National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Jebitta Joseph
Student ID: x23151196

School of Computing
National College of Ireland

Supervisor: Prof.Abdul Qayum

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Jebitta Joseph
Student ID:	x23151196
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof.Abdul Qayum
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	674
Page Count:	8

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:	
Date:	12th December 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

Jebitta Joseph
x23151196

1 System Requirements for Accident Prevention and Insights

1.1 Hardware Requirements

Processor: Intel Core i5 or equivalent (minimum), Core i7 or equivalent (recommended).
RAM: 8 GB (minimum), 16 GB (recommended for faster processing)
Storage: 256 GB SSD (minimum), 512 GB SSD or higher (recommended)
Network: Stable internet connection for downloading libraries and datasets.

1.2 Software Requirements

Operating System: Windows 10/11, macOS, or Linux (Ubuntu 20.04 or higher).
Python Version: Python 3.8 or higher
Jupyter Notebook, VSCode, or PyCharm (preferred for development and testing)
Visualization Tools: Matplotlib and Seaborn (for data visualization).

2 Implementation

2.1 Import Libraries

Key Libraries: Imports required libraries for data processing, visualization, PCA, and regression models (e.g., Pandas, Matplotlib, Scikit-learn).

```
!pip install scikit-learn
!pip install pandas
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2.2 Load and Clean Data

Load Data: Reads the dataset from an Excel file.Then clean Data which Removes irrelevant rows, renames columns, handles missing values, and converts numeric fields for aggregation and analysis.

```
# Load the dataset
file_path = "cleaned_accident_data (2)_1.csv"
data = pd.read_csv(file_path)

# Clean the dataset
data_cleaned = data.drop(index=0)  # Remove the header row
data_cleaned.columns = [
    "Year", "Vehicle_Type", "Fatal_Accidents", "GI_Accidents",
    "MI_Accidents", "NI_Accidents", "Total_Accidents",
    "Persons_Killed", "GI_Persons", "MI_Persons", "Total_Persons", "Misc"
]

# Convert columns to appropriate data types
data_cleaned = data_cleaned.dropna(how="all")  # Drop empty rows
data_cleaned["Year"] = pd.to_numeric(data_cleaned["Year"], errors="coerce").fillna(method="ffill").astype(int)
numeric_cols = [
    "Fatal_Accidents", "GI_Accidents", "MI_Accidents", "NI_Accidents",
    "Total_Accidents", "Persons_Killed", "GI_Persons", "MI_Persons", "Total_Persons"
]
data_cleaned[numeric_cols] = data_cleaned[numeric_cols].apply(pd.to_numeric, errors="coerce")
# Ensure columns for aggregation are numeric
numeric_columns = [
    "Fatal_Accidents", "GI_Accidents", "MI_Accidents", "NI_Accidents",
    "Total_Accidents", "Persons_Killed", "GI_Persons", "MI_Persons", "Total_Persons"]
```

2.3 Drop NA and Aggregate

Drop NA: Removes rows with missing values in critical columns (Total_Accidents, Persons_Killed). and aggregate groups data by year and sums up numeric columns for trend analysis.

```
# Drop rows with NaN values in critical columns (if necessary)
data = data.dropna(subset=["Total_Accidents", "Persons_Killed"])

# Aggregate data by year
annual_data = data.groupby('Year')[numeric_columns].sum()
```

2.4 Annual Trends with Percentage Change

Year-on-Year (YoY) Analysis: Computes percentage change in total accidents.Visualization combines line and bar plots to show accident counts and growth rates over years.

2.5 Accident Severity Trends with Area Plot

Data: Plots trends of different accident severities (Fatal, GI, MI, NI) over years using a stacked area plot.
Insight: Highlights the contribution of each severity type to total accidents.

