

Configuration Manual

MSc Research Project
Master in Data Analytics

Idhaya Bastine Kennedy

Student ID: X23178981

School of Computing
National College of Ireland

Supervisor: Bharat Agarwal

**National College of Ireland
Project Submission Sheet
School of Computing**



Student Name:	Idhaya Bastine Kennedy
Student ID:	X23178981
Programme:	Programme Name
Year:	2024
Module:	MSc Research Project
Supervisor:	Bharat Agarwal
Submission Due Date:	13/12/2024
Project Title:	Configuration Manual
Word Count:	1490
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Idhaya Bastine Kennedy
Date:	12th D e c e m b e r 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Idhaya Bastine Kennedy
X23178981

1 Introduction

The major purpose of this documentation is to assist users in duplicating the code in their own environment. The handbook goes over the hardware specifications, needed libraries to execute the code, dataset source, preprocessing stages, model development, training, and assessment. Some photos of code snippets are also supplied.

2 Environment

To run the code, the environment must first be set up; this part assists in setting up the environment for effective code replication. This research was conducted out in the Kaggle environment since it gives free access to computer resources like as CPU, GPU, and TPU, as well as preinstalled libraries and simple access to datasets.

2.1 Hardware Requirement

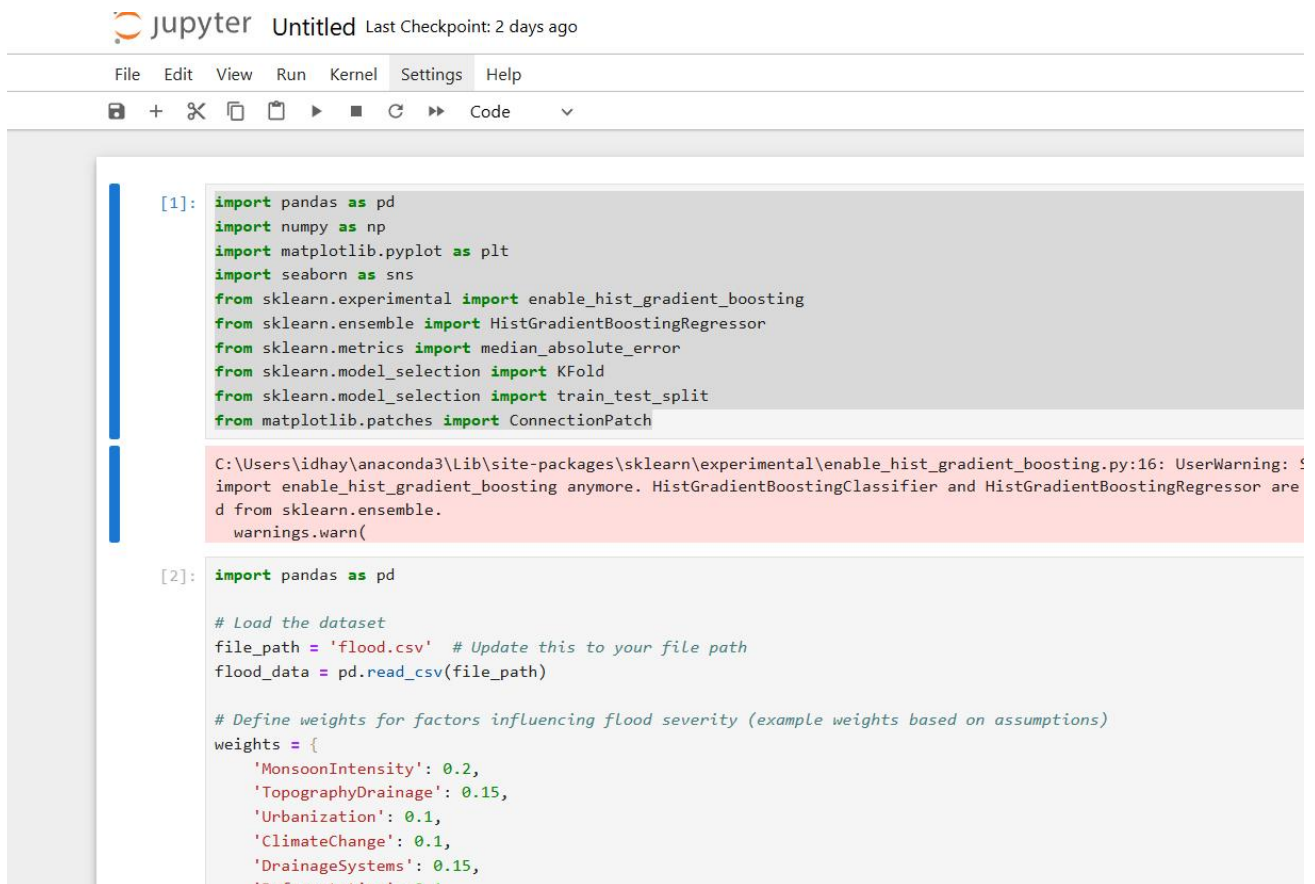
The hardware used for the computational task in this research is Apple Macbook Air. The specification of the hardware in the system is given in detailed in the Table 1.

