# 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
```