```
# 3. Accident Severity Trends with Area Plot
severity_data = annual_data[['Fatal_Accidents', 'GI_Accidents', 'MI_Accidents', 'NI_Accidents']]
severity_data.plot(kind='area', stacked=True, figsize=(12, 6), alpha=0.7)
plt.title("Trends in Accident Severity")
plt.xlabel("Year")
plt.ylabel("Count")
plt.legend(title="Severity")
plt.grid(True)
plt.show()
```

2.6 Correlation Calculation

Correlation Calculation: Computes the correlation matrix for numerical columns in the dataset.
Heatmap Visualization: Displays a heatmap to identify relationships between variables using color intensity.

```
# 5. Correlation Heatmap
correlation_data = data[numeric_columns].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_data, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Accident Data")
plt.show()
```

2.7 PCA

Data Preparation: Selects numerical columns and standardizes them using StandardScaler. PCA: Applies Principal Component Analysis to reduce dimensions while retaining 95% variance.Thomas et al. (2024) Variance Visualization: Plots individual and cumulative explained variance for principalcomponents.

```
# Selecting numerical columns for PCA
numerical_columns = [
    "Fatal_Accidents", "GI_Accidents", "MI_Accidents",
    "NI_Accidents", "Total_Accidents", "Persons_Killed",
    "GI_Persons", "MI_Persons", "Total_Persons"
]
numerical_data = data[numerical_columns].dropna()

# Standardizing the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numerical_data)

# Applying PCA to reduce dimensions
pca = PCA(n_components=0.95)  # Retain 95% of the variance
pca_data = pca.fit_transform(scaled_data)

# Creating a DataFrame for the principal components
pca_df = pd.DataFrame(pca_data, columns=[f"PC{i+1}" for i in range(pca_data.shape[1])])

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_

# Visualizing the explained variance
plt.figure(figsize=(10, 6))
plt.bar(range(1, len(explained_variance) + 1), explained_variance, alpha=0.7, align='center', label='Individual Variance')
plt.step(range(1, len(explained_variance) + 1), explained_variance.cumsum(), where='mid', label='Cumulative Variance')
plt.xlabel('Principal Component Index')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance by Principal Components')
plt.legend(loc='best')
plt.tight_layout()
plt.show()

print("Explained Variance Ratios:", explained_variance)
pca_df.head()
```

2.8 Time Series Handling

Missing Data: Fills missing values using forward fill in the Total_Accidents column.
Grouping: Aggregates total accidents by year for time series analysis.

```
# Handle missing values by forward filling or interpolation
time_series_data = data_cleaned.groupby("Year")["Total_Accidents"].sum().reset_index()
time_series_data.set_index("Year", inplace=True)

# Fill missing values (if any) using forward fill
time_series_data["Total_Accidents"] = time_series_data["Total_Accidents"].fillna(method="ffill")
```

2.9 Stationarity Check (ADF Test)

ADF Test: Checks stationarity of Total_Accidents.
Differencing: Applies differencing to transform non-stationary data into stationary.

```
from statsmodels.tsa.stattools import adfuller

# Perform ADF test
result = adfuller(time_series_data["Total_Accidents"])
print("ADF Statistic:", result[0])
print("p-value:", result[1])

if result[1] > 0.05:
    # Apply differencing if non-stationary
    time_series_data_diff = time_series_data.diff().dropna()
```

2.10 Auto-SARIMA Model

Auto-ARIMA: Automatically selects the best SARIMA model orders for Total_Accidents.
Output: Displays optimal seasonal and non-seasonal parameters.Krishna et al. (2023)

```
from pmdarima import auto_arima

# Auto-SARIMA model
auto_model = auto_arima(time_series_data["Total_Accidents"], seasonal=True, trace=True, m=1)
print("Best Order:", auto_model.order)
print("Best Seasonal Order:", auto_model.seasonal_order)
```

2.11 Data Preparation and Modeling

Standardization: Scales PCA features to normalize data for modeling.
Train-Test Split: Splits data into training and testing sets (80-20).
Model Selection: Includes Random Forest, Decision Tree, Gradient Boosting, and KNN regressors.
Hyperparameter Tuning: Performs Grid Search for optimal parameters (e.g., for Random Forest and Gradient Boosting).
Metrics: Calculates MSE, RMSE, and MAPE to evaluate model performance.
Best Model: Selects the model with the lowest RMSE for predictions.
Display Results: Prints performance metrics and top 10 predictions (reverse log-transformed).