Component	Specification
Model	MacBook Air
Processor	Apple M1 Chip
Memory (RAM)	8 GB
Storage	256 GB SSD
Operating System	macOS
OS Version	Sonoma 14.5
Graphics	Integrated 7-core / 8-core GPU

Table 1: Specifications of MacBook Air

2.2 Setting up the Anaconda Environment

To set up an Anaconda environment for flood prediction, start by downloading and installing Anaconda from the [official website](#). Use the terminal or Anaconda Prompt to create a new environment with `conda create --name flood_pred_env python=3.9` (replace `flood_pred_env` and 3.9 as needed). Activate the environment using `conda activate flood_pred_env`. Install essential libraries such as `numpy`, `pandas`, `matplotlib`, `seaborn`, `scikit-learn`, and optionally `TensorFlow` or `PyTorch` using `conda install`. Add geospatial tools like `geopandas` and `rasterio` if required. For development, install Jupyter Notebook with `conda install jupyter`. Verify the setup by running `conda list` and testing imports in a Jupyter Notebook. To ensure reproducibility, export the environment configuration with `conda env export > flood_pred_env.yml`. Update libraries with `conda update --all` as needed, and remove the environment if no longer required using `conda remove --name flood_pred_env --all`.



```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.metrics import median_absolute_error
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from matplotlib.patches import ConnectionPatch

C:\Users\idhay\anaconda3\Lib\site-packages\sklearn\experimental\enable_hist_gradient_boosting.py:16: UserWarning: 
import enable_hist_gradient_boosting anymore. HistGradientBoostingClassifier and HistGradientBoostingRegressor are
d from sklearn.ensemble.
  warnings.warn(

[2]: import pandas as pd

# Load the dataset
file_path = 'flood.csv' # Update this to your file path
flood_data = pd.read_csv(file_path)

# Define weights for factors influencing flood severity (example weights based on assumptions)
weights = {
    'MonsoonIntensity': 0.2,
    'TopographyDrainage': 0.15,
    'Urbanization': 0.1,
    'ClimateChange': 0.1,
    'DrainageSystems': 0.15,
    'Deforestation': 0.1
```

Figure 1: Anaconda Interface

3 Implementation

After successfully completing the environmental setup and importing the dataset, you may start the implementation phase. The implementation phase begins with the import of the dataset, requisite libraries, and frameworks, and progresses to model development, training, optimisation, and evaluation.

