Configuration Manual

MSc Research Project
Programme Name

Krutika Rajesh Gite
Student ID: x23164441

School of Computing
National College of Ireland

Supervisor: Prof. Athanasios Staikopoulos

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Krutika Rajesh Gite
Student ID:	23164441
Programme:	Master's in data Analytics
Year:	2024
Module:	Msc In Research Project
Supervisor:	Prof. Athanasios Staikopoulos
Submission Due Date:	12/12/2024
Project Title:	Predicting Energy Consumption in Electric Vehicles: A Machine Learning Approach for Enhanced Efficiency and Sustainability
Word Count:	
Page Count:	6

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: ……………………………………Krutika……Gite………………………

Date: ………………………12/12/2024………………………………………

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

Krutika Gite

Student ID:
x23164441

1 Introduction

This configuration manual shall present a step-by-step method on installation, implementation, and management of the framework used in the research project named as "Prediction of Energy Consumption in Electric Vehicles using Machine Learning." The ability to remake the project's processes is also pursued through fixing the need in such hardware; the ways of dealing with the data is fixed as well; lastly, the course of actions needed for the model to be executed is determined. It is meant to help the readers, the researchers, and professionals practicing in the field, to repeat the project and apply it to the range of similar domains. The project exploits ML platforms to estimate and forecast the energy consumption of an electric vehicle in functions of critical parameters inclusive but not limited to SOC, driving distance, and environmental factors. The major developmental stages include data preparation, model building and testing and result assessment.

2 Hardware Requirements

- **Processor**: Minimum Intel Core i5 or AMD equivalent
- **Memory (RAM)**: At least 8 GB (16 GB recommended for larger datasets)
- **Operating System**: Windows 10, macOS, or Linux (64-bit recommended)
- **Storage**: 20 GB of free disk space
- **GPU (optional)**: NVIDIA GPU with at least 4 GB VRAM for faster computation in clustering tasks

3 Software Requirements

Programming Language: Python 3.8 or later
IDE/Environment:

- Jupyter Notebook (via Anaconda Navigator or standalone)
- VS Code (Visual Studio Code)
- PyCharm

Additional Tools:

- A browser for opening Jupyter Notebook
- Git for version control (optional but recommended for collaborative work)

4 Library Package Requirements

The following Python libraries are required to execute the notebook. Use the pip command to

install them:
General Libraries

- pandas (Data manipulation and analysis)
- numpy (Numerical computations)
- seaborn (Data visualization)

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
```

Figure 1. Packages Used

5 Dataset Description

This research used the dataset from the U.S. Department of Energy's EV Data Collection Project as the source of the data for this research. It has the ability to provide published and non-published trip level data including SOC, total distance, idling time, temperature and total energy

6. Model Building

```
# Train a Random Forest Regressor
model = RandomForestRegressor(random_state=42, n_estimators=100)
model.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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

```
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
# Prepare evaluation results
evaluation_results = {
    "Mean Absolute Error": mae,
    "Mean Squared Error": mse,
    "R-squared Score": r2
}
```

```
{'Mean Absolute Error': 0.2607399812999907, 'Mean Squared Error': 0.7063271022928781, 'R-squared Score': 0.9925310115825272}
```

```
]: # Extract feature importances from the trained model
feature_importances = model.feature_importances_

# Create a DataFrame for visualization
importance_df = pd.DataFrame({
    "Feature": columns,
    "Importance": feature_importances
}).sort_values(by="Importance", ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(importance_df["Feature"], importance_df["Importance"], align='center')
plt.title("Feature Importance Analysis")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.gca().invert_yaxis()  # Invert y-axis to display the most important feature on top
plt.tight_layout()
plt.show()

# Print the feature importance DataFrame
print(importance_df)
```

```
n [37]: from sklearn.model_selection import GridSearchCV

# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

```
n [38]: # Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=42),
                           param_grid=param_grid,
                           cv=3,
                           n_jobs=-1,
                           scoring='neg_mean_squared_error',
                           verbose=2)
```

```
n [39]: # Fit the model to the data
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
```

```
ut[39]: GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=42), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]},
```

In [42]:
```
# Evaluate the optimized model
mae_optimized = mean_absolute_error(y_test, y_pred_optimized)
mse_optimized = mean_squared_error(y_test, y_pred_optimized)
r2_optimized = r2_score(y_test, y_pred_optimized)
```

In [43]:
```
# Compile the optimized results
optimized_results = {
    "Best Parameters": best_params,
    "Mean Absolute Error": mae_optimized,
    "Mean Squared Error": mse_optimized,
    "R-squared Score": r2_optimized
}

optimized_results
```

Out[43]:
```
{'Best Parameters': {'bootstrap': True,
  'max_depth': None,
  'min_samples_leaf': 1,
  'min_samples_split': 2,
  'n_estimators': 100},
 'Mean Absolute Error': 0.2607399812999907,
 'Mean Squared Error': 0.7063271022928781,
 'R-squared Score': 0.9925310115825272}
```

In [44]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
```

```python
    # Evaluate the model
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store the results
    results[model_name] = {
        "Mean Absolute Error": mae,
        "Mean Squared Error": mse,
        "R-squared Score": r2
    }

# Convert results to a DataFrame for easy visualization
results_df = pd.DataFrame(results).T

# Display the results
print(results_df)
```

```
                          Mean Absolute Error  Mean Squared Error  \
Linear Regression                    1.395749            5.347487
Decision Tree                        0.346654            1.910750
Gradient Boosting                    0.545158            1.807303
Support Vector Regressor             4.947851          105.878897

                          R-squared Score
Linear Regression                0.943454
Decision Tree                    0.979795
Gradient Boosting                0.980889
Support Vector Regressor        -0.119606
```

```
]: # Define the Gradient Boosting model
gbr_model = GradientBoostingRegressor(random_state=42)
```

```
]: # Perform 5-fold cross-validation
from sklearn.model_selection import train_test_split, cross_val_score
cv_scores = cross_val_score(gbr_model, X, y, cv=5, scoring='r2')
```

```
]:
# Calculate mean and standard deviation of cross-validation scores
cv_mean = cv_scores.mean()
cv_std = cv_scores.std()
```

```
]: # Display results
cross_validation_results = {
    "Cross-Validation Scores": cv_scores,
    "Mean R-squared Score": cv_mean,
    "Standard Deviation of R-squared": cv_std
}
```

```
]: # Convert results to a DataFrame
results_df = pd.DataFrame({
    "Fold": range(1, len(cv_scores) + 1),
    "R-squared Score": cv_scores
})
```

```
]: print(f"Mean R-squared: {cv_mean:.4f}, Standard Deviation: {cv_std:.4f}")

Mean R-squared: 0.8386, Standard Deviation: 0.2656
```


```
    "Cross-Validation Scores": cv_scores,
    "Mean R-squared Score": cv_mean,
    "Standard Deviation of R-squared": cv_std
}
```

```
In [54]: # Convert results to a DataFrame
results_df = pd.DataFrame({
    "Fold": range(1, len(cv_scores) + 1),
    "R-squared Score": cv_scores
})
```

```
In [55]: print(f"Mean R-squared: {cv_mean:.4f}, Standard Deviation: {cv_std:.4f}")

Mean R-squared: 0.8386, Standard Deviation: 0.2656
```


```
In [56]: print(results_df)

   Fold  R-squared Score
0     1         0.959711
1     2         0.953736
2     3         0.984502
3     4         0.986865
4     5         0.308027
```