CPE393 Final Project:
US Adult Census Income Prediction with Three-Models Classification Within Machine Learning Operation Process

Presented By

65070503410	Charunthon	Limseelo	*Leader/Developer*
65070503457	Thanakit	Chokbunsuwan	*Developer*
65070503469	Sawitt	Ngamvilaisiriwong	*Developer*
67540460063	Romain	Pierre Jean Blanchot	*Co-leader/Developer*
67540460066	Rayan	Khalil	*Document Contributor*

Submitted To

Dr. Aye Hninn Khine

Asst. Prof. Dr. Santitham Prom-on

This assignment is one of the assessment activities of the CPE393 Special Topic III: Machine Learning Operations Course of Computer Engineering Department (CPE), Semester 2/2024. Faculty of Engineering (FoE), King Mongkut's University of Technology Thonburi

Preface

This report, titled "US Adult Census Income Prediction with Three-Models Classification Within Machine Learning Operation Process," was developed as part of the requirements for the CPE393 Machine Learning Operations Assessment Activity. The project represents the culmination of our learning throughout the semester, integrating core concepts in strategic planning, customer analysis, team collaboration, and entrepreneurial thinking.

Our group focused on developing and deploying a robust MLOps pipeline for predicting the income level of adults in the United States based on data from the UCI Machine Learning Repository's Adult Census Income dataset. We applied management frameworks and leadership principles to create a classification system for predicting whether a person makes over $50K a year or not given their demographic variation. This was achieved by training and comparing the performance of two ensemble and one linear classification models: Gradient Boosting Classifier, Random Forest Classifier, and Logistic Regression respectively. By encompassing the entire machine learning lifecycle, from data acquisition and preprocessing to model training, evaluation, deployment, and continuous monitoring, our goal was to present a project that ensures reproducibility, scalability, and maintainability of the income prediction system.

We hope this report reflects our commitment to thoughtful analysis and creative problem-solving, and we sincerely thank our instructors, Asst. Prof. Dr. Santitham Prom-on and Professor Aye Hninn Khnine, for the guidance and feedback that helped shape our work.

TBA_MLOps Group Representatives

Table of Contents

Chapter 1
Abstract

This report outlines a project aimed at developing and deploying a robust Machine Learning Operations pipeline for predicting the income level of adults in the United States. The core problem addresses the need to classify whether an individual's income exceeds $50K per year based on various demographic factors.

The project utilizes the "Adult" dataset acquired from the UCI Machine Learning Repository. The dataset will undergo comprehensive preprocessing, including handling missing values, feature engineering, encoding, and scaling. For robust model evaluation, the dataset will be split into training, validation, and testing sets.

Two distinct classification models, Gradient Boosting and Random Forest, will be implemented and trained, with their hyperparameters optimized for best performance. Model evaluation will be conducted using a comprehensive suite of metrics, including Accuracy, Precision, Recall, F1-score, AUC-ROC curve, and Confusion matrix. The Machine Learning Operations pipeline will streamline the entire machine learning workflow, ensuring reproducibility and maintainability, with deployment strategies on containerization with Docker. The project aims to compare the performance of the chosen models to determine the most suitable one for deployment.

Keywords - US Adult Census, Income Prediction, Machine Learning Operations (MLOps), Classification, Gradient Boosting, Random Forest, Logistic Regression, Model Deployment, Continuous Monitoring, Data Preprocessing.

Chapter 2
Introduction

Income prediction has become an essential tool in socioeconomic analysis, financial planning, and policymaking. Understanding the factors that influence an individual's earning potential allows businesses, governments, and researchers to make informed decisions that drive economic growth and social equity. This project, "US Adult Census Income Prediction with Two-Models Classification (Gradient Boosting and Random Forest) and Monitoring," aims to develop a robust predictive framework using machine learning techniques to classify individuals based on their income levels.

The dataset used in this study originates from the 1994 US Census Bureau database, which has been widely utilized in machine learning research. It contains demographic and employment-related attributes, such as age, education, occupation, and work hours, which serve as key indicators in predicting whether an individual's annual income exceeds $50,000. By leveraging Gradient Boosting and Random Forest classification models, this project seeks to enhance prediction accuracy and provide insights into the most influential factors affecting income levels.

Additionally, this study incorporates a monitoring mechanism to assess model performance over time, ensuring adaptability to evolving economic conditions and demographic shifts. The integration of monitoring tools allows for continuous evaluation and refinement of the predictive models, making them more reliable for real-world applications.

Through this project, we aim to contribute to the broader discourse on income inequality, economic forecasting, and data-driven decision-making. The findings from this study can be instrumental in shaping policies, optimizing business strategies, and fostering a deeper understanding of income distribution patterns in the United States.

Chapter 3
Literature Review

3.1 Introduction to Machine Learning Classification Models

Machine learning classification models have transformed predictive analytics, offering powerful techniques for decision-making across various domains. Among the widely used ensemble learning methods, Gradient Boosting Classifier and Random Forest Classifier stand out for their accuracy and versatility. Additionally, Logistic Regression, a fundamental statistical model, remains relevant for classification tasks due to its simplicity and interpretability.

3.2 Gradient Boosting Classifier

Gradient Boosting is an ensemble method that builds models sequentially, optimizing weak learners to minimize errors. It applies boosting principles by training multiple decision trees iteratively, each correcting the mistakes of the previous one. The model assigns higher weights to misclassified instances, ensuring improved predictive accuracy. Research has demonstrated its superiority in structured data, particularly when compared to traditional classifiers. [1]

Advanced implementations such as **XGBoost, LightGBM, and CatBoost** have further enhanced computational efficiency and predictive power. These variants optimize memory usage, handling large datasets with greater speed and precision. [2]

3.3 Random Forest Classifier

Random Forest is another ensemble learning technique that constructs multiple decision trees and aggregates their predictions. Unlike Gradient Boosting, Random Forest builds trees independently by sampling subsets of features and data points, reducing overfitting while improving generalization.

Studies have shown Random Forest's effectiveness in handling high-dimensional datasets, making it a preferred choice in medical diagnostics, financial forecasting, and remote sensing applications. Its ability to provide stable and interpretable results makes it valuable in real-world scenarios. [1]

3.4 Logistic Regression

Logistic Regression is a widely used statistical model for binary classification problems. It operates by estimating probabilities using the logistic function, effectively predicting categorical outcomes based on input features. Unlike ensemble methods, Logistic

Regression does not rely on multiple models but rather a single predictive equation, making it computationally efficient and interpretable.

Despite its simplicity, Logistic Regression remains relevant in many applications, particularly when explainability is a priority. Studies have demonstrated its effectiveness in text classification, medical research, and fraud detection, often serving as a benchmark model in predictive tasks. [3]

3.5 Comparative Analysis

Comparative studies between Gradient Boosting, Random Forest, and Logistic Regression reveal distinct advantages based on dataset characteristics. Gradient Boosting tends to excel in highly structured data with complex patterns, optimizing predictive accuracy through iterative refinements. Random Forest is more resilient to noise, offering robust performance in diverse datasets. Meanwhile, Logistic Regression provides a transparent and interpretable approach, making it valuable in applications requiring direct statistical inference. [1] [2]

The choice among these models depends on factors such as computational efficiency, interpretability, and the nature of the problem being addressed.

3.6 Conclusion

Gradient Boosting, Random Forest, and Logistic Regression represent key approaches in machine learning classification. Their distinct advantages highlight the importance of selecting appropriate models based on data characteristics and application requirements. Future research may explore hybrid methodologies that integrate these techniques to enhance predictive accuracy and generalization.

Chapter 4
Methodology

4.1 Data Collection, Library and Environment Setup

The dataset was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). Predict whether income exceeds $50K/yr based on census data. Also known as adult dataset. Null values have been handled, and the data are not standardized, normalized, or scaled. There are 32561 entries and 15 columns/attributes in this dataset. The target for training the classification will be in the income column.

