ML & DL : Experiment - 3

Aim:

Apply Decision Tree and Random Forest for classification tasks.

Dataset Description:

The Adult Census Income Dataset, often referred to as the "UCI Adult" or "Census Income" dataset, is a classic multivariate dataset extracted from the 1994 United States Census Bureau database. It is widely considered a gold standard for intermediate classification tasks and algorithmic fairness testing. Unlike perfectly balanced academic datasets, this one reflects the socioeconomic reality of the 1990s, featuring a significant class imbalance (roughly 3:1), which makes it a perfect benchmark for testing the pruning capabilities of Decision Trees and the variance-reduction power of Random Forests. The dataset aims to predict whether an individual's annual income exceeds $50,000 based on a diverse set of demographic and financial indicators. The file type used for this dataset is a CSV (Comma Separated Values) file. The dataset consists of 15 columns, some of which are:

Column Name	Data Type	Description
Age	Numerical (Integer)	Age of the individual
Workclass	Categorical (String)	Type of employer or work classification (e.g., Private, Self-emp-not-inc, Federal-gov)
Education	Categorical (String)	Highest level of education achieved (e.g., Bachelors, Doctorate, HS-grad)
Marital-status	Categorical (String)	Individual’s current marital or relationship status

Dataset Source: https://www.kaggle.com/datasets/uciml/adult-census-income

---

Decision Tree

Theory:

A Decision Tree is a non-parametric supervised learning algorithm used for both classification and regression tasks. It is structured like a flowchart, where each internal node represents a "test" or "split" on an attribute (feature), each branch represents the outcome of the test, and each leaf node represents a class label (in classification) or a continuous value (in regression). The goal of a decision tree is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Unlike linear models that use a continuous global equation, a Decision Tree partitions the data into smaller and smaller subsets, creating a "divide and conquer" strategy to handle complex, non-linear relationships. In classification, Entropy measures the amount of disorder or uncertainty in a dataset. For a dataset with c classes, the Entropy is:

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Where p_i is the proportion of samples belonging to class i. Information Gain (IG) is then calculated to see how much entropy is reduced after splitting on a specific feature S:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

The Gini Index is another attribute selection measure used in Decision Tree Induction. Instead of entropy, it measures the impurity (or "mixing") of a dataset. The lower the Gini Index, the purer the dataset is. In practice, the attribute that produces splits with the lowest Gini Index (i.e., highest purity) is chosen.
Then you take the weighted average of all the values of a particular attribute. The lower that value is, the lower the better is the purity of the splitted data set.

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

$$Gini_{split}(S, A) = \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Gini(S_v)$$

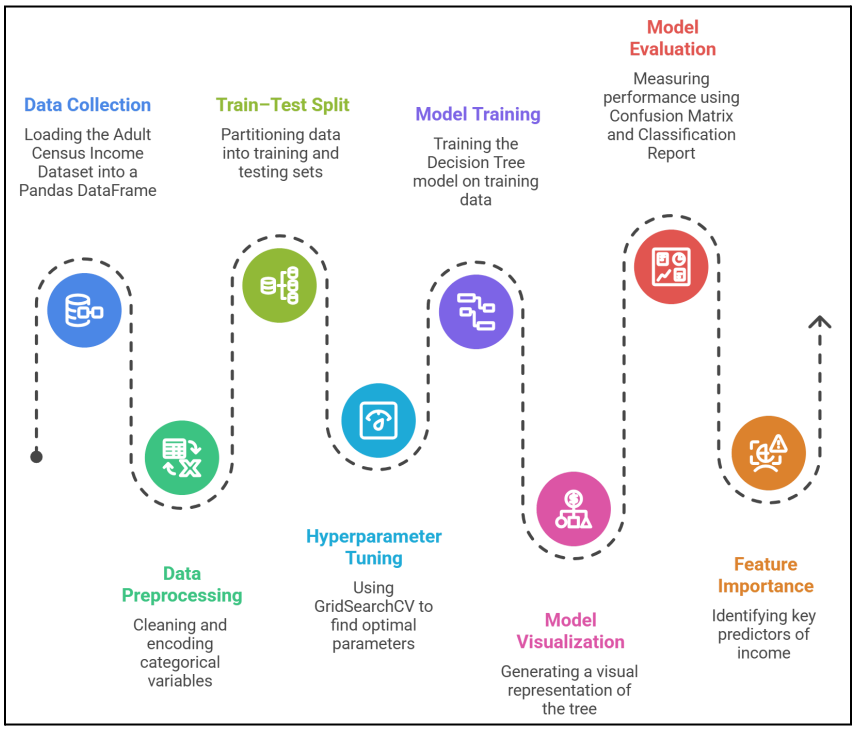
Note that this attribute selection measure is not used in ID3. It is used in CART.

