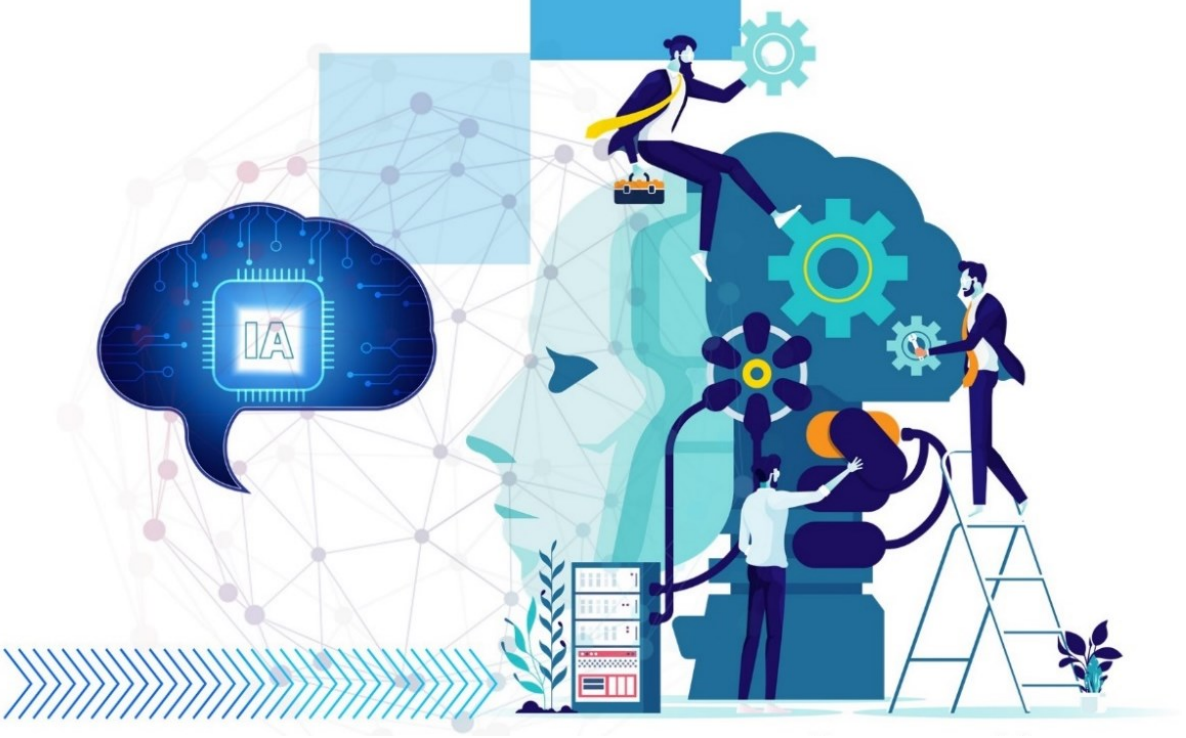
Teaching & Learning
Step-by-Step Guide:

Artificial intelligence applications for Spatial Analysis and Planning



Amila Jayasinghe

Samith Madusanka

Harini Sawandi

Teaching & Learning Step-by-step Guide:

Artificial Intelligence Application for Spatial Analysis and Planning

Authors

Amila Jayasinghe

Samith Madusanka

Harini Sawandi

Publisher

University of Moratuwa

Author contribution

1. Amila Jayasinghe (Supervision, Conceptualisation, Methodology, Validation), Department of Town & Country Planning, University of Moratuwa, Sri Lanka.
2. Samith Madusanka (Formal Analysis, Writing—original draft preparation), Department of Town & Country Planning, University of Moratuwa, Sri Lanka.
3. Harini Sawandi (Project Administration, Review and Editing), Department of Town & Country Planning, University of Moratuwa, Sri Lanka.

All authors have read and agreed to the published version of the book.

Contact authors amilabj@uom.lk

This book was produced with the valuable support of the Erasmus+ Capacity Building in Higher Education (CBHE) project 'Curricula Enrichment for Sri Lankan Universities delivered through the application of Location-Based Services to Intelligent Transport Systems' (LBS2ITS https://lbs2its.net/)

Project Number: 618657-EPP-1-2020-1-AT-EPPKA2-CBHE-JP
Programme: Erasmus+
Key Action: Cooperation for innovation and the exchange of good practices
Action Type: Capacity Building in Higher Education
Co-funding: Erasmus+ Programme of the European Union

This book was reviewed as an Open Education Resource for University students by Dr Rico Wittwer (Technische Universität Dresden — TU Dresden, Germany) under the LBS2ITS project.

lbs2its.net

LBS2ITS

Curricula Enrichment delivered through

the Application of Location-based Services

to Intelligent Transport Systems

Co-funded by the Erasmus+ Programme of the European Union

Edition

First Edition - May 2025

ISBN 978-955-9027-88-1 (ebook)

Citation

Jayasinghe, A., Madusanka, S., & Sawandi, H. (2025). *Teaching & learning step-by-step guide—Artificial intelligence application for spatial analysis and planning* (1st ed.). University of Moratuwa.

Publisher

University of Moratuwa

PREFACE

This book serves as open educational material for both undergraduate and postgraduate degree programs, offering a detailed, step-by-step guide to machine learning techniques using XGBoost and decision trees. Designed to bridge the gap between theoretical knowledge and practical application, this guide is meticulously crafted to meet the needs of students, educators, and practitioners alike.

Within the book, readers will find comprehensive instructions on setting up and configuring machine learning environments, preparing datasets, and implementing XGBoost and decision tree algorithms. It covers various applications, including predictive modeling, classification tasks, regression analysis, spatial analysis, urban planning, and more. The book not only enhances learning in academic settings by providing real-world applications and case studies but also equips industry professionals with the skills necessary to conduct advanced data analysis and contribute meaningful insights in their fields.

Key topics include detailed steps on initializing and configuring machine learning models to ensure optimal performance and accuracy, along with guidelines on effective data preprocessing to achieve high-quality input data. The book provides comprehensive instructions on training, evaluating, and tuning XGBoost and decision tree models. It features real-world examples and case studies that demonstrate the practical applications of these algorithms, particularly in spatial analysis and urban planning, as well as techniques for interpreting model results to support decision-making processes.

Whether you are a student aiming to master machine learning techniques and algorithm implementation, a teacher looking for robust educational tools, or a practitioner in need of refining your technical expertise, this book offers invaluable guidance and support. It ensures that users at all levels gain proficiency in leveraging modern machine learning technologies to explore and solve complex data-driven challenges effectively.

CONTENT

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This book offers comprehensive guidance for individuals and groups engaged in evaluating and tackling data-driven challenges using machine learning techniques. By gathering and analyzing various types of data, this guidebook provides users with the necessary tools and approaches to understand and solve complex problems through the application of XGBoost and decision tree algorithms.

Machine learning technologies, including XGBoost and decision trees, are essential for analyzing and interpreting data. They enable users to build predictive models, classify data, and perform regression analysis with high accuracy. By utilizing these techniques, individuals can gain insights from their data, monitor changes over time, and pinpoint areas of concern with precision.

This book is intended for a wide range of readers who are interested in applying machine learning techniques to solve problems at local, regional, or global levels. Our goal is to equip users with the knowledge and skills needed to carry out effective data analysis projects using XGBoost and decision trees.

The book is structured into two parts, each focusing on key aspects of machine learning:

Chapter 2: XGBoost

- Background on ensemble learning
- Introduction to boosting
- Applications with case studies

Chapter 2: Decision Trees

- Fundamentals of decision trees
- Implementation techniques
- Practical applications and examples

2. XGBOOST OR EXTREME GRADIENT BOOSTING

Machine learning is an independent area of artificial intelligence that focuses on developing algorithms and methodologies that enable computers to gain knowledge from data and use it to make predictions or judgments. It involves the analysis of algorithms that can improve their efficiency over time without needing explicit programming. Before moving into the precise algorithms of XGBoost, it is crucial to have a comprehensive grasp of the fundamental concepts and principles of machine learning. In the upcoming sections, we will examine the fundamental principles of machine learning to gain a deeper comprehension of the XGBoost model.

2.1 What is Ensemble learning?

Ensemble learning is a method in machine learning that combines many learners, to generate more accurate predictions. To clarify, an ensemble model is formed by integrating multiple different models to generate predictions that are more precise than those made by a single model on its own [1].

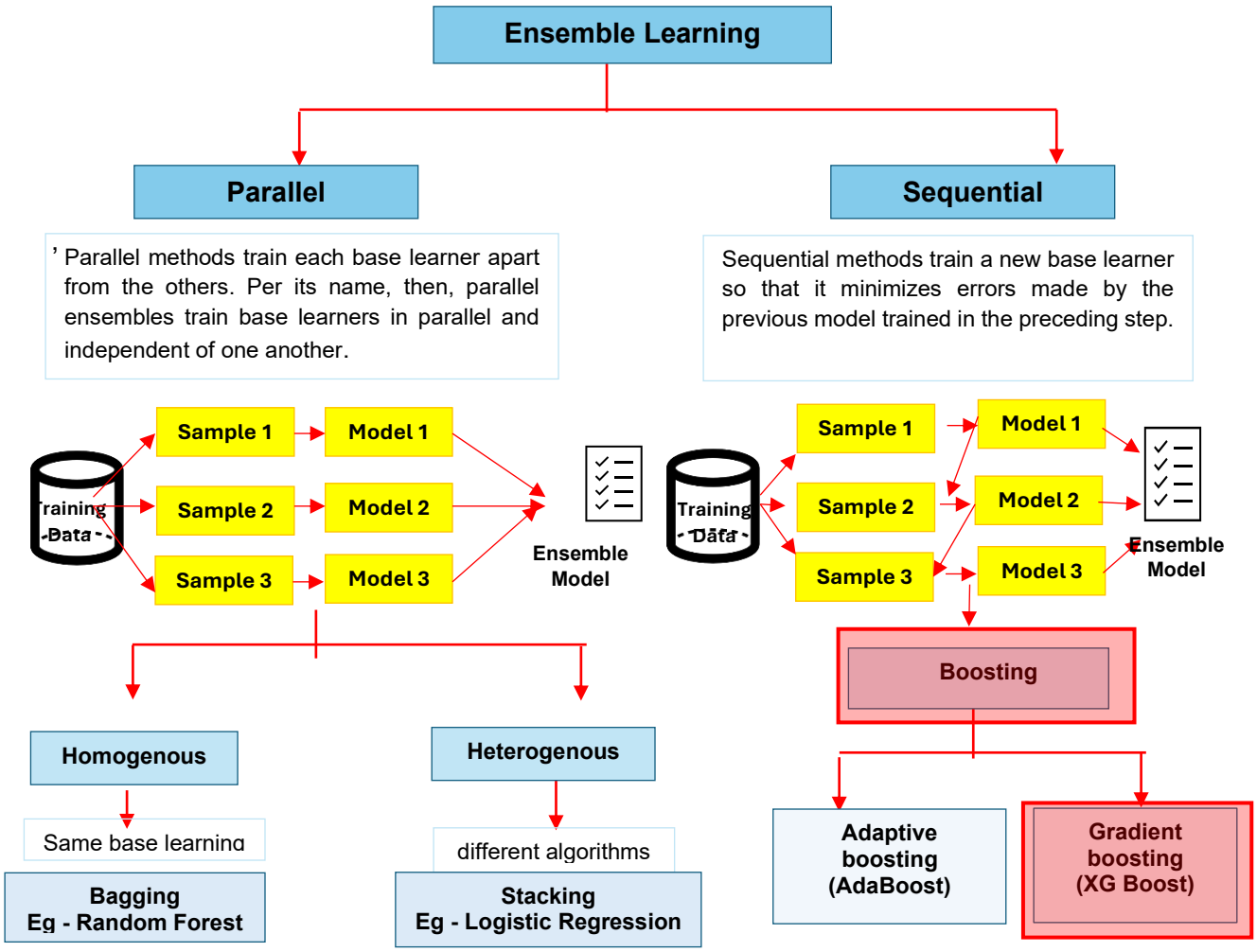


Figure 1- Ensemble Machine Learning

2.2 Boosting

Boosting is a sequential ensemble learning technique that combines multiple weak learners (models that perform slightly better than random guessing) to create a strong learner. It works by sequentially training models, where each new model focuses on correcting the errors made by the previous ones. The final prediction is a weighted sum of the predictions from all the models. Boosting never changes the previous predictor and only corrects the next predictor by learning from mistakes.

