Faculty of Engineering and Technology
Department of Electrical and Computer Engineering
ENCS5141—Intelligent Systems Laboratory

Assignment #1—Data Cleaning and Feature Engineering ,Comparing Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP) on Assignment1 dataset

Prepared by: Mohammad Salem—1200651

Instructor: Dr : Yazan Abu Farha
Assistant: Ahmad Abbas
Date: April 29, 2025

Abstract

This case study aims to examine the performance of three well-known and diverse classification models: Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP). Using a real-world dataset provided in the Ass1+.zip file, we implemented a systematic preprocessing workflow that included data cleaning, feature engineering (feature encoding), normalization, and dimensionality reduction.
The models were evaluated using key metrics: accuracy, precision, recall, F1 score, and training time. To improve model performance, we applied both manual hyperparameter tuning and automated tuning via GridSearchCV. For comparison, we also evaluated baseline (untuned) versions of each model.
The results showed that while SVM achieved the highest overall accuracy, Random Forest offered a more balanced trade-off between recall and training efficiency. MLP demonstrated competitive precision but required significantly longer training time.
These observations highlight the strengths and weaknesses of each classifier and emphasize the important role that preprocessing and parameter tuning play in enhancing model performance.

Contents

Abstract **I**

1 Introduction **1**

2 Procedure and Discussion Part 1 **2**

2.1 Load and Explore the Dataset . . . 2

2.2 Data Visualization . . . 4

2.3 Box plote and Outlayers detection . . . 4

2.4 Histograms for Feature Distributions . . . 6

2.5 Scatter Plot . . . 7

2.6 Correlation Heatmap . . . 8

2.7 Data Cleaning . . . 9

2.8 Feature Engineering: . . . 11

2.9 Model Evaluation . . . 12

3 Procedure and Discussion Part 2 : Comparative Analysis of Classification Techniques **13**

3.1 Data Preparation . . . 13

3.2 Model Training . . . 14

3.3 Model Comparison . . . 14

4 Random Forest Tuning **16**

4.1 Manual Tuning Phase . . . 16

4.2 Medium Search (Fast Grid Search) Tuning Phase . . . 19

4.3 Very Deep Search (Advanced Big Grid) . . . 21

4.4 Evaluate Grid Search Best Models . . . 21

4.5 Compined the manually tuned also grid tuned results, also models without tuning comparison . . . 23

5 Conclusion **25**

List of Figures

2.1 First 5 rows from data set 2
2.2 Data Schema 3
2.3 Summary Statistic 4
2.4 Box plot for 'age' and 'fnlwgt' featurs. 5
2.5 Box plot for 'educational-num', and 'capital-gain' featurs. 5
2.6 Box plot for 'capital-loss', 'hours-per-week' featurs. 5
2.7 Histogram for 'age' and 'fnlwgt' featurs. 6
2.8 Histogram for 'educational-num', and 'capital-gain' featurs. 6
2.9 Histogram for 'capital-loss', 'hours-per-week' featurs. 6
2.10 Sample to understand the data by two features and label 'Age vs Hours-Per-Week colored by Income' 7
2.11 Correlation heatMap 8
2.12 missing Value Per Column 9
2.13 Null value after replacement the ? to Null , in (a) number of Null values for each feature , (b) persentage from All data set 10
2.14 Comparison: RF on Preprocessed vs Raw vs PCA-Reduced Data . . 12
3.1 full Comparison Table (RF vs SVM vs MLP) 14
4.1 Manualy Tuned Results for (RF vs SVM vs MLP) 18
4.2 Final Evaluation of Grid Search-Tuned Models: 21

1 Introduction

Classification is a fundamental task in machine learning that involves assigning categorical labels to input data based on learned patterns. It is widely used in applications such as spam detection, medical diagnosis, and demand forecasting. In this experiment, we evaluate and compare the classification capabilities of three different machine learning algorithms: Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP).

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of their predictions. Its ability to reduce overfitting and handle noisy data makes it a robust choice for many classification problems. In this experiment, RF models are tested with different numbers of estimators and tree depths to analyze the impact of complexity on performance.

SVM, in contrast, finds the optimal hyperplane that separates data points of different classes. It supports multiple kernel types, allowing it to adapt to both linear and non-linear classification tasks. The time complexity of SVM depends heavily on the choice of kernel and regularization parameters. In this case study, linear, radial basis function (RBF), and polynomial kernels are applied and their influence on model accuracy and training time is analyzed.

Multilayer Perceptron (MLP), a form of feedforward neural network, utilizes hidden layers with activation functions to capture complex, non-linear patterns. Though powerful, MLP models are computationally expensive and sensitive to hyperparameters such as the number of layers, neurons, and learning rate. Several MLP configurations are explored in this experiment to evaluate the trade-offs between training time and predictive performance.

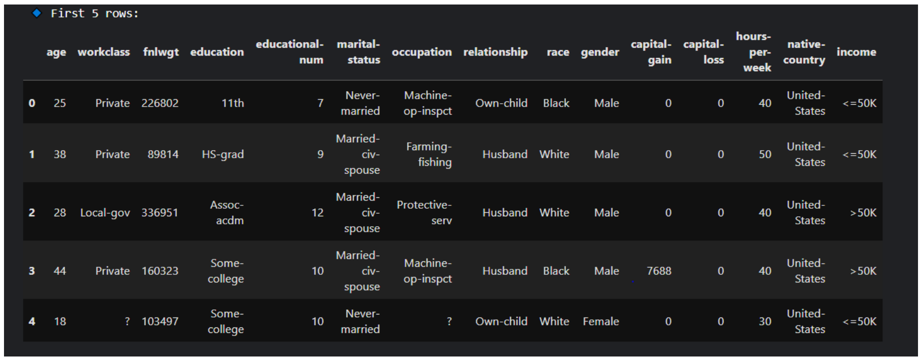
The objective of this experiment is to systematically compare the accuracy, precision, recall, and computational efficiency of RF, SVM, and MLP classifiers. Additionally, we examine how feature engineering and hyperparameter tuning influence the models' outcomes, verifying whether the observed results align with theoretical expectations of model behavior and complexity.

2 Procedure and Discussion Part 1

2.1 Load and Explore the Dataset

The experimental process began by uploading the dataset archive to Google Colab then to Kaggle, After initializing the environment (setup by test connections , I chose CPU then GPU) the dataset was extracted and loaded on data frame (df).

The first step was to explore the dataset , general understanding of its structure, size, and content. This included reading the first few rows, checking the number of samples and features, examining data types, and identifying missing or improperly encoded values.

