

Socioeconomic Determinants of Income Classification: A Predictive Modelling Approach

[Andy Wang-02698886] [Hao Qian-02584193] [Nicholas Elton-02650118] [Atharva Mansabdar-02726708]

Dyson School of Design Engineering
Imperial College London
London, United Kingdom

Abstract

Based on the UCI Adult Census Income dataset, we are using different models and factors to make predictions on whether an individual will earn more than \$50K or not. The analysis compares various factors using Decision Tree, SVM, Logistic regression, and Random Forest models. Models were evaluated using accuracy, precision, recall, F1-score, and confusion matrices. The results show that the selected variables contain useful predictive information, but model performance varies across feature groups and evaluation metrics.

1 Introduction

1.1 Project Context

Income is linked to education, employment, household structure, and demographic background. Machine learning can identify patterns in income classification and support data-driven decisions. This report uses the US Adult Census Income Dataset [1], which includes demographic, education, employment, and household information.

1.2 Research Aim

This report investigates whether socioeconomic factors can predict whether an individual earns more than \$50K per year. The task is treated as binary classification: income above \$50K or less than or equal to \$50K. Four predictor groups are compared: education, employment, household and marital-status, and demographic factors.

2 Background

Income classification is important in socioeconomic research because income is associated with education, employment, household structure, and demographic background. These variables are treated as predictors rather than direct causes. This project uses the Adult Census Income dataset to predict whether individuals earn more than \$50K per year [1]. It compares demographic, household, education, employment, and workclass-related feature groups using different classification models to identify predictive patterns rather than causal relationships.

3 Dataset and Methodology

3.1 Dataset

This project uses the Adult Income Dataset obtained from the UCI Machine Learning Repository. The dataset is based on demographic and socioeconomic information collected from the 1994 United States Census. The target variable is income category, classifying individuals as earning either more than \$50,000 per year or less than or equal to \$50,000 per year.

To investigate the relationship between socioeconomic factors and income, the predictor variables were divided into four feature groups based on different demographic and socioeconomic characteristics.

3.2 Data Preprocessing

Before model development, missing or unknown values were removed, and the income variable was converted into a binary variable. Additional preprocessing steps were applied where necessary for individual analyses, including feature selection, categorical variable encoding, feature scaling, and class imbalance handling.

3.3 Modelling Framework

Four machine learning models were applied to four separate feature groups. Workclass, race and native-country variables were analysed using a Decision Tree classifier. Demographic and household characteristics were analysed using a Support Vector Machine (SVM). Education and career-stage variables were analysed using Logistic Regression. Employment-related characteristics were analysed using a Random Forest classifier.

3.4 Model Evaluation

Model performance was evaluated using accuracy, precision, recall, F1-score and confusion matrices, allowing comparisons across different feature groups and machine learning models.

4 Analysis and Results

4.1 Atharva Mansabdar: Predicting Income using Race, Native Country and Workclass

Can the race, native country and working class of adult males determine if their annual income exceeds 50,000 \$ per year?

4.1.1 Subset and feature group

The UCI Adult Census Income dataset, taken from the 1994 US Census, was used. Data was subset to male individuals only, as a higher proportion of men earn above \$50k (Bureau) and to complement teammates' work. This also removed sex as a confounder, as income distributions differ substantially between sexes (Bureau). The feature group comprises three categorical predictors (race, native country, and workclass) and the binary income target. After cleaning, adult.data contained 20,385 records and adult.test contained 10,149 records, reserved for testing.

4.1.2 Model Selection

A decision tree classifier was selected. Unlike logistic regression, it naturally accommodated categorical splits without assuming a linear relationship, and produced an interpretable structure showing which variables are split on, directly answering whether these attributes matter. The tree recursively split the data on whichever feature most reduced the Gini impurity at each node.

4.1.3 Model Setup

Pre-processing The data was cleaned and unknown values (e.g. "??") were removed. The target (income) was binarised, with \$50k set as 1 and $\leq \$50k$ set as 0. The categorical predictors were one-hot encoded, removing one reference category per variable to produce 50 binary columns. The adult.test file needed extra cleaning: its first line was a non-data comment, and its income labels carried a trailing period that was stripped before binarising. The test set was re-indexed onto the training columns so that both shared an identical feature space.

Representativity and Class Balance Within the male subgroup, 69% earned $\leq \$50k$ and 31% earned $> \$50k$. The adult.test file showed a near-identical split, confirming that the test set was representative. The native-country variable expanded into around 40 categories, several very sparse, which limited the reliability of splits on rare categories. This imbalance was addressed in both training and evaluation.

Split-Sample Validation and Balancing The adult.data subset was split 80/20 into training and validation, using stratified sampling to preserve the class ratio. The entire adult.test file served as an independent test set, so the model was fit on training, tuned on validation, and evaluated once on test. To correct the training imbalance, the majority class was randomly undersampled to match the minority class, giving a balanced training set of 10,234 rows. A standard remedy for class imbalance (He and Garcia) was applied to the training set only; validation and test retained natural proportions to reflect real-world performance.

Hyperparameter Tuning

max_depth was the principal hyperparameter. Although greater depth captured more structure, it risked overfitting. Trees were fitted across depths 1–15 with scikit-learn's DecisionTreeClassifier, recording validation accuracy, precision, and recall (Figure 1).

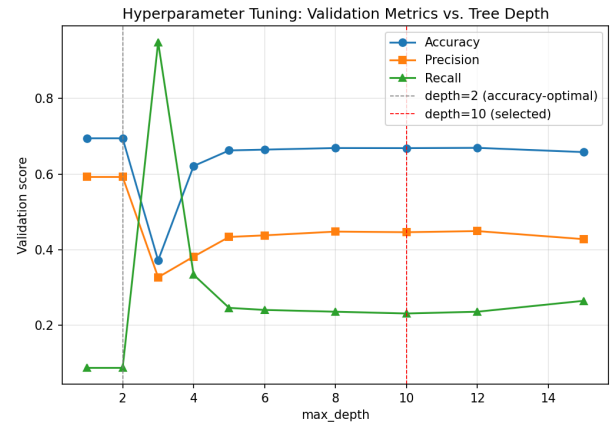


Fig. 1. Validation accuracy, precision & recall vs tree depth

The curve in Figure 1 exhibited three regimes. At depths 1 and 2 the tree was underfit, predicting almost everyone in the majority class (0.70 accuracy, 0.09 recall). At depth 3 it was degenerate: recall spiked to 0.95 while accuracy dropped to 0.37, and the tree flipped to predicting $> \$50k$ for nearly all individuals. From depth 5 onward the metrics stabilised, with accuracy between 0.66 and 0.67 and recall between 0.23 and 0.25. No single depth was optimal on both accuracy and recall, and maximising either metric automatically selected a degenerate model (depth 2 for accuracy, depth 3 for recall). Depth 10 was therefore chosen manually from the stable plateau; it gave near-identical metrics to deeper trees, showing the model would plateau rather than improve.

Choice of Evaluation Metric

As the data was imbalanced, accuracy alone was misleading. This was because a model always predicting $\leq \$50k$ would score 0.69 while learning nothing. The proportion of true high earners correctly identified was prioritised because the positive class was the minority of interest, hence the model's ability to capture it best tested whether the predictors carried usable information.

4.1.4 Results

Decision Tree Structure

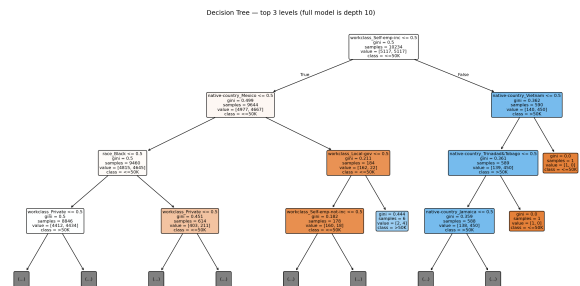


Fig. 2. Top three levels of the final decision tree

As shown in figure 2, the root split is identified on workclass_Self-emp-inc. This indicates incorporated

self-employment most cleanly separating high earners from low ones. The following splits are `native-country_Mexico` and `race_Black`. The form "`feature ≤ 0.5`" is taken as the predictors are one-hot encoded where lesser than 0.5 means not in the category and greater than means in it. Top three levels are shown for readability, the tree reaches depth 10.

As shown in Figure 2, the root split was on `workclass_Self-emp-inc`, indicating that incorporated self-employment separated high earners from low earners. The following splits were `native-country_Mexico` and `race_Black`. Conditions took the form "`feature ≤ 0.5`" because the predictors are one-hot encoded, where 0.5 means not in that category. The top three levels are shown for readability; the tree reaches depth 10, and the full diagram is in the appendix.

The node Gini values showed which groups the tree classified confidently. The root (10,234 balanced samples) had a Gini of 0.500 by construction. Splitting on self-employment had minimal impact on the not self-employed inc branch (Gini 0.499 over 9,644 samples), isolating only a small high-earning minority while leaving the majority almost as mixed. By contrast, a deeper node isolating incorporated self-employed individuals reached a Gini of 0.211 over 184 samples. This pattern explained the modest recall: the tree successfully identified a small subset of high earners but could not resolve most individuals' incomes from these three attributes.

