Sheffield Hallam University
Department of Computing
MSc Big Data Analytics
Programming Concepts and Practice

Machine Learning-based Feature Prediction

Implementation Report
Rahul Bakhtiani

Demo Video: https://drive.google.com/file/d/1FHnRWXWQz9TCOjmbicNBmEt0WVXyxMm8/view?usp=drive_link

Table of Contents

1. Introduction

1.1 Brief overview of the project and its objectives

The project aims to develop a comprehensive data analysis and machine learning solution using object-oriented programming (OOP) concepts. The objective is to preprocess, explore, and analyse the provided dataset, incorporating relevant descriptive statistics and exploratory data analysis (EDA) techniques. This includes assessing feature frequencies, dependencies, and class distributions through visualizations. Additionally, addressing class imbalances using appropriate techniques is crucial. Furthermore, the dataset will be split into training and test sets to train and evaluate three classification models for predicting income, marital status, and work-class. Evaluation metrics such as confusion matrices, precision, recall, and accuracy will be utilized to compare model performance and interpret results, providing insights into the effectiveness of different machine learning algorithms.

2. Problem Analysis

2.1 Problem Analysis

Our project focuses on understanding the lives of real people through a big bunch of data called 'people.data'. We're digging into details like how old people are, what they do for work, and how educated they are. We're especially interested in how these things connect to important stuff like how much money they make, if they're married, and what kind of jobs they have.

By carefully looking at all this information, we hope to find out interesting things, like why some people earn more than others or why some folks have different jobs. We're using smart computer tools to help us predict what might happen in the future, like who might earn more or what job they might have. Our big goal is to help decision-makers make better choices by giving them helpful information about how people's lives are changing and what might happen next.

2.2 Solution Requirements

The solution we're aiming for needs to cover a few important areas. First, it has to be able to handle all the data in 'people.data' efficiently, meaning it should be able to read it, clean it up if needed, and organize it in a way that makes sense.

Next, it should be able to do some smart analysis on the data, like figuring out how different things are connected and finding patterns or trends. Then, it needs to be able to predict things based on the data, like how much money someone might make or what kind of job they might have. But it's not just about crunching numbers – it's about making sense of real people's lives and giving useful insights.

Finally, it should be able to present all this information in a clear and easy-to-understand way, so that decision-makers can use it to make better choices. Overall, the solution needs to be smart, efficient, and user-friendly, helping us unlock the secrets hidden in the data and make informed decisions about people's lives.

3. Implementation Overview

3.1 Summary of implementation decisions

In our implementation strategy, we made deliberate choices to ensure efficiency and practicality. We decided to leverage well-established Python libraries and functions for data preprocessing tasks, streamlining the handling of 'people.data' by addressing issues like missing values and categorical encoding. This decision was driven by our need for reliability and ease of use in managing the dataset's complexities.

Additionally, we prioritized the incorporation of descriptive statistical analysis techniques to extract meaningful insights from the dataset. Moreover, we chose to harness machine learning algorithms for predictive modelling to forecast future trends and outcomes based on historical data patterns. Through the application of these algorithms, our aim is to equip decision-makers with actionable insights, enabling them to navigate socio-economic complexities confidently. Lastly, we opted to include a graphical user interface (GUI) to provide an intuitive and interactive experience for users, making it as easy as possible for them to interact with our solution.

Overall, these decisions aim to make our solution both technically robust and user-friendly, allowing us to unlock the insights hidden within the data while ensuring accessibility for all users.

3.2 Justifications for implementation decisions

In formulating our implementation strategy, four integral facets were considered: data preprocessing, exploratory data analysis (EDA), model training, and the integration of a Graphical User Interface (GUI). Leveraging the capabilities of Python libraries like pandas and scikit-learn, we ensure the reliability and efficiency of our data preprocessing procedures, analogous to tidying up a room before embarking on a journey. Through exploratory data analysis, facilitated by functions such as ‘show_feature_class_distribution’, ‘box_plot’ and ‘heatmap’, we delve into the dataset's intricacies, uncovering patterns and relationships between variables, akin to scrutinizing a map to understand the lay of the land. Subsequently, model training, executed via algorithms like ‘RandomForestClassifier’, ‘LogisticRegression’, ‘KNeighborsClassifier’ and ‘DecisionTreeClassifier’, equips us with predictive capabilities, much like a compass guiding us through uncharted territories.

Finally, the Graphical User Interface (GUI) acts as a user-friendly portal, enabling seamless interaction with our analytical tools and bridging the gap between complex algorithms and user accessibility, akin to a friendly guide facilitating exploration of a vast landscape. Together, these components constitute the foundation of our solution, balancing technical sophistication with intuitive design to unlock actionable insights from the data.

