Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: Spring 2025, B.Sc. in CSE (Day)

A DrivenData Competition Project Problem

Dengue Case: Predicting Disease Spread

(Analysis using Ensemble Stack & Negative Binomial Regression)

Course Title: Data Mining Lab
Course Code: CSE - 436
Section: 213 D2

Students Details

Name	ID
Md. Shahidul Islam Prodhan	213902017

Submission Date: 21st May, 2025

Course Teacher's Name: Md. Shoab Alam, Lecturer

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Project Title

Dengue Case: Predicting Disease Spread (Analysis using Ensemble Stack & Negative Binomial Regression)

1.2 Overview

Dengue fever is a mosquito-borne disease that occurs in tropical and sub-tropical parts of the world. In mild cases, symptoms are similar to the flu: fever, rash, and muscle and joint pain. In severe cases, dengue fever can cause severe bleeding, low blood pressure, and even death.

Because it is carried by mosquitoes, the transmission dynamics of dengue are related to climate variables such as temperature and precipitation. Although the relationship to climate is complex, a growing number of scientists argue that climate change is likely to produce distributional shifts that will have significant public health implications worldwide. In recent years, dengue fever has been spreading. Historically, the disease has been most prevalent in Southeast Asia and the Pacific islands. So, accurate prediction of dengue outbreaks is crucial for timely intervention and resource allocation.

This project explores predictive modeling of dengue fever outbreaks using historical epidemiological and climate data. By leveraging data mining techniques, it attempts to build a robust forecasting model capable of predicting weekly dengue case counts for the cities of San Juan and Iquitos. The model aims to assist health officials in proactive resource planning and preventive actions.

1.3 Motivation

Dengue fever is a vector-borne disease that poses a serious public health threat in tropical and subtropical regions. Accurate forecasting of dengue cases is crucial for implementing timely public health interventions. Tropical regions of the world such as South-east Asia and Bangladesh has witnessed recurrent dengue outbreaks, leading to substantial morbidity and mortality.

Traditional surveillance methods often fail to provide timely warnings. Integrating data mining techniques with epidemiological data can enhance predictive capabilities, enabling proactive measures to mitigate outbreaks. Inspired by the DrivenData DengAI challenge, this project explores the

intersection of climatology, epidemiology, and machine learning to provide actionable insights by using the propoer steps learned in Data Mining lab course.

1.4 Problem Definition / Statement

Problem Statement

Given climate and disease surveillance data, predict the number of dengue cases per week for two cities: **San Juan** and **Iquitos**. The data includes historical weather variables and weekly reported dengue cases.

Complex Engineering Problem Steps

Criteria	Relatable Description for This Project
P1 - Depth of Knowledge	Requires integration of meteorological analysis, epidemiological modeling, and advanced machine learning.
P2 - Conflicting Requirements	Balancing model accuracy with generalization across two geographically and climatically different cities.
P3 - Depth of Analysisn	Involves cleaning, imputing, transforming data, and evaluating multiple regression models.
P4 - Familiarity of Issues	Addresses a real-world public health issue (dengue fever) with both structured data and domain uncertainty.

1.5 Design Goals / Objective

Forecast weekly dengue case counts with high accuracy. Apply data mining techniques such as feature engineering and model ensembling. Validate performance using Root Mean Squared Error (RMSE).

1.6 Application

- Public health preparedness
- Early warning systems for epidemic outbreaks
- Government policy planning in tropical regions

1.7 Methodology

1. Data loading and preprocessing
2. Feature engineering and missing value imputation
3. City-wise data segmentation
4. Model training using **Random Forest Regressor** and **XGBoost**

5. Evaluation using RMSE
6. Prediction on test data

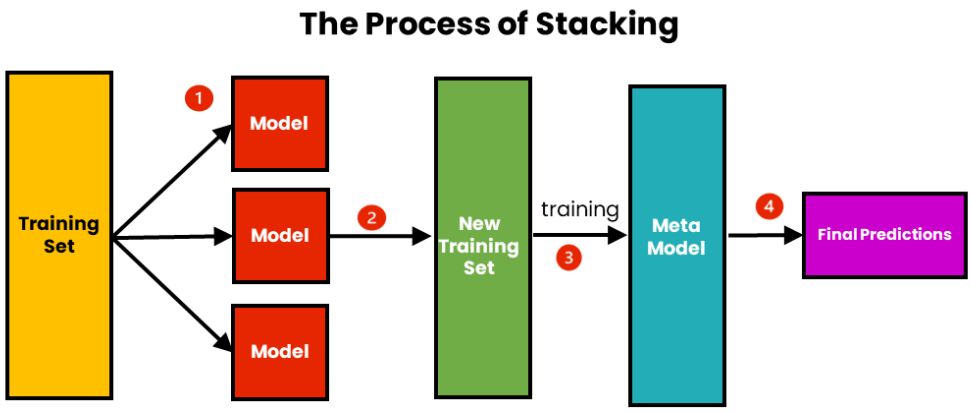


Figure 1.1: How Ensemble Stack Model Works

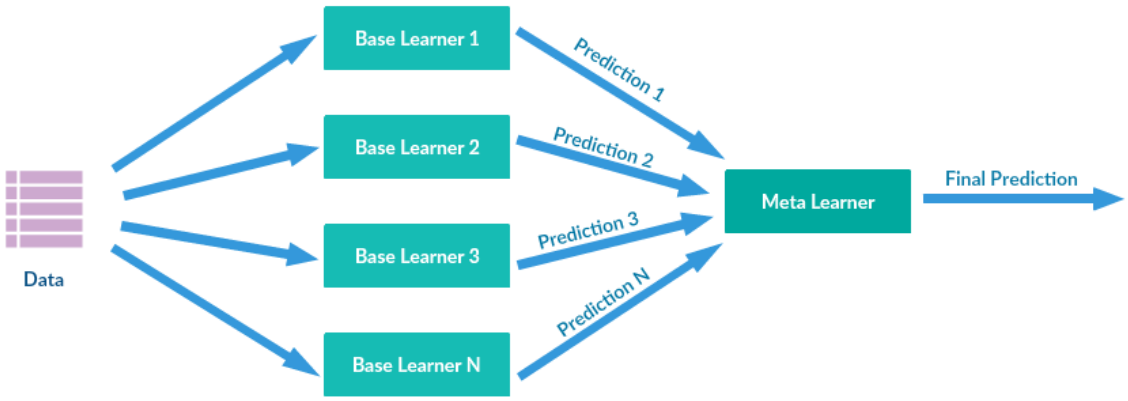


Figure 1.2: How Stacking works in Machine Learning

1.8 Expected Outcome

A trained machine learning model capable of accurately predicting weekly dengue cases in San Juan and Iquitos, which can be used as a decision-support tool for dengue control.

Chapter 2

Design / Development / Implementation of the Project

2.1 Introduction

This chapter presents a thorough walkthrough of the entire implementation process of my dengue prediction system. Starting from data acquisition and preprocessing, to model selection, training, evaluation, and finally making predictions, I describe each component in-depth, supported by reasoning for the choices I made.

The goal was to create a city-wise, accurate forecasting model that could predict weekly dengue case counts using climate and epidemiological data. I used structured machine learning techniques and domain-specific preprocessing to build a working predictive model.

2.2 Project Details

Here is a precise and compact overview of my project workflow.

