Final Project Report

Project Name: EduPredi

Prepared By : Meaad Farag Bayuosef

Contents

Abstract

EduPredi is a web-based system designed for educational data analysis and prediction. utilizing Django, a high-level Python web framework, this system facilitates the uploading of student data through CSV files, feature selection by users, and subsequently provides predictions using decision tree algorithms and linear regression models. This report details the development, implementation, and testing of the system, along with a discussion on its potential impacts in educational environments.

Introduction

- **Project Name**

EduPredi stands for "Educational Prediction System." The name reflects the project's primary goal of predicting student performance using advanced machine learning techniques. By combining "Edu" (Education) with "Predi" (Prediction), the name succinctly conveys its purpose: to provide educators with predictive insights to support student success.

- **Background**

Educational institutions increasingly rely on predictive models to enhance student learning outcomes and retention rates. EduPredi is developed as a tool to assist educators and administrators by providing insights into student performance trends.

- **Problem Statement**

The challenge addressed by EduPredi is the need for a reliable, easy-to-use system that can predict student performance to help educators tailor their teaching approaches to individual needs.

- **Objectives**

- To develop a web-based application that can process educational data.
- To implement machine learning models that predict student outcomes based on historical data.
- To provide a user-friendly interface for user to upload data and receive predictions.

Literature Review for EduPredi

Introduction

EduPredi, a Django-based predictive analytics system, aims to transform educational strategies by providing insights into student performance. This literature review explores the theoretical and empirical foundations that support the use of predictive analytics in education, highlights the contributions of various technologies, and contextualizes the advancements made by EduPredi within this burgeoning field.

Predictive Analytics in Education

Predictive analytics in education leverages historical data and machine learning techniques to forecast student outcomes, providing an opportunity for early intervention and personalized learning strategies. According to Baker and Inventado (2014), educational data mining and learning analytics are pivotal in understanding and improving student learning experiences. They argue that these technologies can identify at-risk students earlier and more accurately than traditional methods.

Romero and Ventura (2013) discuss how predictive models can be integrated into educational systems to enhance decision-making processes. They emphasize that the integration of these models into educational practices can lead to significant improvements in both teaching strategies and student outcomes.

Machine Learning Techniques in Predictive Systems

The use of various machine learning algorithms enhances the ability to address different types of data and prediction requirements. As noted by James et al. (2013), logistic regression, decision trees, and ensemble methods like random forests offer diverse approaches to modeling educational data, each with its strengths and limitations in terms of complexity, interpretability, and accuracy. EduPredi's approach to integrating multiple algorithms allows it to adapt to various educational datasets, optimizing prediction accuracy based on the nature of the data provided.

Challenges and Solutions in Educational Predictive Analytics

One of the main challenges in implementing predictive analytics in education is ensuring data quality and handling diverse data types, from numerical grades to categorical data such as attendance records. As discussed by Dietterich (2000), preprocessing techniques such as feature scaling and encoding categorical variables are crucial for preparing data for effective machine learning applications. EduPredi addresses these challenges by incorporating robust data preprocessing methods that enhance model training and prediction outcomes.

Furthermore, the system's flexibility in handling multiple datasets addresses another significant challenge: the generalization of predictive models across different educational contexts. By allowing educators to upload custom datasets, EduPredi ensures that the insights and predictions are tailored to the specific educational environment, thereby increasing the relevance and applicability of the predictions.

User Interaction and System Usability

An essential aspect of educational technology is its usability. Nielsen (1994) highlights the importance of user-centered design in creating effective and accessible technologies. EduPredi's development of a straightforward, intuitive user interface ensures that educators, students, and parents can easily interact with the system without requiring extensive technical knowledge. This focus on user experience is vital for the adoption and practical use of advanced analytic tools in educational settings.

Future Directions and Innovations

Looking forward, the integration of more advanced machine learning models, such as neural networks and deep learning, could provide even more nuanced insights into student performance and educational needs. These technologies, as explored by Goodfellow, Bengio, and Courville (2016), offer the potential for handling larger datasets and more complex feature interactions, paving the way for more sophisticated predictive capabilities.

Literature Review Conclusion

The literature supports the methodologies and technologies employed by EduPredi, underscoring the potential of predictive analytics to revolutionize educational practices. By integrating advanced data analysis techniques and focusing on user accessibility, EduPredi stands at the forefront of educational innovation, poised to make a significant impact on educational outcomes and strategies.

Literature Review References

- Baker, R. S., & Inventado, P. S. (2014). *Educational data mining and learning analytics*. Springer.
- Romero, C., & Ventura, S. (2013). *Data mining in education*. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 3(1), 12-27.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*. Springer.
- Dietterich, T. G. (2000). *Ensemble methods in machine learning*. Multiple classifier systems, 1857, 1-15.
- Nielsen, J. (1994). *Usability engineering*. Academic Press.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Methodology

1. Selection of Technologies and Frameworks

Initially, the project considered using Flask for the backend and React for the frontend to leverage their respective strengths in quick API development and responsive user interfaces. However, these technologies were ultimately not used. Instead, Django was selected as the sole web framework due to its all-encompassing nature, which simplifies project management by integrating front and back-end functionalities. Django's built-in features also support rapid development and clear structuring of web applications, which was deemed essential for meeting project timelines and requirements.

For data processing and machine learning, Python's libraries—Pandas for data manipulation and Scikit-Learn for model training—were retained due to their robust capabilities and ease of integration into Django.

2. Enhancement of Predictive Capabilities

EduPredi project employs three distinct algorithms—Decision Tree ID3, Decision Tree C4.5, and Linear Regression—to ensure a comprehensive analysis and prediction of student performance. Here's a brief justification for the use of each:

1) **Decision Tree ID3**:
   - **Suitability for Categorical Data**: The ID3 algorithm is efficient in handling categorical data, making it useful for datasets where the features are primarily categorical.
   - **Interpretability**: The ID3 model generates easy-to-understand decision rules, which can be valuable for educators to interpret the model's predictions.
   - **Foundation for Improvement**: Despite its simplicity and lower accuracy compared to other models, using ID3 provides a baseline for evaluating more advanced decision tree algorithms.

2) **Decision Tree C4.5**:

- **Handling Mixed Data Types**: The C4.5 algorithm can handle both continuous and categorical data, making it more versatile and accurate for complex educational datasets.
- **Improved Accuracy**: C4.5 typically outperforms ID3 due to its ability to handle continuous data and its use of gain ratio for attribute selection, which mitigates the bias towards attributes with more levels.
- **Enhanced Decision-Making**: By providing more accurate and nuanced predictions, C4.5 helps educators make better-informed decisions regarding student interventions.

3) **Linear Regression**:

- **Modeling Linear Relationships**: Linear Regression is used to model and predict the numerical outcome based on the linear relationships between features. This is particularly useful for predicting continuous variables such as final grades.
- **Simplicity and Efficiency**: The linear regression model is computationally efficient and provides straightforward interpretability of how each feature influences the predicted outcome.
- **Complementary Analysis**: It serves as a complementary method to the decision trees, offering a different perspective on the data and enhancing the robustness of the predictive analysis by comparing linear and non-linear relationships.

By incorporating these three algorithms, EduPredi leverages the strengths of each to provide a comprehensive predictive tool that can handle diverse types of educational data, offering accurate and interpretable predictions to support student success.

3. **the generalization of predictive models across different educational contexts:**

Instead of using a single data set to train prediction models, we developed and generalization of predictive models across different educational contexts by allowing educators to upload custom datasets, EduPredi ensures that the insights and predictions are tailored to the specific educational environment, thereby increasing the relevance and applicability of the predictions.

4. **Database Management**

Contrary to initial considerations for using a robust database management system, the project opted against using a formal database architecture. This decision was made to simplify the data handling process, where Django's default file-based system, SQLite, was sufficient for the project's scale, avoiding complexities related to database setup and maintenance.

System Implementation

Implementation Phases

1. **Initial Setup and Configuration:**
    - **Django Configuration:** The entire web application was configured using Django, setting up essential components like URL routing, WSGI/ASGI applications for server responses, and application-specific settings.
    - **Modular Design:** Django's apps were structured to handle different functionalities, facilitating maintenance and scalability.
2. **Data Handling Enhancements:**
    - **Flexible Data Input:** The system was enhanced to allow users to upload and use their datasets for predictions, moving beyond the limitations of a single pre-defined dataset. This feature significantly broadened the applicability and utility of EduPredi.
3. **Advanced Predictive Models:**
    - **Multiple Models Integration:** EduPredi integrates three different algorithms, providing users with the flexibility to choose the model that best fits their data characteristics. This approach replaces the previously planned logistic regression model, offering more comprehensive analysis capabilities.
    - **Dynamic Feature Handling:** The system dynamically adjusts to handle various feature types during the model training and prediction phases, which is critical for accommodating diverse educational datasets.