The final depth-10 model was evaluated on the independent `adult.test` set (Table I). Training metrics used the balanced 50/50 set, whereas validation and test used the natural 69/31 distribution, so the training column was not directly comparable. The close validation–test agreement indicates the model was not overfit.

Final Performance

Metric	Training	Validation	Test
Accuracy	0.562	0.669	0.678
Precision	0.668	0.446	0.463
Recall	0.245	0.231	0.249

TABLE I
PERFORMANCE OF THE FINAL DECISION TREE

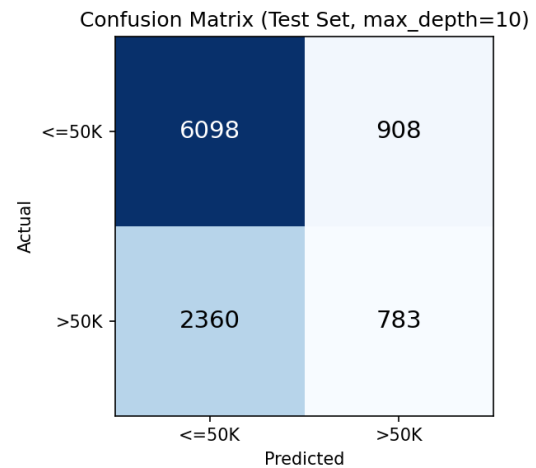


Fig. 3. Confusion matrix for final model

Comparison of Candidate Depths

Test Metric	Depth 2	Depth 10
Accuracy	0.698	0.678
Precision	0.582	0.463
Recall	0.092	0.249

TABLE II
COMPARISON OF CANDIDATE DEPTHS

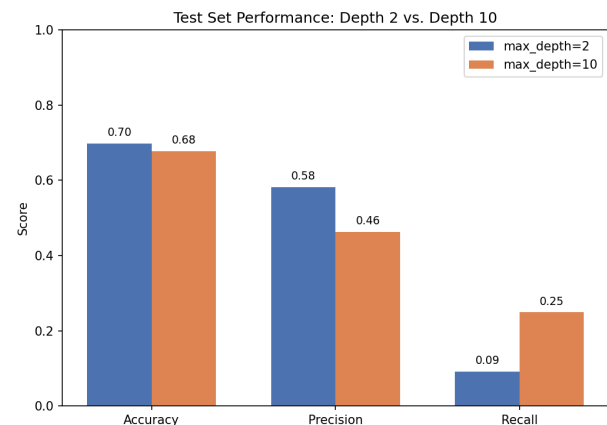


Fig. 4. Depth 2 vs depth 10 performance with test set

The depth-2 model exhibited marginally higher accuracy (0.698 vs 0.678) but a recall of only 0.092. Depth 10 had roughly three times the recall at a small cost to accuracy and precision, a worthwhile trade under the recall-focused criterion, which improves identification of the high-earning class.

Feature Importance

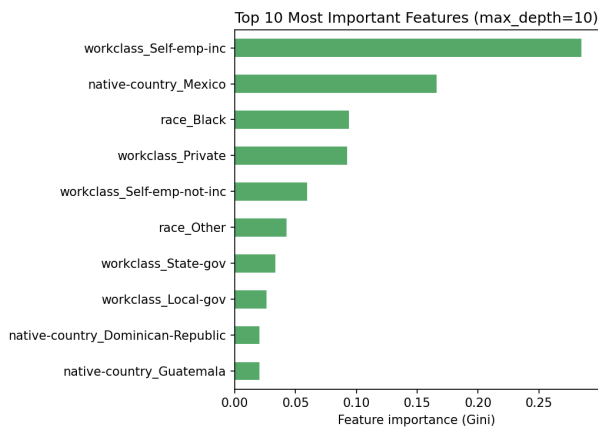


Fig. 5. Top ten most important features by Gini-impurity reduction

Feature importance identified workclass_Self-emp-inc (0.285), native-country_Mexico (0.166), and race_Black (0.094) as the strongest predictors, with other workclass and country categories contributing at lower levels.

4.1.5 Discussion

The groups most associated with earning more than \$50k were self-employed incorporated males (importance 0.285), followed at lower strength by specific race and country categories (Mexico 0.166, Black 0.094). Occupational status was the strongest determining factor, outranking every demographic category. However, the attributes identified high earners only weakly; the tree isolated a few small high-earning pockets but left the majority unresolved. This pointed to association rather than causation, which was expected given that income may depend heavily on absent variables such as education and hours worked.

Several limitations applied: sparse native-country categories meant splits on rare countries rested on very few individuals and were unlikely to generalise, the male-only subgroup cannot be extrapolated to the wider population, and the 1994 data may not reflect current patterns. An improvement would be a random forest, which would yield more stable performance, along with richer predictors such as education. The analysis demonstrated a correct application of the technique and answered the association question clearly, but indicated correlation rather than causation and was limited to the subgroup studied.

4.2 Hao Qian: Age, Sex, Marital-status and Relationship as predictors of Income Classification with SVM

4.2.1 Research Overview

My section will focus on whether demographic and household factors can help to predict an individual's income category using a Support Vector Machine (SVM). The main aim is to determine whether Age, Sex, Marital-status, and Relationship are useful to predict if an individual's annual income exceeds \$50,000.

My study uses the Adult Census Income dataset from the UCI Machine Learning Repository. In the dataset, Age is used as one of the predictor since it is related to individual's work experience and career progression. In addition, Sex, Marital-status, and Relationship are variables that can reflect an individual's demographic features and household factors. These four variables are selected because they are highly associated with employment opportunities, financial responsibilities and the potential of earning. Previous research has shown that marriage and household structure are closely linked to labour market outcomes and income inequality [4].

Since the dependent variable has two possible outcomes (greater than \$50K and less than or equal to \$50K), it can become a binary classification task. A Support Vector Machine (SVM) was selected as the machine learning model. SVM is useful for classification problems and different kernel functions can be used to explore both linear and non-linear decision boundaries.

4.2.2 Methodology

1 Data Preprocessing

The Adult Income dataset was filtered to obtain only the variables required for this analysis: *age*, *marital-status*, *relationship*, *sex*, and *income*, and all other variables were removed from the dataset. Also, rows containing missing values were removed before the data were analysed. The dependent variable, *income*, was converted into a binary variable, where individuals earning more than \$50,000 per year were in one class and those earning \$50,000 or less were in another class. The categorical variables were converted into numerical format so that they can be used in the machine learning model. They were then standardised so that they were measured on the same scale. Finally, the data were split into training and validation sets. A separate Adult test dataset was retained and used only for the final evaluation of the model. It should be noticed that a larger proportion of individuals earning less than or equal to \$50K than those earning more than \$50K which indicates that the income classes are not balanced. Therefore, the model was evaluated using several performance measures, including accuracy, precision, recall, and the confusion matrix, to provide a more comprehensive assessment.

2 Hyper-Parameter Optimisation

a. Kernel Type Selection

Four SVM kernels were compared: *linear*, *rbf*, *poly*, and *sigmoid*. For each kernel, accuracy, precision, and recall were calculated on both the training set and validation set.

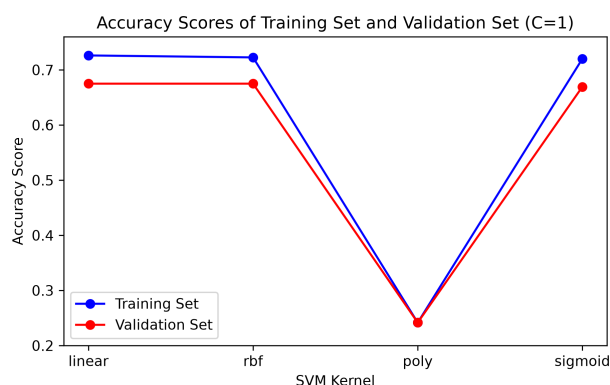


Fig. 6. Accuracy Scores of Training Set & Validation Set by Kernel

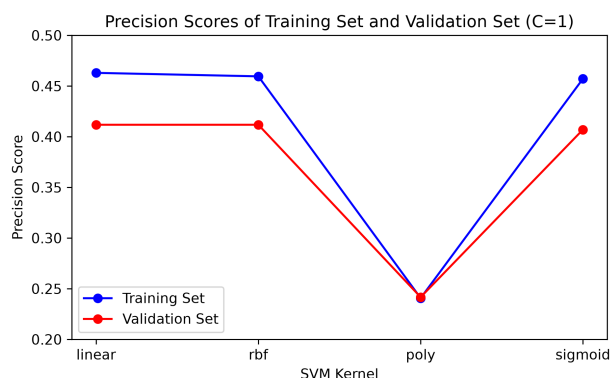


Fig. 7. Precision Scores of Training Set & Validation Set by Kernel

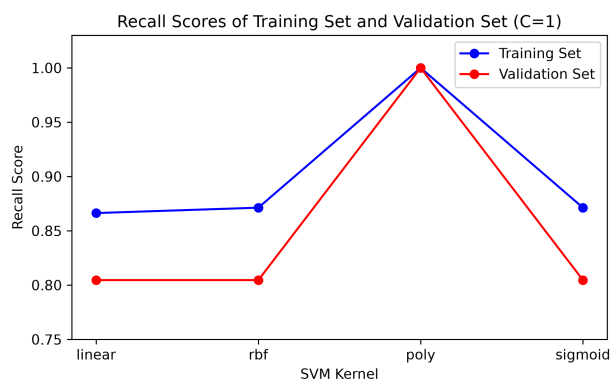


Fig. 8. Recall Scores of Training Set & Validation Set by Kernel

It is shown on the figures that the polynomial kernel achieved a recall score of 1.0 but a relatively lower accuracy and precision. Therefore, the recall performance cannot be considered alone for model selection and other scores should be concerned. The figures above also show that the result of linear, rbf, and sigmoid kernels are similar, but the linear kernel was the final selection since it is simpler, easier to interpret, and faster to train.

b. C Value Selection

After selecting the linear kernel, different values of the regularisation parameter C were tested.