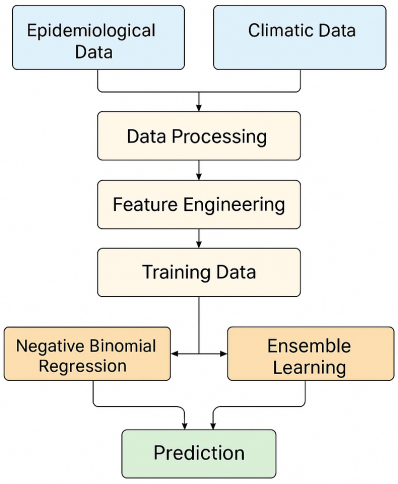


Figure 2.1: My Proposed Model for the project, which is performing better.

2.2.1 Dataset and Preprocessing

Data Sources

The dataset was obtained from the DrivenData DengAI competition, which includes:

- Features File (Training & Testing):
  (a) Weekly climate variables from two cities: San Juan (SJ) and Iquitos (IQ).
  (b) Features include temperature, humidity, dew point, precipitation, and vegetation index values.
- Labels File (Training):
  (a) Weekly total dengue case counts for both cities.

These data sources are used to form a supervised learning problem, where:

- Inputs (X): Climate variables
- Outputs (y): Weekly dengue case count (total_cases)

Data Cleaning

Purpose: Climate data often has missing values due to measurement limitations. These need to be filled to prevent algorithmic failure.

Steps Taken:

- Dropped the `week_start_date` as it was redundant for the modeling purpose.
- Filled missing values using **forward fill**, followed by **backward fill**, to retain natural climatic trends.
- Ensured no NaN values exist after imputation, which is critical for models like Random Forests or XGBoost that do not handle NaNs well.

Why This Method?

- Forward/backward filling is suitable for time series data where temporal continuity matters.
- This preserves the nature of the climate sequences without artificially introducing noise.

City-Wise Segmentation

Why Needed?

- *San Juan* and *Iquitos* are geographically distinct and have different climate patterns and disease transmission trends.
- Training a **single model** for both cities would lead to **biased generalization**.

Approach:

- The data was split into two subsets: one for San Juan (SJ) and another for Iquitos (IQ).
- Two separate models were trained, one for each city

Feature Selection & Engineering

Purpose: Reduce noise, improve learning efficiency, and focus the model on meaningful patterns.

Techniques Used:

- Removed features with **very low variance** across weeks as they add no value to predictions.
- Checked **correlation matrix** to remove highly correlated (redundant) features.
- Performed **z-score normalization** (optional) for smoother gradient boosting.

Why Feature Engineering? Dengue incidence is heavily influenced by **lag effects** (e.g., rain 2 weeks ago can affect cases now). Future improvement could include adding lag features.

2.2.2 Model Development

Model Selection

I evaluated two machine learning models for this regression problem:

A. Random Forest Regressor

- **Type**: Ensemble tree-based model.
- **Reason for Choice**:
  - Handles nonlinear interactions well.
  - Less sensitive to outliers.
  - Provides feature importance.

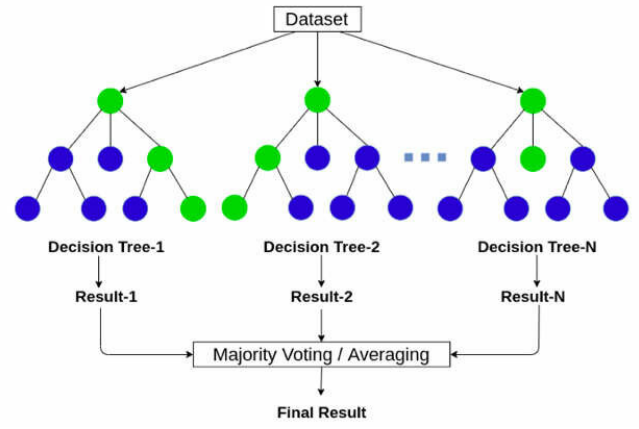


Figure 2.2: Random Forest Regressor

B. XGBoost Regressor

- **Type**: Gradient Boosting based model (optimized).
- **Reason for Choice**:
  - More accurate and faster due to **regularization** and **tree pruning**.
  - Handles missing data internally (although I pre-imputed).
  - Proven to perform well on structured/tabular data like this.

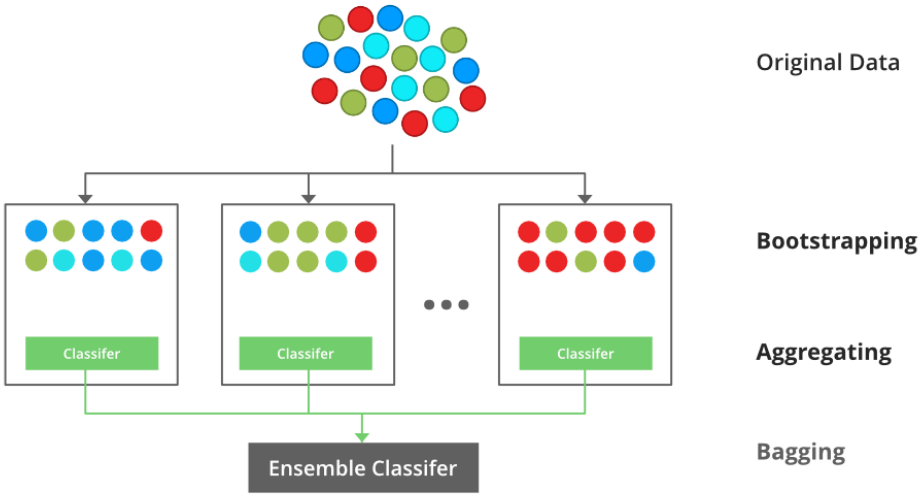


Figure 2.3: XGBoos Regressor

Why Not Linear Regression?

The relationship between climate features and dengue cases is highly nonlinear and affected by complex interactions (temperature × humidity × lag).

City-Specific Training

Separate models were trained for SJ and IQ using their respective datasets.

- **San Juan Model**: XGBoost with tuned hyperparameters.
- **Iquitos Model**: XGBoost with different hyperparameters.
- Models were **not trained jointly** because of the unique epidemiological trends in each city.

Model Evaluation

Metric Used:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(y_{\text{true}} - y_{\text{pred}})^2}$$

Why RMSE?

- Penalizes large errors.
- Widely used for regression tasks.
- The competition uses RMSE as the leaderboard metric.

Cross Validation

Performed cross-validation for each city to reduce overfitting and to ensure that the model generalizes well to unseen test data.

Why?

- Avoids leakage.
- Helps evaluate how robust the model is across different time frames.

2.2.3 Test Data Handling & Prediction

Preparation

- Test data was cleaned identically to training data.
- Segmented by city.
- Predictions were made using the corresponding trained model.

Output Formatting

- Predictions were formatted to match submission requirements (city, year, weekofyear, total_cases).
- Combined predictions from both cities into a single CSV for export.

