Presentation by **Iman Aidi Elham**

MICROCLIMATE DATA ANALYSIS AT CULTURAL HERITAGE SITES USING THE RANDOM FOREST AND XGBOOST ALGORITHMS

Supervised by Assoc. Prof. Dr. Shahizan bin Othman

A20EC5006 ✦ Universiti Teknologi Malaysia ✦ 2024

TABLE OF CONTENTS

Introduction

Problem Background

Objectives

Scopes

Methodology

Literature Review

Experiment Setup & Design

Conclusion

INTRODUCTION

Cultural heritage sites are the basis for our global and historical values. They connect us to the traditions left by our ancestors and contribute significantly to the cultural identity of human society (Lombardo et al., 2020).

The preservation of cultural heritage, whether it be buildings or artifacts, is subject to various risks of damage and deterioration that result from microclimate conditions in the surrounding environment. These conditions are determined by several factors, including microclimate parameters such as temperature, humidity, airborne pollutants concentrations, air speed, and others (Fabbri & Bonora, 2021).

PROBLEM BACKGROUND

- This country is experiencing frequent climate fluctuations.
- These fluctuations are negatively impacting the aesthetic appeal of heritage sites.
- The tourism industry and local economy are being significantly affected.
- Microclimate changes are causing damage to cultural heritage sites and monuments.
- Balancing consumption and conservation strategies is becoming challenging.
- Preserving cultural heritage and promoting sustainable tourism is a priority.
- This supports cultural heritage tourism and the well-being of communities.

OBJECTIVES

Objective 1

To identify and compare the most suitable machine learning algorithms for analyzing microclimate data in cultural heritage sites.

Objective 2

To evaluate the performance and accuracy of the developed machine learning models in predicting microclimate conditions.

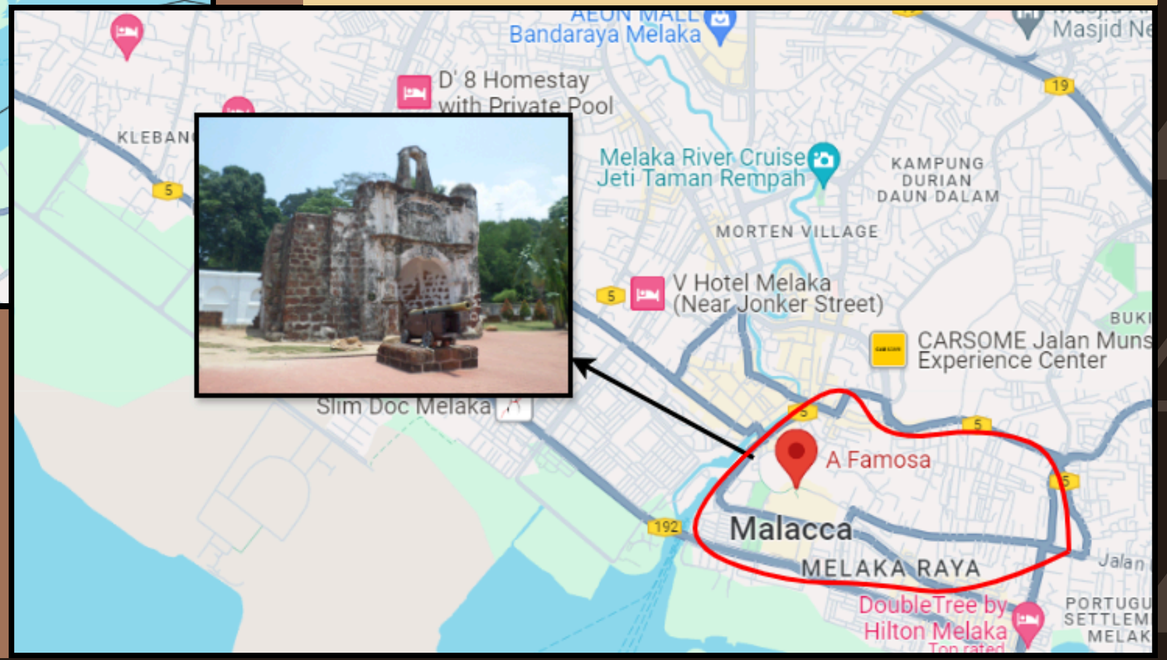
Objective 3

To design and develop a user-friendly dashboard for displaying microclimate data trends and predictions.

STUDY AREA

Sultan Ibrahim Building, Johor Bahru

A Famosa, Melaka

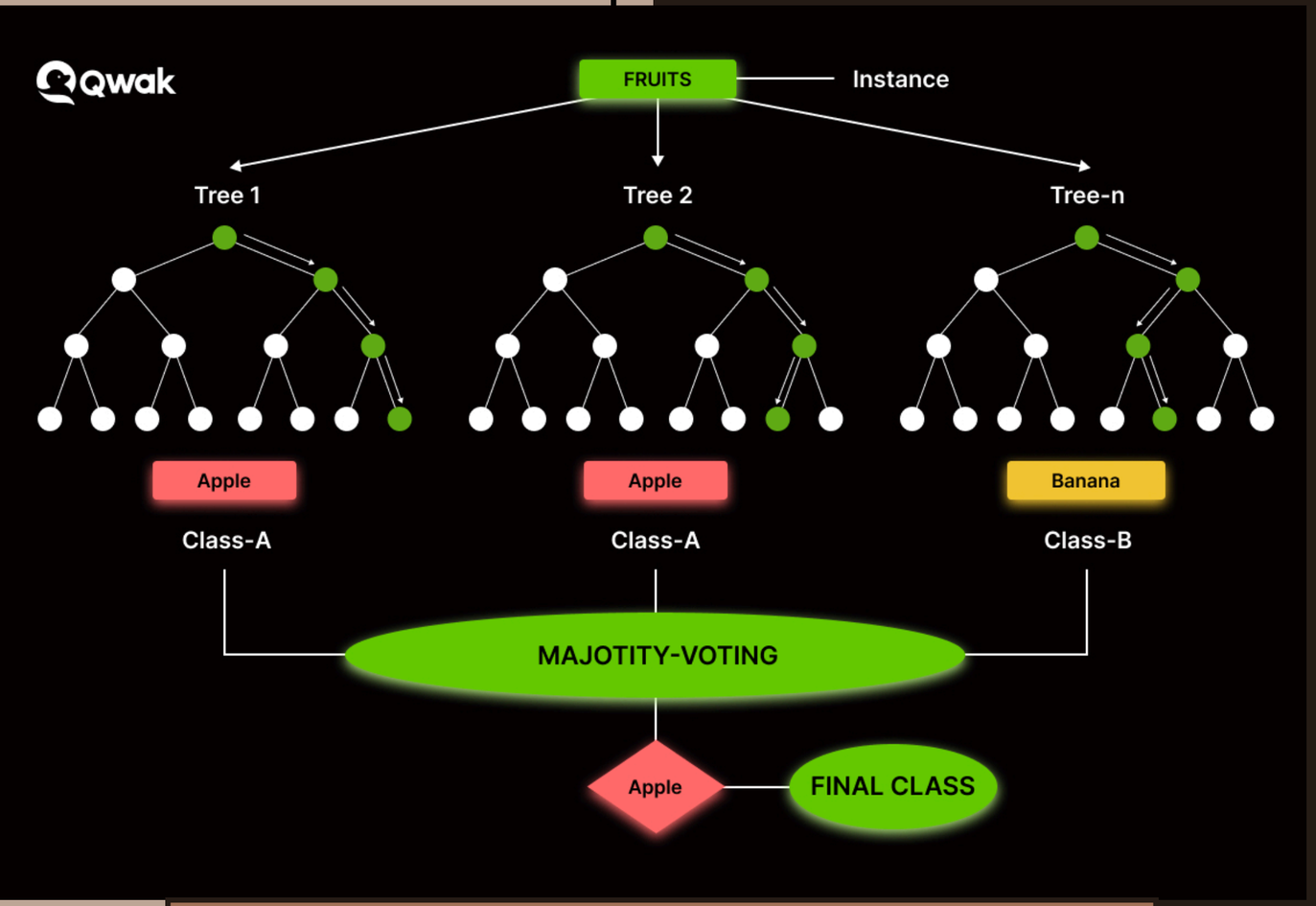


METHODOLOGY

Phase 1	LITERATURE REVIEW
Phase 2	DATA REQUIREMENT & DATA COLLECTION
Phase 3	MACHINE LEARNING MODEL DEVELOPMENT
Phase 4	DASHBOARD DEVELOPMENT

