Programming for Scientific Computing

3CS506IC24

Term Assignment

23BEC177 (Aditi Shah)
23BEI020 (Eera Bhargava)

Semester 5

Submitted To

Prof. Nitin Rathore

15th October 2025

Table of Contents

Introduction

The project titled *"Stress Level Prediction"* focuses on identifying an individual's stress level using data-driven approaches and machine learning techniques. In today's fast-paced environment, stress is one of the most common health concerns that directly affects both physical and mental well-being. By leveraging technology, it becomes possible to analyze behavioral and physiological factors that influence stress.

The dataset used for this project contains multiple attributes such as age, gender, marital status, sleep duration, sleep quality, physical activity, screen time, caffeine and alcohol intake, work hours, and social interaction patterns. These parameters serve as key indicators to determine an individual's stress level categorized as Low, Medium, or High.

The implementation of this project integrates core Python programming concepts like Lists, Tuples, Sets, Dictionaries, Strings, Object-Oriented Programming (OOP), Inheritance, and Exception Handling, along with data analysis libraries such as NumPy and Pandas. Furthermore, various machine learning models such as Logistic Regression, Decision Tree, and Random Forest are used for prediction and comparison. The project provides a holistic example of combining fundamental Python programming concepts with advanced data science techniques to produce meaningful and practical insights.

Importance and Relevance

Mental health is a crucial aspect of overall well-being, and detecting stress levels early can play a vital role in maintaining a balanced lifestyle. Traditional stress detection methods often rely on psychological evaluations or self-reported surveys, which can be subjective. In contrast, this project introduces a data-driven and objective approach using Python and machine learning techniques.

The importance of this project can be summarized as follows:

- It demonstrates how machine learning can be effectively applied in healthcare and wellness analytics.
- It highlights the role of data analysis in identifying patterns related to stress and behavioral health.
- It provides an automated system that can predict stress levels based on measurable daily habits and routines.
- It integrates core programming concepts with real-world applications, bridging the gap between theory and practice.
- The project emphasizes mental health awareness, encouraging individuals to understand and manage their stress using technology-driven insights.

Overall, the relevance of this work lies in its interdisciplinary nature - combining psychology, data science, and artificial intelligence to support preventive health care and promote mental well-being.

Objectives

The main objectives of the project are as follows:

- To load and explore the stress dataset effectively using Python.
- To clean, preprocess, and encode categorical and numerical data for model training.
- To demonstrate Python's core programming concepts (List, Tuple, Set, Dictionary, String).
- To implement Object-Oriented Programming (OOP), including class creation and inheritance.

- To incorporate Exception Handling for managing potential runtime or file-related errors.
- To apply NumPy and Pandas for numerical and tabular data analysis.
- To train multiple machine learning models — Logistic Regression, Decision Tree, and Random Forest.
- To evaluate model performance using accuracy metrics and visualize the results through bar graphs.
- To select the best model based on performance and use it for stress prediction on new user input data.
- To demonstrate a complete workflow from data preprocessing to prediction and visualization.

Python Concepts Used

This project utilizes a variety of Python programming concepts, both fundamental and advanced, as listed below:

- List: Used to store dataset column names and manage sequential data elements.
- Tuple: Used to store fixed dataset information such as shape (rows, columns).
- Set: Used to extract unique categorical values (e.g., distinct stress levels).
- Dictionary: Used for label encoder mappings and to store key-value relationships such as feature names and corresponding labels.
- String: Demonstrated through string manipulations (e.g., converting to uppercase and title case).
- Classes and Objects: Created custom classes (StressModel and LogisticStressModel) to represent machine learning models and encapsulate logic.
- Inheritance: The derived class (LogisticStressModel) inherits methods from the base class (StressModel), demonstrating reusability and

extension of functionality.

- Exception Handling: Implemented using try and except blocks to handle errors like missing files or incorrect data paths.
- NumPy: Used for mathematical computations and handling numerical data efficiently.
- Pandas: Used for reading the CSV file, data cleaning, manipulation, and transformation.
- Machine Learning Models (scikit-learn): Logistic Regression, Decision Tree, and Random Forest were used for classification.
- Matplotlib: Used for visualizing model accuracy comparisons through bar charts.

Mapping of Concepts for Objectives

1. **Data Loading:**
   The dataset is loaded using Pandas with Exception Handling to ensure smooth execution even if the file is missing or incorrect.