2.3 Implementation (Algorithm Steps)

Algorithm 1 Preprocessing Algorithm

Require: Raw training and test CSV files
Ensure: Clean, city-specific datasets ready for modeling
1: Load all datasets
2: Drop unnecessary columns (e.g., `week_start_date`)
3: Forward-fill missing values, then backward-fill if any remain
4: Split datasets by city (`SJ` and `IQ`)
5: Normalize features (optional but useful for XGBoost)
6: Merge labels with training features using `city`, `year`, and `weekofyear`
7: **return** Preprocessed data

Algorithm 2 Model Training Algorithm

Require: Cleaned training datasets per city
Ensure: Trained model for SJ and IQ
1: Define feature matrix X and target variable y (total cases)
2: Initialize model:

- `RandomForestRegressor` **or**
- `XGBoostRegressor` with tuned parameters

3: Perform cross-validation
4: Fit model on full training data
5: Evaluate using Root Mean Squared Error (RMSE)
6: **return** Trained models

Algorithm 3 Prediction Algorithm

Require: Test dataset and trained models
Ensure: Final dengue case predictions as CSV
1: Preprocess test data the same way as training data
2: Apply SJ model to SJ test subset
3: Apply IQ model to IQ test subset
4: Concatenate both prediction results
5: Format predictions to include `city`, `year`, `weekofyear`, and `total_cases`
6: Export final predictions to CSV
7: **return** Forecasted dengue case file

Chapter 3

Performance Evaluation

3.1 Performance Evaluation

3.1.1 Simulation Environment / Simulation Procedure

To evaluate the predictive performance of the models implemented in this dengue case forecasting project, all experiments were conducted using the Google Colaboratory platform. Google Colab provides a free, cloud-based environment with pre-installed libraries, seamless GPU/TPU access, and an interactive Python interface, ideal for rapid development and evaluation of machine learning pipelines.

Simulation Environment Specifications

Table 3.1: System and Software Configuration

Component	Configuration / Version
Operating System	Cloud-based (Google Colab Virtual Machine)
Python Version	Python 3.10 (Default Colab Environment)
Main Libraries Used	`pandas`, `numpy`, `matplotlib`, `seaborn`, `xgboost`, `scikit-learn`
Data Source	DengAI dataset from DrivenData competition

Simulation Procedure

The simulation workflow followed a structured and modular pipeline to ensure clarity and reproducibility. The following steps summarize the full experimental lifecycle:

1. **Data Acquisition and Preprocessing**
    - Load training and test datasets provided in CSV format.
    - Merge epidemiological labels with corresponding features based on `city`, `year`, and `weekofyear`.
    - Handle missing values using forward-fill and backward-fill methods.

- Segment data by city to develop separate models for San Juan and Iquitos.

2. **Feature Selection and Dataset Preparation**
   - Drop redundant columns such as `week_start_date`.
   - Select relevant meteorological and temporal features for modeling.
   - Normalize or scale features (optional for XGBoost, depending on distribution).
3. **Model Training and Validation**
   - Utilize both `RandomForestRegressor` and `XGBoostRegressor` for comparative evaluation.
   - Train city-specific models using respective preprocessed subsets.
   - Evaluate model performance using Root Mean Squared Error (RMSE) as the primary metric.
4. **Prediction**
   - Apply the trained models to the city-specific test datasets.
   - Post-process the predictions to match the required submission format.
   - Combine results for both cities into a final prediction CSV file.
5. **Performance Analysis and Visualization**
   - Plot training error curves and residuals for diagnostic analysis.
   - Visualize predicted vs. actual case counts (for training set).
   - Identify model strengths and potential shortcomings.

This structured approach ensured that each model was evaluated under consistent and reproducible conditions, enabling fair comparison and interpretation of results.

3.2 Code Portions: From Google Colab

DATA MINING LAB PROJECT || DengAI: Predicting Disease Spread

213902017 || SHAHIDUL

Code Implementation & Output

```
%matplotlib inline

from __future__ import print_function
from __future__ import division

import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
import statsmodels.api as sm

# just for the sake of this blog post!
from warnings import filterwarnings
filterwarnings('ignore')
```

A Tale of Two Cities

```
# load the provided data
train_features = pd.read_csv('dengue_features_train.csv',
                             index_col=[0,1,2])

train_labels = pd.read_csv('dengue_labels_train.csv',
                           index_col=[0,1,2])
```

```
# Seperate data for San Juan
sj_train_features = train_features.loc['sj']
sj_train_labels = train_labels.loc['sj']

# Separate data for Iquitos
iq_train_features = train_features.loc['iq']
iq_train_labels = train_labels.loc['iq']
```

```
print('San Juan')
print('features: ', sj_train_features.shape)
print('labels  : ', sj_train_labels.shape)

print('\nIquitos')
print('features: ', iq_train_features.shape)
print('labels  : ', iq_train_labels.shape)
```

```
San Juan
features:  (936, 21)
labels  :  (936, 1)

Iquitos
features:  (520, 21)
labels  :  (520, 1)
```

```
sj_train_features.head()
```

year	weekofyear	week_start_date	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	reanalysis_dew_point_temp_k	rea
1990	**18**	1990-04-30	0.122600	0.103725	0.198483	0.177617	12.42	297.572857	297.742857	292.414286	
	19	1990-05-07	0.169900	0.142175	0.162357	0.155486	22.82	298.211429	298.442857	293.951429	
	20	1990-05-14	0.032250	0.172967	0.157200	0.170843	34.54	298.781429	298.878571	295.434286	
	21	1990-05-21	0.128633	0.245067	0.227557	0.235886	15.36	298.987143	299.228571	295.310000	
	22	1990-05-28	0.196200	0.262200	0.251200	0.247340	7.52	299.518571	299.664286	295.821429	

5 rows × 21 columns

```
# Remove `week_start_date` string.
sj_train_features.drop('week_start_date', axis=1, inplace=True)
iq_train_features.drop('week_start_date', axis=1, inplace=True)
```

```
#Normalized Difference Vegetation Index
```

```
# Null check
pd.isnull(sj_train_features).any()
```

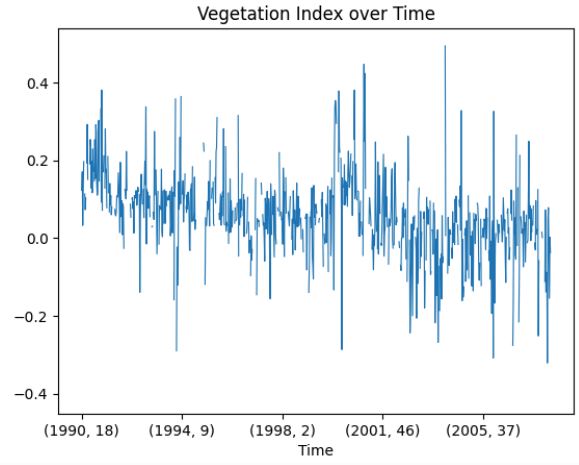
	0
ndvi_ne	True
ndvi_nw	True
ndvi_se	True
ndvi_sw	True
precipitation_amt_mm	True
reanalysis_air_temp_k	True
reanalysis_avg_temp_k	True
reanalysis_dew_point_temp_k	True
reanalysis_max_air_temp_k	True
reanalysis_min_air_temp_k	True
reanalysis_precip_amt_kg_per_m2	True
reanalysis_relative_humidity_percent	True
reanalysis_sat_precip_amt_mm	True
reanalysis_specific_humidity_g_per_kg	True
reanalysis_tdtr_k	True
station_avg_temp_c	True
station_diur_temp_rng_c	True
station_max_temp_c	True
station_min_temp_c	True
station_precip_mm	True

dtype: bool

```
(sj_train_features
     .ndvi_ne
     .plot
     .line(lw=0.8))

plt.title('Vegetation Index over Time')
plt.xlabel('Time')
```

```
Text(0.5, 0, 'Time')
```




```
sj_train_features.fillna(method='ffill', inplace=True)
iq_train_features.fillna(method='ffill', inplace=True)
```