LITERATURE REVIEW

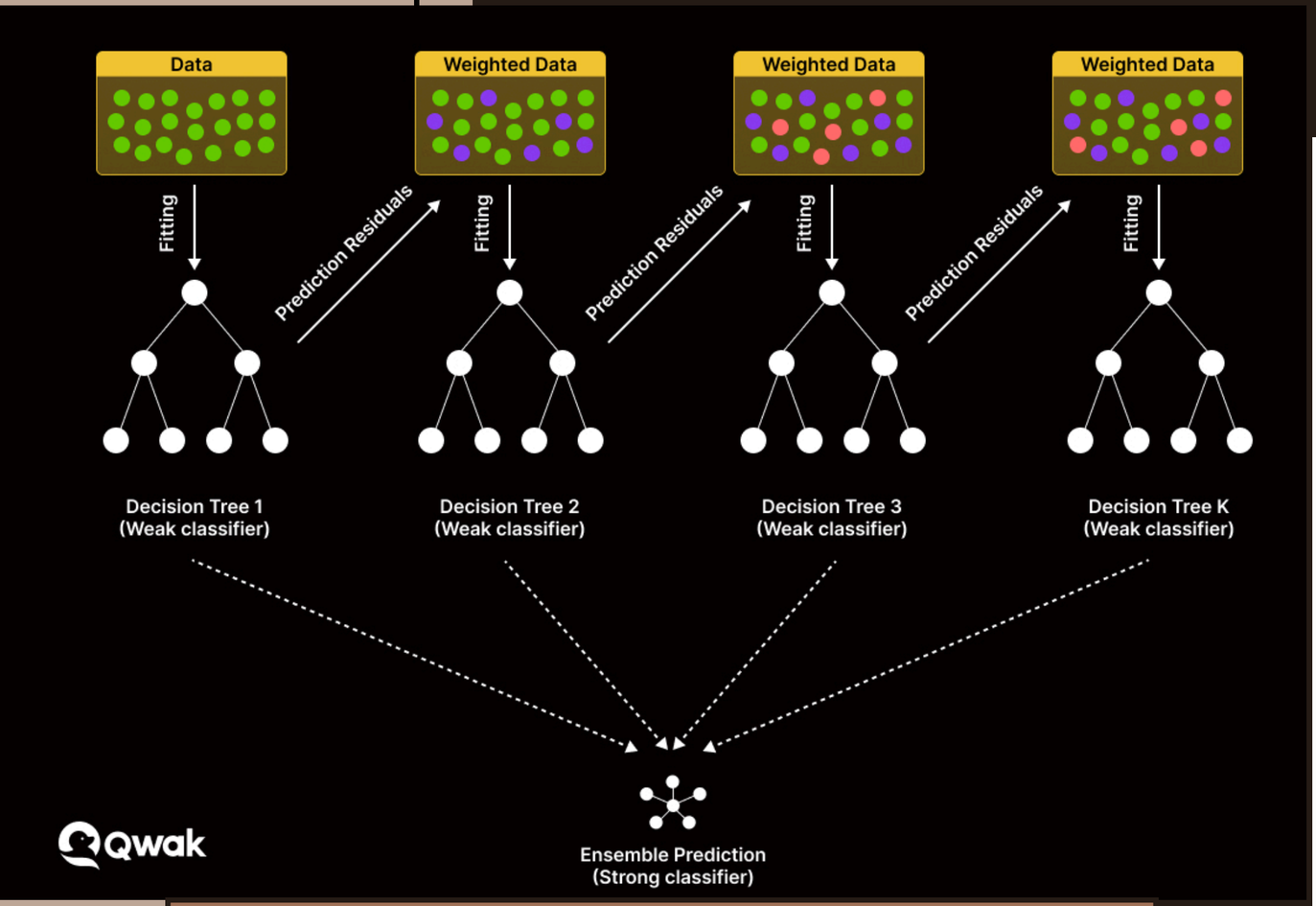
RANDOM FOREST



Random Forest Model Architecture

- Random Forest is a popular and versatile ensemble learning algorithm introduced in 2001.
- Can be used for tasks like classification, regression, clustering, interaction detection, and variable selection.
- Uses decision trees and aggregates their predictions to make final classifications.
- Good at handling complex datasets with high dimensionality, noise, and missing data.
- Prevents overfitting and provides robust and high-performance results.
- The algorithm can handle various types of input variables and deal with missing data.
- It can determine the importance of input variables for predictions.
- Random Forest can identify and model interactions between variables, which is useful for microclimate monitoring at cultural heritage sites.

XGBOOST

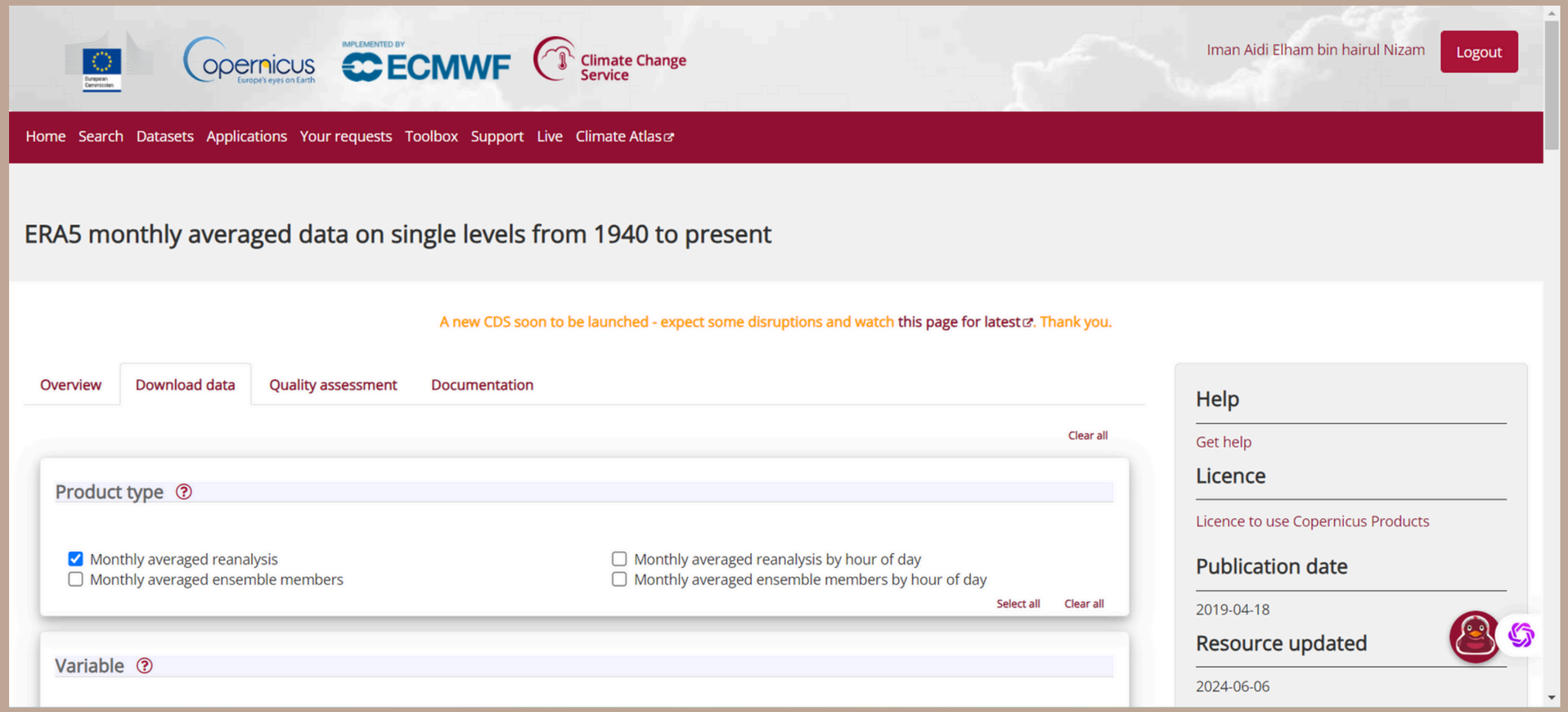


XGBoost Model Architecture

- XGBoost is a popular machine learning algorithm used for classification and regression tasks.
- Combines multiple weak learners to create a strong model through boosting.
- Handles missing data effectively using surrogate splits.
- Optimized for parallel computing, making it fast and efficient.
- Outperforms other algorithms in terms of accuracy and efficiency.
- Scalable and ideal for handling large volumes of data.
- Enables real-time monitoring and provides insights into environmental conditions.
- Optimized for feature selection to identify influential variables.

