National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shivani Vaidya
Student ID: X22210580

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Shivani Prakash Vaidya

Student ID: ……x22210580…

Programme: ……MSc Data Analytics **...** **Year:** ……2023-2024**.**

Module: ……MSc Research Project …

Lecturer: ……Dr. Catherine Mulwa

Submission Due Date: ……12/08/2024……

Project Title: Early Prediction of Autism Spectrum Disorder in toddlers

Word Count: ……1478…… **Page Count:** ……11……

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: ……Shivani Prakash Vaidya ……

Date: ……12/08/2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
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.	□

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

Shivani Vaidya
X22210580:

1 Introduction

Detailed information regarding software and hardware requirements and tools used for conducting this analysis was provided in this configuration manual. This configuration manual is a step-by-step guide to set up the environment, and pre-request needed for writing and execution of code. For the reproduction of results in 'early prediction of autism spectrum disorder in toddlers' research, this manual should be considered. The code snippets for the deep learning and machine learning techniques utilized for the generation of results are attached in this manual which are not added in the main report.

2 Software and hardware requirements

Software requirements for the implementation of the project are listed below.

Data collection:

Platform – Data World website

https://data.world/tanmay0510/autism-spectrum-disorder

Tools and Integrated Development Environments (IDE)

- **Data storage:** MongoDB Compass 2.2.15
- **Data pre-processing, modeling, and evaluation:** Jupyter notebook
- **Data visualization:** R studio

Scripting languages used

- Python: 3.8.2
- R: 4.3.1

Other tools used:

- Microsoft Excel
- Lucid chart
- Zotero

Hardware requirements for the implementation of the project are listed below.

- **Operating system:** Windows 11 Home Single Language, 10.0.22631 Build 22631
- **Processor:** 13th Gen Intel (R) Core (TM) i7- 1360P, 2200 MHz, 12 Core(s), 16Logical Process
- **System Model:** Inspiron 14 5430
- **RAM storage:** 16.0 GB

3 Methodology steps

Data Storage

1. Install the MongoDB compass
2. Select and download the appropriate MongoDB compass version as per Windows compatibility
3. Install and launch MongoDB compass
4. Open MongoDB compass and connect to the local instance of the server mongodb://localhost:27017
5. Create a new database and import a CSV file of the autism spectrum disorder dataset in MongoDB compass and store it as shown in Figure 1.