```
# ---- Step 3: Standardize the Features ----
scaler = StandardScaler()
X_scaled = scaler.fit_transform(pca_df)

# ---- Step 4: Split Data ----
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_log, test_size=0.2, random_state=42)

# ---- Step 5: Define and Tune Models ----
models = {
    "Random Forest": RandomForestRegressor(random_state=42),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
    "K-Nearest Neighbors (KNN)": KNeighborsRegressor()
}

# Grid search for hyperparameter tuning (example for Random Forest)
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'random_state': [42]
}
rf_grid_search = GridSearchCV(RandomForestRegressor(), param_grid_rf, cv=3, scoring='neg_mean_squared_error')
rf_grid_search.fit(X_train, y_train)
models["Random Forest"] = rf_grid_search.best_estimator_

# Example for Gradient Boosting
param_grid_gb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'random_state': [42]
}

gb_grid_search = GridSearchCV(GradientBoostingRegressor(), param_grid_gb, cv=3, scoring='neg_mean_squared_error')
gb_grid_search.fit(X_train, y_train)
models["Gradient Boosting"] = gb_grid_search.best_estimator_
# ---- Step 6: Train and Evaluate Models ----
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred_log = model.predict(X_test)
```

```
    # Reverse log transformation for predictions and ground truth
    y_pred = np.expm1(y_pred_log)
    y_test_original = np.expm1(y_test)

    # Evaluate
    mse = mean_squared_error(y_test_original, y_pred)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(y_test_original, y_pred)
    results[name] = {"MSE": mse, "RMSE": rmse, "MAPE": mape}

# ---- Step 7: Display Results ----
results_df = pd.DataFrame(results).T
print("Model Performance Metrics (MSE, RMSE, and MAPE):")
print(results_df)

# Display predictions from the best-performing model (based on RMSE)
best_model_name = results_df['RMSE'].idxmin()
best_model = models[best_model_name]
predictions = best_model.predict(X_test)
predictions_original = np.expm1(predictions)

print(f"\nPredictions from the Best Model ({best_model_name}):")
print(predictions_original[:10])
```

2.12 Model Evaluation and Visualization

Results DataFrame: Converts model performance metrics (MAPE, MSE, RMSE) into a structured DataFrame for visualization.
Plotting Metrics:Creates bar plots for MAPE, MSE, and RMSE across models and annotates bars with exact metric values for better readability.Choudhary et al. (2024)

2.13 Accident Prevention Assistant (LLM-based)

Initialization: Loads a knowledge base and sets up a BERT-based question-answering pipeline.
Context Handling: Splits large knowledge bases into manageable chunks for better accuracy.
Answer Questions: Processes user queries, finds relevant answers from chunks, and provides detailed explanations.
User Interaction: Continuous loop to take user inputs, fetch answers, and display confidence scores with context snippets.

```python
from transformers import pipeline

class AccidentPreventionLLM:
    def __init__(self, knowledge_base_path, model_name="bert-large-uncased-whole-word-masking-finetuned-squad"):
        """
        Initialize the LLM with the knowledge base and a question-answering model.
        :param knowledge_base_path: Path to the text file containing the knowledge base.
        :param model_name: Pretrained model to use for question answering.
        """
        # Load the knowledge base from the file
        with open(knowledge_base_path, 'r') as file:
            self.context = file.read()

        # Split context into manageable chunks for better QA accuracy
        self.context_chunks = self._split_context(self.context)