DATA REQUIREMENT & DATA COLLECTION

DATA PROVIDER



COPERNICUS CLIMATE DATA STORE (CDS)

PARAMETERS

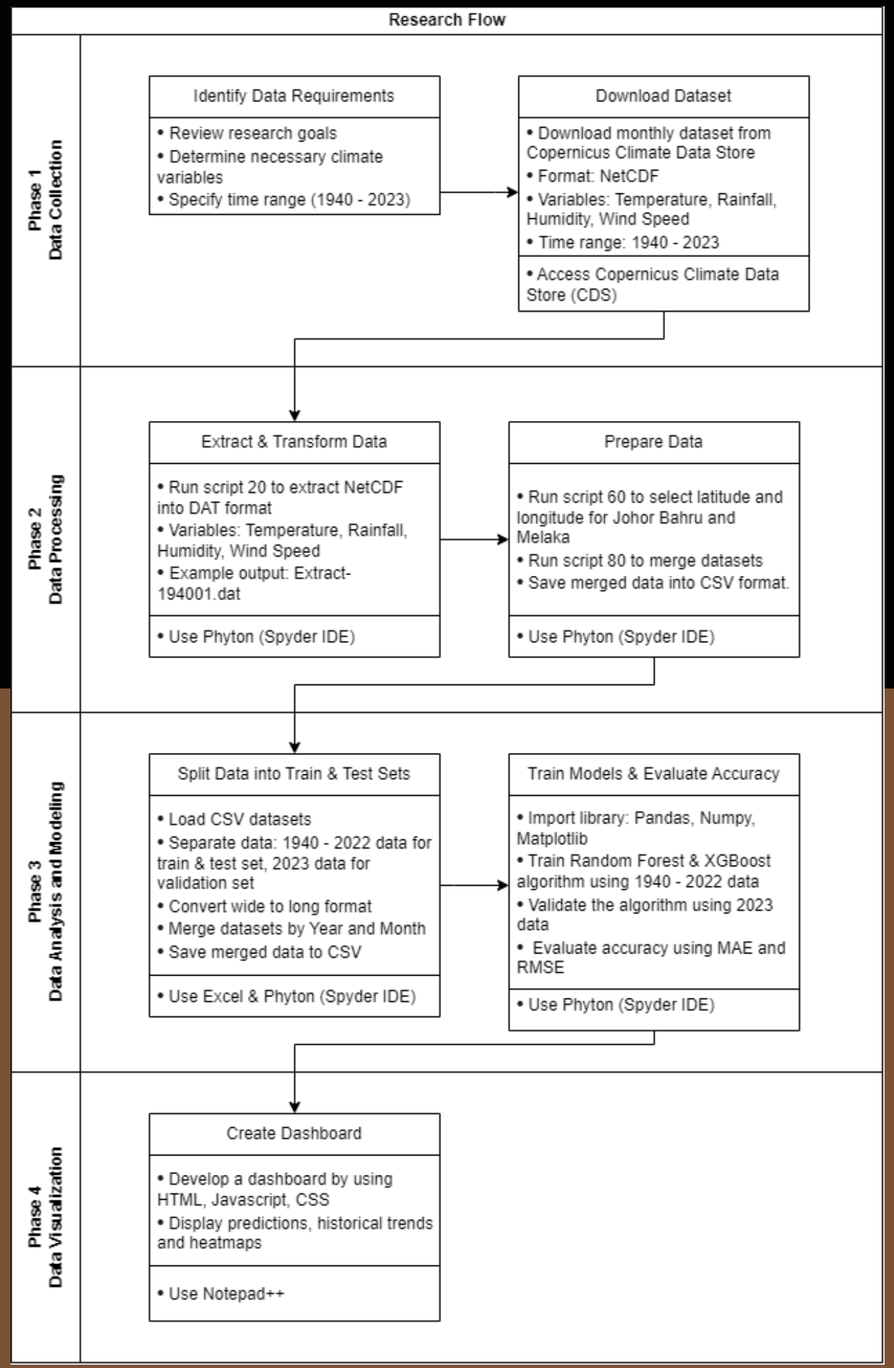
1	**RAINFALL**	Measured in mm, crucial for moisture-related risks
2	**HUMIDITY**	Expressed as a percentage, impacts material preservation
3	**TEMPERATURE**	Recorded in °C, can cause thermal stress
4	**WIND SPEED**	Measured in m/s, influences erosion and pollutant deposition

DATA SPAN

from **1940** — **Monthly records** — to **2023**

EXPERIMENT DESIGN AND SETUP

EXPERIMENT PHASE AND FLOW



```
data_raw='../Data-RAW-RAIN/'

# files_month=['Jun']
files_month=['Jan','Feb','Mac','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov

for m in files_month:
    filename='Monthly-Precip-'+m+'-1940-2023.nc'
    f=nc.Dataset(data_raw+filename)
    v=f.variables; keys=f.variables.keys(); data={}

    for i in keys:
        data[i]=np.squeeze(v[i][:])
        print(i)

    lat=data['latitude']
    lon=data['longitude']
    rain=data['tp'][:]

    times = f.variables['time'][:]
    units=f.variables['time'].units
    ptime = num2date (times[:], units, calendar='gregorian')
    print ('Available Time Observation to Plot : (index,pressure) ')
```

Data Extraction and Processing

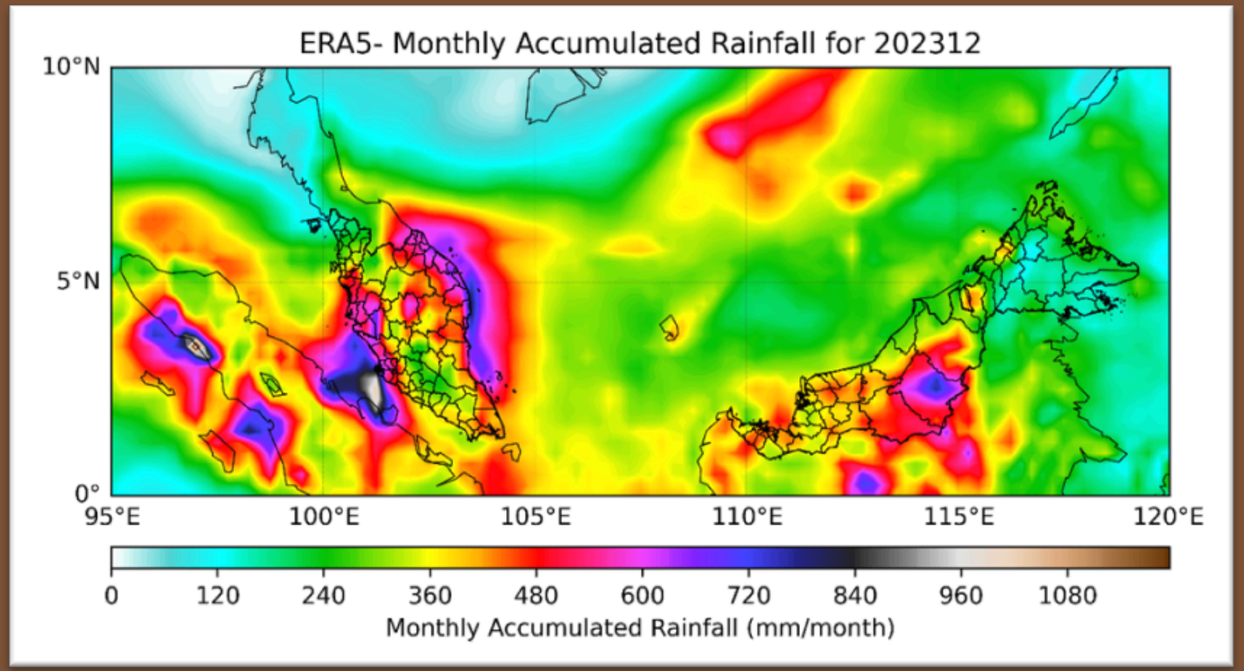
10.000	90.000	26.499
10.000	90.250	26.789
10.000	90.500	26.062
10.000	90.750	24.172
10.000	91.000	21.628
10.000	91.250	19.883
10.000	91.500	18.938
10.000	91.750	17.920
10.000	92.000	18.684
10.000	92.250	17.811
10.000	92.500	17.157
10.000	92.750	16.030
10.000	93.000	14.976
10.000	93.250	13.958

Processed Dataset in DAT file format

```
93      # Define colormap and contour levels
94      cmap = cm.s3pcpn_l
95      clevprecip = np.arange(0, 1200, 50)
96      norm1 = mpl.colors.BoundaryNorm(clevprecip, cmap.N)
97
98      # Create the contour plot
99      cf = m.contourf(X, Y, Z, clevprecip, cmap=cmap, norm=norm1, latlon=True)
100     cbar = m.colorbar(cf, location='bottom', pad="12%")
101     cbar.ax.tick_params(labelsize=10)
102     cbar.set_label('Monthly Accumulated Rainfall (mm/month)', fontsize=10)
103
104     # Add title to the plot
105     title1 = 'ERA5- Monthly Accumulated Rainfall for ' + nama_file
106     plt.title(title1)
107
108     # Save the plot as a PNG file
109     plt.savefig(path3 + nama_file + '.png', dpi=500, bbox_inches='tight')
```

Plotting Monthly Average Accumulated Data

Monthly Accumulated Data Sample Output



```
# Extract latitude, longitude, and rainfall rate
lat = data[:, 0]
lon = data[:, 1]
rain_rate = data[:, 2]

# Define the target latitude and longitude (location to select data for)
in_lats1 = [q]
in_lons1 = [r]

# Find the closest data point to the target location
ind = []
for i in range(1):
    dist = (lat - in_lats1[i])**2 + (lon - in_lons1[i])**2
    ind.append(np.where(dist == np.min(dist))[0][0])

    lat2 = lat[ind]
    lon2 = lon[ind]
    rain_rate2 = rain_rate[ind]

# Combine the date, target location, and selected data into an array
data3 = [np.array([dates]), in_lats1, in_lons1, lat2, lon2, rain_rate2]
data3 = np.transpose(data3)
```