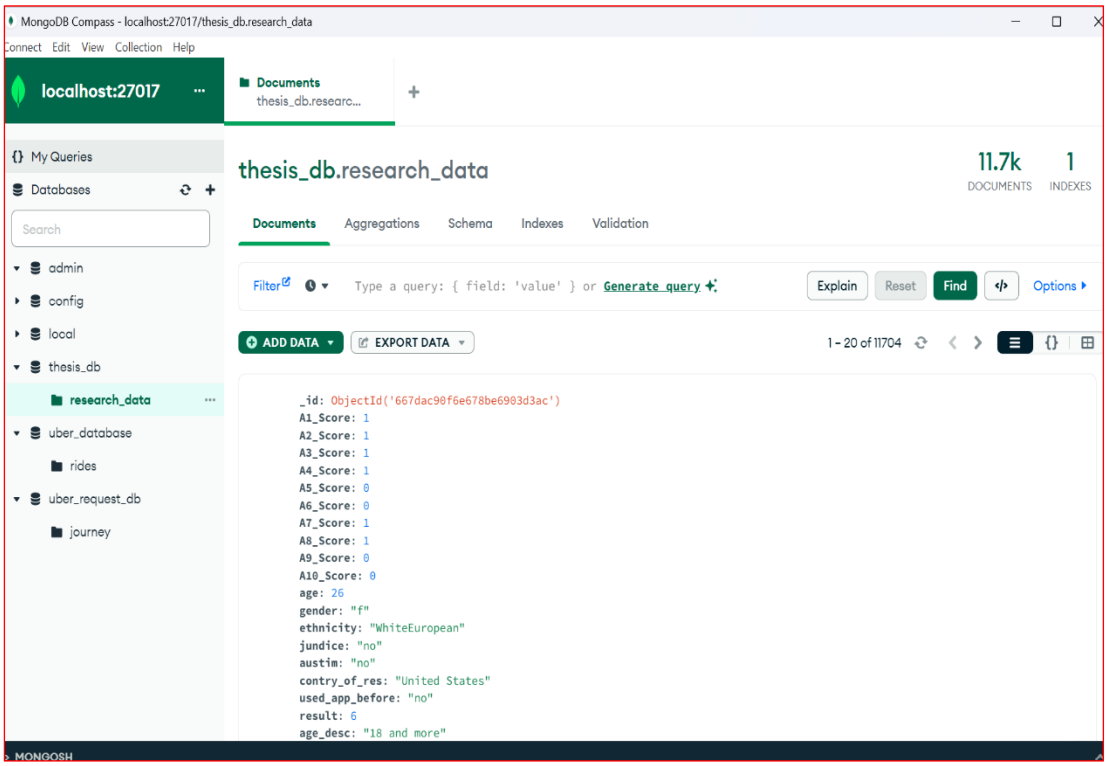

Figure 1: Data storage in MongoD

Creation of a virtual environment

1. Install the Jupyter Notebook web application using pip install jupyter command

2. Initially pip install virtualenv using this command to create a virtual environment.

3. Then navigate to the desired directory and use the python -m venv tf_venv command to create a virtual environment with the name tf_venv

4. Add virtual environment as a kernel into jupyter notebook using python -m ipykernel install --user --name=tf_venv --display-name "Python (tf_venv)" command

5. Install the required libraries

   pip install tensorflow
   pip install pandas
   pip install pymongo
   pip install ipykernel
   pip install scikit-learn
   pip install matplotlib

Data Extraction from MongoDB

1. Connect with the local MongoDB instance to access the research ASD database
2. Retrieve all documents from the established collection and store them in list format.
3. Convert a list of documents into a data frame as shown in Figure 2

```
# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['thesis_db']  # Replace with your database name
collection = db['research_data']  # Replace with your collect

# Retrieve data from MongoDB
data = list(collection.find())

# Convert to DataFrame
df = pd.DataFrame(data)

# Drop the MongoDB '_id' field if it exists
if '_id' in df.columns:
    df = df.drop('_id', axis=1)
```

Figure 2: Data extraction from MongoDB

Exploratory Data Analysis (EDA)

I. Data Cleaning

1. Identify and print columns with missing values handle missing values and later verify that missing values are handled correctly.
2. Identify and remove duplicate values and verify it
3. Convert columns to a data frame as shown in Figure 3.

```
# Data Cleaning
# Handling Missing Values
print("\nHandling Missing Values:")
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])

# Fill missing values with the mean of the column, specifying numeric_only=True
df.fillna(df.mean(numeric_only=True), inplace=True)

# Verify if all missing values are handled
print("\nAfter filling missing values:")
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])

# Removing Duplicates
print("\nRemoving Duplicates:")
print("Number of duplicates: ", df.duplicated().sum())
df.drop_duplicates(inplace=True)
print("After removing duplicates: ", df.duplicated().sum())

# Correcting Data Types
print("\nCorrecting Data Types:")
```