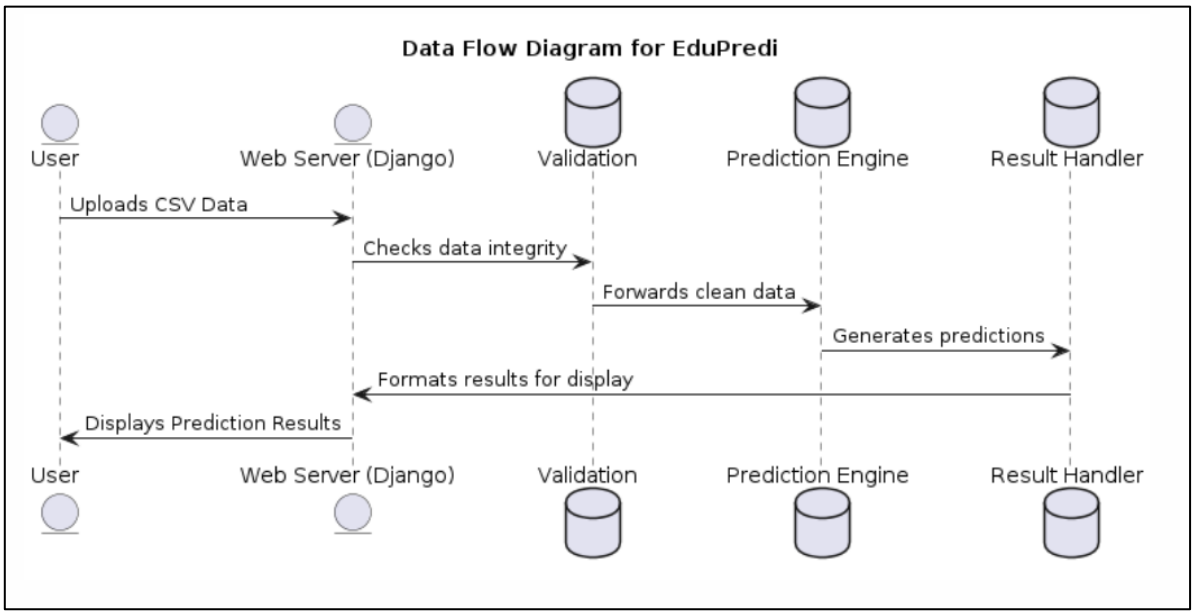
User Interface and Interaction

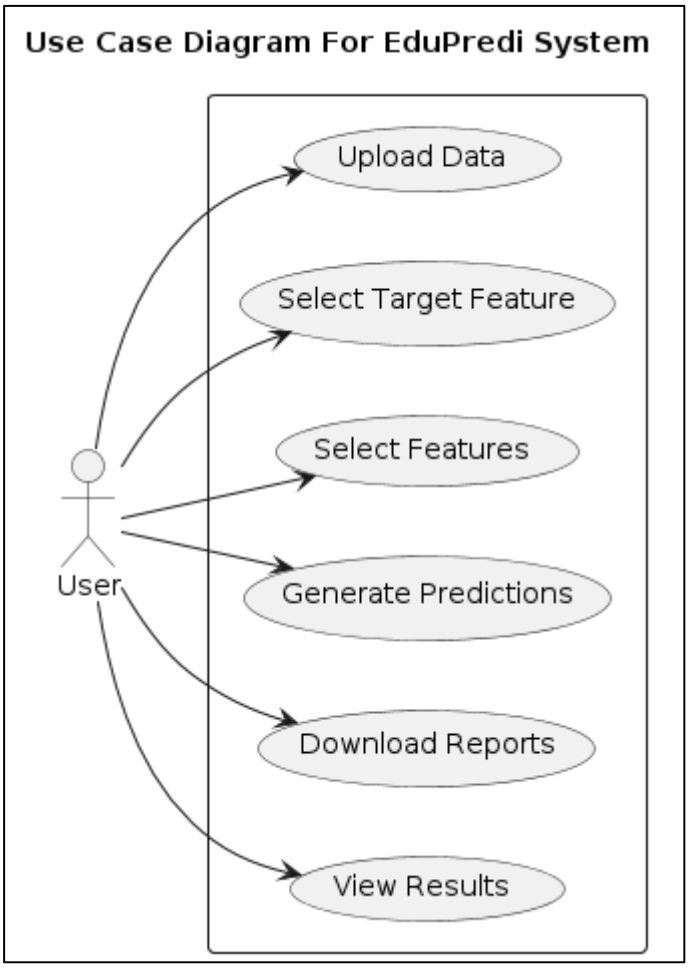
The user interface was developed using Django's templating features, ensuring it was straightforward and accessible to all user types, including students, teachers, and parents. This decision was pivotal in making the system user-friendly and ensuring that users could easily navigate and utilize the platform without specialized technical knowledge.

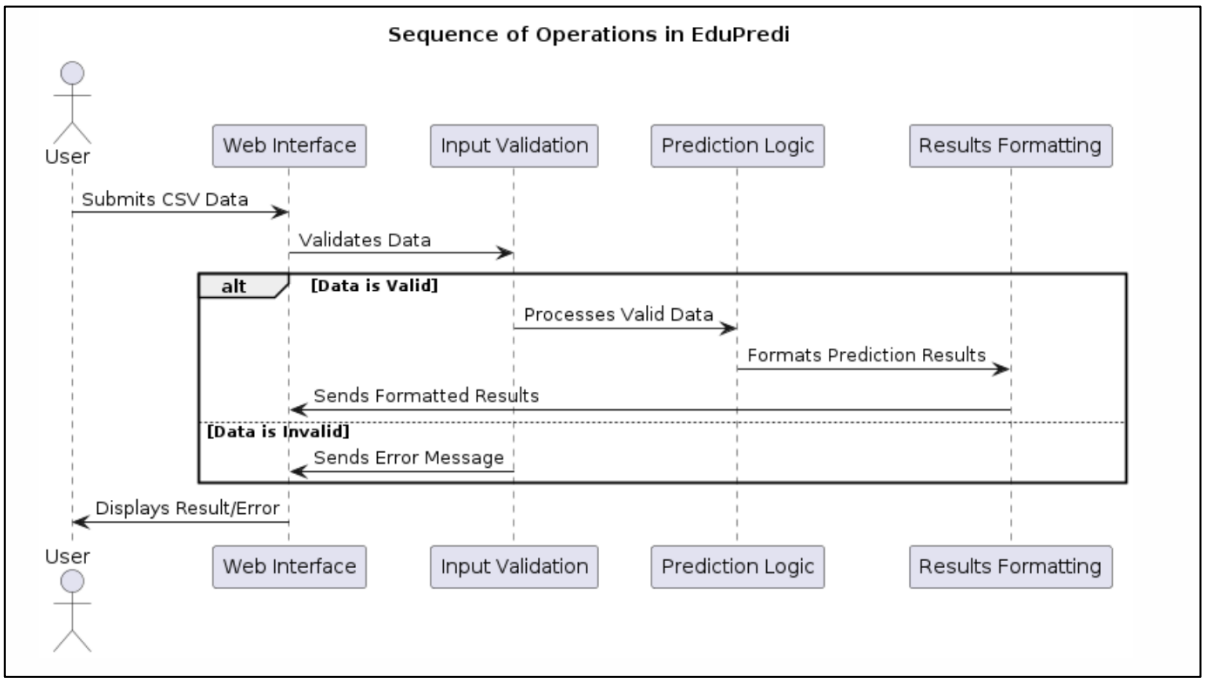
Rationale for Changes

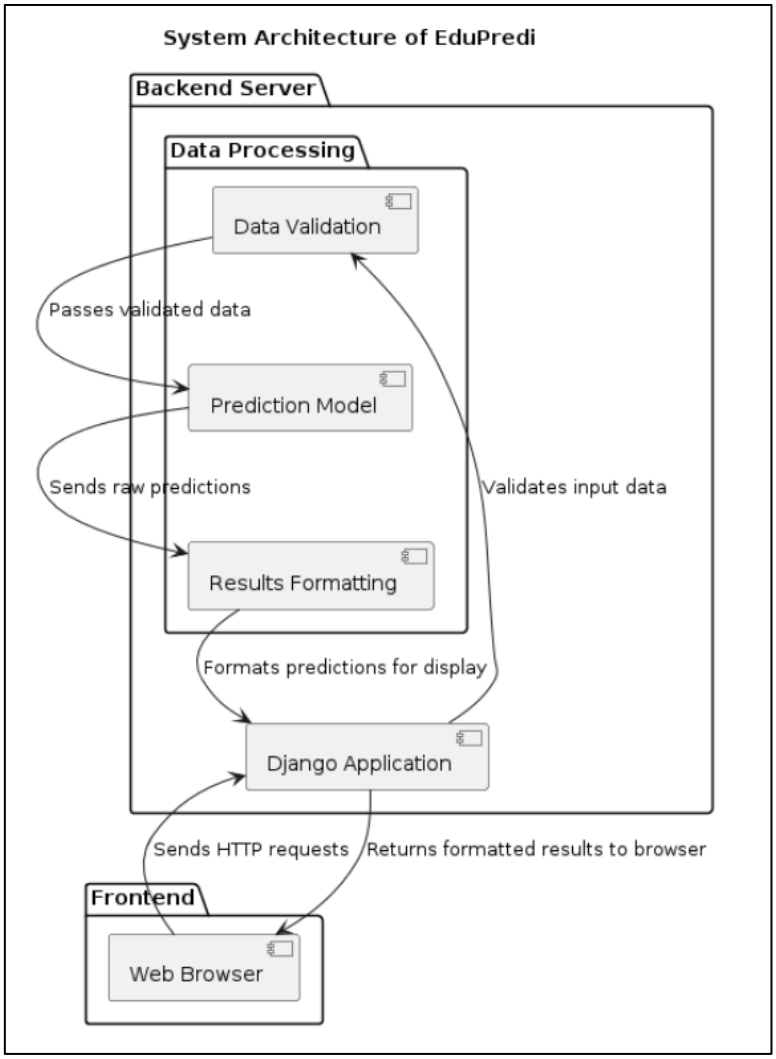
- **Technology Simplification:** The switch to exclusively using Django was driven by the need to streamline development and reduce the complexity associated with managing multiple frameworks.
- **Increased Flexibility and Accuracy:** The implementation of multiple algorithms and the ability to upload diverse datasets were crucial responses to the limitations observed in the initial model, significantly enhancing the system's flexibility and predictive accuracy.
- **User-Centered Design:** Simplifying the user interface and ensuring ease of use were direct outcomes of user feedback, emphasizing the importance of accessibility and practical utility in educational technology.

EduPredi System Diagrams






Sequence of Operations in EduPredi
User
Web Interface
Input Validation
Prediction Logic
Results Formatting
Submits CSV Data
Validates Data
alt
[Data is Valid]
Processes Valid Data
Formats Prediction Results
Sends Formatted Results
[Data is Invalid]
Sends Error Message
Displays Result/Error
User
Web Interface
Input Validation
Prediction Logic
Results Formatting


System Architecture of EduPredi
Backend Server
Data Processing
Data Validation
Passes validated data
Prediction Model
Sends raw predictions
Validates input data
Results Formatting
Formats predictions for display
Django Application
Sends HTTP requests
Returns formatted results to browser
Frontend
Web Browser

Testing and Validation

To ensure the reliability and accuracy of the models used in EduPredi, various testing and validation methods were employed. The dataset was split into training and testing sets with an 80-20 ratio. This approach allows for training the models on a majority of the data while retaining a portion for testing and validating the models' performance.