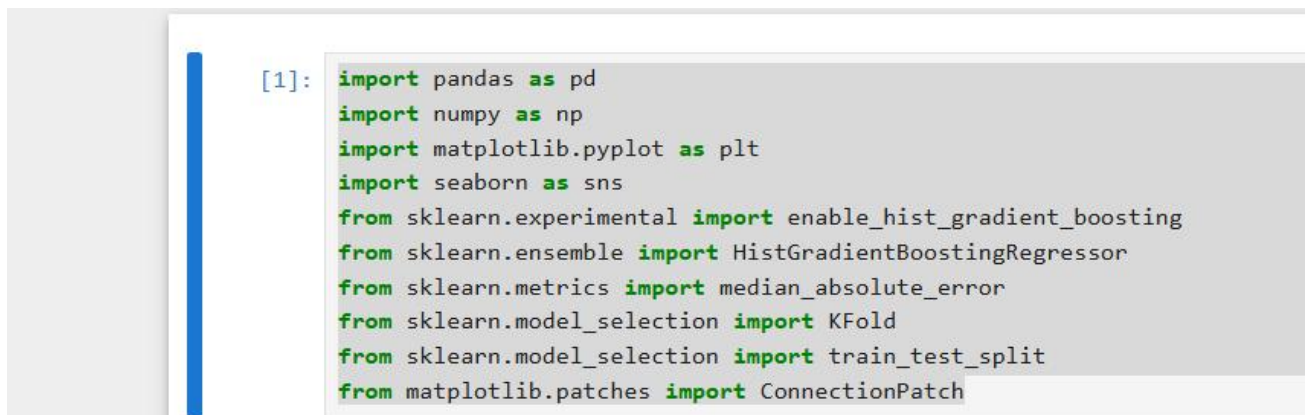
3.1 Importing the Dataset

The initial stage in the implementation is to import the dataset for further processing. The dataset was obtained from the kaggle website. This enables direct data handling within the Kaggle environment, eliminating the requirement for a local environment or cloud storage access. Hence, it would be a straightforward procedure to import the dataset into the notebook. You could see the input area on the left side of the screen. The dataset link has been included for reference purposes.

1. Select the input icon.
2. Click the 'Add Input' icon in the input area.
3. In the search area, enter the name of the dataset, 'Solar Panel Clean and Faulty Images'.
4. Add the dataset.
5. The dataset was uploaded to the input area, as indicated.

3.2 Importing Required Libraries

The research makes use of a variety of libraries for data preparation, model training, assessment, and visualisation. The libraries must be imported first. As a result, import the libraries stated in the first section of the code, as seen in Figure 5.



```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.metrics import median_absolute_error
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from matplotlib.patches import ConnectionPatch
```

Figure 5: Importing the Libraries

3.3 Data Preprocessing

This configuration model forecasts flood severity by weighting numerous environmental and socioeconomic parameters. The script starts by importing data from a CSV file and assigning specified weights to variables including monsoon intensity, urbanisation, and topography drainage. A custom function computes a FloodSeverity score for each record by adding the weighted contributions of these variables. The scores are optionally normalised to a [0, 1] range for consistency and divided into three severity levels: low, moderate, and high. Finally, the processed data, including severity scores and levels, is saved in an updated CSV file for later study. This model can help academics and decision-makers estimate flood risk and handle disasters.

```
# Define weights for factors influencing flood severity (example weights based on assumptions)
weights = {
    'MonsoonIntensity': 0.2,
    'TopographyDrainage': 0.15,
    'Urbanization': 0.1,
    'ClimateChange': 0.1,
    'DrainageSystems': 0.15,
    'Deforestation': 0.1,
    'WetlandLoss': 0.1,
    'Siltation': 0.05,
    'PoliticalFactors': 0.05
}

# Calculate the weighted severity score
def calculate_severity(row, weights):
    score = sum(row[col] * weight for col, weight in weights.items())
    return score

# Apply the function to calculate a new column 'FloodSeverity'
flood_data['FloodSeverity'] = flood_data.apply(calculate_severity, axis=1, weights=weights)

# Normalize the 'FloodSeverity' score (optional)
flood_data['FloodSeverityNormalized'] = (
    (flood_data['FloodSeverity'] - flood_data['FloodSeverity'].min()) /
    (flood_data['FloodSeverity'].max() - flood_data['FloodSeverity'].min())
)

# Classify severity into levels
def classify_severity(value):
    if value < 0.33:
        return "Low"
    elif value < 0.66:
        return "Moderate"
    else:
        return "High"
```

Figure 6 : DataFrame Creation and Preprocessing Steps

3.4 EDA : Exploratory data analysis (EDA)

This code provides a series of visualizations to explore the distribution and relationships within flood-related data. The first part visualizes the distribution of numeric columns in the dataset using kernel density estimation (KDE) plots. Each numeric feature is plotted individually with a clear indication of its skewness and median, providing insights into its overall distribution. This helps identify any imbalances or outliers in the dataset that might affect flood prediction models. The second code snippet creates a scatter plot that explores the relationship between ClimateChange and FloodProbability, with Deforestation as a color-coded hue. This allows for an in-depth analysis of how deforestation impacts the likelihood of floods under changing climate conditions. Lastly, the third code generates histograms for all numeric features, providing a quick overview of their distributions, highlighting trends, and revealing potential issues like skewness or extreme values. These visualizations collectively enable a better understanding of the flood data, helping inform further statistical analysis or model adjustments.

```
numeric_columns = flood_data.select_dtypes(include=[ 'float64' , 'int64' ])

def dist(train_dataset, columns_list, rows, cols):
    fig, axs = plt.subplots(rows, cols, figsize=(40, 20))
    axs = axs.flatten()
    for i, col in enumerate(columns_list):
        sns.kdeplot(train_dataset[col], ax=axs[i], fill=True, alpha=0.5, linewidth=0.5, color='#058279', label='Train')
        axs[i].set_title(f'{col}, Train skewness: {train_dataset[col].skew():.2f}')
        axs[i].legend()
        axs[i].axis('off')
        axs[i].set_xticks([])
        axs[i].set_yticks([])
        median_train = train_dataset[col].median()
        axs[i].axvline(x=median_train, color='#4caba4', linestyle='--')
        axs[i].legend(labels=['Train', 'Median'])

    fig.suptitle('Distribution of Numeric Columns', fontsize=30)
    plt.tight_layout()
    sns.despine(left=True, bottom=True)

dist(train_dataset=flood_data, columns_list=numeric_columns.columns, rows=4, cols=6)
```

