RPC_after_C_count = 0.0
ROV_after_C_count = 0.0

C_after_PC_count = 0.0
PC_after_PC_count = 0.0
OV_after_PC_count = 0.0
R_after_PC_count = 0.0
RPC_after_PC_count = 0.0
ROV_after_PC_count = 0.0

C_after_OV_count = 0.0
PC_after_OV_count = 0.0
OV_after_OV_count = 0.0
R_after_OV_count = 0.0
RPC_after_OV_count = 0.0
ROV_after_OV_count = 0.0

C_after_R_count = 0.0
PC_after_R_count = 0.0
OV_after_R_count = 0.0
R_after_R_count = 0.0
RPC_after_R_count = 0.0
ROV_after_R_count = 0.0

C_after_RPC_count = 0.0
PC_after_RPC_count = 0.0
OV_after_RPC_count = 0.0
R_after_RPC_count = 0.0
RPC_after_RPC_count = 0.0
ROV_after_RPC_count = 0.0

C_after_ROV_count = 0.0
PC_after_ROV_count = 0.0
OV_after_ROV_count = 0.0
R_after_ROV_count = 0.0
RPC_after_ROV_count = 0.0
ROV_after_ROV_count = 0.0
```

[ ]:
```
winter_seasons_train
```

[ ]:
```
      index    datetime              condition
0       181  2002-01-01       partially_cloudy
1       182  2002-01-02  rain_partially_cloudy
2       183  2002-01-03  rain_partially_cloudy
3       184  2002-01-04       partially_cloudy
4       185  2002-01-05       partially_cloudy
…     …          …                     …
```

```
1439   1620  2017-03-27                clear
1440   1621  2017-03-28                clear
1441   1622  2017-03-29                clear
1442   1623  2017-03-30                clear
1443   1624  2017-03-31                clear

[1444 rows x 3 columns]
```

[ ]:
```
winter_seasons_train['condition_shift'] = winter_seasons_train['condition'].
↪shift(-1)
winter_seasons_train.head(30)
```

[ ]:

	index	datetime	condition	condition_shift
0	181	2002-01-01	partially_cloudy	rain_partially_cloudy
1	182	2002-01-02	rain_partially_cloudy	rain_partially_cloudy
2	183	2002-01-03	rain_partially_cloudy	partially_cloudy
3	184	2002-01-04	partially_cloudy	partially_cloudy
4	185	2002-01-05	partially_cloudy	partially_cloudy
5	186	2002-01-06	partially_cloudy	partially_cloudy
6	187	2002-01-07	partially_cloudy	partially_cloudy
7	188	2002-01-08	partially_cloudy	partially_cloudy
8	189	2002-01-09	partially_cloudy	clear
9	190	2002-01-10	clear	clear
10	191	2002-01-11	clear	clear
11	192	2002-01-12	clear	clear
12	193	2002-01-13	clear	partially_cloudy
13	194	2002-01-14	partially_cloudy	partially_cloudy
14	195	2002-01-15	partially_cloudy	partially_cloudy
15	196	2002-01-16	partially_cloudy	clear
16	197	2002-01-17	clear	partially_cloudy
17	198	2002-01-18	partially_cloudy	partially_cloudy
18	199	2002-01-19	partially_cloudy	clear
19	200	2002-01-20	clear	clear
20	201	2002-01-21	clear	partially_cloudy
21	202	2002-01-22	partially_cloudy	clear
22	203	2002-01-23	clear	clear
23	204	2002-01-24	clear	partially_cloudy
24	205	2002-01-25	partially_cloudy	partially_cloudy
25	206	2002-01-26	partially_cloudy	rain_partially_cloudy
26	207	2002-01-27	rain_partially_cloudy	rain_partially_cloudy
27	208	2002-01-28	rain_partially_cloudy	rain_partially_cloudy
28	209	2002-01-29	rain_partially_cloudy	clear
29	210	2002-01-30	clear	clear

[ ]:
```
# Count conditions
```

```
winter_seasons_train['condition_shift'] = winter_seasons_train['condition'].
↪shift(-1)

for i in range(len(winter_seasons_train)):
    # Current 'clear'
    if winter_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'clear':
        C_after_C_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'clear':
        PC_after_C_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'clear':
        OV_after_C_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'clear':
        R_after_C_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and winter_seasons_train.loc[i, 'condition_shift'] == 'clear':
        RPC_after_C_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'clear':
        ROV_after_C_count += 1
    # Current 'partially_cloudy'
    elif winter_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        C_after_PC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        PC_after_PC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        OV_after_PC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        R_after_PC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and winter_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        RPC_after_PC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        ROV_after_PC_count += 1
    # Current 'overcast'
    elif winter_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        C_after_OV_count += 1
```

```
    elif winter_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        PC_after_OV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        OV_after_OV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        R_after_OV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and winter_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        RPC_after_OV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        ROV_after_OV_count += 1
    # Current 'rain'
    elif winter_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain':
        C_after_R_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain':
        PC_after_R_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain':
        OV_after_R_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and winter_seasons_train.loc[i, 'condition_shift'] == 'rain':
        RPC_after_R_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain':
        ROV_after_R_count += 1
    # Current 'rain_partially_cloudy'
    elif winter_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        C_after_RPC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        PC_after_RPC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        OV_after_RPC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
```

```
        R_after_RPC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and winter_seasons_train.loc[i, 'condition_shift'] ==␣
↪'rain_partially_cloudy':
        RPC_after_RPC_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        ROV_after_RPC_count += 1
    # Current 'rain_overcast'
    elif winter_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        C_after_ROV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        PC_after_ROV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        OV_after_ROV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        R_after_ROV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and winter_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        RPC_after_ROV_count += 1
    elif winter_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪winter_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        ROV_after_ROV_count += 1
```

[ ]:
```
current_C_total = C_after_C_count + PC_after_C_count + OV_after_C_count +␣
↪R_after_C_count + RPC_after_C_count + ROV_after_C_count
current_PC_total = C_after_PC_count + PC_after_PC_count + OV_after_PC_count +␣
↪R_after_PC_count + RPC_after_PC_count + ROV_after_PC_count
current_OV_total = C_after_OV_count + PC_after_OV_count + OV_after_OV_count +␣
↪R_after_OV_count + RPC_after_OV_count + ROV_after_OV_count
current_R_total =C_after_R_count + PC_after_R_count + OV_after_R_count +␣
↪R_after_R_count + RPC_after_R_count + ROV_after_R_count
current_RPC_total = C_after_RPC_count + PC_after_RPC_count + OV_after_RPC_count␣
↪+ R_after_RPC_count + RPC_after_RPC_count + ROV_after_RPC_count
current_ROV_total = C_after_ROV_count + PC_after_ROV_count + OV_after_ROV_count␣
↪+ R_after_ROV_count + RPC_after_ROV_count + ROV_after_ROV_count
```

[ ]:
```
C_after_C_prob = C_after_C_count / current_C_total
PC_after_C_prob = PC_after_C_count / current_C_total
OV_after_C_prob = OV_after_C_count / current_C_total
R_after_C_prob = R_after_C_count / current_C_total
RPC_after_C_prob = RPC_after_C_count / current_C_total
```

```
ROV_after_C_prob = ROV_after_C_count / current_C_total

C_after_PC_prob = C_after_PC_count / current_PC_total
PC_after_PC_prob = PC_after_PC_count / current_PC_total
OV_after_PC_prob = OV_after_PC_count / current_PC_total
R_after_PC_prob = R_after_PC_count / current_PC_total
RPC_after_PC_prob = RPC_after_PC_count / current_PC_total
ROV_after_PC_prob = ROV_after_PC_count / current_PC_total

C_after_OV_prob = C_after_OV_count / current_OV_total
PC_after_OV_prob = PC_after_OV_count / current_OV_total
OV_after_OV_prob = OV_after_OV_count / current_OV_total
R_after_OV_prob = R_after_OV_count / current_OV_total
RPC_after_OV_prob = RPC_after_OV_count / current_OV_total
ROV_after_OV_prob = ROV_after_OV_count / current_OV_total

C_after_R_prob = C_after_R_count / current_R_total
PC_after_R_prob = PC_after_R_count / current_R_total
OV_after_R_prob = OV_after_R_count / current_R_total
R_after_R_prob = R_after_R_count / current_R_total
RPC_after_R_prob = RPC_after_R_count / current_R_total
ROV_after_R_prob = ROV_after_R_count / current_R_total

C_after_RPC_prob = C_after_RPC_count / current_RPC_total
PC_after_RPC_prob = PC_after_RPC_count / current_RPC_total
OV_after_RPC_prob = OV_after_RPC_count / current_RPC_total
R_after_RPC_prob = R_after_RPC_count / current_RPC_total
RPC_after_RPC_prob = RPC_after_RPC_count / current_RPC_total
ROV_after_RPC_prob = ROV_after_RPC_count / current_RPC_total

C_after_ROV_prob = C_after_ROV_count / current_ROV_total
PC_after_ROV_prob = PC_after_ROV_count / current_ROV_total
OV_after_ROV_prob = OV_after_ROV_count / current_ROV_total
R_after_ROV_prob = R_after_ROV_count / current_ROV_total
RPC_after_ROV_prob = RPC_after_ROV_count / current_ROV_total
ROV_after_ROV_prob = ROV_after_ROV_count / current_ROV_total
```

[ ]:
```
# Printing our probabilities for 6x6 transition matrix:
print(C_after_C_prob)
print(PC_after_C_prob)
print(OV_after_C_prob)
print(R_after_C_prob)
print(RPC_after_C_prob)
print(ROV_after_C_prob)

print(C_after_PC_prob)
print(PC_after_PC_prob)
```

```
print(OV_after_PC_prob)
print(R_after_PC_prob)
print(RPC_after_PC_prob)
print(ROV_after_PC_prob)

print(C_after_OV_prob)
print(PC_after_OV_prob)
print(OV_after_OV_prob)
print(R_after_OV_prob)
print(RPC_after_OV_prob)
print(ROV_after_OV_prob)

print(C_after_R_prob)
print(PC_after_R_prob)
print(OV_after_R_prob)
print(R_after_R_prob)
print(RPC_after_R_prob)
print(ROV_after_R_prob)

print(C_after_RPC_prob)
print(PC_after_RPC_prob)
print(OV_after_RPC_prob)
print(R_after_RPC_prob)
print(RPC_after_RPC_prob)
print(ROV_after_RPC_prob)

print(C_after_ROV_prob)
print(PC_after_ROV_prob)
print(OV_after_ROV_prob)
print(R_after_ROV_prob)
print(RPC_after_ROV_prob)
print(ROV_after_ROV_prob)
```

```
0.6776611694152923
0.15892053973013492
0.0
0.043478260869565216
0.11694152923538231
0.0029985007496251873
0.34382566585956414
0.4915254237288136
0.01937046004842615
0.009685230024213076
0.1234866828087167
0.012106537530266344
0.07692307692307693
0.8461538461538461
```

```
0.07692307692307693
0.0
0.0
0.0
0.375
0.15
0.0
0.05
0.375
0.05
0.21011673151750973
0.2607003891050584
0.011673151750972763
0.019455252918287938
0.377431906614786
0.12062256809338522
0.03773584905660377
0.39622641509433965
0.018867924528301886
0.0
0.3018867924528302
0.24528301886792453
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(C_after_C_prob + PC_after_C_prob + OV_after_C_prob + R_after_C_prob +␣
 ↪RPC_after_C_prob + ROV_after_C_prob)
print(C_after_PC_prob + PC_after_PC_prob + OV_after_PC_prob + R_after_PC_prob +␣
 ↪RPC_after_PC_prob + ROV_after_PC_prob)
print(C_after_OV_prob + PC_after_OV_prob + OV_after_OV_prob + R_after_OV_prob +␣
 ↪RPC_after_OV_prob + ROV_after_OV_prob)
print(C_after_R_prob + PC_after_R_prob + OV_after_R_prob + R_after_R_prob +␣
 ↪RPC_after_R_prob + ROV_after_R_prob)
print(C_after_RPC_prob + PC_after_RPC_prob + OV_after_RPC_prob +␣
 ↪R_after_RPC_prob + RPC_after_RPC_prob + ROV_after_RPC_prob)
print(C_after_ROV_prob + PC_after_ROV_prob + OV_after_ROV_prob +␣
 ↪R_after_ROV_prob + RPC_after_ROV_prob + ROV_after_ROV_prob)
```

```
1.0
1.0
1.0
1.0
1.0
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_matrix = [[C_after_C_prob, PC_after_C_prob, OV_after_C_prob,␣
 ↪R_after_C_prob, RPC_after_C_prob, ROV_after_C_prob],
```