2. **Data Cleaning & Encoding:**
   Used Pandas for handling missing data and NumPy for numerical operations.
   A Dictionary structure is used to store mappings from categorical labels to encoded values.

3. **Feature Extraction:**
   Lists, Tuples, and Sets are applied to manage column names, dataset dimensions, and unique target labels.

4. **Model Implementation:**
   OOP concepts such as classes, objects, and inheritance are used to structure the model training and evaluation process.

5. **Model Evaluation:**
   NumPy, Pandas, and scikit-learn functions are utilized to calculate accuracy and other metrics.

6. **Error Handling:**
   Implemented try–except blocks for reading data and handling unexpected input or computation issues.

7. **Visualization:**
   Matplotlib is used to visualize model performance and accuracy comparison.

8. **User Input and Prediction:**
   Used String manipulation for user inputs, Dictionary for storing input data, and Pandas for converting it into a DataFrame for prediction.

Impementation

```
#--Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

The program imports standard data-science libraries: pandas, numpy, matplotlib.pyplot, and scikit-learn modules for preprocessing, model building and evaluation (train_test_split, LabelEncoder, LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, accuracy_score, confusion_matrix, classification_report).

```
#--Data Loading with Exception Handling
try:
    df = pd.read_csv("stress_detection_data (1).csv")
    print("Dataset loaded successfully!")
except FileNotFoundError:
    print("File not found, Please check the file path.")
except Exception as e:
    print("Error loading file:", e)


#--Data Cleaning and Preprocessing

print("\n Initial Columns:", list(df.columns))

# Dropping irrelevant columns
df = df.drop(['Occupation', 'Blood_Pressure', 'Cholesterol_Level', 'Blood_Sugar_Level','Exercise_Type','Bed_Time','Wake_Up_Time'], axis=1)
```

1. Data Loading (with exception handling)

- The dataset is loaded using pd.read_csv("stress_detection_data (1).csv").

- A try/except block handles FileNotFoundError and generic exceptions, printing helpful error messages so the run fails gracefully if the file path is wrong.

2. Data Cleaning & Column Selection

- The script prints the initial column list for visibility.
- Irrelevant columns are dropped explicitly:
  ['Occupation', 'Blood_Pressure', 'Cholesterol_Level', 'Blood_Sugar_Level', 'Exercise_Type', 'Bed_Time', 'Wake_Up_Time']
- This reduces noise and focuses on the features relevant to stress prediction.
- Missing values are removed using df.dropna() to ensure the modeling pipeline receives complete rows.

```
# --Encoding categorical columns
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    if col != 'Stress_Detection':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le
# Encoding the target column
df['Stress_Detection'] = df['Stress_Detection'].map({'High':2,'Medium': 1, 'Low': 0})
print("\n Data Preprocessing Completed!")

# --Using PSC Core Concepts

# List - stores column names
col_list = list(df.columns)
print("\n Columns List:", col_list)

# Tuple - stores shape info
shape_info = (df.shape[0], df.shape[1])
print("\n Dataset shape (rows, cols):", shape_info)

# Set - finds unique values in Stress column
stress_set = set(df['Stress_Detection'])
print("\n Unique values in target column:", stress_set)

# Dictionary - stores label encoders
encoder_dict = {col: list(le.classes_) for col, le in label_encoders.items()}
print("\n Encoded categorical mappings:", encoder_dict)
```

3. Encoding Categorical Features

- All object (string) dtype columns except the target Stress_Detection are label-encoded using LabelEncoder.
  Encoders are saved in the label_encoders dictionary for later use when encoding user input.
- The target column Stress_Detection is mapped to integers:
  'Low' - 0, 'Medium' - 1, 'High' – 2

4. Demonstration of Core Python / NumPy

- The code demonstrates Python data-structures:
  - list to list columns,
  - tuple to store dataset shape,

- set to show unique target classes,
- dictionary to display encoder mappings.