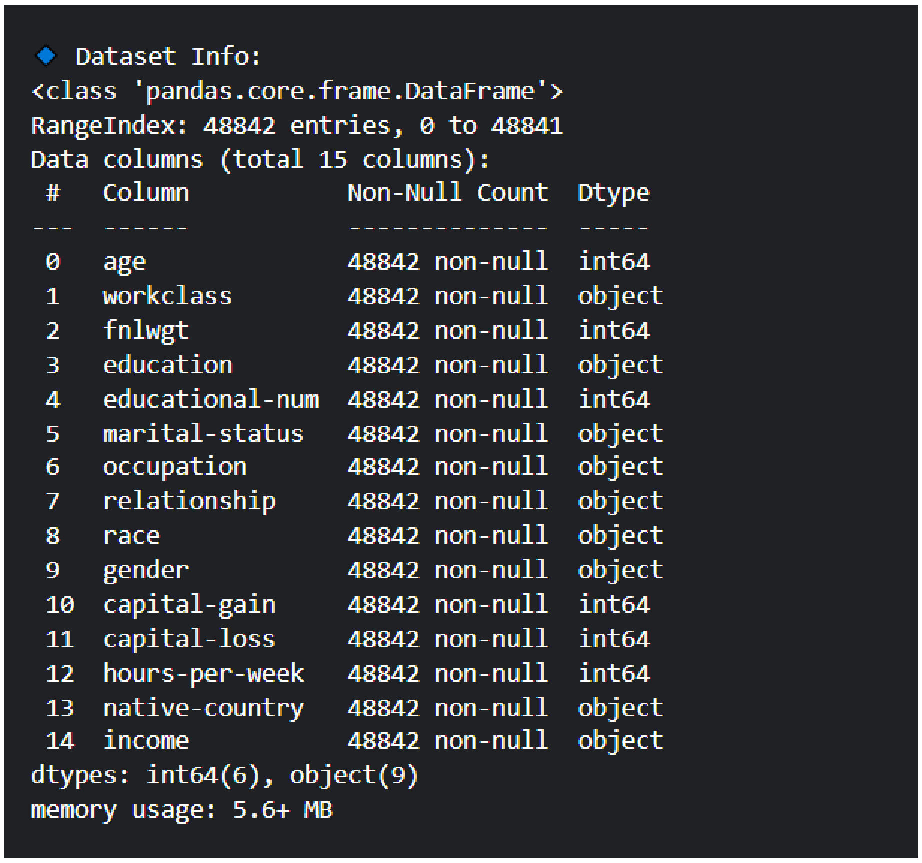


🔹 First 5 rows:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

Figure 2.1: First 5 rows from data set

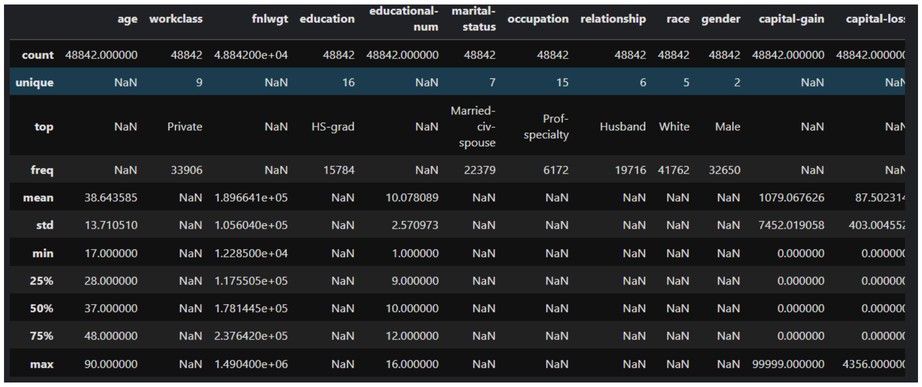
This step to understand the dataset schema, the info() method was used, revealing that it contains 48,842 rows and 15 columns. The data set includes both numerical and categorical characteristics, with the target variable being income (which we need to predict), representing a binary classification task
(income $\leq 50K$ or income $> 50K$).



```
🔹 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Figure 2.2: Data Schema

As you show on Figure 2.2: Data Schema, The dataset consists of 48,842 records with 15 attributes. These attributes include both numerical and categorical variables. Notably, all columns are fully populated, with no missing (NaN) values formally detected.
The numerical features include age, fnlwgt, educational-num, capital-gain, capital-loss, and hours-per-week, all stored as int64 data types.
The remaining attributes, such as workclass, education, marital-status, and income, are categorical and stored as object types. This structure highlights the need for encoding and scaling during the preprocessing phase.



	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-los
count	48842.000000	48842	4.884200e+04	48842	48842.000000	48842	48842	48842	48842	48842	48842.000000	48842.00000
unique	NaN	9	NaN	16	NaN	7	15	6	5	2	NaN	Na
top	NaN	Private	NaN	HS-grad	NaN	Married-civ-spouse	Prof-specialty	Husband	White	Male	NaN	Na
freq	NaN	33906	NaN	15784	NaN	22379	6172	19716	41762	32650	NaN	Na
mean	38.643585	NaN	1.896641e+05	NaN	10.078089	NaN	NaN	NaN	NaN	NaN	1079.067626	87.50231
std	13.710510	NaN	1.056040e+05	NaN	2.570973	NaN	NaN	NaN	NaN	NaN	7452.019058	403.00455
min	17.000000	NaN	1.228500e+04	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.00000
25%	28.000000	NaN	1.175505e+05	NaN	9.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.00000
50%	37.000000	NaN	1.781445e+05	NaN	10.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.00000
75%	48.000000	NaN	2.376420e+05	NaN	12.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.00000