```
                    [C_after_PC_prob, PC_after_PC_prob, OV_after_PC_prob,␣
 ↪R_after_PC_prob, RPC_after_PC_prob, ROV_after_PC_prob],
                    [C_after_OV_prob, PC_after_OV_prob, OV_after_OV_prob,␣
 ↪R_after_OV_prob, RPC_after_OV_prob, ROV_after_OV_prob],
                    [C_after_R_prob, PC_after_R_prob, OV_after_R_prob,␣
 ↪R_after_R_prob, RPC_after_R_prob, ROV_after_R_prob],
                    [C_after_RPC_prob, PC_after_RPC_prob, OV_after_RPC_prob,␣
 ↪R_after_RPC_prob, RPC_after_RPC_prob, ROV_after_RPC_prob],
                    [C_after_ROV_prob, PC_after_ROV_prob, OV_after_ROV_prob,␣
 ↪R_after_ROV_prob, RPC_after_ROV_prob, ROV_after_ROV_prob]]
print(transition_matrix)
```

```
[[0.6776611694152923, 0.15892053973013492, 0.0, 0.043478260869565216,
0.11694152923538231, 0.0029985007496251873], [0.34382566585956414,
0.4915254237288136, 0.01937046004842615, 0.009685230024213076,
0.1234866828087167, 0.012106537530266344], [0.07692307692307693,
0.8461538461538461, 0.07692307692307693, 0.0, 0.0, 0.0], [0.375, 0.15, 0.0,
0.05, 0.375, 0.05], [0.21011673151750973, 0.2607003891050584,
0.011673151750972763, 0.019455252918287938, 0.377431906614786,
0.12062256809338522], [0.03773584905660377, 0.39622641509433965,
0.018867924528301886, 0.0, 0.3018867924528302, 0.24528301886792453]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.67766117 0.15892054 0.         0.04347826 0.11694153 0.0029985 ]
 [0.34382567 0.49152542 0.01937046 0.00968523 0.12348668 0.01210654]
 [0.07692308 0.84615385 0.07692308 0.         0.         0.        ]
 [0.375      0.15       0.         0.05       0.375      0.05      ]
 [0.21011673 0.26070039 0.01167315 0.01945525 0.37743191 0.12062257]
 [0.03773585 0.39622642 0.01886792 0.         0.30188679 0.24528302]]
```

[ ]:
```
winter_seasons_test.head(1)
```

[ ]:
```
   index    datetime condition
0   1625  2018-01-01     clear
```

First Day of spring 2018: partially_cloudy

[ ]:
```
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 = []
    result.append(state[start_state])
    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(winter_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime         condition predicted_condition
```

```
   0   1625  2018-01-01             clear               clear
   1   1626  2018-01-02             clear               clear
   2   1627  2018-01-03             clear               clear
   3   1628  2018-01-04  partially_cloudy   partially_cloudy
   4   1629  2018-01-05  partially_cloudy   partially_cloudy
   0.3878116343490305
```

[ ]:
```
run_predictions_return_avg_accuracy(winter_seasons_test, 100)
```

[ ]:
```
0.35728531855955675
```

spring_six_conditions

December 7, 2022

1 Spring Season - 6 different weather conditions(long time frame)

1.1 Import libraries and dataset

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

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

[ ]:
```
spring_seasons.head(200)
```

[ ]:
```
       datetime              conditions
0    2000-04-01                   Clear
1    2000-04-02                   Clear
2    2000-04-03                   Clear
3    2000-04-04       Partially cloudy
4    2000-04-05       Partially cloudy
..          …                       …
195  2002-04-14       Partially cloudy
196  2002-04-15  Rain, Partially cloudy
197  2002-04-16       Partially cloudy
198  2002-04-17       Partially cloudy
199  2002-04-18                   Clear

[200 rows x 2 columns]
```

1.2 Classify and separate data

[ ]:
```
classifier = {'Overcast':'overcast', 'Partially cloudy':'partially_cloudy',␣
↪'Clear':'clear', 'Rain, Partially cloudy':'rain_partially_cloudy', 'Rain':
↪'rain', 'Rain, Overcast':'rain_overcast'}

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

[ ]:
```
spring_seasons.head()
```

[ ]:

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

[ ]:
```
spring_seasons = spring_seasons[['datetime', 'condition']]
```

[ ]:
```
spring_seasons.head()
```

[ ]:

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

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
spring_seasons_train = spring_seasons.loc[spring_seasons['datetime'].
 ↪between(train_start_date, train_end_date)]
spring_seasons_train = spring_seasons_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
spring_seasons_test = spring_seasons.loc[spring_seasons['datetime'].
 ↪between(test_start_date, test_end_date)]
spring_seasons_test = spring_seasons_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

[ ]:
```
# 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

C_after_C_count = 0.0
PC_after_C_count = 0.0
OV_after_C_count = 0.0
R_after_C_count = 0.0
```

```
RPC_after_C_count = 0.0
ROV_after_C_count = 0.0

C_after_PC_count = 0.0
PC_after_PC_count = 0.0
OV_after_PC_count = 0.0
R_after_PC_count = 0.0
RPC_after_PC_count = 0.0
ROV_after_PC_count = 0.0

C_after_OV_count = 0.0
PC_after_OV_count = 0.0
OV_after_OV_count = 0.0
R_after_OV_count = 0.0
RPC_after_OV_count = 0.0
ROV_after_OV_count = 0.0

C_after_R_count = 0.0
PC_after_R_count = 0.0
OV_after_R_count = 0.0
R_after_R_count = 0.0
RPC_after_R_count = 0.0
ROV_after_R_count = 0.0

C_after_RPC_count = 0.0
PC_after_RPC_count = 0.0
OV_after_RPC_count = 0.0
R_after_RPC_count = 0.0
RPC_after_RPC_count = 0.0
ROV_after_RPC_count = 0.0

C_after_ROV_count = 0.0
PC_after_ROV_count = 0.0
OV_after_ROV_count = 0.0
R_after_ROV_count = 0.0
RPC_after_ROV_count = 0.0
ROV_after_ROV_count = 0.0
```

[ ]:
```
spring_seasons_train
```

[ ]:

	index	datetime	condition
0	182	2002-04-01	partially_cloudy
1	183	2002-04-02	partially_cloudy
2	184	2002-04-03	partially_cloudy
3	185	2002-04-04	partially_cloudy
4	186	2002-04-05	partially_cloudy
…	…	…	…

```
1451   1633  2017-06-26             clear
1452   1634  2017-06-27             clear
1453   1635  2017-06-28  partially_cloudy
1454   1636  2017-06-29  partially_cloudy
1455   1637  2017-06-30  partially_cloudy

[1456 rows x 3 columns]
```

[ ]:
```
spring_seasons_train['condition_shift'] = spring_seasons_train['condition'].
↪shift(-1)
spring_seasons_train.head(30)
```

[ ]:

	index	datetime	condition	condition_shift
0	182	2002-04-01	partially_cloudy	partially_cloudy
1	183	2002-04-02	partially_cloudy	partially_cloudy
2	184	2002-04-03	partially_cloudy	partially_cloudy
3	185	2002-04-04	partially_cloudy	partially_cloudy
4	186	2002-04-05	partially_cloudy	rain_partially_cloudy
5	187	2002-04-06	rain_partially_cloudy	partially_cloudy
6	188	2002-04-07	partially_cloudy	partially_cloudy
7	189	2002-04-08	partially_cloudy	partially_cloudy
8	190	2002-04-09	partially_cloudy	partially_cloudy
9	191	2002-04-10	partially_cloudy	partially_cloudy
10	192	2002-04-11	partially_cloudy	partially_cloudy
11	193	2002-04-12	partially_cloudy	partially_cloudy
12	194	2002-04-13	partially_cloudy	partially_cloudy
13	195	2002-04-14	partially_cloudy	rain_partially_cloudy
14	196	2002-04-15	rain_partially_cloudy	partially_cloudy
15	197	2002-04-16	partially_cloudy	partially_cloudy
16	198	2002-04-17	partially_cloudy	clear
17	199	2002-04-18	clear	clear
18	200	2002-04-19	clear	clear
19	201	2002-04-20	clear	clear
20	202	2002-04-21	clear	clear
21	203	2002-04-22	clear	partially_cloudy
22	204	2002-04-23	partially_cloudy	rain_partially_cloudy
23	205	2002-04-24	rain_partially_cloudy	partially_cloudy
24	206	2002-04-25	partially_cloudy	rain_partially_cloudy
25	207	2002-04-26	rain_partially_cloudy	partially_cloudy
26	208	2002-04-27	partially_cloudy	clear
27	209	2002-04-28	clear	clear
28	210	2002-04-29	clear	partially_cloudy
29	211	2002-04-30	partially_cloudy	clear

[ ]:
```
# Count conditions
```

```
spring_seasons_train['condition_shift'] = spring_seasons_train['condition'].
↪shift(-1)

for i in range(len(spring_seasons_train)):
    # Current 'clear'
    if spring_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'clear':
        C_after_C_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'clear':
        PC_after_C_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'clear':
        OV_after_C_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'clear':
        R_after_C_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and spring_seasons_train.loc[i, 'condition_shift'] == 'clear':
        RPC_after_C_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'clear':
        ROV_after_C_count += 1
    # Current 'partially_cloudy'
    elif spring_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        C_after_PC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        PC_after_PC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        OV_after_PC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        R_after_PC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and spring_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        RPC_after_PC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        ROV_after_PC_count += 1
    # Current 'overcast'
    elif spring_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        C_after_OV_count += 1
```

```
    elif spring_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        PC_after_OV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        OV_after_OV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        R_after_OV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and spring_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        RPC_after_OV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'overcast':
        ROV_after_OV_count += 1
    # Current 'rain'
    elif spring_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain':
        C_after_R_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain':
        PC_after_R_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain':
        OV_after_R_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and spring_seasons_train.loc[i, 'condition_shift'] == 'rain':
        RPC_after_R_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain':
        ROV_after_R_count += 1
    # Current 'rain_partially_cloudy'
    elif spring_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        C_after_RPC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        PC_after_RPC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        OV_after_RPC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
```

```
        R_after_RPC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and spring_seasons_train.loc[i, 'condition_shift'] ==␣
↪'rain_partially_cloudy':
        RPC_after_RPC_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        ROV_after_RPC_count += 1
    # Current 'rain_overcast'
    elif spring_seasons_train.loc[i, 'condition'] == 'clear' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        C_after_ROV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        PC_after_ROV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        OV_after_ROV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        R_after_ROV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_partially_cloudy'␣
↪and spring_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        RPC_after_ROV_count += 1
    elif spring_seasons_train.loc[i, 'condition'] == 'rain_overcast' and␣
↪spring_seasons_train.loc[i, 'condition_shift'] == 'rain_overcast':
        ROV_after_ROV_count += 1
```

[ ]:
```
current_C_total = C_after_C_count + PC_after_C_count + OV_after_C_count +␣
↪R_after_C_count + RPC_after_C_count + ROV_after_C_count
current_PC_total = C_after_PC_count + PC_after_PC_count + OV_after_PC_count +␣
↪R_after_PC_count + RPC_after_PC_count + ROV_after_PC_count
current_OV_total = C_after_OV_count + PC_after_OV_count + OV_after_OV_count +␣
↪R_after_OV_count + RPC_after_OV_count + ROV_after_OV_count
current_R_total =C_after_R_count + PC_after_R_count + OV_after_R_count +␣
↪R_after_R_count + RPC_after_R_count + ROV_after_R_count
current_RPC_total = C_after_RPC_count + PC_after_RPC_count + OV_after_RPC_count␣
↪+ R_after_RPC_count + RPC_after_RPC_count + ROV_after_RPC_count
current_ROV_total = C_after_ROV_count + PC_after_ROV_count + OV_after_ROV_count␣
↪+ R_after_ROV_count + RPC_after_ROV_count + ROV_after_ROV_count
```