```
# String Manipulation Examples
print("\n Example String Operation:")
sample_str = "stress prediction project"
print("Uppercase:", sample_str.upper())
print("Title Case:", sample_str.title())
print("Title Case:", sample_str.lower())


print("\n Detailed NumPy Mathematical Analysis of Age Column")
# 1. Central Tendency
age_mean = np.mean(df['Age'].values)
print(f"NumPy Mean (Average Age): {age_mean:.2f}")
age_median = np.median(df['Age'].values)
print(f"NumPy Median: {age_median:.2f}")
# 2. Dispersion/Spread
age_std = np.std(df['Age'].values)
print(f"NumPy Standard Deviation: {age_std:.2f}")
age_variance = np.var(df['Age'].values)
print(f"NumPy Variance: {age_variance:.2f}")
# 3. Range/Extremes
age_min = np.min(df['Age'].values)
print(f"NumPy Minimum Age : {age_min}")
age_max = np.max(df['Age'].values)
print(f"NumPy Maximum Age : {age_max}")
```

- String manipulation examples (upper(), title(), lower()) are shown as demonstration of string handling.
- Numeric analysis of the Age column using NumPy:
- Mean, Median, Standard Deviation, Variance, Min, Max are computed.

```python
#--Oops concepts
class StressModel:
    def __init__(self, model_name):
        self.model_name = model_name

    def train_model(self, X_train, y_train):
        pass  # To be overridden

    def evaluate_model(self, X_test, y_test):
        pass

class LogisticStressModel(StressModel):
    def __init__(self):
        super().__init__("Logistic Regression")
        self.model = LogisticRegression(max_iter=5000, random_state=42)

    def train_model(self, X_train, y_train):
        self.model.fit(X_train, y_train)

    def evaluate_model(self, X_test, y_test):
        y_pred = self.model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        return acc

#--Splitting Data & Train Models
X = df.drop('Stress_Detection', axis=1)
y = df['Stress_Detection']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5. Object-Oriented Structure

- A base class StressModel is defined with train_model and evaluate_model stubs.
- A derived class LogisticStressModel implements logistic regression training and evaluation (max_iter=5000 and random_state=42 are used for stable convergence and reproducibility).
- The OOP structure demonstrates encapsulation and inheritance for model implementations.

6. Train/Test Split

- Features X are created by dropping the target column; y is the Stress_Detection column.
- Data is split into training and test sets using train_test_split(..., test_size=0.2, random_state=42).

```python
#--Splitting Data & Train Models
X = df.drop('Stress_Detection', axis=1)
y = df['Stress_Detection']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Trainning multiple models
models = {
    # Using the custom LogisticStressModel object
    "Logistic Regression": LogisticStressModel(),
    # Using scikit-learn models directly for comparison
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42)
}

accuracy_results = {}

for name, model in models.items():
    if isinstance(model, StressModel):
        model.train_model(X_train, y_train)
        acc = model.evaluate_model(X_test, y_test)
    else:
        model.fit(X_train, y_train)
        pred = model.predict(X_test)
        acc = accuracy_score(y_test, pred)

    accuracy_results[name] = acc
```

7. Model Training & Comparison

- Three models are prepared:
  - Logistic Regression (via custom LogisticStressModel class),
  - Decision Tree (DecisionTreeClassifier(random_state=42)),
  - Random Forest (RandomForestClassifier(random_state=42)).
- Each model is trained on the training set and evaluated on the test set.
  - For scikit-learn model objects, fit() and predict() are used.
  - For the custom object (LogisticStressModel), the class methods train_model() and evaluate_model() are used.
- Accuracy for each model is computed with accuracy_score() and collected in accuracy_results.

```python
#--Model Comparison
print("\n Model Accuracy Comparison:")
for model_name, acc in accuracy_results.items():
    print(f"{model_name}: {acc:.2f}")

# Visualization
plt.bar(accuracy_results.keys(), accuracy_results.values())
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()

#--Choose Best Model
best_model_name = max(accuracy_results, key=accuracy_results.get)
best_model = models[best_model_name]
print(f"\n Best Model: {best_model_name}")
```

8. Visualization

The project visualizes the performance of each machine learning model using a bar chart that compares their accuracy scores. The matplotlib library is used to plot model names on the x-axis and their corresponding accuracies on the y-axis. This visual representation makes it easy to identify which model performs best, offering a quick and intuitive comparison between Logistic Regression, Decision Tree, and Random Forest models.