Figure 7: Line graph for Columns With Numeric Value

```
flood_data.hist(figsize=(12, 10), color='skyblue', edgecolor='black')
plt.gcf().set_facecolor('yellow')
plt.tight_layout()
plt.show()
```

Figure 8 : Bar Graph for al columnl

```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.scatterplot(data=flood_data, x='ClimateChange', y='FloodProbability', hue='Deforestation', palette='Set1')
plt.title('FloodProbability vs. ClimateChange by Deforestation')
plt.xlabel('ClimateChange')
plt.ylabel('FloodProbability')
plt.legend(title='Deforestation')
plt.gcf().set_facecolor('lightgrey')
plt.show()
```

Figure 10: line graph between FloodProbability vs. ClimateChange by Deforestation

3.5

This script employs clustering analysis of flood risk data to discover trends for evaluation. It requires Python 3.7 or above, as well as the pandas, scikit-learn, and matplotlib libraries, which may be installed using pip. The dataset (flood_data) should be in CSV or Excel format, with essential columns including topography, drainage, deforestation, and urbanization.

The script inserts the dataset into a pandas DataFrame, selecting columns from the features list. Missing values are replaced with the mean of each column to ensure a complete dataset. The data is then standardized with StandardScaler, which ensures that all variables contribute equally. The resulting scaled dataset (X_scaled) is ready to cluster.

To run, save and execute the script (e.g., clustering_analysis.py). Ensure that the file path and columns are properly setup. You can alter the feature list.

3.5 Elbow Plot and silhouette Score

This code uses the Elbow Method to calculate the best number of clusters for a clustering algorithm applied to flood data. The graph, which plots the inertia (within-cluster sum of squared distances) versus the number of clusters, aids in identifying the "elbow," or point at which inertia begins to drop at a slow rate. This point provides the appropriate number of clusters for effective data segmentation. The graphic shows a range of 2 to 10 clusters, making it easy to compare inertia values and indicating the best number of clusters. This approach helps to identify the optimal clustering configuration, resulting in meaningful segmentation for future flood prediction analysis.

```
# Step 4a: Determine the optimal number of clusters using the Elbow Method
inertia = []
silhouette_scores = []
for k in range(2, 11): # Trying from 2 to 10 clusters
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))

# Plot the Elbow curve to visualize the optimal number of clusters
plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), inertia, marker='o', label='Inertia')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()

# Step 4b: Plot Silhouette Scores to validate the cluster count
plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o', label='Silhouette Score', color='orange')
```

Figure 9: Elbow Plot

```
#custom callback to calculate TPR and FPR
class ROCCallback(tf.keras.callbacks.Callback):
    def __init__(self, validation_data):
        self.validation_data = validation_data
        self.fpr = []
        self.tpr = []

    def on_epoch_end(self, epoch, logs=None):
        X_val, y_val = self.validation_data
        y_pred = self.model.predict(X_val)
        fpr_epoch, tpr_epoch, _ = roc_curve(y_val.ravel(), y_pred.ravel())
        self.fpr.append(fpr_epoch)
        self.tpr.append(tpr_epoch)

#for compile and train the model
def LR_verify(model):
    METRICS = [
        'accuracy',
    ]
    model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
        filepath='best_model_weights.h5',
        save_weights_only=True,
        monitor='val_accuracy',
        mode='max',
        save_best_only=True)
    roc_callback = ROCCallback(validation_data=(X_test, y_test))
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=METRICS
    )
    history = model.fit(X_train, Y_train,
                        epochs=EPOCHS,
                        verbose=1,
                        callbacks=[model_checkpoint_callback, roc_callback])
    return model, history, roc_callback

model = mod()
model, history, roc_callback = LR_verify(model)
```

This code provides a figure to assess the efficacy of clustering models based on the Silhouette Score for various cluster sizes. The Silhouette Score compares an object's similarity to its own cluster to that of other clusters, with higher values suggesting more defined and distinct clusters. The figure shows the Silhouette Scores for cluster numbers ranging from 2 to 10, allowing you to visually determine the ideal number of clusters. By inspecting the graph, one may discover which value of k has the best clustering quality, providing useful information for improving flood prediction models. This research extends the Elbow Method by offering an extra metric for evaluating clustering efficacy.

