Model Code and Documentation

1. Introduction

This section outlines the code used to build, train, and test the predictive model. It includes explanations of key methodologies and steps taken during development.

Model Code and Documentation

1. Introduction

This section outlines the code used to build, train, and test the predictive model. It includes explanations of key methodologies and steps taken during development.

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import seaborn as sns

import matplotlib.pyplot as plt
```

3. Data Loading

Load the datasets

```
# Load datasets

Dtrain = pd.read_csv("C://Users// Desktop//GST//Dataset//X_Train_Data_Input.csv")

Dtest = pd.read_csv("C://Users// Desktop//GST//Dataset//X_Test_Data_Input.csv")

Ytrain = pd.read_csv("C://Users// Desktop//GST//Dataset//Y_Train_Data_Target.csv")

Ytest = pd.read_csv("C://Users// Desktop//GST//Dataset//Y_Test_Data_Target.csv")
```

4. Data Preprocessing

Handle missing values and categorical encoding

```
# Fill missing values with mean for numeric columns

for column in Dtrain.select_dtypes(include=[np.number]).columns:

    Dtrain[column].fillna(Dtrain[column].mean(), inplace=True)

# Fill missing values with mode for categorical columns

for column in Dtrain.select_dtypes(include=[object]).columns:

    Dtrain[column].fillna(Dtrain[column].mode()[0], inplace=True)


# One-hot encoding for categorical variables
```

```
Dtrain = pd.get_dummies(Dtrain)

Dtest = pd.get_dummies(Dtest)

# Align Dtest with Dtrain

Dtest = Dtest.reindex(columns=Dtrain.columns, fill_value=0)
```

5. Splitting the Data

Create training and validation sets

```
X_train = Dtrain.values

y_train = Ytrain.values.flatten()

# Split the data into training and validation sets

X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

6. Model Development

Train the model

```
# Initialize the model

model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data

model.fit(X_train_split, y_train_split)
```

7. Model Evaluation

Evaluate model performance

```
# Predictions

y_pred_train = model.predict(X_train_split)

y_pred_val = model.predict(X_val_split)

# Calculate evaluation metrics

mse = mean_squared_error(y_val_split, y_pred_val)

r2 = r2_score(y_val_split, y_pred_val)

mae = mean_absolute_error(y_val_split, y_pred_val)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

print("Mean Absolute Error:", mae)
```

8. Predictions on Test Data

Generate predictions for the test dataset

```
# Predictions on the test set
y_pred_test = model.predict(Dtest)
# Output predictions
output = pd.DataFrame({'Predicted': y_pred_test})
output.to_csv("Predictions.csv", index=False)
```

9. Visualizations

Feature Importance

```
# Plot feature importance
importances = model.feature_importances_
features = Dtrain.columns
plt.figure(figsize=(10, 6))
plt.barh(features, importances)
plt.title('Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```

10. Key Methodology

- **Data Preparation**: Handled missing values and encoded categorical variables to ensure the model can process the input data effectively.
- **Model Selection**: Chose Random Forest Regressor due to its robustness and ability to handle various data types and distributions.
- **Evaluation**: Employed metrics like MSE, R^2, and MAE to quantify model performance and ensure generalizability on unseen data.

11. Conclusion

- The code provided demonstrates a structured approach to building a predictive model. The methodology emphasizes thorough data preprocessing, model training, evaluation, and analysis of results, ensuring a comprehensive framework for predictive modeling tasks.