[ ]:
```
C_after_C_prob = C_after_C_count / current_C_total
PC_after_C_prob = PC_after_C_count / current_C_total
OV_after_C_prob = OV_after_C_count / current_C_total
R_after_C_prob = R_after_C_count / current_C_total
RPC_after_C_prob = RPC_after_C_count / current_C_total
```

```
ROV_after_C_prob = ROV_after_C_count / current_C_total

C_after_PC_prob = C_after_PC_count / current_PC_total
PC_after_PC_prob = PC_after_PC_count / current_PC_total
OV_after_PC_prob = OV_after_PC_count / current_PC_total
R_after_PC_prob = R_after_PC_count / current_PC_total
RPC_after_PC_prob = RPC_after_PC_count / current_PC_total
ROV_after_PC_prob = ROV_after_PC_count / current_PC_total

C_after_OV_prob = C_after_OV_count / current_OV_total
PC_after_OV_prob = PC_after_OV_count / current_OV_total
OV_after_OV_prob = OV_after_OV_count / current_OV_total
R_after_OV_prob = R_after_OV_count / current_OV_total
RPC_after_OV_prob = RPC_after_OV_count / current_OV_total
ROV_after_OV_prob = ROV_after_OV_count / current_OV_total

C_after_R_prob = C_after_R_count / current_R_total
PC_after_R_prob = PC_after_R_count / current_R_total
OV_after_R_prob = OV_after_R_count / current_R_total
R_after_R_prob = R_after_R_count / current_R_total
RPC_after_R_prob = RPC_after_R_count / current_R_total
ROV_after_R_prob = ROV_after_R_count / current_R_total

C_after_RPC_prob = C_after_RPC_count / current_RPC_total
PC_after_RPC_prob = PC_after_RPC_count / current_RPC_total
OV_after_RPC_prob = OV_after_RPC_count / current_RPC_total
R_after_RPC_prob = R_after_RPC_count / current_RPC_total
RPC_after_RPC_prob = RPC_after_RPC_count / current_RPC_total
ROV_after_RPC_prob = ROV_after_RPC_count / current_RPC_total

C_after_ROV_prob = C_after_ROV_count / current_ROV_total
PC_after_ROV_prob = PC_after_ROV_count / current_ROV_total
OV_after_ROV_prob = OV_after_ROV_count / current_ROV_total
R_after_ROV_prob = R_after_ROV_count / current_ROV_total
RPC_after_ROV_prob = RPC_after_ROV_count / current_ROV_total
ROV_after_ROV_prob = ROV_after_ROV_count / current_ROV_total
```

[ ]:
```
# Printing our probabilities for 6x6 transition matrix:
print(C_after_C_prob)
print(PC_after_C_prob)
print(OV_after_C_prob)
print(R_after_C_prob)
print(RPC_after_C_prob)
print(ROV_after_C_prob)

print(C_after_PC_prob)
print(PC_after_PC_prob)
```

```
print(OV_after_PC_prob)
print(R_after_PC_prob)
print(RPC_after_PC_prob)
print(ROV_after_PC_prob)

print(C_after_OV_prob)
print(PC_after_OV_prob)
print(OV_after_OV_prob)
print(R_after_OV_prob)
print(RPC_after_OV_prob)
print(ROV_after_OV_prob)

print(C_after_R_prob)
print(PC_after_R_prob)
print(OV_after_R_prob)
print(R_after_R_prob)
print(RPC_after_R_prob)
print(ROV_after_R_prob)

print(C_after_RPC_prob)
print(PC_after_RPC_prob)
print(OV_after_RPC_prob)
print(R_after_RPC_prob)
print(RPC_after_RPC_prob)
print(ROV_after_RPC_prob)

print(C_after_ROV_prob)
print(PC_after_ROV_prob)
print(OV_after_ROV_prob)
print(R_after_ROV_prob)
print(RPC_after_ROV_prob)
print(ROV_after_ROV_prob)
```

```
0.6495901639344263
0.2725409836065574
0.004098360655737705
0.01639344262295082
0.05737704918032787
0.0
0.196405648267009
0.693196405648267
0.044929396662387676
0.0025673940949935813
0.05648267008985879
0.006418485237483954
0.017543859649122806
0.7368421052631579
```

```
0.19298245614035087
0.0
0.03508771929824561
0.017543859649122806
0.45454545454545453
0.36363636363636365
0.0
0.0
0.18181818181818182
0.0
0.11538461538461539
0.5192307692307693
0.038461538461538464
0.009615384615384616
0.2403846153846154
0.07692307692307693
0.0
0.375
0.3125
0.0
0.1875
0.125
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(C_after_C_prob + PC_after_C_prob + OV_after_C_prob + R_after_C_prob +␣
 ↪RPC_after_C_prob + ROV_after_C_prob)
print(C_after_PC_prob + PC_after_PC_prob + OV_after_PC_prob + R_after_PC_prob +␣
 ↪RPC_after_PC_prob + ROV_after_PC_prob)
print(C_after_OV_prob + PC_after_OV_prob + OV_after_OV_prob + R_after_OV_prob +␣
 ↪RPC_after_OV_prob + ROV_after_OV_prob)
print(C_after_R_prob + PC_after_R_prob + OV_after_R_prob + R_after_R_prob +␣
 ↪RPC_after_R_prob + ROV_after_R_prob)
print(C_after_RPC_prob + PC_after_RPC_prob + OV_after_RPC_prob +␣
 ↪R_after_RPC_prob + RPC_after_RPC_prob + ROV_after_RPC_prob)
print(C_after_ROV_prob + PC_after_ROV_prob + OV_after_ROV_prob +␣
 ↪R_after_ROV_prob + RPC_after_ROV_prob + ROV_after_ROV_prob)
```

```
1.0
1.0
0.9999999999999998
1.0
1.0
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_matrix = [[C_after_C_prob, PC_after_C_prob, OV_after_C_prob,␣
 ↪R_after_C_prob, RPC_after_C_prob, ROV_after_C_prob],
```

```
                    [C_after_PC_prob, PC_after_PC_prob, OV_after_PC_prob,␣
 ↪R_after_PC_prob, RPC_after_PC_prob, ROV_after_PC_prob],
                    [C_after_OV_prob, PC_after_OV_prob, OV_after_OV_prob,␣
 ↪R_after_OV_prob, RPC_after_OV_prob, ROV_after_OV_prob],
                    [C_after_R_prob, PC_after_R_prob, OV_after_R_prob,␣
 ↪R_after_R_prob, RPC_after_R_prob, ROV_after_R_prob],
                    [C_after_RPC_prob, PC_after_RPC_prob, OV_after_RPC_prob,␣
 ↪R_after_RPC_prob, RPC_after_RPC_prob, ROV_after_RPC_prob],
                    [C_after_ROV_prob, PC_after_ROV_prob, OV_after_ROV_prob,␣
 ↪R_after_ROV_prob, RPC_after_ROV_prob, ROV_after_ROV_prob]]
print(transition_matrix)
```

```
[[0.6495901639344263, 0.2725409836065574, 0.004098360655737705,
0.01639344262295082, 0.05737704918032787, 0.0], [0.196405648267009,
0.693196405648267, 0.044929396662387676, 0.002567394094993581,
0.05648267008985879, 0.006418485237483954], [0.017543859649122806,
0.7368421052631579, 0.19298245614035087, 0.0, 0.03508771929824561,
0.017543859649122806], [0.45454545454545453, 0.36363636363636365, 0.0, 0.0,
0.18181818181818182, 0.0], [0.11538461538461539, 0.5192307692307693,
0.038461538461538464, 0.009615384615384616, 0.2403846153846154,
0.07692307692307693], [0.0, 0.375, 0.3125, 0.0, 0.1875, 0.125]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.64959016 0.27254098 0.00409836 0.01639344 0.05737705 0.        ]
 [0.19640565 0.69319641 0.0449294  0.00256739 0.05648267 0.00641849]
 [0.01754386 0.73684211 0.19298246 0.         0.03508772 0.01754386]
 [0.45454545 0.36363636 0.         0.         0.18181818 0.        ]
 [0.11538462 0.51923077 0.03846154 0.00961538 0.24038462 0.07692308]
 [0.         0.375      0.3125     0.         0.1875     0.125     ]]
```

[ ]:
```
spring_seasons_test.head(1)
```

[ ]:
```
   index    datetime         condition
0   1638  2018-04-01  partially_cloudy
```

First Day of spring 2018: partially_cloudy

[ ]:
```
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 = 1 # 1 = partially_cloudy
    test_result = test_data.copy()

    prev_state = start_state
```

```
    result = []
    result.append(state[start_state])
    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(spring_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index     datetime          condition predicted_condition
```

```
    0   1638  2018-04-01  partially_cloudy    partially_cloudy
    1   1639  2018-04-02  partially_cloudy               clear
    2   1640  2018-04-03  partially_cloudy               clear
    3   1641  2018-04-04          overcast    partially_cloudy
    4   1642  2018-04-05  partially_cloudy    partially_cloudy
    0.36538461538461536
```

[ ]:
```
run_predictions_return_avg_accuracy(spring_seasons_test, 100)
```

[ ]:
```
0.40475274725274724
```

summer_six_conditions

December 7, 2022

1 Summer Season - 6 different weather conditions(long time frame)

1.1 Import libraries and dataset

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

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

[ ]:
```
summer.head()
```

[ ]:

	datetime	conditions
0	2000-07-01	Partially cloudy
1	2000-07-02	Partially cloudy
2	2000-07-03	Clear
3	2000-07-04	Partially cloudy
4	2000-07-05	Clear

1.2 Classify and separate data

[ ]:
```
classifier = {'Overcast':'overcast', 'Partially cloudy':'partially_cloudy',␣
↪'Clear':'clear', 'Rain, Partially cloudy':'rain_partially_cloudy', 'Rain':
↪'rain', 'Rain, Overcast':'rain_overcast'}

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

[ ]:
```
summer.head()
```

[ ]:

	datetime	conditions	condition
0	2000-07-01	Partially cloudy	partially_cloudy
1	2000-07-02	Partially cloudy	partially_cloudy
2	2000-07-03	Clear	clear
3	2000-07-04	Partially cloudy	partially_cloudy

```
4  2000-07-05            Clear             clear
```

[ ]:
```
summer = summer[['datetime', 'condition']]
```

[ ]:
```
summer.head()
```

[ ]:
```
     datetime         condition
0  2000-07-01  partially_cloudy
1  2000-07-02  partially_cloudy
2  2000-07-03             clear
3  2000-07-04  partially_cloudy
4  2000-07-05             clear
```

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
summer_train = summer.loc[summer['datetime'].between(train_start_date,␣
 ↪train_end_date)]
summer_train = summer_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
summer_test = summer.loc[summer['datetime'].between(test_start_date,␣
 ↪test_end_date)]
summer_test = summer_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

[ ]:
```
# 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

C_after_C_count = 0.0
PC_after_C_count = 0.0
OV_after_C_count = 0.0
R_after_C_count = 0.0
RPC_after_C_count = 0.0
ROV_after_C_count = 0.0

C_after_PC_count = 0.0
PC_after_PC_count = 0.0
OV_after_PC_count = 0.0
```

```
R_after_PC_count = 0.0
RPC_after_PC_count = 0.0
ROV_after_PC_count = 0.0

C_after_OV_count = 0.0
PC_after_OV_count = 0.0
OV_after_OV_count = 0.0
R_after_OV_count = 0.0
RPC_after_OV_count = 0.0
ROV_after_OV_count = 0.0

C_after_R_count = 0.0
PC_after_R_count = 0.0
OV_after_R_count = 0.0
R_after_R_count = 0.0
RPC_after_R_count = 0.0
ROV_after_R_count = 0.0

C_after_RPC_count = 0.0
PC_after_RPC_count = 0.0
OV_after_RPC_count = 0.0
R_after_RPC_count = 0.0
RPC_after_RPC_count = 0.0
ROV_after_RPC_count = 0.0

C_after_ROV_count = 0.0
PC_after_ROV_count = 0.0
OV_after_ROV_count = 0.0
R_after_ROV_count = 0.0
RPC_after_ROV_count = 0.0
ROV_after_ROV_count = 0.0
```

[ ]:
```
summer_train
```

[ ]:
```
      index    datetime         condition
0       184  2002-07-01             clear
1       185  2002-07-02  partially_cloudy
2       186  2002-07-03  partially_cloudy
3       187  2002-07-04  partially_cloudy
4       188  2002-07-05  partially_cloudy
…     …          …                 …
1467   1651  2017-09-26             clear
1468   1652  2017-09-27             clear
1469   1653  2017-09-28             clear
1470   1654  2017-09-29             clear
1471   1655  2017-09-30  partially_cloudy
```