Extracting Location-Specific Data from Data Files

```
# Load the data for each month of the year
data1 = np.loadtxt(path3 + 'Data-Location-' + year + '01' + '.dat', dtype='float')
data2 = np.loadtxt(path3 + 'Data-Location-' + year + '02' + '.dat', dtype='float')
data3 = np.loadtxt(path3 + 'Data-Location-' + year + '03' + '.dat', dtype='float')
data4 = np.loadtxt(path3 + 'Data-Location-' + year + '04' + '.dat', dtype='float')
data5 = np.loadtxt(path3 + 'Data-Location-' + year + '05' + '.dat', dtype='float')
data6 = np.loadtxt(path3 + 'Data-Location-' + year + '06' + '.dat', dtype='float')
data7 = np.loadtxt(path3 + 'Data-Location-' + year + '07' + '.dat', dtype='float')
data8 = np.loadtxt(path3 + 'Data-Location-' + year + '08' + '.dat', dtype='float')
data9 = np.loadtxt(path3 + 'Data-Location-' + year + '09' + '.dat', dtype='float')
data10 = np.loadtxt(path3 + 'Data-Location-' + year + '10' + '.dat', dtype='float')
data11 = np.loadtxt(path3 + 'Data-Location-' + year + '11' + '.dat', dtype='float')
data12 = np.loadtxt(path3 + 'Data-Location-' + year + '12' + '.dat', dtype='float')

# Extract rainfall data from each month's data
rain1 = data1[5]
rain2 = data2[5]
rain3 = data3[5]
rain4 = data4[5]
rain5 = data5[5]
rain6 = data6[5]
rain7 = data7[5]
rain8 = data8[5]
rain9 = data9[5]
rain10 = data10[5]
rain11 = data11[5]
rain12 = data12[5]
```

Merging and Consolidating Location-Specific Microclimate Data

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2	1940	24.549	25.042	26.119	25.932	26.282	26.482	26.675	26.337	25.94	25.828	25.25	24.782
3	1941	24.982	25.426	25.895	26.328	26.235	26.62	26.15	26.18	25.448	25.553	25.39	25.116
4	1942	24.564	25.072	25.609	25.785	26.501	26.454	25.768	25.865	25.575	25.617	24.99	24.281
5	1943	24.674	25.025	25.299	25.763	26.428	26.838	26.198	25.878	25.512	25.234	25.021	24.619
6	1944	24.499	25.281	25.693	25.856	25.845	26.13	26.082	26.211	25.835	25.918	25.385	25.165
7	1945	24.75	24.888	24.575	25.675	26.159	26.177	25.934	25.729	26.139	25.651	24.93	24.874
8	1946	24.578	24.238	25.653	25.76	26.182	26.382	26.264	26.14	25.96	25.579	25.425	25.128
9	1947	25.069	25.25	25.737	26.157	26.188	26.277	25.837	25.643	25.456	25.6	25.367	24.638
10	1948	24.486	25.178	26.134	26.634	26.367	26.221	26.012	25.999	25.904	25.96	25.224	25.188
11	1949	25.35	25.665	26.729	27.245	27.275	26.793	26.201	25.969	25.78	26.288	25.854	25.336
12	1950	25.803	25.858	26.145	26.334	26.5	26.408	25.953	25.552	26.118	25.862	25.375	25.13
13	1951	24.79	25.221	25.679	26.427	26.419	26.408	25.746	26.243	26.124	26.163	26.233	25.664
14	1952	25.483	25.852	26.28	26.399	26.834	26.506	26.125	26.175	26.116	26.287	25.62	25.329
15	1953	25.184	25.311	25.88	26.542	26.496	26.491	25.781	26.269	25.927	26.012	26.095	25.619
16	1954	25.32	25.376	25.865	26.142	26.323	26.299	25.771	26.012	25.967	25.514	25.268	24.897
17	1955	24.515	25.534	25.858	26.124	26.938	26.585	26.05	25.678	25.692	25.949	26.044	24.779
18	1956	24.589	25.348	25.887	26.361	26.446	26.386	26.106	25.863	25.724	25.429	25.352	24.983
19	1957	24.906	25.485	25.863	26.233	26.265	26.7	26.355	26.282	26.209	25.993	25.791	25.211
20	1958	25.808	25.669	26.322	26.834	27.036	26.993	26.97	26.223	26.481	26.211	25.74	25.484
21	1959	25.218	25.838	25.997	26.354	26.768	26.522	26.429	26.18	26.426	26.233	25.706	25.627
22	1960	25.276	25.376	26.07	26.524	27.058	26.595	26.161	26.454	26.065	26.193	25.903	25.413
23	1961	25.18	25.685	26.466	26.644	27.087	26.525	26.123	26.131	26.145	26.405	25.721	25.214
24	1962	24.946	25.165	25.825	26.338	27.072	26.48	26.469	25.608	26.128	26.214	25.726	25.345
25	1963	24.696	24.651	25.892	26.749	27.03	26.754	26.213	26.22	26.26	25.93	25.847	25.401
26	1964	25.702	25.496	26.054	26.722	27.049	26.449	25.669	26.185	26.135	25.873	25.709	24.398

Data-Merge-Sultan Ibrahim Build

Merged Accumulated Data

```
    # Apply month mapping
    temperature_melted['Month_Num'] = temperature_melted['Month'].map(month_mapping)
    humidity_melted['Month_Num'] = humidity_melted['Month'].map(month_mapping)
    rainfall_melted['Month_Num'] = rainfall_melted['Month'].map(month_mapping)
    wind_speed_melted['Month_Num'] = wind_speed_melted['Month'].map(month_mapping)

    # Merge all the dataframes on Year and Month_Num
    merged_data = pd.merge(temperature_melted, humidity_melted, on=['Year', 'Month', 'Month_Num'])
    merged_data = pd.merge(merged_data, rainfall_melted, on=['Year', 'Month', 'Month_Num'])
    merged_data = pd.merge(merged_data, wind_speed_melted, on=['Year', 'Month', 'Month_Num'])

    # Drop the 'Month_Num' column as it is no longer needed
    merged_data = merged_data.drop(columns=['Month_Num'])

    # Create historical data excluding 2023 and 2024
    historical_data = merged_data[(merged_data['Year'] < 2023)]

    # Create actual data for 2023
    actual_data_2023 = merged_data[(merged_data['Year'] == 2023)]

    # Save to CSV files
    historical_data.to_csv(historical_output_file, index=False)
    actual_data_2023.to_csv(actual_output_file, index=False)

# Process data for JB
process_and_save_data(
    'temperature_data_jb.csv',
    'humidity_data_jb.csv',
    'rainfall_data_jb.csv',
    'wind_data_jb.csv',
    'historical_data_jb_2022.csv',
    'actual_data_jb_2023.csv'
```

Split the pre-processed data into training and testing sets

Read and Preprocess Data

```python
# Function to read and preprocess data
def read_and_preprocess_data(data_file, exclude_year=None):
    data = pd.read_csv(data_file)

    month_mapping = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4,
        'May': 5, 'June': 6, 'July': 7, 'August': 8,
        'September': 9, 'October': 10, 'November': 11, 'December': 12,
        'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
        'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
        'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12
    }

    data['Month_Num'] = data['Month'].map(month_mapping)

    if exclude_year is not None:
        data = data[data['Year'] != exclude_year]

    return data
```

Feature Engineering

```python
# Function for feature engineering
def prepare_data(data, variable):
    # Create lagged features
    for var in ['Temperature', 'Humidity', 'Rainfall', 'Wind Speed']:
        data[f'{var}_lag1'] = data.groupby('Year')[var].shift(1)
        data[f'{var}_lag2'] = data.groupby('Year')[var].shift(2)

    # Create rolling averages
    for var in ['Temperature', 'Humidity', 'Rainfall', 'Wind Speed']:
        data[f'{var}_rolling3'] = data.groupby('Year')[var].rolling(window=3, min_periods=1).

    # Create seasonality features
    data['month_sin'] = np.sin(2 * np.pi * data['Month_Num']/12)
    data['month_cos'] = np.cos(2 * np.pi * data['Month_Num']/12)

    # Create interaction terms
    data['temp_humid'] = data['Temperature'] * data['Humidity']
```

Data Quality Check

```python
# Function to check data quality
def check_data_quality(X, y):
    print("Checking for NaN or infinity values:")
    print("X contains NaN:", X.isna().any().any())
    print("y contains NaN:", y.isna().any())
    print("X contains infinity:", np.isinf(X).any().any())
    print("y contains infinity:", np.isinf(y).any())
```

Train Random Forest and XGBoost

```python
def train_model(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }

    rf = RandomForestRegressor(random_state=42)

    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                               cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')

    grid_search.fit(X_train, y_train)

    print("Best parameters for Random Forest:", grid_search.best_params_)
    return grid_search.best_estimator_

# Function to train XGBoost models with hyperparameter tuning
def train_xgboost_model(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.3],
        'max_depth': [3, 5, 7],
        'min_child_weight': [1, 3, 5]
    }

    xgb = XGBRegressor(random_state=42)

    grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,
                               cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')

    grid_search.fit(X_train, y_train)

    print("Best parameters for XGBoost:", grid_search.best_params_)
    return grid_search.best_estimator
```