Figure 10: Silhouette Score

3.6 K-Means Clustering for Flood Data Segmentation

This code uses K-Means clustering to categorise flood data according to environmental and socioeconomic variables. After determining the best number of clusters (e.g., four using the Elbow Method or Silhouette Score), the KMeans model is applied to the scaled data, and each data point is allocated to a cluster. The centroids of each cluster, which indicate the average values of the characteristics in each group, are visible for inspection. Furthermore, the initial few rows of the dataset are provided, together with their accompanying cluster allocations, giving insight into how the data is organised. This segmentation can assist identify distinct flood risk levels and aspects in the data, allowing for additional flood analysis or decision-making.

```
# Step 5: Choose the optimal number of clusters (e.g., 4 based on elbow method or silhouette score)
optimal_k = 4 # Adjust this based on the Elbow Method or Silhouette Score

# Step 6: Fit the KMeans model with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
flood_data['Cluster'] = kmeans.fit_predict(X_scaled)

# Step 7: Analyze the results
# Show the clusters with their centroid values
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=features)
print("Cluster Centroids:")
print(centroids)

# Show the first few rows with their assigned cluster
print("\nFirst few rows with assigned clusters:")
```

3.6 Random Forest Regression for Flood Probability Prediction

```
assuming 'SeverityLevel' is the categorical column
severity_mapping = {'Low': 1, 'Moderate': 2, 'High': 3}
flood_data['SeverityLevel'] = flood_data['SeverityLevel'].map(severity_mapping)

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

flood_data.drop(labels=['FloodProbability'],axis=1)
flood_data['FloodProbability']

X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.20, random_state=42)

model = RandomForestRegressor()
model.fit(X_train,Y_train)
predicted=model.predict(X_test)
print('Accuracy: ',model.score(X_test,Y_test))
```

This program forecasts the probability of floods based on socioeconomic and environmental factors using the Random Forest Regressor. Prior to being utilised in the regression model, the 'SeverityLevel' category column is converted into numerical values. After that, the dataset is separated into training and testing sets. Just 20% of the data is utilised for testing; the remaining 80% is used for training. The Random Forest model is trained using the training data and then produces predictions for the test dataset. By contrasting the predicted flood probability with the actual data, the model's accuracy is assessed. By assessing how well a range of variables predict flood risk, the technique creates a reliable tool for predicting the probability of flooding and creating risk management measures.

3.7 Model Comparison and Evaluation for Flood Prediction

This tool examines and evaluates how effectively a variety of regression models predict flood-related events, such as the likelihood of flooding. Each model, including the MLPRegressor, SupportVectorRegressor, RandomForestRegressor, and HistGradientBoostingRegressor, is trained on the training data before generating predictions on the test set. The model's prediction and accuracy are evaluated using metrics such as R2 and MSE. Compare the results to determine which model best meets your needs. The residual plots generated for each model are then used to visually assess the difference between observed and expected values. The correlation heatmap from the dataset is displayed to understand more about the relationships between the variables. This extensive study assists in determining the most efficient.

```

# Evaluate Model Function
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2, y_pred

# # Split Data
# X, y, scaler = preprocess_data(data)
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Models
models = {
    "HistGradientBoostingRegressor": HistGradientBoostingRegressor(),
    "RandomForestRegressor": RandomForestRegressor(random_state=42),
    "SupportVectorRegressor": SVR(),
    "MLPRegressor": MLPRegressor(random_state=42, max_iter=1000)
}

# Evaluate Models
results = {}
predictions = {}
for model_name, model in models.items():
    mse, r2, y_pred = evaluate_model(model, X_train, X_test, y_train, y_test)
    results[model_name] = {"MSE": mse, "R2": r2}
    predictions[model_name] = y_pred

# Display Results
print("Model Evaluation Results:")
for model_name, metrics in results.items():
    print(f"{model_name}: MSE={metrics['MSE']:.4f}, R2={metrics['R2']:.4f}")

```

4 Results

Results may appear once the implementation code has been successfully finished. Criteria like as accuracy, precision, F1 score, recall, and a confusion matrix are used to assess the results.