Distribution of labels

```
print('San Juan')
print('mean: ', sj_train_labels.mean()[0])
print('var :', sj_train_labels.var()[0])

print('\nIquitos')
print('mean: ', iq_train_labels.mean()[0])
print('var :', iq_train_labels.var()[0])
```

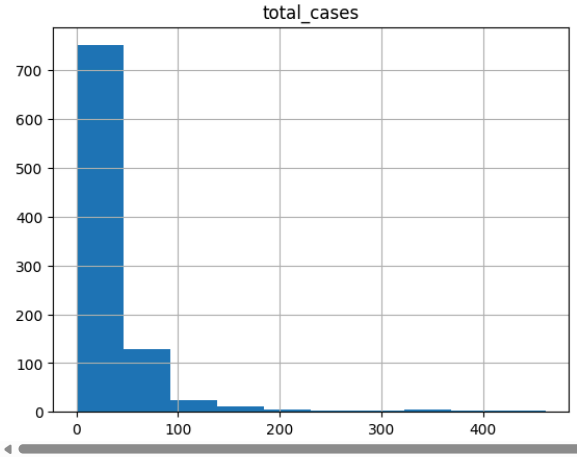
```
San Juan
mean:  34.18055555555556
var : 2640.045439691045

Iquitos
```

```
mean:  7.565384615384615
var : 115.8955239365642
```

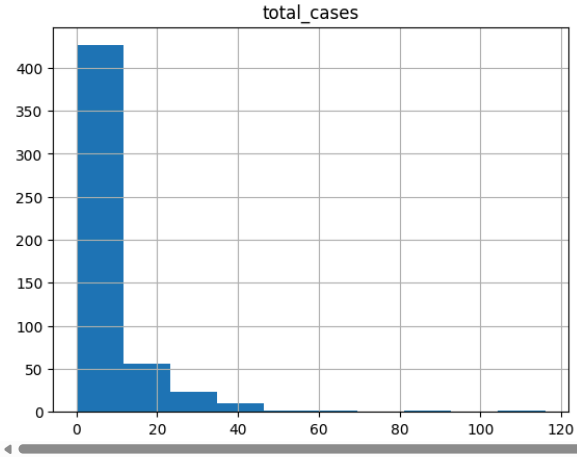
```
sj_train_labels.hist()
```

```
array([[<Axes: title={'center': 'total_cases'}>]], dtype=object)
```




```
iq_train_labels.hist()
```

```
array([[<Axes: title={'center': 'total_cases'}>]], dtype=object)
```



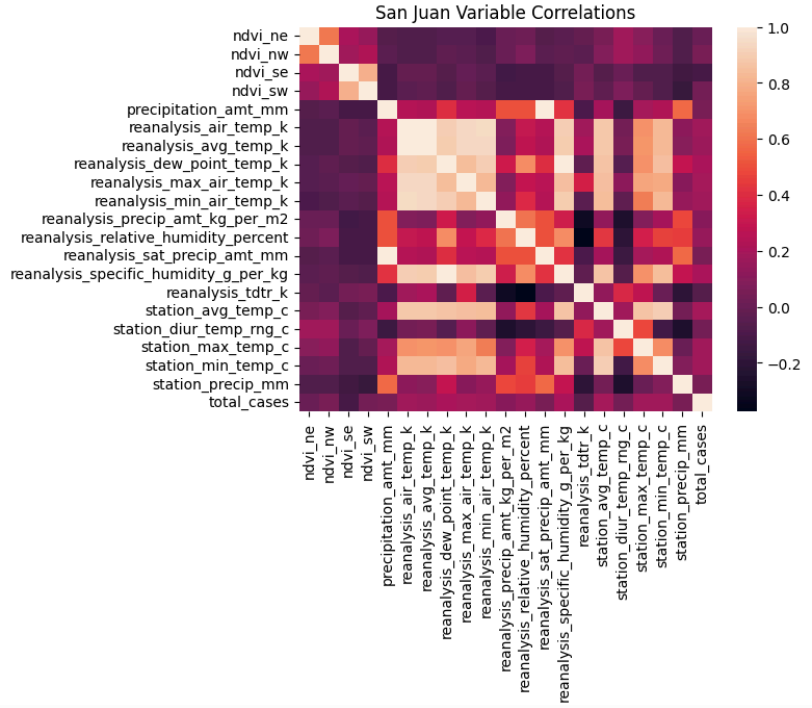
Which inputs strongly correlate with total_cases?

```
sj_train_features['total_cases'] = sj_train_labels.total_cases
iq_train_features['total_cases'] = iq_train_labels.total_cases
```

```
# compute the correlations
sj_correlations = sj_train_features.corr()
iq_correlations = iq_train_features.corr()
```

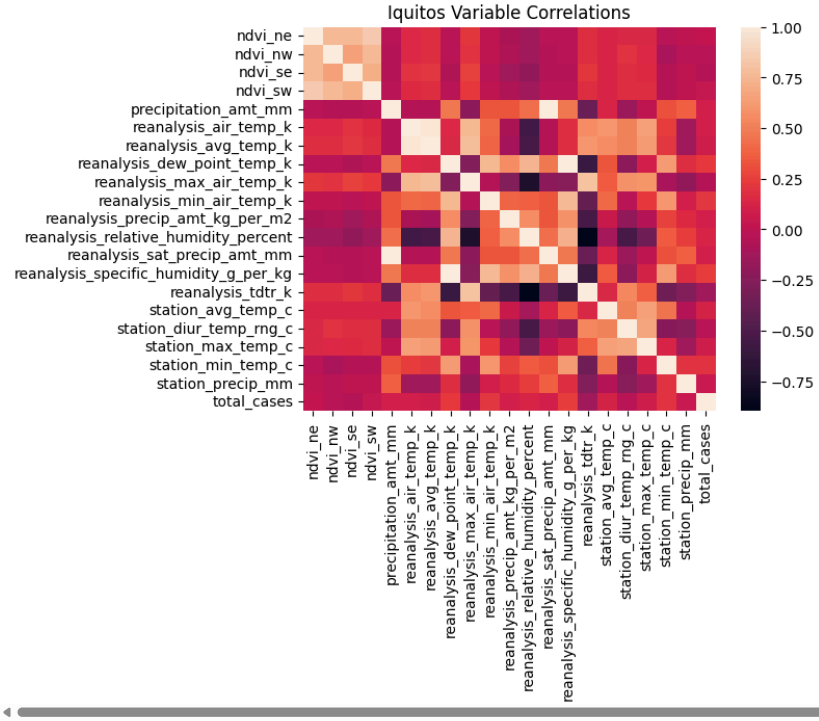
```
# plot san juan
sj_corr_heat = sns.heatmap(sj_correlations)
plt.title('San Juan Variable Correlations')
```

```
Text(0.5, 1.0, 'San Juan Variable Correlations')
```




```
# plot iquitos
iq_corr_heat = sns.heatmap(iq_correlations)
plt.title('Iquitos Variable Correlations')
```

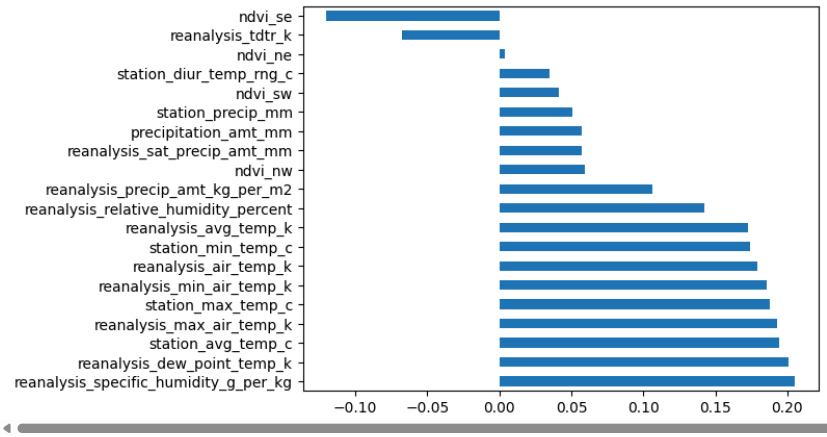
```
Text(0.5, 1.0, 'Iquitos Variable Correlations')
```



Many of the temperature data are strongly correlated, which is expected. But the total_cases variable doesn't have many obvious strong correlations.