Evaluate the Accuracy of Random Forest and XGBoost

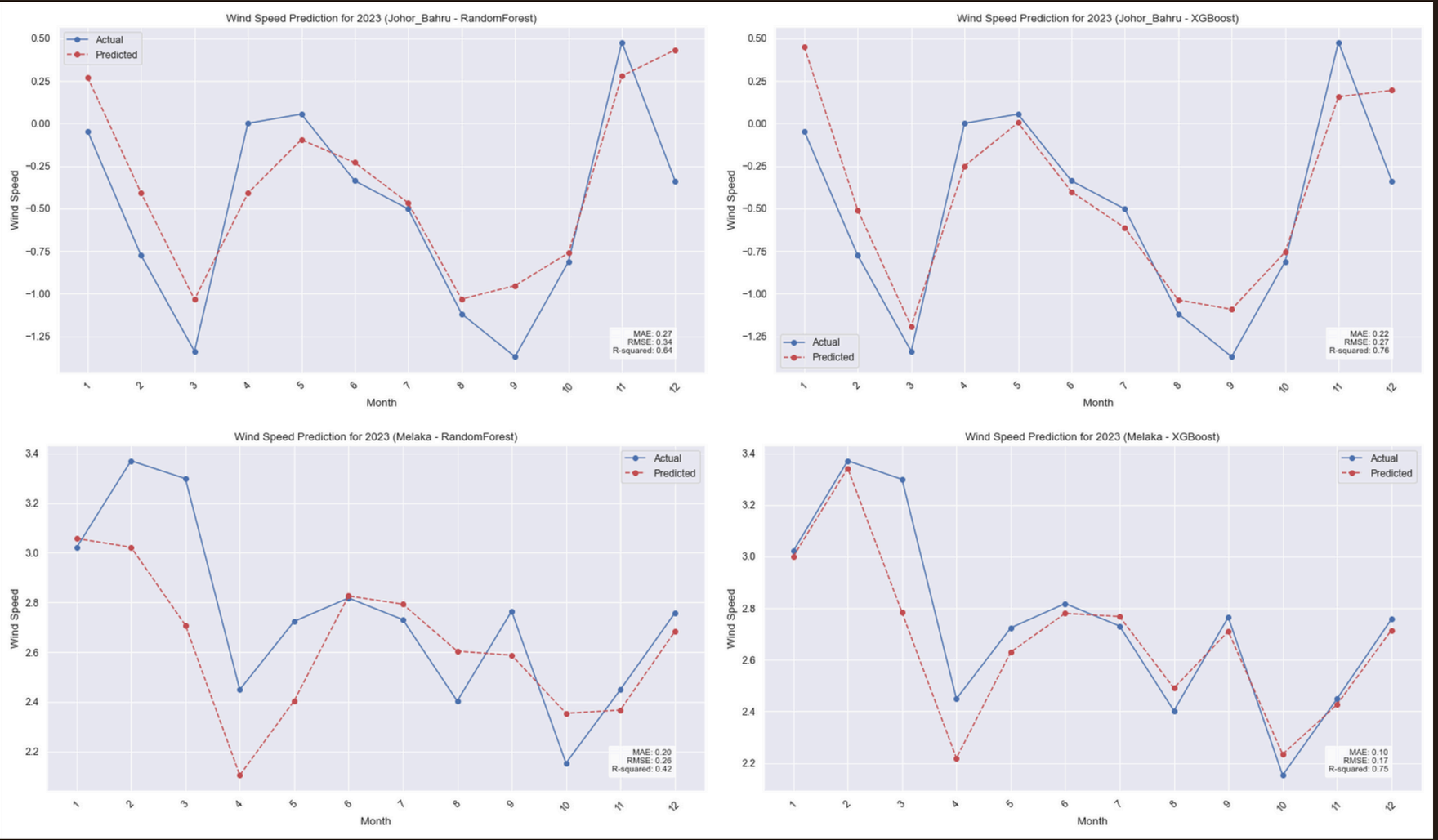
```python
# Function to evaluate predictions against actual data
def evaluate(predictions, actual_data):
    mae = mean_absolute_error(actual_data, predictions)
    rmse = np.sqrt(mean_squared_error(actual_data, predictions))
    r2 = r2_score(actual_data, predictions)

    metrics = {'MAE': mae, 'RMSE': rmse, 'R-squared': r2}
    return metrics

# Function to evaluate with cross-validation
def evaluate_with_cv(X, y, model, n_splits=5):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    metrics = {'MAE': [], 'RMSE': [], 'R-squared': []}

    for train_index, test_index in tscv.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

        fold_metrics = evaluate(predictions, y_test)
        for key in metrics:
            metrics[key].append(fold_metrics[key])

    # Average the metrics across folds
    return {key: np.mean(values) for key, values in metrics.items()}
```

Create plots and accuracy results

```python
# New function to create combined plots
def create_combined_plots(predictions, actual_data, months, variables, location, models, save_dir, metrics
    fig = plt.figure(figsize=(20, 20))
    gs = gridspec.GridSpec(4, 2, figure=fig)

    for i, variable in enumerate(variables):
        for j, model in enumerate(models):
            ax = fig.add_subplot(gs[i, j])

            ax.plot(months, actual_data[variable], marker='o', linestyle='-', color='b', label='Actual')
            ax.plot(months, predictions[f'{variable}_{model}'], marker='o', linestyle='--', color='r', lab

            ax.set_xlabel('Month')
            ax.set_ylabel(variable)
            ax.set_title(f'{variable} - {model}')
            ax.set_xticks(range(1, 13))
            ax.set_xticklabels([str(month) for month in range(1, 13)], rotation=45)
            ax.legend()
            ax.grid(True)
```