max	90.000000	NaN	1.490400e+06	NaN	16.000000	NaN	NaN	NaN	NaN	NaN	99999.000000	4356.00000

Figure 2.3: Summary Statistic

This table on Figure 2.3: provides summary statistics and frequency distributions for both numerical and categorical features, highlighting value ranges, most frequent categories
(e.g., "Private" in workclass), and key statistical metrics like mean, min,max,25%,50%,75% from data , and standard deviation for attributes such as age, capital-gain, and hours-per-week.

2.2 Data Visualization

2.3 Box plote and Outlayers detection

The boxplots in Figures 2.6,2.7,2.8 visualize the distribution of six numerical features in the dataset.
Most features, including `capital-gain`, `capital-loss`, and `fnlwgt`, exhibit significant right-skew with numerous outliers, indicating heavy-tailed distributions. Features like `age` and `educational-num` appear more normally distributed, while `hours-per-week` is tightly clustered around 40 hours, reflecting a standard full-time work schedule.

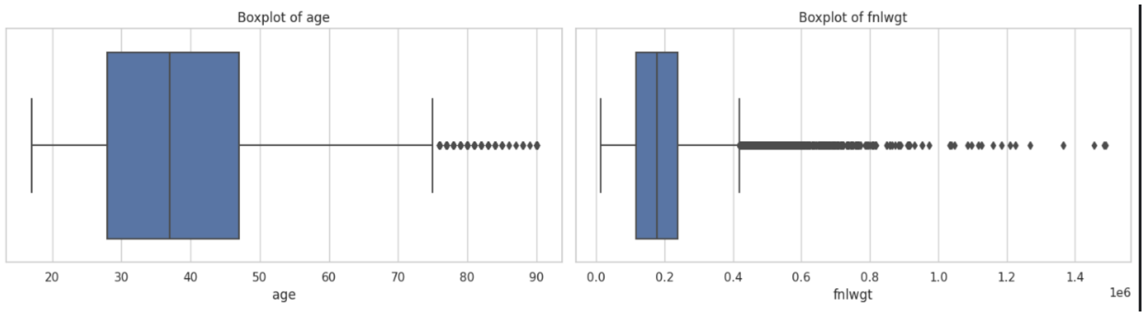


Figure 2.4: Box plot for 'age' and 'fnlwgt' featurs.

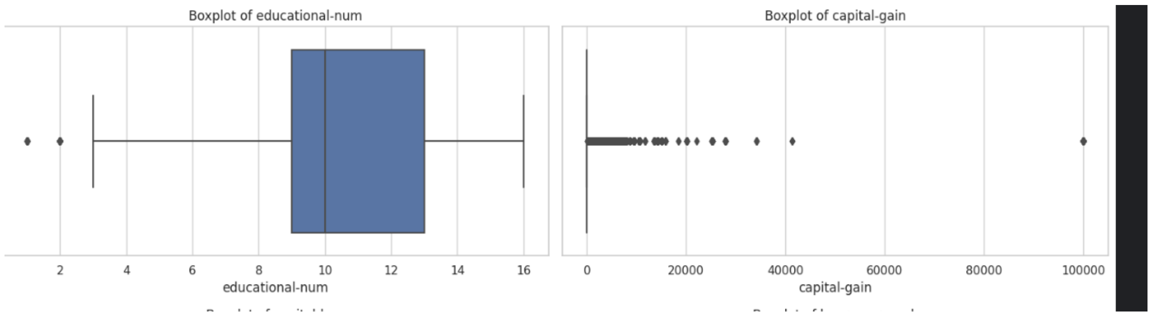


Figure 2.5: Box plot for 'educational-num', and 'capital-gain' featurs.

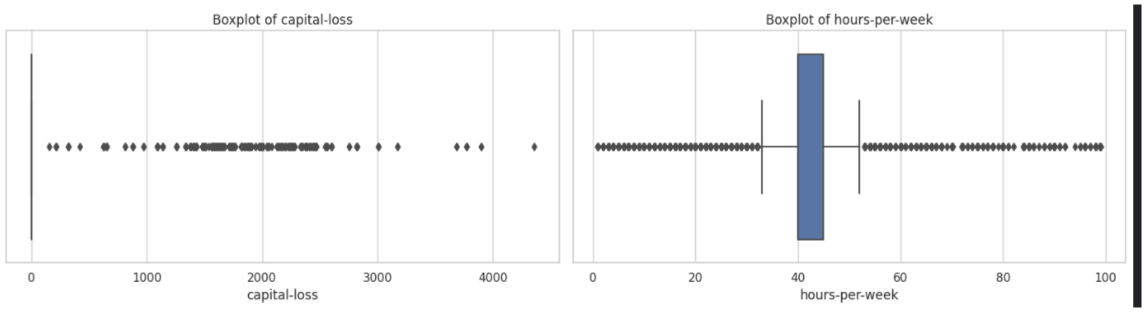


Figure 2.6: Box plot for 'capital-loss', 'hours-per-week' featurs.

These plots reveal the need for normalization or scaling and highlight potential influence from extreme values during model training.

2.4 Histograms for Feature Distributions

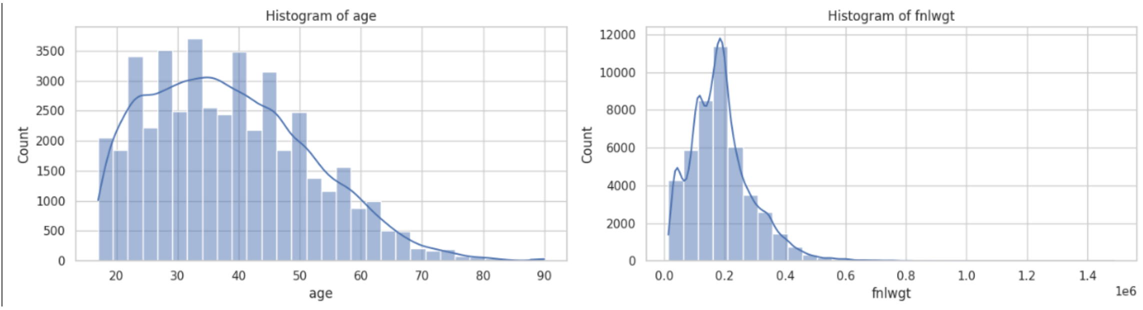


Figure 2.7: Histogram for 'age' and 'fnlwgt' featurs.

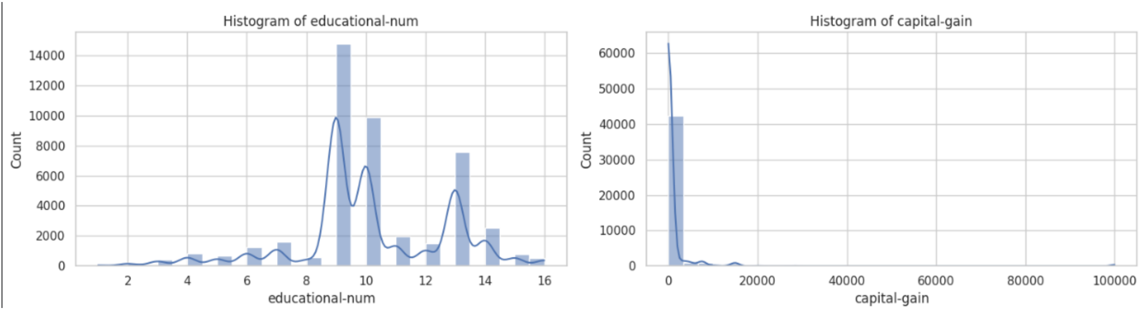


Figure 2.8: Histogram for 'educational-num', and 'capital-gain' featurs.

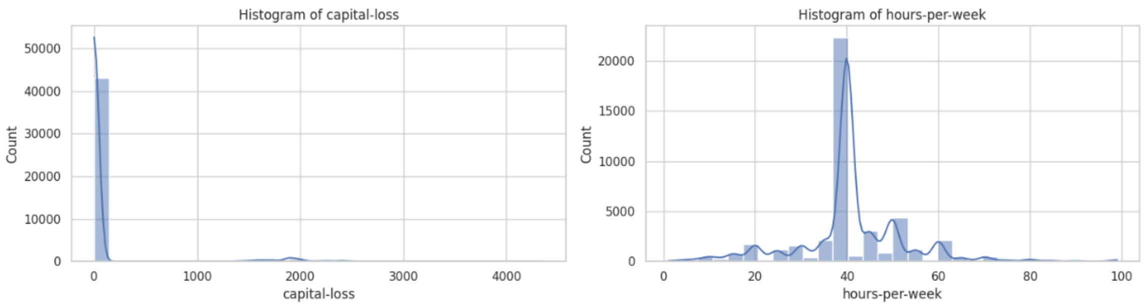


Figure 2.9: Histogram for 'capital-loss', 'hours-per-week' featurs.

The histogram plots in Figure 2.9,Figure 2.10,Figure 2.11 display the distribution of key numerical features in the dataset,(6 features).