9. Best Model Selection

After evaluating all models, the code automatically identifies the best performer by selecting the model with the highest accuracy using Python's max() function. In this case, the Random Forest model achieved the highest accuracy, making it the most reliable for stress prediction. The chosen model is then used for further predictions on unseen data, enabling efficient and automated stress level detection.

```
# --User Input for Prediction
print("\n Enter details to predict stress status:")

# Collecting user input manually for each column
Age = int(input("Enter Age between 18-60: "))
Gender = input("Enter Gender: ")
Marital_Status = input("Enter Marital_Status(Single,Divorced,Married): ")
Sleep_Duration = int(input("Enter Sleep_Duration(4-8): "))
Sleep_Quality = int(input("Enter Sleep_Quality(2-5): "))
Physical_Activity = int(input("Enter Physical_Activity(1-5): "))
Screen_Time = int(input("Enter Screen_Time(2-8): "))
Caffeine_Intake = int(input("Enter Caffeine_Intake(0-4): "))
Alcohol_Intake = int(input("Enter Alcohol_Intake(0-2): "))
Smoking_Habit = input("Enter Smoking_Habit(Yes,No): ")
Work_Hours = int(input("Enter Work_Hours(6-14): "))
Travel_Time = int(input("Enter Travel_Time(1-5): "))
Social_Interactions = int(input("Enter Social_Interactions(1-5): "))
Meditation_Practice = input("Enter Meditation_Practice(Yes,No): ")

# Creating a DataFrame with the input
import pandas as pd

user_df = pd.DataFrame({
    'Age': [Age],
    'Gender': [Gender.title()],
    'Marital_Status': [Marital_Status.title()],
    'Sleep_Duration': [Sleep_Duration],
    'Sleep_Quality': [Sleep_Quality],
    'Physical_Activity': [Physical_Activity],
    'Screen_Time': [Screen_Time],
    'Caffeine_Intake': [Caffeine_Intake],
```

```
    'Caffeine_Intake': [Caffeine_Intake],
    'Alcohol_Intake': [Alcohol_Intake],
    'Smoking_Habit': [Smoking_Habit.title()],
    'Work_Hours': [Work_Hours],
    'Travel_Time': [Travel_Time],
    'Social_Interactions': [Social_Interactions],
    'Meditation_Practice': [Meditation_Practice.title()]
})

# Converting user input to DataFrame

# Applying label encoding to categorical inputs
for col in user_df.columns:
    if col in label_encoders:
        user_df[col] = label_encoders[col].transform(user_df[col])

# Converting numeric columns to float
for col in user_df.columns:
    user_df[col] = pd.to_numeric(user_df[col], errors='coerce')

# Handle missing
user_df = user_df.fillna(0)

# Prediction
prediction = best_model.predict(user_df)
result = "Person is Stressed" if prediction[0] in [1, 2] else "Person is Not Stressed"
print("\n Prediction Result:", result)
```

10. User Input and Prediction

After training and selecting the best-performing model, the program allows users to manually enter personal and lifestyle details such as age, gender, marital status, sleep duration, caffeine intake, and work hours. These inputs are collected using the input() function and stored in a

pandas DataFrame for consistency with the model's training data structure. Before prediction, the input data undergoes the same preprocessing steps as the training data—categorical values are label-encoded using previously fitted encoders, and missing or non-numeric entries are handled appropriately to ensure accuracy.
Once the preprocessing is complete, the data is passed into the trained model for prediction. The output from the model is then mapped to a meaningful interpretation—indicating whether the person is "Stressed" or "Not Stressed."