3.3 Overview of programming processes explored

The programming processes, some important steps to handle data smartly were explored. We started with getting the data ready, just like laying a solid foundation before building a house. We made sure the data was clean by fixing any missing bits and organizing it neatly. Then, we went on a hunt through the data, like exploring a hidden treasure map. We used simple tricks to spot patterns and connections hiding in the numbers. Next, we trained our computer to learn from the data and make predictions, a bit like guessing tomorrow's weather based on yesterday's. All along, we kept things simple and user-friendly, making sure anyone could understand and use our tools. It's like making a map that's easy for everyone to read and follow, guiding them through the world of data with ease and confidence.

4. Solution

4.1 Explanation of program structure and major components/modules

The program consists of two major components/modules:

1. Exploratory Data Analysis (EDA) Module (eda_module.py):
    - Purpose: Handles data loading, pre-processing and EDA.
    - Functions:
        - loadData(): Loads data from a file into a pandas dataframe.
        - show_feature_class_distribution(): Provides a bar plot on raw data of a feature input by user.
        - clean_data(): Removes null data and provides cleaned data for EDA
        - box_plot(): Provides a box plot on cleaned data of a features input by user.
        - heatmap(): Provides a heatmap on cleaned data of a features input by user.
        - class_distribution_features_needed(): Provides a bar plot on cleaned data of a feature input by user.
        - corr_heatmap(): Provides correlation matrix of all features on encoded data.
    - Role: Loads the dataset and provides EDA functions.
2. Training Module (training_module.py):
    - Purpose: Trains models for specific features.
    - Functions:
        - encode_variables(): Encodes categorical data using LabelEncoder() to train models.
        - split_data(): Splits encoded dataset in training and testing data
        - train_and_evaluate_models():
            - Balances categorical features models are trained on.
            - Calls split_data() for training.
            - Uses StandardScaler() to scale features before training.
            - Trains models and shows results.
    - Role: Provides a modular and reusable set of functions to train models.
3. Graphical User Interface (GUI) (gui_module.py):
    - Purpose: Provides an interactive interface for users to perform actions and view results.
    - Functions:
        - create_widgets(): Creates GUI with different frames and buttons to access functions of eda and training modules.
        - button functions():
            - Executes the selected function based on user choice.
            - Displays results using text box and graph/plots using pop-ups.
            - Provides results of training models.
    - Role: Bridges the gap between users and the eda and training modules, providing an intuitive way to interact with the data and view results.
4. And finally Main program (PCPAssignment2.ipynb) brings all these components work together to offer users a seamless experience in exploring and analysing the dataset.

4.2 Results

Model	Precision	Recall	Accuracy	F1-Score
K-Nearest Neighbors	74.9669	74.9079	74.9079	74.6224
Random Forest	85.4686	85.5229	85.5229	85.4693
Decision Tree	72.5066	72.5425	72.5425	72.5076

Table 1: Classification of 'WorkClass' feature

Model	Precision	Recall	Accuracy	F1-Score
K-Nearest Neighbors	80.4055	80.6307	80.6307	80.2396
Random Forest	88.948	89.0153	89.0153	88.9432
Decision Tree	81.192	81.3011	81.3011	81.2303

Table 2: Classification of 'Marital Status' feature

Model	Precision	Recall	Accuracy	F1-Score
K-Nearest Neighbors	84.4894	84.4847	84.4847	84.4805
Random Forest	88.7329	88.7332	88.7332	88.733
Logistic Regression	77.138	77.1353	77.1353	77.1363

Table 3: Classification of 'Income' feature

The implementation of machine learning models for the classification tasks yielded valuable insights. 'Random Forest' consistently outperformed 'K-Nearest Neighbors' and 'Decision Tree'/ 'Logistic Regression' (income feature) across all features, demonstrating its effectiveness in predicting 'workclass', 'marital_status', and 'income' features.

5. Instructions for Execution

5.1 Step-by-step guide on how to run the application

- Download the eda_module.py, training_module.py, gui_module.py and PCPAssignment2.ipynb files and save them in a folder alongside the dataset.
- Open anaconda, jupyter notebook and navigate to the folder containing the files.
- Open PCPAssignment2.ipynb file and run each cell in sequence.
- Once GUI window is open click buttons in the window to perform actions. Follow any prompts for input.
- Action results will appear in output window in bottom half of the GUI.
- The graphs and plots will appear in a pop-up window once necessary inputs are provided.
- Close the window or click "Quit" to exit.

6. Reflection

6.1 Reflection on lessons learned and professional development

This project has been instrumental in enhancing my technical proficiency and professional development within the realm of data analysis. One of the key takeaways has been the refinement of my problem-solving skills, particularly in navigating through the intricacies of data preprocessing and exploratory data analysis.