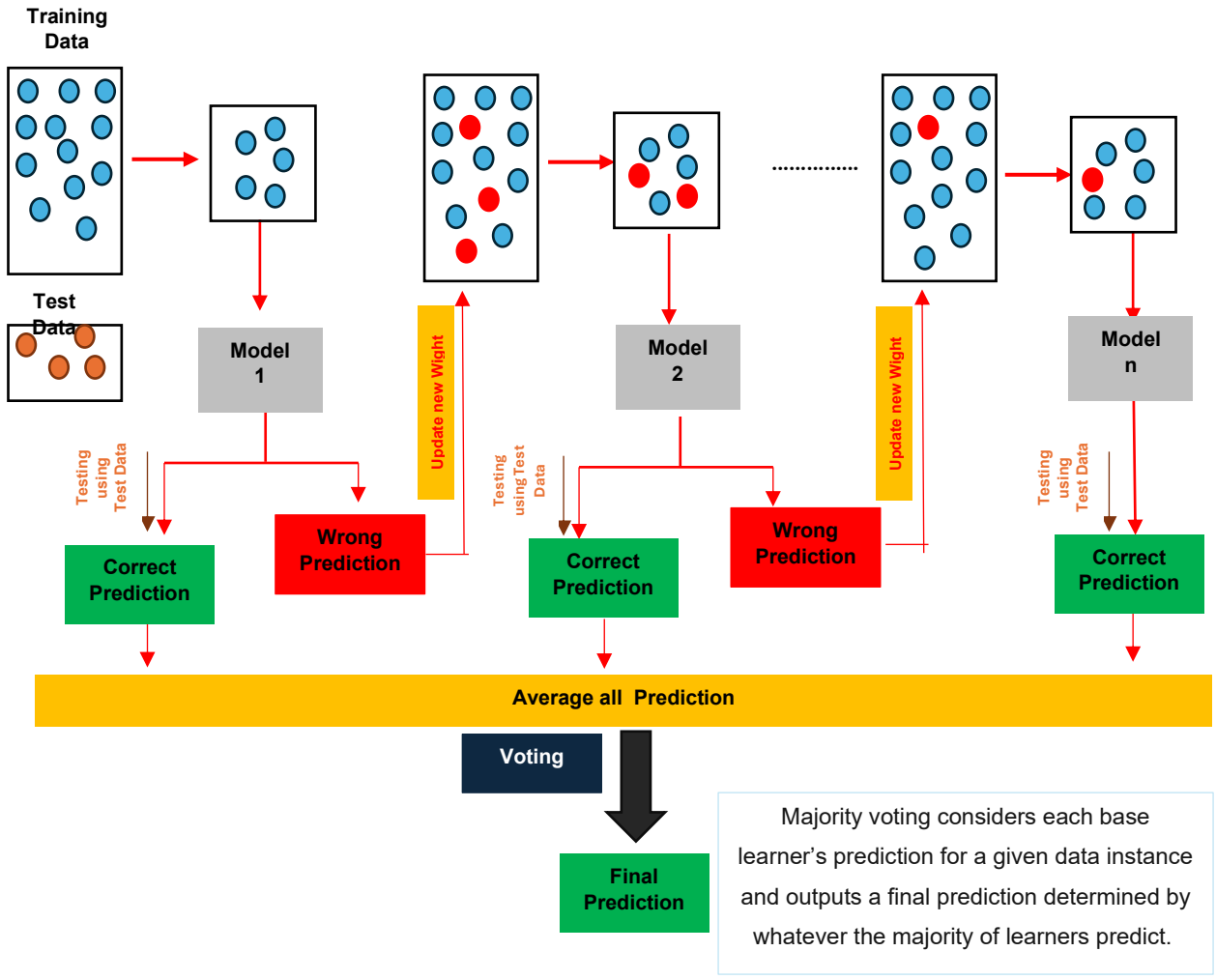


Figure 2 - Boosting Process

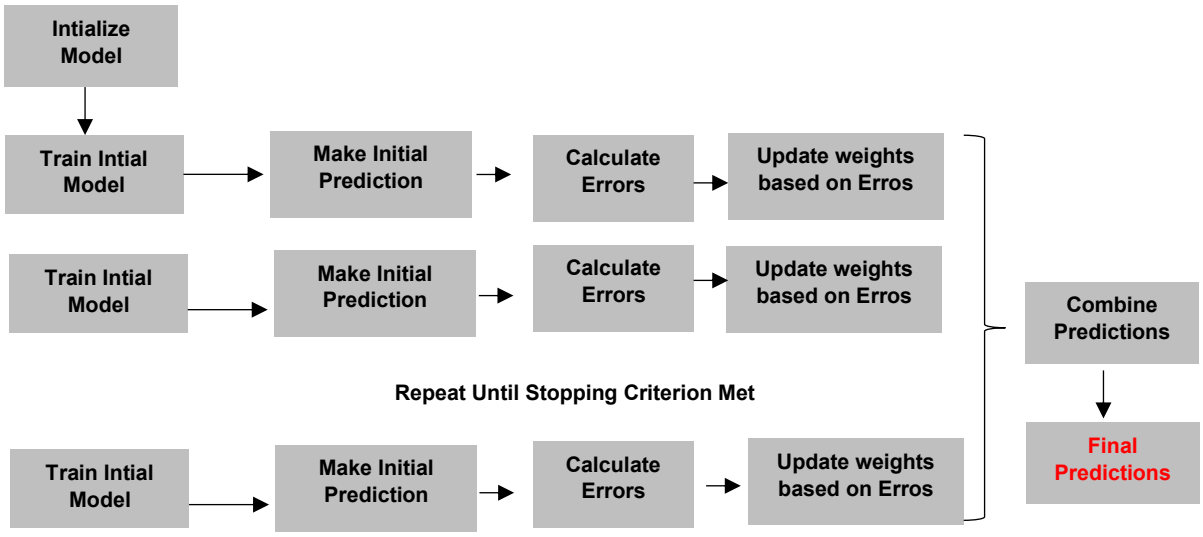


Figure 3 -Summery of Boosting Algorithm system

There are Two of the most prominent boosting methods,

Features	Gradient boosting	Adaptive boosting (AdaBoost)
Training Approach	Models are trained to correct residuals of previous models using gradient descent.	Models are trained to focus on misclassified instances by adjusting weights.
Error Focus	Focuses on minimizing the residual errors of previous models.	Focuses on correctly classifying the hard instances that previous models got wrong.
Combination Method	Combines models by summing their predictions.	Combines models by a weighted vote based on accuracy.
Loss Function	Flexible; can use various loss functions (e.g., MSE, log-loss).	Primarily uses exponential loss function.
Application	Suitable for both regression and classification.	Primarily used for classification (binary and multi-class)
Complexity	More complex, can be computationally intensive, and prone to overfitting.	Simpler, and faster, but can be sensitive to noisy data and outliers.

Table 1- Different Between Boosting Algorithms

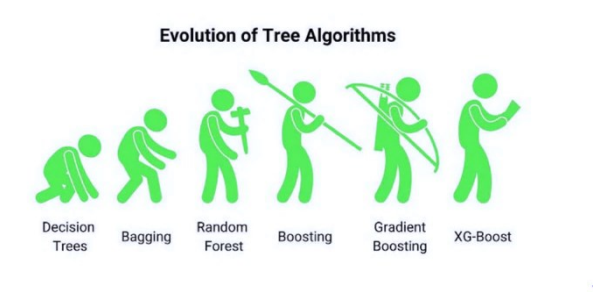
- Adaptive boosting (AdaBoost) weights model errors - AdaBoost adds weights to the previous learner's misclassified samples, causing the next learner to prioritize those misclassified samples.

- Gradient boosting uses residual errors when training new learners - In this way, it attempts to close the gap of error left by one model.

Unfortunately, sklearn contains no pre-defined functions for implementing boosting. The Extreme Gradient Boosting (XGBoost) open-source library, however, provides code for implementing gradient boosting in Python. This study uses XGBoost for its powerful gradient boosting capabilities.

Do You Know?

Decision trees were able to solve both classification and regression problems but suffered from the overfitting issues quickly to tackle this we assemble multiple decision trees with slight modification in data formation it creates begin and random forest. After that researcher thought that assembly transdermally was time consuming and computationally inefficient. Then why not build trees sequentially and improve over those parts where previous trees filled. That is where Boosting came into the picture. Later this boost in algorithm started utilizing the gradient descent algorithm to form trees sequentially and minimize the errors in prediction hence these algorithms are called gradient boosting.



2.2.1 Extreme Gradient Boosting (XGBoost) Characteristics

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

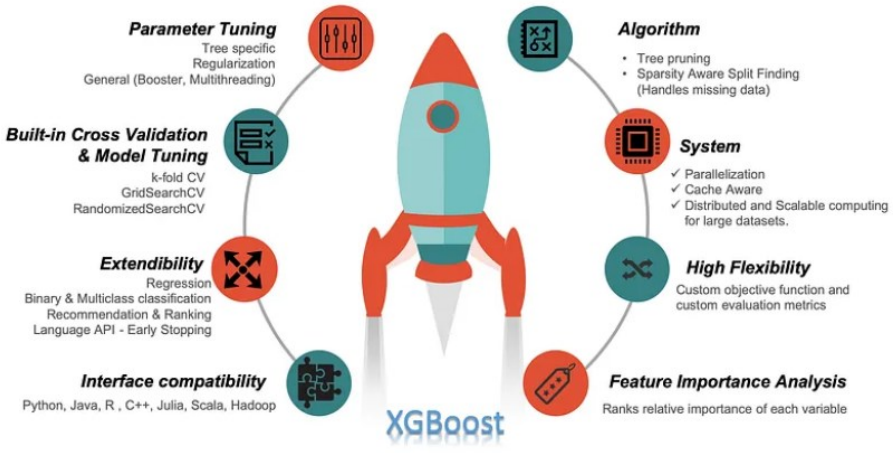


Figure 4 -Extreme Gradient Boosting (XGBoost)

Source- https://medium.com/sfu-cspmp/xgboost-a-deep-dive-into-boosting-f06c9c41349

2.2.2. Application of Extreme Gradient Boosting (XGBoost)

- Data Science Competitions
- Finance -Credit Scoring, Fraud Detection, Stock Market Prediction
- Marketing- Customer Churn Prediction, Customer Segmentation
- Healthcare- Disease Prediction, Medical Imaging, Survival Analysis
- Manufacturing- Predictive Maintenance, Quality Control
- Retail- Sales Forecasting, Inventory Management, Price Optimization
- Text and Speech Analytics-Sentiment Analysis, Speech Recognition
- Energy- Load Forecasting, Fault Detection

2.2.3 Example case study XGBoost: Analyzing Pedestrian Walking Behavior with spatiotemporal Factors

Using machine learning to analyze how spatiotemporal factors influence pedestrian walking behavior patterns. Specifically, the relationship between walking speed and various spatial factors (such as image segmentation factors, isovist factors, and space syntax factors) is modeled using XGBoost. The aim is to understand the influence of these factors on walking speed among university students.

Problem Description:

Dependent Variable (Target): Walking speed of pedestrians.

Independent Variables (Features): Spatial factors, including:

- Image Segmentation Factors: Attributes derived from processing images of the walking environment.
- Isovist Factors: Measures related to the visibility and spatial layout from a specific vantage point.
- Space Syntax Factors: Characteristics of spatial configurations influencing movement and behavior.

By examining the feature importance, you can determine which spatial factors (image segmentation, isovist, space syntax) have the most significant impact on walking speed. This analysis can help urban planners, architects, and researchers understand the key determinants of pedestrian behavior and design more effective and efficient urban spaces.

This approach showcases the power of XGBoost in handling complex, multi-faceted data and extracting valuable insights from it.

Step 01

Google Colab,

Google Colab is a cloud-based platform provided by Google that allows for the creation and sharing of Jupyter notebook files. It offers free access to GPU (Graphical Processing Unit) and TPU (Tensor Processing Unit) resources, making it ideal for running machine learning algorithms and data analysis tasks. And you need to have a google account.

Visual Studio Code (VS Code) with Python Extension, PyCharm, RStudio also can use.

You can access to the google colab through the website https://colab.research.google.com/

To begin coding, click on 'File' in the menu bar, then select 'New notebook' to create a new notebook. Once the notebook is created, you can start writing and executing code by following the provided code snippets below.

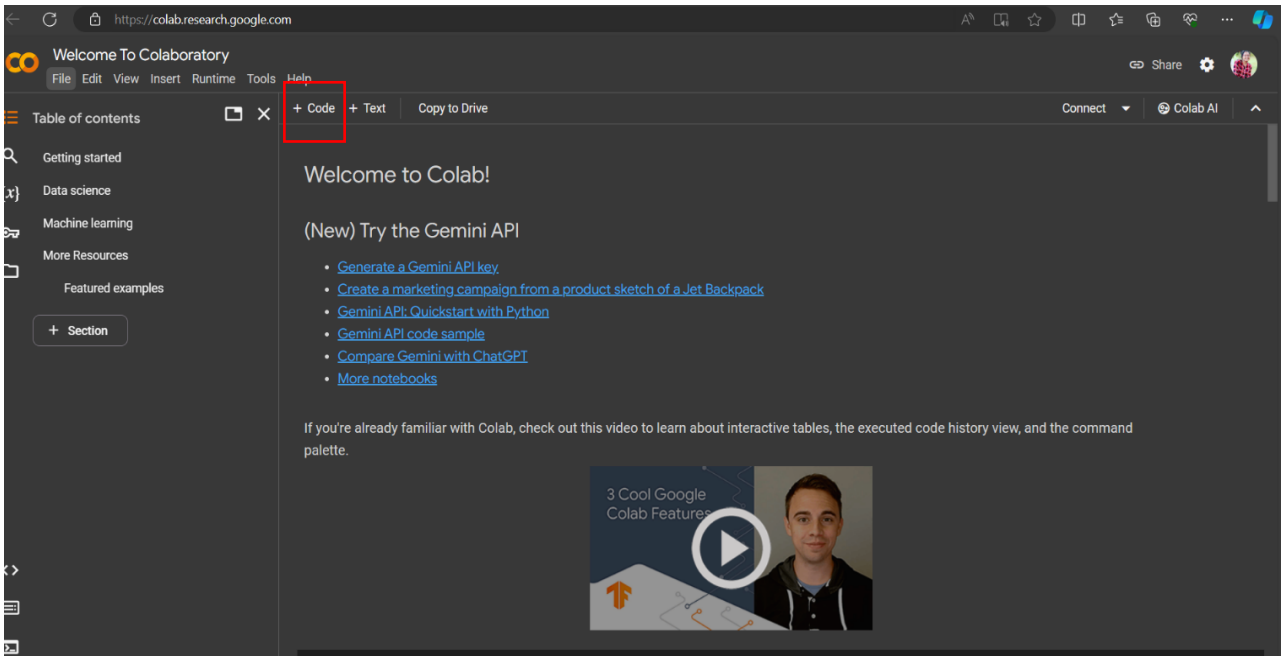


Figure 5 - Google Colab Environment

Editable Code (As per the purpose of the study, user need to change the code

Fixed\Common Code (User can use this code as it is)

Step 02

Data Preprocessing and Initial Model Training:

To import the file (CSV) you need to upload it into the Google Colab environment as shown in Figure 6 and then right-click and copy the path and paste it.