- Kohavi, R. (1996). Census Income [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5GP7S.
  (https://archive.ics.uci.edu/dataset/20/census+income) [4]

Variable Name	Role	Type	Demo graphic	Description	Units	Missing Values
age	Feature	Integer	Age	N/A		no
workclass	Feature	Categorical	Income	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.		yes
Final Weight	Feature	Integer				no
education	Feature	Categorical	Education Level	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.		no
Education-num	Feature	Integer	Education Level			no
Marital-status	Feature	Categorical	Other	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.		no
occupation	Feature	Categorical	Other	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.		yes

relationship	Feature	Categorical	Other	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.		no
race	Feature	Categorical	Race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.		no
sex	Feature	Binary	Sex	Female, Male.		no
capital-gain	Feature	Integer				no
capital-loss	Feature	Integer				no
hours-per-week	Feature	Integer				no
native-country	Feature	Categorical	Other	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad & Tobago, Peru, Hong, Holland-Netherlands.		yes
income	Target	Binary	Income	>50K, <=50K.		no

The project was conducted using the Anaconda Python environment. The primary libraries used in the study include Matplotlib for data visualization and Numpy for numerical computing in Python. Additionally, the study utilizes Scikit-learn (sklearn) for building and evaluating machine learning models, as Scikit-learn is employed for Random Forests Classifier (RF) and Gradient Boosting Classifier (GBM). This library supports binary classification tasks, which are essential for this project. These libraries collectively provide tools for data loading, data processing, model development, and performance assessment within the research workflow. Along with Flask for making a live local deployment using Application Programming Interface (API), and Evidently AI for monitoring the model quality and dataset drift.

Another set of tools that we need to implement for Machine Learning Operations for this project will include Apache Airflow as using Asst. Prof. Dr. Santitham Prom-on’ s server for running as checking the process of Directed Acyclic Graph (DAG) on data pipeline, Docker for model containerization, and MLFlow for model experiment tracking and models performance comparison.

4.2 Data Processing and Exploratory Data Analysis (EDA)

The data preprocessing stage is essential to ensure data quality and reliability in heart disease prediction. The entire process follows a structured pipeline, as depicted in Fig 1, and includes several key steps: Exploratory Data Analysis (EDA), missing value imputation, value correction, feature engineering, outlier checking, and encoding.

Data Description

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

Data Shape = (32561, 15)

data.isna().sum()

```
age                 0
workclass           0
fnlwgt              0
education           0
education.num       0
marital.status      0
occupation          0
relationship        0
race                0
sex                 0
capital.gain        0
capital.loss        0
hours.per.week      0
native.country      0
income              0

dtype: int64
```

Not a Number

```
for column in data.columns:
    print(f"{column} = {data[data[column] == '?'].shape[0]}")
```

```
age = 0
workclass = 1836
fnlwgt = 0
education = 0
```

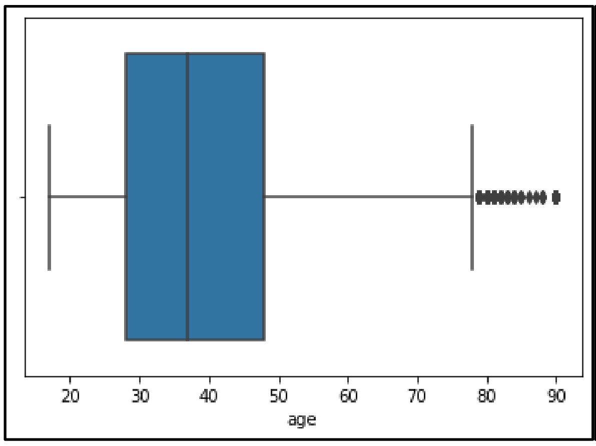
```
education.num = 0                       capital.gain = 0
marital.status = 0                      capital.loss = 0
occupation = 1843                       hours.per.week = 0
relationship = 0                        native.country = 583
race = 0                                income = 0
sex = 0
```

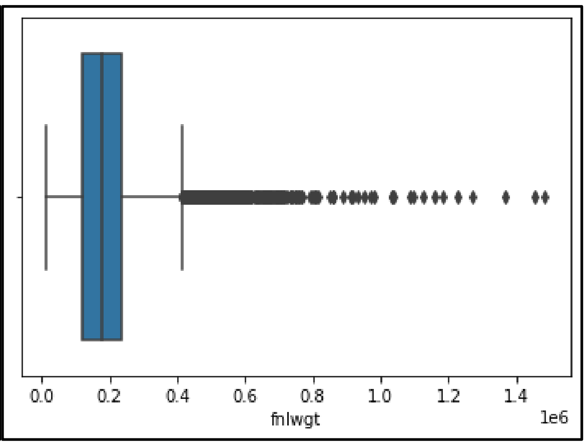
Slice out….

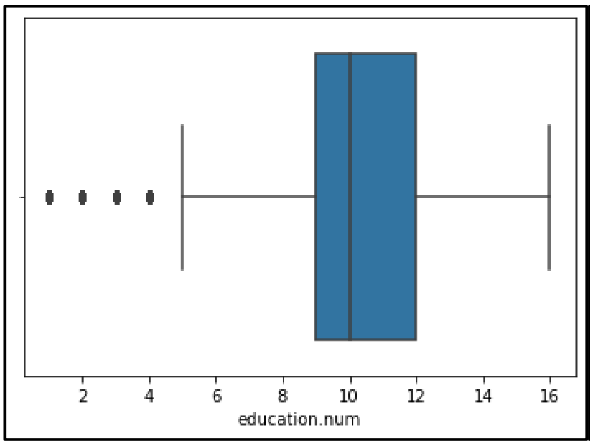
```
data["workclass"][data["workclass"] == "?"] = data["workclass"].mode()[0]
data["occupation"][data["occupation"] == "?"] = data["occupation"].mode()[0]
data["native.country"][data["native.country"] == "?"] = data["native.country"].mode()[0]
```

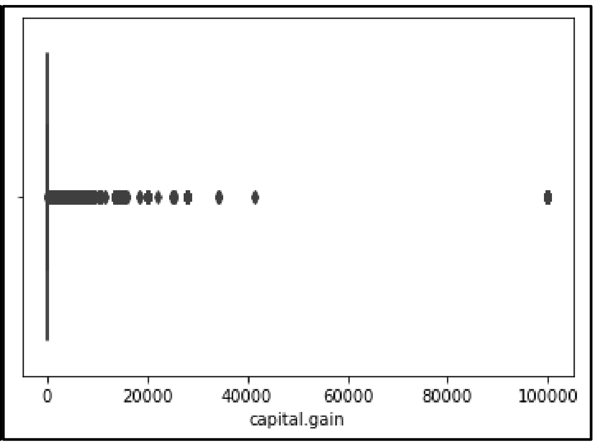
```
age = 0                                 race = 0
workclass = 0                           sex = 0
fnlwgt = 0                              capital.gain = 0
education = 0                           capital.loss = 0
education.num = 0                       hours.per.week = 0
marital.status = 0                      native.country = 0
occupation = 0                          income = 0
relationship = 0
```