1. **Decision Tree ID3 Model:**
    - The **data_testing** function:
        - Loads the decision tree model from JSON.
        - Initializes the test dataset.
        - Iterates over the test data, making predictions using the **get_prediction** function.
        - Calculates the accuracy as the percentage of correct predictions.
    - The **process_data** function:
        - Processes the incoming data.
        - Runs the ID3 algorithm to create the decision tree.
        - Converts the tree to JSON and stores it in **settings.TREE**.
        - Calls **data_testing** to calculate the accuracy of the decision tree.
        - Returns the accuracy and the decision tree in the response.

Test Data and Calculate Accuracy for Decision Tree ID3 Model:

```
2.
3.  def data_testing(mylist):
4.      tree = json.loads(settings.TREE)
5.      rows = settings.TEST_DATA
6.      settings.TEST_DATA = {feature:[] for feature in rows[0]}
7.      for lst in rows[1:]:
8.          for i in range(len(rows[0])):
9.              settings.TEST_DATA[str(rows[0][i])].append(lst[i])
10.     true_pre = 0
11.     for i in range(len(settings.TEST_DATA[mylist[-1]])):
12.         line = {col:[] for col in mylist[:-1]}
13.         for feature in mylist[:-1]:
14.             line[feature] = settings.TEST_DATA[feature][i]
15.         if get_prediction(tree,line)==settings.TEST_DATA[mylist[-1]][i]:
16.             true_pre += 1
17.     return (true_pre*100)/len(settings.TEST_DATA[mylist[-1]])
```

2. Decision Tree C4.5 Model:

- **Processing Data and Building the Model**:
  - The **process_data2** function reads and processes the data, builds the C4.5 decision tree, and stores it in **settings.TREE**.
  - After building the tree, it calls the **data_testing** function to evaluate the model's performance on the test data.
- **Calculating Accuracy**:
  - The **data_testing** function loads the test data, makes predictions using the built tree, and compares them with the actual values.
  - The accuracy is calculated as the percentage of correct predictions over the total number of predictions.

- **Calculation Steps**
  - **Read and process the CSV file** to split into training and testing sets (80-20 split).
  - **Build the C4.5 decision tree** using the training data.
  - **Evaluate the model** on the test data:
    - For each test data point, predict the output using the decision tree.
    - Compare the predicted output with the actual output.
    - Calculate the accuracy as the number of correct predictions divided by the total number of predictions, multiplied by 100.

Test Data and Calculate Accuracy for Decision Tree C4.5 Model:

```
2.
3.  def data_testing(mylist):
4.      tree = json.loads(settings.TREE)
5.      rows = settings.TEST_DATA
6.      settings.TEST_DATA = {feature:[] for feature in rows[0]}
7.      for lst in rows[1:]:
8.          for i in range(len(rows[0])):
9.              settings.TEST_DATA[str(rows[0][i])].append(lst[i])
10.     true_pre = 0
11.     for i in range(len(settings.TEST_DATA[mylist[-1]])):
12.         line = {col:[] for col in mylist[:-1]}
13.         for feature in mylist[:-1]:
14.             line[feature] = settings.TEST_DATA[feature][i]
15.         if get_prediction(tree,line)==settings.TEST_DATA[mylist[-1]][i]:
16.             true_pre += 1
17.     return (true_pre*100)/len(settings.TEST_DATA[mylist[-1]])
```

3. Linear Regression Model:

- **Training and Splitting Data**:
  - The dataset is split into training and testing sets using **train_test_split**.
  - The linear regression model is trained on the training set using **model.fit(X_train, y_train)**.
- **Predicting and Calculating Metrics**:
  - Predictions are made on the test set using **model.predict(X_test)**.
  - The Mean Squared Error is calculated using **mean_squared_error(y_test, y_pred)**.
  - The R-squared value is calculated using **r2_score(y_test, y_pred)**.
  - The accuracy is obtained using **model.score(X_test, y_test)**.
- **Returning Results**:
  - The calculated metrics (MSE, R-squared, accuracy) are formatted and included in the **response_data**.

Test Data and Calculate Accuracy for Linear Regression Model:

```
def linear_regression (X,y,columns):

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

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    accuracy = model.score(X_test, y_test)

    all_coef = [float(model.intercept_)]
    all_coef = all_coef + model.coef_.tolist()

    training_output = f"{columns[-1]} = {all_coef[0]}"
    key = 0
    for coef in all_coef[1:]:
        if coef > 0:
            training_output += f" + {coef} * {columns[key]}"
        elif coef < 0:
            training_output += f" - {abs(coef)} * {columns[key]}"
        key += 1

    return training_output,all_coef,mse,r2,accuracy
```

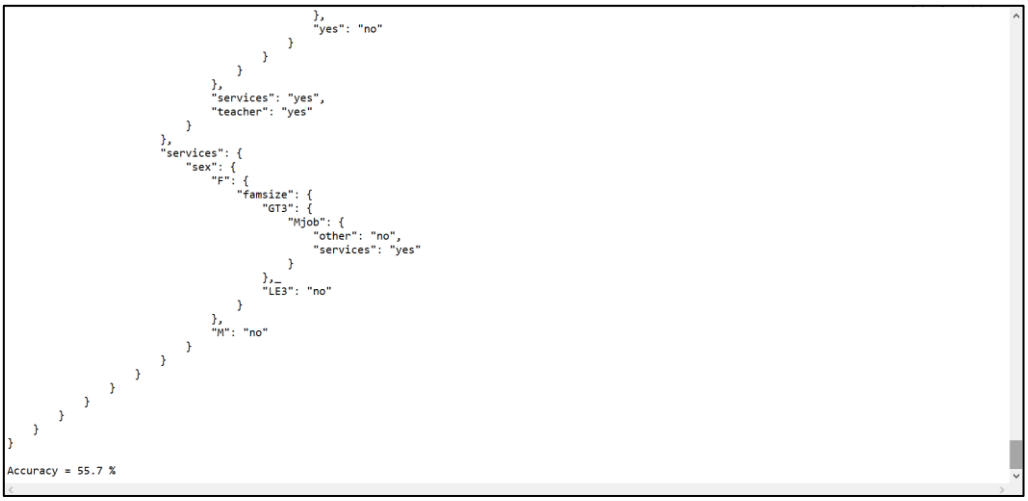
Results

We obtained these results using specific training dataset for about 400 students from a warehouse uci.

(https://archive.ics.uci.edu/ml/datasets/student+performance)

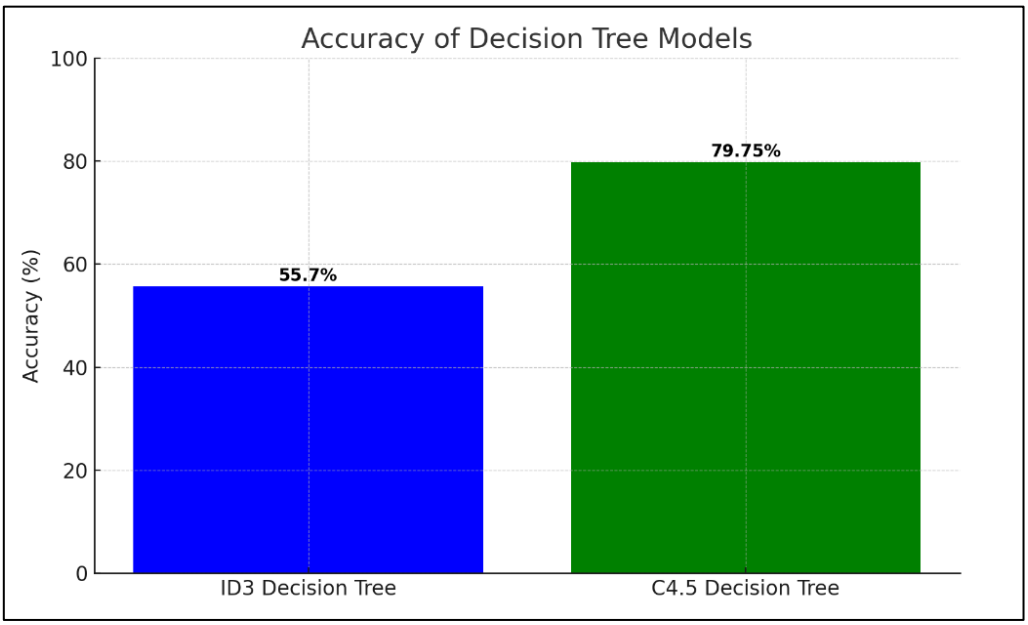
The results represent an approximate result, and the accuracy results may differ slightly when using a different scenario.

1. **Decision Tree (ID3) Results:**
    - The ID3 decision tree achieved an accuracy of 55.7% based on the testing set.



2. **Decision Tree (C4.5) Results:**

- The C4.5 decision tree achieved an accuracy of 79.75% on the testing set.