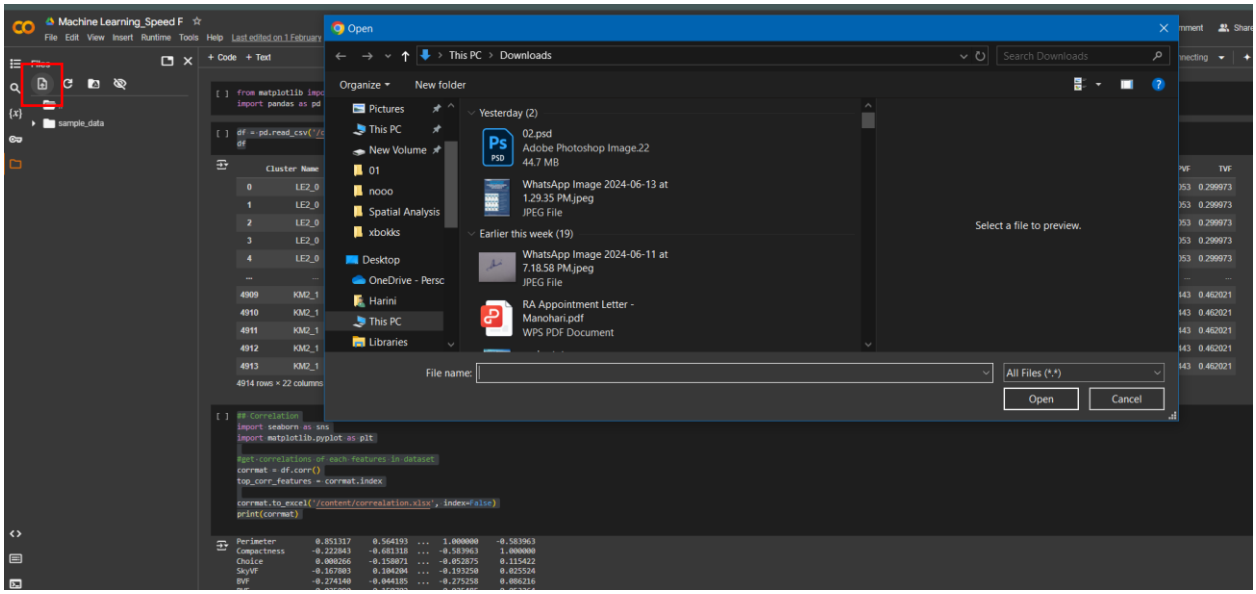


Figure 6 -Import Data

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

# Read the CSV file into a DataFrame
df = pd.read_csv('/content/All Points.csv')
df
```

This section imports necessary libraries (matplotlib.pyplot) for plotting and “pandas” for data manipulation) and reads a CSV file ('All Points.csv') into a Pandas DataFrame “df”.

Step 03

Correlation Calculation :

- This part calculates the correlation matrix (corrmat) for the DataFrame df using df.corr() from Seaborn (sns) and Matplotlib (plt) libraries.
- Saving to Excel: It saves the correlation matrix to an Excel file named 'correlation.xlsx' in the /content/ directory.
- Top Correlated Features: top_corr_features contains the indices of the top correlated features based on the correlation matrix

```
## Correlation
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate correlation matrix
corrmat = df.corr()
top_corr_features = corrmat.index

# Save correlation matrix to an Excel file
corrmat.to_excel('/content/correlation.xlsx', index=False)
print(corrmat)
```

.

Step 04

Feature Selection :

This snippet selects the features (X) and the target variable (Y) from the DataFrame df.

- X contains columns in CSV from index 3 to 23 assuming these are the features (Spatial Factors).
- Y contains a column at index 2, assuming this is the target variable (Walking speed)

```
from numpy import int64

# Selecting features (X) and target variable (Y)
X = df.iloc[:, 3:23]  # Assuming columns 3 to 23 are the features
Y = df.iloc[:, 2]     # Assuming column 2 is the target variable

print(X.head())
```

XGBoost Model Training and Evaluation :

- Binary Conversion: Y_binary_xgb converts the target variable Y to binary based on whether each value is greater than the mean of Y.
- Model Initialization: xgb_classifier initializes an XGBoost classifier (XGBClassifier).
- Train-Test Split: The data (X and Y_binary_xgb) is split into training (X_train_xgb, Y_train_xgb) and test sets (X_test_xgb, Y_test_xgb).
- Model Training and Prediction: The XGBoost classifier is trained (xgb_classifier.fit) on the training data and used to predict (xgb_classifier.predict) outcomes on the test set.
- Evaluation Metrics: Metrics like accuracy, precision, recall, F1-score, and AUC score are calculated to evaluate the model's performance on the test set.

```
# Convert target variable to binary for classification
Y_binary_xgb = (Y > Y.mean()).astype(int)

# Assuming XGBClassifier
xgb_classifier = xgboost.XGBClassifier()

# Split the data into training and test sets
X_train_xgb, X_test_xgb, Y_train_xgb, Y_test_xgb = train_test_split(X, Y_binary_xgb, test_size=0.2)

# Fit the XGBoost model
xgb_classifier.fit(X_train_xgb, Y_train_xgb)

# Make predictions on the test set
y_pred_xgb = xgb_classifier.predict(X_test_xgb)

# Evaluate the model
accuracy_xgb = accuracy_score(Y_test_xgb, y_pred_xgb)
precision_xgb = precision_score(Y_test_xgb, y_pred_xgb)
recall_xgb = recall_score(Y_test_xgb, y_pred_xgb)
f1_xgb = f1_score(Y_test_xgb, y_pred_xgb)
roc_auc_xgb = roc_auc_score(Y_test_xgb, xgb_classifier.predict_proba(X_test_xgb)[:, 1])
```

```
# Print the evaluation metrics
print("Accuracy (XGBoost):", accuracy_xgb)
print("Precision (XGBoost):", precision_xgb)
print("Recall (XGBoost):", recall_xgb)
print("F1-score (XGBoost):", f1_xgb)
print("AUC score (XGBoost):", roc_auc_xgb)
```

Step 06
Hyperparameter Tuning with RandomizedSearchCV :

Hyperparameter Grid (params): Defines a dictionary params containing different values for hyperparameters like learning rate, max depth, min child weight, gamma, and colsample by tree.

RandomizedSearchCV Initialization: RandomizedSearchCV is initialized with a classifier (an XGBRegressor), params, and other parameters for hyperparameter tuning.

Hyperparameter Tuning Loop: Continuously trains models with different sets of hyperparameters until either the R-squared score (r2) reaches 0.9 or the maximum iterations (max_iterations) is reached.

Model Evaluation: For each iteration, it prints the best parameters found by RandomizedSearchCV, the time taken for tuning, and evaluation metrics like RMSE, R-squared, and MAE for the best model.

Parameters :

- **learning_rate:**

Purpose: Controls the step size shrinkage during each boosting iteration. Lower values make the model more robust by shrinking the weights on each step.

- **max_depth:**

Purpose: Maximum depth of a tree. Increasing this value makes the model more complex and more likely to overfit.

- **min_child_weight:**

Purpose: Minimum sum of instance weight (hessian) needed in a child. Higher values prevent the model from learning relationships that might be highly specific to the particular sample selected for a tree.

- **gamma:**

Purpose: Minimum loss reduction required to make a further partition on a leaf node of the tree. Higher values lead to fewer splits.

- **colsample_bytree:**

Purpose: Fraction of features to consider when constructing each tree. A lower value can prevent overfitting by introducing randomness.

Parameter	Scikit-learn Alias	Recommended Range
eta	learning_rate	[0.01, 0.3]
max_depth	max_depth	[3, 10]
min_child_weight	min_child_weight	[1, 10]
gamma	gamma	[0, 0.5]
subsample	subsample	[0.5, 1]
colsample_bytree	colsample_bytree	[0.5, 1]
colsample_bylevel	colsample_bylevel	[0.5, 1]
lambda	reg_lambda	0, 10
alpha	reg_alpha	[0, 10]
num_boost_round	n_estimators	[100, 1000]
scale_pos_weight	scale_pos_weight	[1, 10]
base_score	base_score	Typically 0.5
objective	objective	Depends on the task
booster	booster	gbtree
tree_method	tree_method	auto'

Table 2 -Main Parameters

```
# Define hyperparameter grid for RandomizedSearchCV
params = {
    "learning_rate": [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
    "max_depth": [3, 4, 5, 6, 8, 10, 12, 15],
    "min_child_weight": [1, 3, 5, 7],
    "gamma": [0.0, 0.1, 0.2, 0.3, 0.4],
    "colsample_bytree": [0.3, 0.4, 0.5, 0.7]
}
# Initialize XGBRegressor and RandomizedSearchCV
classifier = xgboost.XGBRegressor()
random_search = RandomizedSearchCV(classifier, param_distributions=params, n_iter=5, scoring='accuracy', n_jobs=-1, cv=3, verbose=3)


# Hyperparameter tuning loop
r2 = 0
max_iterations = 20
iteration = 0
while r2 < 0.9 and iteration < max_iterations:
    # Split the data into training and test sets
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

```
# Fit RandomizedSearchCV to find best hyperparameters
    start_time = datetime.now()
    random_search.fit(X_train, Y_train)
    end_time = datetime.now()

   # Print the best parameters and time taken
    print("Best estimator:", random_search.best_estimator_)
    print("Best params:", random_search.best_params_)
    print("Time taken:", end_time - start_time)

    # Fit the model with best hyperparameters
    xgb = XGBRegressor(**random_search.best_params_)
    xgb.fit(X_train, Y_train)

    # Make predictions on the test set
    y_pred = xgb.predict(X_test)

    # Calculate evaluation metrics
    rmse = np.sqrt(mean_squared_error(Y_test, y_pred))
    r2 = r2_score(Y_test, y_pred)
    mae = mean_absolute_error(Y_test, y_pred)

    # Print evaluation metrics
    print("RMSE:", rmse)
    print("R-squared:", r2)
    print("MAE:", mae)

    iteration += 1

print("Finished after", iteration, "iterations.")
```

Step 07

Further XGBoost Model Training and Evaluation :

Model Initialization: regressor initializes an XGBRegressor with specific hyperparameters (learning_rate, max_depth, min_child_weight, gamma, colsample_bytree).

Model Training and Prediction: It fits (regressor.fit) the model on the training data (X_train, Y_train) and makes predictions (regressor.predict) on the test set (X_test).

```
# Assuming XGBRegressor and the same hyperparameters
regressor = xgboost.XGBRegressor(base_score=None, booster='gbtree', colsample_bylevel=None,
                                 colsample_bytree=0.7, gamma=0.3, learning_rate=0.05,
                                 max_delta_step=None, max_depth=8, min_child_weight=3,
                                 n_estimators=None, n_jobs=None, nthread=None,
                                 objective=None, random_state=None, reg_alpha=None,
                                 reg_lambda=None, scale_pos_weight=None, seed=None, silent=None,
                                 subsample=None, missing=np.nan)  # Set missing parameter


# Fit the model with training data
regressor.fit(X_train, Y_train)
# Make predictions on the test set
predictions = regressor.predict(X_test)
```

Step 08

Cross-validation and Evaluation Metrics :

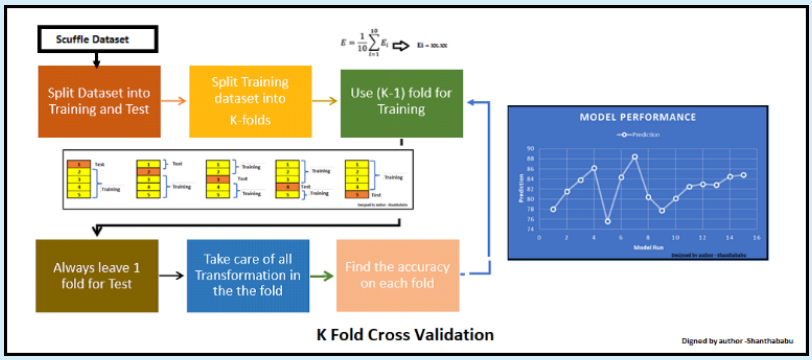
This snippet performs cross-validation (cross_val_score) with 5 folds (cv=5) on the regressor model using training data

```
from sklearn.model_selection import cross_val_score
# Perform cross-validation
score = cross_val_score(regressor, X_train, Y_train, cv=5)
# Print cross-validation scores and mean score
print(score)
print(score.mean())
```

Cross validation -

Cross validation is a statistical method to evaluate machine learning models on unseen data. It comes in handy when the dataset is limited and prevents overfitting by not taking an independent sample (holdout) from training data for validation. By reducing the size of training data, we are compromising with the features and patterns hidden in the data which can further induce errors in our model.

k-fold Cross-validation — In k-fold cross-validation, data is shuffled and divided into k equal sized subsamples. One of the k subsamples is used as a test/validation set and remaining (k -1) subsamples are put together to be used as training data. Then we fit a model using training data and evaluate it using the test set. This process is repeated k times so that every data point stays in validation set exactly once. The k results from each model should be averaged to get the final estimation. The advantage of this method is that we significantly reduce bias, variance, and also increase the robustness of the model.



Source - https://medium.com/sfu-cspmp/xgboost-a-deep-dive-into-boosting-f06c9c41349

Step 09
Evaluating XGBoost Regressor :

This snippet performs cross-validation (cross_val_score) with 5 folds (cv=5) on the regressor model using training data

- Imports: Import necessary libraries and functions (mean_squared_error, r2_score, mean_absolute_error, XGBRegressor, numpy).
- Fitting the Model: regressor.fit(X_train, Y_train) trains the XGBoost regressor model on the training data (X_train, Y_train).
- Prediction: y_pred = regressor.predict(X_test) makes predictions on the test data (X_test).