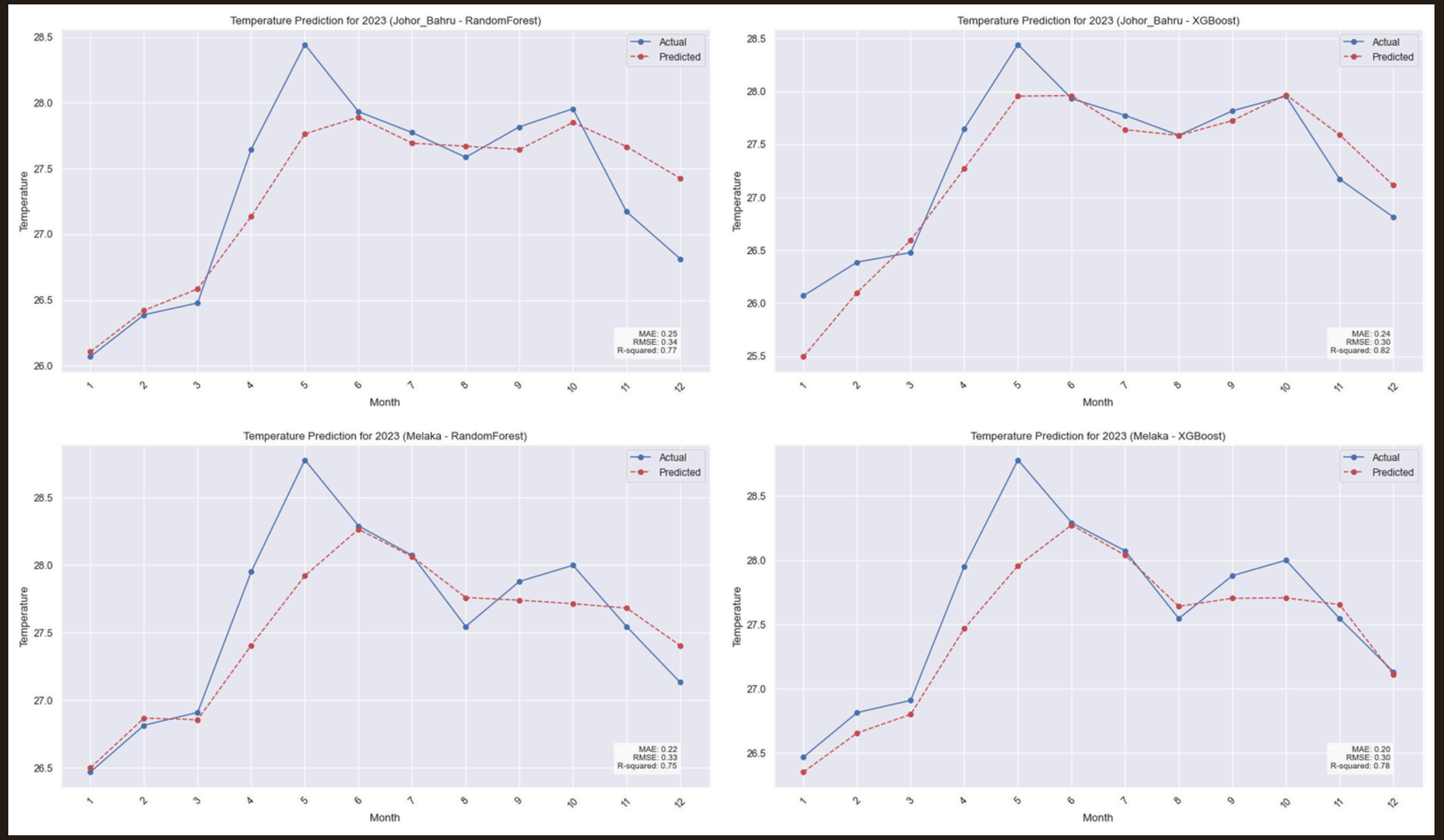
RESULT ANALYSIS

WIND SPEED



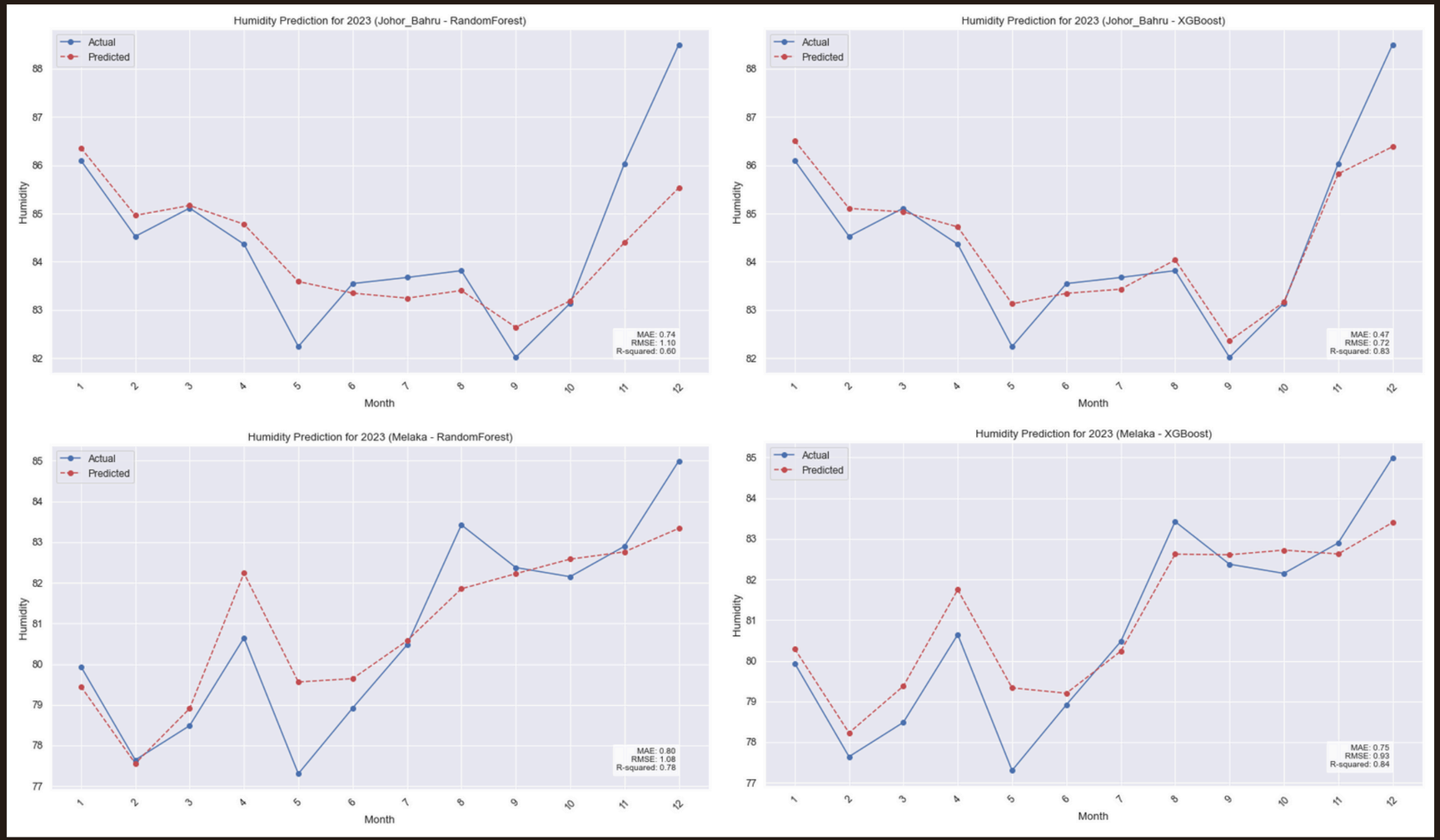
The figure presents wind speed predictions for Johor Bahru and Melaka in 2023, comparing Random Forest and XGBoost models. Both algorithms capture general wind speed trends throughout the year, with XGBoost showing slightly better performance metrics. Melaka consistently experiences higher wind speeds than Johor Bahru. While the models generally track seasonal patterns, there are notable discrepancies between actual and predicted values, particularly during peak wind periods.

TEMPERATURE



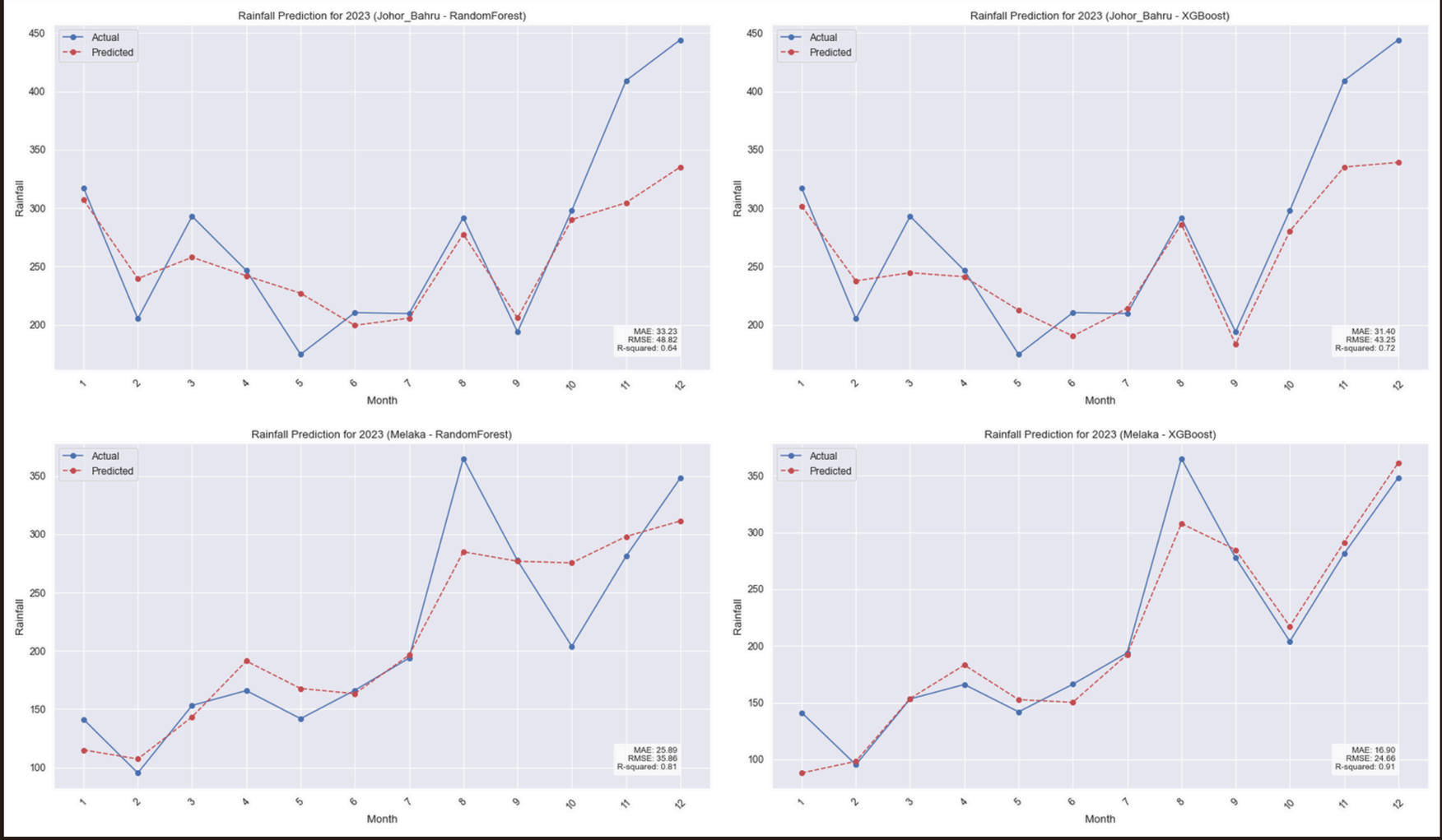
The figure illustrates temperature predictions for the same locations using the same models. Both algorithms effectively capture overall temperature patterns, with predictions appearing more accurate for Melaka than Johor Bahru. A clear temperature peak is observed around May for both areas. Interestingly, XGBoost demonstrates marginally better performance for Melaka, while Random Forest shows a slight edge in Johor Bahru predictions.