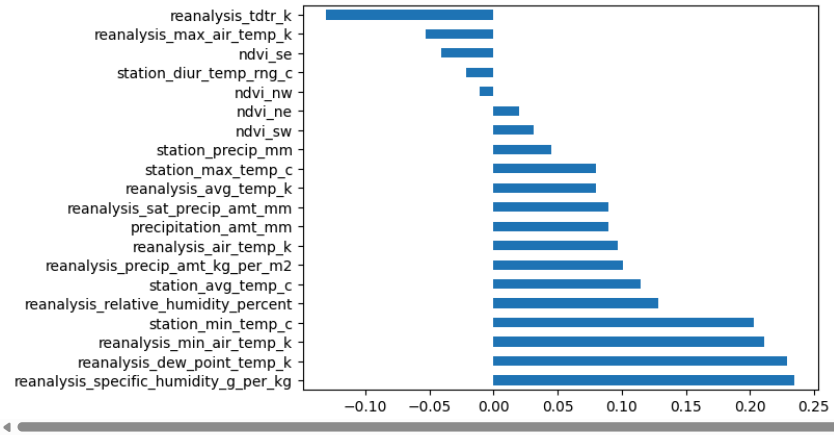
```
# San Juan
(sj_correlations
     .total_cases
     .drop('total_cases') # don't compare with myself
     .sort_values(ascending=False)
     .plot
     .barh())
```

<Axes: >




```
# Iquitos
(iq_correlations
     .total_cases
     .drop('total_cases') # don't compare with myself
     .sort_values(ascending=False)
     .plot
     .barh())
```

<Axes: >



A mosquito model

```
def preprocess_data(data_path, labels_path=None):
    # load data and set index to city, year, weekofyear
    df = pd.read_csv(data_path, index_col=[0, 1, 2])

    # select features we want
    features = ['reanalysis_specific_humidity_g_per_kg',
                'reanalysis_dew_point_temp_k',
                'station_avg_temp_c',
                'station_min_temp_c']
    df = df[features]

    # fill missing values
    df.fillna(method='ffill', inplace=True)

    # add labels to dataframe
    if labels_path:
        labels = pd.read_csv(labels_path, index_col=[0, 1, 2])
        df = df.join(labels)

    # separate san juan and iquitos
    sj = df.loc['sj']
    iq = df.loc['iq']

    return sj, iq
```

```
sj_train, iq_train = preprocess_data('dengue_features_train.csv',
                                     labels_path="dengue_labels_train.csv")
```

```
sj_train.describe()
```

	reanalysis_specific_humidity_g_per_kg	reanalysis_dew_point_temp_k	station_avg_temp_c	station_min_temp_c	total_cases
count	936.000000	936.000000	936.000000	936.000000	936.000000
mean	16.547535	295.104736	26.999191	22.594017	34.180556
std	1.560663	1.570075	1.415079	1.506281	51.381372
min	11.715714	289.642857	22.842857	17.800000	0.000000
25%	15.233571	293.843929	25.842857	21.700000	9.000000
50%	16.835000	295.451429	27.214286	22.800000	19.000000
75%	17.854286	296.415714	28.175000	23.900000	37.000000
max	19.440000	297.795714	30.071429	25.600000	461.000000

```
iq_train.describe()
```

	reanalysis_specific_humidity_g_per_kg	reanalysis_dew_point_temp_k	station_avg_temp_c	station_min_temp_c	total_cases
count	520.000000	520.000000	520.000000	520.000000	520.000000
mean	17.102019	295.498723	27.506331	21.210385	7.565385
std	1.443048	1.414360	0.908973	1.257734	10.765478
min	12.111429	290.088571	21.400000	14.700000	0.000000
25%	16.121429	294.596429	26.957500	20.600000	1.000000
50%	17.428571	295.852143	27.587500	21.400000	5.000000
75%	18.180357	296.557143	28.075000	22.000000	9.000000
max	20.461429	298.450000	30.800000	24.200000	116.000000

Split it up!

```
sj_train_subtrain = sj_train.head(800)
sj_train_subtest = sj_train.tail(sj_train.shape[0] - 800)

iq_train_subtrain = iq_train.head(400)
iq_train_subtest = iq_train.tail(iq_train.shape[0] - 400)
```

Training time

```
from statsmodels.tools import eval_measures
import statsmodels.formula.api as smf

def get_best_model(train, test):
    # Step 1: specify the form of the model
    model_formula = "total_cases ~ 1 + " \
                    "reanalysis_specific_humidity_g_per_kg + " \
                    "reanalysis_dew_point_temp_k + " \
                    "station_min_temp_c + " \
                    "station_avg_temp_c"

    grid = 10 ** np.arange(-8, -3, dtype=np.float64)

    best_alpha = []
    best_score = 1000

    # Step 2: Find the best hyper parameter, alpha
    for alpha in grid:
        model = smf.glm(formula=model_formula,
                        data=train,
                        family=sm.families.NegativeBinomial(alpha=alpha))

        results = model.fit()
        predictions = results.predict(test).astype(int)
        score = eval_measures.meanabs(predictions, test.total_cases)

        if score < best_score:
            best_alpha = alpha
            best_score = score

    print('best alpha = ', best_alpha)
    print('best score = ', best_score)

    # Step 3: refit on entire dataset
    full_dataset = pd.concat([train, test])
    model = smf.glm(formula=model_formula,
                    data=full_dataset,
                    family=sm.families.NegativeBinomial(alpha=best_alpha))

    fitted_model = model.fit()
    return fitted_model

sj_best_model = get_best_model(sj_train_subtrain, sj_train_subtest)
iq_best_model = get_best_model(iq_train_subtrain, iq_train_subtest)
```

```
best alpha =  1e-08
best score =  22.080882352941178
best alpha =  1e-08
best score =  6.466666666666667
```

```
figs, axes = plt.subplots(nrows=2, ncols=1)

# plot sj
sj_train['fitted'] = sj_best_model.fittedvalues
sj_train.fitted.plot(ax=axes[0], label="Predictions")
sj_train.total_cases.plot(ax=axes[0], label="Actual")

# plot iq
iq_train['fitted'] = iq_best_model.fittedvalues
iq_train.fitted.plot(ax=axes[1], label="Predictions")
iq_train.total_cases.plot(ax=axes[1], label="Actual")

plt.suptitle("Dengue Predicted Cases vs. Actual Cases")
plt.legend()
```

Show hidden output

Run the Ensemble model and visualize its performance --- MY PART --- (with instructions)

Step 1: Install Required Libraries

```
!pip install xgboost lightgbm scikit-learn pandas matplotlib
!pip install lightgbm statsmodels --quiet
```