Outliers









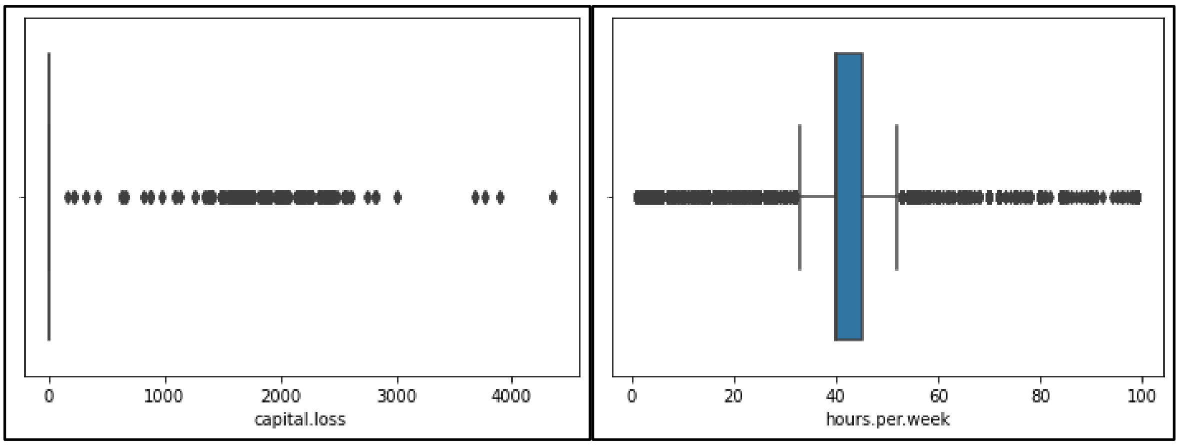


Fig1. Outlier Checking for the certain dataset

Visualization

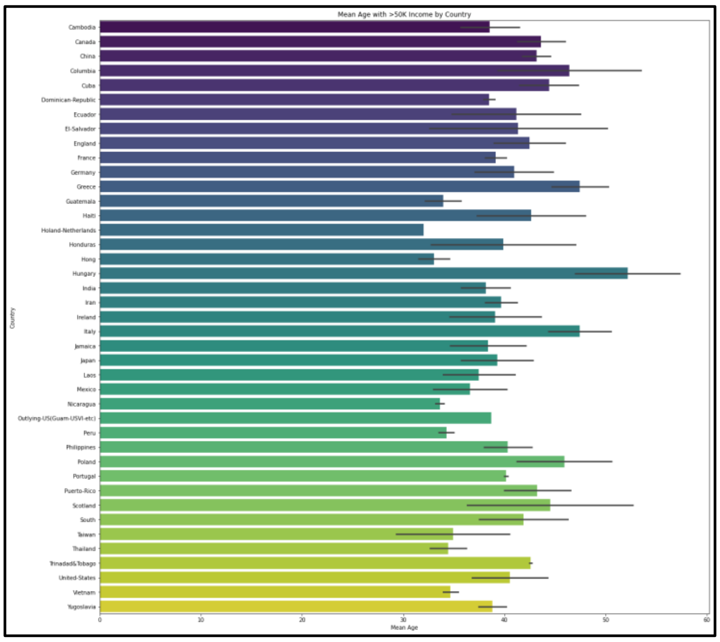


Fig2. Mean Age with >50K Income by Country

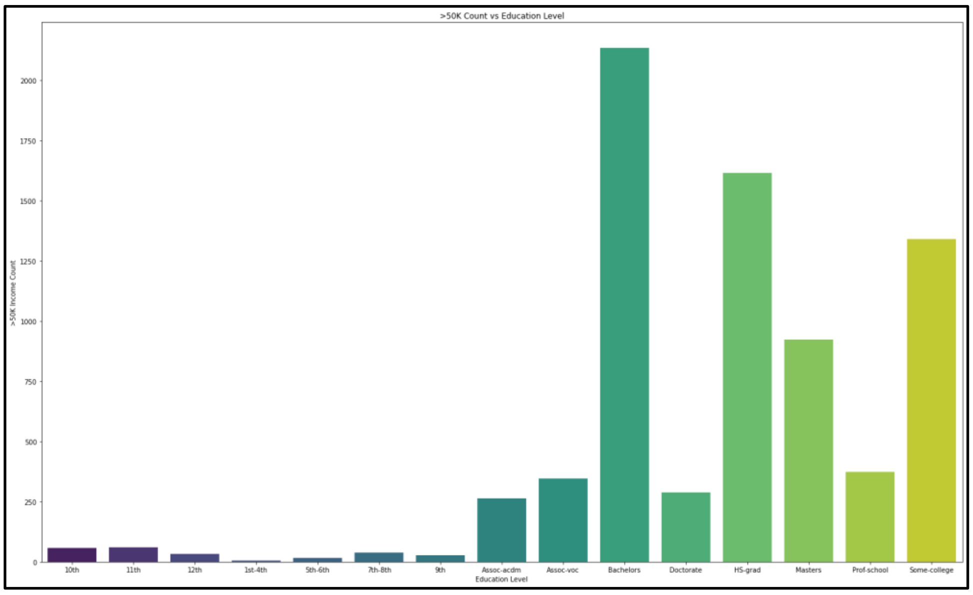


Fig3. >50K Count vs Education Level

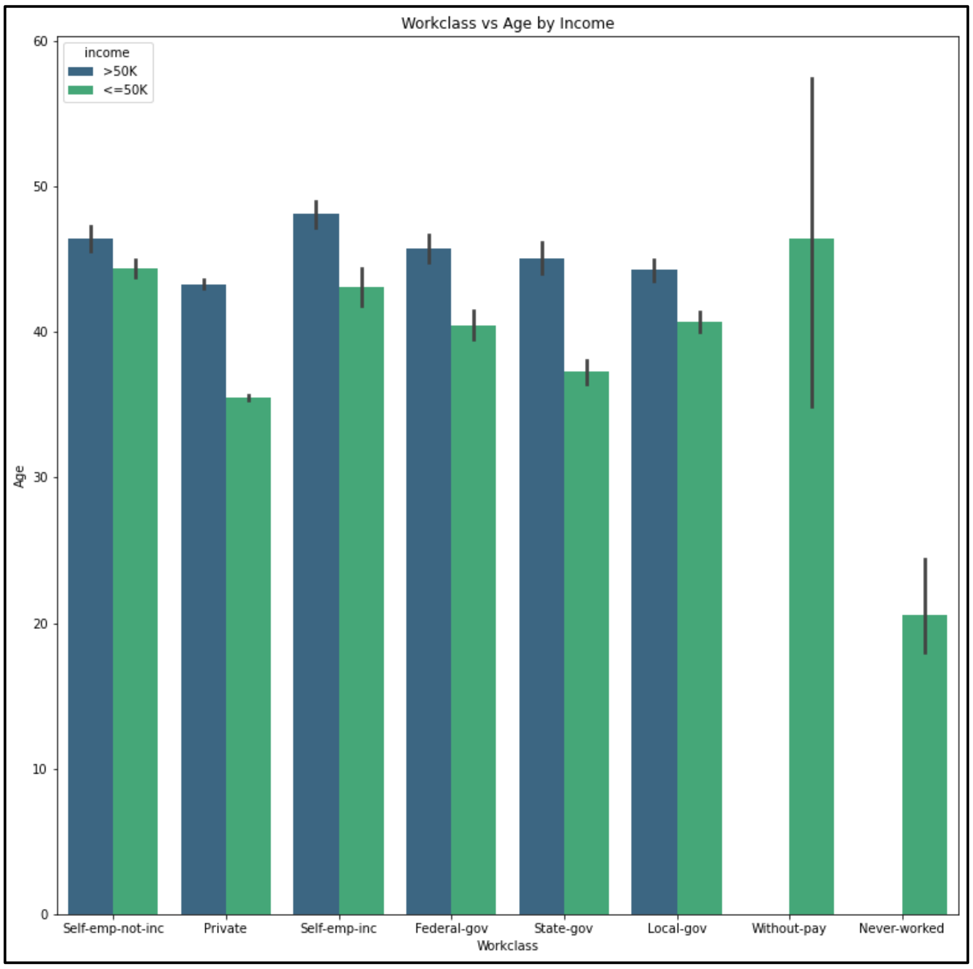


Fig4. Workclass vs Age by Income

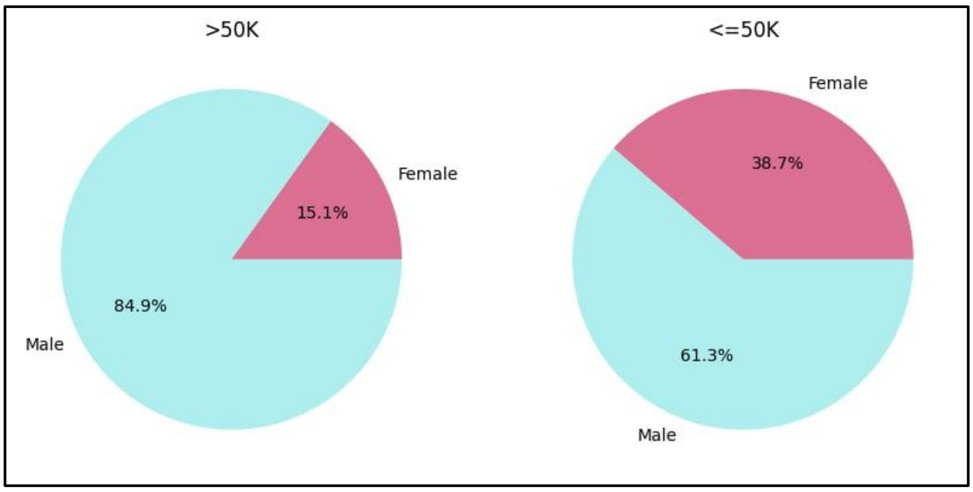


Fig5. Portion of Gender for those who are greater than 50K and less-than-or-equal-to 50K

4.3 Feature Engineering

Feature engineering plays a critical role in improving the performance of machine learning models by refining and transforming raw data into meaningful inputs. This process involves handling missing values, removing outliers, consolidating categorical variables, and applying scaling techniques to numerical features. In this project, the following steps have been implemented to enhance the dataset for effective classification.

Handling Missing Values

The dataset contains categorical variables with missing values represented by '?'. To address this issue, the most frequent value for each affected column (workclass, occupation, and native.country) is used for imputation. This ensures a consistent representation of categories and prevents data loss due to missing values.

Removing Outliers

Outliers in numeric features such as age, fnlwgt, education.num, capital.gain, capital.loss, and hours.per.week are detected using the **interquartile range (IQR)** method. Any values outside **1.5 times the IQR** are considered outliers and removed to maintain data integrity and improve model generalization.