HUMIDITY



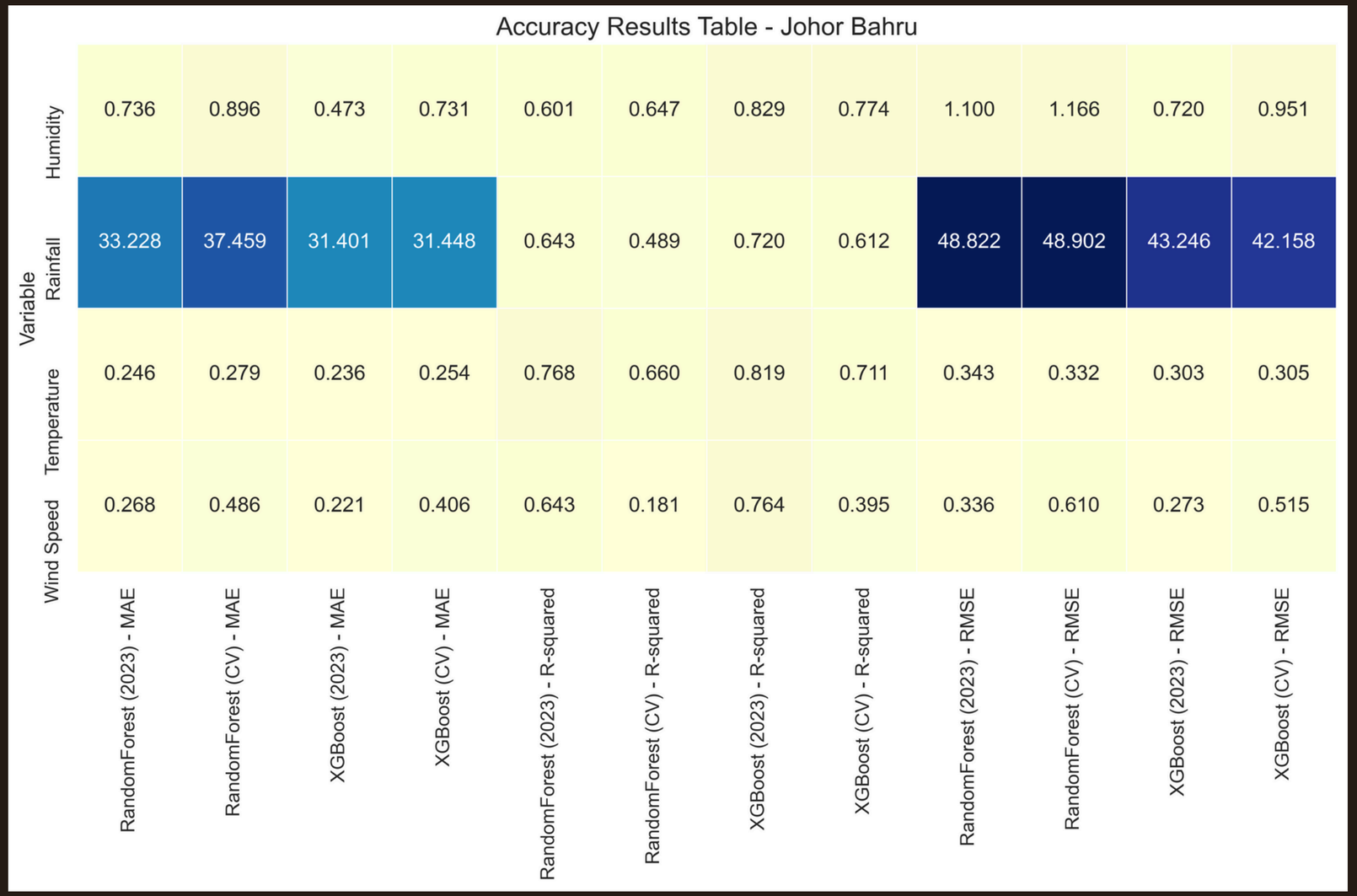
The figure depicts humidity predictions for 2023 in Johor Bahru and Melaka. The graphs reveal significant differences in humidity patterns between the two locations, with Johor Bahru exhibiting higher overall levels. Both models struggle to accurately predict extreme humidity fluctuations, especially in Johor Bahru. Generally, the XGBoost model outperforms Random Forest in terms of prediction accuracy for humidity across both locations.

RAINFALL



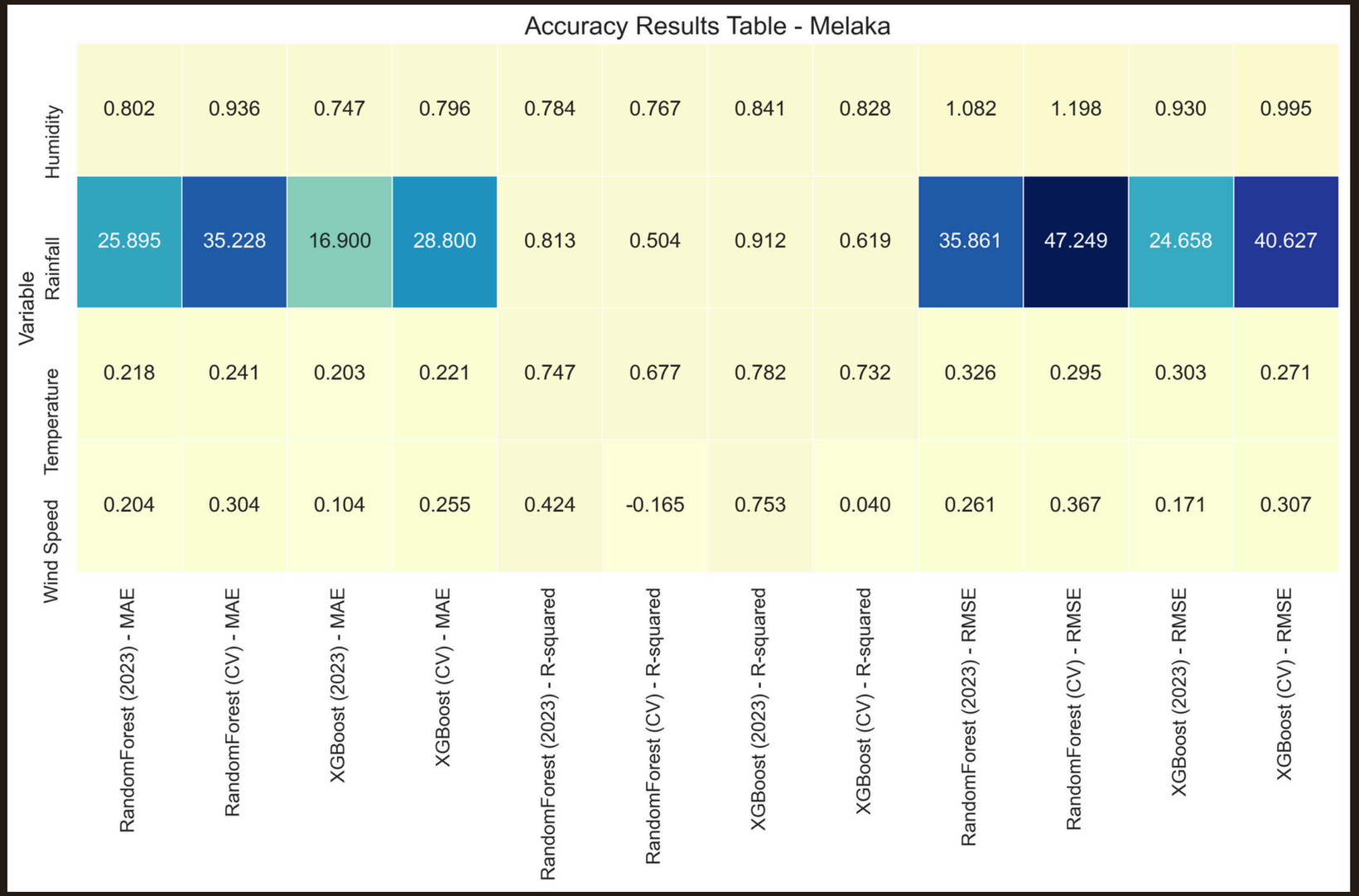
The figure shows rainfall predictions for both cities in 2023. The data indicates high variability in rainfall patterns throughout the year, with both models struggling to accurately predict extreme events, particularly towards year-end. Johor Bahru typically experiences higher rainfall compared to Melaka. In terms of model performance, XGBoost shows a slight advantage in predicting Melaka's rainfall, while Random Forest performs marginally better for Johor Bahru.

ACCURACY - JB



- RF and XGBoost models exhibited varied performance across different microclimate variables.
- For humidity, XGBoost demonstrated superior accuracy with a MAE of 0.473 for 2023 and 0.731 for cross-validation (CV), compared to RF's MAE of 0.736 and 0.896, respectively.The R-squared values also favored XGBoost, indicating a better fit for the data.
- Both models struggled with rainfall, showing higher MAE and lower R-squared values, suggesting challenges in capturing the variability of rainfall patterns accurately.
- Temperature predictions were more successful for both models, with low MAE and RMSE values indicating precise forecasting abilities.
- Wind speed predictions showed moderate performance, with both models achieving reasonable MAE and RMSE values.