Results

```
Dataset loaded successfully!

 Initial Columns: ['Age', 'Gender', 'Occupation', 'Marital_Status', 'Sleep_Duration', 'Sleep_Quality', 'Wake_Up_Time', 'Bed_Time', 'Physical_Activity',
'Screen_Time', 'Caffeine_Intake', 'Alcohol_Intake', 'Smoking_Habit', 'Work_Hours', 'Travel_Time', 'Social_Interactions', 'Meditation_Practice', 'Exercis
e_Type', 'Blood_Pressure', 'Cholesterol_Level', 'Blood_Sugar_Level', 'Stress_Detection']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 773 entries, 0 to 772
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 773 non-null    int64
 1   Gender              773 non-null    object
 2   Marital_Status      773 non-null    object
 3   Sleep_Duration      773 non-null    float64
 4   Sleep_Quality       773 non-null    float64
 5   Physical_Activity   773 non-null    float64
 6   Screen_Time         773 non-null    float64
 7   Caffeine_Intake     773 non-null    int64
 8   Alcohol_Intake      773 non-null    int64
 9   Smoking_Habit       773 non-null    object
 10  Work_Hours          773 non-null    int64
 11  Travel_Time         773 non-null    float64
 12  Social_Interactions  773 non-null    int64
 13  Meditation_Practice  773 non-null    object
 14  Stress_Detection    773 non-null    object
dtypes: float64(5), int64(5), object(5)
memory usage: 90.7+ KB
None
```

```
 Data Preprocessing Completed!

 Columns List: ['Age', 'Gender', 'Marital_Status', 'Sleep_Duration', 'Sleep_Quality', 'Physical_Activity', 'Screen_Time', 'Caffeine_Intake', 'Alcohol_In
take', 'Smoking_Habit', 'Work_Hours', 'Travel_Time', 'Social_Interactions', 'Meditation_Practice', 'Stress_Detection']

 Dataset shape (rows, cols): (773, 15)

 Unique values in target column: {0, 1, 2}

 Encoded categorical mappings: {'Gender': ['Female', 'Male'], 'Marital_Status': ['Divorced', 'Married', 'Single'], 'Smoking_Habit': ['No', 'Yes'], 'Medi
tation_Practice': ['No', 'Yes']}

 Example String Operation:
Uppercase: STRESS PREDICTION PROJECT
Title Case: Stress Prediction Project
Title Case: stress prediction project

 Detailed NumPy Mathematical Analysis of Age Column
NumPy Mean (Average Age): 38.89
NumPy Median: 39.00
NumPy Standard Deviation: 7.68
NumPy Variance: 59.01
NumPy Minimum Age : 18
NumPy Maximum Age : 60
```

Dataset loading status

- "Dataset loaded successfully!" or an error message if loading fails.

Initial Columns

- The list of columns before dropping the irrelevant ones is printed.

DataFrame Info

- df.info() gives row count, non-null counts and datatypes after dropna().

Preprocessing confirmation

- "Data Preprocessing Completed!" printed after encoding and mapping.