```
[1472 rows x 3 columns]
```

[ ]:
```
# Count conditions

summer_train['condition_shift'] = summer_train['condition'].shift(-1)

for i in range(len(summer_train)):
    # Current 'clear'
    if summer_train.loc[i, 'condition'] == 'clear' and summer_train.loc[i,␣
↪'condition_shift'] == 'clear':
        C_after_C_count += 1
    elif summer_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'clear':
        PC_after_C_count += 1
    elif summer_train.loc[i, 'condition'] == 'overcast' and summer_train.loc[i,␣
↪'condition_shift'] == 'clear':
        OV_after_C_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain' and summer_train.loc[i,␣
↪'condition_shift'] == 'clear':
        R_after_C_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'clear':
        RPC_after_C_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_overcast' and summer_train.
↪loc[i, 'condition_shift'] == 'clear':
        ROV_after_C_count += 1
    # Current 'partially_cloudy'
    elif summer_train.loc[i, 'condition'] == 'clear' and summer_train.loc[i,␣
↪'condition_shift'] == 'partially_cloudy':
        C_after_PC_count += 1
    elif summer_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        PC_after_PC_count += 1
    elif summer_train.loc[i, 'condition'] == 'overcast' and summer_train.loc[i,␣
↪'condition_shift'] == 'partially_cloudy':
        OV_after_PC_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain' and summer_train.loc[i,␣
↪'condition_shift'] == 'partially_cloudy':
        R_after_PC_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'partially_cloudy':
        RPC_after_PC_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_overcast' and summer_train.
↪loc[i, 'condition_shift'] == 'partially_cloudy':
        ROV_after_PC_count += 1
    # Current 'overcast'
```

```
    elif summer_train.loc[i, 'condition'] == 'clear' and summer_train.loc[i,␣
↪'condition_shift'] == 'overcast':
        C_after_OV_count += 1
    elif summer_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'overcast':
        PC_after_OV_count += 1
    elif summer_train.loc[i, 'condition'] == 'overcast' and summer_train.loc[i,␣
↪'condition_shift'] == 'overcast':
        OV_after_OV_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain' and summer_train.loc[i,␣
↪'condition_shift'] == 'overcast':
        R_after_OV_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'overcast':
        RPC_after_OV_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_overcast' and summer_train.
↪loc[i, 'condition_shift'] == 'overcast':
        ROV_after_OV_count += 1
    # Current 'rain'
    elif summer_train.loc[i, 'condition'] == 'clear' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain':
        C_after_R_count += 1
    elif summer_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'rain':
        PC_after_R_count += 1
    elif summer_train.loc[i, 'condition'] == 'overcast' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain':
        OV_after_R_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'rain':
        RPC_after_R_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_overcast' and summer_train.
↪loc[i, 'condition_shift'] == 'rain':
        ROV_after_R_count += 1
    # Current 'rain_partially_cloudy'
    elif summer_train.loc[i, 'condition'] == 'clear' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain_partially_cloudy':
        C_after_RPC_count += 1
    elif summer_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        PC_after_RPC_count += 1
    elif summer_train.loc[i, 'condition'] == 'overcast' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain_partially_cloudy':
```

```
        OV_after_RPC_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain_partially_cloudy':
        R_after_RPC_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        RPC_after_RPC_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_overcast' and summer_train.
↪loc[i, 'condition_shift'] == 'rain_partially_cloudy':
        ROV_after_RPC_count += 1
    # Current 'rain_overcast'
    elif summer_train.loc[i, 'condition'] == 'clear' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain_overcast':
        C_after_ROV_count += 1
    elif summer_train.loc[i, 'condition'] == 'partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'rain_overcast':
        PC_after_ROV_count += 1
    elif summer_train.loc[i, 'condition'] == 'overcast' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain_overcast':
        OV_after_ROV_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain' and summer_train.loc[i,␣
↪'condition_shift'] == 'rain_overcast':
        R_after_ROV_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_partially_cloudy' and␣
↪summer_train.loc[i, 'condition_shift'] == 'rain_overcast':
        RPC_after_ROV_count += 1
    elif summer_train.loc[i, 'condition'] == 'rain_overcast' and summer_train.
↪loc[i, 'condition_shift'] == 'rain_overcast':
        ROV_after_ROV_count += 1
```

[ ]:
```
current_C_total = C_after_C_count + PC_after_C_count + OV_after_C_count +␣
↪R_after_C_count + RPC_after_C_count + ROV_after_C_count
current_PC_total = C_after_PC_count + PC_after_PC_count + OV_after_PC_count +␣
↪R_after_PC_count + RPC_after_PC_count + ROV_after_PC_count
current_OV_total = C_after_OV_count + PC_after_OV_count + OV_after_OV_count +␣
↪R_after_OV_count + RPC_after_OV_count + ROV_after_OV_count
current_R_total =C_after_R_count + PC_after_R_count + OV_after_R_count +␣
↪R_after_R_count + RPC_after_R_count + ROV_after_R_count
current_RPC_total = C_after_RPC_count + PC_after_RPC_count + OV_after_RPC_count␣
↪+ R_after_RPC_count + RPC_after_RPC_count + ROV_after_RPC_count
current_ROV_total = C_after_ROV_count + PC_after_ROV_count + OV_after_ROV_count␣
↪+ R_after_ROV_count + RPC_after_ROV_count + ROV_after_ROV_count
```

[ ]:
```
C_after_C_prob = C_after_C_count / current_C_total
PC_after_C_prob = PC_after_C_count / current_C_total
OV_after_C_prob = OV_after_C_count / current_C_total
```

```
R_after_C_prob = R_after_C_count / current_C_total
RPC_after_C_prob = RPC_after_C_count / current_C_total
ROV_after_C_prob = ROV_after_C_count / current_C_total

C_after_PC_prob = C_after_PC_count / current_PC_total
PC_after_PC_prob = PC_after_PC_count / current_PC_total
OV_after_PC_prob = OV_after_PC_count / current_PC_total
R_after_PC_prob = R_after_PC_count / current_PC_total
RPC_after_PC_prob = RPC_after_PC_count / current_PC_total
ROV_after_PC_prob = ROV_after_PC_count / current_PC_total

C_after_OV_prob = C_after_OV_count / current_OV_total
PC_after_OV_prob = PC_after_OV_count / current_OV_total
OV_after_OV_prob = OV_after_OV_count / current_OV_total
R_after_OV_prob = R_after_OV_count / current_OV_total
RPC_after_OV_prob = RPC_after_OV_count / current_OV_total
ROV_after_OV_prob = ROV_after_OV_count / current_OV_total

C_after_R_prob = C_after_R_count / current_R_total
PC_after_R_prob = PC_after_R_count / current_R_total
OV_after_R_prob = OV_after_R_count / current_R_total
R_after_R_prob = R_after_R_count / current_R_total
RPC_after_R_prob = RPC_after_R_count / current_R_total
ROV_after_R_prob = ROV_after_R_count / current_R_total

C_after_RPC_prob = C_after_RPC_count / current_RPC_total
PC_after_RPC_prob = PC_after_RPC_count / current_RPC_total
OV_after_RPC_prob = OV_after_RPC_count / current_RPC_total
R_after_RPC_prob = R_after_RPC_count / current_RPC_total
RPC_after_RPC_prob = RPC_after_RPC_count / current_RPC_total
ROV_after_RPC_prob = ROV_after_RPC_count / current_RPC_total

C_after_ROV_prob = C_after_ROV_count / current_ROV_total
PC_after_ROV_prob = PC_after_ROV_count / current_ROV_total
OV_after_ROV_prob = OV_after_ROV_count / current_ROV_total
R_after_ROV_prob = R_after_ROV_count / current_ROV_total
RPC_after_ROV_prob = RPC_after_ROV_count / current_ROV_total
ROV_after_ROV_prob = ROV_after_ROV_count / current_ROV_total
```

[ ]:
```
# Printing our probabilities for 6x6 transition matrix:
print(C_after_C_prob)
print(PC_after_C_prob)
print(OV_after_C_prob)
print(R_after_C_prob)
print(RPC_after_C_prob)
print(ROV_after_C_prob)
```

```
print(C_after_PC_prob)
print(PC_after_PC_prob)
print(OV_after_PC_prob)
print(R_after_PC_prob)
print(RPC_after_PC_prob)
print(ROV_after_PC_prob)

print(C_after_OV_prob)
print(PC_after_OV_prob)
print(OV_after_OV_prob)
print(R_after_OV_prob)
print(RPC_after_OV_prob)
print(ROV_after_OV_prob)

print(C_after_R_prob)
print(PC_after_R_prob)
print(OV_after_R_prob)
print(R_after_R_prob)
print(RPC_after_R_prob)
print(ROV_after_R_prob)

print(C_after_RPC_prob)
print(PC_after_RPC_prob)
print(OV_after_RPC_prob)
print(R_after_RPC_prob)
print(RPC_after_RPC_prob)
print(ROV_after_RPC_prob)

print(C_after_ROV_prob)
print(PC_after_ROV_prob)
print(OV_after_ROV_prob)
print(R_after_ROV_prob)
print(RPC_after_ROV_prob)
print(ROV_after_ROV_prob)
```

```
0.693069306930693
0.2623762376237624
0.0016501650165016502
0.02145214521452145
0.02145214521452145
0.0
0.21106821106821108
0.7348777348777349
0.021879021879021878
0.006435006435006435
0.02574002574002574
0.0
```

```
0.13636363636363635
0.7272727272727273
0.09090909090909091
0.0
0.0
0.045454545454545456
0.6190476190476191
0.2857142857142857
0.0
0.047619047619047616
0.047619047619047616
0.0
0.16279069767441862
0.5348837209302325
0.046511627906976744
0.046511627906976744
0.18604651162790697
0.023255813953488372
0.0
0.5
0.0
0.0
0.5
0.0
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(C_after_C_prob + PC_after_C_prob + OV_after_C_prob + R_after_C_prob +␣
 ↪RPC_after_C_prob + ROV_after_C_prob)
print(C_after_PC_prob + PC_after_PC_prob + OV_after_PC_prob + R_after_PC_prob +␣
 ↪RPC_after_PC_prob + ROV_after_PC_prob)
print(C_after_OV_prob + PC_after_OV_prob + OV_after_OV_prob + R_after_OV_prob +␣
 ↪RPC_after_OV_prob + ROV_after_OV_prob)
print(C_after_R_prob + PC_after_R_prob + OV_after_R_prob + R_after_R_prob +␣
 ↪RPC_after_R_prob + ROV_after_R_prob)
print(C_after_RPC_prob + PC_after_RPC_prob + OV_after_RPC_prob +␣
 ↪R_after_RPC_prob + RPC_after_RPC_prob + ROV_after_RPC_prob)
print(C_after_ROV_prob + PC_after_ROV_prob + OV_after_ROV_prob +␣
 ↪R_after_ROV_prob + RPC_after_ROV_prob + ROV_after_ROV_prob)
```

```
0.9999999999999999
1.0
1.0
1.0
0.9999999999999999
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_matrix = [[C_after_C_prob, PC_after_C_prob, OV_after_C_prob,␣
 ↪R_after_C_prob, RPC_after_C_prob, ROV_after_C_prob],
                     [C_after_PC_prob, PC_after_PC_prob, OV_after_PC_prob,␣
 ↪R_after_PC_prob, RPC_after_PC_prob, ROV_after_PC_prob],
                     [C_after_OV_prob, PC_after_OV_prob, OV_after_OV_prob,␣
 ↪R_after_OV_prob, RPC_after_OV_prob, ROV_after_OV_prob],
                     [C_after_R_prob, PC_after_R_prob, OV_after_R_prob,␣
 ↪R_after_R_prob, RPC_after_R_prob, ROV_after_R_prob],
                     [C_after_RPC_prob, PC_after_RPC_prob, OV_after_RPC_prob,␣
 ↪R_after_RPC_prob, RPC_after_RPC_prob, ROV_after_RPC_prob],
                     [C_after_ROV_prob, PC_after_ROV_prob, OV_after_ROV_prob,␣
 ↪R_after_ROV_prob, RPC_after_ROV_prob, ROV_after_ROV_prob]]
print(transition_matrix)
```