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from xgboost import XGBRegressor
import numpy as np
regressor.fit(X_train, Y_train)
# Predict on the test set
y_pred = regressor.predict(X_test)

# Calculate the RMSE
rmse = np.sqrt(mean_squared_error(Y_test, y_pred))
print("RMSE:", rmse)

# Calculate the R-squared
r2 = r2_score(Y_test, y_pred)
print("R-squared:", r2)

# Calculate the MAE
mae = mean_absolute_error(Y_test, y_pred)
print("MAE:", mae)
```

Common metrics:

- RMSE (Root Mean Squared Error): rmse = np.sqrt(mean_squared_error(Y_test, y_pred)). Measures the average magnitude of the error in predicting numeric values. Lower values indicate better fit.
- R-squared: r2 = r2_score(Y_test, y_pred). Represents the proportion of variance in the dependent variable that is predictable from the independent variables. Values closer to 1 indicate better fit.
- MAE (Mean Absolute Error): mae = mean_absolute_error(Y_test, y_pred). Measures the average absolute difference between predicted and actual values. Similar to RMSE but gives equal weight to all errors.

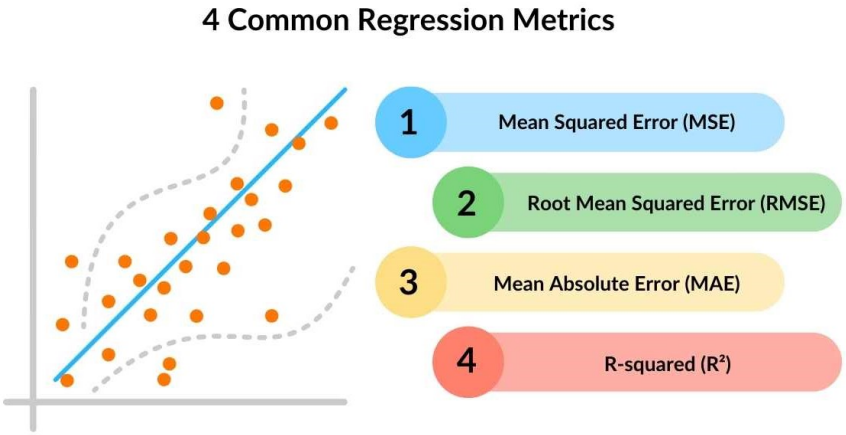


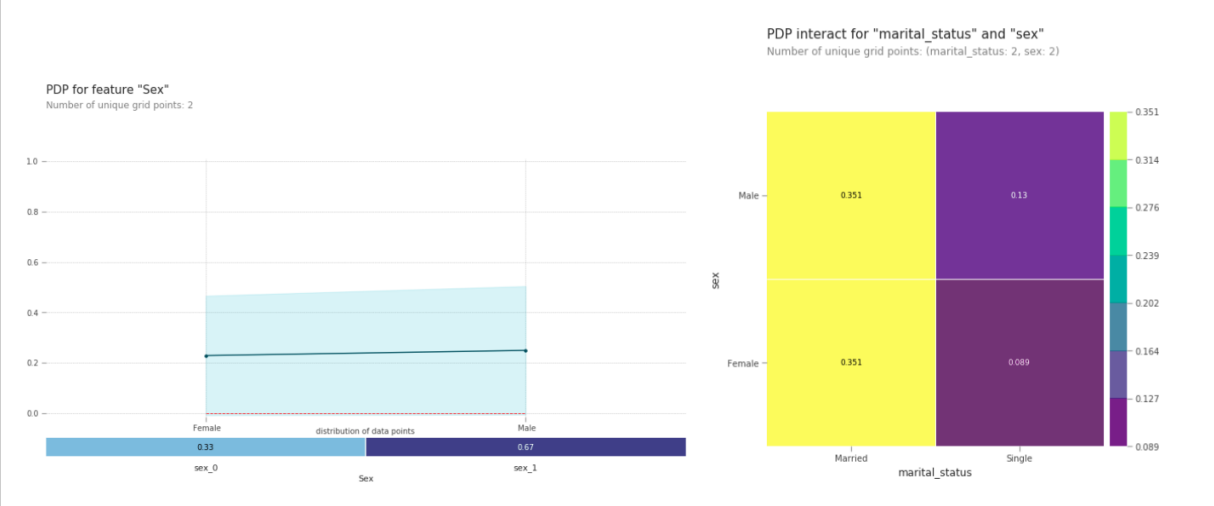
Figure 7 – Metrics

Source - https://spotintelligence.com/2024/03/27/regression-metrics-for-machine-learning/

Step 10
Feature Importance Analysis :

What is PDP Plot

The partial dependence plot (short PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a machine learning model (J. H. Friedman 200130). A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic or more complex.



Import plotly. express, pandas, and XGBRegressor.

- Get Feature Importances: importance regressor.get_booster().get_score(importance_type='gain') retrieves feature importance scores based on gain.
- Normalize Importances: Normalizes importance scores to range from 0 to 100.
- Convert to DataFrame: Converts the dictionary of feature importances into a pandas DataFrame (imp_df).
- Save to Excel: Saves the DataFrame to an Excel spreadsheet named feature_importance.xlsx.
- Plot with Plotly: Uses Plotly to create a horizontal bar plot (px.bar) showing feature importances (imp_df).

```
import plotly.express as px
import pandas as pd
from xgboost import XGBRegressor
# Assuming you have already trained your XGBoost model and named it regressor
# Get feature importances
importance = regressor.get_booster().get_score(importance_type='gain')

import plotly.express as px
import pandas as pd
from xgboost import XGBRegressor
# Assuming you have already trained your XGBoost model and named it regressor
# Get feature importances
importance = regressor.get_booster().get_score(importance_type='gain')

# Normalize the importance values to 0-100
importance_sum = sum(importance.values())
importance_normalized = {key: value / importance_sum * 100 for key, value in importance.items()}
```

```
# Convert to DataFrame
imp_df = pd.DataFrame.from_dict(importance_normalized, orient='index', columns=['importance'])
imp_df = imp_df.reset_index().rename(columns={'index': 'feature'})
imp_df = imp_df.sort_values('importance', ascending=False)
# Save the dataframe to an Excel spreadsheet
imp_df.to_excel('/content//feature_importance.xlsx', index=False)

# Plot with Plotly
fig = px.bar(imp_df, x='importance', y='feature', orientation='h', title='Feature Importance')
fig.show()
```

Output -

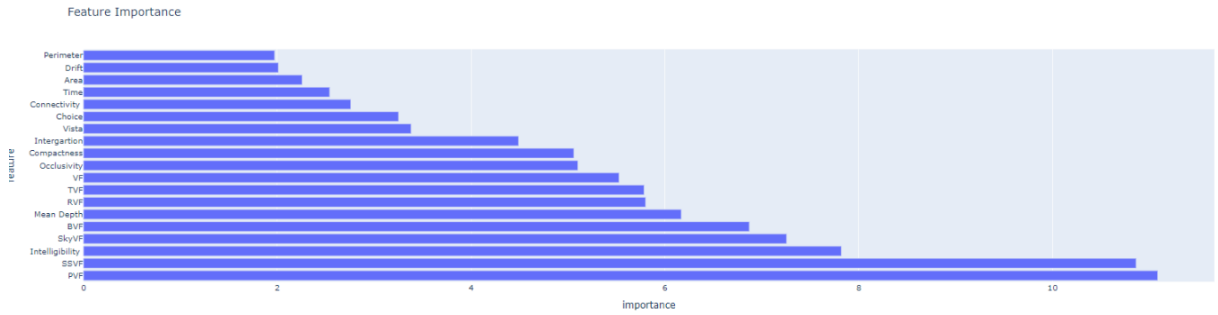


Figure 8 -Output of Feature Import

The bar chart above shows the importance of various features in predicting walking speed. Feature importance indicates how much each feature contributes to the model's predictions. In this context, several features have been considered, ranging from "Perimeter" to "PVF."

Key Features:

PVF (People View Factor): This feature stands out as the most important one in the model. The PVF represents how much of the road is visible to people, which can significantly affect their walking speed.

Step 11

Explanation of Parameters in PDPIsolate :

- model: This is your trained XGBoost regressor model (regressor).
- df: The dataset (X) that was used to train the model. It should contain the features.
- model_features: List of feature names used in the model.
- feature: The specific feature ('BVF') for which you want to plot the PDP.
- feature_name: A descriptive name for the feature being plotted ("Building View Factor")

```
pdp_one = PDPIsolate(
    model=regressor,
    df=X,
    model_features=feature_names,
    feature='TVF',
    feature_name="Tree View Factor",
    n_classes=0,
)
```

Step 12

Parameters in pdp_one.plot() :

- center: Whether to center the plot lines.
- plot_lines: Whether to plot the lines.
- frac_to_plot: Fraction of grid to plot. Set to 100 to plot all points.
- cluster: Whether to cluster the data.
- n_cluster_centers: Number of cluster centers.
- cluster_method: Method to use for clustering.
- plot_pts_dist: Whether to plot points distribution.
- to_bins: Whether to group x values into bins.
- show_percentile: Whether to show percentile.
- which_classes: Which classes to plot. Not used in regression (None).
- figsize: Figure size. Automatically determined or set to None.
- dpi: Dots per inch for figure resolution (1080).
- ncols: Number of columns for subplots (2).
- plot_params: Additional parameters for plotting ({"pdp_hl": True} includes high and low confidence intervals).
- engine: Plotting engine ("matplotlib").
- template: Plotly template for styling ("plotly_white").

```
fig, axes = pdp_one.plot(
    center=True,
    plot_lines=False,
    frac_to_plot=100,
    cluster=False,
    n_cluster_centers=None,
    cluster_method='accurate',
    plot_pts_dist=True,
    to_bins=True,
    show_percentile=True,
    which_classes=None,
    figsize=None,
    dpi=1080,
    ncols=2,
    plot_params={"pdp_hl": True},
    engine="matplotlib",
    template="plotly_white",
)
```

Output :

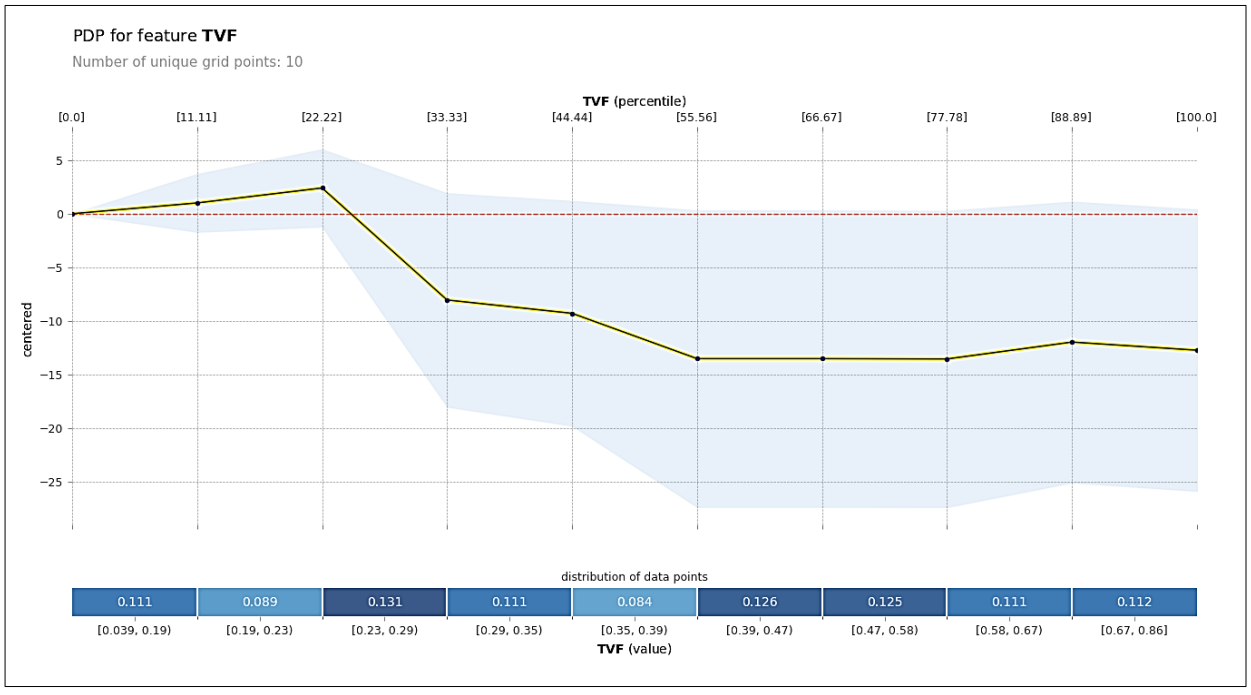


Figure 9 -PD Plot

The analysis of the Partial Dependence (PD) plot focuses on the Tree View Factor (TVF) and its influence on walking speed, revealing a significant non-linear relationship. According to the graphic, as the TVF value increases, the change in the speed values becomes negative.

Figure 10 depicts the raw values of the speed. It is important to note that while the speed decreases as the TVF increases, the actual speed values vary within the same range from 100 to 120 which is the normal range. This variety shows that although there is a noticeable pattern of decreased speeds as TVF increases, the effect could vary within the same range.

Step 13

Explanation of Parameters in PDPInteract :

- model: Your trained XGBoost regressor model (regressor).
- df: The dataset (X) that was used to train the model. It should contain the features.
- model_features: List of feature names used in the model.
- n_classes: Number of classes for classification tasks. Since you're doing regression, it's set to 0.
- features: List of features for which you want to plot the interaction (['BVF', 'Time']).
- feature_names: List of descriptive names for the features being plotted (['Building View Factor', 'Time']).