        # Initialize the question-answering pipeline
        self.qa_pipeline = pipeline("question-answering", model=model_name)

    def _split_context(self, context, max_chunk_size=500):
        """
        Split the knowledge base into smaller chunks for more accurate answers.
        :param context: The full knowledge base text.
        :param max_chunk_size: Maximum size of each chunk.
        :return: List of context chunks.
        """
        chunks = []
        for i in range(0, len(context), max_chunk_size):
            chunks.append(context[i:i + max_chunk_size])
        return chunks

    def answer_question(self, question):
        """
        Answer a question based on the knowledge base.
        :param question: The question string.
        :return: A detailed answer string.
        """
        best_response = {"answer": "", "score": -1, "chunk": ""}

        try:
            # Process each chunk and find the best response
            for chunk in self.context_chunks:
                response = self.qa_pipeline(question=question, context=chunk)
                if response["score"] > best_response["score"]:
                    best_response = {"answer": response["answer"], "score": response["score"], "chunk": chunk}
```

2.14 Save the LLM Model

Model Initialization: Loads the AccidentPreventionLLM with a specified knowledge base.
Model Saving: Uses Python's pickle to serialize and save the LLM instance into a file (accident_prevention_llm.pkl) for future use.

```
# prompt: save the llm model

import pickle

knowledge_base_path = r"C:\Users\justin joseph\preventing_accidents_measures.txt"
llm = AccidentPreventionLLM(knowledge_base_path)


# Save the model to a file
with open('accident_prevention_llm.pkl', 'wb') as f:
    pickle.dump(llm, f)

print("Model saved to accident_prevention_llm.pkl")
```

2.15 Simulated Response System

Response Logic: Simulates responses based on accident severity in prompts (e.g., fatalities vs. overall trends).
Interactive Assistant: Allows users to input criteria (vehicle type/year) to fetch filtered insights.

```
# Create natural language summaries from the data
def generate_prompt(row):
    return (f"In {row['Year']}, vehicles of type '{row['Vehicle_Type']}' were involved in "
            f"{row['Total_Accidents']} accidents, leading to {row['Persons_Killed']} fatalities and "
            f"affecting {row['Total_Persons']} persons overall.")

prompts = data.apply(generate_prompt, axis=1).tolist()

# Step 4: Local Response Function
# Simulate an LLM response locally using pre-defined logic
def simulate_llm_response(prompt):
    # A simple simulated response system for demonstration
    if "fatalities" in prompt:
        return "The data highlights significant safety concerns for this vehicle type."
    else:
        return "The accident trends seem to be consistent with overall expectations."

# Test the simulated response system with one prompt
sample_prompt = prompts[0]
print("Sample Prompt:", sample_prompt)
response = simulate_llm_response(sample_prompt)
print("Simulated Response:", response)

# Step 5: Create an interactive assistant
# Function to query insights
while True:
    print("\nEnter 'exit' to stop.")
    vehicle_type = input("Enter a vehicle type to get accident insights: ")
    if vehicle_type.lower() == 'exit':
        break

    year = input("Enter a year to filter (or type 'all'): ")
    if year.lower() != 'all':
        try:
            year = int(year)
            filtered_data = data[(data['Vehicle_Type'].str.lower() == vehicle_type.lower()) & (data['Year'] == year)]
        except ValueError:
            print("Invalid year entered. Please try again.")
            continue
    else:
        filtered_data = data[data['Vehicle_Type'].str.lower() == vehicle_type.lower()]

    if filtered_data.empty:
        print("No data found for the given criteria.")
    else:
        for _, row in filtered_data.iterrows():
            prompt = generate_prompt(row)
            response = simulate_llm_response(prompt)
            print(f"\nPrompt: {prompt}\nResponse: {response}")
```

References

Choudhary, A., Garg, R., Jain, S. and Khan, A. (2024). Impact of traffic and road infrastructural design variables on road user safety–a systematic literature review, *International Journal of Crashworthiness* **29**(4): 583–596.

Krishna, C., Singh, A. and Jha, K. (2023). Safety improvement on indian highways, *J Saf Eng* **12**(1): 1–12.

Thomas, R., Ajesh, F., John, A. and Sonia, K. (2024). Automated detection of traffic rule violation using deep learning techniques, *2024 IEEE Recent Advances in Intelligent Computational Systems (RAICS)* pp. 1–6.