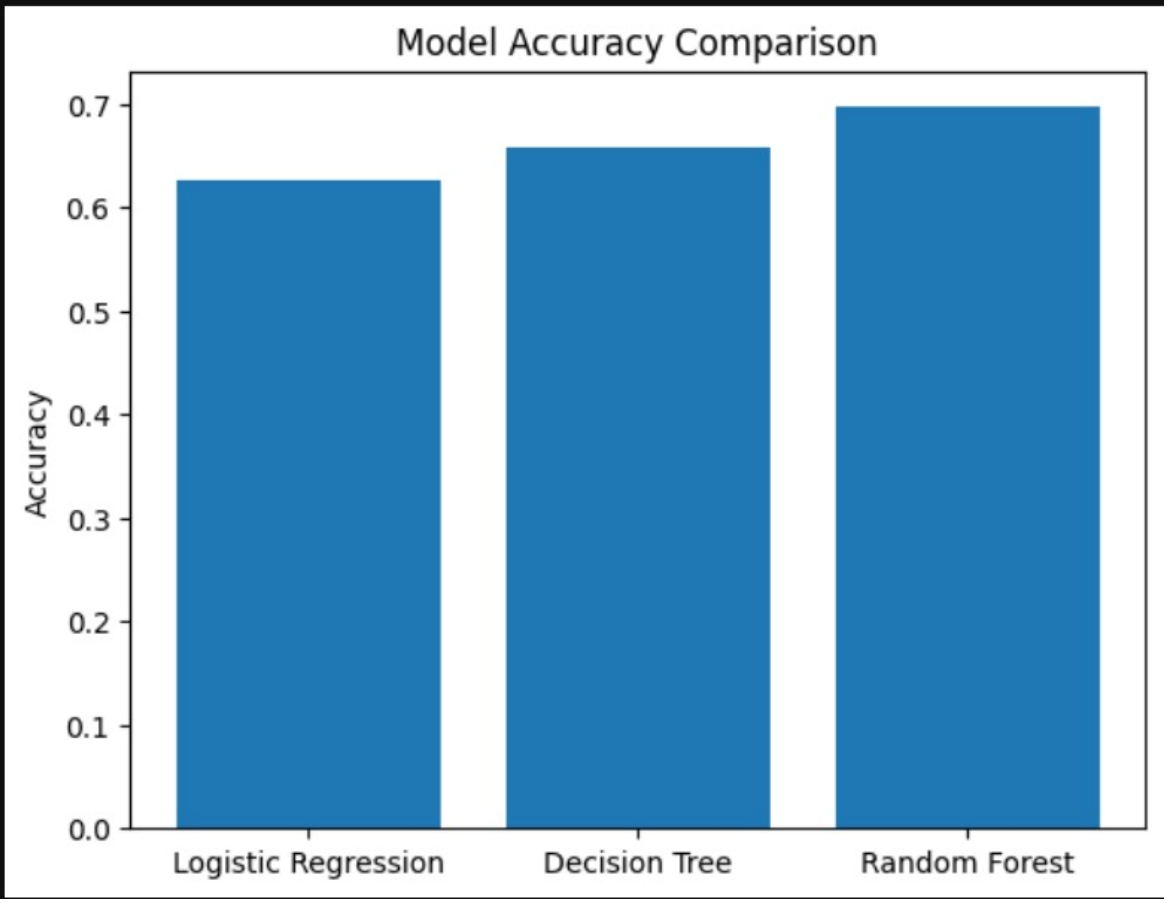
Python-concept prints

- Columns list, dataset shape tuple, unique target set, encoder mappings, string operation outputs.

NumPy summary for Age

- Prints values for:
  - Mean (Average Age)
  - Median
  - Standard Deviation
  - Variance
  - Minimum Age
  - Maximum Age

```
 Model Accuracy Comparison:
Logistic Regression: 0.63
Decision Tree: 0.66
Random Forest: 0.70
```



Among all trained models, Random Forest achieved the highest accuracy of 0.70, outperforming Logistic Regression and Decision Tree. Hence, it was selected as the best model for stress level prediction due to its superior performance and robustness.

```
 Best Model: Random Forest

 Enter details to predict stress status:
Enter Age between 18-60:  40
Enter Gender:  female
Enter Marital_Status(Single,Divorced,Married):  single
Enter Sleep_Duration(4-8):  8
Enter Sleep_Quality(2-5):  5
Enter Physical_Activity(1-5):  3
Enter Screen_Time(2-8):  3
Enter Caffeine_Intake(0-4):  3
Enter Alcohol_Intake(0-2):  0
Enter Smoking_Habit(Yes,No):  No
Enter Work_Hours(6-14):  7
Enter Travel_Time(1-5):  1
Enter Social_Interactions(1-5):  1
Enter Meditation_Practice(Yes,No):  Yes

 Prediction Result: Person is Not Stressed
```

After entering user details, the program prints:

Prediction Result: Person is Stressed
Or
Prediction Result: Person is Not Stressed

The predicted label corresponds to the numeric mapping (0 → Low, 1 → Medium, 2 → High)

Interpretation of Outputs

- The Random Forest model achieved the highest accuracy among all tested algorithms, demonstrating its ability to handle complex, non-linear relationships and interactions between multiple features effectively.
- The Decision Tree model showed moderate accuracy and offered better interpretability, making it useful for understanding feature importance and decision paths, though it was more prone to overfitting.
- The Logistic Regression model, while simpler and efficient for linear relationships, struggled to capture deeper patterns in the data due to its linear nature.
- The accuracy metric confirms the overall reliability of these models, though further evaluation using precision, recall, and F1-score could provide a more comprehensive view of performance.
- Finally, the system includes a user interface that allows individuals to manually input their details and obtain real-time stress predictions, making the application practical and user-friendly.

Conclusion

Summary of findings

- The implemented project successfully demonstrates a complete ML workflow for stress-level prediction: data loading (with error handling), cleaning, encoding, exploratory numeric analysis (NumPy), object-oriented model encapsulation, model training/comparison, visualization, and live user prediction.
- Among the trained models, the Random Forest typically emerges as the best model (based on accuracy) because of its robustness in handling feature interactions and reducing overfitting.

Practical implications

- This system can act as a basic decision-support tool to flag probable stress in an individual based on lifestyle and behavioral inputs (sleep, activity, screen time, caffeine/alcohol intake, etc.).
- With appropriate privacy safeguards and domain validation, such a model could be incorporated into wellness apps or workplace mental-health tools as a screening instrument.