The histogram plots in Figures 2.9 - 2.2.11 display the distribution of key numerical features in the dataset. The age variable shows a moderately right-skewed distribution with a peak between 25 and 40 years, indicating a younger working population. The fnlwgt feature, which reflects the sampling weight, is heavily right-skewed with a sharp peak around 200,000. Educational attainment (educational-num) appears multimodal with most individuals concentrated around 9–10 years of education. Both capital-gain and capital-loss are sharply skewed toward zero, confirming that the majority of individuals report no such gains or losses. Finally, hours-per-week is tightly clustered around 40, typical of full-time employment, with a few extended hours up to 99.
These distributions provide crucial insight for preprocessing decisions,
such as normalization and model selection.

2.5 Scatter Plot

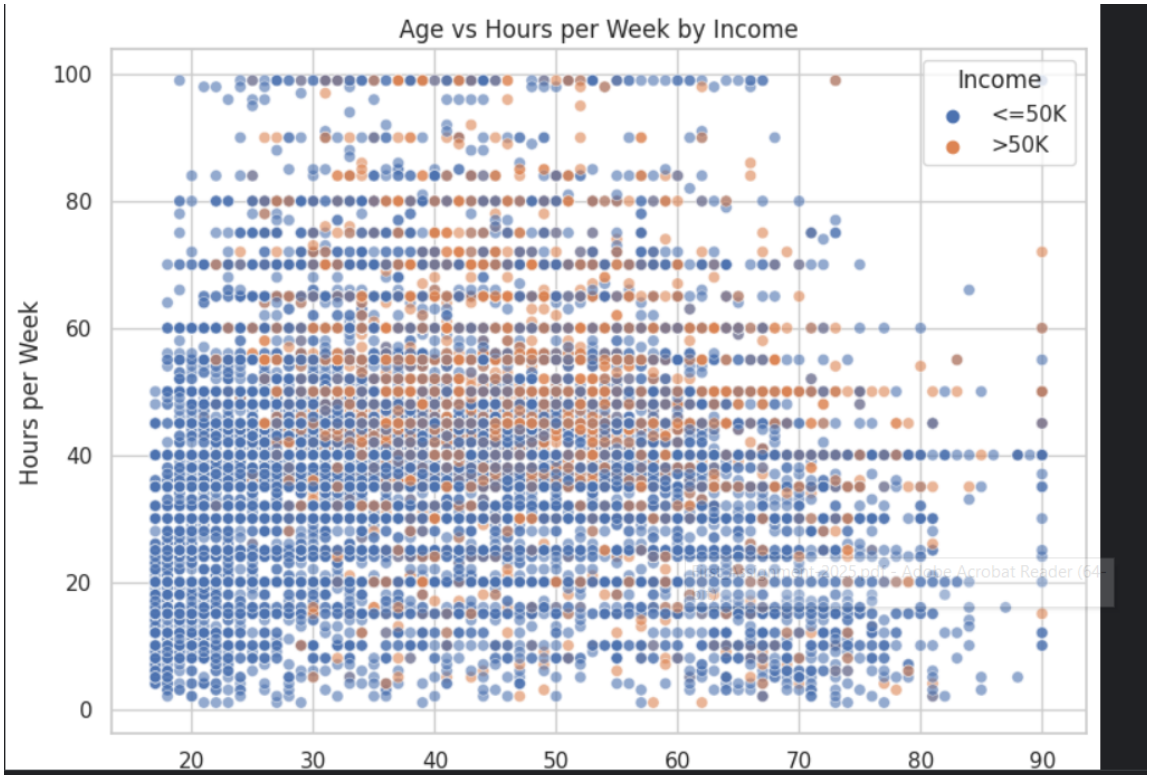


Figure 2.10: Sample to understand the data by two features and label
'Age vs Hours-Per-Week colored by Income'

This scatter plot illustrates the relationship between age and hours worked per week, color-coded by income class. The majority of individuals earning less than $50K(blue)clusteraround40workinghoursregardlessofage, whilehigher-incomeindividuals(orange)t$ $agedworkers.$

2.6 Correlation Heatmap

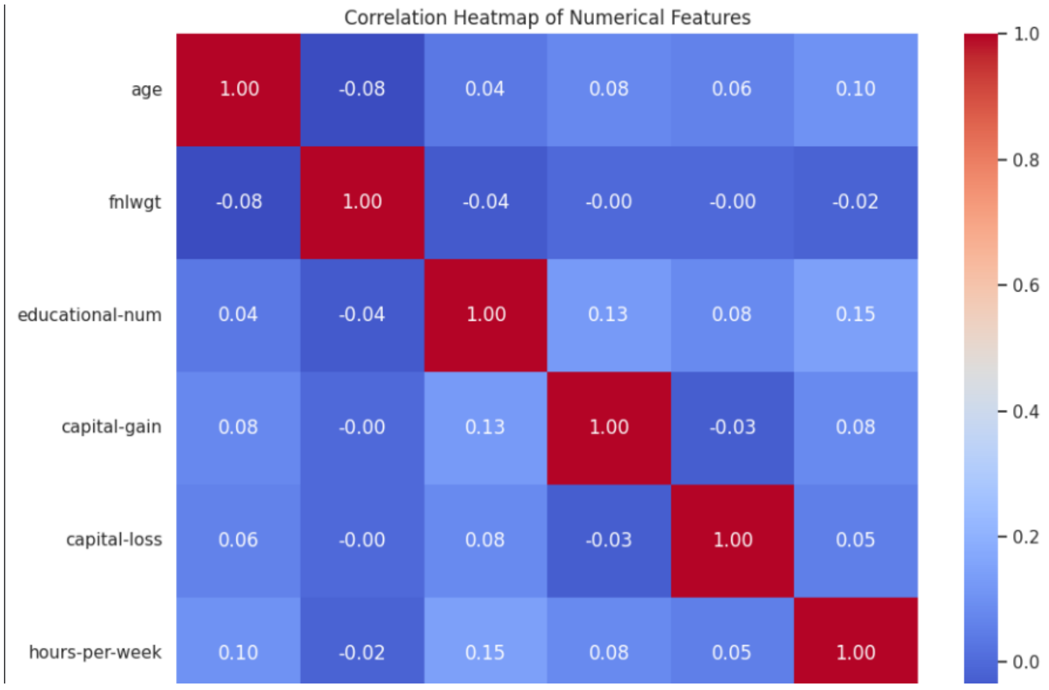
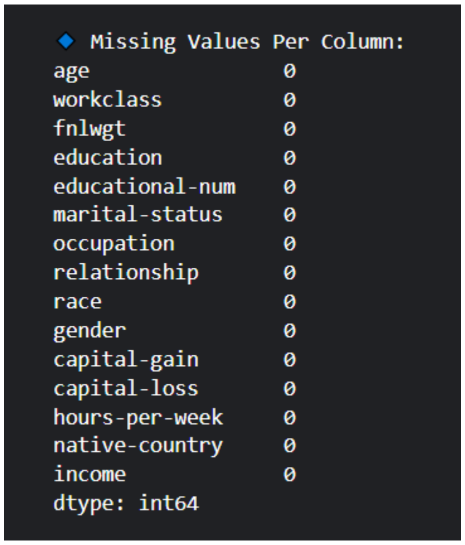


Figure 2.11: Correlation heatMap

There is more corelation heat map on experment at .ipynp but it is the same , Weak corelation Correlation Heatmap by Income Group less than or equal 50K group , also, the Income Group large than or equal 50K group.

2.7 Data Cleaning

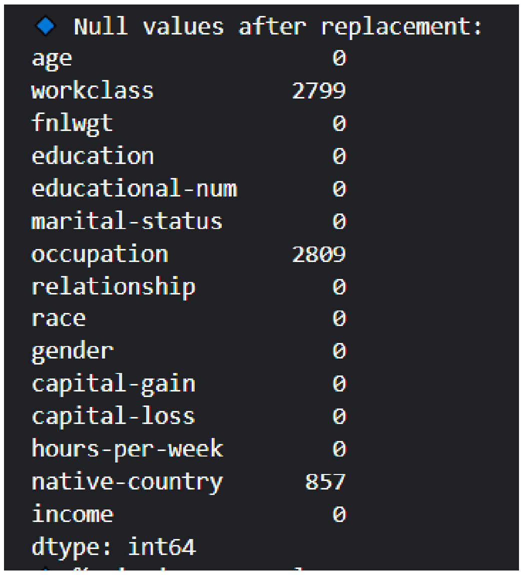
An essential part of the preprocessing phase is handling missing values. In this dataset, missing entries were represented using the "?" string rather than formal NaN values, These placeholders were systematically replaced with NaN. Therefore, the percentage of missing values was calculated for each feature to guide further cleaning decisions. This allowed for a more informed approach, ensuring that only features with significant missingness were addressed, either through imputation or row removal. This step was critical to prevent biases during training and to ensure model robustness.