I've learned to leverage various Python libraries and techniques to address challenges such as missing data imputation, categorical encoding, and feature engineering, thereby enhancing the quality and interpretability of the dataset. Additionally, I've gained a deeper understanding of machine learning algorithms and model evaluation metrics through the process of training predictive models for classification tasks. This hands-on experience has enabled me to make informed decisions regarding algorithm selection, hyperparameter tuning, and model evaluation, contributing to my proficiency in machine learning techniques.

Moving forward, I aim to continue exploring advanced topics in data analysis and machine learning, further enriching my technical skill set and professional expertise.

6.2 Evaluation of what went well and what could be improved

What Went Well:

The technical implementation of the application was successful, covering key functionalities such as data pre-processing, EDA, model training and testing, and GUI interaction. Challenges encountered during development were effectively addressed through problem-solving strategies, enhancing understanding of Python programming, EDA methods, model training algorithms, OOPs concepts and GUI development.

Areas for Improvement:

Code structure and modularity could be enhanced for better organization and maintainability, ensuring scalability as the application grows. More graphs, charts could be added to help in EDA further. Implementing robust error handling mechanisms would improve the application's reliability and user-friendliness. Enhancing the user experience through improvements in GUI design and navigation would increase usability and satisfaction.

7. Conclusion

7.1 Final thoughts

In wrapping up, this project has been a fantastic learning adventure, filled with valuable lessons and moments of growth in the world of data analysis. From tidying up messy data to uncovering hidden insights through exploration and training predictive models, each step felt like solving a. Creating a user-friendly interface was like crafting a map that guides people through the data jungle with ease.

Overall, this journey has not only sharpened my skills but also deepened my love for making sense of data. As I move forward, I'm excited to apply what I've learned to new challenges and continue exploring the fascinating world of data science.