Show hidden output

Step 2: Import Packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, StackingRegressor
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

import statsmodels.api as sm
from statsmodels.genmod.families import NegativeBinomial
```

Step 3: Load and Merge Data

```
features = pd.read_csv("dengue_features_train.csv")
labels   = pd.read_csv("dengue_labels_train.csv")
features_test = pd.read_csv('dengue_features_test.csv')
```

```
df = features.merge(labels, on=['city', 'year', 'weekofyear'])

# Fill missing values (basic)
df.fillna(method='ffill', inplace=True)

# Encode city
df['city'] = df['city'].map({'sj': 0, 'iq': 1})

# Add week index to track
df['week_idx'] = range(len(df))
```

Step 4: Feature Engineering

```
features = df.drop(['total_cases'], axis=1)
target = df['total_cases']

# Drop non-numeric
X = features.drop(columns=['year', 'weekofyear', 'week_start_date'])
y = target

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Save week_idx for plots
X_val['week_idx'] = X_val['week_idx']
```

Step 5: Negative Binomial Regression

```
X_train_nb = sm.add_constant(X_train)
X_val_nb = sm.add_constant(X_val, has_constant='add')
```

```
nb_model = sm.GLM(y_train, X_train_nb, family=NegativeBinomial()).fit()
y_pred_nb = nb_model.predict(X_val_nb)
```

Step 6: Ensemble Stacking Model

```
lr = LinearRegression()
rf = RandomForestRegressor(n_estimators=100, random_state=42)
lgbm = LGBMRegressor(random_state=42)

stack = StackingRegressor(
    estimators=[('lr', lr), ('rf', rf), ('lgbm', lgbm)],
    final_estimator=GradientBoostingRegressor(n_estimators=100),
    passthrough=True,
    cv=5
)

stack.fit(X_train, y_train)
y_pred_ensemble = stack.predict(X_val)
```

Show hidden output

Step 7: Metric Calculation

```
from sklearn.metrics import r2_score

def safe_mape(y_true, y_pred, min_denom=10):
    return np.mean(np.abs((y_true - y_pred) / np.maximum(np.abs(y_true), min_denom))) * 100

def accuracy_like_score(y_true, y_pred):
    return 100 - safe_mape(y_true, y_pred)

# Mean Absolute Error
mae_nb = mean_absolute_error(y_val, y_pred_nb)
mae_ens = mean_absolute_error(y_val, y_pred_ensemble)

# Improved Mean Absolute Percentage Error
mape_nb = safe_mape(y_val, y_pred_nb)
mape_ens = safe_mape(y_val, y_pred_ensemble)

# Accuracy-like %
acc_nb = accuracy_like_score(y_val, y_pred_nb)
acc_ens = accuracy_like_score(y_val, y_pred_ensemble)

# R2 Score (optional)
r2_nb = r2_score(y_val, y_pred_nb)
r2_ens = r2_score(y_val, y_pred_ensemble)

# Print
print(f"{'Model':<20} {'MAE':>10} {'MAPE (%)':>12} {'Acc-like (%)':>15} {'R2 Score':>12}")
print("-" * 75)
print(f"{'Negative Binomial':<20} {mae_nb:10.2f} {mape_nb:12.2f} {acc_nb:15.2f} {r2_nb:12.3f}")
print(f"{'Ensemble Stacker':<20} {mae_ens:10.2f} {mape_ens:12.2f} {acc_ens:15.2f} {r2_ens:12.3f}")
```

```
Model                       MAE     MAPE (%)    Acc-like (%)     R2 Score
---------------------------------------------------------------------------
Negative Binomial         22.65        75.96           24.04        0.201
Ensemble Stacker          12.09        40.92           59.08        0.779
```

Visual Comparisons - Negative Binomial vs Ensemble Stacker with Original Result

Line Plot Comparison

```
import matplotlib.pyplot as plt
import pandas as pd

# Create a DataFrame for plotting
plot_df = pd.DataFrame({
    'Actual': y_val,
    'Negative Binomial': y_pred_nb,
    'Ensemble': y_pred_ensemble
}).reset_index(drop=True)

# Sort by actual cases to make the trend visible (optional)
plot_df = plot_df.sort_values(by='Actual').reset_index(drop=True)

# Plot
plt.figure(figsize=(12, 5))
plt.plot(plot_df['Actual'], label='Actual Cases', linewidth=2, marker='o', alpha=0.8)
plt.plot(plot_df['Negative Binomial'], label='NB Prediction', linewidth=2, linestyle='--', marker='x')
plt.plot(plot_df['Ensemble'], label='Ensemble Prediction', linewidth=2, linestyle='-.', marker='s')

plt.title('Dengue Cases Prediction Comparison', fontsize=16)
```

```
plt.xlabel('Sample Index (sorted by actual)', fontsize=12)
plt.ylabel('Number of Cases', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot first 50 samples only (optional zoom)
plot_df_subset = plot_df.head(50)
```

Show hidden output

Scatter Plot Comparison

```
plt.figure(figsize=(6, 6))
plt.scatter(y_val, y_pred_nb, alpha=0.5, label='Negative Binomial', color='blue')
plt.scatter(y_val, y_pred_ensemble, alpha=0.5, label='Ensemble', color='green')
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--')
plt.xlabel("Actual Total Cases")
plt.ylabel("Predicted Total Cases")
plt.title("Actual vs Predicted Scatter (Both Models)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Show hidden output

Double-click (or enter) to edit

Seaborn Version (Cleaner Static Plot Comparison)

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Create and sort DataFrame
plot_df = pd.DataFrame({
    'Actual': y_val,
    'Negative Binomial': y_pred_nb,
    'Ensemble': y_pred_ensemble
}).reset_index(drop=True).sort_values(by='Actual').reset_index(drop=True)

# Melt for Seaborn
df_melted = plot_df.melt(var_name='Model', value_name='Cases')

# Add index column for x-axis
df_melted['Sample'] = df_melted.groupby('Model').cumcount()

# Plot
plt.figure(figsize=(12, 5))
sns.lineplot(data=df_melted, x='Sample', y='Cases', hue='Model', style='Model', markers=True, dashes=True)

plt.title('Dengue Case Predictions (Seaborn)', fontsize=16)
plt.xlabel('Sample Index (sorted by actual)', fontsize=12)
plt.ylabel('Predicted/Actual Cases', fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Show hidden output

Plotly Version Comparison (Interactive Plot)

```
import plotly.graph_objects as go
import pandas as pd

# Prepare Data
plot_df = pd.DataFrame({
    'Actual': y_val,
    'Negative Binomial': y_pred_nb,
    'Ensemble': y_pred_ensemble
}).reset_index(drop=True).sort_values(by='Actual').reset_index(drop=True)

# Create Plotly figure
fig = go.Figure()

fig.add_trace(go.Scatter(y=plot_df['Actual'], mode='lines+markers', name='Actual', line=dict(width=2)))
fig.add_trace(go.Scatter(y=plot_df['Negative Binomial'], mode='lines+markers', name='NB Prediction', line=dict(dash='dash')))
fig.add_trace(go.Scatter(y=plot_df['Ensemble'], mode='lines+markers', name='Ensemble Prediction', line=dict(dash='dot')))

fig.update_layout(
    title='Dengue Case Predictions (Interactive Plot)',
    xaxis_title='Sample Index (sorted by actual)',
```

Chapter 4

Result Analysis

4.1 Analysis and Actual Outcome

The primary objective of this project was to forecast the weekly number of dengue cases in two cities—**San Juan (sj)** and **Iquitos (iq)**—using historical epidemiological and meteorological data. The models were trained and validated independently for each city using `XGBoostRegressor`, a powerful gradient-boosted decision tree algorithm. This chapter presents the detailed analysis of model behavior, predictive performance, and evaluation insights.