```
pdp_inter = PDPInteract(
    model=regressor,
    df=X,
    model_features=feature_names,
    n_classes=0,
    features=['Choice', 'Time'],
    feature_names=[Choice, 'Time'],
)
```

Step 14

Parameters in pdp_inter.plot() :

- plot_type: Type of plot to generate ("contour" in this case for contour plot).
- to_bins: Whether to group x values into bins.
- plot_pdp: Whether to plot partial dependence plots (PDPs) for individual features.
- show_percentile: Whether to show percentile (not used here).
- which_classes: Which classes to plot. Not used in regression (None).
- figsize: Figure size. Automatically determined or set to None.
- dpi: Dots per inch for figure resolution (1080).
- ncols: Number of columns for subplots (2).

- plot_params: Additional parameters for plotting (None).
- engine: Plotting engine ("plotly").
- template: Plotly template for styling ("plotly_white").

```
fig, axes = pdp_inter.plot(
    plot_type="contour",
    to_bins=True,
    plot_pdp=True,
    show_percentile=False,
    which_classes=None,
    figsize=None,
    dpi=1080,
    ncols=2,
    plot_params=None,
    engine="plotly",
    template="plotly_white",
)
```

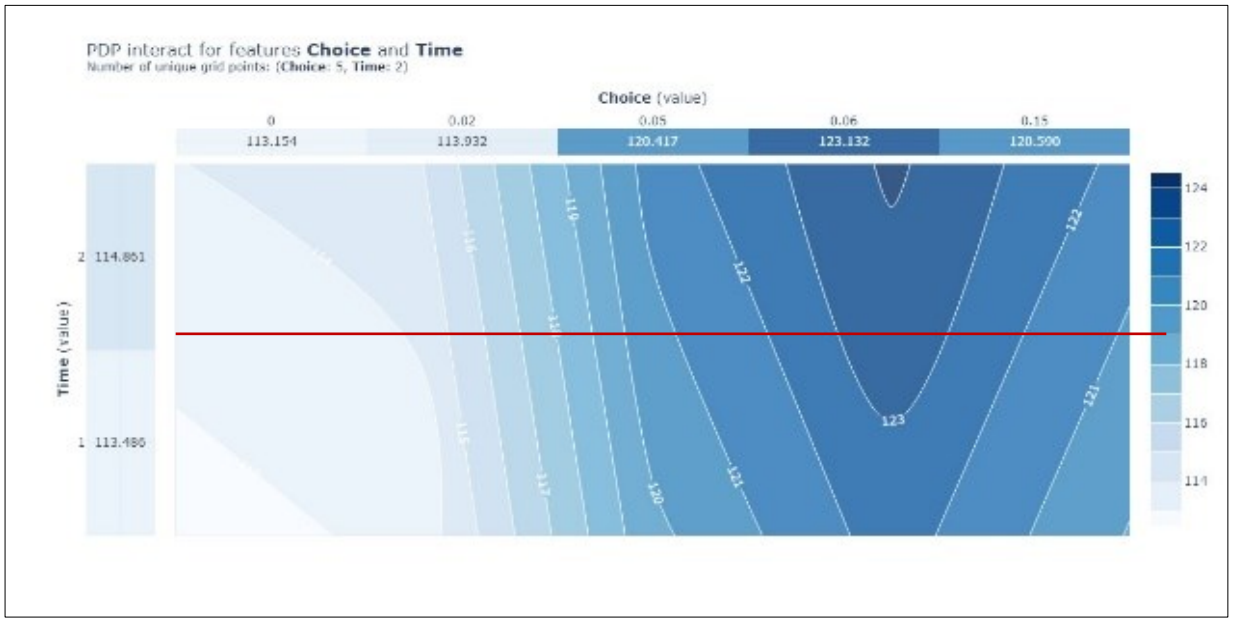


Figure 10 -Pd plot

The figure shows that tree cover did not change the speed of walking with temporal factors.

X-Axis (Time): Represents the values of the Time (Morning 1 and Evening 2). Y-Axis (choice) T’his could refer to the number of route choices or options available to pedestrians.

Color Gradient: Indicates the predicted values of the target variable. The color scale ranges from lighter shades (lower values) to darker shades (higher values).

When the Choice value is around 0.6 and time is approximately 114.600, the target value is at its peak, indicated by the darkest shade of blue. This suggests a strong positive effect when there are more choices available at this specific time. hen the Choice value is low (around 0.2-0.3), the target values are generally lower, indicated by lighter shades. This indicates a less positive effect of having fewer choices.

3. DECISION TREE ANALYSIS

3.1 Introduction

Decision tree analysis is a widely utilized data mining technique for both classification and regression tasks. This method involves developing a model that predicts the value of a target variable by learning decision rules derived from the data features. The WEKA software application excels in transforming extensive databases into decision trees, based on these inferred rules, and identifying potential clusters within the data**.**

ZoneID	Sand Dune	Coral	Mangrove	Lagoon & Canel	Coastal Shape	Vegetation Line	Width of conti. Shelf	Inundation
25	No_influence	Moderate	No_influence	No_influence	No_influence	Low	High	Moderate
26	Very_Low	Moderate	No_influence	No_influence	No_influence	Moderate	High	Low
27	Very_Low	Moderate	No_influence	No_influence	No_influence	No_influence	High	Low
28	Moderate	Moderate	No_influence	Negative_Very_High	Negative_Moderate	Low	High	Low
29	No_influence	Moderate	No_influence	No_influence	Positive_High	Low	High	Low
30	No_influence	High	Very_High	No_influence	Positive_High	Low	High	Low