0.1, 0.5, 1, 2, 5, 7, 10, 20, 50

For each C value, validation accuracy, precision, and recall were calculated.

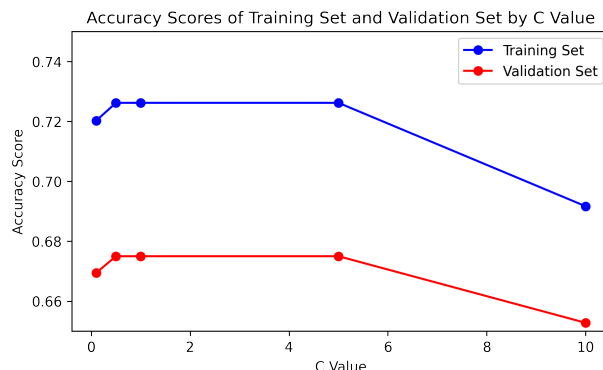


Fig. 9. Accuracy Scores of Training Set & Validation Set, Varying C

The final C value was selected based on the highest validation accuracy. The best C value was:

$$C = 0.5$$

4.2.3 Results

The final SVM model used the linear kernel with $C = 0.5$. The final model was evaluated on the test set using accuracy, precision, recall, and the confusion matrix.

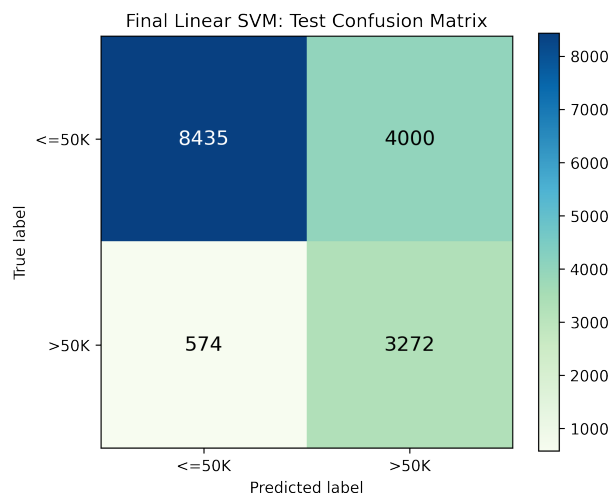


Fig. 10. Final Linear SVM Test Confusion Matrix

The final model achieved a test accuracy of 0.7191, a precision score of 0.4499, and a recall score of 0.8508. The similarity between the validation and test results suggests that the model appeared to generalise reasonably well to unseen data and showed no strong evidence of overfitting.

4.2.4 Discussion

The final SVM model has achieved a test accuracy of 71.9%, a precision score of 44.99%, and a recall score of 85.08%. To better understand the results, the confusion matrix was analysed to provide further insight into the model's performance. The model correctly identified 3,272 individuals earning more than \$50K and it missed only 574 high-income individuals. This explains the high recall score, indicating that the model was good at identifying most individuals belonging to the class of higher-income. However, 4,000 individuals earning less than or equal to \$50K were incorrectly classified as high-income earners. As a result, the model produced a large number of false positives, which is the reason why there is a relatively low precision score.

In the kernel selection, the polynomial kernel achieved a recall score of 1.0. However, the confusion matrix suggested that the model classified almost all observations as belonging to the positive class. As a result, there were very few false negatives, which led to the very high recall score. However, this also produced a large number of false positives and reduced the overall accuracy of the model. These results show that recall cannot be used on its own when evaluating the model performance. In this situation, it should be considered together with other measures such as accuracy, precision, and the confusion matrix.

Overall, the results suggest that demographic and household factors, including age, sex, marital status, and relationship status, can be used as predictors for the classification of individual's income. However, these variables alone are not sufficient to fully explain whether an individual's annual income exceeds \$50K.

To further improve the predictive performance of the model, additional variables such as education level, occupation, and hours worked per week may be considered. These factors are possible to have a direct relationship with income and may improve both accuracy and precision. A further limitation is that the Adult Income dataset is based on demographic and socioeconomic information collected from the 1994 United States Census. Consequently, the relationships identified by the model may not fully reflect current labour market conditions, as labour market structures and income inequality have changed substantially over recent decades [5]. Furthermore, because the dataset is based solely on the United States population, the conditions may not be similar to other countries. Future studies could use more recent and geographically diverse datasets to assess the generalisability of the model.

4.3 Andy Wang: Education and Career Stage as predictors of Income Classification with logistic regression

4.3.1 Research Overview

My section focuses on how education and career stage can help to predict income class using Logistic Regression.

The main question is whether education level and age are useful predictors of whether an individual earns more than

\$50,000\$ per year. In the dataset, education-num represents educational attainment, while age serves as an indicator of potential work experience because there is no direct feature for work experience.

Lastly, hours-per-week are added to test whether working hours can improve prediction accuracy. Since income is a binary classification outcome, Logistic Regression is well-suited for this prediction [6]. It also allows the effect of each variable to be interpreted through its model coefficient, making it easier to compare education-num, age, and hours-per-week. Compared with more complex models, Logistic Regression is more interpretable while still allowing multiple factors to be considered together.

4.3.2 Methodology

1 Variable Selection

The selected variables are education-num, age, and hours-per-week. Education-num was selected because it provides a numerical representation of education level that can be used directly as input for Logistic Regression. It was used instead of the education category to avoid using two variables describing similar education information.

Working experience may affect the importance of education, as education level may be more important for recent graduates than for people with longer industry experience. As the dataset does not include a direct work experience feature, age was used as a proxy for potential work experience or career stage. Lastly, hours-per-week was added to test whether working time could improve prediction accuracy, as people working longer hours may have different income patterns.

2 Data Preparation

The dataset was provided in two files: `adult.data` and `adult.test`. The `adult.data` file was used for training and validation, while `adult.test` was kept separate for final testing. As the dataset did not include column names, the variable names were assigned according to the dataset description file.

Before training, missing values were removed, and income was converted into a binary target. Individuals earning less than or equal to \$50K were coded as 0, while individuals earning more than \$50K were coded as 1. Three training-validation split ratios, 60:40, 70:30, and 80:20, were compared to check whether the results were affected by the amount of training data. The validation results were also used to assess potential overfitting by comparing training and validation performance. The `adult.test` dataset remained independent and was only used for final testing.

3 Model Training and Evaluation

Three Logistic Regression models were trained with increasing feature complexity. Model 1 used education-num only. Model 2 used education-num and age. Model 3 used education-num, age, and hours-per-week. The models were compared using accuracy, precision, recall, F1-score, and the confusion matrix [7].

4.3.3 Logistic Regression Model development

Logistic Regression predicts the probability of an individual earning above \$50K using the selected variables as inputs.

A probability of 0.5 or above was classified as above \$50K; otherwise, it was classified as less than or equal to \$50K.

Three models with increasing feature complexity were tested using 60:40, 70:30, and 80:20 training-validation splits to assess stability and possible overfitting before final testing on adult.test.

4.3.4 Results

The results for three different data split ratios across three different models are shown in Figure 1-3. The figure shows the accuracy compared to model type(complexity):

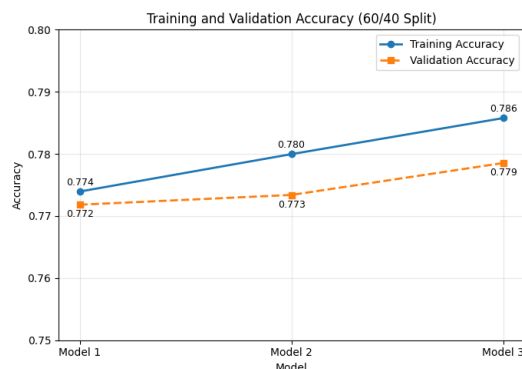


Fig. 11. Training and validation accuracy for the Logistic Regression models under the 60:40 split.

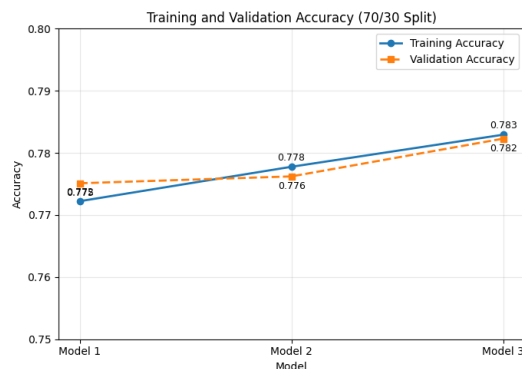


Fig. 12. Training and validation accuracy for the Logistic Regression models under the 70:30 split.