```
[[0.693069306930693, 0.2623762376237624, 0.0016501650165016502,
0.02145214521452145, 0.02145214521452145, 0.0], [0.21106821106821108,
0.7348777348777349, 0.021879021879021878, 0.006435006435006435,
0.02574002574002574, 0.0], [0.13636363636363635, 0.7272727272727273,
0.09090909090909091, 0.0, 0.0, 0.045454545454545456], [0.6190476190476191,
0.2857142857142857, 0.0, 0.047619047619047616, 0.047619047619047616, 0.0],
[0.16279069767441862, 0.5348837209302325, 0.046511627906976744,
0.046511627906976744, 0.18604651162790697, 0.023255813953488372], [0.0, 0.5,
0.0, 0.0, 0.5, 0.0]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.69306931 0.26237624 0.00165017 0.02145215 0.02145215 0.        ]
 [0.21106821 0.73487773 0.02187902 0.00643501 0.02574003 0.        ]
 [0.13636364 0.72727273 0.09090909 0.         0.         0.04545455]
 [0.61904762 0.28571429 0.         0.04761905 0.04761905 0.        ]
 [0.1627907  0.53488372 0.04651163 0.04651163 0.18604651 0.02325581]
 [0.         0.5        0.         0.         0.5        0.        ]]
```

[ ]:
```
summer_test.head(1)
```

[ ]:

	index	datetime	condition
0	1656	2018-07-01	partially_cloudy

First day of summer 2018: partially_cloudy

[ ]:
```
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 = [state[start_state]]
    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(summer_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime         condition predicted_condition
0   1656  2018-07-01  partially_cloudy               clear
1   1657  2018-07-02  partially_cloudy               clear
2   1658  2018-07-03  partially_cloudy               clear
3   1659  2018-07-04  partially_cloudy               clear
4   1660  2018-07-05             clear               clear
0.46195652173913043
```

[ ]:
```
run_predictions_return_avg_accuracy(summer_test, 100)
```

[ ]:
```
0.45853260869565204
```

UCSD_all_seasons_simplified

December 7, 2022

1 All Seasons - Simplifed(long time frame - UCSD)

1.1 Import libraries and dataset

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

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

[ ]:
```
all_seasons_UCSD.head()
```

[ ]:
```
     datetime              conditions
0  2000-01-01  Rain, Partially cloudy
1  2000-01-02       Partially cloudy
2  2000-01-03                  Clear
3  2000-01-04                  Clear
4  2000-01-05                  Clear
```

1.2 Classify and separate data

[ ]:
```
simplifier = {'Snow':'rain', 'Snow, Rain, Overcast':'rain', 'Snow, Rain,␣
 ↪Partially cloudy':'rain', 'Snow, Rain':'rain','Overcast':'no_rain',␣
 ↪'Partially cloudy':'no_rain', 'Clear':'no_rain', 'Rain, Partially cloudy':
 ↪'rain', 'Rain':'rain', 'Rain, Overcast':'rain'}

all_seasons_UCSD['condition'] = all_seasons_UCSD['conditions'].map(simplifier)
all_seasons_UCLA['condition'] = all_seasons_UCLA['conditions'].map(simplifier)
```

[ ]:
```
all_seasons_UCSD.head()
```

[ ]:
```
     datetime              conditions condition
0  2000-01-01  Rain, Partially cloudy      rain
```

```
1  2000-01-02        Partially cloudy   no_rain
2  2000-01-03                   Clear   no_rain
3  2000-01-04                   Clear   no_rain
4  2000-01-05                   Clear   no_rain
```

[ ]:
```
all_seasons_UCSD = all_seasons_UCSD[['datetime', 'condition']]
all_seasons_UCLA = all_seasons_UCLA[['datetime', 'condition']]
```

[ ]:
```
all_seasons_UCSD.head()
```

[ ]:
```
     datetime condition
0  2000-01-01      rain
1  2000-01-02   no_rain
2  2000-01-03   no_rain
3  2000-01-04   no_rain
4  2000-01-05   no_rain
```

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
all_seasons_UCLA_train = all_seasons_UCLA.loc[all_seasons_UCSD['datetime'].
 ↪between(train_start_date, train_end_date)]
all_seasons_UCLA_train = all_seasons_UCLA_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
all_seasons_UCSD_test = all_seasons_UCSD.loc[all_seasons_UCSD['datetime'].
 ↪between(test_start_date, test_end_date)]
all_seasons_UCSD_test = all_seasons_UCSD_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

We will refer to rain is 'R' and no rain as 'N'

[ ]:
```
# Initialize count variables
R_after_R_count = 0.0
N_after_R_count = 0.0

R_after_N_count = 0.0
N_after_N_count = 0.0
```

[ ]:
```
all_seasons_UCLA_train
```

[ ]:
```
      index    datetime condition
0       731  2002-01-01   no_rain
1       732  2002-01-02      rain
2       733  2002-01-03      rain
3       734  2002-01-04   no_rain
```

```
4       735  2002-01-05   no_rain
…       …          …        …
5839   6570  2017-12-27   no_rain
5840   6571  2017-12-28   no_rain
5841   6572  2017-12-29   no_rain
5842   6573  2017-12-30   no_rain
5843   6574  2017-12-31   no_rain

[5844 rows x 3 columns]
```

[ ]:
```
# Count conditions

all_seasons_UCLA_train['condition_shift'] = all_seasons_UCLA_train['condition'].
 ↪shift(-1)

for i in range(len(all_seasons_UCLA_train)):
    if all_seasons_UCLA_train.loc[i, 'condition'] == 'rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif all_seasons_UCLA_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'rain':
        N_after_R_count += 1
    elif all_seasons_UCLA_train.loc[i, 'condition'] == 'rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'no_rain':
        R_after_N_count += 1
    elif all_seasons_UCLA_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'no_rain':
        N_after_N_count += 1
```

[ ]:
```
current_R_total = R_after_R_count + N_after_R_count
current_N_total = R_after_N_count + N_after_N_count
```

[ ]:
```
R_after_R_prob = R_after_R_count / current_R_total
N_after_R_prob = N_after_R_count / current_R_total

R_after_N_prob = R_after_N_count / current_N_total
N_after_N_prob = N_after_N_count / current_N_total
```

[ ]:
```
# Printing our probabilities for 2x2 transition matrix:
print(R_after_R_prob)
print(N_after_R_prob)
print(R_after_N_prob)
print(N_after_N_prob)
```

```
0.4674887892376682
0.5325112107623319
0.09594021409816199
```

```
0.904059785901838
```

```
[ ]: # Checking that each row in the transition matrix adds up to 1:
print(R_after_R_prob + N_after_R_prob)
print(R_after_N_prob + N_after_N_prob)
```

```
1.0
1.0
```

```
[ ]: # Creating the transition matrix:
transition_name = [['RR', 'RN'], ['RN', 'NN']]
transition_matrix = [[R_after_R_prob, N_after_R_prob], [R_after_N_prob,␣
 ↪N_after_N_prob]]
print(transition_matrix)
```

```
[[0.4674887892376682, 0.5325112107623319], [0.09594021409816199,
0.904059785901838]]
```

```
[ ]: t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.46748879 0.53251121]
 [0.09594021 0.90405979]]
```

First Day of 2018: No Rain

```
[ ]: def predict_weather_simplified(test_data):
    state = {0:'rain', 1:'no_rain'}
    n = len(test_data) #how many steps to test
    start_state = 1 #1 = No Rain
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1], 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], 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_simplified(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_simplified(all_seasons_UCSD_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   6575  2018-01-01   no_rain              no_rain
1   6576  2018-01-02   no_rain              no_rain
2   6577  2018-01-03   no_rain              no_rain
3   6578  2018-01-04   no_rain              no_rain
4   6579  2018-01-05   no_rain              no_rain
0.7535934291581109
```

[ ]:
```
run_predictions_return_avg_accuracy(all_seasons_UCSD_test, 100)
```

[ ]:
```
0.7351403148528406
```

NYU_all_seasons_simplified

December 7, 2022

1 All Seasons - Simplifed(long time frame - NYU)

1.1 Import libraries and dataset

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

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

[ ]:
```
all_seasons_NYU.head()
```

[ ]:

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

1.2 Classify and separate data

[ ]:
```
simplifier = {'Snow, Rain, Ice, Overcast':'rain', 'Rain, Freezing Drizzle/
↪Freezing Rain, Ice, Partially cloudy':'rain', 'Snow, Rain, Freezing Drizzle/
↪Freezing Rain, Overcast':'rain', 'Snow':'rain', 'Snow, Rain, Overcast':
↪'rain', 'Snow, Rain, Partially cloudy':'rain', 'Snow, Rain':
↪'rain','Overcast':'no_rain', 'Partially cloudy':'no_rain', 'Clear':
↪'no_rain', 'Rain, Partially cloudy':'rain', 'Rain':'rain', 'Rain, Overcast':
↪'rain'}

all_seasons_NYU['condition'] = all_seasons_NYU['conditions'].map(simplifier)
all_seasons_UCLA['condition'] = all_seasons_UCLA['conditions'].map(simplifier)
```

[ ]:
```
all_seasons_NYU.head()
```

[ ]:

	datetime	conditions	condition
0	2000-01-01	Partially cloudy	no_rain
1	2000-01-02	Overcast	no_rain
2	2000-01-03	Overcast	no_rain
3	2000-01-04	Rain, Overcast	rain
4	2000-01-05	Rain, Partially cloudy	rain

[ ]:
```
all_seasons_NYU = all_seasons_NYU[['datetime', 'condition']]
all_seasons_UCLA = all_seasons_UCLA[['datetime', 'condition']]
```

[ ]:
```
all_seasons_NYU.head()
```

[ ]:

	datetime	condition
0	2000-01-01	no_rain
1	2000-01-02	no_rain
2	2000-01-03	no_rain
3	2000-01-04	rain
4	2000-01-05	rain

[ ]:
```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
all_seasons_UCLA_train = all_seasons_UCLA.loc[all_seasons_NYU['datetime'].
 ↪between(train_start_date, train_end_date)]
all_seasons_UCLA_train = all_seasons_UCLA_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
all_seasons_NYU_test = all_seasons_NYU.loc[all_seasons_NYU['datetime'].
 ↪between(test_start_date, test_end_date)]
all_seasons_NYU_test = all_seasons_NYU_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

We will refer to rain is 'R' and no rain as 'N'

[ ]:
```
# Initialize count variables
R_after_R_count = 0.0
N_after_R_count = 0.0

R_after_N_count = 0.0
N_after_N_count = 0.0
```

[ ]:
```
all_seasons_UCLA_train
```

[ ]:

	index	datetime	condition
0	731	2002-01-01	no_rain
1	732	2002-01-02	rain

```
2        733  2002-01-03      rain
3        734  2002-01-04   no_rain
4        735  2002-01-05   no_rain
…        …          …        …
5839    6570  2017-12-27   no_rain
5840    6571  2017-12-28   no_rain
5841    6572  2017-12-29   no_rain
5842    6573  2017-12-30   no_rain
5843    6574  2017-12-31   no_rain

[5844 rows x 3 columns]
```

[ ]:
```
# Count conditions

all_seasons_UCLA_train['condition_shift'] = all_seasons_UCLA_train['condition'].
 ↪shift(-1)

for i in range(len(all_seasons_UCLA_train)):
    if all_seasons_UCLA_train.loc[i, 'condition'] == 'rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'rain':
        R_after_R_count += 1
    elif all_seasons_UCLA_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'rain':
        N_after_R_count += 1
    elif all_seasons_UCLA_train.loc[i, 'condition'] == 'rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'no_rain':
        R_after_N_count += 1
    elif all_seasons_UCLA_train.loc[i, 'condition'] == 'no_rain' and␣
 ↪all_seasons_UCLA_train.loc[i, 'condition_shift'] == 'no_rain':
        N_after_N_count += 1
```

[ ]:
```
current_R_total = R_after_R_count + N_after_R_count
current_N_total = R_after_N_count + N_after_N_count
```

[ ]:
```
R_after_R_prob = R_after_R_count / current_R_total
N_after_R_prob = N_after_R_count / current_R_total

R_after_N_prob = R_after_N_count / current_N_total
N_after_N_prob = N_after_N_count / current_N_total
```

[ ]:
```
# Printing our probabilities for 2x2 transition matrix:
print(R_after_R_prob)
print(N_after_R_prob)
print(R_after_N_prob)
print(N_after_N_prob)
```

```
0.4674887892376682
0.5325112107623319
0.09594021409816199
0.904059785901838
```

[ ]:
```
# Checking that each row in the transition matrix adds up to 1:
print(R_after_R_prob + N_after_R_prob)
print(R_after_N_prob + N_after_N_prob)
```

```
1.0
1.0
```

[ ]:
```
# Creating the transition matrix:
transition_name = [['RR', 'RN'], ['RN', 'NN']]
transition_matrix = [[R_after_R_prob, N_after_R_prob], [R_after_N_prob,␣
 ↪N_after_N_prob]]
print(transition_matrix)
```

```
[[0.4674887892376682, 0.5325112107623319], [0.09594021409816199,
0.904059785901838]]
```

[ ]:
```
t_array = np.array(transition_matrix)
print(t_array)
```

```
[[0.46748879 0.53251121]
 [0.09594021 0.90405979]]
```