Consolidating Categorical Variables

To simplify categorical features and enhance interpretability, similar categories are merged:

- **Education Levels**: High school-related education levels (HS-grad, 11th, 10th, 9th, 12th) are grouped under **HS-grad**, while elementary schooling levels (1st-4th, 5th-6th, 7th-8th) are categorized as **elementary_school**.
- **Marital Status**: Married-spouse-absent, Married-civ-spouse, and Married-AF-spouse are grouped as **Married**, while Separated and Divorced are consolidated as **Separated**.

- **Workclass Categories**: Self-emp-not-inc and Self-emp-inc are grouped as **Self_employed**, while government-related occupations (Local-gov, State-gov, Federal-gov) are categorized as **Govt_employees**.

Feature Transformation

Numerical features are normalized using the **MinMaxScaler**, ensuring they remain within a range suitable for machine learning models while preserving variance. Categorical features are encoded using **one-hot encoding**, with specific categories excluded to prevent redundancy.

Pipeline Implementation

To streamline preprocessing, transformation pipelines are created for numeric and categorical features. The numerical pipeline applies **scaling** and **selection**, while the categorical pipeline applies **dummy encoding** and **selection**. Both pipelines are then merged to construct the final dataset for model training.

Final Dataset Construction

After preprocessing, the dataset is split into features (X) and target labels (y), with unnecessary columns (id and fnlwgt) removed. The refined dataset ensures optimal input for the classification models, improving predictive accuracy and reliability.

This feature engineering process significantly enhances data quality, ensuring that the Gradient Boosting and Random Forest classifiers can extract meaningful patterns for income prediction.

4.4 Directed Acyclic Graph (DAG) Procedure

A **Directed Acyclic Graph (DAG)** is an essential framework for orchestrating data workflows, ensuring that tasks execute in a structured and logical sequence. In this project, **Apache Airflow** has been utilized to manage and automate the execution of various data preprocessing and model training steps.

DAG Overview

The DAG, named **"DAG_TBA"**, defines a series of interconnected tasks that flow from a start node to an end node. Each task represents a distinct process within the pipeline, ensuring the smooth execution of feature engineering, model training, and monitoring.

Task Breakdown

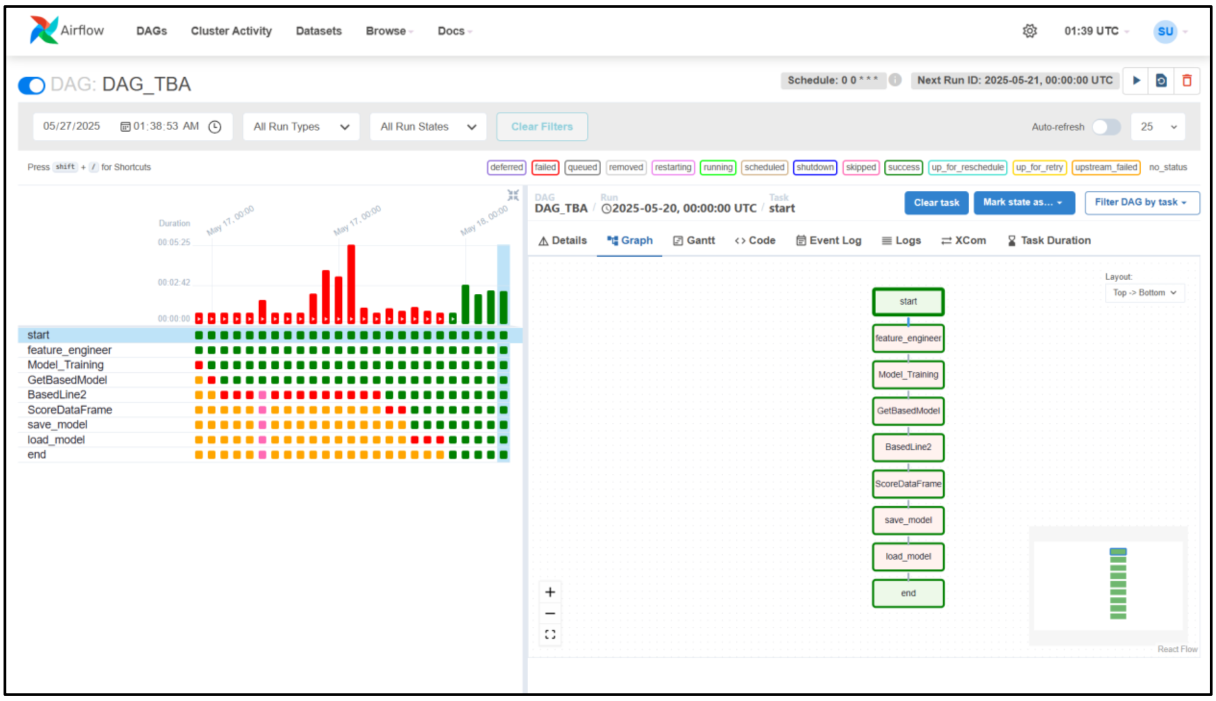


Fig6. Overall Directed Acyclic Graph (DAG) Procedure

The DAG consists of the following key tasks:

- **start** – Initiates the pipeline execution.
- **feature_engineer** – Performs feature engineering, including data cleaning, transformation, and preprocessing.
- **Model_Training** – Executes the training process for the machine learning models.
- **GetBasedModel** – Establishes a baseline model for comparison.
- **BasedLine2** – Generates an alternative baseline model.
- **ScoreDataFrame** – Evaluates the trained models based on defined metrics.
- **save_model** – Stores the trained model for future use.
- **load_model** – Retrieves the stored model for inference.
- **end** – Marks the completion of the pipeline execution.

DAG Execution & Monitoring

Apache Airflow provides a **graphical representation** of the DAG, allowing users to visually track the execution status of each task. Each task is color-coded based on its current state, such as **running, success, failed, skipped, or queued**, ensuring real-time monitoring of progress.

By implementing a DAG-based workflow, this project achieves efficient task scheduling, dependency resolution, and automated execution. The modularity of this approach enables adaptability to changes in dataset characteristics and model performance over time.

DAG Procedure Result

The execution of the DAG provides valuable insights into the data pipeline's efficiency and reliability. By monitoring the task duration graph, users can analyze performance bottlenecks and optimize task dependencies. Successful execution ensures that:

- Feature engineering is completed without missing values or outliers.
- Machine learning models are trained using refined datasets.
- Baseline models are compared to assess improvements in accuracy.
- Scoring and evaluation processes help determine model effectiveness.
- Trained models are properly saved and loaded for future predictions.

These results enable continuous improvements in the workflow, making the pipeline adaptable to changes in dataset characteristics and modeling strategies. Below is the result that we have captured from XCOM of the **GetBasedModel** procedure, which shows that the **LR (Logistic Regression)** get the highest value of the accuracy between three classification models, which we will be going to use this model to implement in the further section.

```
{4 items
 "best_model":{3 items
  "mean_score":0.831813654387287
  "name":"LR"
  "std_score":0.002675618439977783
 }
 "model_results":[3 items
  0:{4 items
   "mean_score":0.831813654387287
   "name":"LR"
   "scores":[3 items
    0:0.8291296625222024
    1:0.8354655294953802
    2:0.8308457711442786
   ]
   "std_score":0.002675618439977783
  }
  1:{4 items
   "mean_score":0.829800123967829
   "name":"GBM"
   "scores":[3 items
    0:0.8273534635879218
    1:0.8308457711442786
    2:0.8312011371712864
   ]
   "std_score":0.0017361224062535155
  }
  2:{4 items
```

```
      "mean_score":0.7953337852730765
      "name":"RF"
      "scores":[3 items
        0:0.7921847246891652
        1:0.7999289267945985
        2:0.7938877043354655
      ]
      "std_score":0.0033228029454102355
    }
  ]
  "names":[3 items
    0:"LR"
    1:"GBM"
    2:"RF"
  ]
  "results":[3 items
    0:[3 items
      0:0.8291296625222024
      1:0.8354655294953802
      2:0.8308457711442786
    ]
    1:[3 items
      0:0.8273534635879218
      1:0.8308457711442786
      2:0.8312011371712864
    ]
    2:[3 items
      0:0.7921847246891652
      1:0.7999289267945985
      2:0.7938877043354655
    ]
  ]
}
```

4.5 Model Training, Selection, and Optimization

Data Preparation

To ensure effective model training, the dataset is first split into **training** and **testing** sets. The prepare_train_test_data function applies the **train-test split** technique, allocating **85%** of the data for training and **15%** for testing, using a fixed **random seed** (random_state=42) for reproducibility.