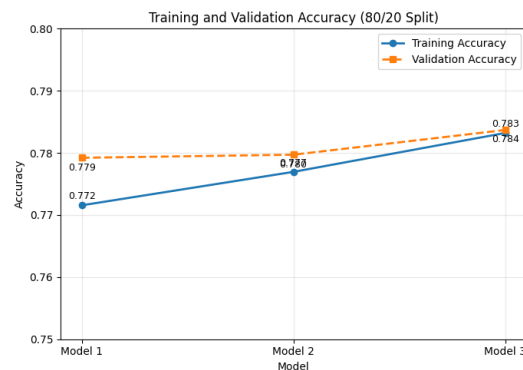


Fig. 13. Training and validation accuracy for the Logistic Regression models under the 80:20 split.

The Figure 11 shows that both training and validation accuracy increased as more variables were added to the model. Under this split, the validation accuracies of the three models were 0.772, 0.773, and 0.779, respectively. The training accuracy was 0.774, 0.780 and 0.786, respectively. Although the accuracy increased for both lines, the validation accuracy is consistently lower than the training accuracy, indicating that a small training set may limit model performance due to this generalisation gap. But as the difference is less than 1%, there's no strong evidence of overfitting.

The Figure 12 shows that both training and validation accuracy increased as more variables were added. Unlike the 60:40 split, validation accuracy was higher than training accuracy for Model 1, while training accuracy was slightly higher for Models 2 and 3.

The average training-validation difference was 0.0017, lower than 0.0053 for 60:40 and 0.0036 for 80:20. This suggests that the 70:30 split produced more stable performance than the 60:40 split.

Lastly, Figure 13 shows that the validation accuracy is higher than the training accuracy, but the difference is relatively small. The overall accuracy of the 80/20 split is highest among the three split ratios. Which is 0.7791 compared to 0.778 for the 70/30 split and 0.7772 for the 60/40 split.

Split	Average Training Accuracy	Average Validation Accuracy	Average Absolute Difference
60/40	0.7799	0.7746	0.0053
70/30	0.7776	0.7779	0.0017
80/20	0.7772	0.7809	0.0036

Fig. 14. Summary table of average training and validation accuracy for different training-validation split ratios.

Therefore, the 80/20 split is used for the final prediction, as it achieves the highest overall validation accuracy of 0.7809 across three splits. It also allow the use of largest training set which can help improve the accuracy of the model. Therefore, for final testing, the 80/20 split will be used.

4.3.5 Final model testing and evaluation

After selecting the 80:20 split, further evaluation of Models 1, 2, and 3 was conducted to obtain accuracy, precision, recall, F1-score, and confusion matrices for each model. This allows for a more detailed comparison of the model performance.

The next figure shows the accuracy, precision, recall and F1-score of three models.

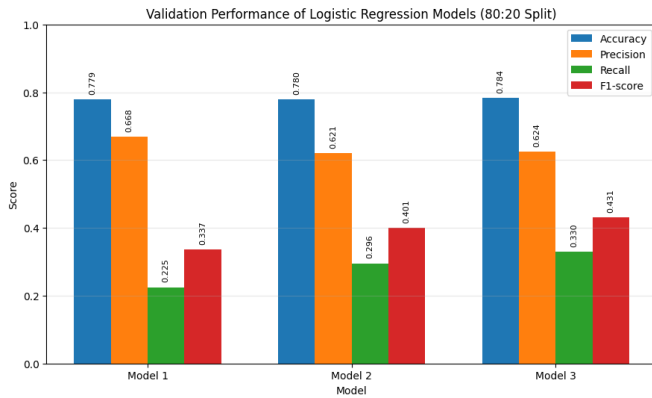


Fig. 15. Accuracy, precision, recall, and F1-score of three models using the 80:20 split.

Figure 15 compares the validation performance of the three Logistic Regression models. Accuracy increased slightly from 0.779 for Model 1 to 0.784 for Model 3, but accuracy alone is not sufficient for comparison.

As more variables were added, recall increased from 0.223 in Model 1 to 0.330 in Model 3, meaning that Model 3 identified more higher-income individuals. Although Model 3 had slightly lower precision than Model 1, it achieved the highest F1-score of 0.431, compared with 0.337 for Model 1 and 0.401 for Model 2. As the F1-score provides a more balanced measure of precision and recall, Model 3 was selected for final testing on the adult dataset.test dataset.

Final Testing Results

The selected Model 3 was retrained using the full adult.data dataset and then tested using the separate adult.test dataset. The results are shown below:

Final Test Performance of Model 3 on adult.test

Model	Accuracy	Precision	Recall	F1-score
Model 3	0.7837	0.6088	0.3343	0.4316

Fig. 16. Final testing results of Model 3 using the adult.test dataset.

4.3.6 Discussion of Results

The results show that Model 3, when tested on the adult.test dataset, achieved an accuracy of 0.7837, a precision of 0.6088, a recall of 0.3343, and an F1-score of 0.4316. These results were close to the validation results, suggesting stable performance

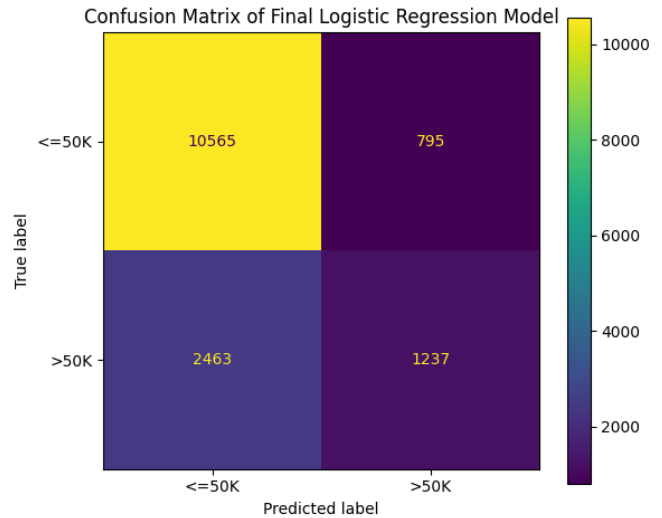


Fig. 17. Confusion matrix of Model 3 using the adult.test dataset.

on unseen data. Although the accuracy was relatively high, the lower F1-score due to the low recall value shows that the model was less effective at consistently identifying individuals earning more than \$50k

On the other hand, the confusion matrix indicates that the model successfully identified 1237 higher-income individuals, but missed 2463 individuals earning more than \$50k. This explains the low recall value. Overall, these results suggest that while education, age, and hours per week are useful predictors of income class, they may not be sufficient to accurately predict it. Furthermore, a lack of work-experience data may limit the model's performance. To further improve the model, we can consider adding more features to achieve optimal performance.

4.4 Nicholas Elton: Predicting employment characteristics and income

4.4.1 Research Overview

Predicting income from employment characteristics is an important problem in labour market analysis, workforce planning and socioeconomic research.

This study uses the Adult Census Income dataset from the UCI Machine Learning Repository, focusing specifically on employment-related influences, where only workclass, occupation, hours worked per week, capital gain and capital loss were included in the analysis.

The aim of this investigation is to determine whether employment characteristics alone can predict whether an individual earns more than \$50,000 per year. The research question is therefore:

Can employment-related characteristics predict whether an individual earns more than \$50,000 per year?

A Random Forest classifier was selected because it can model complex non-linear relationships, provides measures

of feature importance that enable interpretation of the factors driving predictions, and reduces the risk of overfitting.

4.4.2 Methodology

The Adult Census Income dataset was used to develop a Random Forest classifier to predict whether an individual earned more than \$50,000 per year.

Five employment-related predictor variables were selected: workclass, occupation, hours worked per week, capital gain and capital loss. Missing observations were removed, and categorical variables were transformed into numerical features using one-hot encoding. The target variable was converted into a binary format suitable for classification.

An 80:20 train-test split was applied using stratified sampling to preserve class proportions in both datasets. A Random Forest classifier was trained with balanced class weights to mitigate class imbalance. Model performance was evaluated using accuracy, precision, recall, F1-score and confusion matrices.

4.4.3 Data Filtering, Cleaning and Balancing

Only variables directly related to employment characteristics were retained for analysis. These included workclass, occupation, hours worked per week, capital gain and capital loss. Records containing missing values were removed, resulting in a final dataset containing 46,033 observations.

The income variable was converted into a binary target, where individuals earning more than \$50,000 were assigned a value of 1 and individuals earning \$50,000 or less were assigned a value of 0. Because Random Forest models require numerical inputs, the categorical variables workclass and occupation were transformed using one-hot encoding.

Figure 18 shows the distribution of income classes following preprocessing. A substantial class imbalance is evident, with considerably more individuals earning \$50,000 or less than those earning above this. To reduce potential bias towards the majority class, balanced class weights were applied during model training. Stratified sampling was subsequently used when splitting the dataset to ensure both training and testing subsets maintained similar class distributions.

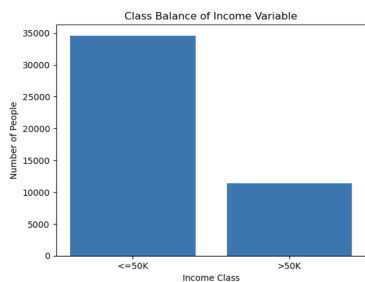


Fig. 18. Distribution of income classes after data cleaning and preprocessing.