Limitations:

1. **Overfitting (High Variance):** Decision Trees are highly prone to overfitting, especially when they are allowed to grow deep without constraints. The algorithm continues to split nodes to achieve maximum purity, which often leads to the model "memorizing" the noise and outliers in the training data rather than learning the underlying general pattern. When a tree becomes too complex, it creates very specific rules that only apply to the training set. This results in high accuracy during training but poor performance on new, unseen data, as the model fails to generalize the logic it has learned.
2. **Instability (High Sensitivity to Data Changes):** One of the primary weaknesses of a single Decision Tree is its instability. Since the algorithm uses a "greedy" approach—making the best possible split at the current node without looking ahead—even a small change in the data can result in a completely different tree structure. If you remove just a few rows from the Adult Census dataset, the root node might change from "Education" to "Age," which then cascades down to change every subsequent branch and leaf. This lack of robustness makes individual trees less reliable than ensemble methods.
3. **Difficulty Capturing Linear Relationships:** While Decision Trees are excellent at capturing non-linear "step" patterns, they are surprisingly inefficient at modeling simple linear relationships. Since a tree splits data into axis-aligned rectangles, it must use many small "stairs" or splits to approximate a diagonal line.

Workflow:

1. **Data Collection:** The Adult Census Income dataset from the UCI Machine Learning Repository is loaded into a Pandas DataFrame. It includes 14 demographic and socioeconomic features used to predict whether an individual earns over $50,000 annually.
2. **Data Preprocessing:** Placeholder values such as '?' are replaced with nulls and removed. Categorical features (e.g., workclass, education, sex) are converted to numerical form using Label Encoding for compatibility with the Decision Tree model.
3. **Train–Test Split:** The cleaned dataset is split into 80% training and 20% testing data to evaluate model performance on unseen samples.
4. **Hyperparameter Tuning with Grid Search:** GridSearchCV is used to reduce overfitting by testing combinations of parameters like max_depth, min_samples_split, and criterion (Gini vs. Entropy).
5. **Model Training and Visualization:** The best-performing Decision Tree is trained on the full training set, and a truncated visualization of the tree shows key decision rules such as the influence of marital status or capital gain.
6. **Model Evaluation:** A Confusion Matrix and Classification Report (precision, recall, F1-score) assess performance, while a Feature Importance plot highlights the most influential variables.
7. **Conclusion:** The Decision Tree offers an interpretable approach to income prediction, with strong predictors like education and age. Performance could be further improved using ensemble methods such as Random Forest or XGBoost.



Performance Analysis:

TThe performance of the optimized Decision Tree model on the Adult Census Income dataset is evaluated using Accuracy, a Confusion Matrix, and class-specific metrics like Precision and Recall.

1. **Overall Accuracy:** The model achieves an accuracy of 84.63%. This indicates that the Decision Tree correctly predicts an individual's income category in roughly 8.5 out of 10 cases. While this is a solid baseline for a single model, accuracy can be misleading in this dataset due to class imbalance (there are significantly more people earning <= 50K than >50K).

2. **Class-Specific Performance (Precision & Recall):** To understand how the model handles the two different income groups, we look at the classification report:
    a. Group 0 (<=50K): The model performs exceptionally well here, with an F1-score of 0.90. This is expected, as the model has more data to learn from for this category.
    b. Group 1 (>50K): Performance drops for the higher income bracket, with an F1-score of 0.68.
        i. Recall (0.66): This means the model captures only 66% of high earners, missing about 34% of them (False Negatives).
        ii. Precision (0.71): This indicates that when the model predicts someone earns >50K, it is correct about 71% of the time.
3. **Confusion Matrix Breakdown:** The confusion matrix provides a visual count of the model's hits and misses:
    a. True Negatives (4122): The model is highly reliable at identifying those earning <=50K.
    b. True Positives (984): The model successfully identified nearly a thousand high earners.
    c. False Negatives (516): These are individuals who actually earn >50K but were incorrectly labeled as low earners. This "leakage" is common in Decision Trees when branches are pruned to prevent overfitting.
    d. False Positives (411): These are low earners incorrectly flagged as high earners.