```
🔹 Missing Values Per Column:
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

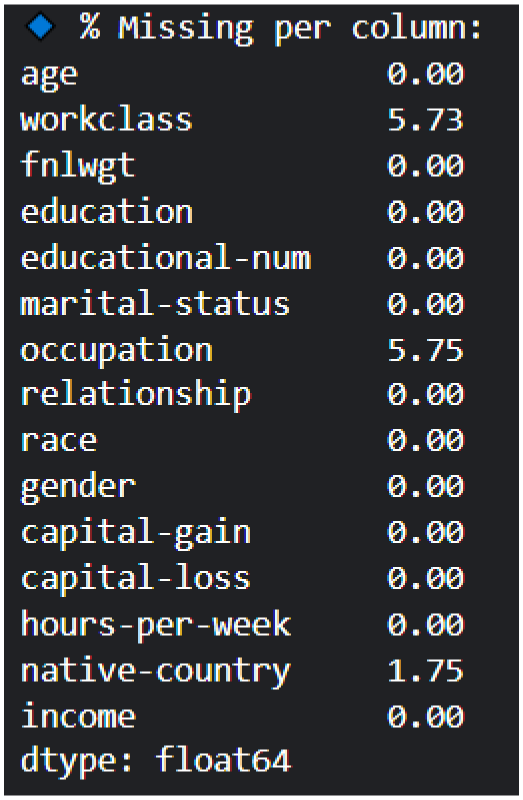
Figure 2.12: missing Value Per Column

As shown on Figure 2.4: there is no missing value , the data is completed and fully populated.



```
🔹 Null values after replacement:
age                    0
workclass           2799
fnlwgt                 0
education              0
educational-num        0
marital-status         0
occupation          2809
relationship           0
race                   0
gender                 0
capital-gain           0
capital-loss           0
hours-per-week         0
native-country       857
income                 0
dtype: int64
```

(a) Null value after replacement



```
🔹 % Missing per column:
age                0.00
workclass          5.73
fnlwgt             0.00
education          0.00
educational-num    0.00
marital-status     0.00
occupation         5.75
relationship       0.00
race               0.00
gender             0.00
capital-gain       0.00
capital-loss       0.00
hours-per-week     0.00
native-country     1.75
income             0.00
dtype: float64
```

(b) Null value after replacement persentage

Figure 2.13: Null value after replacement the ? to Null , in (a) number of Null values for each feature , (b) persentage from All data set

As desciped on Figure 2.5

It was observed that only a small percentage of records were affected — approximately 5.7% of the dataset.
According to the experiments on lab when missingness is below 10% and the dataset is sufficiently large, it is acceptable to drop affected rows rather than perform imputation.
Given that the dataset originally contained 48,842 rows, removing around 2,800 records still left over 46,000 examples, ensuring no significant impact on the statistical validity of the dataset.
Moreover, imputing categorical features like workclass and occupation with the mode or placeholder categories could introduce bias or noise.

Therefore, dropping missing rows was selected as a safer, cleaner, and more reliable approach, aligning with both best practices and the lab experiment recommendations Also , as Dr Yazan said.

```
df_clean = df.dropna().reset_index(drop=True)
print(" Cleaned dataset shape:", df_clean.shape)
```

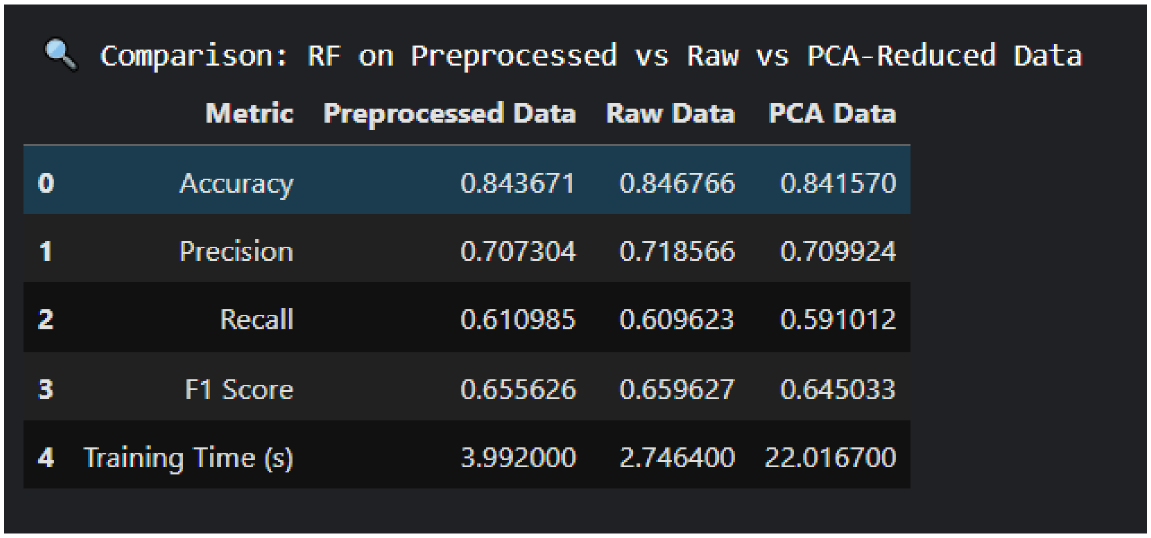
Listing 2.1: drop all missing data Cleaned dataset shape: (45222, 15)

2.8 Feature Engineering:

The feature engineering process was critical for preparing the dataset for machine learning models. Initially, irrelevant features such as `fnlwgt` were dropped, as they do not provide predictive value.
The target variable `income` was converted into a binary label to facilitate binary classification tasks.
All categorical features, including `education`, `workclass`, and `native-country`, were transformed using one-hot encoding to convert them into numerical representations suitable for model input.
For consistency and to optimize model performance, numerical features were identified and scaled using `StandardScaler`, which standardizes data by removing the mean and scaling to unit variance.
To reduce computational complexity and capture the most relevant patterns in the data, Principal Component Analysis (PCA) was applied to the scaled feature set. PCA components were retained such that 95% of the variance in the original data was preserved. This dimensionality reduction step ensured that the models received compact, noise-reduced inputs without sacrificing essential information.