Figure 3: Data Cleaning

1. Removal of Outliers through the interquartile method

2. Define a function for filtering outliers from a particular column using the interquartile method.
3. Compute the first and third quartiles and calculate the interquartile range and upper and lower bounds.
4. Store the initial number of rows in the Data Frame before eliminating outliers identify and select Numeric Columns iterate them and apply the function.
5. Store the final rows after eliminating outliers and print the number of outliers removed as shown in Figure 4.

```python
# Removing Outliers using IQR Method
print("\nRemoving Outliers:")
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

initial_row_count = len(df)
numeric_cols = df.select_dtypes(include=[float, int]).columns
for col in numeric_cols:
    df = remove_outliers_iqr(df, col)

final_row_count = len(df)
outliers_removed = initial_row_count - final_row_count

print(f"Initial row count: {initial_row_count}")
print(f"Final row count after removing outliers: {final_row_count}")
print(f"Number of outliers removed: {outliers_removed}")
```

Figure 4: Removal of Outliers using the Interquartile Method

II. Data Transformation

1. Standardize the numeric columns and print the initial rows of the transformed data.
2. Encode the output variable in the data 'Class/ASD' using LabelEncoder.
3. Convert categorical variables to one-hot encoded columns and print the initial rows of the encoded DataFrame as shown in Figure 5.

```python
# Data Transformation
# Normalization or Standardization
print("\nStandardizing Data:")
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
print(df[numeric_cols].head())

# Encoding target variable before one-hot encoding to prevent issues
label_encoder = LabelEncoder()
df['Class/ASD'] = label_encoder.fit_transform(df['Class/ASD'])

# Encoding Categorical Variables
print("\nEncoding Categorical Variables:")
df_encoded = pd.get_dummies(df, drop_first=True)
print(df_encoded.head())
```

Figure 5: Data Transformation

III. Principle component analysis

1. Create a PCA instance with 2 components and
2. Create a data frame with principal components and appropriate column names
3. Save the data frame to a CSV file as shown in Figure 6.

```
# PCA
print("\nPerforming PCA:")
pca = PCA(n_components=2)  # You can change the number of components as needed
principal_components = pca.fit_transform(df_selected)

# Create a DataFrame with the PCA results
pca_df = pd.DataFrame(data=principal_components, columns=['Principal Component 1', 'Principal Component 2'])
print(pca_df.head())

# Save the PCA-transformed Data to CSV
# Define the file path for the CSV
csv_file_path = 'pca_transformed_data.csv'

# Save the DataFrame to CSV
pca_df.to_csv(csv_file_path, index=False)

print(f"\nPCA-transformed data saved to {csv_file_path}")
```

Figure 6: Principal Component Analysis

IV. Data Splitting

1. Split the encoded DataFrame into features and target
2. Split the data into training (80%) and testing (20%) sets with a random of 42 for reproducibility as shown in Figure 7.

```
# Split the data into features and target
X = df_encoded.drop(columns=['Class/ASD'])
y = df['Class/ASD']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 7: Data Splitting

V. Hyper Parameter Tunning using a Random search method

1. Mention the parameter distributions for hyperparameter tuning such as hidden layers, activation functions, and learning rates.
2. Create a RandomizedSearchCV object for the classifier then fit the object to the training data and then extract the best hyperparameters and store them as showcased.

```python
# Hyperparameter tuning using RandomizedSearchCV with fewer iterations and simpler search space
param_distributions = {
    'hidden_layer_sizes': [(50,), (100,)],
    'activation': ['tanh', 'relu'],
    'learning_rate': ['constant'],
    'alpha': [0.0001, 0.001]
}

random_search = RandomizedSearchCV(
    MLPClassifier(solver='adam', max_iter=200, random_state=42, early_stopping=True),
    param_distributions,
    n_iter=10,  # Reduce n_iter for quicker results
    n_jobs=-1,
    cv=3,
    scoring='accuracy',
    random_state=42
)

random_search.fit(X_train, y_train)
best_params = random_search.best_params_
```

Figure 7: Hyper Parameter Tunning using a Random search method

VI. Implementataion

1) Multilayer Perceptron model :

1. Establish and train MLPClassifier with the best hyperparameters
2. Save the accuracy and loss history of the training process into a CSV file
3. Forecast the test set labels and then print the classification report and perform accuracy calculation
4. Add the estimated and actual values to the DataFrame and store it in a CSV file as shown in Figure 8