Hyperparameter Tuning:

Unlike Standard Linear Regression, which has a fixed mathematical solution, a Decision Tree is highly flexible and will grow indefinitely until it perfectly classifies every training point—a state known as overfitting. Hyperparameter tuning acts as the "constraint system" that stops the tree from becoming too complex. The code utilizes GridSearchCV, which performs an exhaustive search through a manually specified subset of the hyperparameter space.

- **Search Strategy:** It constructs a model for every possible combination of the parameters defined in param_grid.
- **Cross-Validation (cv=5):** To ensure the results aren't due to luck, it splits the training data into 5 folds. It trains on 4 and validates on 1, rotating this process 5 times for every single parameter combination.

The code tunes four specific "knobs" that control the tree's architecture:

- **criterion ('gini' vs 'entropy'):** This determines the mathematical formula used to calculate "impurity" or how mixed the classes are at a specific node. While Gini is usually faster to calculate, Entropy can sometimes produce more balanced trees.
- **max_depth:** This is the most impactful setting. It limits how many "levels" the tree can have. Without a limit (None), the tree grows until all leaves are pure, which almost always leads to overfitting on the Adult Census dataset.
- **min_samples_split:** This sets the minimum number of data points required to even consider splitting a node. Increasing this number prevents the model from creating rules based on very small, statistically insignificant groups of people.
- **min_samples_leaf:** This ensures that every final "leaf" (prediction) is based on at least a certain number of samples. It smoothens the model by preventing it from making a decision based on just one or two outliers in the census data.

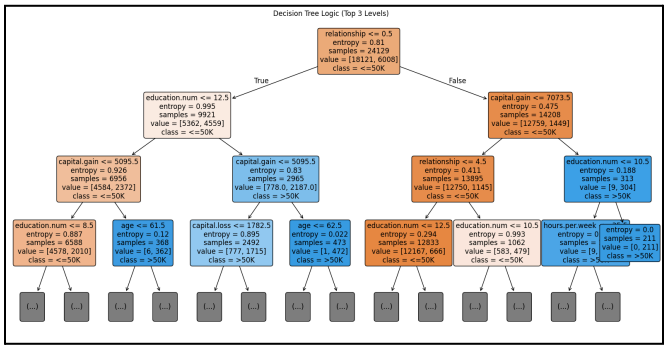
Code & Output:

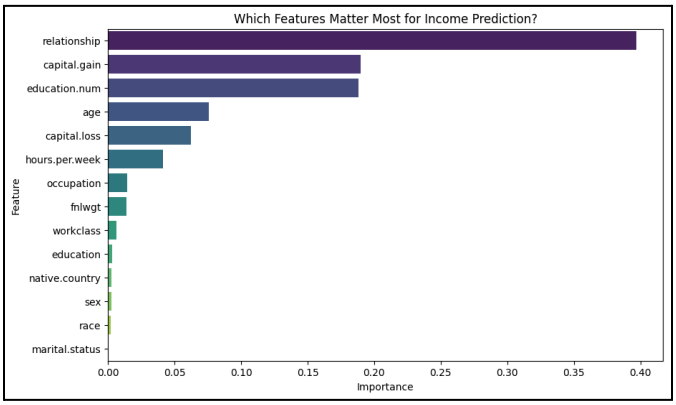
```
import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.tree import
DecisionTreeClassifier
from sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import
accuracy_score,
classification_report

df = pd.read_csv('adult.csv')

df.replace('?', np.nan,
inplace=True)
df.dropna(inplace=True)

le = LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] =
le.fit_transform(df[col])

X = df.drop('income', axis=1)
y = df['income']
X_train, X_test, y_train, y_test =
train_test_split(X, y,
test_size=0.2, random_state=42)

dtree =
DecisionTreeClassifier(random_state=
42)

# HYPERPARAMETERS
param_grid = {
    'criterion': ['gini',
'entropy'],
    'max_depth': [None, 5, 10, 15,
20],
    'min_samples_split': [2, 10,
20],
    'min_samples_leaf': [1, 5, 10]
}

grid_search =
GridSearchCV(estimator=dtree,
param_grid=param_grid, cv=5,
n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

print(f"Best Parameters:
{grid_search.best_params_}")
print(f"Best Cross-Validation Score:
{grid_search.best_score_:.4f}")

best_model =
grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print("\n--- Final Model Evaluation
---")
print(f"Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test,
y_pred))

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
# Plotting the first 3 levels for
readability
plot_tree(best_model,
          max_depth=3,
          feature_names=X.columns,
          class_names=['⩽50K',
'>50K'],
          filled=True,
          rounded=True,
          fontsize=12)
plt.title("Decision Tree Logic (Top
3 Levels)")
plt.show()
import seaborn as sns
# Get feature importances from the
best model
importances =
best_model.feature_importances_
feature_names = X.columns
feature_importance_df =
pd.DataFrame({'Feature':
```