First Day of 2018: No Rain

[ ]:
```
def predict_weather_simplified(test_data):
    state = {0:'rain', 1:'no_rain'}
    n = len(test_data) #how many steps to test
    start_state = 1 #1 = No Rain
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1], 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], 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_simplified(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_simplified(all_seasons_NYU_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   6575  2018-01-01   no_rain            no_rain
1   6576  2018-01-02   no_rain            no_rain
2   6577  2018-01-03      rain               rain
3   6578  2018-01-04      rain               rain
4   6579  2018-01-05   no_rain               rain
0.4414784394250513
```

[ ]:
```
run_predictions_return_avg_accuracy(all_seasons_NYU_test, 100)
```

[ ]:
```
0.4355099247091032
```

Weather Prediction with Markov Chains

Math 42 Final Project
Department of Mathematics, UCLA

Junwon Choi
Brian Chau
Yifan Jiang
Mingyeong Kim
Nalin Chopra

Table of contents

01

Problem description

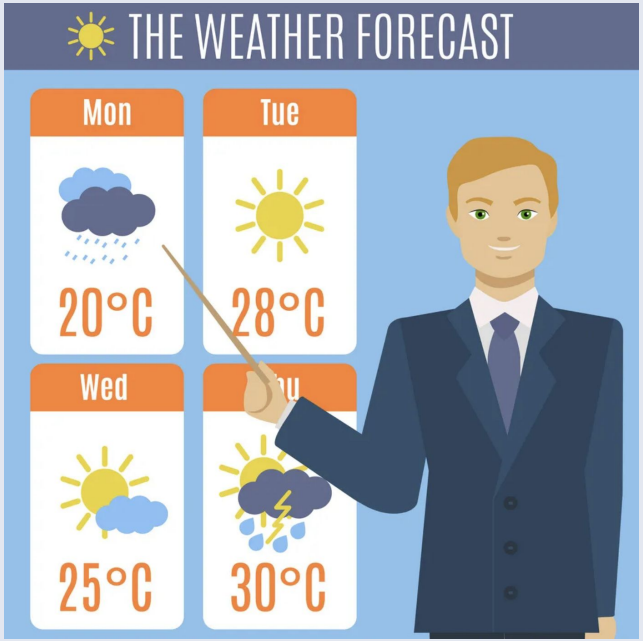
Present the problem we are attempting to solve with background information

How can we predict the weather?

Weather app? Guessing? News Forecasting?

Problem description

- The only method to predict the weather was to use the past data of local experience
- Mathematical model + the data = weather prediction



Three questions about weather prediction

- Which mathematical model is most appropriate for weather prediction?
- How can we fit the selected model to predict weather in a particular location?
- How accurate/significant is the model? How does it perform when fitted to different locations?

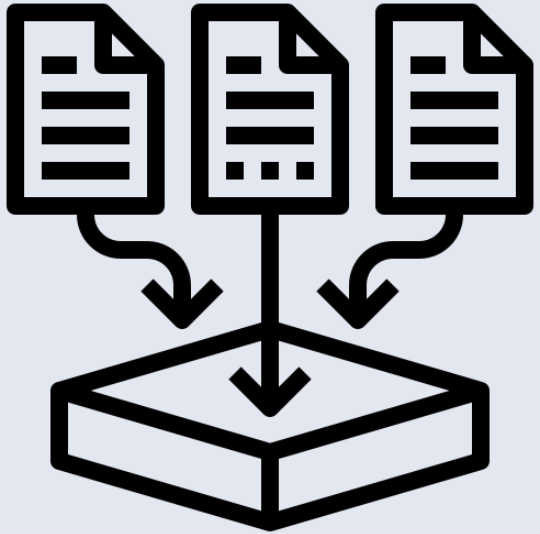
02

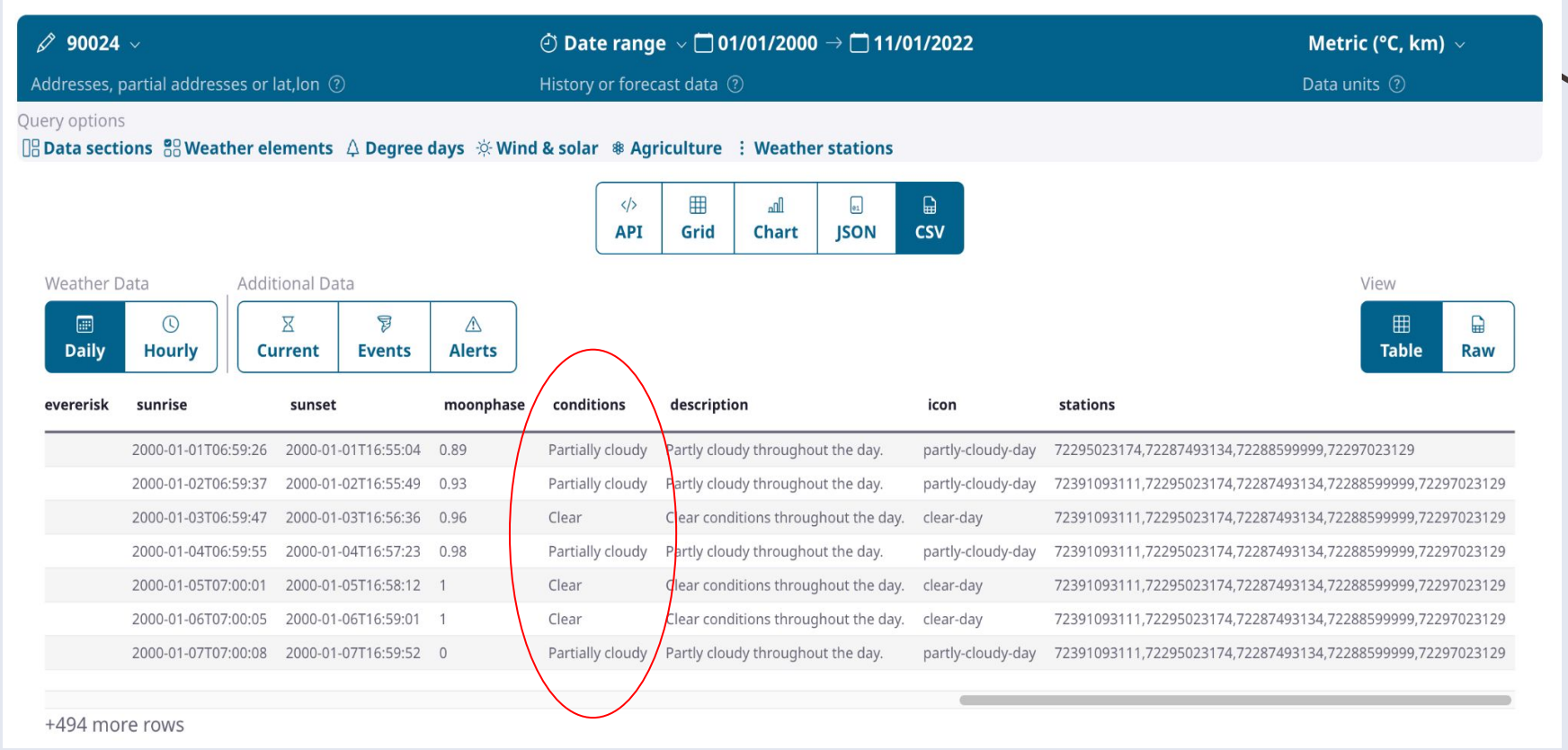
Simplifications

Explain the simplified way of the original problem. Justify the assumptions.

Data collection

- Weather data was collected from “https://www.visualcrossing.com”
  - UCLA (zip code - 90024)
  - From the year 2000 and 2022
- Task:
  - Find categorical variables for discrete Markov chain




90024
Addresses, partial addresses or lat,lon
Date range
01/01/2000
11/01/2022
History or forecast data
Metric (°C, km)
Data units
Query options
Data sections
Weather elements
Degree days
Wind & solar
Agriculture
Weather stations
API
Grid
Chart
JSON
CSV
Weather Data
Daily
Hourly
Additional Data
Current
Events
Alerts
View
Table
Raw
evererisk sunrise sunset moonphase conditions description icon stations
2000-01-01T06:59:26 2000-01-01T16:55:04 0.89 Partially cloudy Partly cloudy throughout the day. partly-cloudy-day 72295023174,72287493134,72288599999,72297023129
2000-01-02T06:59:37 2000-01-02T16:55:49 0.93 Partially cloudy Partly cloudy throughout the day. partly-cloudy-day 72391093111,72295023174,72287493134,72288599999,72297023129
2000-01-03T06:59:47 2000-01-03T16:56:36 0.96 Clear Clear conditions throughout the day. clear-day 72391093111,72295023174,72287493134,72288599999,72297023129
2000-01-04T06:59:55 2000-01-04T16:57:23 0.98 Partially cloudy Partly cloudy throughout the day. partly-cloudy-day 72391093111,72295023174,72287493134,72288599999,72297023129
2000-01-05T07:00:01 2000-01-05T16:58:12 1 Clear Clear conditions throughout the day. clear-day 72391093111,72295023174,72287493134,72288599999,72297023129
2000-01-06T07:00:05 2000-01-06T16:59:01 1 Clear Clear conditions throughout the day. clear-day 72391093111,72295023174,72287493134,72288599999,72297023129
2000-01-07T07:00:08 2000-01-07T16:59:52 0 Partially cloudy Partly cloudy throughout the day. partly-cloudy-day 72391093111,72295023174,72287493134,72288599999,72297023129
+494 more rows

Analysis steps

- All seasons + Simplified (Long time frame)
- All seasons + Simplified (Short time frame)
- All seasons + 6 different weather conditions (Long time frame)
- All seasons + 6 different weather conditions (Short time frame)
- By season + 6 different weather conditions (Long time frame)
- All seasons + Simplified applied to UCSD/NYU (Long time frame)
    - For validation

Assumptions and Limitations

Assumptions:

- Weather condition for a day only depend on weather condition of the day before

Limitations:

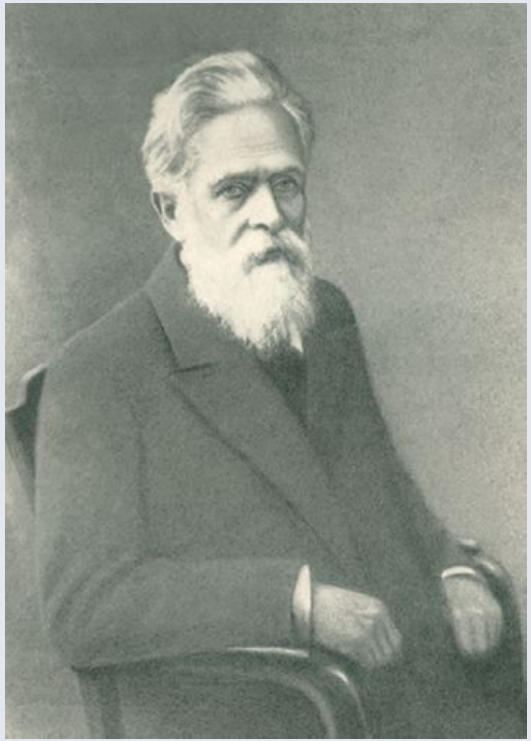
- Failure to account for other factors like temperature and humidity changes

03

Mathematical model

Markov chain explanation with the standard mathematical paradigm

What are Markov Chains?



