COLLEGE *of* COMPUTER STUDIES

# Final Project and Final Exam CSEC 413-MODELING AND SIMULATION

## Modeling and Simulation with Python

**Presented by:**
**Kyle Magistrado**
**Jolina Caganda**
**Aaron Francis Pacardo**

**Mr. Allan O. Ibo Jr.**
*Instructor*

## Introduction

This project focuses on the application of Python for modeling and simulation tasks, with a main emphasis on gaining practical experience with popular Python libraries and tools. The overall goal is to learn how to use these tools effectively in the stages of data creation, exploratory data analysis (EDA), model building, and interactive forecasting. By leveraging Python's flexible features, the project seeks to offer a thorough understanding of the modeling and simulation methods, enhancing practical abilities and knowledge in the field of data-driven forecasts. The project consists of creating synthetic weather data, examining it through exploratory data visualization, designing and training a weather forecasting model, and developing an interactive forecasting component, all using Python and well-known libraries such as NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn, and Jupyter widgets. This hands-on strategy ensures a comprehensive and applied grasp of modeling and simulation techniques within the Python environment.

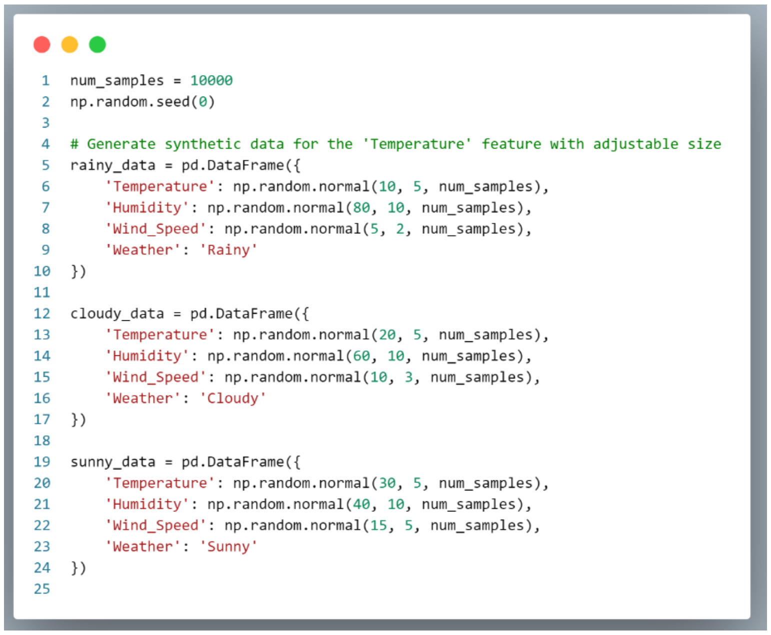
## Project Overview

The project involves several key steps, each contributing to the overall understanding and effectiveness of the weather prediction model. These steps include data generation, exploratory data analysis (EDA), modeling using the KNN classifier, simulation of weather conditions, and the evaluation and analysis of the model's performance.

## Data generation

*Weather Classification*

In the data generation step, synthetic data is created to simulate weather conditions. The following code snippet utilizes the numpy and pandas libraries to generate synthetic data for three weather classes (Rainy, Cloudy, Sunny), each with specific characteristics:



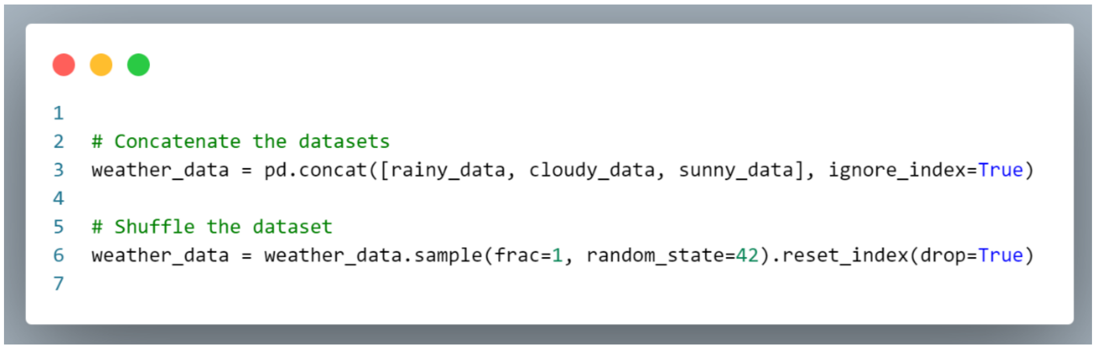
```python
num_samples = 10000
np.random.seed(0)

# Generate synthetic data for the 'Temperature' feature with adjustable size
rainy_data = pd.DataFrame({
    'Temperature': np.random.normal(10, 5, num_samples),
    'Humidity': np.random.normal(80, 10, num_samples),
    'Wind_Speed': np.random.normal(5, 2, num_samples),
    'Weather': 'Rainy'
})

cloudy_data = pd.DataFrame({
    'Temperature': np.random.normal(20, 5, num_samples),
    'Humidity': np.random.normal(60, 10, num_samples),
    'Wind_Speed': np.random.normal(10, 3, num_samples),
    'Weather': 'Cloudy'
})

sunny_data = pd.DataFrame({
    'Temperature': np.random.normal(30, 5, num_samples),
    'Humidity': np.random.normal(40, 10, num_samples),
    'Wind_Speed': np.random.normal(15, 5, num_samples),
    'Weather': 'Sunny'
})

```

Figure 1. Synthetic Data Generation

The provided code generates synthetic weather data to simulate diverse meteorological conditions, encapsulated in three distinct DataFrames: rainy_data, cloudy_data, and sunny_data. Each DataFrame corresponds to a specific weather condition—Rainy, Cloudy, and Sunny—and encompasses simulated values for key meteorological features: Temperature, Humidity, and Wind Speed. The generation process employs the NumPy library to create data points following normal distributions with specified mean and standard deviation parameters, mimicking realistic weather variations. For the Rainy condition, the 'Temperature' is generated with a mean of 10 and a standard deviation of 5, 'Humidity' with a mean of 80 and a standard deviation of 10, and 'Wind_Speed' with a mean of 5 and a standard deviation of 2. Similarly, Cloudy and Sunny conditions have their respective characteristic distributions for these features. The 'Weather' column in each DataFrame is appropriately labeled to denote the associated weather condition.

*Data Preprocessing*

This synthetic weather data serves as the foundation for subsequent phases in the weather prediction project, facilitating data exploration, model training, and interactive prediction. The diverse scenarios captured in the generated datasets aim to ensure that the weather prediction model is exposed to a comprehensive range of conditions, enhancing its ability to generalize and make accurate predictions across various weather scenarios.



```

# Concatenate the datasets
weather_data = pd.concat([rainy_data, cloudy_data, sunny_data], ignore_index=True)

# Shuffle the dataset
weather_data = weather_data.sample(frac=1, random_state=42).reset_index(drop=True)

```

Figure 2. Data Preprocessing

In the preprocessing phase of the weather prediction project, the code snippet weather_data = pd.concat([rainy_data, cloudy_data, sunny_data], ignore_index=True) combines synthetic weather data from three distinct conditions (Rainy, Cloudy, Sunny) into a single comprehensive DataFrame named weather_data. The ignore_index=True parameter is employed to reset the index of the resulting DataFrame, ensuring a continuous sequence of indices. This concatenation step is pivotal as it consolidates diverse weather conditions into a unified dataset, laying the foundation for subsequent model training. Subsequently, the line weather_data = weather_data.sample(frac=1, random_state=42).reset_index(drop=True) introduces a shuffling mechanism to the dataset. The sample method is utilized to randomly shuffle the rows of the weather_data DataFrame, with frac=1 indicating that the entire dataset is to be shuffled. The random_state=42 parameter is set to maintain reproducibility by fixing the random seed. Following the shuffling, the reset_index method is employed with drop=True to reset the DataFrame's index, discarding the previous index without incorporating it as a new column. This strategic randomization ensures that the model encounters a diverse range of examples during training, mitigating the risk of learning patterns associated with the original order of the data.

