Problem Description

The Kaggle challenge is to predict the occurrence of the West Nile virus (WNV) in mosquitos across Chicago, using historical data on mosquito tests and weather conditions. The City of Chicago have been monitoring WNV through a comprehensive surveillance program, which includes testing mosquitos trapped across the city.

The training dataset includes the date and location of mosquito traps, the species of mosquitos, the number of mosquitos caught, and whether WNV was detected. The dataset covers the years 2007, 2009, 2011, and 2013. The weather dataset provides daily weather data from the National Oceanic and Atmospheric Administration for the same years as the training data. The weather data includes temperature, precipitation, and other relevant meteorological variables.

The goal is to predict the probability that a mosquito trap will test positive for WNV on a given date. The model is evaluated using ROC-AUC metric, which assesses the its ability to distinguish between positive and negative cases of WNV presence. This problem is of significant relevance to biostatistics and epidemiology, as it involves analyzing complex datasets to understand the relationship between environmental factors and disease prevalence. Accurate prediction of WNV presence in mosquitos is vital for public health, as it can inform targeted intervention strategies to reduce the risk of transmission to humans, ultimately aiding in the prevention and control of infectious diseases.

Methodology, Results & Evaluation

I merged the training data with the weather data based on the date of the mosquito tests to create a comprehensive dataset for modeling. This merged dataset has been preprocessed to handle missing values, outliers, and one-hot-encoding is employed to transform categorical variables for machine learning algorithms. I normalized numerical features using **StandardScaler** in sklearn to ensure they are on the same scale.

I then conducted an EDA to identify potential patterns. The initial approach involved building a logistic regression model. The dataset was split into 70% training and 30% testing sets. the AUC score was the primary metric due to its effectiveness in evaluating the true positive rate against the false positive rate, which is particularly important in the context of imbalanced classes. However, this model yielded an AUC score 0.55, indicating performance just a bit better than a random guess. In the meantime, the accuracy score is 0.95. This suggests the model is not effectively capturing the minority class due to class imbalance. The value count of 19,910 entries of 0's and only 1102 of 1's in the column **WnvPresent** proved that my judgement is correct. To address the issue, undersampling is used with **RandomUnderSampler** in sklearn to balance the distribution of classes by reducing the size of the majority class to match the size of the minority class. At another logistic regression attempt, AUC score increased to 0.79.

Tree-based models, to my knowledge, are often preferred over logistic regression because they can capture complex, non-linear relationships between features, and are more robust to outliers and imbalanced datasets. A baseline decision tree model was created with the Gini criterion for measuring the quality of splits. The model was then fitted to the resampled training set. I only used the default parameters for the decision tree. The model's performance was evaluated on the test set, and the AUC score is also 0.79. I was a bit surprised it performed not bad at all, considering its drawbacks in that decisions trees tend to have high variance, and a small change in the training data can produce big changes in the estimated tree.

I was then motivated by the desire to improve model performance and address the limitations of a single decision tree. I wanted to see potential for further improvement by leveraging the ensemble approach of random forests. Random forests build upon the simplicity of decision trees by creating an ensemble of multiple trees, which reduces the risk of overfitting and increases the model's ability to generalize to unseen data.

The hyperparameter tuning process for the model involved a grid search over a range of values with the cross-validation folds set to 3. The best parameters identified were **max_depth** of 10, **min_samples_split** of 2, and **n_estimators** of 100. These parameters led to an improved AUC score of 0.81 on the test set. This improvement conformed my preconception of the effectiveness of random forests in handling the complexities of the dataset.

Having secured a solid AUC score with the random forest, I turned my attention to XGBoost due to its reputation for exceling in situations where the data involves complex and non-linear relationships, as it employs a gradient boosting framework that builds trees sequentially to minimize errors from previous trees. This iterative refinement is further enhanced by XGBoost's regularization feature, which penalizes more complex models to avoid overfitting, an edge that the Random Forest model lacks.