Model Selection

In this project, three classification models are implemented to predict income levels based on the dataset features:

- **Logistic Regression (LR)** – A widely used statistical model for binary classification. It operates using the **liblinear solver**, with a regularization parameter (C=1.0) and an increased number of iterations (max_iter=1000) to ensure convergence.
- **Gradient Boosting Classifier (GBM)** – A powerful ensemble learning algorithm that builds decision trees sequentially, optimizing misclassified instances at each iteration. The model is trained using **100 estimators** and a **learning rate of 0.1**, ensuring a balance between performance and computational efficiency.
- **Random Forest Classifier (RF)** – A robust ensemble method that constructs multiple independent decision trees. It is configured with **100 estimators** and an unrestricted depth (max_depth=None), allowing the model to extract complex patterns from the dataset.

Model Optimization

Each model is carefully tuned to achieve optimal performance:

- **Hyperparameter Tuning** – The selected parameters (n_estimators, learning_rate, C, max_iter, etc.) are optimized based on empirical experimentation and prior research findings.
- **Cross-Validation** – Models undergo cross-validation during training to evaluate generalization and prevent overfitting.
- **Performance Metrics** – Standard evaluation metrics such as **accuracy, precision, recall, and F1-score** are used to assess the effectiveness of each model.

Selection Criteria

Model selection is based on comparative performance across key metrics. The final trained models are evaluated on the test dataset to identify the most suitable approach for income prediction. Depending on the results, further refinements may be considered, such as adjusting hyperparameters or experimenting with ensemble techniques.

This methodological approach ensures that the deployed model (Logistic Regression) is well-optimized and capable of delivering reliable predictions.

4.6 Model Evaluation

Evaluation Strategy

To assess the performance of the trained classification models, **cross-validation** is employed as the primary evaluation strategy. The evaluate_models function implements **Stratified K-Fold Cross-Validation** with **three folds**, ensuring that each fold maintains the same distribution of income classes. The evaluation metric used is **accuracy**, which provides a measure of how well each model predicts the correct income category.

Cross-Validation Process

Each model is trained and tested within multiple cross-validation iterations. The steps involved are:

1. The dataset is split into **three equal subsets** using **Stratified K-Fold**, maintaining class distribution.
2. Each model undergoes training and testing across different folds, allowing it to generalize better and avoid overfitting.
3. The mean accuracy and standard deviation of accuracy are computed across all validation folds.

Model Performance Comparison

The models evaluated include:

- **Logistic Regression (LR)** – Measures how well a linear model can classify individuals based on transformed features.
- **Gradient Boosting Classifier (GBM)** – Analyzes how well sequential tree-based learning enhances classification accuracy.
- **Random Forest Classifier (RF)** – Assesses the effectiveness of multiple independent decision trees in prediction robustness.

For each model, the **mean cross-validation score** and **standard deviation** are printed, allowing for direct comparison of stability and predictive capability.

Score Visualization

To structure the evaluation results efficiently, the create_score_dataframe function converts the list of model names and results into a **pandas DataFrame**. This table organizes the evaluation metrics, summarizing each model's average accuracy score for better interpretability.

Conclusion

The evaluation process ensures that model selection is grounded in **data-driven insights**, preventing overfitting and optimizing predictive performance. Models with higher mean accuracy and lower variance are prioritized for deployment. Future enhancements may include **hyperparameter tuning**, additional performance metrics such as **precision, recall, and F1-score**, and alternative validation methods.

4.7 Model Containerization in Pickle Format

Overview

Model containerization is an essential step in machine learning pipelines, ensuring trained models are stored efficiently and can be reused for inference. In this project, **Pickle (.pkl) format** is used to serialize and save models, allowing seamless model retrieval for evaluation and deployment.

Model Saving Procedure

The save_model function facilitates storing trained models using **Python's pickle library**. The process follows these key steps:

1. **Creating a Models Directory** – If the 'models' directory does not exist, it is automatically created to organize saved models.
2. **Generating a Filename with Timestamp** – To ensure version control, models are named dynamically based on their type (**Logistic Regression, Gradient Boosting, Random Forest**) and a **timestamp** (YYYYMMDD_HHMMSS).
3. **Serializing the Model** – Using pickle.dump(), the trained model is saved as a .pkl file for future retrieval.

Model Loading Procedure

The load_model function allows retrieving stored models by reading the respective .pkl file. It supports loading from a **default models directory** or a **custom path** if specified.

Model Training and Logging

The train_and_evaluate_model function integrates model training with **MLflow**, a tool for tracking machine learning experiments. It performs:

- **Training** – Models are trained on **X_train, y_train** datasets.
- **Prediction** – Test set predictions are generated.
- **Evaluation Metrics Logging** – Metrics such as **accuracy, precision, recall, F1-score, and AUC** are recorded.
- **Model Parameter Logging** – Hyperparameters are stored for reference.
- **Model Saving** – After training and evaluation, the final model is stored using the save_model function.

Advantages of Model Containerization

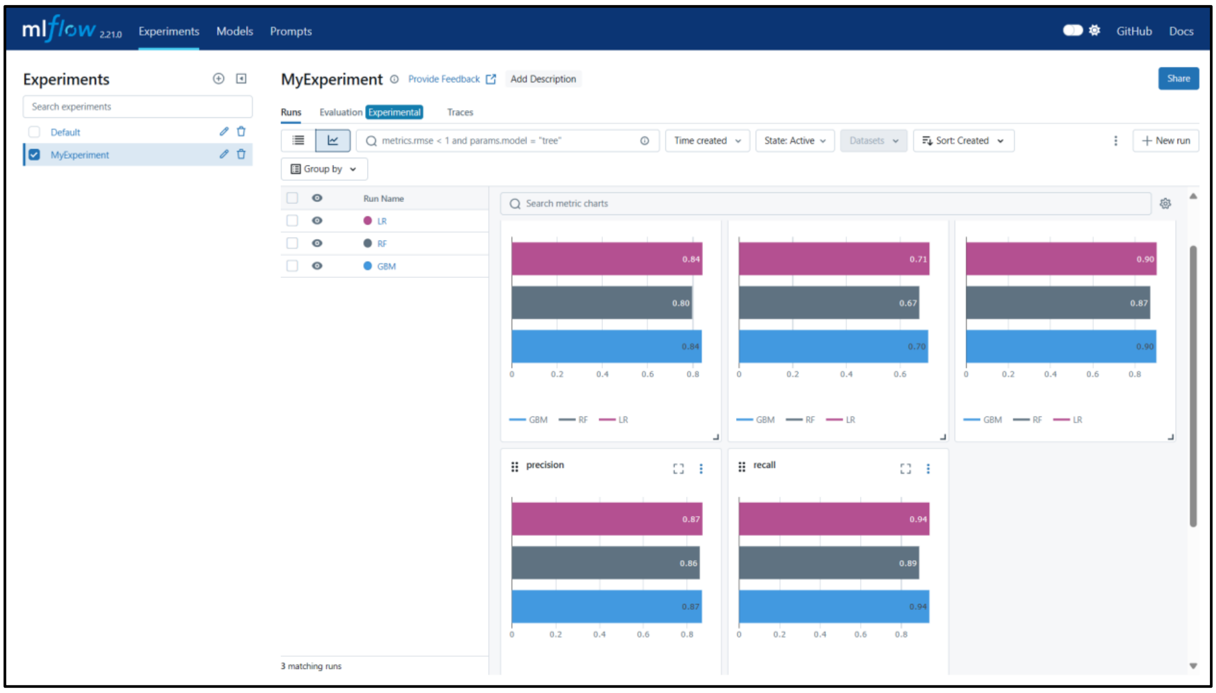
Storing models in **PKL format** provides:

- **Reusability** – Trained models can be loaded without retraining.
- **Version Control** – Models saved with timestamps allow for easy comparison.

- **Efficient Deployment** – Serialized models enable integration into production environments.

This structured approach ensures that trained models are efficiently managed, improving the reproducibility of the machine learning pipeline.

4.8 Model Experiment Tracking on ML-Flow



Overview

Tracking machine learning experiments is essential for evaluating model performance, comparing different approaches, and ensuring reproducibility. **MLflow**, an open-source platform for managing ML experiments, is used in this project to log model parameters, metrics, and artifacts systematically.

MLflow Experiment Setup

The experiment tracking process begins with defining an **MLflow experiment** named "MyExperiment". Within this experiment, multiple runs are logged, corresponding to different models:

- **LR (Logistic Regression)**
- **RF (Random Forest Classifier)**
- **GBM (Gradient Boosting Classifier)**

Each run records key metrics such as **precision, recall, and mean squared error (MSE)**, allowing for direct comparison of model performance.