## Exploratory Data Analysis

Before training the weather prediction model, Exploratory Data Analysis (EDA) is conducted to gain insights into the synthetic weather data. The generated data is combined into a unified dataset, and the order of entries is randomized to prevent bias from the original order. The EDA focuses on the distribution of numerical features—Temperature, Humidity, and Wind Speed—across different weather conditions (Rainy, Cloudy, Sunny).

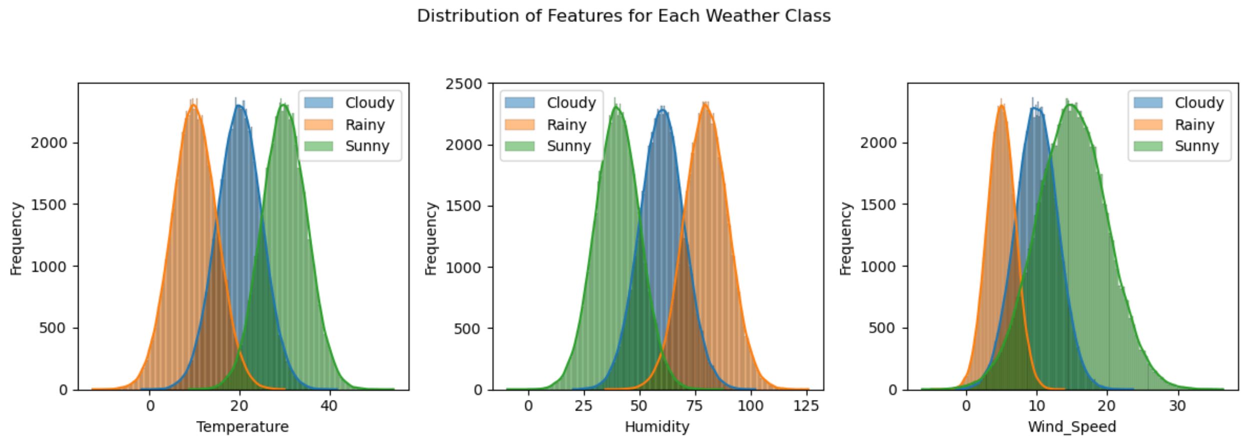
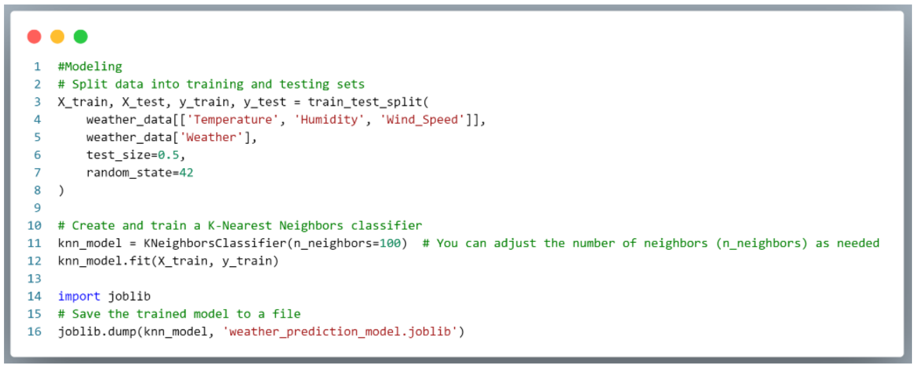


Figure 3. Class Feature Distribution

The resulting visualizations provide a comprehensive understanding of each numerical feature's distribution within distinct weather classes. For instance, the histogram depicting the distribution of temperatures reveals insights into the temperature ranges associated with Rainy, Cloudy, and Sunny conditions. Similarly, the histograms for Humidity and Wind Speed showcase the variability and patterns within each weather class. These visualizations are instrumental in identifying potential correlations or distinctions that can influence the model's learning during training. For example, if the EDA indicates that Rainy conditions consistently exhibit lower temperatures and higher humidity compared to other weather classes, the model may learn to recognize these patterns. This preliminary exploration guides subsequent modeling decisions, ensuring that the model is equipped with a nuanced understanding of the relationships between features and weather conditions. The EDA output serves as a valuable foundation for building a robust and informed weather prediction model.