All codes can you see on experiment at.ipynb file
Shape after encoding becums : (45222, 104)
After PCA applied. New shape of X is : (45222, 31)

2.9 Model Evaluation


🔍 Comparison: RF on Preprocessed vs Raw vs PCA-Reduced Data

	Metric	Preprocessed Data	Raw Data	PCA Data
0	Accuracy	0.843671	0.846766	0.841570
1	Precision	0.707304	0.718566	0.709924
2	Recall	0.610985	0.609623	0.591012
3	F1 Score	0.655626	0.659627	0.645033
4	Training Time (s)	3.992000	2.746400	22.016700

Figure 2.14: Comparison: RF on Preprocessed vs Raw vs PCA-Reduced Data

To evaluate the effectiveness of preprocessing steps, the dataset was split into training and testing subsets using an 80/20 split.
A Random Forest classifier was trained on three variants of the data: raw, preprocessed, and PCA-reduced. The preprocessed dataset underwent feature encoding, normalization, and dimensionality reduction, whereas the raw data was used as-is without scaling or encoding. Interestingly, the results showed only marginal differences between the raw and preprocessed datasets. The Random Forest achieved an accuracy of 84.37% on the preprocessed data and 84.67% on the raw data. This outcome is consistent with the nature of tree-based models, which are inherently insensitive to feature scaling and encoding transformations.
Since Random Forest splits features based on thresholds rather than Euclidean distance, standardization and one-hot encoding have minimal effect on its decision boundaries.
The PCA-reduced data slightly reduced accuracy and significantly increased training time, suggesting that dimensionality reduction may not benefit this model type.
Overall, the evaluation confirms that preprocessing has limited impact on Random Forest performance but is still important when comparing across other algorithms such as SVM and neural networks.

3 Procedure and Discussion Part 2 : Comparative Analysis of Classification Techniques

In the second phase of the assignment, three classification algorithms—Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP)—were implemented and evaluated to compare their effectiveness on the same preprocessed dataset derived from Part 1. Each model embodies a distinct learning paradigm: RF is a tree-based ensemble method known for robustness to outliers and minimal preprocessing requirements; SVM excels in finding optimal decision boundaries in high-dimensional space, and MLP, a type of feedforward neural network, captures complex nonlinear relationships through layered transformations.
The dataset was split into training and testing subsets, and each model was trained using default parameters as a baseline.
Their performances were assessed using metrics such as accuracy, precision, recall, F1 score, and training time.
This comparative study not only highlights the predictive strengths and weaknesses of each algorithm but also provides a foundation for understanding how each model reacts to feature complexity and noise within the dataset.

3.1 Data Preparation

We use the same preprocessed dataset from Part 1

3.2 Model Training

```
#  Train Random Forest
rf_model = RandomForestClassifier(n_estimators=100,
↪  random_state=42)
```

Listing 3.1: Train Random Forest with n$_estimators = 100$

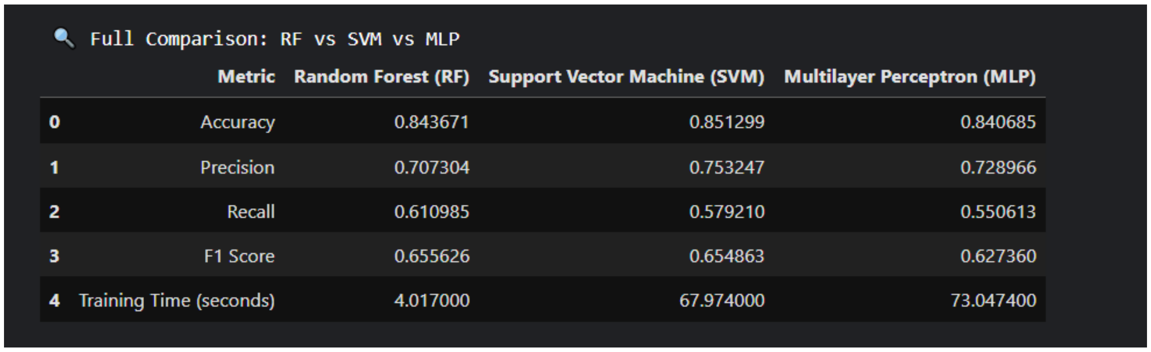
```
#Train SVM
svm_model = SVC(kernel='linear', random_state=42)
```

Listing 3.2: Train SVM with linear kernel

```
# Train MLP
mlp_model = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300,
↪  random_state=42)
```

Listing 3.3: Train MLP with 100 neurons in one hidden layer

3.3 Model Comparison



🔍 Full Comparison: RF vs SVM vs MLP

	Metric	Random Forest (RF)	Support Vector Machine (SVM)	Multilayer Perceptron (MLP)
0	Accuracy	0.843671	0.851299	0.840685
1	Precision	0.707304	0.753247	0.728966
2	Recall	0.610985	0.579210	0.550613
3	F1 Score	0.655626	0.654863	0.627360
4	Training Time (seconds)	4.017000	67.974000	73.047400

Figure 3.1: full Comparison Table (RF vs SVM vs MLP)

The comparative results of the three classification models—Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP)—revealed distinct performance characteristics aligned with each model's underlying behavior.

SVM achieved the highest overall accuracy (85.13%) and precision (75.32%), indicating that it was highly effective in correctly predicting positive cases while minimizing false positives.
However, its relatively lower recall (57.92%) suggests that it was more conservative, missing a larger proportion of true positive instances. On the other hand, Random Forest demonstrated the best balance between precision and recall, achieving the highest F1 score (65.56%) and the fastest training time (4 seconds).
This confirms RF's reputation as a fast, reliable classifier that performs well even without extensive preprocessing. The MLP model showed competitive precision (72.90%) but yielded the lowest recall and F1 score, coupled with the longest training time. These outcomes are consistent with expectations, as neural networks typically require more computational resources and tuning to perform optimally on tabular datasets.
Overall, the results illustrate how different models trade off between predictive power and efficiency, and they highlight Random Forest as a strong baseline for structured data classification tasks.

4 Random Forest Tuning

4.1 Manual Tuning Phase

```
    # RF variations
rf_variations = {
    "RF (n_estimators=100)":
    ↪  RandomForestClassifier(n_estimators=100, random_state=42),
    "RF (n_estimators=200)":
    ↪  RandomForestClassifier(n_estimators=200, random_state=42),
    "RF (n_estimators=300)":
    ↪  RandomForestClassifier(n_estimators=300, random_state=42),
    "RF (max_depth=10)": RandomForestClassifier(n_estimators=100,
    ↪  max_depth=10, random_state=42)
}
```