3. **Linear Regression Results:**

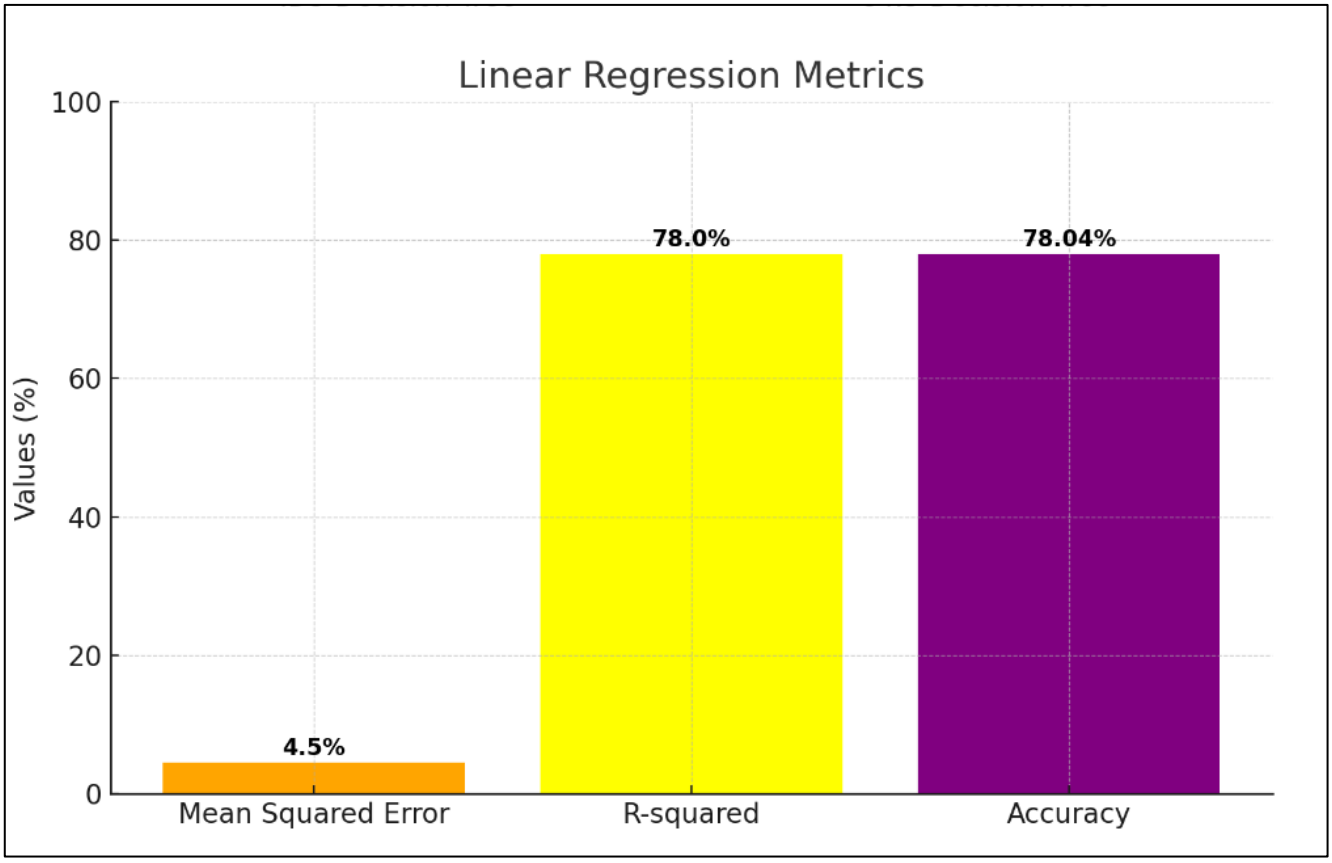
- The linear regression model's performance was evaluated with the following metrics:
  - **Mean Squared Error (MSE):** 4.5%
  - **R-squared:** 0.78
  - **Accuracy:** 78.04%

```
G3 = -0.4684438438758498 - 0.1980109486405556 * age + 0.09419389409743767 * Medu - 0.18833815433656328 *
 Fedu + 0.13100922612513963 * traveltime - 0.06604957943372079 * studytime - 0.4162580489147707 *
failures + 0.334612050486799 * famrel + 0.01062302940703275 * freetime + 0.13763275035221764 *
goout - 0.10501089426559077 * Dalc + 0.06175641322359782 * Walc + 0.05907769619931408 *
 health + 0.044864353410287605 * absences + 0.160747539229821 * G1 + 0.9775341874539358 * G2

Mean Squared Error: 4.5%

R-squared Error: 0.78%

Accuracy: 78.04%
```



Discussion

The exploration of different predictive models to assess student performance has revealed varied accuracies and statistical metrics that highlight the strengths and limitations of each approach. The ID3 and C4.5 decision trees, alongside the linear regression model, provide a comprehensive framework for understanding the dynamics of student academic outcomes.

Model Performance:

- **ID3 Decision Tree:** The ID3 model achieved an accuracy of 55.7%, which is lower compared to the other models. This highlights ID3's efficiency in handling categorical data. Despite its simplicity and reliance on attribute selection based purely on information gain, there is potential for improved performance through more refined tuning and preprocessing. Future enhancements could further optimize its predictive capabilities.
- **C4.5 Decision Tree:** With an accuracy of 79.75%, the C4.5 model significantly outperformed the ID3 model. The improvement can be attributed to the C4.5's ability to handle both discrete and continuous attributes, and its use of gain ratio which normalizes the information gain. This makes it more adept at managing the diverse types of data involved in student performance prediction.
- **Linear Regression:** The linear regression model exhibited an accuracy of 78.04%, with an R-squared value of 0.78, indicating that 78% of the variance in the student's final grade is predictable from the model's inputs. While its accuracy is slightly below that of the C4.5 decision tree, the high R-squared value suggests that it is quite effective in predicting outcomes based on linear relationships between features.

Implications:

- The robust performance of the **C4.5 decision tree** model suggests it as a preferable choice for scenarios where a balance between handling different data types and model interpretability is required.
- The **linear regression model** is particularly useful for predictions where a linear approximation of the relationships is sufficient, making it a valuable tool for quick and reasonably accurate forecasting.
- Despite its lower accuracy, the **ID3 decision tree** still offers valuable insights, particularly in settings where the interpretability of straightforward decision rules is crucial, such as in educational settings where interventions might need to be explained and justified to stakeholders.

Limitations:

- **ID3's** lower performance might limit its applicability in complex scenarios where nuanced distinctions between categories are necessary.
- **C4.5**, while versatile, may still suffer from overfitting, particularly in datasets with many noisy or irrelevant features.
- **Linear regression** assumes linearity in the data it models, which might not always hold true, potentially leading to underfitting in more complex datasets.

Future Work:

- Future studies could explore hybrid models or ensemble techniques that combine the interpretative benefits of decision trees with the predictive accuracy of regression models.
- Advanced preprocessing techniques, including feature selection and transformation, could also be employed to enhance the performance of the ID3 model.
- Investigating the impact of additional variables, such as student engagement or psychological factors, might yield models that capture a broader spectrum of influences on student performance.

This discussion underscores the nuanced capabilities and specific use cases of each model, guiding future strategies for educational data analysis and the development of predictive tools.

Challenges and Solutions

Challenge 1: Data Quality and Preprocessing

One of the main challenges in implementing predictive analytics in education is ensuring data quality and handling diverse data types, from numerical grades to categorical data such as attendance records.

Solution: EduPredi addresses these challenges by incorporating robust data preprocessing methods that enhance model training and prediction outcomes. Techniques such as feature scaling, encoding categorical variables, and handling missing values are crucial for preparing data for effective machine learning applications.

Challenge 2: Model Flexibility and Generalization

Ensuring that predictive models can generalize across different educational contexts is another significant challenge. Each educational institution may have different data structures, making it difficult to develop a one-size-fits-all model.

Solution: EduPredi allows educators to upload custom datasets, ensuring that the insights and predictions are tailored to the specific educational environment. This flexibility increases the relevance and applicability of the predictions, making the system adaptable to various educational settings.

Challenge 3: Selection of Technologies and Frameworks

Initially, there was uncertainty about which technologies and frameworks to use. The project considered using Flask for the backend and React for the frontend to leverage their strengths in quick API development and responsive user interfaces.

Solution: The project ultimately chose Django due to its all-encompassing nature, which simplifies project management by integrating front and back-end functionalities. Django’s built-in features support rapid development and clear structuring of web applications, which was essential for meeting project timelines and requirements.

Challenge 4: Database Management

Managing data efficiently was a challenge, especially when considering the scale and complexity of the datasets involved. Initially, a robust database management system was considered, but it posed a complexity in setup and maintenance.

Solution: The project opted to use Django's default file-based system, SQLite, which was sufficient for the project's scale. This decision simplified the data

handling process and avoided complexities related to database setup and maintenance.

Challenge 5: Enhancing Predictive Capabilities

The initial approach using a single logistic regression model proved to be insufficient for handling different types of data features.

Solution: The project evolved to incorporate three distinct algorithms—Decision Tree ID3, Decision Tree C4.5, and Linear Regression. This diversification allows EduPredi to offer tailored predictive insights based on the specific characteristics of the dataset being analyzed, enhancing the system's flexibility and accuracy.

Challenge 6: User Interface and Interaction

Creating a user-friendly interface that educators, students, and parents can easily navigate without requiring extensive technical knowledge was crucial.

Solution: The user interface was developed using Django's templating features, ensuring it was straightforward and accessible to all user types. This decision was pivotal in making the system user-friendly and ensuring that users could easily navigate and utilize the platform.

Challenge 7: Testing and Validation

Ensuring the reliability and accuracy of the models used in EduPredi required thorough testing and validation.

Solution: The dataset was split into training and testing sets with an 80-20 ratio, allowing for training the models on a majority of the data while retaining a portion for testing and validating the models' performance. Various testing and validation methods were employed to ensure the models' effectiveness.

By addressing these challenges with targeted solutions, EduPredi has developed into a robust and flexible educational predictive analytics system that provides valuable insights into student performance and helps educators tailor their teaching approaches to individual needs.

Future Directions

As the EduPredi system progresses, there are several future directions and enhancements that can be considered to further improve its functionality and impact:

1. **Integration of Advanced Machine Learning Models**:
    - **Deep Learning**: Incorporating neural networks and deep learning techniques could significantly enhance the system's ability to handle complex patterns in educational data. Neural networks, particularly recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, can be beneficial in capturing temporal dependencies in student performance data over time.
    - **Ensemble Methods**: Implementing ensemble techniques, such as random forests and gradient boosting, can improve prediction accuracy by combining the strengths of multiple models.
2. **Enhanced Data Management**:
    - **Database Integration**: Transitioning from a file-based data storage system to a robust database management system (DBMS) such as PostgreSQL or MySQL would facilitate better data management, scalability, and security. This integration would support efficient data retrieval, storage, and management, accommodating larger datasets and more complex queries.
    - **Real-Time Data Processing**: Developing capabilities for real-time data analysis and prediction would allow educators to receive immediate insights and make timely interventions. This could involve implementing streaming data processing frameworks such as Apache Kafka or Apache Flink.