## Modeling

In the modeling phase of the weather prediction project, a K-Nearest Neighbors (KNN) classifier is employed to discern patterns and relationships within the synthetic weather data. The dataset is initially split in half into training and testing sets, with features encompassing 'Temperature', 'Humidity', and 'Wind_Speed', and the target variable being 'Weather'. Notably, the split is stratified to ensure a representative distribution of weather classes in both sets. The KNN classifier, instantiated with n_neighbors=100, signifies that the model considers 100 nearest neighbors when making predictions. This parameter choice influences the model's sensitivity to local patterns in the data; a larger value of n_neighbors tends to result in a smoother decision boundary and may mitigate the impact of noise. The model is then trained on the training data, empowering it to learn the associations between the features and corresponding weather conditions.



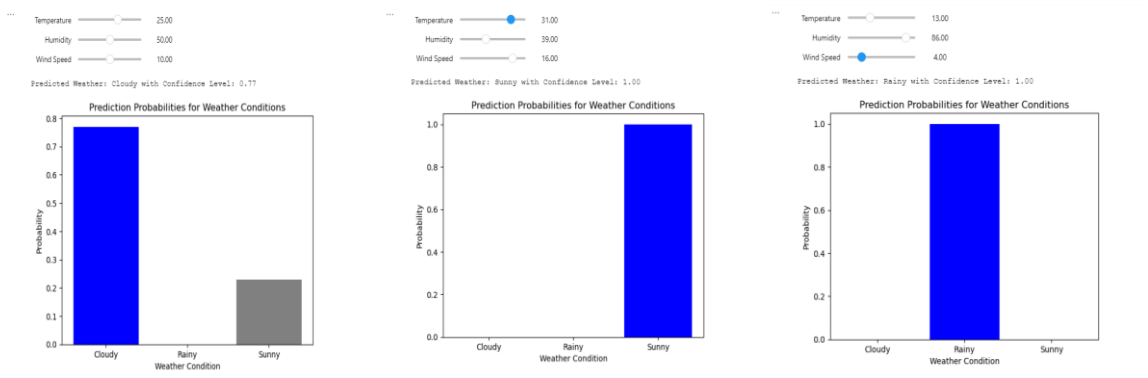
```python
#Modeling
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    weather_data[['Temperature', 'Humidity', 'Wind_Speed']],
    weather_data['Weather'],
    test_size=0.5,
    random_state=42
)

# Create and train a K-Nearest Neighbors classifier
knn_model = KNeighborsClassifier(n_neighbors=100)  # You can adjust the number of neighbors (n_neighbors) as needed
knn_model.fit(X_train, y_train)

import joblib
# Save the trained model to a file
joblib.dump(knn_model, 'weather_prediction_model.joblib')
```

Figure 4. KNN Modeling

Following training, the KNN model is saved to a file, 'weather_prediction_model.joblib', utilizing the joblib.dump function. This strategic storage facilitates the reuse of the trained model for subsequent predictions without the necessity for retraining. The choice of n_neighbors=100 is a hyperparameter that can be fine-tuned based on the specific characteristics of the dataset and the desired model performance. As a critical component of the weather prediction pipeline, the KNN modeling phase sets the stage for evaluating the model's accuracy and generalizability. Adjustments to hyperparameters, such as n_neighbors, can be made iteratively to optimize the model for robust predictions across various weather scenarios.

## Simulation

In this model simulation, the goal is to provide users with a hands-on experience exploring the functionality of a pre-trained K-Nearest Neighbors (KNN) model. By simulating weather predictions based on user-input values for temperature, humidity, and wind speed, this interactive tool offers a dynamic way to understand how environmental factors influence the model's forecasts. The use of IPython widgets enables users to actively participate in the simulation, adjusting sliders to simulate different weather conditions and observing real-time updates in predictions and confidence levels.



```
Temperature   25.00
Humidity      50.00
Wind Speed    10.00

Predicted Weather: Cloudy with Confidence Level: 0.77
```

```
Temperature   31.00
Humidity      39.00
Wind Speed    16.00

Predicted Weather: Sunny with Confidence Level: 1.00
```

```
Temperature   13.00
Humidity      86.00
Wind Speed    4.00

Predicted Weather: Rainy with Confidence Level: 1.00
```