Andrey Andreyevich Markov

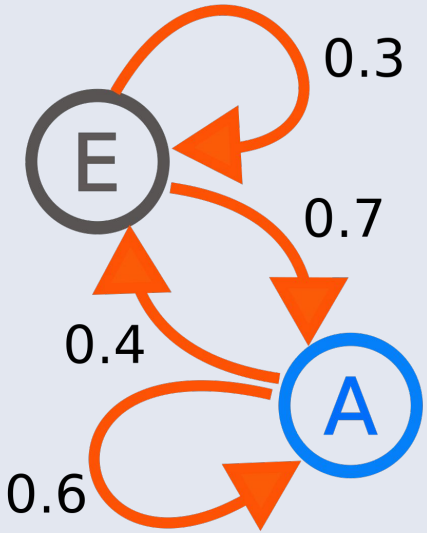
Markov chain

- A stochastic system that describes the sequence of possible events
- Properties:
  - The Markov assumption:

$$P(s_t|s_{t-1}, s_{t-2}, \ldots, s_0) = P(s_t|s_{t-1})$$

  - Conservation: the sum of the probabilities out of a state is equal to 1

Markov chain visual sample



Transition Matrix

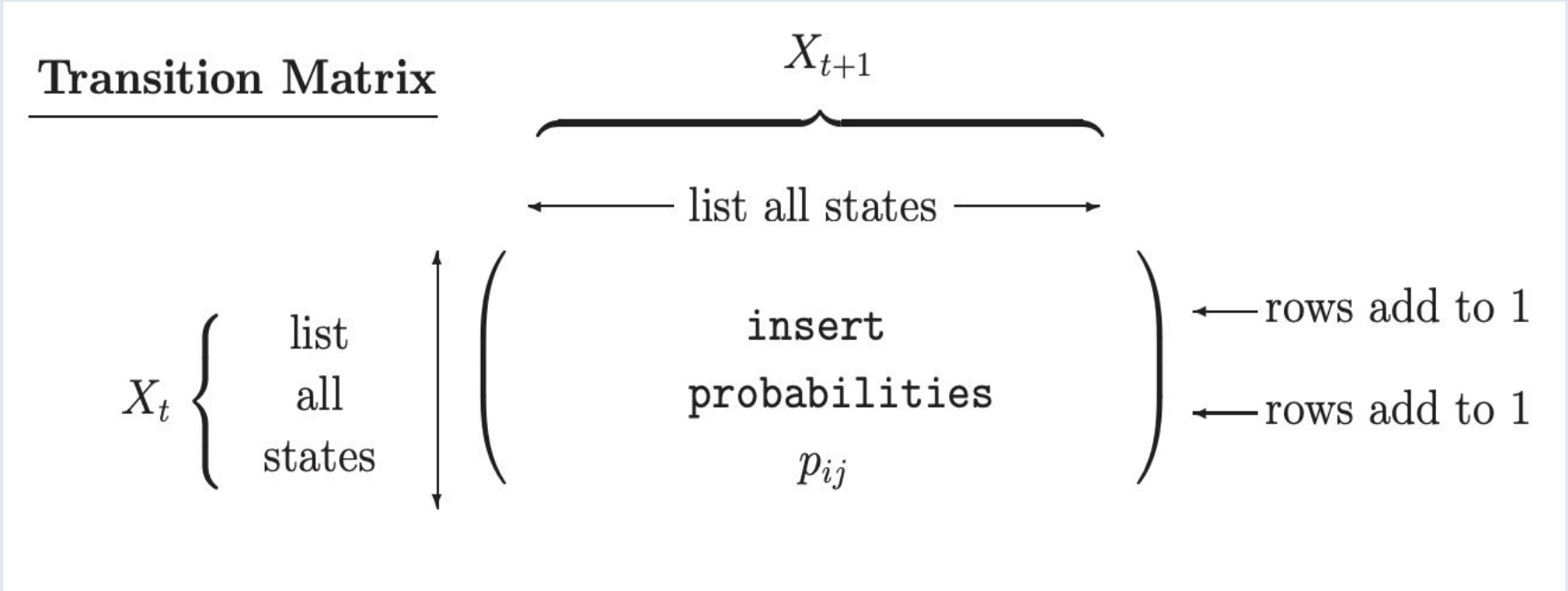


Image Source: https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf

Application to Weather

- S = {Clear; Partially cloudy; Overcast; Rain; Rain, Partially cloudy; Rain, Overcast}
- A = {Rain, No Rain}

$$\mathrm{T}_{\mathrm{simplified}} = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} P(R|R) & P(NR|R) \\ P(R|NR) & P(NR|NR) \end{pmatrix} \end{array}$$

04

Solution

Techniques to solve the mathematical problem

Calculating Transition Matrix

Training Data:

Day 1: Rain

Day 2: Rain

Day 3: No Rain

Day 4: Rain

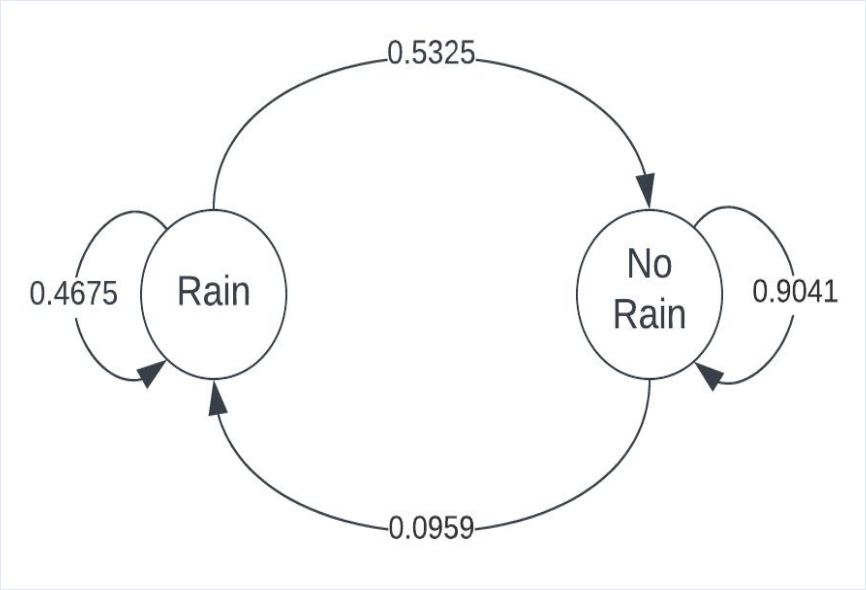
...

Day N: No Rain

- P(Rain | Rain)
- P(No Rain | Rain)
- P(Rain | No Rain)
- P(No Rain | No Rain)

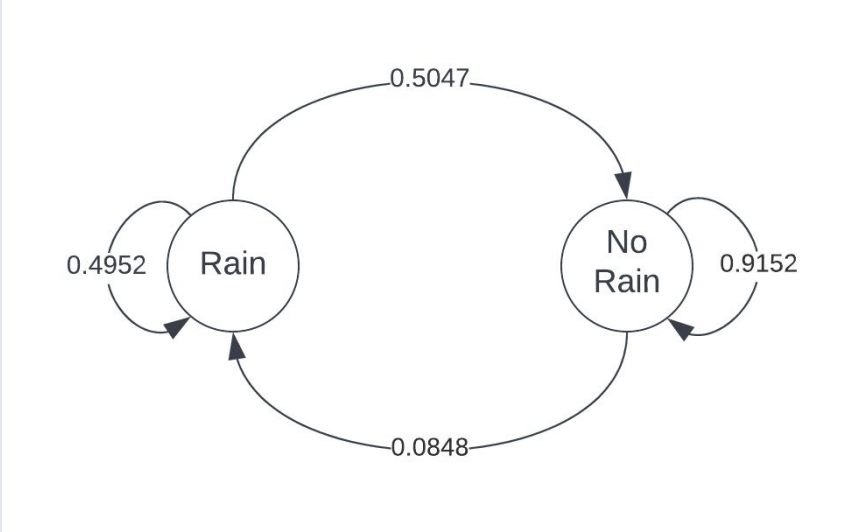
All Seasons, simplified analysis (long time frame)

$$T = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.467 & 0.533 \\ 0.096 & 0.904 \end{pmatrix} \end{array}$$



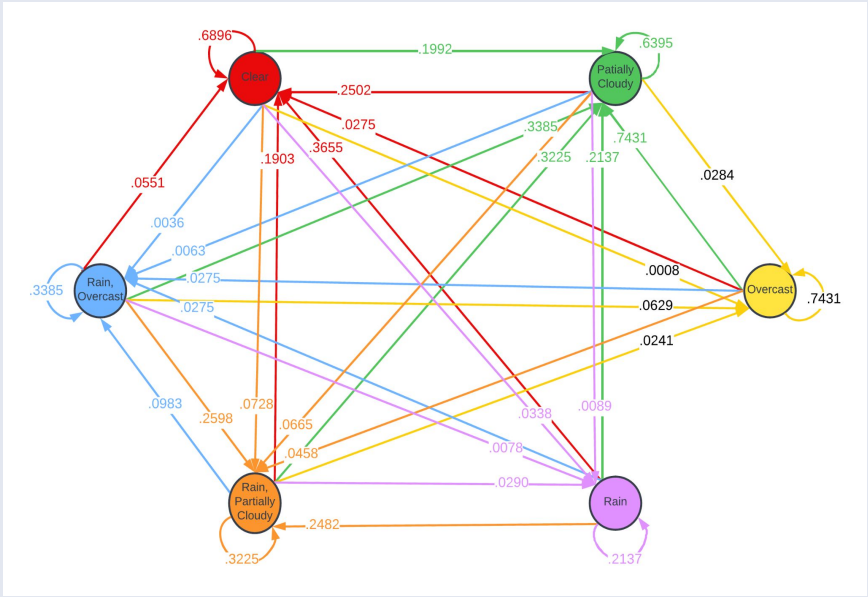
All Seasons, simplified analysis (short time frame)

$$T = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.495 & 0.505 \\ 0.085 & 0.915 \end{pmatrix} \end{array}$$



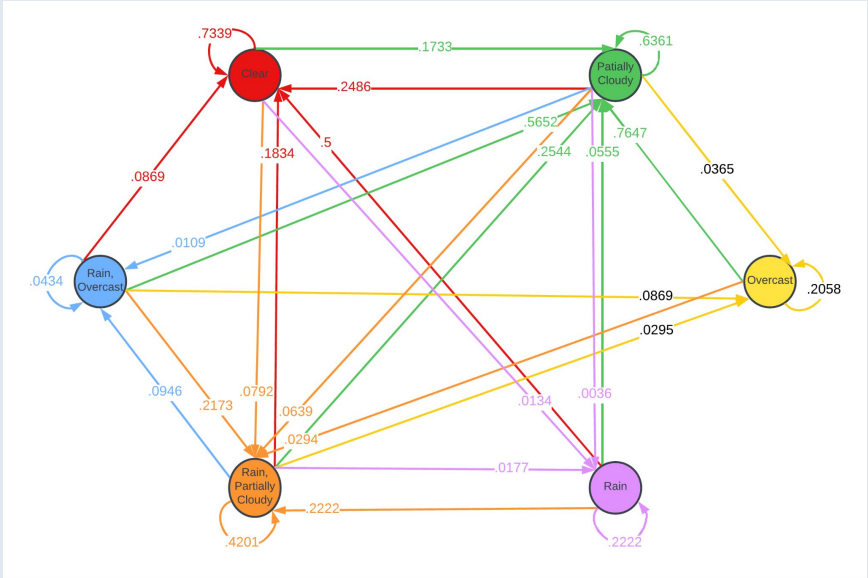
All Seasons, 6 weather conditions (long time frame)

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.690 & 0.199 & 0.001 & 0.034 & 0.073 & 0.004 \\ 0.250 & 0.640 & 0.028 & 0.009 & 0.067 & 0.006 \\ 0.028 & 0.743 & 0.156 & 0.0 & 0.046 & 0.028 \\ 0.366 & 0.214 & 0.0 & 0.145 & 0.248 & 0.028 \\ 0.190 & 0.323 & 0.0242 & 0.029 & 0.335 & 0.098 \\ 0.055 & 0.339 & 0.0630 & 0.008 & 0.260 & 0.276 \end{pmatrix} \end{array}$$



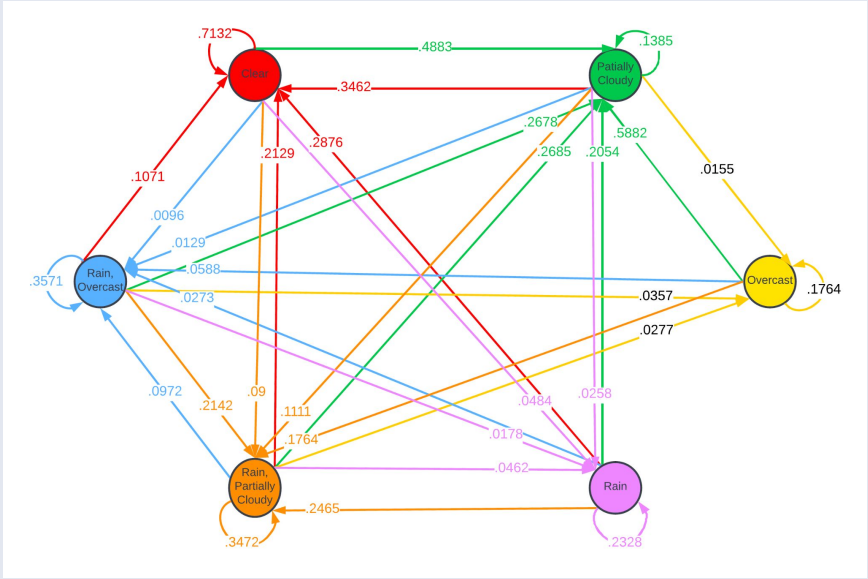
All Seasons, 6 weather conditions (short time frame)