4.4.4 Hyperparameter Optimisation

Hyperparameter optimisation was performed using Grid-SearchCV with five-fold cross-validation. Several combinations

of the number of trees, maximum tree depth, minimum samples required for splitting and minimum samples per leaf were evaluated.



Fig. 19. Effect of maximum tree depth on cross-validation F1 score during Random Forest hyperparameter optimisation.

Figure 19 shows the relationship between maximum tree depth and the mean cross-validation F1 score. Increasing tree depth improved model performance, with the highest score achieved at a depth of 15. This suggests that deeper trees were able to capture more complex relationships between employment characteristics and income without substantial overfitting. The optimal model used 100 trees, a maximum depth of 15, a minimum split size of 2 and a minimum leaf size of 1.

4.4.5 Results

Figure 20 summarises the performance of the initial and tuned Random Forest models. The initial model achieved an accuracy of 75.4%, precision of 50.2%, recall of 71.6% and an F1-score of 59.0%. Following hyperparameter optimisation, performance improved slightly, achieving an accuracy of 76.6%, precision of 52.2%, recall of 70.4% and an F1-score of 59.9%.

	Model	Accuracy	Precision	Recall	F1 Score
0	Initial Random Forest	0.753666	0.502459	0.715849	0.590466
1	Tuned Random Forest	0.766373	0.521584	0.703590	0.599068

Fig. 20. Comparison of performance metrics for the initial and tuned Random Forest models.

The increase in accuracy and F1-score suggests that hyperparameter optimisation improved the model's overall predictive performance. Although recall decreased marginally, the increase in precision indicates a reduction in false positive predictions. Overall, the tuned model provided a better balance between identifying high-income individuals and minimising incorrect classifications.

The confusion matrix for the tuned model is shown in Figure 21.

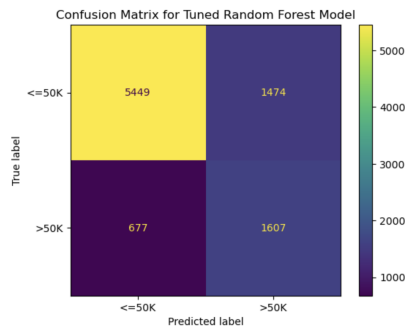


Fig. 21. Confusion matrix for the tuned Random Forest model.

The model correctly classified 5,449 individuals earning \$50,000 or less and 1,607 individuals earning more than \$50,000. However, 1,474 low-income individuals were incorrectly classified as high-income, while 677 high-income individuals were incorrectly classified as low-income. These results indicate that the model was generally effective but still struggled to distinguish some individuals near the decision boundary. Feature importance analysis was conducted to identify the variables contributing most strongly to model predictions. Figure 22 shows that occupation was the most influential feature group (0.302), followed by capital gain (0.277) and hours worked per week (0.237). Capital loss contributed moderately (0.125), whereas workclass had relatively little influence (0.059).

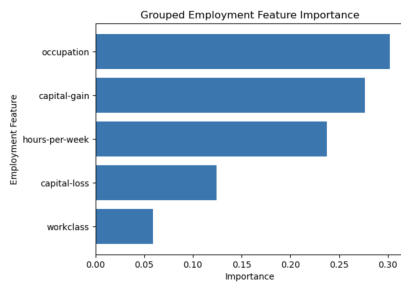


Fig. 22. Grouped employment feature importance for the tuned Random Forest model.

These findings suggest that occupation and financial indicators are more important predictors of income than the type of employer.

4.4.6 Discussion

The results demonstrate that employment-related characteristics contain substantial predictive information regarding income level. The tuned Random Forest model achieved an accuracy of 76.6%, indicating that employment variables alone can predict whether an individual earns more than \$50,000 per year with reasonable success.

Feature importance analysis identified occupation as the strongest predictor of income, followed by capital gain and hours worked per week. This suggests that job role and financial activity are more influential determinants of income than employer type. The relatively low importance of workclass

indicates that the nature of an individual's occupation provides more predictive information than the sector in which they are employed.

Despite achieving reasonable predictive performance, the model exhibited only moderate precision, indicating that some individuals were incorrectly classified as high-income earners. This suggests that employment characteristics alone cannot fully explain income variation. Additional variables such as education level, age and marital status may improve predictive performance.

Several limitations should be acknowledged. The Adult Census Income dataset is based on data collected in 1994 and may not fully represent modern labour market conditions. In addition, removing observations with missing values may have introduced bias. Finally, although Random Forest models provide strong predictive performance, they are less interpretable than simpler models such as logistic regression.

Overall, the findings support the hypothesis that employment-related characteristics can be used to predict income category, with occupation, capital gain and hours worked per week identified as the most influential predictors.

5 Conclusion

Across the four analyses, socioeconomic factors provided useful but limited information for predicting whether an individual earns more than \$50K per year. Employment and education-related variables showed the strongest predictive value. The Random Forest model using employment characteristics achieved the best overall balance, with an accuracy of 0.766 and an F1-score of 0.599. The SVM achieved the highest recall of 0.851, but its lower precision showed that many lower-income individuals were predicted as high-income. The Decision Tree and Logistic Regression achieved more modest performance but provided clearer interpretation of the selected variables.

These results show that the feature groups can support income classification, but no individual group was sufficient to fully predict income. The 1994 dataset, class imbalance, and use of sensitive variables such as race and sex are important limitations. Future work could combine the feature groups, use more recent and diverse data, and include fairness analysis before applying these models in practice.

Appendix A Source Code Screenshots

This appendix contains screenshots of the main code used for each individual analysis.

A.1 Atharva Mansabdar: Decision Tree Analysis

A.2 Hao Qian: Support Vector Machine Analysis

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.metrics import train_test_split
from sklearn.metrics import classification_report, export_graphviz, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

df = pd.read_csv('figure-01.csv')

column_names =
    'name', 'workclass', 'education', 'education-num',
    'marital-status', 'occupation', 'relationship', 'race', 'sex',
    'native-born', 'native-born', 'foreign-born', 'native-union',
    'income'

df = df.read_csv(
    'dataset/census.data',
    header=None,
    names=column_names,
    skipinitialspace=True)

print('census data shape:', df.shape)

df_test_row = df.read_csv(
    'dataset/census.test',
    header=None,
    names=column_names,
    skipinitialspace=True,
    skiprows=1)

print('census test shape:', df_test_row.shape)

df = df[['name', 'workclass', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-born', 'foreign-born', 'native-union', 'income']]
df = df.dropna()
df_test_row = df_test_row[['name', 'workclass', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-born', 'foreign-born', 'native-union', 'income']]
df_test_row = df_test_row.dropna()

print('census data after cleaning (only output):', df.shape)

```

```

train = accnet(train_x_train, train_y_train, accnet=
accnet)
train(train_x_train, train_y_train)

accuracy = accuracy_score(train_labels, train_predictions)
train_balanced = p.compute(train_labels, train_predictions)

train_x_train, train_y_train, train_balanced = train_data_loader(
train_x_train, train_y_train, train_balanced)

train_balanced['training set size'] = train_labels
train_balanced['class counts'] = dict(train_labels.items())

figs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

for fig in figs:
    model = ResNet50Classifier(num_depth=figs, random_state=0)
    model.fit(train_x_train, train_y_train)
    pred_val = model.predict_val(train_x_val, train_y_val)
    result.append(
        {
            'num_depth': num_depth,
            'accuracy': accuracy_score(pred_val, train_y_val),
            'precision': precision_score(pred_val, train_y_val),
            'recall': recall_score(pred_val, train_y_val)
        })

results_df = pd.DataFrame(results)
print(results_df.groupby('num_depth').mean())

fig, ax = plt.subplots(figsize=(10, 10))
ax.plot(results_df['num_depth'], results_df['accuracy'], label='accuracy')
ax.plot(results_df['num_depth'], results_df['precision'], label='precision')
ax.plot(results_df['num_depth'], results_df['recall'], label='recall')
ax.legend()

ax.plot(results_df['num_depth'], results_df['accuracy'], label='accuracy')
ax.plot(results_df['num_depth'], results_df['precision'], label='precision')
ax.plot(results_df['num_depth'], results_df['recall'], label='recall')
ax.legend()

ax.set_xlabel('Validation error')
ax.set_ylabel('Validation score')
ax.set_title('Validation error vs. True Depth')

```