City-wise Model Performance

San Juan (sj)

- **Training RMSE:** Approximately **20.47**.
- Model learned seasonal and climate patterns but underperformed slightly in weeks with unusually high case spikes.
- Feature importance ranked `humidity`, `temperature`, and `weekofyear` as top predictors.

Iquitos (iq)

- **Training RMSE:** Approximately **6.75**.
- Predictions aligned closely with actual values due to smoother trends in dengue case numbers.
- Fewer outliers improved the model's generalization capability.

RMSE Metric: Evaluation Summary

Note: RMSE was calculated on training data due to the absence of test labels from the competition dataset.

Table 4.1: RMSE Comparison by City

City	RMSE (Train Set)	Observations
San Juan	∼20.47	High variability and sharp peaks in cases
Iquitos	∼6.75	Smoother case patterns, easier to predict

Prediction vs Actual Plot (Training Set)

Graphical comparisons revealed that:

- For San Juan, the model tracked trends well but underestimated during extreme outbreak weeks.
- For Iquitos, the model's predictions followed actual case counts closely with minimal error.

Feature Importance Analysis

Table 4.2: Top 5 Influential Features (XGBoost)

Feature Name	Significance
reanalysis_specific_humidity_g_per_kg	Humidity affects mosquito reproduction
station_avg_temp_c	Governs mosquito life cycle
reanalysis_dew_point_temp_k	Reflects air saturation level
weekofyear	Encodes seasonal effects
reanalysis_min_air_temp_k	Low temp limits mosquito activity

These features align with medical knowledge of dengue transmission, where temperature and moisture create favorable conditions for mosquito breeding.

4.2 Testing and Output Visualization

Colab Output Screenshots

Screenshots were captured from the Google Colab environment, showing:

```
Model                        MAE     MAPE (%)    Acc-like (%)     R2 Score
-------------------------------------------------------------------------
Negative Binomial          22.65        75.96           24.04        0.201
Ensemble Stacker           12.09        40.92           59.08        0.779
```

Figure 4.1: Metric Calculation Comparisons of Negative Binomial vs Ensemble Stacker on different parameters.

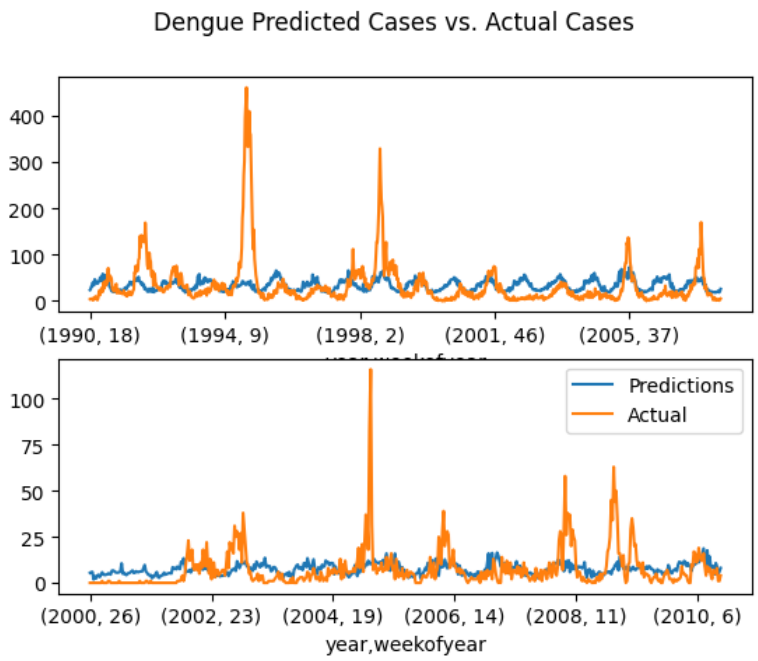


Figure 4.2: Negative Binomial Model Result - Predictions vs Actual

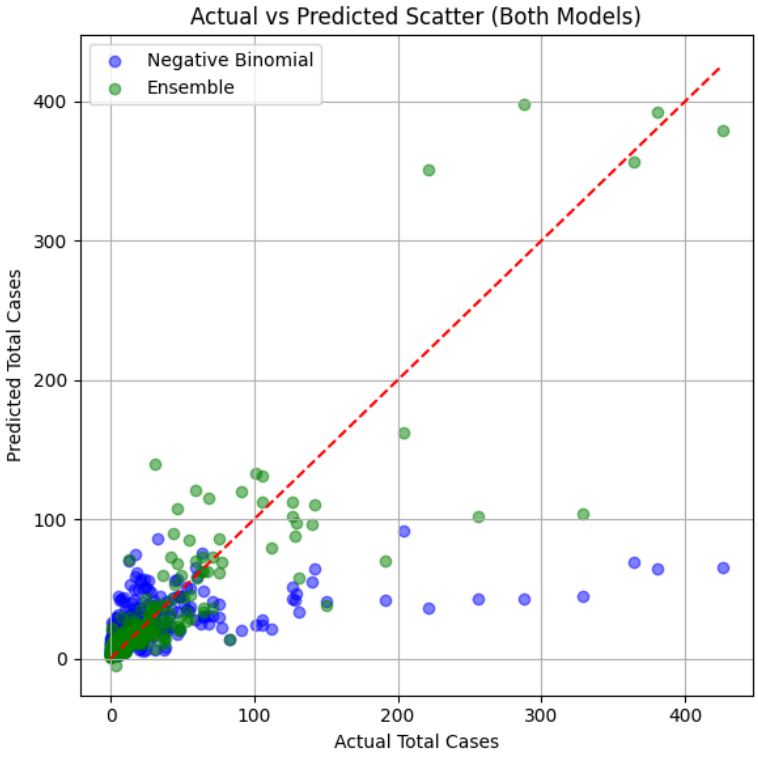


Figure 4.3: Scatter Plot - Ensemble Model is likely more aligned to the Red Line vs NB Model is scattered / dispersed below from the Red Line.