31	Moderate	Moderate	No_influence	Negative_Very_High	Negative_High	Low	High	Low
32	Moderate	Low	Low	No_influence	Positive_Moderate	Moderate	High	Low
33	Moderate	No_influence	Low	Negative_High	Negative_Low	Moderate	High	Very_Low
34	No_influence	No_influence	No_influence	No_influence	No_influence	High	High	Very_Low
35	No_influence	No_influence	No_influence	No_influence	No_influence	Moderate	High	Very_Low
36	No_influence	No_influence	No_influence	No_influence	Negative_Low	No_influence	High	Low
37	No_influence	No_influence	High	No_influence	No_influence	No_influence	High	Low
38	Low	No_influence	High	No_influence	Positive_High	Very_High	High	Low
39	No_influence	No_influence	High	No_influence	Negative_High	No_influence	High	Low
40	No_influence	No_influence	High	No_influence	Positive_Moderate	Very_High	Moderate	Low
41	No_influence	No_influence	High	No_influence	Negative_Moderate	Very_High	Moderate	Moderate
42	No_influence	No_influence	High	No_influence	Negative_Moderate	Very_High	Moderate	Moderate
43	No_influence	No_influence	High	No_influence	Negative_Moderate	Very_High	Moderate	Moderate
44	No_influence	No_influence	High	No_influence	Negative_Moderate	Very_High	Moderate	Moderate
45	No_influence	No_influence	High	No_influence	Negative_Moderate	No_influence	Moderate	Moderate
46	No_influence	No_influence	High	No_influence	Positive_Low	No_influence	Moderate	Moderate
47	No_influence	No_influence	High	No_influence	Positive_Low	Very_High	Moderate	Moderate
48	No_influence	High	High	No_influence	No_influence	High	High	Low
49	Very_Low	Very_High	Moderate	Negative_High	No_influence	High	High	Moderate

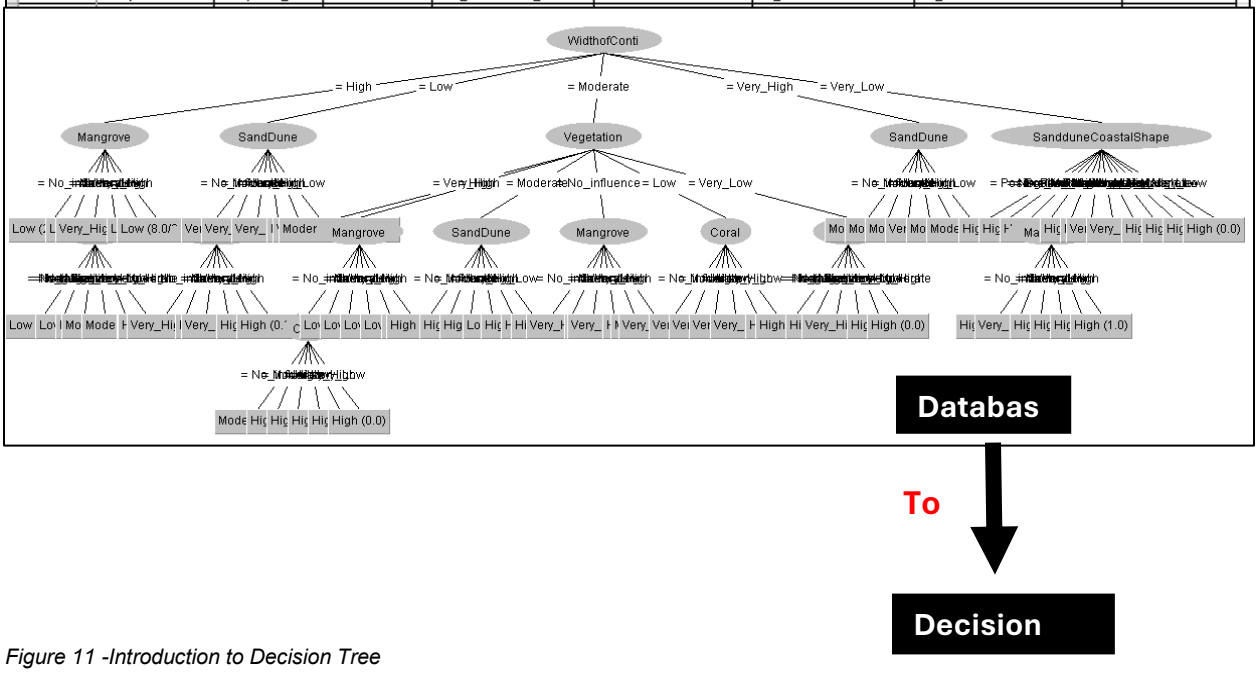


Figure 11 -Introduction to Decision Tree

3.2 Required Applications

Excel

Notpad

WEKA

WEKA Software

Software written in Java, developed at the University of Waikato, New Zealand. It provides a comprehensive collection of tools for data preprocessing, classification, regression, clustering, association rules, and visualization. WEKA features an intuitive graphical user interface (GUI) that makes it accessible for users without extensive programming experience. It includes robust tools for cleaning, transforming, and preparing data, as well as a wide range of algorithms for various machine-learning tasks. WEKA supports model evaluation and validation methods such as cross-validation and offers visualization capabilities to help users understand data patterns. Additionally, it allows for scripting through Java and integration with other languages like Python and is extensible via user-defined plugins. Commonly used in academic research and various industries, WEKA is ideal for data exploration, experimentation, and finding optimal machine learning solutions. Its comprehensive documentation and active community support make it a valuable resource for anyone involved in data mining and machine learning.

3.3 Methodology Workflow

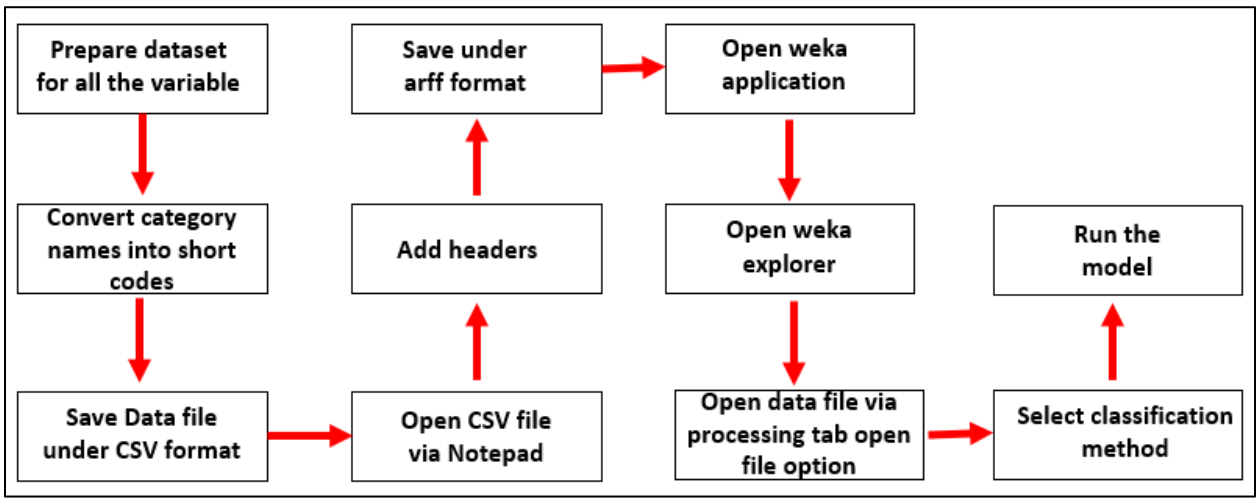
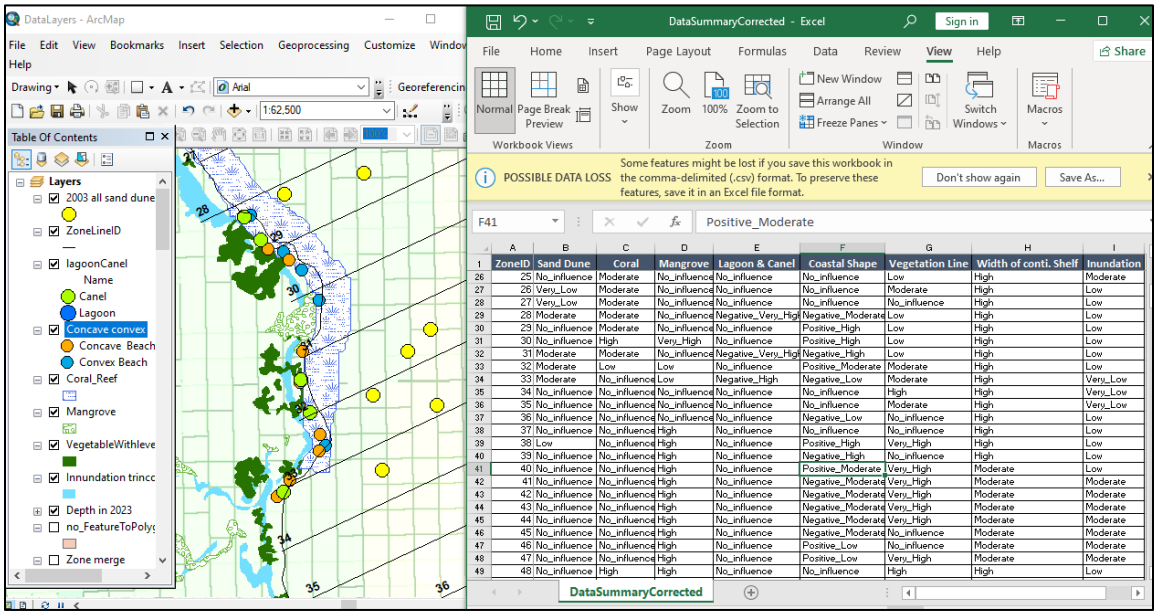


Figure 12 - Methodology

Step 1: Data Preparation

Data preparation is required for each variable. This study has divided the study area into 300 zones, with each zone containing 8 variables. The value for each variable can be either numerical or textual, but every variable must have a value in all zones.