```
for i, col in enumerate(confusion_matrices):
    for j, val in enumerate(confusion_matrices[i]):
        offset = -min(dims of i & j) + dims of column
        ax.text(i, offset, val, s=10, z=10, color='black', fontsize=8)
plt.tight_layout()
plt.savefig('fig5_dptpt_confusion_matrices.png', dpi=150)
plt.show()

cm = confusion_matrix(y_test, pred_dpt8)
print('Confusion Matrix (dpt8dpt8):')
print(cm)

fig, ax = plt.subplots(figsize=(15, 8))
im = ax.imshow(cm, cmap='Blues')
ax.set_title('cm').ax.set_xticklabels(['dpt8dpt8', '>dpt8'])
ax.set_yticklabels(['>dpt8', 'dpt8dpt8']).ax.set_yticklabels(['>dpt8', '>dpt8'])
ax.set_xlabel('Predicted class')ax.set_ylabel('Actual')
ax.set_title('Confusion Matrix (Test Set, max_depth=8)')
ax.set_ylabel('')
for i in range(2):
    for j in range(2):
        ax.text(i, 0.5 + j, cm[i, j], s=10, color='black', fontsize=8)
ax.set_title('dpt8')
plt.savefig('fig6_dptpt_confusion_matrices.png', dpi=150)
plt.show()

importances = pd.Series(fit_max_depth_feature_importances_.importance_scores_, index=X.columns)
top_features = importances.sort_values(ascending=False).head(18)

fig, ax = plt.subplots(figsize=(15, 8))
top_features.sort_values(ascending=False).plot(kind='barh', ax=ax, color='#FF5566')
ax.set_xlabel('Feature importance scores')
ax.set_ylabel('Top 18 most Important Features (max_depth=8)')
ax.set_title('fig5')
plt.savefig('fig5_feature_importance_scores.png', dpi=150)
plt.show()

print(top_features.round(3))

fig, ax = plt.subplots(figsize=(15, 8))
plt.title('
    final_model, max_depth=,
    feature_names_columns=,
    class_names=[feature_names],

```

[illegible]

```
ax.legend()
ax.grid(alpha=0.3)
plt.title('Layout 1')
plt.savefig('fig2_hyparameter_tuning.png', dpi=150)
plt.show()

model_d2 = DecisionTreeClassifier(max_depth=2, random_state=0)
model_d2.fit(X_train_val, y_train_val)
pred_d2 = model_d2.predict(X_test)

final_model = DecisionTreeClassifier(max_depth=10, random_state=0)
final_model.fit(X_train_val, y_train_val)
pred_d10 = final_model.predict(X_test)

comparison = pd.DataFrame({
    'max_depth': [accuracy_optimal]: {
        'Accuracy': accuracy_score(y_test, pred_d2),
        'Precision': precision_score(y_test, pred_d2),
        'Recall': recall_score(y_test, pred_d2),
    },
    'max_depth': [selected]: {
        'Accuracy': accuracy_score(y_test, pred_d10),
        'Precision': precision_score(y_test, pred_d10),
        'Recall': recall_score(y_test, pred_d10),
    }
})

print(comparison.round(3))

fig, ax = plt.subplots(figsize=(7, 5))
c = np.arange(len(comparison.index))
width = 0.35

ax.bar(c - width/2, comparison['max_depth2 (accuracy-optimal)'],
       width, label='max_depth2 (accuracy-optimal)',
       color='#4682B4')
ax.bar(c + width/2, comparison['max_depth10 (selected)',
                                width, label='max_depth10 (selected)',
                                color='#FFD700')

ax.set_xticks(c)
ax.set_xticklabels(comparison.index)
ax.set_ylabel('Score')
ax.set_title('Test Set Performance: Depth 2 vs. Depth 10')
ax.legend()
ax.set_ylim(0, 1)
```

```
filled=True, rounded=True,
fontsize=8, ax=ax

ax.set_title('Decision Tree - top 3 levels (full model is depth 10)')
plt.tight_layout()
plt.savefig('fig6_tree_top3levels.png', dpi=150)
plt.show()

fig, ax = plt.subplots(figsize=(80, 40))
plt.title(
    final_model,
    max_depth=10,
    feature_names=X.columns,
    class_names=['<=50K', '>50K'],
    filled=True,
    rounded=True,
    fontsize=6,
    ax=ax
)
ax.set_title('Full Decision Tree - all 10 levels, fontsize=20')
plt.tight_layout()
plt.savefig('fig7_full_tree.png', dpi=100, bbox_inches='tight')
print('Saved as fig7_full_tree.png')
```

```
1 import warnings
2
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 from sklearn.exceptions import ConvergenceWarning
7 from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
8 from sklearn.model_selection import train_test_split
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.svm import SVC
11
12
13 warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

```

17 columns = [
18     "age", "workclass", "fnlwtgt", "education", "education-num",
19     "marital-status", "occupation", "relationship", "race", "sex",
20     "capital-gain", "capital-loss", "hours-per-week", "native-country",
21     "income",
22 ]
23
24 data = pd.read_csv(
25     "adult.data",
26     names=columns,
27     na_values="?",
28     skipinitialspace=True,
29 )
30
31 test_data = pd.read_csv(
32     "adult.test",
33     names=columns,
34     na_values="?",
35     skipinitialspace=True,
36     skiprows=1,
37 )
38
39
40 selected_columns = ["age", "marital-status", "relationship", "sex", "income"]
41
42 data = data[selected_columns]
43 test_data = test_data[selected_columns]
44
45
46
47

```

```

49 print("Training data missing values before cleaning:")
50 print(data.isnull().sum())
51
52 print("\nTest data missing values before cleaning:")
53 print(test_data.isnull().sum())
54
55 data = data.dropna()
56 test_data = test_data.dropna()
57
58 print("\nTraining data shape after cleaning:", data.shape)
59 print("Test data shape after cleaning:", test_data.shape)
60
61
62 data["income"] = data["income"].str.replace(".", "", regex=False)
63 test_data["income"] = test_data["income"].str.replace(".", "", regex=False)
64
65 data["income"] = data["income"].map({"<=50K": 0, ">50K": 1})
66 test_data["income"] = test_data["income"].map({"<=50K": 0, ">50K": 1})
67
68
69
70 X = data[["age", "marital-status", "relationship", "sex"]]
71 y = data["income"]
72
73 X_test = test_data[["age", "marital-status", "relationship", "sex"]]
74 y_test = test_data["income"]

```

```
80 print("\ntraining predictor shape after encoding:", X.shape)
81 print("Test predictor shape after encoding:", X_test.shape)
82
83
84
85
86 sample_size = 1280
87
88 X_model, unused_X, y_model, unused_y = train_test_split(
89     X,
90     y,
91     train_size=sample_size,
92     random_state=42,
93     stratify=y,
94 )
95
```

```

95 X_train, X_valid, y_train, y_valid = train_test_split(
96     X_model,
97     y_model,
98     test_size=0.30,
99     random_state=42,
100     stratify=y_model,
101 )
102
103 print("\nModel sample shape:", X_model.shape)
104 print("Training set shape:", X_train.shape)
105 print("Validation set shape:", X_valid.shape)
106
107
108 scaler = StandardScaler()
109
110 X_train_scaled = scaler.fit_transform(X_train)
111 X_valid_scaled = scaler.transform(X_valid)
112 X_test_scaled = scaler.transform(X_test)
113
114 kernels = ["linear", "rbf", "poly", "sigmoid"]
115 kernel_results = []
116
117
118 for kernel in kernels:
119     model = SVC(C=1, kernel=kernel, gamma=0.005, class_weight="balanced")
120     model.fit(X_train_scaled, y_train)
121
122     train_pred = model.predict(X_train_scaled)
123     valid_pred = model.predict(X_valid_scaled)
124
125     kernel_results.append([
126         kernel,
127         accuracy_score(y_train, train_pred),
128         accuracy_score(y_valid, valid_pred),
129         precision_score(y_train, train_pred, zero_division=0),
130         precision_score(y_valid, valid_pred, zero_division=0),
131         recall_score(y_train, train_pred, zero_division=0),
132         recall_score(y_valid, valid_pred, zero_division=0),
133     ])
134
135 kernel_results_df = pd.DataFrame(
136     kernel_results,
137     columns=[
138         "Kernel",
139         "Training Accuracy",
140         "Validation Accuracy",
141         "Training Precision",

```

```

142         "Validation Precision",
143         "Training Recall",
144         "Validation Recall",
145     ],
146 )
147
148 print("\nKernel comparison results:")
149 print(kernel_results_df)
150
151
152
153 plt.figure(figsize=(7, 4))
154 plt.plot(kernel_results_df["Kernel"], kernel_results_df["Training Accuracy"],
155          marker="o", color="blue", label="Training Set")
156 plt.plot(kernel_results_df["Kernel"], kernel_results_df["Validation Accuracy"],
157          marker="o", color="red", label="Validation Set")
158 plt.title("Accuracy Scores of Training Set and Validation Set (C=1)")
159 plt.xlabel("SVM Kernel")
160 plt.ylabel("Accuracy Score")
161 plt.ylim(0.20, 0.76)
162 plt.legend()
163 plt.savefig("figure_accuracy_vs_kernel.png", dpi=300, bbox_inches="tight")
164 plt.show()
165
166 plt.figure(figsize=(7, 4))
167 plt.plot(kernel_results_df["Kernel"], kernel_results_df["Training Precision"],
168          marker="o", color="blue", label="Training Set")
169 plt.plot(kernel_results_df["Kernel"], kernel_results_df["Validation Precision"],
170          marker="o", color="red", label="Validation Set")
171 plt.title("Precision Scores of Training Set and Validation Set (C=1)")
172 plt.xlabel("SVM Kernel")
173 plt.ylabel("Precision Score")
174 plt.ylim(0.20, 0.50)
175 plt.legend()
176 plt.savefig("figure_precision_vs_kernel.png", dpi=300, bbox_inches="tight")
177 plt.show()
178
179 plt.figure(figsize=(7, 4))
180 plt.plot(kernel_results_df["Kernel"], kernel_results_df["Training Recall"],
181          marker="o", color="blue", label="Training Set")
182 plt.plot(kernel_results_df["Kernel"], kernel_results_df["Validation Recall"],
183          marker="o", color="red", label="Validation Set")
184 plt.title("Recall Scores of Training Set and Validation Set (C=1)")
185 plt.xlabel("SVM Kernel")
186 plt.ylabel("Recall Score")
187 plt.ylim(0.75, 1.03)
188 plt.legend()