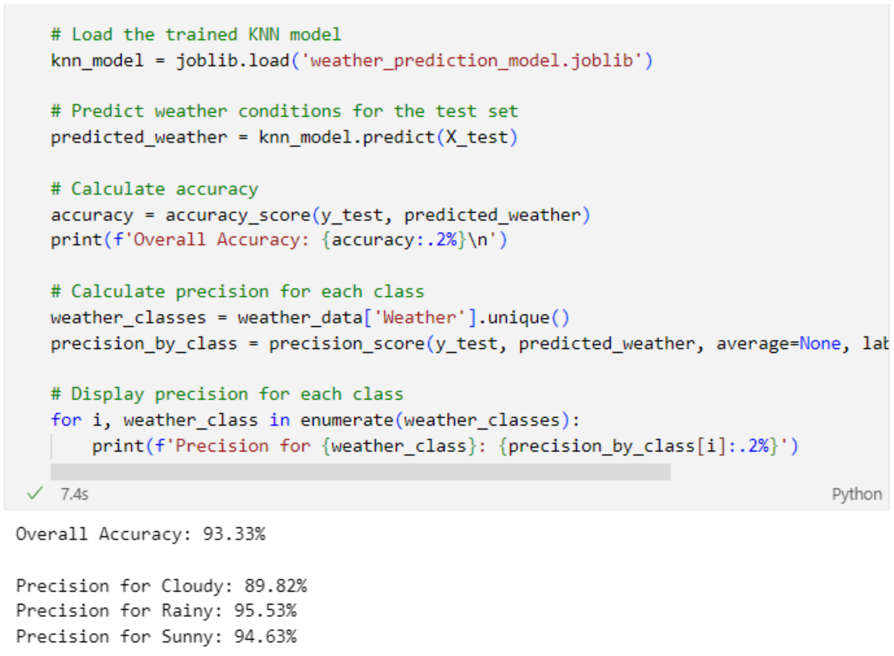
Figure 5. Model Simulation

The interactive weather prediction component features sliders for temperature, humidity, and wind speed, providing users with an intuitive means to explore the model's predictions. Each slider represents a specific weather parameter, allowing users to interactively adjust these values and observe the immediate impact on the model's forecast. As users move the sliders, the interact function dynamically triggers the prediction function, updating the forecast and associated visualizations in real-time. For example, increasing the temperature slider might result in a shift from predicting a 'Cloudy' to a 'Sunny' condition, reflecting the model's sensitivity to changes in temperature. This interactive adjustment of input parameters enables users to simulate different weather scenarios on-the-fly, fostering a hands-on understanding of how variations in temperature, humidity, and wind speed influence the model's predictions. The responsiveness of the component to slider adjustments enhances the user's ability to experiment with and grasp the intricate relationship between environmental factors and weather predictions.

## Evaluation and analysis

### *Model Evaluation*

The evaluation results showcase the robust performance of the weather prediction model on the test set. With an impressive overall accuracy of 93.33%, the model demonstrates its ability to make correct predictions across diverse weather conditions. Precision, which measures the accuracy of the model's predictions for each specific weather class, further accentuates its reliability.



```python
# Load the trained KNN model
knn_model = joblib.load('weather_prediction_model.joblib')

# Predict weather conditions for the test set
predicted_weather = knn_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, predicted_weather)
print(f'Overall Accuracy: {accuracy:.2%}\n')

# Calculate precision for each class
weather_classes = weather_data['Weather'].unique()
precision_by_class = precision_score(y_test, predicted_weather, average=None, lab

# Display precision for each class
for i, weather_class in enumerate(weather_classes):
    print(f'Precision for {weather_class}: {precision_by_class[i]:.2%}')
```

```
Overall Accuracy: 93.33%

Precision for Cloudy: 89.82%
Precision for Rainy: 95.53%
Precision for Sunny: 94.63%
```

Figure 6. Evaluation Results

For Cloudy conditions, the model exhibits a precision of 89.82%, indicating a high level of accuracy when identifying Cloudy weather. Similarly, the Rainy weather class attains a precision of 95.53%, reflecting the model's accuracy in predicting Rainy conditions. The Sunny class achieves a precision of 94.63%, underscoring the model's proficiency in making accurate predictions for Sunny weather. These precision values suggest that when the model predicts a specific weather condition, it is highly likely to be correct, with minimal instances of false positives. The strong overall accuracy, coupled with high precision for each weather class, reinforces the model's effectiveness in providing accurate and reliable weather predictions across a spectrum of scenarios. This performance evaluation instills confidence in the model's ability to generalize well to various weather conditions and underscores its utility for practical weather forecasting applications.

