

**SUMMER TRAINING PROJECT REPORT**

(Term June-July 2025)

## Income Category Prediction using Adult Census Data

Submitted by

**Kallakuri SSS Lakshmi Bharadwaj**

**Registration Number: 12318329**

**Course Code:** PETV79

Under the Guidance of

Prof. Mahipal Singh Papola

# Bonafide Certificate

Certified that this project report “Income Category Prediction using Adult Census Data“ is a Bonafide work of Kallakuri SSS Lakshmi Bharadwaj who carried out the project work under my supervision   
 SIGNAURE

Prof. Mahipal Singh Papola

Kallakuri SSS Lakshmi Bharadwaj

SIGNATURE

<<SIGNATURE OF THE HEAD OF THE DEPARTMENT >>

SIGNATURE

# Declaration

I, Kallakuri SSS Lakshim Bharadwaj, a student of Bachelor of Technology under CSE discipline at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is genuine.

Name: Kallakuri SSS Lakshmi Bharadwaj

# Table of Contents

1. Chapter 1: Introduction

* Objective of the Project

1. Chapter 2: Training Overview

* Tools & Technologies Used
* Areas Covered During Training

1. Chapter 3: Project Details

* Title of the Project
* Problem Definition
* Scope and Objectives
* System Requirements

1. Chapter 4: Implementation

* Tools Used
* Methodology

1. Chapter 5: Results and Discussion

* Output / Report
* Challenges Faced
* Learnings

1. Chapter 6: Conclusion

* Summary

# Chapter 1: Introduction

## Objective of the Project

The primary objective of this project was to create a machine learning model capable of predicting whether a person earns more than $50K per year based on various demographic features, including age, education, and occupation. This project involved: - Exploring different machine learning models such as Logistic Regression, Decision Tree, Random Forest, and XGBoost. - Comparing the performance of these models using key metrics like accuracy, precision, recall, and F1 score. - Optimizing the models using hyperparameter tuning techniques like GridSearchCV and RandomizedSearchCV. - Deploying the best model through a Streamlit application for real-time predictions.

# Chapter 2: Training Overview

## Tools & Technologies Used

* Programming Language: Python
* Libraries: Pandas, Numpy, Scikit-learn, XGBoost, Seaborn, Matplotlib, SHAP, Streamlit

• Pandas: I used this library for data manipulation, like reading the dataset and handling missing values. Specifically, I replaced any "?" values with NaN and dropped rows that had missing data.

• NumPy: This library helps with numerical operations, and I used it here mainly to handle missing values in the dataset.

• Scikit-learn: This is a core library for machine learning that I used in multiple ways:

* + train\_test\_split: I used this to split the data into training and testing sets, ensuring that the model is trained and tested on separate data.
  + RandomizedSearchCV and GridSearchCV: These are hyperparameter tuning methods. I used them to search for the best parameters for my models to improve their performance.
  + StandardScaler: This was used to standardize the features in the dataset, which helps improve the model’s performance.
  + Pipeline: I created a pipeline to combine both the preprocessing steps (like scaling and imputing) and the classifier into a single workflow.
  + Imputer: This tool helped me handle any missing values in the dataset before training the model.
  + XGBoost: This is a gradient boosting algorithm, and I used it to train a classification model. XGBoost is known for its speed and performance, especially with structured/tabular data like the one I’m working with.
  + Joblib: After training the model, I used Joblib to save the entire pipeline, which allows me to reload and reuse the trained model without retraining it every time.
  + Matplotlib and Seaborn: These libraries helped me visualize the results. I used Matplotlib for plotting the feature importance of the XGBoost model and Seaborn for comparing the performance of different models in terms of accuracy, precision, recall, and F1-score.
  + SHAP (SHapley Additive exPlanations): I used SHAP to explain the predictions made by the XGBoost model. It gives insight into which features have the most impact on the model’s decisions.

## Areas Covered During Training

Data Cleaning and Preprocessing, Pipelining, Machine Learning Model Building, Hyperparameter Tuning, Model Evaluation, Deployment

# Chapter 3: Project Details

## Title of the Project

Income Prediction Using Machine Learning Models

## Problem Definition

The task at hand is to predict whether a person’s income exceeds $50K per year based on census data.

## Scope and Objectives

Scope: The project aims to explore machine learning algorithms for predicting income using various demographic features. Objectives: Build, optimize, and compare machine learning models. Deploy the final model using Streamlit.

## System Requirements

Hardware: Minimum 4 GB RAM, ideally 8 GB for better performance during training.  
Software: Python 3.7 or higher, Libraries: Pandas, Numpy, Scikit-learn, XGBoost, SHAP, Streamlit, Matplotlib, Seaborn. IDE: Jupyter Notebook for development and Streamlit for deployment.