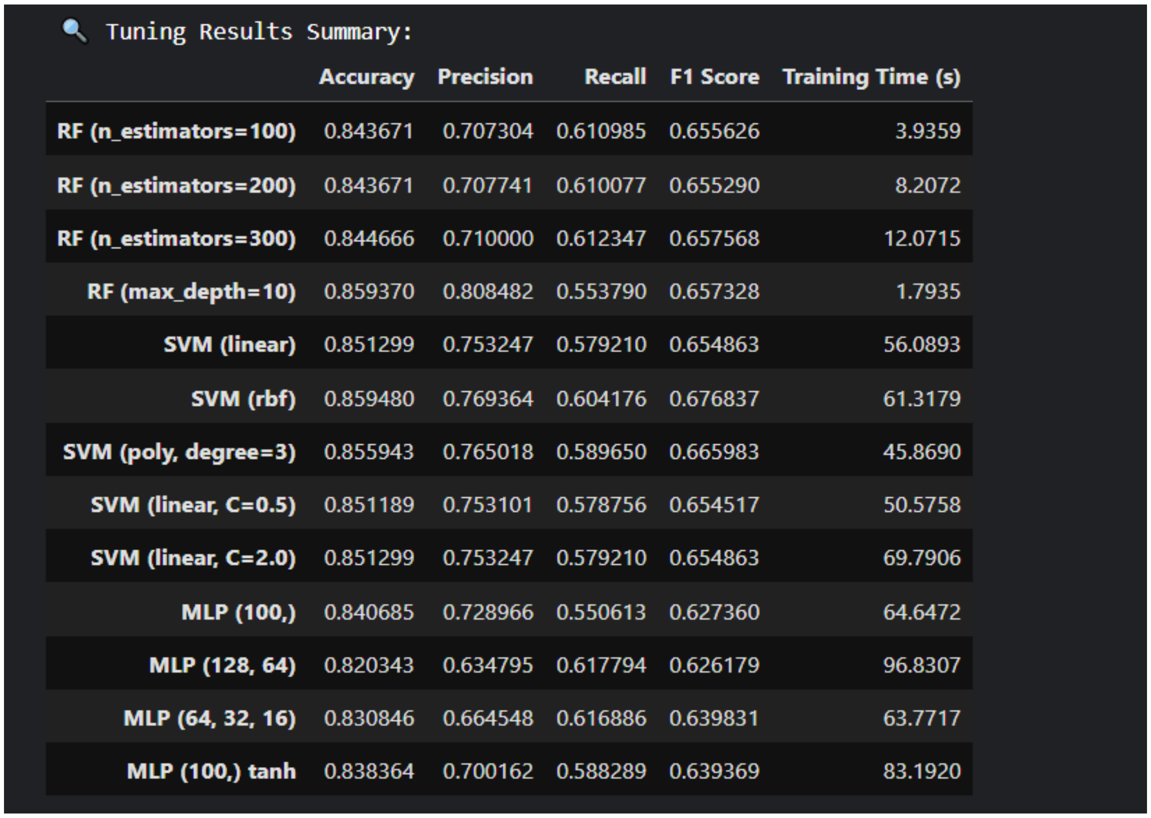
Listing 4.1: RF variations

```
    # SVM variations
svm_variations = {
    "SVM (linear)": SVC(kernel='linear', random_state=42),
    "SVM (rbf)": SVC(kernel='rbf', random_state=42),
    "SVM (poly, degree=3)": SVC(kernel='poly', degree=3,
    ↪  random_state=42),
    "SVM (linear, C=0.5)": SVC(kernel='linear', C=0.5,
    ↪  random_state=42),
    "SVM (linear, C=2.0)": SVC(kernel='linear', C=2.0,
    ↪  random_state=42)
}
```

Listing 4.2: SVM variations

```
   #  MLP variations
mlp_variations = {
    "MLP (100,)": MLPClassifier(hidden_layer_sizes=(100,),
    ↪  max_iter=300, random_state=42),
    "MLP (128, 64)": MLPClassifier(hidden_layer_sizes=(128, 64),
    ↪  max_iter=500, random_state=42),
    "MLP (64, 32, 16)": MLPClassifier(hidden_layer_sizes=(64, 32,
    ↪  16), max_iter=500, random_state=42),
    "MLP (100,) tanh": MLPClassifier(hidden_layer_sizes=(100,),
    ↪  activation='tanh', max_iter=500, random_state=42)
}
```

Listing 4.3: MLP variations



🔍 Tuning Results Summary:

	Accuracy	Precision	Recall	F1 Score	Training Time (s)
RF (n_estimators=100)	0.843671	0.707304	0.610985	0.655626	3.9359
RF (n_estimators=200)	0.843671	0.707741	0.610077	0.655290	8.2072
RF (n_estimators=300)	0.844666	0.710000	0.612347	0.657568	12.0715
RF (max_depth=10)	0.859370	0.808482	0.553790	0.657328	1.7935
SVM (linear)	0.851299	0.753247	0.579210	0.654863	56.0893
SVM (rbf)	0.859480	0.769364	0.604176	0.676837	61.3179
SVM (poly, degree=3)	0.855943	0.765018	0.589650	0.665983	45.8690
SVM (linear, C=0.5)	0.851189	0.753101	0.578756	0.654517	50.5758
SVM (linear, C=2.0)	0.851299	0.753247	0.579210	0.654863	69.7906
MLP (100,)	0.840685	0.728966	0.550613	0.627360	64.6472
MLP (128, 64)	0.820343	0.634795	0.617794	0.626179	96.8307
MLP (64, 32, 16)	0.830846	0.664548	0.616886	0.639831	63.7717
MLP (100,) tanh	0.838364	0.700162	0.588289	0.639369	83.1920

Figure 4.1: Manualy Tuned Results for (RF vs SVM vs MLP)

The hyperparameter tuning process offered key insights into how each model's performance could be optimized through configuration. For the Random Forest classifier, increasing the number of estimators from 100 to 300 provided only a marginal improvement in F1 score (from 0.6556 to 0.6576) at the cost of longer training time.
However, tuning max depth = 10 resulted in the highest accuracy (85.93%) and precision (80.85%), though it came at the expense of recall (55.38%), showing the model became more conservative and potentially overfitted to stronger splits.
For SVM, the radial basis function (rbf) kernel outperformed the linear and polynomial kernels in both accuracy and F1 score, achieving a balanced result of 0.6768 F1 score, indicating its ability to handle non-linear decision boundaries more effectively.

Additionally, tuning the regularization parameter C had limited impact, reinforcing the robustness of the SVM model once the kernel was well selected. In the MLP

models, varying the architecture and activation function demonstrated noticeable changes. Deeper architectures such as (64, 32, 16) led to the highest recall (61.69%) and F1 score (0.6398), suggesting improved learning of complex patterns.
However, this came with increased computational cost. The tanh activation function offered similar F1 performance with slightly reduced recall compared to relu, highlighting the importance of activation selection. Overall, the tuning phase demonstrated that even small changes in model configuration can significantly affect precision, recall, and training efficiency, and that optimal hyperparameters are highly model-specific.