ACCURACY - MELAKA



- XGBoost consistently outperformed Random Forest across most metrics.
- XGBoost showed lower MAE values for humidity, rainfall, temperature, and wind speed predictions compared to RF.
- MAE and higher R-squared values for rainfall prediction with XGBoost, indicating its superior accuracy in capturing the complex dynamics of rainfall patterns in Melaka.
- Potential issues in model fit for wind speed predictions, as suggested by slightly negative R-squared values for XGBoost in cross-validation.
- Overall, both models demonstrated the potential of machine learning in accurately predicting microclimate variables,

DASHBOARD DEVELOPMENT

DASHBOARD HOMEPAGE

Welcome to the Microclimate Monitoring & Prediction Dashboard

Select a Region

Melaka

Johor Bahru

DASHBOARD - JOHOR BAHRU

Johor Bahru 1.4560° N, 103.7602° E — Tue, July 2, 2024 at 04:14:59 PM — Go to Map — Change Site

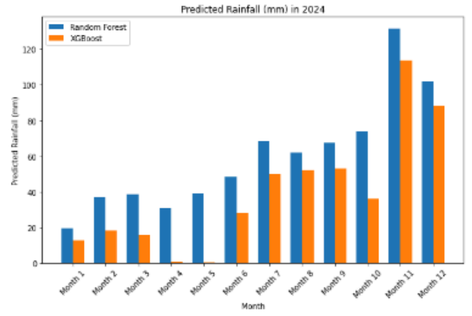
Select Weather Parameter

Rainfall | Temperature | Relative Humidity | Wind Speed

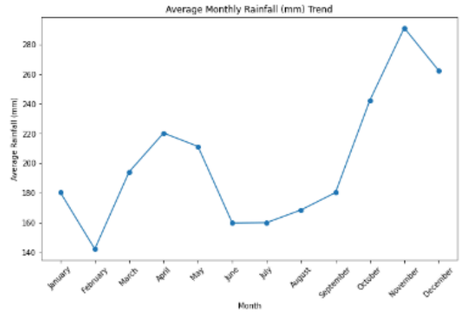
Live Weather

Weather: broken clouds | Temperature: 30.94°C | Humidity: 70% | Wind Speed: 3.6 m/s

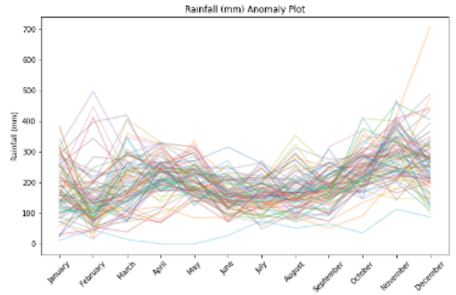
Predicted Values for 2024



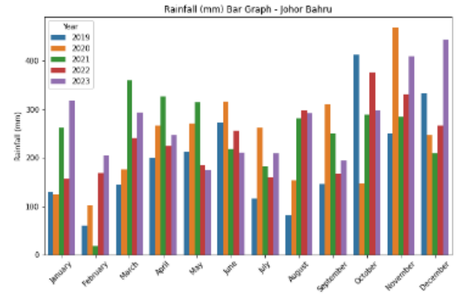
Average Monthly Trend 1940 - 2023



Anomaly Plot 1940 - 2023

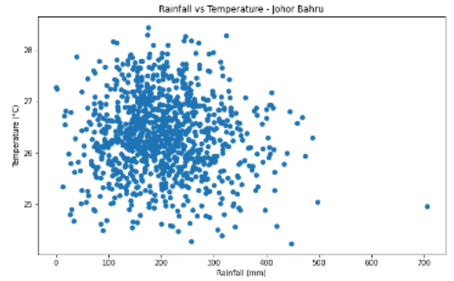


Last 5 Years Trend 2019 - 2023

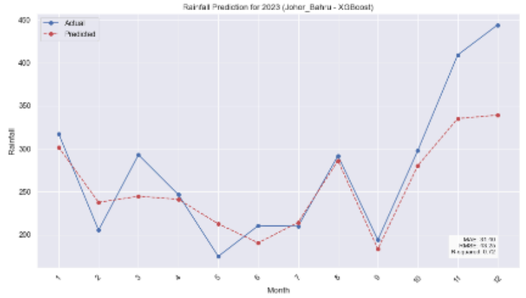


Correlation Matrix

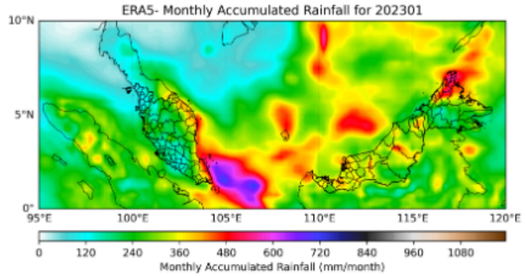
Scatter Plots 1940 - 2023



Actual vs Predicted 2023



Monthly Heatmaps



DASHBOARD - MELAKA

MELAKA 2.1918° N, 102.2504° E

Wed, July 3, 2024 at 12:19:52 PM **Go to Map** **Change Site**

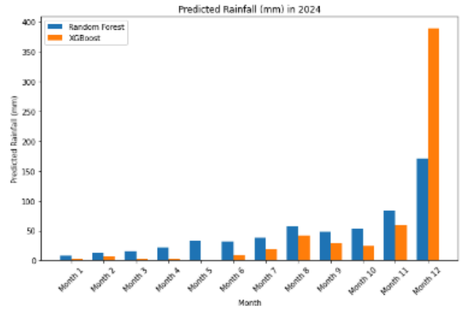
Select Weather Parameter

Rainfall | Temperature | Relative Humidity | Wind Speed

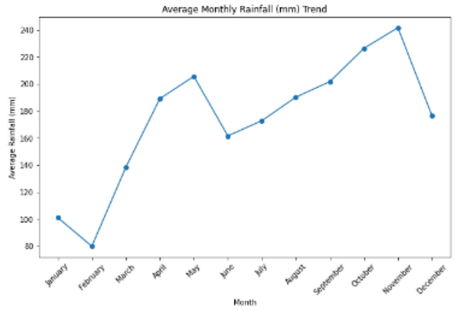
Live Weather

Weather: broken clouds | Temperature: 31.69°C | Humidity: 78% | Wind Speed: 2.68 m/s

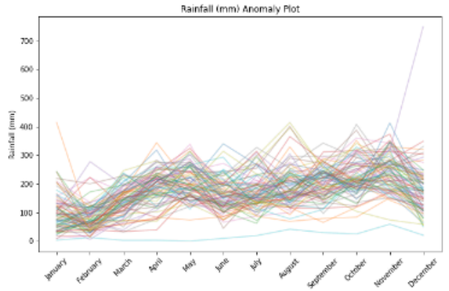
Predicted Values for 2024



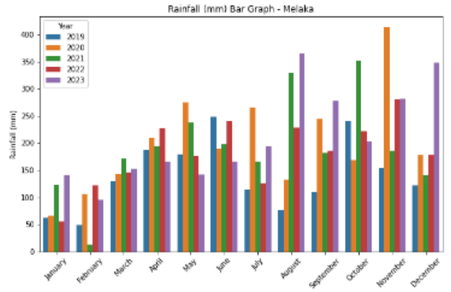
Average Monthly Trend 1940 - 2023



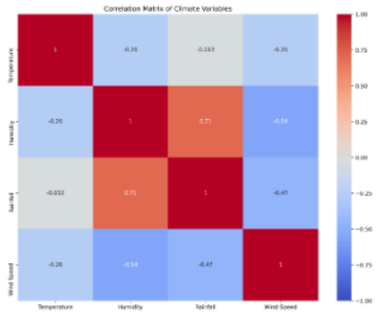
Anomaly Plot 1940 - 2023



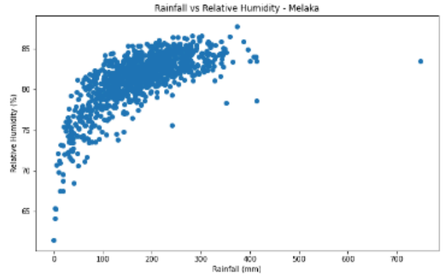
Last 5 Years Trend 2019 - 2023



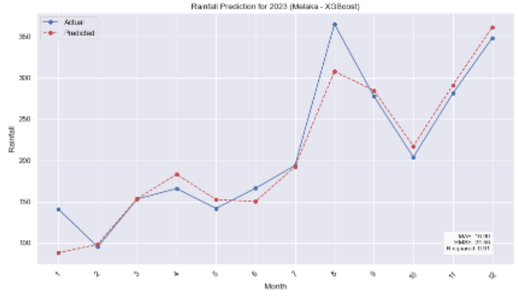
Correlation Matrix



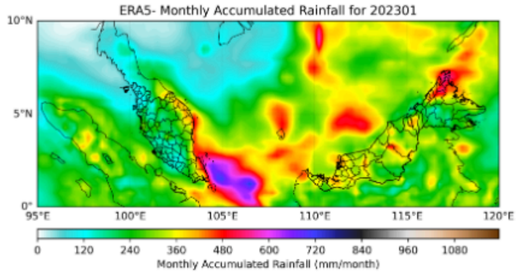
Scatter Plots 1940 - 2023



Actual vs Predicted 2023



Monthly Heatmaps



ACHIEVEMENT OF PROJECT OBJECTIVES

To identify and compare the most suitable machine learning algorithms for analyzing microclimate data in cultural heritage sites.

Achieved in Chapter 5

To evaluate the performance and accuracy of the developed machine learning models in predicting microclimate conditions.

Achieved in Chapter 5

To design and develop a user-friendly dashboard for displaying microclimate data trends and predictions.

Achieved in Chapter 4