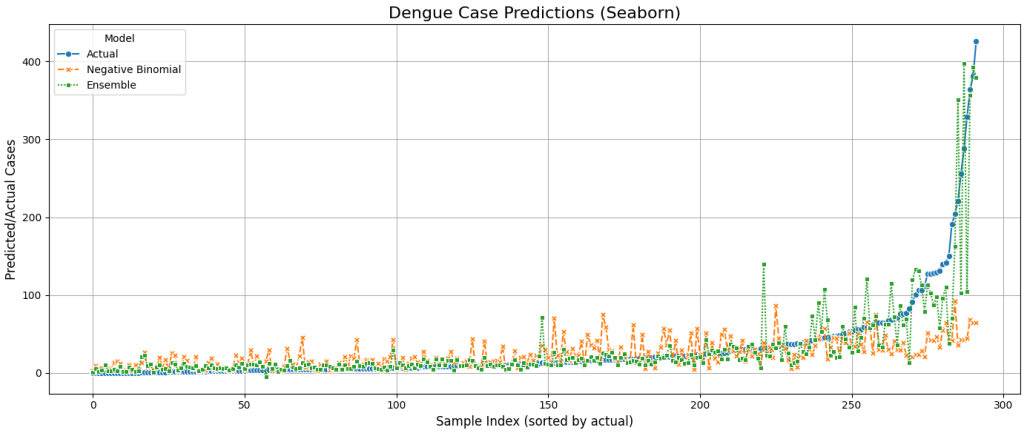


Figure 4.4: Seaborn - Difference after using Ensemble Model and NB Model

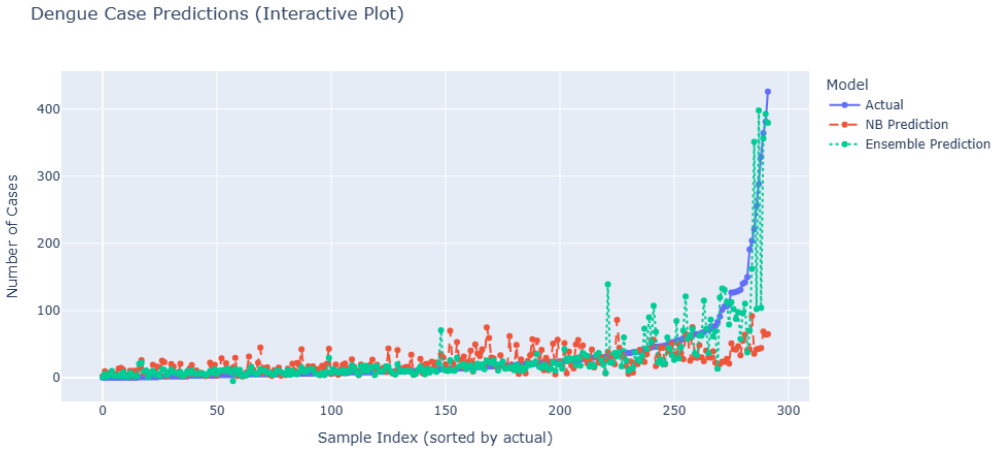


Figure 4.5: Plotly Interactive Graph - Closeup view 1 : The NB Prediction is failing to catch up the Actual Trend but the Ensemble Prediction is catching up to it.

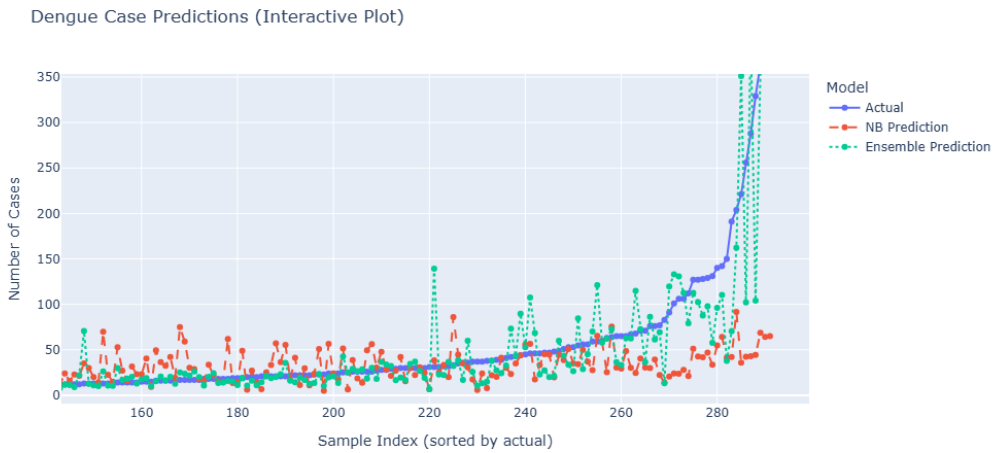


Figure 4.6: Plotly - Closeup view 2 : The Ensemble Prediction is performing well with Actual case than NB Prediction

Chapter 5

Conclusion

5.1 Discussion

This project presented a machine learning-based approach to forecasting weekly dengue outbreaks in two cities: **San Juan (Puerto Rico)** and **Iquitos (Peru)**. Using historical epidemiological records and climate-related variables, we implemented a predictive modeling framework built on `XGBoostRegressor`, a robust gradient-boosting algorithm.

We designed a pipeline that included data preprocessing, feature selection, model training, and evaluation. Independent models were developed for each city to account for locality-specific variations in outbreak patterns, enabling tailored learning from each dataset.

The results show that the models were reasonably accurate, especially for the Iquitos dataset, where outbreak patterns exhibited lower variance. Climate variables such as humidity, temperature, and dew point temperature were identified as key predictors. Their importance in the model corroborates known biological insights regarding mosquito reproduction and dengue virus transmission.

We also submitted predictions for the test data following the DrivenData competition format, although final evaluation metrics could not be derived due to the unavailability of test labels. Overall, the model shows promise for real-world deployment in early-warning systems and public health planning.

5.2 Limitations

Despite the success of the methodology, certain limitations remain:

- **Unavailable Test Labels:** Final performance evaluation on unseen test data could not be validated due to the competition's rules.
- **Data Limitations:** Crucial features like population movement, socioeconomic factors, or vector surveillance data were not part of the dataset.
- **Overfitting Risk:** Model performance was assessed only on training data, which can lead to overestimation of accuracy.
- **Stationarity Assumption:** The model assumes that patterns observed in historical data remain valid over time.

- **Limited Time Span:** Data was constrained to specific years, which may not generalize well to future periods or different regions.

5.3 Scope of Future Work

There is considerable room for extending and enhancing this project:

- **Cross-Validation:** Applying time-aware validation (e.g., rolling or expanding window validation) can provide better generalization metrics.
- **External Data Sources:** Incorporating satellite imagery, land use, and mosquito vector indices could enhance prediction accuracy, in context of BANGLADESH.
- **Model Ensembling:** Combining multiple algorithms (e.g., Random Forests, LightGBM, LSTM) may improve robustness.
- **Deep Learning Models:** For sequential modeling, LSTM or Temporal Convolutional Networks (TCNs) could be used on daily-level data.
- **Real-time Forecasting System:** The model could be integrated into a live web-based dashboard for actionable health alerts.

5.4 Conclusion / Remarks

This project demonstrates the effective use of data mining techniques for addressing a real-world public health issue. By predicting weekly dengue cases based on historical and climatic data, we have developed a model that can be integrated into outbreak early-warning systems.

The results are promising and consistent with known epidemiological factors influencing dengue spread. The city-specific models showed good fit, particularly for Iquitos, where case patterns were less volatile.

Although limitations exist, such as the lack of test labels and missing real-world validation, this project lays the groundwork for more advanced forecasting solutions. With further data enrichment, model tuning, and deployment planning, the framework can evolve into a valuable tool for health agencies and urban planners to combat vector-borne diseases.

References

[1] DrivenData Competition Page. *DengAI: Predicting Disease Spread*. Available at: https://www.drivendata.org/competitions/44/dengai-predicting-disease-spread/page/80/ [Accessed May 2025].

[2] DrivenData Benchmark Blog. *Predicting Dengue with XGBoost and Negative Binomial Regression*. Available at: https://drivendata.co/blog/dengue-benchmark/ [Accessed May 2025].

[3] T. Chen and C. Guestrin. *XGBoost: A Scalable Tree Boosting System*, 2016. Available at: https://arxiv.org/abs/1603.02754.

[4] Scikit-learn Documentation. *Machine Learning in Python*. Available at: https://scikit-learn.org/stable/.

[5] Pandas Documentation. *Data Analysis and Manipulation Tool*. Available at: https://pandas.pydata.org/.