*Model analysis*

The Kernel Density Estimation (KDE) plots generated from the provided code offer a visual analysis of the relationship between actual and predicted weather conditions for key meteorological features—Temperature, Humidity, and Wind Speed. These plots serve as an insightful tool to evaluate the performance of the weather prediction model.

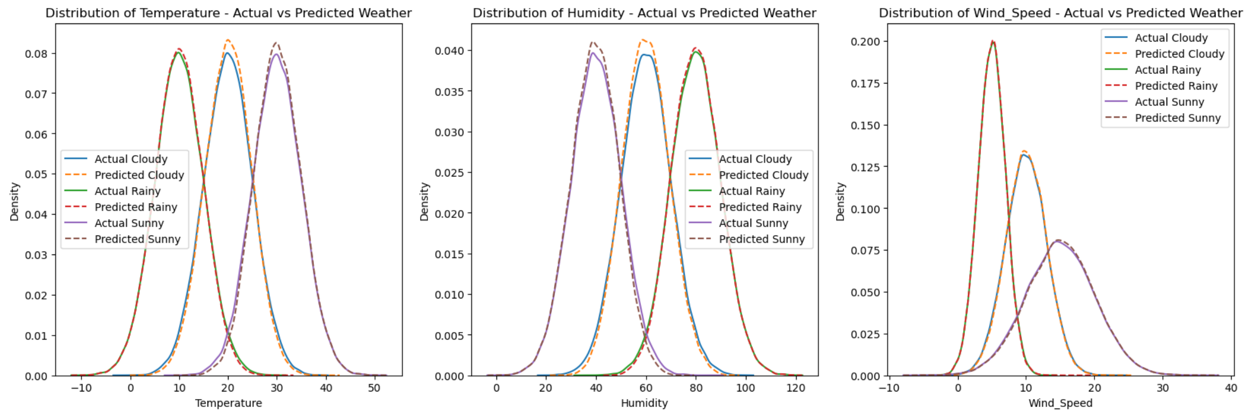


Figure 7. Distributions

The KDE plots vividly illustrate how well the model aligns with the true distribution of each numerical feature for various weather conditions. Discrepancies between the solid (actual) and dashed (predicted) lines in the plots may highlight areas where the model struggles or excels in capturing specific patterns. By examining these visualizations, one can gain valuable insights into the model's ability to reproduce the underlying relationships between numerical features and weather classes. Consistency between actual and predicted distributions signifies the model's effectiveness in learning and generalizing patterns from the training data to make accurate predictions on unseen test data. These visualizations serve as a powerful tool for model interpretation and performance assessment in the context of weather prediction.

## Conclusion

This project provides a thorough and systematic approach to weather prediction using a K-Nearest Neighbors (KNN) classification model. The project successfully covers crucial stages in the data science workflow, starting with the generation of synthetic data that mimics various weather conditions.

The exploratory data analysis (EDA) section illuminates the inherent patterns in the data, laying the groundwork for model training. The modeling and simulation phase showcase the implementation of a KNN classifier, with an emphasis on effective data splitting, training, and model persistence. The evaluation metrics, including overall accuracy and precision for each weather class, offer a comprehensive assessment of the model's predictive capabilities. Visualizations play a pivotal role, allowing for an intuitive comparison of actual and predicted distributions of numerical features across different weather classes.

The addition of an interactive prediction component enhances the practicality of the model, making it user-friendly and accessible. This functionality extends the applicability of the model beyond the notebook environment, encouraging users to interact with and derive predictions based on their input. In summary, this notebook not only serves as a tutorial for weather prediction but also emphasizes the importance of hands-on experience in data science. It provides a solid foundation for further exploration and application of machine learning concepts. As a parting thought, the interactive prediction component invites users to engage with the model, fostering a deeper understanding of its behavior and potentially inspiring future enhancements or applications.