Step 2: Convert the data set into sort codes

After completing data preparation, it is necessary to reduce the character length of the data. For example, the category name "No_influence" is quite long and affects the readability of the decision trees (see figure below). Therefore, all categories should be converted into shortcodes to enhance visibility.

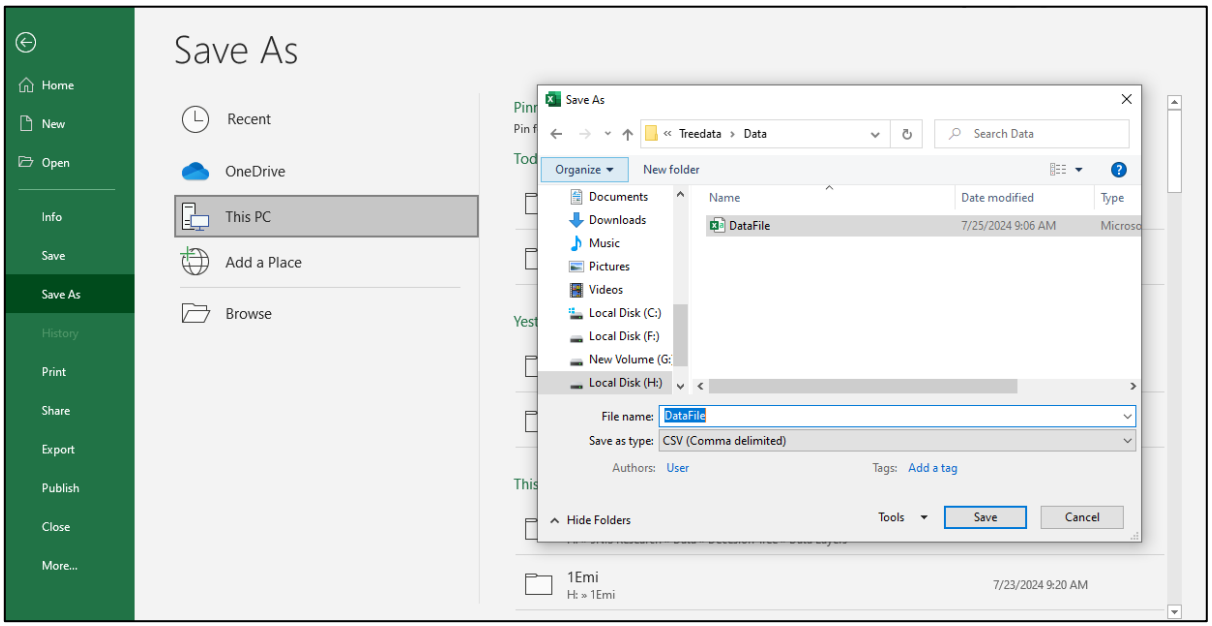
	A	B	C	D	E	F	G	H
1	Sand Dune	Sand Dune	Coral	Coral	Mangrove	Mangrove	Lagoon & Canel	Lagoon & Canel
2	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
3	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
4	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
5	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
6	Moderate	M	No_influence	NI	No_influence	NI	No_influence	NI
7	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
8	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
9	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
10	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
11	No_influence	NI	No_influence	NI	Low	L	No_influence	NI
12	No_influence	NI	No_influence	NI	Moderate	M	Negative_Low	NL
13	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
14	Moderate	M	No_influence	NI	No_influence	NI	No_influence	NI
15	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
16	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
17	Moderate	M	No_influence	NI	No_influence	NI	No_influence	NI
18	No_influence	NI	No_influence	NI	No_influence	NI	No_influence	NI
19	Low	L	No_influence	NI	No_influence	NI	No_influence	NI
20	Very_High	VH	No_influence	NI	Very_Low	VL	Negative_Low	NL

Step 3: After converting long names to shortcodes

	A	B	C	D
1	Sand Dune	Coral	Mangrove	Lagoon & Canel
29	M	M	NI	NVL
30	NI	M	NI	NI
31	NI	H	VH	NI
32	M	M	NI	NVL
33	H	L	H	NI
34	VH	H	L	NL
35	VH	H	NI	NI
36	H	M	H	NI
37	NI	NI	NI	NI
38	NI	NI	H	NI
39	M	NI	H	NI

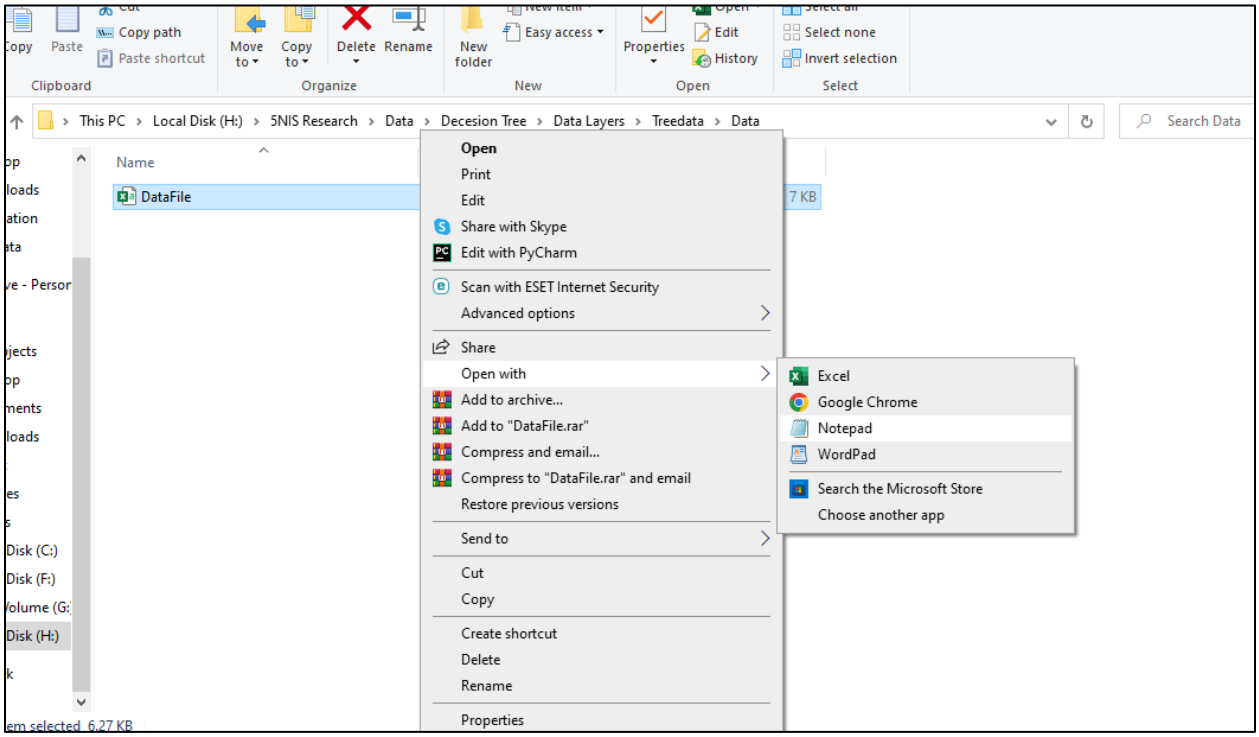
Step 4: Save the Excel file under the CSV file format

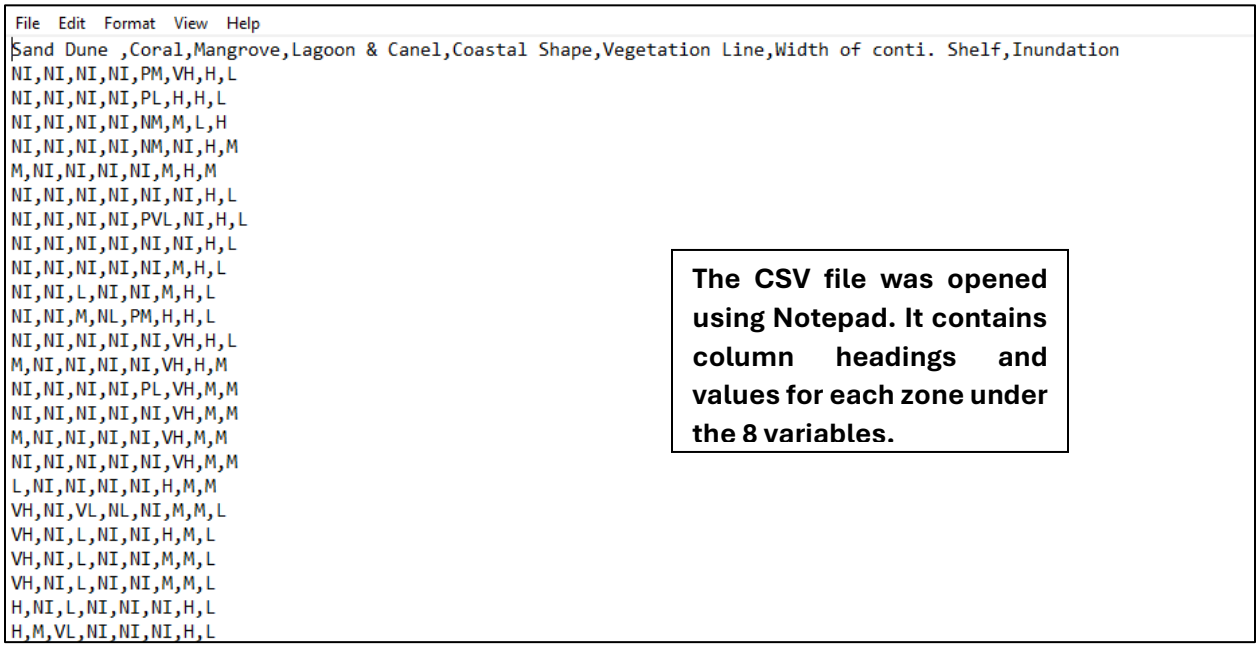
Save the Excel data file under CSV file format. [File tab >>> Save as >>> Browse and select save folder>>> Type file name >>> change save as type to CSV >>> save]



Step 5: Open CSV file via Notepad

Need to open saved CSV file via Notepad [right click on the CSV file >>> open with >>> Notepad]

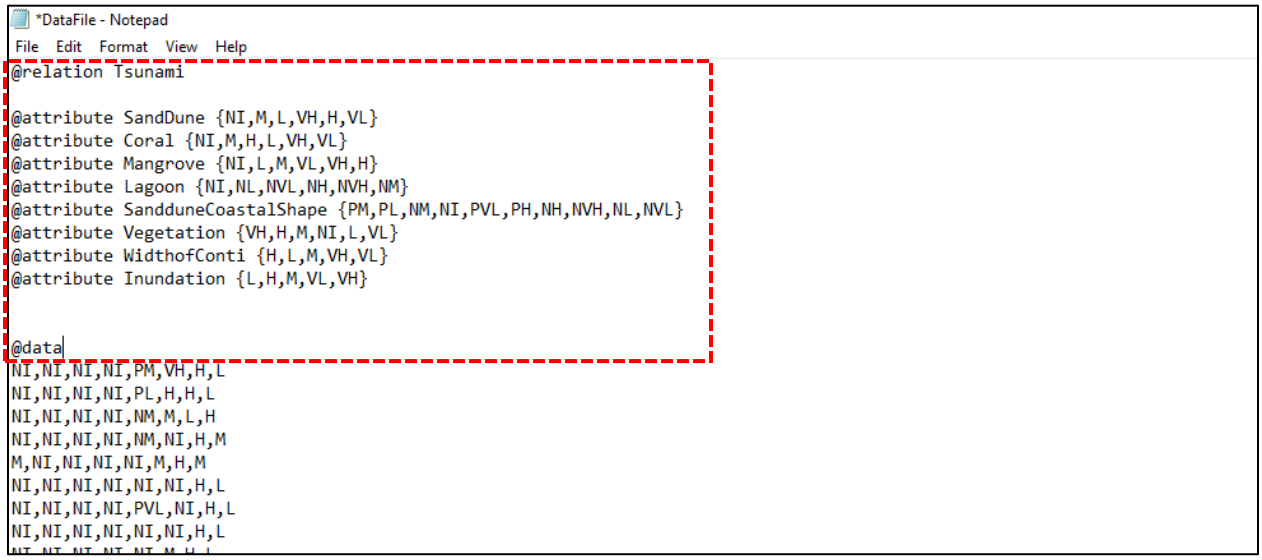




The CSV file was opened using Notepad. It contains column headings and values for each zone under the 8 variables.

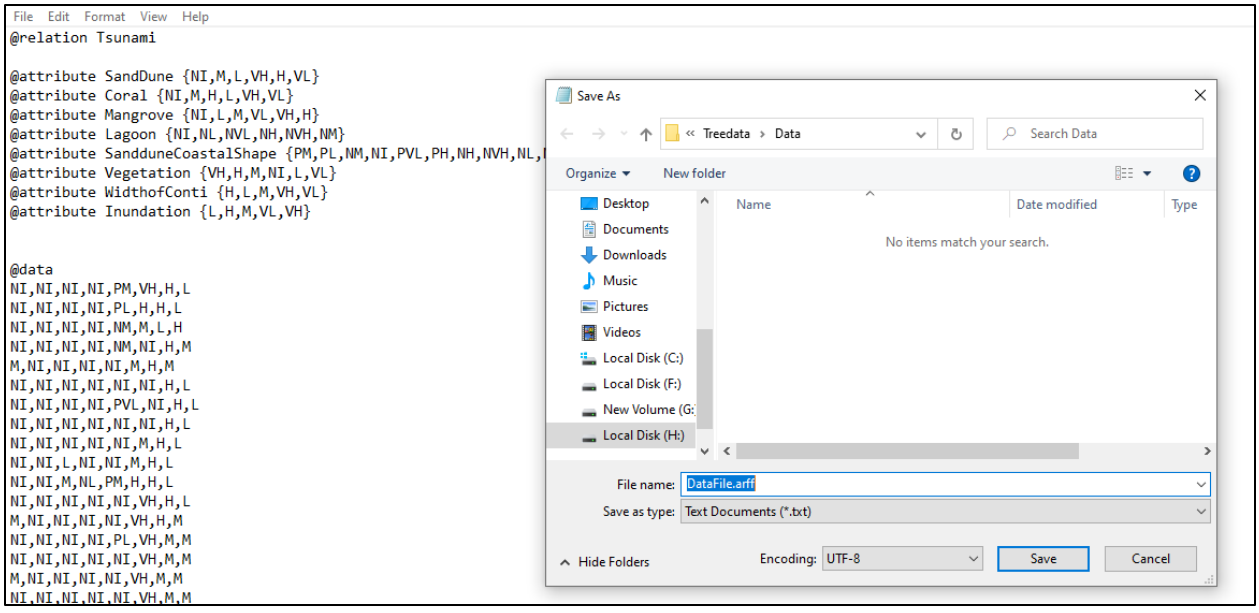
Step 6: assign headings with attribute names

Data column headings need to be included in the file. First, assign the name of the file using @relation. Next, assign the heading for the first variable with @attribute and include its unique values within brackets. Repeat this process for the next variable, assigning its heading and unique values. Follow the same procedure for all 8 variables. Finally, assign the data value area by using the @data code, as illustrated in the figure below.



Step 7: save text [notpad file] under .arff format.