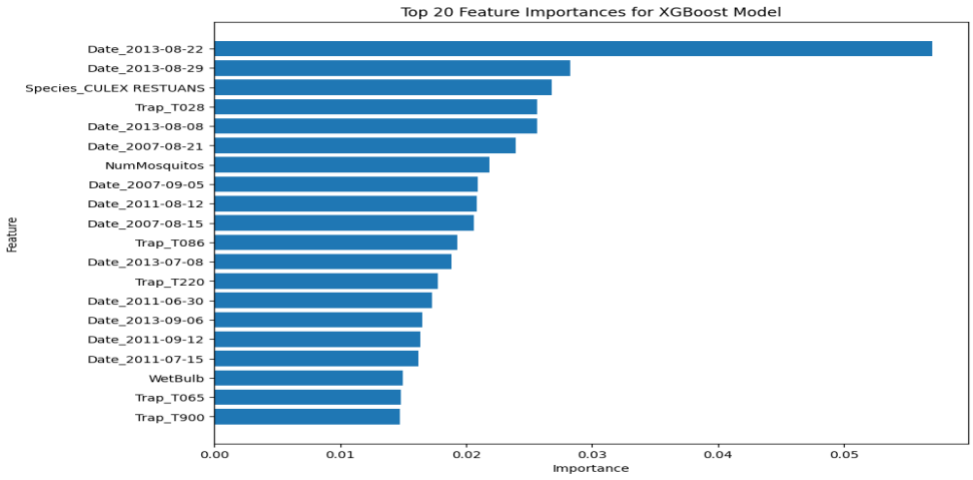
To optimize the XGBoost model, a grid search was conducted using **GridSearchCV** in sklearn with a concise yet strategic set of hyperparameters: **n_estimators, max_depth, subsample**, **colsample_bytree**, and **eta**, the learning rate. These particular hyperparameters were chosen to control the boosting process and the construction of the trees effectively. The grid search was executed over a 3-fold cross-validation to reduce computational load, while still providing a robust search across the parameter space. The result demonstrated an improved AUC score over the Random Forest model, which shot to 0.84. Therefore, XGBoost became my final and best choice.

Below is a comparative table of the AUC scores for each model:

Model	**AUC Score**
Logistic Regression	0.55
Logistic Regression with undersampling	0.79
Decision Tree	0.79
Random Forest with hyperparameter tuning	0.81
XGBoost with hyperparameter tuning	0.84

The increasing AUC scores from logistic regression to XGBoost underscore the value of exploring complex models and fine-tuning their parameters. Through this analysis, it became evident that while simpler models can offer valuable insights and serve as good baselines, advanced models like XGBoost, when properly tuned, can substantially improve predictive performance. Such improvements are instrumental for the practical application of the model, potentially aiding in more effective allocation of resources for the control and prevention of West Nile virus outbreaks.

At last, we can examine the top 20 feature importances for an XGBoost model by the chart below. The importance score indicates how useful each feature was in the construction of the boosted decision trees within the model.



From the chart, it appears that dates in summer and certain mosquito species ("Species_CULEX RESTUANS") are among the most influential factors in predicting the presence of the WNV. Additionally, some specific trap locations have a significant impact on the predictions. The presence of dates as top features might indicate that certain times of the year are more associated with the prevalence of the virus, possibly due to environmental factors like temperature and humidity. The importance of specific traps could reflect geographical areas that are more prone to WNV outbreaks, possibly due to local ecological conditions that favor mosquito breeding and virus transmission.

Appendix

Code Snippets for the XGBoost model

```
from sklearn.model_selection import GridSearchCV

# Define a simpler parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 150],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'eta': [0.1, 0.2]
}

# Instantiate the XGBoost classifier
xgb_model = xgb.XGBClassifier(seed=3)

# Set up the grid search with fewer cross-validation folds
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=2, scoring='roc_auc', n_jobs=-1, verbose=1)

# Fit the grid search to the resampled training data
grid_search_xgb.fit(X_train_res, y_train_res)

# Get the best parameters and the best AUC score
best_params_xgb = grid_search_xgb.best_params_
best_auc_score_xgb = grid_search_xgb.best_score_

print(f"Best parameters for XGBoost: {best_params_xgb}")
print(f"Best AUC score for XGBoost: {best_auc_score_xgb}")

# Use the best estimator for predictions
best_xgb_model = grid_search_xgb.best_estimator_
y_pred_best_xgb = best_xgb_model.predict(X_test_scaled)

# Compute the AUC score for the test set
auc_score_best_xgb = roc_auc_score(y_test, y_pred_best_xgb)
print(f"AUC score of the best XGBoost model: {round(auc_score_best_xgb, 2)}")
```

```
Fitting 2 folds for each of 32 candidates, totalling 64 fits
Best parameters for XGBoost: {'colsample_bytree': 1.0, 'eta': 0.2, 'max_depth': 5, 'n_estimators': 150, 'subsample': 0.8}
Best AUC score for XGBoost: 0.8442147231007335
```