Logging Model Parameters and Metrics

During training, MLflow logs essential parameters and evaluation metrics:

- **Hyperparameters** (e.g., learning rate, number of estimators)
- **Performance Metrics** (e.g., accuracy, precision, recall, F1-score)
- **Artifacts** (e.g., trained models, feature importance plots)

These logs enable researchers to analyze how different configurations impact model effectiveness.

Experiment Visualization

MLflow provides an interactive **UI dashboard** where users can filter and compare experiment runs. In this project, the interface displays bar charts for **precision and recall**, visually comparing the performance of the three models. The results indicate that **GBM and LR outperform RF** in these metrics, reinforcing the importance of selecting the right model for deployment.

Filtering and Model Selection

MLflow allows filtering runs based on specific conditions. In this project, the filter "metrics.mse < 1 and params.model = 'tree'" is applied to identify models with low error rates and tree-based architectures. This ensures that only the most promising models are considered for further optimization.

Conclusion

By leveraging MLflow for experiment tracking, this project ensures **structured model evaluation, reproducibility, and informed decision-making**. The ability to log, compare, and visualize model performance streamlines the selection process, ultimately leading to more effective machine learning applications.

4.9 Model Deployment with Docker

Overview

Containerization using **Docker** provides an efficient way to package, deploy, and manage machine learning models. This approach ensures that all dependencies, configurations, and models are encapsulated within a **self-contained environment**, eliminating compatibility issues across different computing setups.

In this project, **Docker** is used to deploy the **Random Forest model** for real-time income prediction. The containerized deployment streamlines execution by maintaining **a reproducible development environment**, allowing for seamless application portability across different platforms.

Dockerfile Structure

The deployment pipeline is structured within a **Dockerfile**, ensuring consistency across development and production environments. The key components include:

1. **Base Image Selection**
   - The Docker container is built on **Miniconda3**, a lightweight version of Conda that simplifies package management and dependency handling.
2. **Setting Up the Working Directory**
   - The working directory is defined as /app to structure all essential files within the container.
3. **Environment Configuration**
   - environment.yml is copied into the container, and a new Conda environment named "mlops-project" is created using this configuration.
   - This ensures that all necessary dependencies for model execution are installed.
4. **Shell Configuration**
   - The Conda environment is activated as the default shell for executing subsequent commands, ensuring that processes are run within the correct Python environment.
5. **Copying Model Files and Application Code**
   - The key components (app.py, RF_model_prod.pkl, and features.json) are copied into the container.
   - app.py contains the **Flask-based API**, enabling users to send input data and receive predictions from the model.
   - RF_model_prod.pkl stores the trained **Random Forest model**, which is loaded and used for inference.
   - features.json defines the model's input features to standardize API requests.
6. **Exposing the Application Port**
   - **Port 5000** is exposed to allow external access to the deployed Flask application.
7. **Running the Application**
   - The final command initializes the Flask server inside the Conda environment and runs the prediction service.

Advantages of Docker Deployment

Deploying the model within a **Docker container** provides several benefits:

- **Reproducibility** – Ensures a consistent runtime environment across development, testing, and production.
- **Portability** – Can be deployed on **local machines, cloud servers, or edge devices** without modification.

- **Dependency Management** – Eliminates system-level conflicts by encapsulating dependencies within the container.
- **Scalability** – Can be integrated with **Kubernetes** or cloud services for large-scale inference operations.

Future Enhancements

While Docker ensures efficient deployment, future iterations may explore **cloud deployment** using **AWS, Azure, or Google Cloud**, integrating **CI/CD pipelines** for automated updates and implementing **load-balancing strategies** for high-traffic prediction requests.

4.10 Model Monitoring for Dataset Drift and Model Quality

Overview

Model monitoring is a crucial step in ensuring that machine learning models maintain their predictive accuracy and reliability over time. This process involves tracking **dataset drift** and **model quality**, identifying changes in data distribution, and assessing model stability. In this project, **Evidently AI** is used to automate monitoring and detect potential issues in the dataset and model performance.

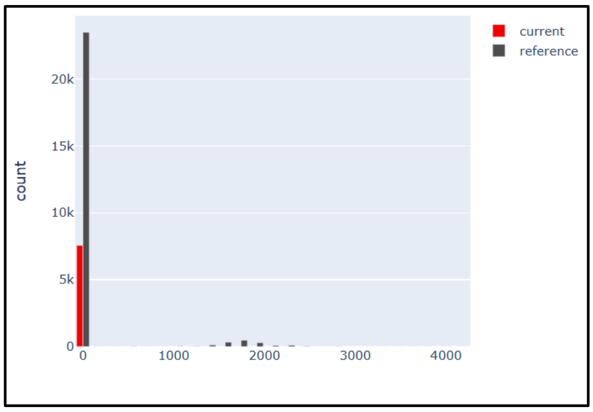
Dataset Drift Detection

Dataset drift occurs when the statistical properties of the input data change over time, leading to degraded model performance. To monitor dataset drift:

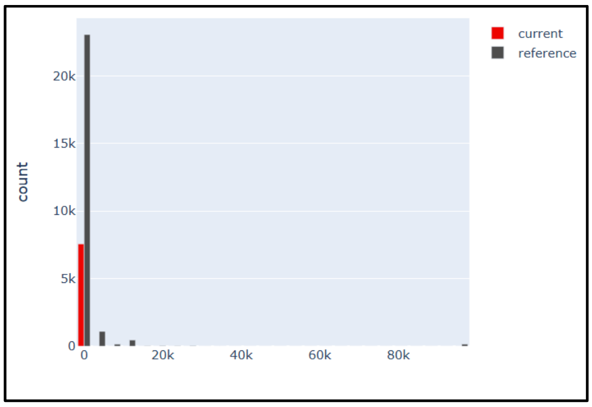
1. **Reference and Current Data Splitting** – The dataset is divided into **reference data** (first 25,000 samples) and **current data** (remaining samples).
2. **Z-score Analysis** – Outliers in numerical features are detected using **Z-score thresholding**, ensuring that extreme values do not distort model predictions.
3. **Data Drift Test Suite** – The DataDriftTestPreset from Evidently AI is applied to compare distributions between reference and current datasets.
4. **Automated Drift Detection** – If significant drift is detected, an HTML report is generated for further analysis.

For the result that we have tested, there are 16 tests in total for making Data Drift Suite, which 14 tests are success and 2 are failed, both of them are Drift per Column:

1. The drift score for the feature **capital.loss** is 0.249. The drift detection method is Wasserstein distance (normed). The drift detection threshold is 0.1.



2. The drift score for the feature **capital.gain** is 0.167. The drift detection method is Wasserstein distance (normed). The drift detection threshold is 0.1.



Model Quality Assessment

Model quality monitoring ensures that the trained models continue to perform effectively. The following steps are implemented:

1. **Data Quality Test Suite** – The DataQualityTestPreset is used to evaluate feature consistency and missing values.
2. **Model Stability Analysis** – The DataStabilityTestPreset assesses whether the model maintains its predictive accuracy across different data distributions.
3. **Automated Reporting** – If model quality issues are detected, an HTML report is generated for review.

For the result that we have tested, there are 57 tests in total for making Model Quality Suite, which 50 tests are success and 7 are failed, which contain Share of the Most Common Value, Number of Constant Columns, Number of Duplicate Rows, and Number of Rows:

1. Share of the Most Common Value: The most common value in the column **age** is 28. Its share is 0.0304. The test threshold is eq=0.0274 ± 0.00274.
2. Share of the Most Common Value: The most common value in the column **capital.gain** is 0. Its share is 1. The test threshold is eq=0.892 ± 0.0892.

3. Share of the Most Common Value: The most common value in the column **fnlwgt** is 125000. Its share is 0.0007. The test threshold is eq=0.0005 ± 5e-05.
4. Number of Constant Columns: The number of constant columns is 2. The test threshold is lte=0.
5. Number of Duplicate Columns: The number of duplicate columns is 1. The test threshold is lte=0.
6. Number of Duplicate Rows: The number of duplicate rows is 1. The test threshold is eq=4.54 ± 0.454.
7. Number of Rows: The number of rows is 7561. The test threshold is eq=2.5e+04 ± 2.5e+03.

Monitoring Implementation

The monitoring pipeline is executed within a **Jupyter Notebook**, integrating Evidently AI's test suites to automate drift detection and model evaluation. The results provide insights into:

- **Feature consistency across datasets**
- **Changes in data distribution affecting predictions**
- **Model stability over time**

Conclusion

By implementing dataset drift and model quality monitoring, this project ensures that the predictive models remain reliable and adaptable to evolving data patterns. Future improvements may include **real-time monitoring dashboards** and **automated model retraining** based on detected drift.