Appendix A: Screenshots

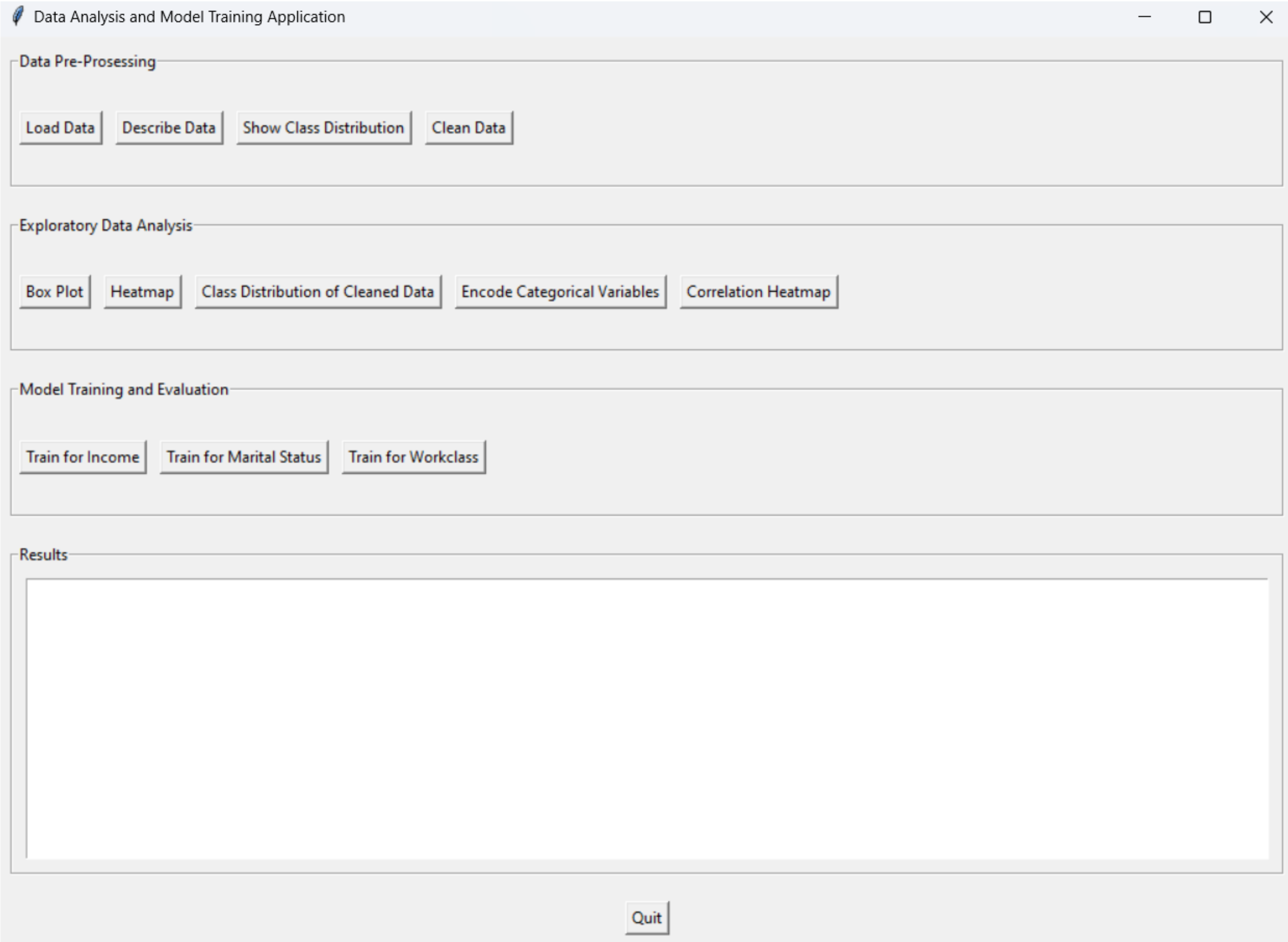


Figure 1 – GUI

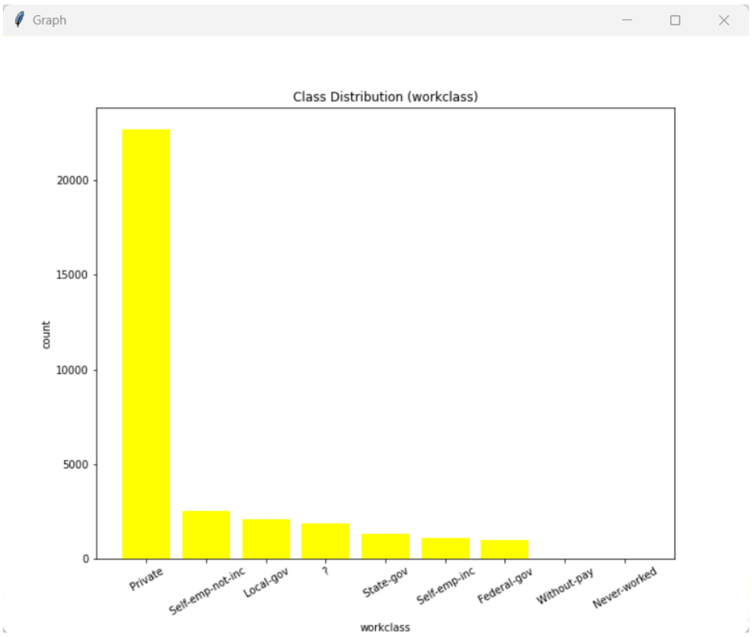


Figure 2 – Pop-up Window for Graphs/ Plots

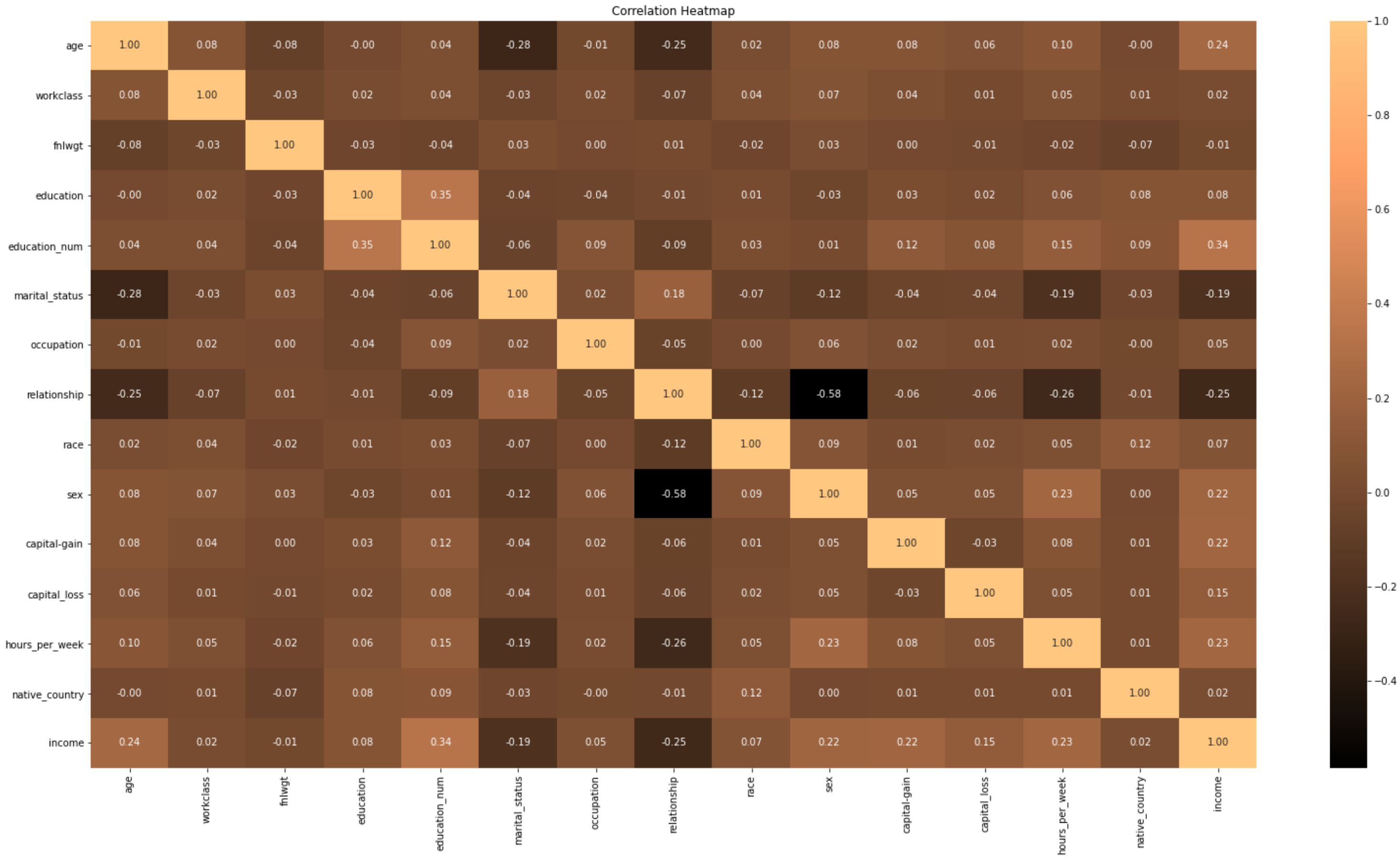
Figure 3 – Descriptive Statistics of 'people.data'