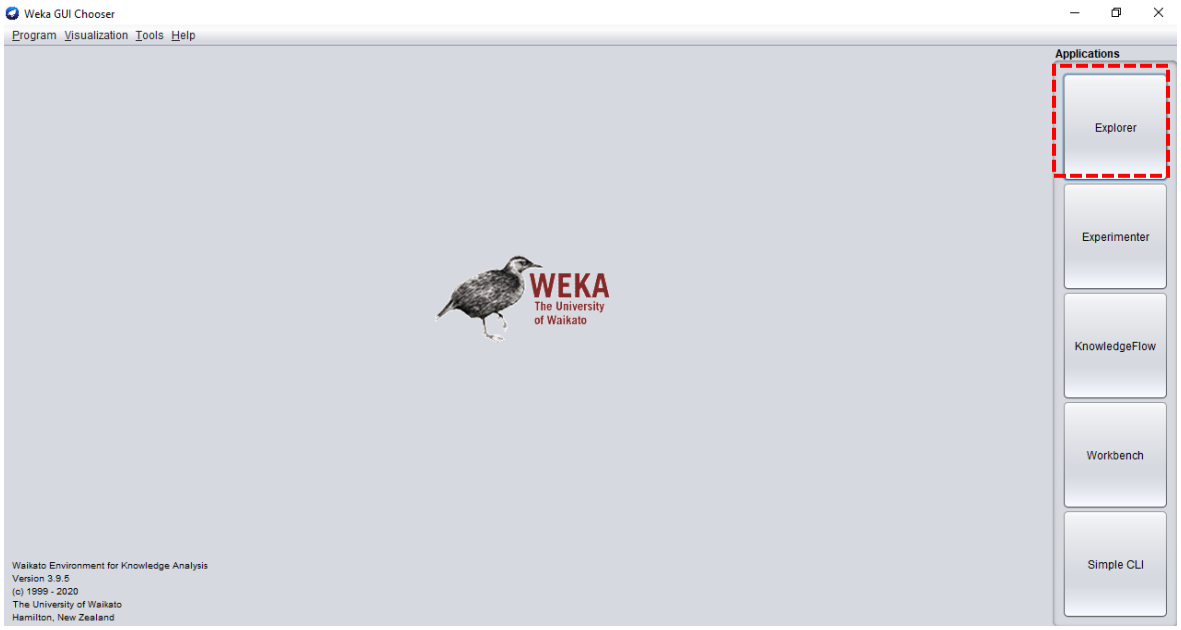
File >>> Save as >>> find save location >>> provide a save file name and at the end need to type .arff >>> click save



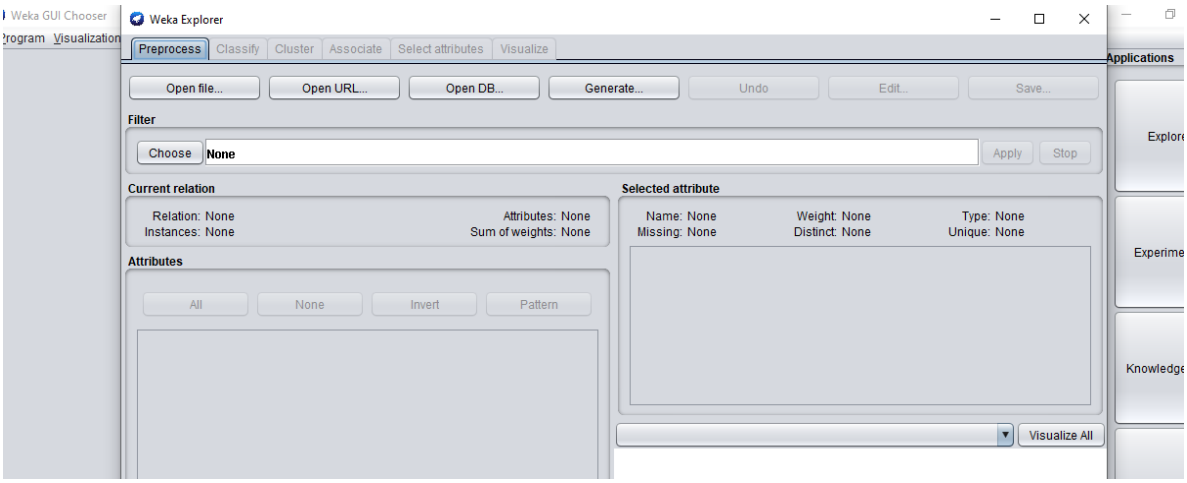
Step 8: Open the WEKA application and click on the Explorer

Download link: https://waikato.github.io/weka-wiki/downloading_weka/

Please visit the above link to download the WEKA application compatible with your computer. Once downloaded, install the application by double-clicking on the file. After installation, open the WEKA software and click on "Explorer".”

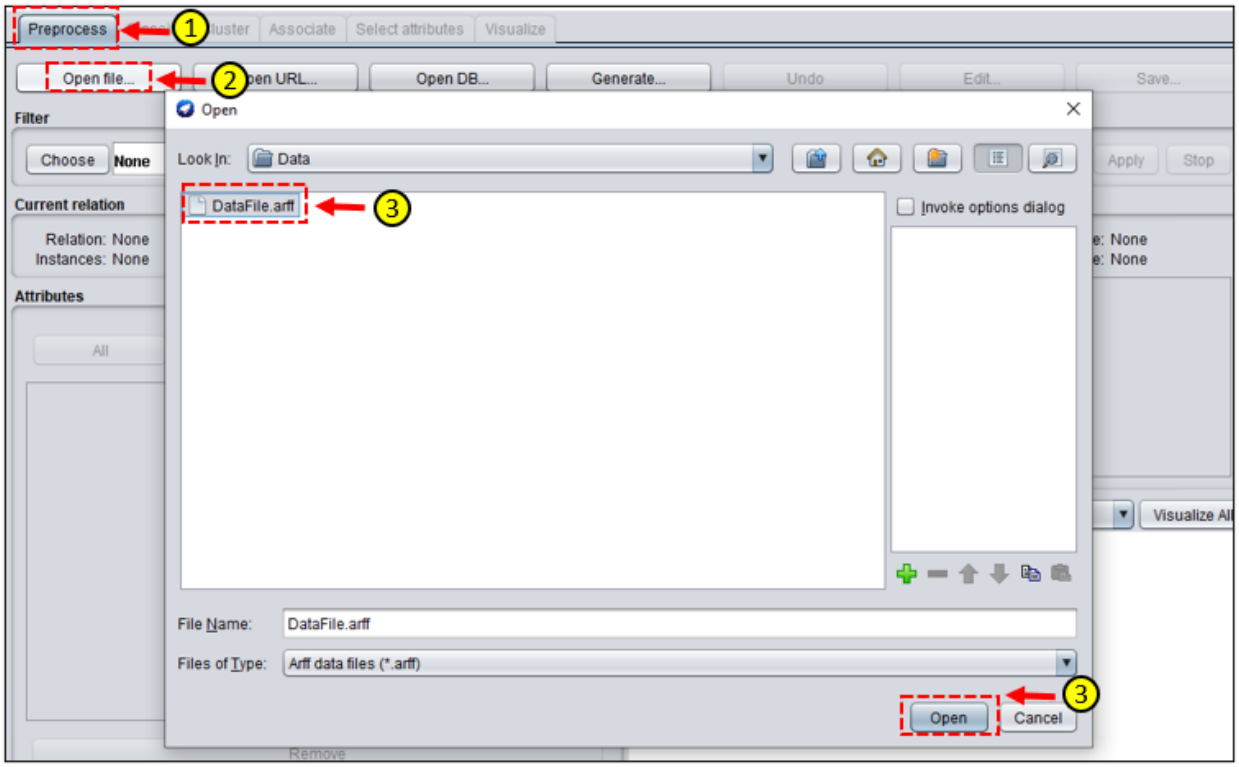


Once you click on "Explorer", the following interface will appear.



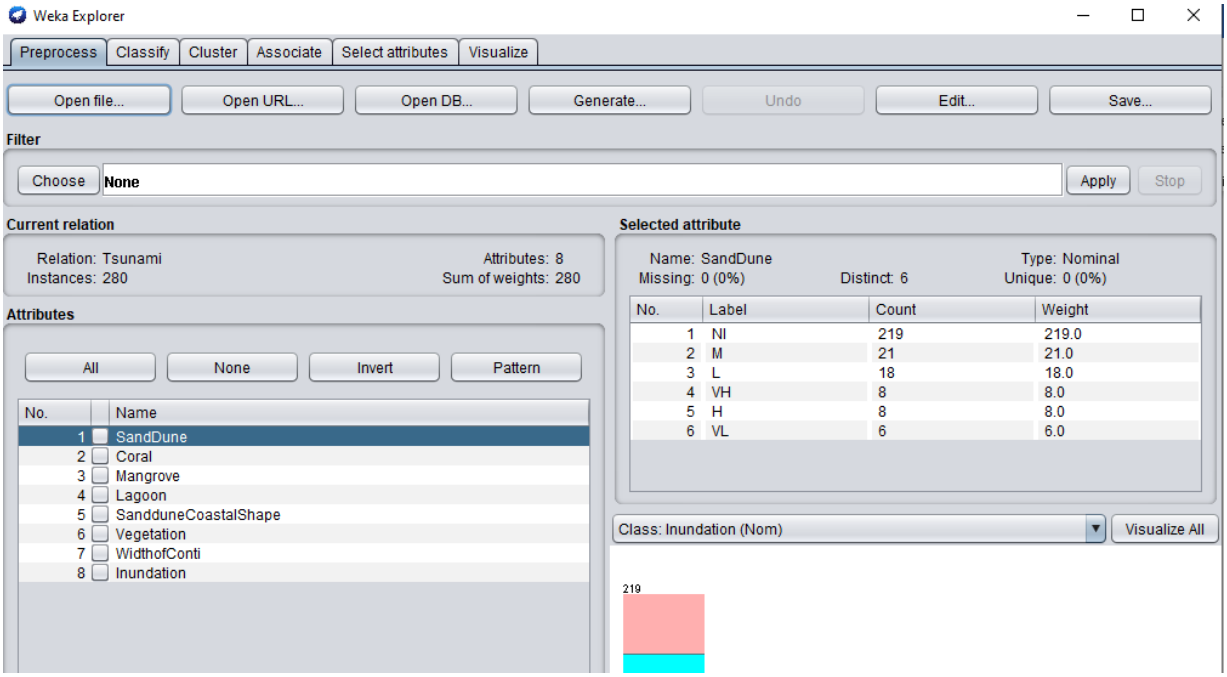
Step 9: Open data file

Click on the processing tab >>> open file >>> select data file >>> click open

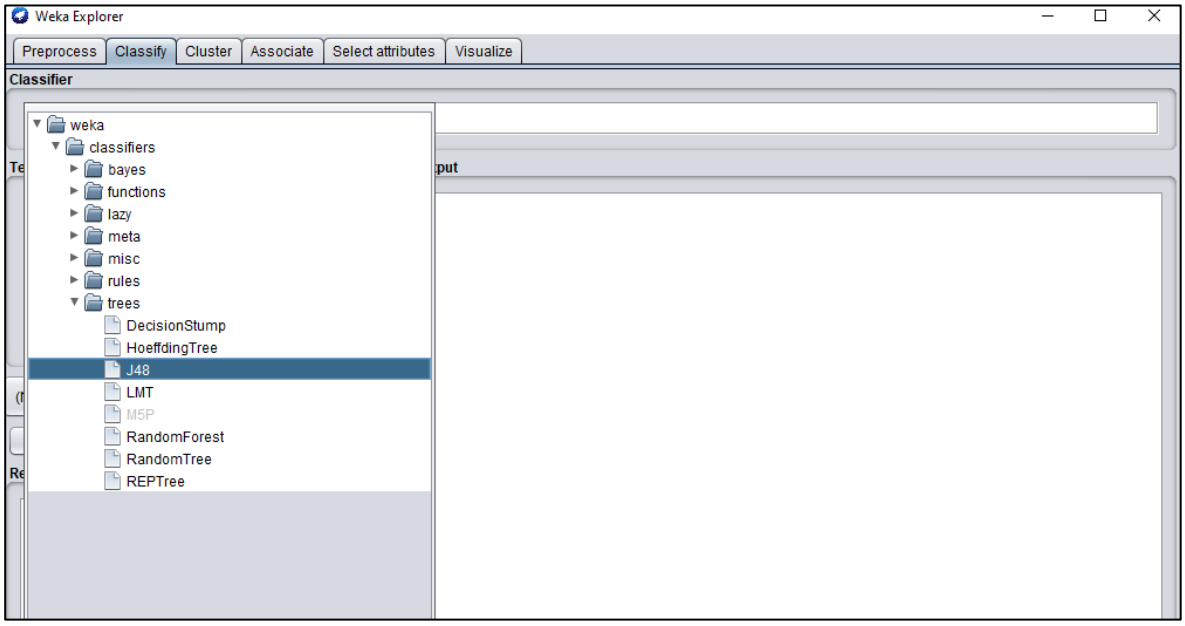


The following interface will appear once you open the datafile.

Step 10: select the reclassification method and validation method.

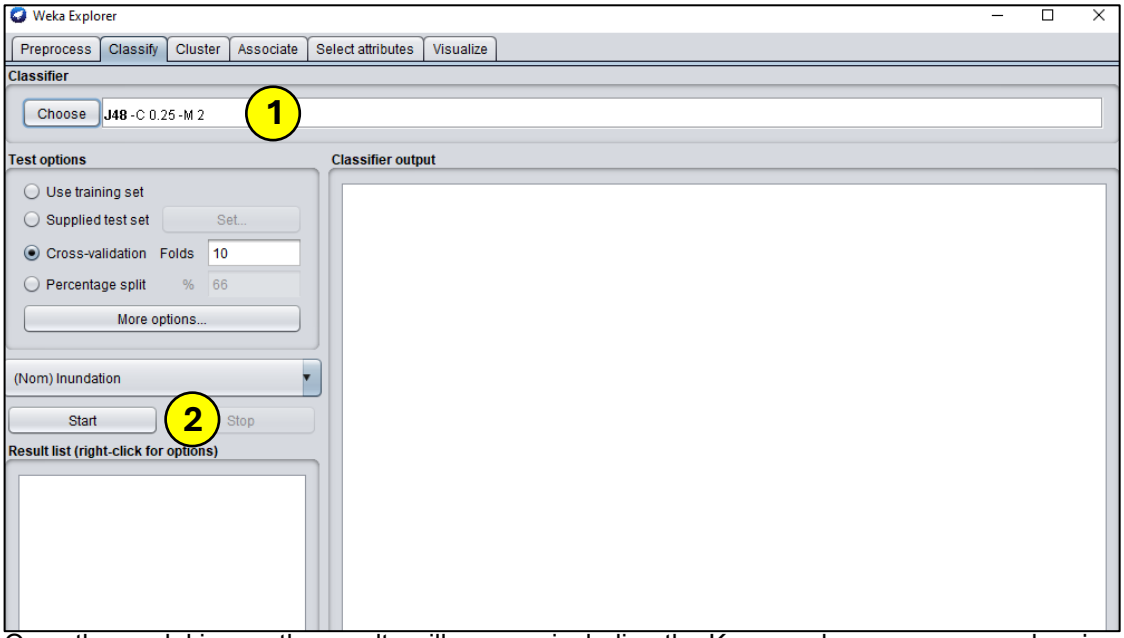


Click on the classify tab as below figure. To select the tree model, click on Choose >>> Trees >>> J48. There are several models available for classification. To visualize the dataset classification as a tree, select the J48 model. Next, choose a validation method. Both cross-validation and percentage split methods can be used. Select either cross-validation or percentage split according to your requirements.

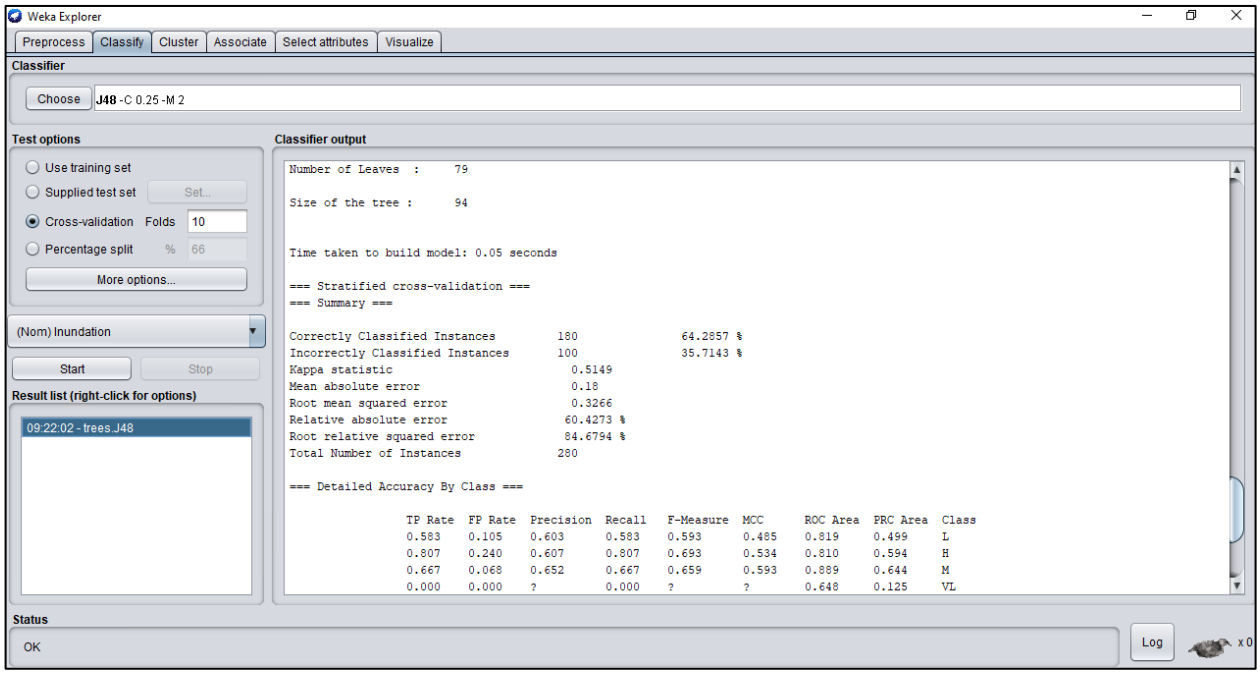


Step 11: Run the model

Once the J48 model is set up, it will appear next to the "Choose" option. After completing the model selection and choosing the validation method, click on the "Start" button.

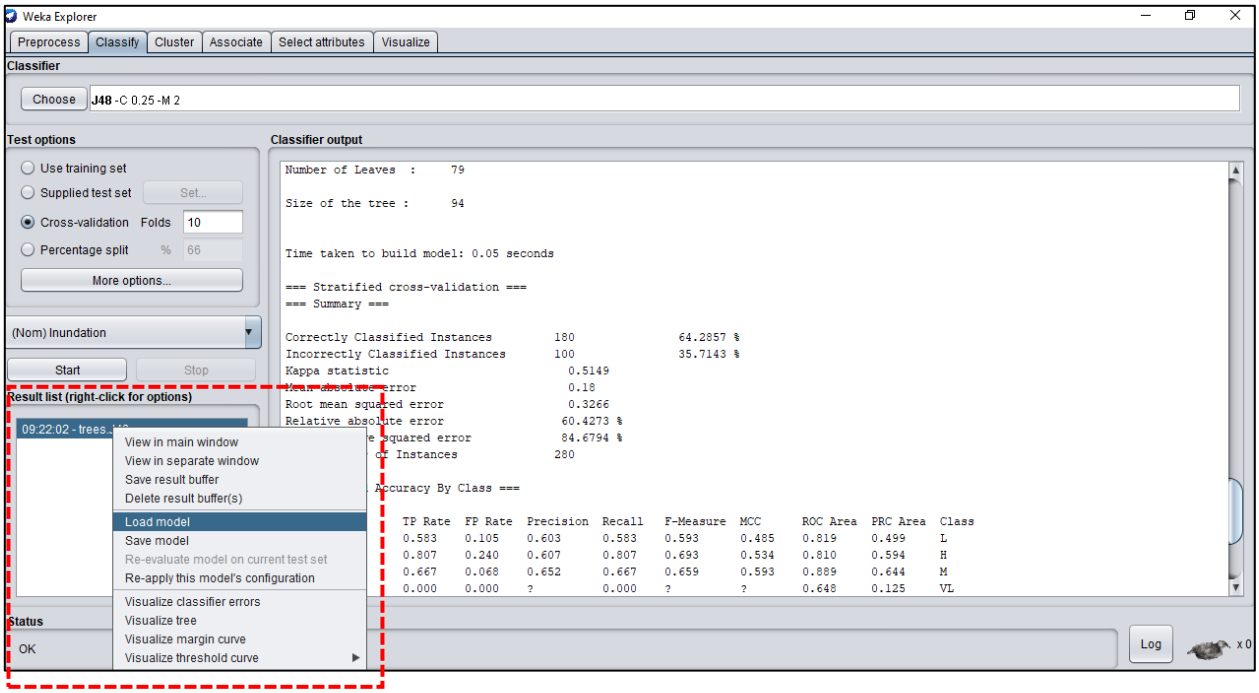


Once the model is run, the results will appear, including the Kappa value, accuracy, and major statistical values.



Step 12:

To view the decision tree, navigate to the highlighted red area and locate the result file in the result list window. Right-click on the result file and select the "Visualize tree" option.



The following decision tree will appear once select 1the visualize tree option.

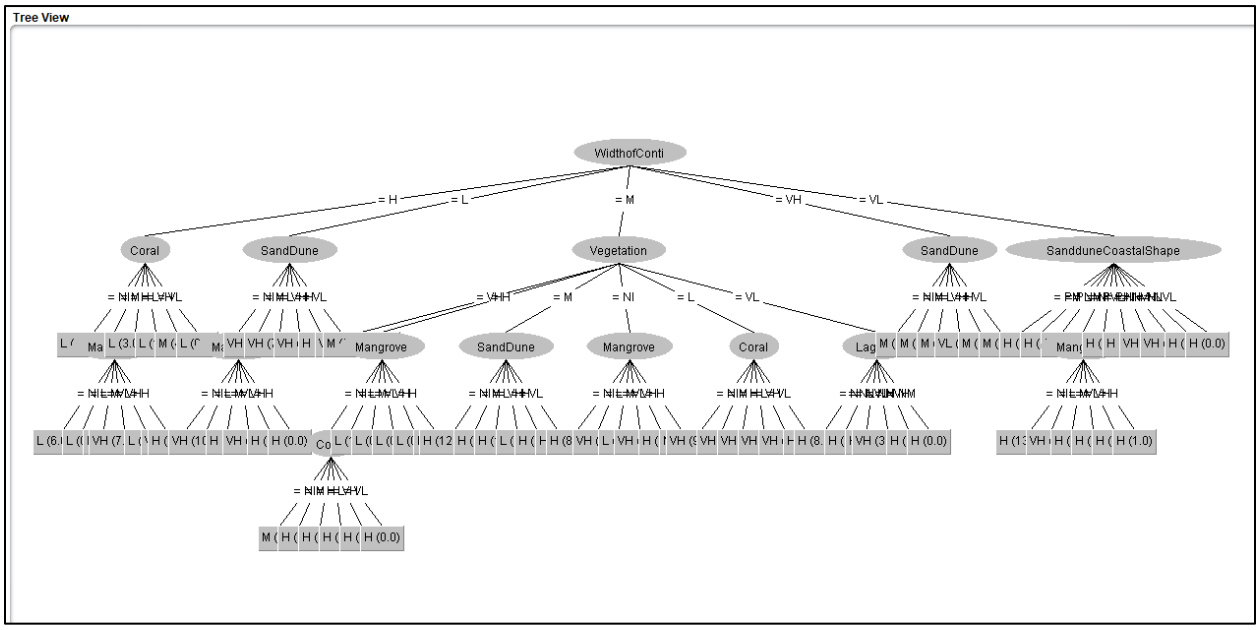


Figure 13 - Visualizing Decision Tree

AFTERWORD

This book marks a step forward in making advanced machine-learning techniques accessible, practical, and relevant across academic and professional landscapes. By focusing on XGBoost and decision tree algorithms within real-world contexts such as urban planning and spatial analysis, it empowers readers to move beyond theory and engage directly with meaningful applications.

We hope this guide inspires continued exploration, critical thinking, and innovation in the field of data-driven decision-making. Whether applied in classrooms, research labs, or industry settings, the tools and techniques presented here are intended to support informed solutions to complex challenges.

As the fields of machine learning and spatial analysis continue to evolve, so too must our commitment to lifelong learning, collaboration, and ethical data use. This book is just one step in that journey.

lbs2its.net

618657-EPP-1-2020-1-AT-EPPKA2-CBHE-JP