3. **Broader Data Integration**:
    - **Additional Educational Metrics**: Expanding the system to incorporate a wider range of educational metrics, such as attendance records, engagement levels, and socio-economic factors, would provide a more comprehensive analysis of student performance.
    - **External Data Sources**: Developing APIs to integrate with external educational databases and systems could enhance data diversity and enrich predictive analytics.
4. **User Experience and Accessibility**:
    - **Mobile Application Development**: Creating a mobile application for EduPredi would increase accessibility and convenience for users, allowing them to access predictions and insights on-the-go.
    - **User Interface Enhancements**: Continuously improving the user interface to make it more intuitive and user-friendly, incorporating feedback from educators and administrators.
5. **Scalability and Performance**:
    - **Cloud Computing**: Leveraging cloud platforms such as AWS, Google Cloud, or Microsoft Azure for scalable and efficient computing resources. This would support handling large datasets and high computational demands associated with advanced machine learning models.
    - **Batch Predictions**: Implementing batch processing features to allow bulk predictions for multiple student records simultaneously, enhancing the system's efficiency and usability.

By pursuing these future directions, EduPredi can evolve into a more powerful, versatile, and user-friendly tool, significantly contributing to educational data analytics and supporting the goal of improving student outcomes.

Conclusion

The EduPredi project has successfully demonstrated the practical application and benefits of predictive analytics in educational settings. By leveraging advanced machine learning techniques, EduPredi provides educators and administrators with valuable insights into student performance, which can facilitate more informed decision-making and personalized educational strategies.

Key Findings:

- The integration of multiple predictive models (Decision Tree ID3, Decision Tree C4.5, and Linear Regression) has allowed EduPredi to cater to diverse data types and prediction needs, ensuring broad applicability and enhanced accuracy.
- Testing and validation have shown that the Decision Tree C4.5 model and Linear Regression are particularly effective, with accuracies of 79.75% and 78.04% respectively, highlighting their robustness in predicting student performance.
- The system's user-friendly interface has been pivotal in ensuring accessibility and ease of use, making EduPredi a practical tool for a wide range of users without requiring advanced technical skills.

Impact:

- EduPredi has the potential to significantly improve educational outcomes by providing early identification of at-risk students and suggesting tailored interventions.
- The system's flexibility in handling various types of educational data makes it a valuable tool for continuous learning improvement across different educational contexts.

Future Directions:

- **Enhanced Models:** Further development could include the integration of more sophisticated machine learning models, such as neural networks or ensemble methods, to improve prediction accuracy.
- **Expanding the system**: to incorporate additional educational metrics and broader data sources, such as student engagement levels or socio-economic factors, could provide a more comprehensive view of student performance.
- **Batch Predictions:** Implement a feature to upload files with multiple student records for bulk predictions and ensure the system handles various file formats and efficiently processes large datasets.
- **Database Integration:** Use robust databases (e.g., PostgreSQL, MySQL) for efficient data storage and management, and develop APIs for seamless data exchange and synchronization with external educational databases.
- **Real-time Data Analysis::** Implementing real-time data analysis features and developing a mobile application could increase the system's accessibility and real-time functionality.

In conclusion, EduPredi stands as a testament to the capabilities of modern educational technology and sets a promising foundation for further innovations in the field of educational data analytics.

References

1. **Django Documentation**
   - Django Documentation Available at: https://docs.djangoproject.com
   - https://youtu.be/eOVLhM6_6t0?si=S2qHAAcl6WjcvYit
   - https://www.w3schools.com/django/django
2. **Python Libraries**
   - McKinney, W. (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference* (pp. 56-61).
   - Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct), 2825-2830.
   - ChatGPT search https://chatgpt.com/?oai-dm=1
3. **Machine Learning and Data Mining**
   - Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques* (3rd ed.). Morgan Kaufmann.
   - James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning* (Vol. 112). New York: Springer.
4. **Educational Technology and Predictive Analytics**
   - Baker, R. S., & Yacef, K. (2009). The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining*, 1(1), 3-17.
   - Romero, C., & Ventura, S. (2013). Data mining in education. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(1), 12-27.
5. **Other references and resources :**
   - Google: https://www.google.com
   - UCI Machine Learning Repository.Student Performance Data Set: https://archive.ics.uci.edu/ml/datasets/student+performance
   - Datasets https://www.kaggle.com/

Appendices

- **Appendix A: How to run the project**

  To run the EduPredi project, follow these steps:

  1. **download the project:**

     download the project files to your local machine.

  2. **Set Up the Environment:**

     Install Python if not already installed.

     Set up a virtual environment with this code

     `python -m venv env`

     `source env/bin/activate  # On Windows use `env\Scripts\activate``

  3. **Install Django:**

     `Python -m pip install Django`

  4. **Install the required Python library:**

     `Pip install scikit-learn`

  5. **apply migrations:**

     Run the following commands to apply migrations

     `python manage.py makemigrations`

     `python manage.py migrate`

  6. **Run the Development Server:**

     Start the Django development server:

     `python manage.py runserver`

  7. **Access the Application:**

     Open a web browser and navigate to http://127.0.0.1:8000 to access the application.

- **Appendix B: User manual: EduPredi System**

Introduction

Welcome to the EduPredi user manual. EduPredi is a web-based educational tool designed to predict student performance across various metrics using machine learning algorithms. This manual guides you through using the EduPredi system effectively.

System Requirements

- Web Browser: Chrome, Firefox, Safari, or Edge
- Internet Connection: Required for accessing the EduPredi platform

Getting Started

1. To begin using EduPredi, navigate to the website URL provided by the system administrator.

2. Model Selection:

   Choose the appropriate model based on the type of data that you want to predict:

- Decision Tree ID3 for categorical data(It gives a categorical prediction based on categorical data)
- Decision Tree C4.5 for mixed data(It gives a categorical prediction based on numerical or categorical data)
- Linear Regression for numerical data(It gives a numerical prediction based on numerical data)

  Select the model by clicking on the corresponding button.

3. Data Upload:

- Navigate to the 'Upload Data' section.
- Click on 'Choose File' and select the CSV file from your device that you want to upload (students' previous data).
- Press 'Upload' to submit the file. The system will process the file and prepare it for analysis.

4. Choose a feature to predict:
    - After you load the data, you will be directed to select the feature (variable) you want to predict
    - In the selection list you will see only the features that can be selected as a prediction result based on the prediction algorithm you previously chose.
    - Select the feature you want to predict and then Click the "Select Feature".
5. Features Selection:
    - After Choose a feature to predict and Click the "Select Feature", You will be directed to select the features (variables) that you want to influence the prediction model.
    - Check the boxes next to the features you wish to analyze.
    - Click "Get Linear Regression/Get Decision Tree" to confirm your selections.

6. View Model Results:

    After clicking "Get Linear Regression/Get Decision Tree", the system will process the selected features to generate a prediction model based on the chosen algorithm. Once the model has been created:

    - For Decision Trees (ID3 or C4.5), you will be able to view the tree diagram that shows how decisions are made based on the input features.
    - For Linear Regression, the system will display the regression equation and possibly a plot showing the fit of the model to your data.

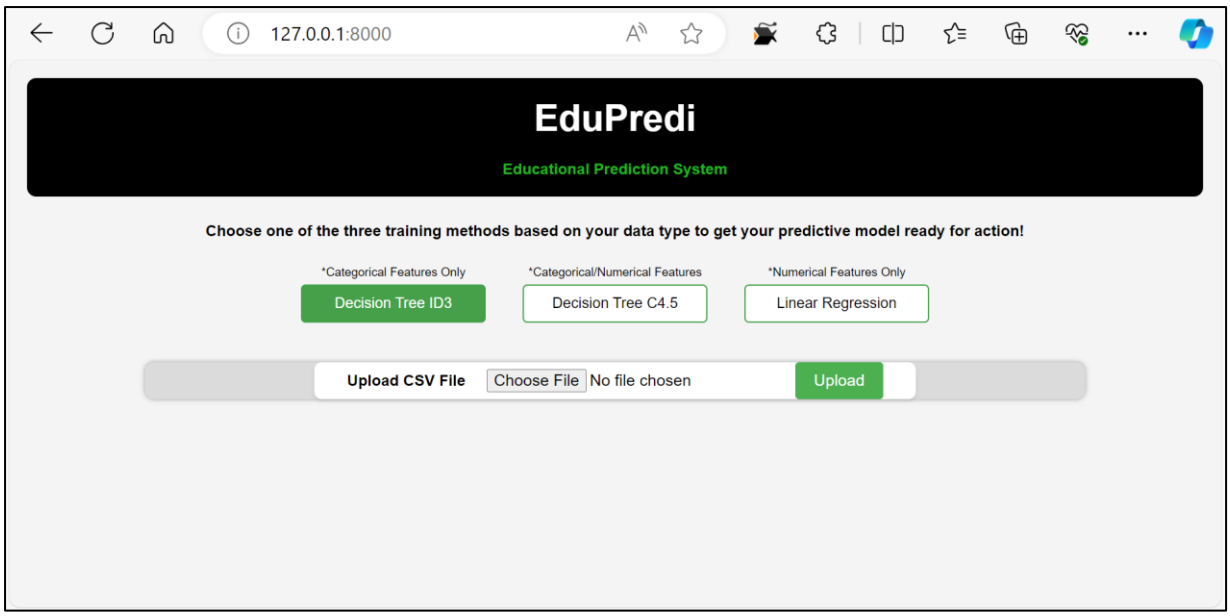
7. Download the Prediction Model:

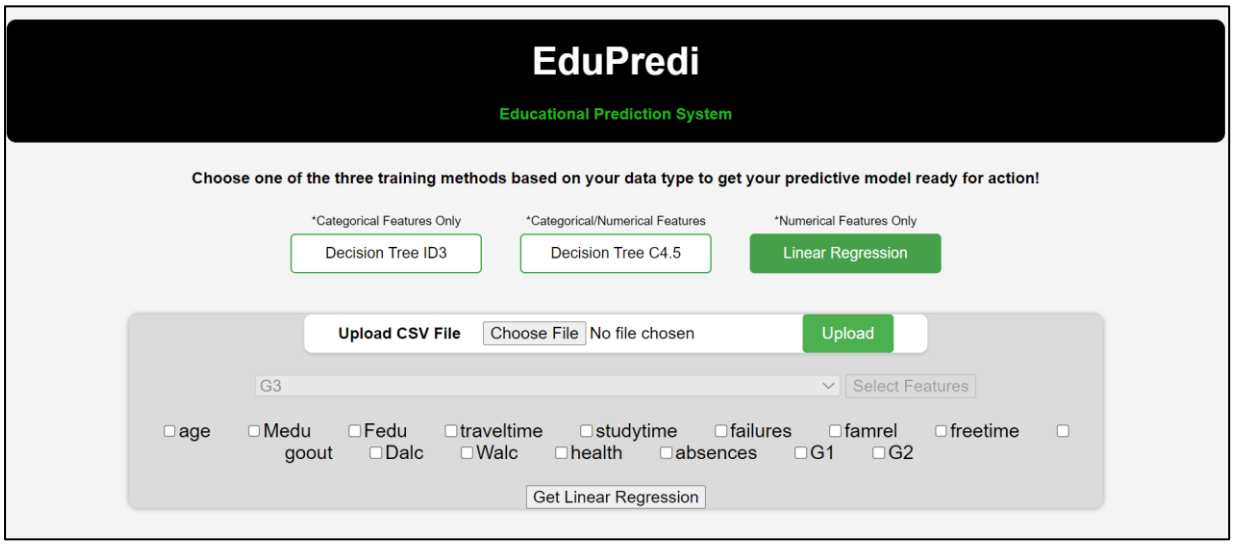
- If you wish to download the results of the model (like the decision tree structure or regression equation), click on the "Download" button.
- This allows you to save the output as a text file or a CSV file on your device for further analysis or record-keeping.

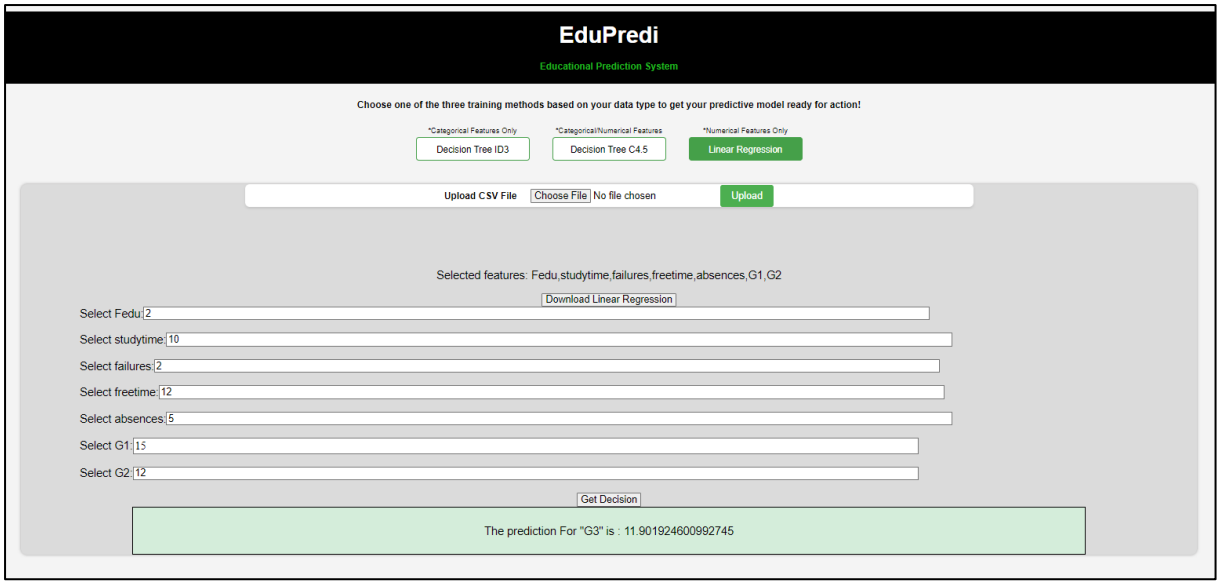
8. Make predictions:

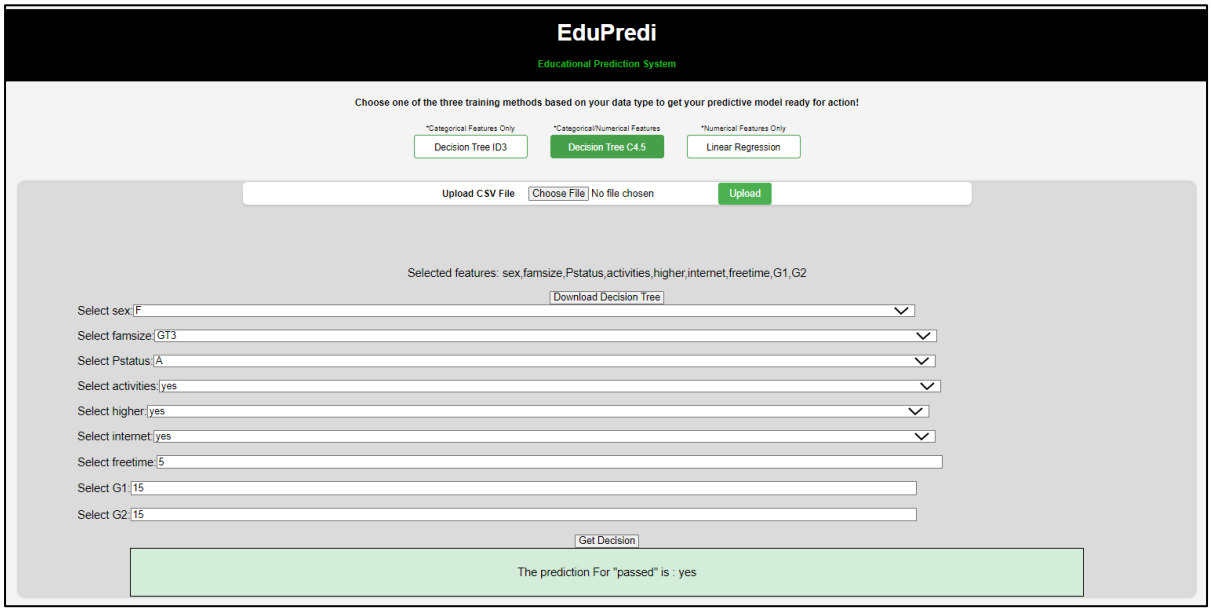
- Now, in this step, the system will display boxes for you to enter the data of the student who you want make prediction for him .
- Fill in all the fields that appear in front of you, then click Get Decision and the prediction result will appear on the screen .

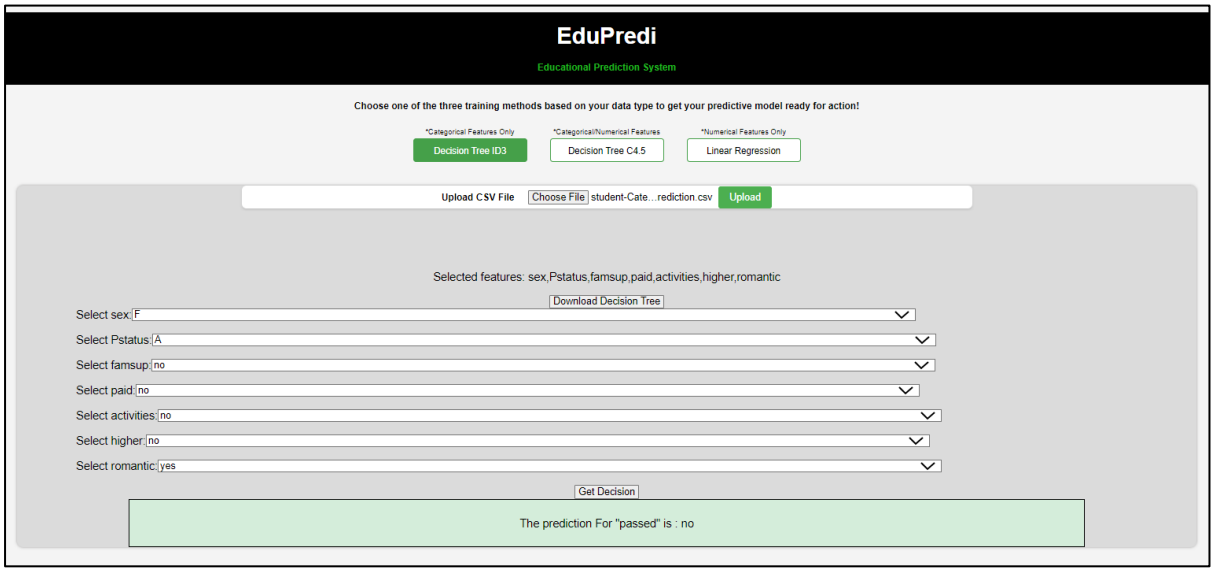
- **Appendix C: Interface of EduPredi system**






EduPredi
Educational Prediction System
Choose one of the three training methods based on your data type to get your predictive model ready for action!
*Categorical Features Only
Decision Tree ID3
*Categorical/Numerical Features
Decision Tree C4.5
*Numerical Features Only
Linear Regression
Upload CSV File
Choose File
No file chosen
Upload
Selected features: Fedu,studytime,failures,freetime,absences,G1,G2
Download Linear Regression
Select Fedu: 2
Select studytime: 10
Select failures: 2
Select freetime: 12
Select absences: 5
Select G1: 15
Select G2: 12
Get Decision
The prediction For "G3" is : 11.901924600992745