```

```

189 plt.savefig("figure_recall_vs_kernel.png", dpi=300, bbox_inches="tight")
190 plt.show()
191
192
193
194 kernel_results_df["Simplicity Rank"] = kernel_results_df["Kernel"].map({
195     "linear": 1,
196     "rbf": 2,
197     "poly": 3,
198     "sigmoid": 4,
199 })
200
201 best_kernel = kernel_results_df.sort_values(
202     ["Validation Accuracy", "Simplicity Rank"],
203     ascending=[False, True],
204     ).iloc[0]["Kernel"]
205
206 print("\nBest kernel:", best_kernel)
207
208
209 kernel_check_results = []
210
211 for kernel in kernels:
212     model = SVC(C=1, kernel=kernel, gamma=0.005, class_weight="balanced")
213     model.fit(X_train_scaled, y_train)
214     valid_pred = model.predict(X_valid_scaled)
215
216     tn, fp, fn, tp = confusion_matrix(y_valid, valid_pred).ravel()
217     predicted_under_50k = sum(valid_pred == 0)
218     predicted_over_50k = sum(valid_pred == 1)
219
220     kernel_check_results.append([
221         kernel,
222         predicted_under_50k,
223         predicted_over_50k,
224         tn,
225         fp,
226         fn,
227         tp,
228     ])
229
230 kernel_check_df = pd.DataFrame(
231     kernel_check_results,
232     columns=["Kernel", "Predicted <=50K", "Predicted >50K", "TN", "FP", "FN", "TP"],
233 )
234
235 print("\nKernel prediction count and confusion matrix check:")

```

```

236 print(kernel_check_df)
237
238
239 C_values = [0.1, 0.5, 1, 2, 5, 7, 10, 20, 50]
240 C_results = []
241
242 for C in C_values:
243     model = SVC(C=C, kernel=best_kernel, gamma=0.005, class_weight="balanced")
244     model.fit(X_train_scaled, y_train)
245
246     train_pred = model.predict(X_train_scaled)
247     valid_pred = model.predict(X_valid_scaled)
248
249     C_results.append([
250         C,
251         accuracy_score(y_train, train_pred),
252         accuracy_score(y_valid, valid_pred),
253         precision_score(y_valid, valid_pred, zero_division=0),
254         recall_score(y_valid, valid_pred, zero_division=0),
255     ])
256
257 C_results_df = pd.DataFrame(
258     C_results,
259     columns=[
260         "C",
261         "Training Accuracy",
262         "Validation Accuracy",
263         "Validation Precision",
264         "Validation Recall",
265     ],
266 )
267
268 print("\nC value comparison results:")
269 print(C_results_df)
270
271
272
273 plt.figure(figsize=(7, 4))
274 plt.plot(C_results_df["C"], C_results_df["Training Accuracy"],
275          marker="o", color="blue", label="Training Set")
276 plt.plot(C_results_df["C"], C_results_df["Validation Accuracy"],
277          marker="o", color="red", label="Validation Set")
278 plt.title("Accuracy Scores of Training Set and Validation Set by C Value")
279 plt.xlabel("C Value")
280 plt.ylabel("Accuracy Score")
281 plt.ylim(0.65, 0.75)
282 plt.legend()

```

A.3 Andy Wang: Logistic Regression Analysis

```
283 plt.savefig("figure_validation_accuracy_vs_c.png", dpi=300, bbox_inches="tight")
284 plt.show()
285
286 best_C = C_results_df.sort_values("Validation Accuracy", ascending=False).iloc[0]["C"]
287
288 best_model = SVC(C=best_C, kernel=best_kernel, gamma=0.005, class_weight="balanced")
289 best_model.fit(X_train_scaled, y_train)
290
291 print("\nBest kernel:", best_kernel)
292 print("Best C:", best_C)
293
294
295
296
297 train_pred = best_model.predict(X_train_scaled)
298 valid_pred = best_model.predict(X_valid_scaled)
299 test_pred = best_model.predict(X_test_scaled)
300
301 training_accuracy = accuracy_score(y_train, train_pred)
302 validation_accuracy = accuracy_score(y_valid, valid_pred)
303 test_accuracy = accuracy_score(y_test, test_pred)
304 test_precision = precision_score(y_test, test_pred, zero_division=0)
305 test_recall = recall_score(y_test, test_pred, zero_division=0)
306 test_cm = confusion_matrix(y_test, test_pred)
307
308 final_results = pd.DataFrame({
309     "Metric": [
310         "Training Accuracy",
311         "Validation Accuracy",
312         "Final Test Accuracy",
313         "Final Test Precision",
314         "Final Test Recall",
315     ],
316     "Score": [
317         training_accuracy,
318         validation_accuracy,
319         test_accuracy,
320         test_precision,
321         test_recall,
322     ],
323 })
324
325 print("\nFinal model results:")
326 print(final_results)
327
328 print("\nConfusion Matrix for the final model on the test set:")
329 print(test_cm)
```

```
330
331
332 plt.figure(figsize=(6, 5))
333 plt.imshow(test_cm, cmap="GnBu")
334
335 plt.title("Final Linear SVM: Test Confusion Matrix")
336 plt.xlabel("Predicted label")
337 plt.ylabel("True label")
338
339 class_labels = ["<=50K", ">50K"]
340 plt.xticks([0, 1], class_labels)
341 plt.yticks([0, 1], class_labels)
342
343 plt.colorbar()
344
345 for i in range(2):
346     for j in range(2):
347         value = test_cm[i, j]
348         text_colour = "white" if value > test_cm.max() / 2 else "black"
349         plt.text(j, i, value, ha="center", va="center", color=text_colour, fontsize=13)
350
351 plt.savefig("Figure_final_confusion_matrix.png", dpi=300, bbox_inches="tight")
352 plt.show()
353
```

```
~
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    ConfusionMatrixDisplay
)

columns = [
    "age", "workclass", "fnlwt", "education", "education-num",
    "marital-status", "occupation", "relationship", "race", "sex",
    "capital-gain", "capital-loss", "hours-per-week", "native-country",
    "income"
]

data = pd.read_csv(
    "adult.data",
    names=columns,
    na_values="?",
    skipinitialspace=True
)

data = data.dropna().copy()

data["income"] = (
    data["income"]
    .astype(str)
    .str.strip()
    .str.rstrip(".")
)

data["income_binary"] = (
    data["income"] == ">50K"
).astype(int)

feature_sets = [
    "Model 1": ["education-num"],
    "Model 2": ["education-num", "age"],
    "Model 3": ["education-num", "age", "hours-per-week"]
]
```

```

split_settings = [
    {"split_name": "60:40", "validation_size": 0.40, "figure_file": "Figure 1.png"},
    {"split_name": "70:30", "validation_size": 0.30, "figure_file": "Figure 2.png"},
    {"split_name": "80:20", "validation_size": 0.20, "figure_file": "Figure 3.png"}
]

all_split_results = []

for split_info in split_settings:

    training_set, validation_set = train_test_split(
        data,
        test_size=split_info["validation_size"],
        random_state=42,
        stratify=data["income_binary"]
    )

    model_names = []
    training_accuracies = []
    validation_accuracies = []

    for model_name, selected_features in feature_sets.items():

        X_train = training_set[selected_features]
        y_train = training_set["income_binary"]

        X_validation = validation_set[selected_features]
        y_validation = validation_set["income_binary"]

        model = LogisticRegression(max_iter=2000)
        model.fit(X_train, y_train)

        training_prediction = model.predict(X_train)
        validation_prediction = model.predict(X_validation)

        training_accuracy = accuracy_score(y_train, training_prediction)
        validation_accuracy = accuracy_score(y_validation, validation_prediction)

        model_names.append(model_name)
        training_accuracies.append(training_accuracy)
        validation_accuracies.append(validation_accuracy)

    all_split_results.append({
        "Split": split_info["split_name"],
        "Model": model_name,
        "Training Accuracy": training_accuracy,
        "Validation Accuracy": validation_accuracy,
        "Absolute Difference": abs(training_accuracy - validation_accuracy)
    })

```

```

detailed_split_results = pd.DataFrame(all_split_results)

summary_table = detailed_split_results.groupby("Split").agg(
    Average_Training_Accuracy=("Training Accuracy", "mean"),
    Average_Validation_Accuracy=("Validation Accuracy", "mean"),
    Average_Absolute_Difference=("Absolute Difference", "mean")
).reset_index()

split_order = ["60:40", "70:30", "80:20"]

summary_table = (
    summary_table
    .set_index("Split")
    .reindex(split_order)
    .reset_index()
)

summary_table = summary_table.round(4)

display_summary_table = summary_table.copy()

display_summary_table.columns = [
    "Split",
    "Average Training Accuracy",
    "Average Validation Accuracy",
    "Average Absolute Difference"
]

for column in display_summary_table.columns[1:]:
    display_summary_table[column] = display_summary_table[column].map(
        lambda value: f"{value:.4f}"
    )

figure, axis = plt.subplots(figsize=(8.5, 2.2))

axis.axis("off")

table = axis.table(
    cellText=display_summary_table.values,
    colLabels=display_summary_table.columns,
    cellLoc="center",
    colLoc="center",
    loc="center"
)

table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1.1, 1.8)

axis.set_title(
    "Average Training and Validation Accuracy for Different Split Ratios",
    pad=12
)

plt.tight_layout()

plt.savefig(
    "Figure 4.png",
    dpi=300,
    bbox_inches="tight"
)