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.734 & 0.173 & 0.0 & 0.013 & 0.079 & 0.0 \\ 0.249 & 0.636 & 0.037 & 0.004 & 0.064 & 0.011 \\ 0.0 & 0.765 & 0.206 & 0.0 & 0.029 & 0.0 \\ 0.5 & 0.056 & 0.0 & 0.222 & 0.222 & 0.0 \\ 0.183 & 0.254 & 0.0296 & 0.018 & 0.420 & 0.095 \\ 0.087 & 0.565 & 0.087 & 0. & 0.217 & 0.043 \end{pmatrix} \end{array}$$



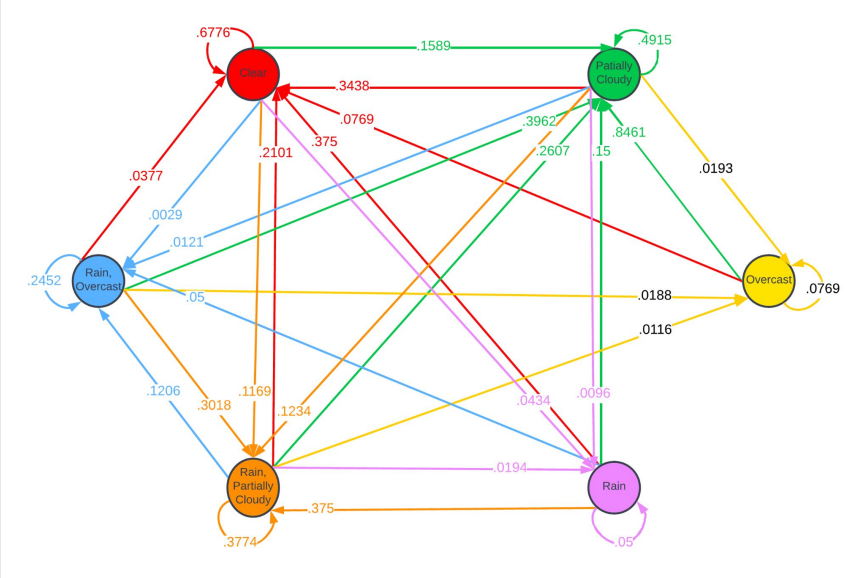
6 weather conditions, Fall

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.713 & 0.139 & 0.0 & 0.048 & 0.090 & 0.010 \\ 0.346 & 0.488 & 0.016 & 0.026 & 0.111 & 0.013 \\ 0.0 & 0.588 & 0.176 & 0.0 & 0.176 & 0.059 \\ 0.288 & 0.205 & 0.0 & 0.233 & 0.247 & 0.027 \\ 0.213 & 0.269 & 0.028 & 0.046 & 0.347 & 0.097 \\ 0.107 & 0.268 & 0.036 & 0.018 & 0.214 & 0.357 \end{pmatrix} \end{array}$$



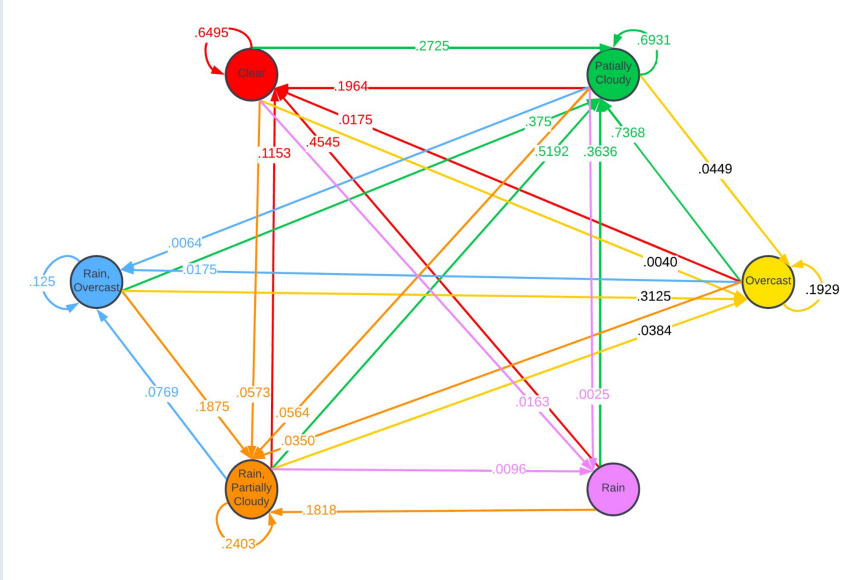
6 weather conditions, Winter

$$T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array} \begin{array}{c} \begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\ \begin{pmatrix} 0.677 & 0.158 & 0.0 & 0.043 & 0.116 & 0.002 \\ 0.343 & 0.491 & 0.019 & 0.009 & 0.123 & 0.012 \\ 0.076 & 0.846 & 0.076 & 0.0 & 0.0 & 0.0 \\ 0.375 & 0.15 & 0.0 & 0.05 & 0.375 & 0.05 \\ 0.210 & 0.261 & 0.012 & 0.019 & 0.377 & 0.121 \\ 0.038 & 0.396 & 0.019 & 0.0 & 0.302 & 0.245 \end{pmatrix} \end{array}$$



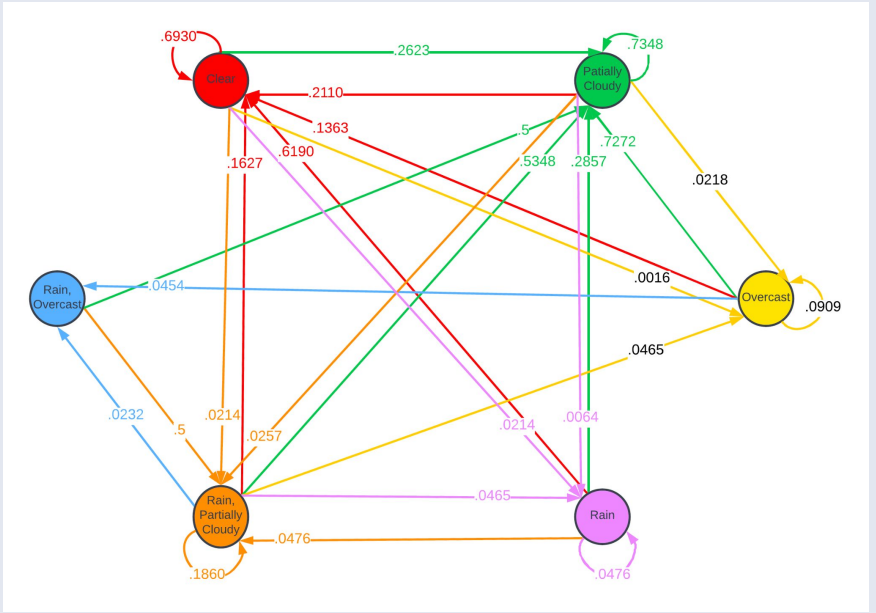
6 weather conditions, Spring

$$
T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array}
\begin{array}{c}
\begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\
\begin{pmatrix}
0.649 & 0.272 & 0.004 & 0.016 & 0.057 & 0.0 \\
0.196 & 0.693 & 0.044 & 0.002 & 0.056 & 0.006 \\
0.017 & 0.736 & 0.192 & 0.0 & 0.035 & 0.017 \\
0.454 & 0.363 & 0.0 & 0.0 & 0.181 & 0.0 \\
0.115 & 0.519 & 0.038 & 0.009 & 0.240 & 0.076 \\
0.0 & 0.375 & 0.312 & 0.0 & 0.187 & 0.125
\end{pmatrix}
\end{array}
$$



6 weather conditions, Summer

$$
T = \begin{array}{c} \\ C \\ PC \\ OV \\ R \\ RPC \\ ROV \end{array}
\begin{array}{c}
\begin{array}{cccccc} C & PC & OV & R & RPC & ROV \end{array} \\
\begin{pmatrix}
0.693 & 0.262 & 0.001 & 0.021 & 0.021 & 0.0 \\
0.211 & 0.734 & 0.021 & 0.006 & 0.025 & 0.0 \\
0.136 & 0.727 & 0.090 & 0.0 & 0.0 & 0.045 \\
0.619 & 0.285 & 0.0 & 0.047 & 0.047 & 0.0 \\
0.162 & 0.534 & 0.046 & 0.046 & 0.186 & 0.023 \\
0.0 & 0.5 & 0.0 & 0.0 & 0.5 & 0.0
\end{pmatrix}
\end{array}
$$



UCSD / NYU

- UCSD:

$$T = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.467 & 0.532 \\ 0.095 & 0.904 \end{pmatrix} \end{array}$$

- NYU:

$$T = \begin{array}{c} \\ \text{Rain (R)} \\ \text{No Rain (NR)} \end{array} \begin{array}{c} \begin{array}{cc} \text{Rain (R)} & \text{No Rain (NR)} \end{array} \\ \begin{pmatrix} 0.467 & 0.532 \\ 0.095 & 0.904 \end{pmatrix} \end{array}$$

Same transition matrices as UCLA!

Applying Transition Matrix

Transition Matrix → **Testing Data**

Applying Transition Matrix

Testing Data + Predictions

Date	Original Condition	Predicted Condition
Test Start Date … Test End Date	↑ Already downloaded ↓	↑ Appended at each iteration ↓

Prediction process

At each step:

For example, with an all season simplified model (long time frame):
Let Day 1 of the model be a 'No Rain' day, which would give a vector $V = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. We take transition matrix $T = \begin{pmatrix} 0.467 & 0.533 \\ 0.096 & 0.904 \end{pmatrix}$.

- Multiply V and the transpose of T using a Python function.
- This outputs a vector with two elements containing probabilities of 'Rain' or 'No Rain' given the past day is 'No Rain'.
- Markov chain model's key feature is randomness: Python function to use the two probabilities of the transposed second row as inputs to make a random choice.
- This choice determines whether Day 2 will be 'Rain' or 'No Rain'.
- Repeat this step by a desired amount.

```python
def predict_weather_simplified(test_data):
    state = {0:'rain', 1:'no_rain'}
    n = len(test_data) #how many steps to test
    start_state = 1 #1 = No Rain
    test_result = test_data.copy()

    prev_state = start_state
    result = []
    result.append(state[start_state])
    while n-1:
        curr_state = np.random.choice([0,1], 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], 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_simplified(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_simplified(all_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   6575  2018-01-01   no_rain              no_rain
1   6576  2018-01-02   no_rain              no_rain
2   6577  2018-01-03   no_rain              no_rain
3   6578  2018-01-04   no_rain              no_rain
4   6579  2018-01-05   no_rain              no_rain
0.7652292950034223
```

```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

```
0.7703011635865848
```

05

Result / Improvement

Analysis of results and discovery improvements

Result

The accuracy for all 10 of our models:

- All Seasons - Simplified(long time frame): **77%**
- All Seasons - Simplified(short time frame): **79%**
- All Seasons - 6 different weather conditions(long time frame): **37%**
- All Seasons - 6 different weather conditions(short time frame): **39%**
- By Season - 6 different weather conditions(long time frame)
  - Fall: **37% /** Winter: **35% /** Spring: **40% /** Summer: **45%**
- All Seasons - Simplified(long time frame) applied to UCSD/NYU
  - UCSD: **73%**
  - NYU: **43%**

Improvement

- Implementation of time-inhomogeneous Markov chains
  - Our Transition Matrix stays constant
  - Time-inhomogeneous Markov chains Transition Matrix is generated again at each time step
  - Could lead to increase in accuracy

Improvement

- Extending time frame of analysis with second-order Markov chains
  - What is it?(This model uses the past two states to predict the next state, instead of one past state)
  - Attempted to implement
  - Met problem that prevents us from doing it.
    - No observation where the two previous states were ‘clear’ --> ’overcast’
    - Need more data

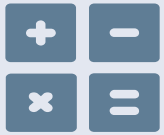
06

Conclusion

Summarize what we have done and what we have learned

What we have done / learned

Project achievements:



- make a prediction model for weather that is independent of climate measurements
- expanded our model's scope by applying the transition matrices derived

Project takeaways:

- real world application of the Markov chain model
- common thinking process in a more accurate and systematic manner.

References

[1] How has weather forecasting changed over the past two hundred years? *The American Geosciences Institute*, 2022.

[2] Visual Crossing Corporation, 2022.

[3] Randall Swift Douglas D. Mooney. *A Course in Mathematical Modeling*, page 122. The Mathematical Association of America, 1999.

[4] Soumya Karlamangla. California is expected to enter a fourth straight year of drought. *The New York Times*, October.

[5] Henry Maltby, et al. Markov chains. *Brilliant.org*, December 2022.

Questions?