Results

```
Descriptive Statistics for Numerical Variables:
            age       fnlwgt  education_num  capital-gain  capital_loss  hours_per_week
count  32561.00     32561.00       32561.00      32561.00      32561.00        32561.00
mean      38.58    189778.37          10.08       1077.65         87.30           40.44
std       13.64    105549.98           2.57       7385.29        402.96           12.35
min       17.00     12285.00           1.00          0.00          0.00            1.00
25%       28.00    117827.00           9.00          0.00          0.00           40.00
50%       37.00    178356.00          10.00          0.00          0.00           40.00
75%       48.00    237051.00          12.00          0.00          0.00           45.00
max       90.00   1484705.00          16.00      99999.00       4356.00           99.00

Descriptive Statistics for Categorical Variables:
       workclass education      marital_status      occupation relationship   race    sex native_country income
count      32561     32561               32561           32561        32561  32561  32561          32561  32561
unique         9        16                   7              15            6      5      2             42      2
top      Private   HS-grad  Married-civ-spouse  Prof-specialty      Husband  White   Male  United-States  <=50K
freq       22696     10501               14976            4140        13193  27816  21790          29170  24720
```

Figure 4 – Correlation Heatmap



```
Model: K-Nearest Neighbors
Confusion Matrix:
[[3944  181   98  115   76  171    4]
 [ 459 2815  257  203  215  386    8]
 [ 306  410 2865  286  297  245   18]
 [ 170  195  155 3645  208   98    2]
 [ 287  414  373  576 2604  200   27]
 [ 398  423  202  205  153 3100    8]
 [   0    0    0    0    0    0 4399]]
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.86      0.78      4589
           1       0.63      0.65      0.64      4343
           2       0.73      0.65      0.68      4427
           3       0.72      0.81      0.77      4473
           4       0.73      0.58      0.65      4481
           5       0.74      0.69      0.71      4489
           6       0.98      1.00      0.99      4399

    accuracy                           0.75     31201
   macro avg       0.75      0.75      0.75     31201
weighted avg       0.75      0.75      0.75     31201


Model: Random Forest
Confusion Matrix:
[[4168   80  118   52   71   99    1]
 [ 158 3313  267  141  198  260    6]
 [ 107  178 3624  131  249  131    7]
 [  32   70  120 3988  204   56    3]
 [  79  217  316  276 3442  141   10]
 [ 117  212  171   82  126 3772    9]
 [   0    0    0    0    1    0 4398]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.91      0.90      4589
           1       0.81      0.76      0.79      4343
           2       0.79      0.82      0.80      4427
           3       0.85      0.89      0.87      4473
           4       0.80      0.77      0.78      4481
           5       0.85      0.84      0.84      4489
           6       0.99      1.00      1.00      4399

    accuracy                           0.86     31201
   macro avg       0.86      0.86      0.86     31201
weighted avg       0.86      0.86      0.86     31201


Model: Decision Tree
Confusion Matrix:
[[3618  252  203   99  163  242   12]
 [ 277 2763  303  200  358  437    5]
 [ 249  407 2721  284  437  308   21]
 [ 145  175  218 3337  439  144   15]
 [ 186  395  394  427 2793  263   23]
 [ 268  414  274  156  267 3105    5]
 [  12   11    2   32   45    0 4297]]
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.79      0.77      4589
           1       0.63      0.64      0.63      4343
           2       0.66      0.61      0.64      4427
           3       0.74      0.75      0.74      4473
           4       0.62      0.62      0.62      4481
           5       0.69      0.69      0.69      4489
           6       0.98      0.98      0.98      4399

    accuracy                           0.73     31201
   macro avg       0.73      0.73      0.73     31201
weighted avg       0.73      0.73      0.73     31201
```