# Chapter 4: Implementation

## Tools Used

Jupyter Notebook, Streamlit, XGBoost, Scikit-learn

## Methodology

- Data Loading and Preprocessing:

- I started by loading the dataset using Pandas to work with it more efficiently.

- The dataset had some missing values represented by "?", so I used NumPy to replace those with NaN and then dropped any rows with missing values.

- For the target variable income, I mapped the categorical values (<=50K to 0 and >50K to 1) to make them suitable for model training.

- I then selected a few numerical features (age, education.num, capital.gain, capital.loss, hours.per.week) that seemed relevant to predicting income.

- Splitting the Data:

- I used train\_test\_split from Scikit-learn to split the data into training and testing sets. I made sure that the distribution of the target variable was maintained across both sets by using stratification.

- Pipelining for Data Preprocessing and Modeling:

- I created a Pipeline to streamline both the preprocessing steps and the model training in one cohesive workflow.

- In the preprocessing step, I used SimpleImputer to fill in missing values with the median and StandardScaler to standardize the features.

- For the modeling step, I included the classifiers: XGBoost, Logistic Regression, Decision Tree, and Random Forest.

- The great thing about the Pipeline is that it ensures the same preprocessing steps are applied to both the training and test data, which eliminates any risk of data leakage.

- Modeling:

- After preprocessing, I trained several models:

- XGBoost because it’s known for its performance on tabular data.

- Logistic Regression, Decision Tree, and Random Forest as baseline models.

- To get the best performance from each model, I tuned the hyperparameters using RandomizedSearchCV and GridSearchCV. These helped me find the optimal parameters, and I did all of this within the Pipeline for seamless integration.

- Model Evaluation:

- After training the models, I evaluated them using metrics like accuracy, confusion matrix, and classification report. These gave me an idea of how well the models were performing, especially in terms of precision, recall, and F1-score.

- I also created visualizations to compare the performance of all the models using Matplotlib and Seaborn, which helped me see how each model was doing briefly.

- Model Interpretation:

- Since XGBoost is a bit of a black-box model, I used SHAP to interpret the results. This helped me understand which features were most important in the model’s decision-making process, making the model’s predictions more transparent.

- Saving the Model:

- Once I was happy with the model's performance, I saved the entire Pipeline (including preprocessing steps and the trained model) using Joblib. This way, I could easily load the model later without needing to retrain it.

A screenshot of a computer

AI-generated content may be incorrect. Fig 1. System Architecture Diagram

A graph with blue bars

AI-generated content may be incorrect.

Fig 2. Checking which features are the most important

A graph of different colored bars

AI-generated content may be incorrect.

Fig 3. Comparing the performance of all the models I built

A graph with different colored lines

AI-generated content may be incorrect.

Fig 4. SHAP analysis to see which input impacts the most on the output

# Chapter 5: Results and Discussion

## Output / Report

The final XGBoost model achieved an accuracy of 87.4%, F1 score of 0.72, recall of 0.67, and precision of 0.79.  
  
A screenshot of a computer

AI-generated content may be incorrect.

Fig 5. XGBoost results

## A black background with white spots AI-generated content may be incorrect. Fig 6. Tuned XGBoost results

A screen shot of a computer

AI-generated content may be incorrect.

Fig 7. Logistic Regression Results

A black screen with white text

AI-generated content may be incorrect.

Fig 8. Decision Tree Results

A black screen with white text

AI-generated content may be incorrect. Fig 9. Random Forest Results

## Challenges Faced

1. Lack of Familiarity with Pipelines:

* Problem: Before starting this project, I wasn’t fully familiar with the concept of Pipelines in Scikit-learn. I didn’t initially realize how valuable they could be for automating preprocessing and model training steps together.
* Solution: As I progressed, I learned how Pipelines could streamline the entire workflow, ensuring that transformations (like scaling and imputation) were applied consistently to both training and test data, preventing data leakage. Once I integrated Pipelines, it made my workflow much cleaner and more efficient.

1. Model Overfitting:

* Problem: Some models, particularly the Decision Tree, showed signs of overfitting. It performed well on the training set but struggled to generalize on the test data.
* Solution: I tackled this by using cross-validation and adjusting the model’s hyperparameters, such as max\_depth, to control complexity and avoid overfitting. I also used ensemble methods like Random Forest for better generalization.

1. Dealing with Class Imbalance:

* Problem: The dataset was imbalanced with respect to the target variable (income), which made it difficult for some models to predict the minority class (>50K) effectively.
* Solution: I addressed this by using class weighting in models like Logistic Regression and Random Forest to penalize misclassifications of the minority class. This helped improve the model's performance on the underrepresented class, though further balancing methods could have been explored.

## Chapter 6: Conclusion

## Summary

This project involved building a machine learning model to predict income based on demographic data, optimizing various models, and deploying the final model using Streamlit for real-time predictions.