EduPredi
Educational Prediction System
Choose one of the three training methods based on your data type to get your predictive model ready for action!
*Categorical Features Only
Decision Tree ID3
*Categorical/Numerical Features
Decision Tree C4.5
*Numerical Features Only
Linear Regression
Upload CSV File
Choose File
No file chosen
Upload
Selected features: sex,famsize,Pstatus,activities,higher,internet,freetime,G1,G2
Download Decision Tree
Select sex: F
Select famsize: GT3
Select Pstatus: A
Select activities: yes
Select higher: yes
Select internet: yes
Select freetime: 5
Select G1: 15
Select G2: 15
Get Decision
The prediction For "passed" is : yes


EduPredi
Educational Prediction System
Choose one of the three training methods based on your data type to get your predictive model ready for action!
*Categorical Features Only
Decision Tree ID3
*Categorical/Numerical Features
Decision Tree C4.5
*Numerical Features Only
Linear Regression
Upload CSV File
Choose File
student-Cate...rediction.csv
Upload
Selected features: sex,Pstatus,famsup,paid,activities,higher,romantic
Download Decision Tree
Select sex: F
Select Pstatus: A
Select famsup: no
Select paid: no
Select activities: no
Select higher: no
Select romantic: yes
Get Decision
The prediction For "passed" is : no

- **Appendix D: source code**

C4.5 Decision Tree:

```
from django.shortcuts import render
import csv
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from django.conf import settings
from io import StringIO
import pandas as pd
import numpy as np
import random

def upload_view2(request):
    if request.method == 'POST' and request.FILES.get('csv_file'):
        csv_file = request.FILES['csv_file']
        if not csv_file.name.endswith('.csv'):
            return render(request, 'upload2.html', {'error': 'File is not a CSV'})

        # Read the CSV file
        try:
            data = []
            csv_data = csv_file.read().decode('utf-8')
            csv_data = csv_data.replace(",",";")
            csv_io = StringIO(csv_data)
            csv_data = csv.reader(csv_io, delimiter=';',quotechar='"')
            for row in csv_data:
                numeric_row = []
                for val in row:
                    try:
                        numeric_val = int(val)
                    except ValueError:
                        try:
                            numeric_val = float(val)
                        except ValueError:
                            numeric_val = val
                    numeric_row.append(numeric_val)
                data.append(numeric_row)

            new_data = data[1:].copy()
            random.shuffle(new_data)

            split_index = int(0.8 * len(new_data))
            # Split the data into 80% and 20%
            train_data = new_data[:split_index]
            test_data = new_data[split_index:]
            settings.DATA = [data[0]] + train_data
            settings.TEST_DATA = [data[0]] + test_data

            filtered_data = {feature:[] for feature in data[0]}
            for lst in data[1:]:
                for i in range(len(data[0])):
                    filtered_data[str(data[0][i])].append(lst[i])
            my_features = []
            for key, val in filtered_data.items():
                if not contains_only_numbers2(val): my_features.append(key)
        except Exception as e:
            return render(request, 'upload2.html', {'error': f'Error reading CSV file: {e}'})
```

```
        return render(request, 'upload2.html', {'features': my_features, 'all_features': data[0]})

    return render(request, 'upload2.html')



@csrf_exempt
def process_data2(request):
    if request.method == 'POST':
        data = json.loads(request.POST.get('toSend'))
        rows = settings.DATA
        settings.DATA = {feature:[] for feature in rows[0]}
        for lst in rows[1:]:
            for i in range(len(rows[0])):
                settings.DATA[str(rows[0][i])].append(lst[i])
        mylist = data['checkedFeatures']
        mylist.extend([data['selectedFeature']])
        newdata = {attr: [] for attr in mylist}
        for key, val in newdata.items():
            newdata[key] = settings.DATA[key]
        settings.DATA = newdata
        newDATASET = {attr: [] for attr in mylist[:-1]}
        for attr in mylist[:-1]:
            if not contains_only_numbers2(set(settings.DATA[attr])):
                newDATASET[attr] = list(set(settings.DATA[attr]))

        data = pd.DataFrame(newdata)

        # Build the C4.5 tree
        features = list(data.columns[:-1])
        c45_tree = c45(data, data, features, mylist[-1])
        c45_tree = json.dumps(c45_tree, indent=4)

        settings.TREE = c45_tree
        accuracy = data_testing(mylist)

        response_data = {
            'status': 'success',
            'message': 'Data processed successfully',
            'tree': c45_tree + '\n\nAccuracy = ' + str(round(accuracy,2)) + ' %',
            'newDATASET': newDATASET,
        }
        return JsonResponse(response_data)
    else:
        return JsonResponse({'error': 'Invalid request method'})

def data_testing(mylist):
    tree = json.loads(settings.TREE)
    rows = settings.TEST_DATA
    settings.TEST_DATA = {feature:[] for feature in rows[0]}
    for lst in rows[1:]:
        for i in range(len(rows[0])):
            settings.TEST_DATA[str(rows[0][i])].append(lst[i])
    true_pre = 0
    for i in range(len(settings.TEST_DATA[mylist[-1]])):
        line = {col:[] for col in mylist[:-1]}
        for feature in mylist[:-1]:
            line[feature] = settings.TEST_DATA[feature][i]
        if get_prediction(tree,line)==settings.TEST_DATA[mylist[-1]][i]:
            true_pre += 1
    return (true_pre*100)/len(settings.TEST_DATA[mylist[-1]])

@csrf_exempt
def predict_data2(request):
    if request.method == 'POST':
        data = json.loads(request.POST.get('toSend'))
```

```
        tree = json.loads(settings.TREE)

        print(tree)
        print(data)

        decision = get_prediction(tree, data)

        response_data = {
            'status': 'success',
            'message': 'Data processed successfully',
            'decision': decision
        }
        return JsonResponse(response_data)
    else:
        return JsonResponse({'error': 'Invalid request method'})

def get_prediction(tree, test_set):
    if isinstance(tree,dict):
        for attribute in tree.keys():
            if attribute in test_set:
                value = test_set[attribute]
                subtree = tree[attribute]
                # If the attribute is numerical, determine the split condition
                if isinstance(subtree, dict):
                    for key in subtree.keys():
                        if "<=" in key and float(value) <= float(key.split('<=')[1]):
                            if isinstance(subtree[key], dict):return get_prediction(subtree[key], test_set)
                            else: return subtree[key]
                        elif ">" in key and float(value) > float(key.split('>')[1]):
                            if isinstance(subtree[key], dict):return get_prediction(subtree[key], test_set)
                            else: return subtree[key]
                # If the attribute is categorical, traverse accordingly
                if value in subtree:
                    result = subtree[value]
                    if isinstance(result, dict):
                        return get_prediction(result, test_set)
                    else:
                        return result
                else:
                    return 'none'
        return 'none'
    else : return tree

def contains_only_numbers2(lst):
    for item in lst:
        if not str(item).isdigit():
            return False
    return True



# Function to calculate the entropy of a dataset
def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    entropy = np.sum([(-counts[i]/np.sum(counts)) * np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
    return entropy

# Function to calculate the information gain of a dataset
def info_gain(data, split_attribute_name, target_name):
    total_entropy = entropy(data[target_name])
    if data[split_attribute_name].dtype.kind in 'bifc':  # if attribute is numeric
        median = data[split_attribute_name].median()
        left_split = data[data[split_attribute_name] <= median]
        right_split = data[data[split_attribute_name] > median]
        weighted_entropy = (len(left_split)/len(data) * entropy(left_split[target_name]) +
                            len(right_split)/len(data) * entropy(right_split[target_name]))
```

```
        information_gain = total_entropy - weighted_entropy
        return information_gain, median
    else:  # if attribute is categorical
        vals, counts = np.unique(data[split_attribute_name], return_counts=True)
        weighted_entropy = np.sum([(counts[i]/np.sum(counts)) * entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[target_name]) for i in range(len(vals))])
        information_gain = total_entropy - weighted_entropy
        return information_gain, None

# Function to split the dataset
def split_data(data, attribute, value):
    if data[attribute].dtype.kind in 'bifc':  # if attribute is numeric
        left_split = data[data[attribute] <= value].reset_index(drop=True)
        right_split = data[data[attribute] > value].reset_index(drop=True)
        return left_split, right_split
    else:  # if attribute is categorical
        return data[data[attribute] == value].reset_index(drop=True)

# Function to build the C4.5 tree
def c45(data, original_data, features, target_attribute_name, parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) == 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return np.unique(original_data[target_attribute_name])[np.argmax(np.unique(original_data[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]
        item_values = [info_gain(data, feature, target_attribute_name) for feature in features]
        best_feature_index = np.argmax([item[0] for item in item_values])
        best_feature, best_split = item_values[best_feature_index]
        best_feature_name = features[best_feature_index]
        tree = {best_feature_name: {}}
        features = [i for i in features if i != best_feature_name]
        if best_split is not None:  # numeric attribute
            left_split, right_split = split_data(data, best_feature_name, best_split)
            subtree_left = c45(left_split, data, features, target_attribute_name, parent_node_class)
            subtree_right = c45(right_split, data, features, target_attribute_name, parent_node_class)
            tree[best_feature_name]['<=' + str(best_split)] = subtree_left
            tree[best_feature_name]['>' + str(best_split)] = subtree_right
        else:  # categorical attribute
            for value in np.unique(data[best_feature_name]):
                sub_data = split_data(data, best_feature_name, value)
                subtree = c45(sub_data, data, features, target_attribute_name, parent_node_class)
                tree[best_feature_name][value] = subtree
        outcomes = [str(tree[best_feature_name][value]) for value in tree[best_feature_name]]
        if len(set(outcomes)) == 1:
            return outcomes[0]
        return tree
```