4.11 Web Application Deployment and API Usage

This section describes the development of the web application and its integration with the backend API to provide a user-friendly interface for interacting with the machine learning model.

Frontend Development

The frontend of the application was developed using the Next.js framework, leveraging TypeScript for type safety and Tailwind CSS for styling. The project structure is modular, with components organized under the components directory. Key features of the frontend include:

1. User Input Forms: The CompleteAnalysis component (complete-analysis.tsx) provides a form for users to input data such as age, education level, marital status, and occupation. The form uses the react-hook-form library for state management and validation.

2. API Integration: The frontend communicates with the backend API hosted at http://localhost:5000 (or http://backend:5000 in production). API endpoints are defined in the API_ENDPOINTS object in the MainContent component (main-content.tsx)
3. Visualization of Model Metrics: The ModelMetrics component (model-metrics.tsx) displays performance metrics such as accuracy, F1 score, precision, and recall. These metrics are fetched from the /api/metrics endpoint and rendered dynamically.
4. API Documentation: The ApiList component (apiList.tsx) provides a list of available API endpoints, including their HTTP methods and paths, for easy reference.

Backend API

The backend API, implemented using Flask, serves as the interface between the machine learning model and the frontend. Key endpoints include:

1. Prediction:
    a. Endpoint: /api/predict
    b. Method: POST
    c. Description: Accepts user input in JSON format and returns a binary prediction (>50K or <=50K).
2. Prediction Probabilities:
    a. Endpoint: /api/predict_proba
    b. Method: POST
    c. Description: Returns the probabilities for each income class.
3. Model Information:
    a. Endpoint: /api/model_info
    b. Method: GET
    c. Description: Provides metadata about the model, such as the number of features and training date.
4. Feature Explanation:
    a. Endpoint: /api/explain
    b. Method: POST
    c. Description: Returns feature importance scores to explain the model's predictions.
5. Health Check:
    a. Endpoint: /api/health
    b. Method: GET
    c. Description: Confirms that the API is running and accessible.
6. Feature List:
    a. Endpoint: /api/features
    b. Method: GET
    c. Description: Returns the list of features required for prediction.

Workflow

1. User Interaction: Users interact with the web application to input data and submit it for prediction.
2. API Requests: The frontend sends requests to the backend API using the fetch API or libraries like useSWR for data fetching.
3. Response Handling: The backend processes the request, performs the prediction or retrieves the requested information, and sends a JSON response to the frontend.
4. Result Display: The frontend dynamically updates the UI to display the results, such as the predicted income class or model metrics.

Development

The frontend and backend are containerized using Docker. The backend is exposed on port 5000, while the frontend runs on port 3000. This setup ensures seamless integration and portability across environments.

4.12 CI/CD Integration with GitHub Actions and Future Plans

Overview

Continuous Integration and Continuous Deployment (**CI/CD**) play a crucial role in automating software development workflows, ensuring efficient testing, deployment, and monitoring of machine learning models. In this project, **GitHub Actions** is utilized to streamline the CI/CD pipeline, enabling automated model training, testing, and deployment.

CI/CD Workflow with GitHub Actions

The CI/CD pipeline is structured to automate key processes, ensuring seamless integration and deployment. The workflow includes:

1. **Code Integration & Testing**
    a. Upon each push to the repository, GitHub Actions triggers automated tests to validate model performance.
    b. Unit tests and integration tests ensure that changes do not introduce errors.
2. **Containerization & Deployment**
    a. The trained model is packaged using **Docker**, ensuring consistency across environments.
    b. The containerized application is deployed to a **staging environment** for validation before production release.
3. **Automated Model Monitoring**
    a. Evidently AI is integrated into the pipeline to track dataset drift and model quality.
    b. If significant drift is detected, alerts are triggered for model retraining.

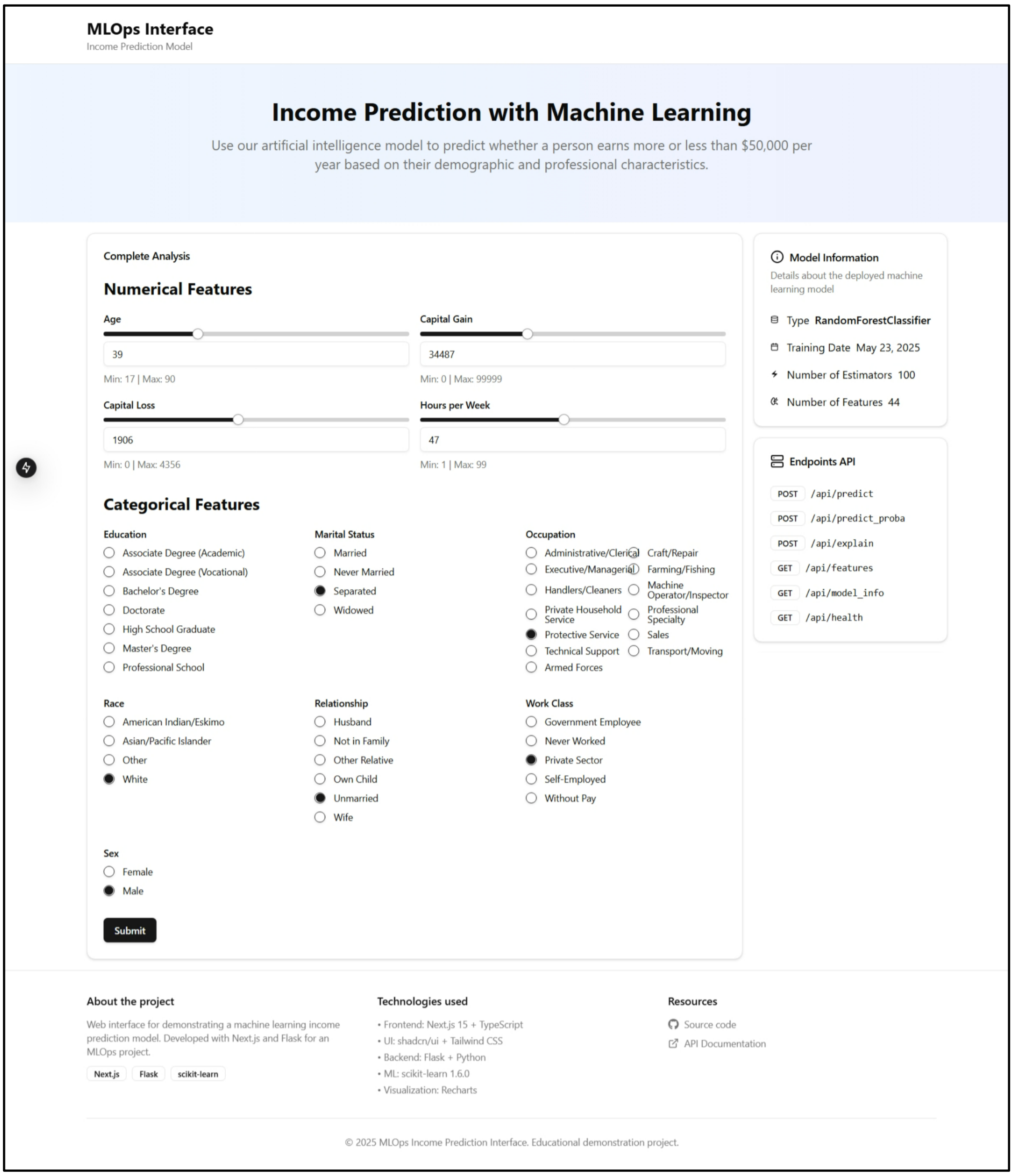
Future Plans: Cloud Deployment

As the project evolves, **cloud deployment** will be explored to enhance scalability and accessibility. Potential cloud solutions include:

- **AWS Lambda & S3** – Serverless deployment for efficient model inference.
- **Azure ML & Kubernetes** – Scalable model hosting with automated retraining.
- **Google Cloud AI Platform (Vertex AI)** – Managed ML services for streamlined deployment.

By integrating **CI/CD with cloud solutions**, the project will achieve **real-time model updates, automated scaling, and enhanced security**, ensuring long-term sustainability and performance optimization.

Chapter 5
Final Result



MLOps Interface
Income Prediction Model

Income Prediction with Machine Learning

Use our artificial intelligence model to predict whether a person earns more or less than $50,000 per year based on their demographic and professional characteristics.