Figure 5 – 'WorkClass'feature confusion matrix and classification report

```
Model: K-Nearest Neighbors
Confusion Matrix:
[[1517   10  124  288  215  400  253]
 [   0 2781    1    0    0    0    0]
 [  86   28 2519   47   67   32   25]
 [  84    5    6 2645   19   58   19]
 [ 378   16  127  215 1842  228   45]
 [ 282   11   17  168  128 2057   85]
 [ 154    0   20   72    9   92 2516]]
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.54      0.57      2807
           1       0.98      1.00      0.99      2782
           2       0.90      0.90      0.90      2804
           3       0.77      0.93      0.84      2836
           4       0.81      0.65      0.72      2851
           5       0.72      0.75      0.73      2748
           6       0.85      0.88      0.87      2863

    accuracy                           0.81     19691
   macro avg       0.80      0.81      0.80     19691
weighted avg       0.80      0.81      0.80     19691


Model: Random Forest
Confusion Matrix:
[[2048    0    0   88  240  234  197]
 [   0 2778    4    0    0    0    0]
 [   3    3 2760    4   28    1    5]
 [  54    0    0 2714   27   26   15]
 [ 370    2    0   74 2211  158   36]
 [ 191    1    0   55   93 2340   68]
 [ 126    0    0   17    7   41 2672]]
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.73      0.73      2807
           1       1.00      1.00      1.00      2782
           2       1.00      0.98      0.99      2804
           3       0.92      0.96      0.94      2836
           4       0.85      0.78      0.81      2851
           5       0.84      0.85      0.84      2748
           6       0.89      0.93      0.91      2863

    accuracy                           0.89     19691
   macro avg       0.89      0.89      0.89     19691
weighted avg       0.89      0.89      0.89     19691


Model: Decision Tree
Confusion Matrix:
[[1637   11    6  164  385  371  233]
 [  13 2727   17    1    5   15    4]
 [   7   14 2751    3   22    4    3]
 [ 120    2    2 2453  108  105   46]
 [ 406   11   33  121 1970  264   46]
 [ 311    4    5  136  179 2013  100]
 [ 226    0    2   50   40   87 2458]]
Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.58      0.59      2807
           1       0.98      0.98      0.98      2782
           2       0.98      0.98      0.98      2804
           3       0.84      0.86      0.85      2836
           4       0.73      0.69      0.71      2851
           5       0.70      0.73      0.72      2748
           6       0.85      0.86      0.85      2863

    accuracy                           0.81     19691
   macro avg       0.81      0.81      0.81     19691
weighted avg       0.81      0.81      0.81     19691
```

Figure 6 – 'Marital Status' feature confusion matrix and classification report

```
Model: K-Nearest Neighbors
Confusion Matrix:
[[3689  743]
 [ 663 3967]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      4432
           1       0.84      0.86      0.85      4630

    accuracy                           0.84      9062
   macro avg       0.84      0.84      0.84      9062
weighted avg       0.84      0.84      0.84      9062


Model: Random Forest
Confusion Matrix:
[[3917  515]
 [ 519 4111]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.88      0.88      4432
           1       0.89      0.89      0.89      4630

    accuracy                           0.89      9062
   macro avg       0.89      0.89      0.89      9062
weighted avg       0.89      0.89      0.89      9062


Model: Logistic Regression
Confusion Matrix:
[[3406 1026]
 [1046 3584]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.77      0.77      4432
           1       0.78      0.77      0.78      4630

    accuracy                           0.77      9062
   macro avg       0.77      0.77      0.77      9062
weighted avg       0.77      0.77      0.77      9062
```