```python
# Train the MLP model with the best parameters
mlp_best = MLPClassifier(**best_params, solver='adam', max_iter=200, random_state=42, early_stopping=True)
mlp_best.fit(X_train, y_train)

# Save accuracy and loss history
history_df = pd.DataFrame({
    'epoch': np.arange(len(mlp_best.loss_curve_)),
    'loss': mlp_best.loss_curve_,
    'accuracy': mlp_best.score(X_train, y_train)
})
history_df.to_csv('mlp_training_history.csv', index=False)

# Evaluate the model
y_pred = mlp_best.predict(X_test)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score of the MLP model:")
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

df_selected['Predicted'] = mlp_best.predict(X)
df_selected['Actual'] = df['Class/ASD']
df_selected.to_csv('MLP_model.csv', index=False)
```

Figure 8: Multilayer Perceptron model implementation

2) Convolution Neural Network model

1. Build a sequential CNN model with different layers like convolution, dropout, flattened, dense layers

2. Compile the model with the Adam optimizer, categorical cross-entropy loss, and accuracy metric. Train the CNN model and save the history.

3. Make predictions on test sets and convert predictions and true labels to class labels.

4. Perform accuracy calculation and print classification report shown is Figure 8 and Figure 9 respectively.

```
# Build the CNN model
cnn_model = Sequential()
cnn_model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(X_train_cnn.shape[1], 1)))
cnn_model.add(MaxPooling1D(pool_size=2))
cnn_model.add(Dropout(0.5))
cnn_model.add(Flatten())
cnn_model.add(Dense(50, activation='relu'))
cnn_model.add(Dense(y_train_cnn.shape[1], activation='softmax'))

# Compile the model
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model and capture the history
history = cnn_model.fit(X_train_cnn, y_train_cnn, epochs=50, batch_size=32, validation_data=(X_test_cnn, y_test_cnn))

# Save the training history
history_df = pd.DataFrame(history.history)
history_df['epoch'] = history.epoch
history_df.to_csv('cnn_training_history.csv', index=False)

# Evaluate the model
y_pred_cnn = cnn_model.predict(X_test_cnn)
y_pred_cnn_classes = np.argmax(y_pred_cnn, axis=1)
y_test_cnn_classes = np.argmax(y_test_cnn, axis=1)
```

Figure 8: Convolution of Neural Network model implementation

```
# Print classification report and accuracy score
print("\nClassification Report:")
print(classification_report(y_test_cnn_classes, y_pred_cnn_classes))
print("\nAccuracy Score of the CNN model :")
accuracy = accuracy_score(y_test_cnn_classes, y_pred_cnn_classes)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Figure 9: Classification report for the CNN model

3) Long Short-term memory model

1. Train the LSTM model with the best hyperparameters obtained using hyperparameter tunning
2. Save the training history and store it in a CSV file
3. Predict the test data and convert predictions into a binary outcome
4. Calculate accuracy and print classification report as shown in Figure 10.

```
# Train the LSTM model with the best parameters for more epochs
best_model = create_lstm_model(optimizer=best_params['optimizer'],
                               activation=best_params['activation'],
                               dropout_rate=best_params['dropout_rate'],
                               neurons=best_params['neurons'])
history = best_model.fit(X_train, y_train, epochs=20, batch_size=best_params['batch_size'], validation_split=0.2, verbose=1)

# Save the training history
history_df = pd.DataFrame(history.history)
history_df['epoch'] = range(1, len(history_df) + 1)
history_df.to_csv('lstm_training_history.csv', index=False)

# Make predictions
y_pred = (best_model.predict(X_test) > 0.5).astype("int32")

# Evaluate the model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score of the LSTM model :")
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Figure 10: Long Short-Term Memory Model implementation

4) Ranodm forest model

1. Implement a random forest model on the training data and use the trained model to predict test data.
2. Perform accuracy calculation and print classification report and print confusion matrix as shown in Figure 11.