```

```

plt.figure(figsize=(7, 5))

plt.plot(
    model_names,
    training_accuracies,
    marker="o",
    linewidth=2,
    label="Training Accuracy"
)

plt.plot(
    model_names,
    validation_accuracies,
    marker="s",
    linewidth=2,
    linestyle="--",
    label="Validation Accuracy"
)

plt.title(
    f"Training and Validation Accuracy ({split_info['split_name']} Split)"
)

plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.ylim(0.750, 0.800)
plt.grid(True, alpha=0.3)
plt.legend()

plt.tight_layout()

plt.savefig(
    split_info["figure_file"],
    dpi=300,
    bbox_inches="tight"
)

plt.show()

```

```

training_set_80, validation_set_80 = train_test_split(
    data,
    test_size=0.20,
    random_state=42,
    stratify=data["income_binary"]
)

validation_metrics_results = []

for model_name, selected_features in feature_sets.items():

    X_train = training_set_80[selected_features]
    y_train = training_set_80["income_binary"]

    X_validation = validation_set_80[selected_features]
    y_validation = validation_set_80["income_binary"]

    model = LogisticRegression(max_iter=2000)
    model.fit(X_train, y_train)

    validation_prediction = model.predict(X_validation)

    validation_metrics_results.append({
        "Model": model_name,
        "Accuracy": accuracy_score(y_validation, validation_prediction),
        "Precision": precision_score(
            y_validation,
            validation_prediction,
            zero_division=0
        ),
        "Recall": recall_score(
            y_validation,
            validation_prediction,
            zero_division=0
        ),
        "F1-score": f1_score(
            y_validation,
            validation_prediction,
            zero_division=0
        )
    })

validation_metrics_table = pd.DataFrame(validation_metrics_results)

metric_names = [
    "Accuracy",
    "Precision",
    "Recall",
    "F1-score"
]

x = np.arange(len(validation_metrics_table["Model"]))
bar_width = 0.18

plt.figure(figsize=(9, 5.5))

for metric_index, metric_name in enumerate(metric_names):

    plt.bar(
        x + metric_index * bar_width,
        validation_metrics_table[metric_name],
        width=bar_width,
        label=metric_name
    )

```

```

test_data = pd.read_csv(
    "adult.test",
    names=columns,
    skiprows=1,
    na_values="",
    skipinitialspace=True
)

test_data = test_data.dropna().copy()

test_data["income"] = (
    test_data["income"]
    .astype(str)
    .str.strip()
    .str.rstrip(".")
)

test_data["income_binary"] = (
    test_data["income"] == ">50K"
).astype(int)

final_features = [
    "education-num",
    "age",
    "hours-per-week"
]

X_train_final = data[final_features]
y_train_final = data["income_binary"]

X_test_final = test_data[final_features]
y_test_final = test_data["income_binary"]

final_model = LogisticRegression(max_iter=2000)

final_model.fit(X_train_final, y_train_final)

final_prediction = final_model.predict(X_test_final)

test_accuracy = accuracy_score(y_test_final, final_prediction)

test_precision = precision_score(
    y_test_final,
    final_prediction,
    zero_division=0
)

test_recall = recall_score(
    y_test_final,
    final_prediction,
    zero_division=0
)

test_f1 = f1_score(
    y_test_final,
    final_prediction,
    zero_division=0
)

final_metrics_table = pd.DataFrame({
    "Model": ["Model 3"],

```

```

plt.title(
    "Validation Performance of Logistic Regression Models (80:20 Split)"
)

plt.xlabel("Model")
plt.ylabel("Score")

plt.xticks(
    x + bar_width * 1.5,
    validation_metrics_table["Model"]
)

plt.ylim(0, 1.0)
plt.grid(axis="y", alpha=0.3)
plt.legend()

plt.tight_layout()

plt.savefig(
    "Figure 5.png",
    dpi=300,
    bbox_inches="tight"
)

plt.show()

```

```

final_metrics_table = pd.DataFrame({
    "Model": ["Model 3"],
    "Accuracy": [test_accuracy],
    "Precision": [test_precision],
    "Recall": [test_recall],
    "F1-score": [test_f1]
})

cm = confusion_matrix(
    y_test_final,
    final_prediction
)

figure, axis = plt.subplots(figsize=(6, 5))

display = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=["<=50K", ">50K"]
)

display.plot(
    ax=axis,
    values_format="d"
)

axis.set_title(
    "Confusion Matrix of Model 3 on adult.test"
)

plt.tight_layout()

plt.savefig(
    "Figure 7.png",
    dpi=300,
    bbox_inches="tight"
)

```

A.4 Nicholas Elton: Random Forest Analysis

```

# 1. Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier

# 2. Load adult dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
adult = pd.read_csv(url)

# 3. Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    adult.drop("income", axis=1),
    adult["income"],
    test_size=0.2,
    random_state=42
)

# 4. Create Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=5
)

# 5. Train the model
rf_model.fit(X_train, y_train)

# 6. Make predictions
y_pred = rf_model.predict(X_test)

# 7. Evaluate model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))

# 8. Classification report
print(classification_report(y_test, y_pred, target_names=["<=50K", ">50K"]))

```

```

# 9. Feature importance
feature_importance = rf_model.feature_importances_

# 10. Plot top 10 feature importances
top_features = feature_importance.argsort()[::-1]
top_features = feature_importance[top_features]
plt.bar(range(10), top_features)
plt.xticks(range(10), adult.columns[top_features])
plt.title("Top 10 Feature Importances")
plt.show()

# 11. Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Confusion Matrix")
plt.show()

# 12. ROC curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=2)
plt.plot([0, 1], [0, 1], lw=1)
plt.title("ROC Curve")
plt.show()

# 13. Final confusion table
results_table = pd.DataFrame({
    "Model": "Random Forest",
    "Accuracy": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1-score": f1
})

```

```

# 14. Check class balance
class_counts = adult.groupby("income").size()
print(class_counts)

# 15. Check for missing values
adult.isnull().sum()

# 16. Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    adult.drop("income", axis=1),
    adult["income"],
    test_size=0.2,
    random_state=42
)

# 17. Create Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=5
)

# 18. Train the model
rf_model.fit(X_train, y_train)

# 19. Make predictions
y_pred = rf_model.predict(X_test)

# 20. Evaluate model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))

# 21. Classification report
print(classification_report(y_test, y_pred, target_names=["<=50K", ">50K"]))

```

```

# 22. Feature importance
feature_importance = rf_model.feature_importances_

# 23. Plot top 10 feature importances
top_features = feature_importance.argsort()[::-1]
top_features = feature_importance[top_features]
plt.bar(range(10), top_features)
plt.xticks(range(10), adult.columns[top_features])
plt.title("Top 10 Feature Importances")
plt.show()

# 24. Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Confusion Matrix")
plt.show()

# 25. ROC curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=2)
plt.plot([0, 1], [0, 1], lw=1)
plt.title("ROC Curve")
plt.show()

# 26. Final confusion table
results_table = pd.DataFrame({
    "Model": "Random Forest",
    "Accuracy": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1-score": f1
})

```

References

- [1] D. Becker and R. Kohavi, "Adult," UCI Machine Learning Repository, 1996. [Online]. Available: <https://archive.ics.uci.edu/dataset/2/adult>
- [2] Bureau, US Census. "Income, Poverty, and Valuation of Noncash Benefits: 1994." Census.Gov, Apr. 1996, <https://www.census.gov/library/publications/1996/demo/p60-189.html>. Accessed 20 June 2026.
- [3] Haibo He, and E. A. Garcia. "Learning from Imbalanced Data." IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, Institute of Electrical and Electronics Engineers (IEEE), Sept. 2009, pp. 1263–84, <https://doi.org/10.1109/tkde.2008.239>. Accessed 21 June 2026.
- [4] P. Calvo, I. Lindenlaub, and A. Reynoso, "Marriage Market and Labor Market Sorting," *Review of Economic Studies*, vol. 91, no. 3, pp. 1230–1268, 2024. Available: <https://academic.oup.com/restud/article/91/3/1230/7227149>. Accessed: 21 June 2026.
- [5] F. Cingano, "Trends in Income Inequality and its Impact on Economic Growth," OECD Social, Employment and

Migration Working Papers, No. 163, OECD Publishing, Paris, 2014. Available: https://www.oecd.org/en/publications/trends-in-income-inequality-and-its-impact-on-economic-growth_5jxrjncwv6j-en.html Accessed: 21 June 2026.

- [6] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd ed. Hoboken, NJ, USA: Wiley, 2013, doi: 10.1002/9781118548387.
- [7] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009, doi: 10.1016/j.ipm.2009.03.002.