Figure 7 – 'Income' feature confusion matrix and classification report

Appendix B: Pseudo-Code

 # pseudo-code for eda_module.py

```
IMPORT necessary libraries

DEFINE class EDA:

  DEFINE __init__ method:
    INITIALIZE self.data to None
    INITIALIZE self.cleaned_data to Non

  DEFINE load_data method with parameter file_path="people.data":
    CREATE custom_headers list with tuples of (column name, data type)
    CREATE column_dtype_map dictionary from custom_headers
    READ CSV file with pandas, no header, column names and data types from column_dtype_map
    RETURN the loaded data

  DEFINE show_feature_class_distribution method with parameters data, feature and optional ax:
    CALCULATE class_counts as the value counts of the specified feature column in data
    IF ax is None:
      PLOT a bar chart of class_counts with yellow color
      ROTATE x-axis labels 30 degrees
      LABEL x-axis as feature, y-axis as count and title as 'Class Distribution ({feature})'
    ELSE:
      PLOT a bar chart on ax with class_counts and yellow color
      SET x-axis labels on ax with 30 degrees rotation
      SET x-axis label, y-axis label and title on ax

  DEFINE clean_data method with parameter data:
    CHECK if any column contains ' ?' and STORE result in has_question_mark
    FIND columns that contain ' ?' and STORE in columns_with_question_mark
    CREATE data_cleaned as a copy of data
    FOR each feature in columns_with_question_mark:
      FILTER data_cleaned to exclude rows where feature is ' ?'
    RESET the index of data_cleaned
    RETURN data_cleaned

  DEFINE box_plot method with parameters data_cleaned, x_feature, y_feature and optional ax:
    IF ax is None:
      PLOT a seaborn boxplot with x_feature and y_feature from data_cleaned
      ROTATE x-axis labels 30 degrees
      SET title as 'Box Plot: {x_feature} vs {y_feature}'
    ELSE:
      PLOT a seaborn boxplot on ax with x_feature and y_feature from data_cleaned
      SET x-axis labels on ax with 30 degrees rotation
      SET title on ax
```

DEFINE heatmap method with parameters data_cleaned, x_feature, y_feature and optional ax:
  CREATE a crosstab of y_feature and x_feature from data_cleaned
  IF ax is None:
    PLOT a seaborn heatmap of the crosstab with annotations, integer format and Blues color
    SET title as 'Heatmap: {x_feature} vs {y_feature}'
  ELSE:
    PLOT a seaborn heatmap on ax with the crosstab, annotations, integer format, and Blues color
    SET title on ax

DEFINE class_distribution_features_needed method with parameters data_cleaned, feature and optional ax:
  CALCULATE class_counts as the value counts of the specified feature column in data_cleaned
  IF ax is None:
    PLOT a bar chart of class_counts with green color
    ROTATE x-axis labels 30 degrees
    LABEL x-axis as feature, y-axis as count and title as 'Class Distribution ({feature})'
  ELSE:
    PLOT a bar chart on ax with class_counts and green color
    SET x-axis labels on ax with 30 degrees rotation
    SET x-axis label, y-axis label and title on ax

DEFINE corr_heatmap method with parameters data_encoded and optional ax:
  IF ax is None:
    PLOT a seaborn heatmap of the correlation matrix of data_encoded with annotations, copper color and 2 decimal format
    SET title as 'Correlation Heatmap'
  ELSE:
    PLOT a seaborn heatmap on ax of the correlation matrix of data_encoded with annotations, copper color map, and 2 decimal format
    SET title on ax

#pseudo-code for training_module.py

IMPORT necessary libraries

DEFINE class Training:

  DEFINE __init__ method:
    INITIALIZE self.label_encoders as an empty dictionary

  DEFINE encode_variables method with parameter data_cleaned:
    FIND categorical columns in data_cleaned
    CREATE a copy of data_cleaned called data_encoded

    FOR each categorical column:
      INITIALIZE a LabelEncoder for the column and store it in self.label_encoders
      ENCODE the column in data_encoded using the LabelEncoder
    RETURN data_encoded

```
DEFINE split_data method with parameters data_encoded and target:
    PRINT message indicating the feature being split
    ASSIGN X as data_encoded with target column dropped
    ASSIGN y as the target column from data_encoded
    RETURN result of train_test_split on X and y with test size of 20% and random state 21

DEFINE train_and_evaluate_models method with parameters data_encoded and target:
    PRINT message indicating the feature being trained

    # Balance the data using SMOTE
    ASSIGN X as data_encoded with target column dropped
    ASSIGN y as the target column from data_encoded
    INITIALIZE SMOTE with random state 21
    FIT and RESAMPLE X and y using SMOTE
    CREATE data_sm as X with target column added from y

    # Train Test Split
    CALL split_data method with data_sm and target to get X_train, X_test, y_train, y_test

    # Feature scaling
    INITIALIZE StandardScaler
    FIT and TRANSFORM X_train using StandardScaler
    FIT and TRANSFORM X_test using StandardScaler

    # Initialize models
    IF target is 'income':
        INITIALIZE models dictionary with KNeighborsClassifier, RandomForestClassifier, and
LogisticRegression
    ELSE:
        INITIALIZE models dictionary with KNeighborsClassifier, RandomForestClassifier, and
DecisionTreeClassifier

    # DataFrame to store results
    INITIALIZE results DataFrame with columns ['Model', 'Precision', 'Recall', 'Accuracy', 'F1-Score']

    # Train models and collect results
    FOR each model in models:
        FIT the model with X_train and y_train
        PREDICT y_pred using the model on X_test
        CONVERT y_pred and y_test to integer series
        # Store results
        APPEND to results DataFrame with model name and metrics (precision, recall, accuracy, f1-score)
        PRINT model name
        PRINT confusion matrix of y_test and y_pred
        PRINT classification report of y_test and y_pred

    RETURN results
```