```
# Apply Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# Print classification report and accuracy
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy Score of the Random Forest model: {accuracy * 100:.2f}%")

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Figure 11: Random Forest Model implementation

5) Feature importance analysis using SHapley Additive exPlanations technique

1. Calculate SHapley Additive exPlanations values and mean absolute SHAP values for every feature
2. Create a data frame that has these values and respective feature names
3. Sort and compute the percentage contribution of each feature
4. Sort the data frame as per the feature importance and percentage contribution and print the SHAP values.
5. Print the bar plot for feature vs percentage contribution as shown in Figure 12.

```
# SHAP feature importance using KernelExplainer
explainer = shap.KernelExplainer(model.predict_proba, X_train_subset)
shap_values = explainer.shap_values(X_train_subset)

# Calculate mean absolute SHAP values for each feature
shap_values_mean = np.abs(shap_values[1]).mean(axis=0)
shap_importance = pd.DataFrame(list(zip(X.columns, shap_values_mean)), columns=['Feature', 'Importance'])

# Sort by importance
shap_importance = shap_importance.sort_values(by='Importance', ascending=False)

# Calculate percentage contribution
shap_importance['Percentage'] = 100 * shap_importance['Importance'] / shap_importance['Importance'].sum()

# Save SHAP values to CSV
shap_importance.to_csv('shap_values.csv', index=False)

# Print the SHAP values DataFrame
print(shap_importance)

# Print the results for each feature
for index, row in shap_importance.iterrows():
    print(f"Feature: {row['Feature']}, Importance: {row['Importance']:.6f}, Percentage: {row['Percentage']:.6f}%"

# Plotting the SHAP values
plt.figure(figsize=(10, 6))
plt.bar(shap_importance['Feature'], shap_importance['Percentage'])
plt.xlabel('Features')
plt.ylabel('Percentage Contribution')
plt.title('Feature Importance based on SHAP values')
plt.xticks(rotation=90)
plt.show()
```

Figure 12: Feature Importance Analysis using the SHAP technique.

VII. Data Visualizations

1. Load the required R libraries like ggplot2, readr, etc
2. Read the training history from CSV loaded in R from Python code
3. Plot the accuracy and loss
4. Similarly load the training CSV files for other models as well and plot accuracy and loss as shown in Figure 13.

```r
# Load necessary libraries
library(ggplot2)
library(data.table)
library(readr)

history <- fread("mlp_training_history.csv")

# Plot accuracy with a white background
accuracy_plot <- ggplot(history, aes(x=epoch)) +
  geom_line(aes(y=train_accuracy, color="train")) +
  geom_line(aes(y=validation_accuracy, color="validation")) +
  labs(title="Model Accuracy", x="Epoch", y="Accuracy") +
  scale_color_manual(values=c("train"="blue", "validation"="orange"),
                     labels=c("train", "validation")) +
  theme_minimal(base_size = 15) +
  theme(panel.background = element_rect(fill = "white", color = "black"),
        plot.background = element_rect(fill = "white", color = NA),
        legend.background = element_rect(fill = "white"))

# Plot loss with a white background
loss_plot <- ggplot(history, aes(x=epoch)) +
  geom_line(aes(y=train_loss, color="train")) +
  geom_line(aes(y=validation_loss, color="validation")) +
  labs(title="Model Loss", x="Epoch", y="Loss") +
  scale_color_manual(values=c("train"="blue", "validation"="orange"),
                     labels=c("train", "validation")) +
  theme_minimal(base_size = 15) +
  theme(panel.background = element_rect(fill = "white", color = "black"),
        plot.background = element_rect(fill = "white", color = NA),
        legend.background = element_rect(fill = "white"))

# Display the plots in R
print(accuracy_plot)
print(loss_plot)
```

Figure 13: Data Visulizations