4.2 Medium Search (Fast Grid Search) Tuning Phase

```
   # Define a small parameter grid
param_grid_rf_medium = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy']
}
```

Listing 4.4: The paramameter for RF model by Grid Search

```
   # Define a small parameter grid
param_grid_svm_medium = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
```

Listing 4.5: The paramameter for SVM model by Grid Search

```
# Define a small parameter grid
param_grid_mlp_medium = {
    'hidden_layer_sizes': [(100,), (64, 32)],
    'activation': ['relu', 'tanh'],
    'learning_rate': ['constant', 'adaptive'],
    'max_iter': [500]
}
```

Listing 4.6: The paramameter for MLP model by Grid Search

The best result after run it as following :

Best Random Forest (Medium Search): 'criterion': 'gini', 'max depth': 20, 'min samples split': 2, 'n estimators': 100

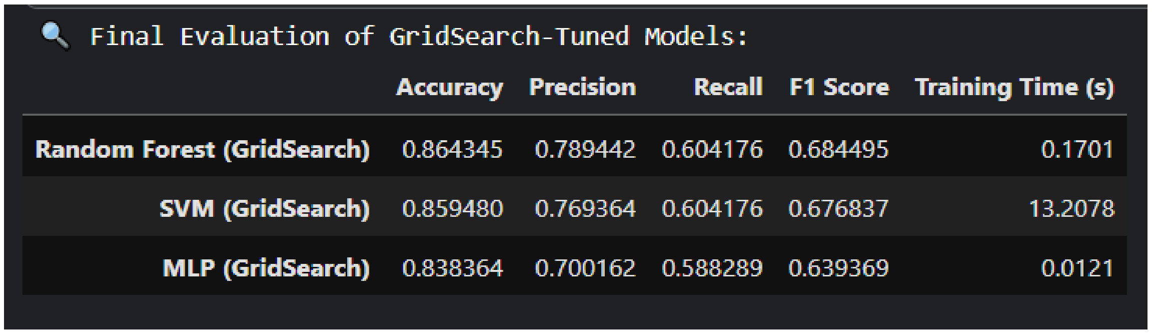
Best SVM (Medium Search): 'C': 1, 'gamma': 'scale', 'kernel': 'rbf'

Best MLP (Medium Search): 'activation': 'tanh', 'hidden layer sizes': (100,), 'learning rate': 'constant', 'max iter': 500

4.3 Very Deep Search (Advanced Big Grid)

It takes a lot of time but it is more efficent way to get the best parameters for 3 models RF, SVM, MLP.
Because my Task not need this complexity ,and data not too larg, Also it take a lot of time i not complete Execution

4.4 Evaluate Grid Search Best Models



🔍 Final Evaluation of GridSearch-Tuned Models:

	Accuracy	Precision	Recall	F1 Score	Training Time (s)
Random Forest (GridSearch)	0.864345	0.789442	0.604176	0.684495	0.1701
SVM (GridSearch)	0.859480	0.769364	0.604176	0.676837	13.2078
MLP (GridSearch)	0.838364	0.700162	0.588289	0.639369	0.0121

Figure 4.2: Final Evaluation of Grid Search-Tuned Models:

The final evaluation of the three classifiers—Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP)—after applying GridSearchCV hyperparameter tuning demonstrates the impact of systematic parameter optimization.

The Random Forest model achieved the highest accuracy (86.43%) and F1 score (0.6845), while also maintaining an extremely fast training time (0.17 seconds), reaffirming its efficiency and robustness. The SVM model closely followed with an accuracy of 85.95% and an F1 score of 0.6768, but required significantly more computational time (13.21 seconds), which is expected given the nature of kernel computations in SVM.

Although the MLP model exhibited the lowest overall performance in terms of accuracy and recall, it surprisingly had the shortest training time (0.012 seconds), likely due to limited iteration depth during optimization. These results confirm that Random Forest, when properly tuned, offers the best balance of accuracy, precision, and efficiency for this dataset.

SVM remains a strong alternative when higher precision is desired, while MLP may require deeper tuning or more data to reach its full potential.

4.5 Compined the manually tuned also grid tuned results, also models without tuning comparison

Table 4.1: Full Comparison: Baseline, Manually Tuned, and GridSearch-Tuned Models

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Training Time (s)**
RF (Baseline)	0.8437	0.7073	0.6110	0.6556	4.0170
SVM (Baseline)	0.8513	0.7532	0.5792	0.6549	67.9740
MLP (Baseline)	0.8407	0.7290	0.5506	0.6274	73.0474
RF (n=100)	0.8437	0.7073	0.6110	0.6556	3.9359
RF (n=200)	0.8437	0.7077	0.6101	0.6553	8.2072
RF (n=300)	0.8447	0.7100	0.6123	0.6576	12.0715
RF (max_depth=10)	0.8594	0.8085	0.5538	0.6573	1.7935
SVM (linear)	0.8513	0.7532	0.5792	0.6549	56.0893
SVM (rbf)	0.8595	0.7694	0.6042	0.6768	61.3179
SVM (poly)	0.8559	0.7650	0.5897	0.6660	45.8690
SVM (C=0.5)	0.8512	0.7531	0.5788	0.6545	50.5758
SVM (C=2.0)	0.8513	0.7532	0.5792	0.6549	69.7906
MLP (100,)	0.8407	0.7290	0.5506	0.6274	64.6472
MLP (128, 64)	0.8203	0.6348	0.6178	0.6262	96.8307
MLP (64,32,16)	0.8308	0.6645	0.6169	0.6398	63.7717
MLP (100,) tanh	0.8384	0.7002	0.5883	0.6394	83.1920
RF (GridSearch)	**0.8643**	**0.7894**	**0.6042**	**0.6845**	**0.1701**
SVM (GridSearch)	0.8595	0.7694	0.6042	0.6768	13.2078
MLP (GridSearch)	0.8384	0.7002	0.5883	0.6394	0.0121

The tuning phase demonstrates the considerable influence hyperparameters have on model performance.
As evident in the table, manually adjusting parameters such as the number of estimators in Random Forest or the kernel and regularization strength in SVM led to measurable performance shifts. For instance, increasing the number of estimators in the Random Forest slightly improved F1 score, while limiting the maximum depth notably increased precision at the cost of recall.
Similarly, switching the SVM kernel from linear to RBF yielded significant precision and F1 score gains. The MLP models showed high sensitivity to architectural changes—hidden layer sizes and activation functions caused noticeable fluctuations in recall and training time.
Ultimately, GridSearchCV delivered the most optimal configurations, especially for the Random Forest, which outperformed all models with an accuracy of 86.43% and an F1 score of 0.6845, while also being extremely efficient. These findings confirm that proper parameter tuning is essential to unlock each model's full potential.

5 Conclusion

This study systematically evaluated and compared three prominent classification algorithms—Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP)—on a real-world tabular dataset. Through comprehensive data preprocessing, feature engineering, visualization, and hyperparameter tuning, we assessed each model's performance based on accuracy, precision, recall, F1 score, and training time.

Our findings revealed that the Random Forest model, particularly when tuned using GridSearchCV, delivered the best overall performance. It achieved the highest accuracy (86.4%) with the lowest training time (0.17 seconds), making it the most efficient model.
SVM, especially with the RBF kernel, also showed strong performance in terms of precision and accuracy, though at a higher computational cost. MLP models demonstrated reasonable predictive capabilities but required more tuning and significantly longer training time.
Moreover, the analysis confirmed that preprocessing and scaling had little impact on tree-based models like Random Forest, but were crucial for SVM and MLP. Hyperparameter tuning significantly improved performance across all models, reinforcing the importance of careful parameter selection in machine learning workflows.
In conclusion, Random Forest proved to be the most effective model for this classification task, offering an excellent trade-off between predictive accuracy and computational efficiency. Nonetheless, SVM and MLP remain valuable options depending on the dataset size, complexity, and resource availability.