Complete Analysis

Numerical Features

Age: 39 (Min: 17 | Max: 90)	Capital Gain: 34487 (Min: 0 | Max: 99999)
Capital Loss: 1906 (Min: 0 | Max: 4356)	Hours per Week: 47 (Min: 1 | Max: 99)

Categorical Features

Education: Associate Degree (Academic) / Associate Degree (Vocational) / Bachelor's Degree / Doctorate / High School Graduate / Master's Degree / Professional School

Marital Status: Married / Never Married / [✓ Separated] / Widowed

Occupation: Administrative/Clerical / Craft/Repair / Executive/Managerial / Farming/Fishing / Handlers/Cleaners / Machine Operator/Inspector / Private Household Service / Professional Specialty / [✓ Protective Service] / Sales / Technical Support / Transport/Moving / Armed Forces

Race: American Indian/Eskimo / Asian/Pacific Islander / Other / [✓ White]

Relationship: Husband / Not in Family / Other Relative / Own Child / [✓ Unmarried] / Wife

Work Class: Government Employee / Never Worked / [✓ Private Sector] / Self-Employed / Without Pay

Sex: Female / [✓ Male]

Submit

Model Information
Details about the deployed machine learning model

Type	RandomForestClassifier
Training Date	May 23, 2025
Number of Estimators	100
Number of Features	44

Endpoints API

POST	/api/predict
POST	/api/predict_proba
POST	/api/explain
GET	/api/features
GET	/api/model_info
GET	/api/health

About the project
Web interface for demonstrating a machine learning income prediction model. Developed with Next.js and Flask for an MLOps project.
Next.js Flask scikit-learn

Technologies used
- Frontend: Next.js 15 + TypeScript
- UI: shadcn/ui + Tailwind CSS
- Backend: Flask + Python
- ML: scikit-learn 1.6.0
- Visualization: Recharts

Resources
Source code
API Documentation

© 2025 MLOps Income Prediction Interface. Educational demonstration project.

5.1 Web Interface Overview

The final implementation of the **Income Prediction with Machine Learning** model is presented through a web interface designed for ease of use and accessibility. The interface allows users to input demographic and professional attributes to predict whether an individual earns **more or less than $50,000 per year**.

Interface Features
The web application is structured into several key sections:

- **Complete Analysis**: Displays both **numerical** and **categorical** features used in the prediction model.
- **Model Information**: Provides details about the deployed **Random Forest Classifier**, including the number of estimators (**100**) and the number of features (**44**).
- **API Endpoints**: Lists available API routes for prediction, probability estimation, feature retrieval, and model health checks.
- **Error Handling**: Displays error messages when model metrics fail to load.
- **Technology Stack**: Developed using **NextJS 15 + TypeScript** for the frontend, **Flask + Python** for the backend, and **scikit-learn 1.0.0** for machine learning.

5.2 Model Selection for Deployment

Choosing Random Forest for Deployment

While **Logistic Regression** demonstrated the best performance in terms of accuracy during model evaluation and income classification, **Random Forest** was selected for trial deployment due to its robustness and ability to handle complex relationships in data. The key reasons for this choice include:

- **Interpretability & Feature Importance**: Random Forest provides insights into feature importance, making it easier to understand which attributes contribute most to income prediction.
- **Handling Non-Linear Relationships**: Unlike Logistic Regression, Random Forest can model complex interactions between features without assuming linearity.
- **Resilience to Noisy Data**: Random Forest is less sensitive to outliers and missing values, ensuring stable predictions in real-world applications.
- **Scalability & Parallel Processing**: The model can efficiently process large datasets using multiple decision trees, making it suitable for deployment scenarios with high user traffic.

Recommendation for Using Logistic Regression on Final Development

Despite Random Forest being chosen for deployment, **Logistic Regression remains the best model in terms of accuracy**. It was primarily used during testing and development due to its **computational efficiency** and **ease of interpretation**. Logistic Regression serves as a benchmark model, ensuring that the deployed Random Forest classifier meets expected performance standards.

5.3 Conclusion

The final implementation successfully integrates machine learning into a web-based prediction system. The choice of **Random Forest for deployment** ensures robustness and adaptability, while **Logistic Regression remains the best model for testing and validation**. Future improvements may include **hyperparameter tuning**, **real-time model retraining**, and **enhanced API functionalities** to further optimize prediction accuracy and user experience.

Chapter 6
Conclusion and Discussion

6.1 Summary of Findings

This project successfully implemented machine learning techniques to predict income levels based on **US Adult Census data**. The classification models—**Logistic Regression, Gradient Boosting, and Random Forest**—were evaluated for performance using cross-validation metrics. While **Logistic Regression** demonstrated the highest accuracy in model testing, **Random Forest** was selected for deployment due to its robustness, interpretability, and ability to handle complex feature interactions.

Additionally, the implementation of **Directed Acyclic Graph (DAG) workflows** facilitated efficient automation and monitoring of data preprocessing, model training, and deployment. **Evidently AI** was leveraged for dataset drift detection and model quality assessment, ensuring the predictive models remain reliable over time.

6.2 Discussion and Implications

The key insights from this project include:

- **Feature Engineering Enhancements**: Consolidating categorical variables and scaling numeric features improved model accuracy and generalization.
- **Model Selection and Deployment**: Despite Logistic Regression outperforming other models in testing, Random Forest was deployed due to its stability and adaptability in real-world settings.
- **Model Monitoring for Continuous Improvement**: The integration of Evidently AI enables proactive detection of data drift, allowing timely updates to maintain model accuracy.
- **Scalability with NextJS and Docker**: The web application structure ensures smooth integration of machine learning predictions into an interactive interface.

The findings from this study emphasize the **importance of balancing model accuracy with real-world deployment considerations**. While high accuracy is desirable, factors such as **interpretability, adaptability, and monitoring capabilities** play a crucial role in selecting a production-ready model.

6.3 Cloud Deployment Considerations

For future improvements, **cloud deployment** presents an opportunity to enhance scalability, accessibility, and operational efficiency. Deploying the income prediction model on cloud platforms such as **AWS, Azure, or Google Cloud** offers the following advantages:

- **Elastic Computing Power** – Dynamically scale resources based on real-time usage.
- **Containerized Deployment** – Utilize **Docker and Kubernetes** for seamless updates and version control.
- **Automated Model Retraining** – Implement **scheduled updates** to refine predictions as new data becomes available.
- **Security and Data Management** – Ensure encrypted storage and compliance with **GDPR and privacy regulations**.

Future iterations of this project may explore **serverless architectures**, API integrations, and **ML pipeline automation** to optimize the deployment framework further. Due to a factor like cost management, we decided not to deploy with a cloud solution at the moment.

6.4 Limitations and Future Work

Despite the successes of this project, certain limitations remain:

- **Data Bias & Representativeness**: The dataset is based on census records from 1994, potentially limiting generalization to modern income trends.
- **Hyperparameter Optimization**: While models were optimized based on standard parameters, further tuning could enhance performance.
- **Automated Retraining Strategies**: Future work could focus on **developing real-time retraining pipelines**, allowing models to continuously adapt to new data.

Future research may explore alternative **ensemble learning techniques**, improved **dataset collection methods**, and enhanced **real-time monitoring dashboards** to further optimize predictive performance.

6.5 Conclusion

This project demonstrates the potential of **machine learning** in socioeconomic analysis, offering a scalable and interpretable approach to **income prediction**. The structured methodology—from **data preprocessing** to **model deployment and monitoring**—ensures that predictions remain **accurate, adaptable, and relevant**. By continuously refining the workflow, this approach can serve as a foundation for future **data-driven policymaking, financial forecasting, and social impact analysis**.

Acknowledgement

We hope this report reflects our commitment to thoughtful analysis and creative problem solving, and we sincerely give an appreciation to our CPE393 Special Topics III: Machine Learning Operations course instructors, both Asst. Prof. Dr. Santitham Prom-on and Prof. Dr. Aye Hninn Khine, for the guidance and feedback that helped in shaping our work.

Source Code

All data and source code for this work achieve in this link: https://github.com/chrnthnkmutt/CPE393_TBA_MLOps

References

[1] GeeksforGeeks. (2024, April 9). *Gradient Boosting vs Random Forest*. GeeksforGeeks. https://www.geeksforgeeks.org/gradient-boosting-vs-random-forest/

[2] *1.11. Ensembles: Gradient boosting, random forests, bagging, voting, stacking*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/ensemble.html

[3] Haitao Du. Answer to "Boosting A Logistic Regression Model", *Meta Stack Exchange*. 16 February, 2018. <https://stats.stackexchange.com/questions/329066/boosting-a-logistic-regression-model>.
(Licensed under CC BY-SA 3.0: <https://creativecommons.org/licenses/by-sa/3.0/>)

[4] Kohavi, R. (1996). Census Income [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5GP7S and https://archive.ics.uci.edu/dataset/20/census+income