ID3 Decision Tree:

```
from django.shortcuts import render
import csv
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from django.conf import settings
from io import StringIO
import pandas as pd
import numpy as np
import random

def upload_view(request):
    if request.method == 'POST' and request.FILES.get('csv_file'):
        csv_file = request.FILES['csv_file']
        if not csv_file.name.endswith('.csv'):
            return render(request, 'upload.html', {'error': 'File is not a CSV'})

        try:
            data = []
            csv_data = csv_file.read().decode('utf-8')
            csv_data = csv_data.replace(",",";")
            csv_io = StringIO(csv_data)
            csv_data = csv.reader(csv_io, delimiter=';',quotechar='"')
            for row in csv_data:
                numeric_row = []
                for val in row:
                    try:
                        numeric_val = int(val)
                    except ValueError:
                        try:
                            numeric_val = float(val)
                        except ValueError:
                            numeric_val = val
                    numeric_row.append(numeric_val)
                data.append(numeric_row)

            new_data = data[1:].copy()
            random.shuffle(new_data)

            split_index = int(0.8 * len(new_data))
            # Split the data into 80% and 20%
            train_data = new_data[:split_index]
            test_data = new_data[split_index:]
            settings.DATA = [data[0]] + train_data
            settings.TEST_DATA = [data[0]] + test_data

            filtered_data = {feature:[] for feature in data[0]}
            for lst in data[1:]:
                for i in range(len(data[0])):
                    filtered_data[str(data[0][i])].append(lst[i])
            my_features = []
            for key, val in filtered_data.items():
                if not contains_only_numbers(val): my_features.append(key)
        except Exception as e:
            return render(request, 'upload.html', {'error': f'Error reading CSV file: {e}'})

        return render(request, 'upload.html', {'features': my_features})

    return render(request, 'upload.html')
@csrf_exempt
```

```
def process_data(request):
    if request.method == 'POST':
        data = json.loads(request.POST.get('toSend'))
        rows = settings.DATA
        settings.DATA = {feature:[] for feature in rows[0]}
        for lst in rows[1:]:
            for i in range(len(rows[0])):
                settings.DATA[str(rows[0][i])].append(lst[i])
        mylist = data['checkedFeatures']
        mylist.extend([data['selectedFeature']])
        newdata = {attr: [] for attr in mylist}
        for key, val in newdata.items():
            newdata[key] = settings.DATA[key]
        settings.DATA = newdata
        newDATASET = {attr: [] for attr in mylist[:-1]}
        for attr in mylist[:-1]:
            if not contains_only_numbers(set(settings.DATA[attr])):
                newDATASET[attr] = list(set(settings.DATA[attr]))
        df = pd.DataFrame(newdata)
        # Running the ID3 algorithm
        tree = id3(df, df, df.columns[:-1],mylist[-1])
        tree = json.dumps(tree, indent=4)

        settings.TREE = tree
        accuracy = data_testing(mylist)
        response_data = {
            'status': 'success',
            'message': 'Data processed successfully',
            'tree': tree + '\n\nAccuracy = ' + str(round(accuracy,2)) + ' %',
            'newDATASET': newDATASET,
        }
        return JsonResponse(response_data)
    else:
        return JsonResponse({'error': 'Invalid request method'})

def data_testing(mylist):
    tree = json.loads(settings.TREE)
    rows = settings.TEST_DATA
    settings.TEST_DATA = {feature:[] for feature in rows[0]}
    for lst in rows[1:]:
        for i in range(len(rows[0])):
            settings.TEST_DATA[str(rows[0][i])].append(lst[i])
    true_pre = 0
    for i in range(len(settings.TEST_DATA[mylist[-1]])):
        line = {col:[] for col in mylist[:-1]}
        for feature in mylist[:-1]:
            line[feature] = settings.TEST_DATA[feature][i]
        if get_prediction(tree,line)==settings.TEST_DATA[mylist[-1]][i]:
            true_pre += 1
    return (true_pre*100)/len(settings.TEST_DATA[mylist[-1]])
@csrf_exempt
def predict_data(request):
    if request.method == 'POST':
        data = json.loads(request.POST.get('toSend'))
        tree = json.loads(settings.TREE)
        decision = get_prediction(tree, data)

        response_data = {
            'status': 'success',
            'message': 'Data processed successfully',
            'decision': decision
        }
        return JsonResponse(response_data)
    else:
        return JsonResponse({'error': 'Invalid request method'})
```

```
def get_prediction(tree, test_data):
    for key in tree.keys():
        try:
            tree = tree[key][test_data[key]]
            if isinstance(tree, dict):
                tree = get_prediction(tree, test_data)
        except:
            tree = "no"
    return tree

def contains_only_numbers(lst):
    for item in lst:
        if not str(item).isdigit():
            return False
    return True

def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    entropy = np.sum([(-counts[i]/np.sum(counts)) * np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
    return entropy

# Function to calculate information gain
def info_gain(data, split_attribute_name, target_name):
    total_entropy = entropy(data[target_name])

    vals, counts = np.unique(data[split_attribute_name], return_counts=True)

    weighted_entropy = np.sum([(counts[i]/np.sum(counts)) * entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[target_name]) for i in range(len(vals))])

    information_gain = total_entropy - weighted_entropy
    return information_gain

# ID3 Algorithm
def id3(data, original_data, features, target_attribute_name, parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    elif len(data) == 0:
        return np.unique(original_data[target_attribute_name])[np.argmax(np.unique(original_data[target_attribute_name], return_counts=True)[1])]

    elif len(features) == 0:
        return parent_node_class

    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]

        item_values = [info_gain(data, feature, target_attribute_name) for feature in features]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]

        tree = {best_feature: {}}

        features = [i for i in features if i != best_feature]

        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()

            subtree = id3(sub_data, data, features, target_attribute_name, parent_node_class)

            tree[best_feature][value] = subtree

        outcomes = [str(tree[best_feature][value]) for value in tree[best_feature]]
        if len(set(outcomes)) == 1:
            return outcomes[0]
        return tree
```

linear regression:

```
from django.shortcuts import render
import csv
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from django.conf import settings
from io import StringIO
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Create your views here.
def upload_view3(request):
    if request.method == 'POST' and request.FILES.get('csv_file'):
        csv_file = request.FILES['csv_file']
        if not csv_file.name.endswith('.csv'):
            return render(request, 'upload3.html', {'error': 'File is not a CSV'})

        # Read the CSV file
        try:
            data = []
            csv_data = csv_file.read().decode('utf-8')
            csv_data = csv_data.replace(",",";")
            csv_io = StringIO(csv_data)
            csv_data = csv.reader(csv_io, delimiter=';',quotechar='"')
            for row in csv_data:
                numeric_row = []
                for val in row:
                    try:
                        numeric_val = int(val)
                    except ValueError:
                        try:
                            numeric_val = float(val)
                        except ValueError:
                            numeric_val = val
                    numeric_row.append(numeric_val)
                data.append(numeric_row)
            settings.DATA = data
            filtered_data = {feature:[] for feature in data[0]}
            for lst in data[1:]:
                for i in range(len(data[0])):
                    filtered_data[str(data[0][i])].append(lst[i])
            my_features = []
            for key, val in filtered_data.items():
                if contains_only_numbers(val): my_features.append(key)
        except Exception as e:
            return render(request, 'upload3.html', {'error': f'Error reading CSV file: {e}'})

        return render(request, 'upload3.html', {'features': my_features})

    return render(request, 'upload3.html')



@csrf_exempt
def process_data3(request):
    if request.method == 'POST':
        data = json.loads(request.POST.get('toSend'))
        rows = settings.DATA
        settings.DATA = {feature:[] for feature in rows[0]}
        for lst in rows[1:]:
```

```
            for i in range(len(rows[0])):
                settings.DATA[str(rows[0][i])].append(lst[i])
        mylist = data['checkedFeatures']
        mylist.extend([data['selectedFeature']])
        newdata = {attr: [] for attr in mylist}
        for key, val in newdata.items():
            newdata[key] = settings.DATA[key]
        settings.DATA = newdata
        y = newdata[data['selectedFeature']]
        del newdata[data['selectedFeature']]
        newDATASET = {attr: [] for attr in mylist[:-1]}

        mydata = []
        for i in range(len(newdata[mylist[0]])):
            lst = []
            for key, val in newdata.items():
                lst.append(val[i])
            mydata.append(lst)

        tree, settings.TREE, mse, r2, accuracy = linear_regression(mydata, y, mylist)

        response_data = {
            'status': 'success',
            'message': 'Data processed successfully',
            'tree': tree + '\n\nMean Squared Error: ' + str(round(mse,2)) + '%' + '\n\nR-squared Error: ' + str(round(r2,2)) + '%' + '\n\nAccuracy: ' + str(round(accuracy * 100, 2)) +'%',
            'newDATASET': newDATASET,
        }
        return JsonResponse(response_data)
    else:
        # If not a POST request, return an error response
        return JsonResponse({'error': 'Invalid request method'})

@csrf_exempt
def predict_data3(request):
    if request.method == 'POST':
        data = json.loads(request.POST.get('toSend'))
        tree = settings.TREE

        decision = get_prediction(tree, data)

        response_data = {
            'status': 'success',
            'message': 'Data processed successfully',
            'decision': decision
        }
        return JsonResponse(response_data)
    else:
        return JsonResponse({'error': 'Invalid request method'})

def get_prediction(all_coef, test_data):

    y = all_coef[0]
    if len(all_coef)==2: y+=all_coef[1]*test_data[0]
    else:
        for key in range(len(test_data)): y+=float(all_coef[key+1])*float(test_data[key])
    return y

def contains_only_numbers(lst):
    for item in lst:
        if not str(item).isdigit():
            return False
    return True

def linear_regression (X,y,columns):

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

```
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = model.score(X_test, y_test)



all_coef = [float(model.intercept_)]
all_coef = all_coef + model.coef_.tolist()

training_output = f"{columns[-1]} = {all_coef[0]}"
key = 0
for coef in all_coef[1:]:
    if coef > 0:
        training_output += f" + {coef} * {columns[key]}"
    elif coef < 0:
        training_output += f" - {abs(coef)} * {columns[key]}"
    key += 1

return training_output,all_coef,mse,r2,accuracy
```

The End Of Dacument

Tanke You