#pseudo-code for gui_module.py

```
IMPORT necessary libraries

DEFINE class App that inherits from tk.Tk:

  DEFINE __init__ method:
    CALL superclass __init__ method
    SET window title to "Data Analysis and Model Training Application"
    SET window geometry to "1000x700"
    INITIALIZE EDA and Training instances
    CALL create_widgets method

  DEFINE create_widgets method:
    CREATE eda_frame as a LabelFrame for Data Pre-Processing
    PACK eda_frame with padding and expand options
    CREATE and PACK buttons in eda_frame: Load Data, Describe Data, Show Class Distribution, Clean Data
    CREATE another eda_frame as a LabelFrame for Exploratory Data Analysis
    PACK this eda_frame with padding and expand options
    CREATE and PACK buttons in this eda_frame: Box Plot, Heatmap, Class Distribution of Cleaned Data, Encode Categorical Variables, Correlation Heatmap
    CREATE model_frame as a LabelFrame for Model Training and Evaluation
    PACK model_frame with padding and expand options
    CREATE and PACK buttons in model_frame for training: Train for Income, Train for Marital Status, Train for Workclass
    CREATE results_frame as a LabelFrame for displaying results
    PACK results_frame with padding and expand options
    CREATE and PACK results_text Text widget in results_frame
    CREATE and PACK Quit button

  DEFINE load_data method:
    CALL load_data method of EDA instance to load data
    INSERT "Data Loaded Successfully" message in results_text

  DEFINE describe_data method:
    IF data is loaded:
      DESCRIBE numerical and categorical data
      INSERT descriptive statistics into results_text
    ELSE:
      INSERT "Please load data first" message into results_text

  DEFINE show_class_distribution method:
    IF data is loaded:
      PROMPT user to input feature
      IF feature is valid:
        CALL show_graph_in_popup with show_feature_class_distribution method of EDA instance and the selected feature
      ELSE:
        SHOW error message
    ELSE:
      INSERT "Please load data first" message into results_text
```

```
    DEFINE clean_data method:
      IF data is loaded:
        CALL clean_data method of EDA instance to clean data
        INSERT "Data Cleaned Successfully" message into results_text
      ELSE:
        INSERT "Please load data first" message into results_text

    DEFINE plot_box method:
      IF cleaned data is available:
        PROMPT user to input x_feature and y_feature
        IF features are valid:
          CALL show_graph_in_popup with box_plot method of EDA instance and the selected features
        ELSE:
          SHOW error message
      ELSE:
        INSERT "Please clean data first" message into results_text

    DEFINE plot_heatmap method:
      IF cleaned data is available:
        PROMPT user to input x_feature and y_feature
        IF features are valid:
          CALL show_graph_in_popup with heatmap method of EDA instance and the selected features
        ELSE:
          SHOW error message
      ELSE:
        INSERT "Please clean data first" message into results_text

    DEFINE class_distribution_cleaned method:
      IF cleaned data is available:
        PROMPT user to input feature
        IF feature is valid:
          CALL show_graph_in_popup with class_distribution_features_needed method of EDA instance
  and the selected feature
        ELSE:
          SHOW error message
      ELSE:
        INSERT "Please clean data first" message into results_text

    DEFINE encode_data method:
      IF cleaned data is available:
        CALL encode_variables method of Training instance to encode data
        INSERT "Categorical variables Encoded Successfully" message into results_text
      ELSE:
        INSERT "Please clean data first" message into results_text

    DEFINE plot_corr_heatmap method:
      IF encoded data is available:
        CALL show_graph_in_popup with corr_heatmap method of EDA instance and the encoded data
      ELSE:
        INSERT "Please encode categorical variables first" message into results_text
```

```
DEFINE train_and_evaluate method with parameter target:
  IF encoded data is available:
    CALL train_and_evaluate_models method of Training instance with the encoded data and target
    INSERT training results into results_text
  ELSE:
    INSERT "Please encode categorical variables first" message into results_text

DEFINE show_graph_in_popup method with parameters plot_function and *args:
  CREATE a new Toplevel window for graph
  SET window title to "Graph"
  CREATE a figure and subplot
  CALL plot_function with *args and subplot as ax
  CREATE a FigureCanvasTkAgg for the figure
  DRAW the canvas
  GET the Tk widget from the canvas and PACK it

DEFINE quit_application method:
  SHOW "Goodbye" message
  DESTROY the application
```

#pseudo-code for PCPAssignment2.ipynb

```
IMPORT necessary libraries

IF the script is run as the main module:
  CREATE an instance of the App class from gui_module
  CALL mainloop method on the app instance to start the application's main event loop
```