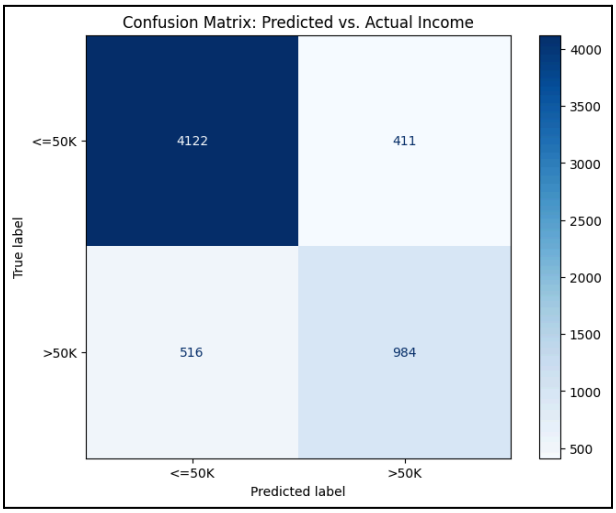
```
feature_names, 'Importance':
importances})
feature_importance_df =
feature_importance_df.sort_values(by
='Importance', ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance',
y='Feature',
data=feature_importance_df,
palette='viridis')
plt.title('Which Features Matter
Most for Income Prediction?')
plt.show()
from sklearn.metrics import
confusion_matrix,
ConfusionMatrixDisplay
```

```
cm = confusion_matrix(y_test,
y_pred)
disp =
ConfusionMatrixDisplay(confusion_mat
rix=cm, display_labels=['⩽50K',
'>50K'])
fig, ax = plt.subplots(figsize=(8,
6))
disp.plot(cmap='Blues', ax=ax)
plt.title('Confusion Matrix:
Predicted vs. Actual Income')
plt.show()
```

```
--- Final Model Evaluation ---
Accuracy: 0.8463

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.91      0.90      4533
           1       0.71      0.66      0.68      1500

    accuracy                           0.85      6033
   macro avg       0.80      0.78      0.79      6033
weighted avg       0.84      0.85      0.84      6033
```







Conclusion:

The Decision Tree model achieved a solid accuracy of 84.63% on the Adult Census Income dataset after proper preprocessing and hyperparameter tuning. It performed strongly on the majority class while showing moderate recall for high-income individuals due to class imbalance. The model's interpretability through tree structure and feature importance is a key advantage. However, its limitations suggest that ensemble methods like Random Forests can further improve performance and stability.

Random Forest

Theory:

A Random Forest is an ensemble learning method used for both classification and regression that operates by constructing a multitude of decision trees during training. It is based on the principle of Bootstrap Aggregating, or "Bagging." While a single Decision Tree is prone to high variance and overfitting (memorizing the noise in the data), a Random Forest reduces this variance by averaging the results of many diverse trees. The "Random" in Random Forest comes from two levels of stochasticity:

- **Bootstrap Sampling:** Each individual tree in the forest is trained on a random sample of the data, drawn with replacement (a bootstrap sample).
- **Feature Randomness:** When splitting a node, instead of searching for the best feature among all available features, the algorithm selects a random subset of features. This ensures that the trees are de-correlated; even if one feature is a very strong predictor, it won't be used in every single tree, allowing other patterns in the data to be discovered.

For classification, the final prediction is determined by Majority Voting (the class predicted by the most trees). For regression, the final prediction is the average of the individual tree outputs. For regression the formula used is:

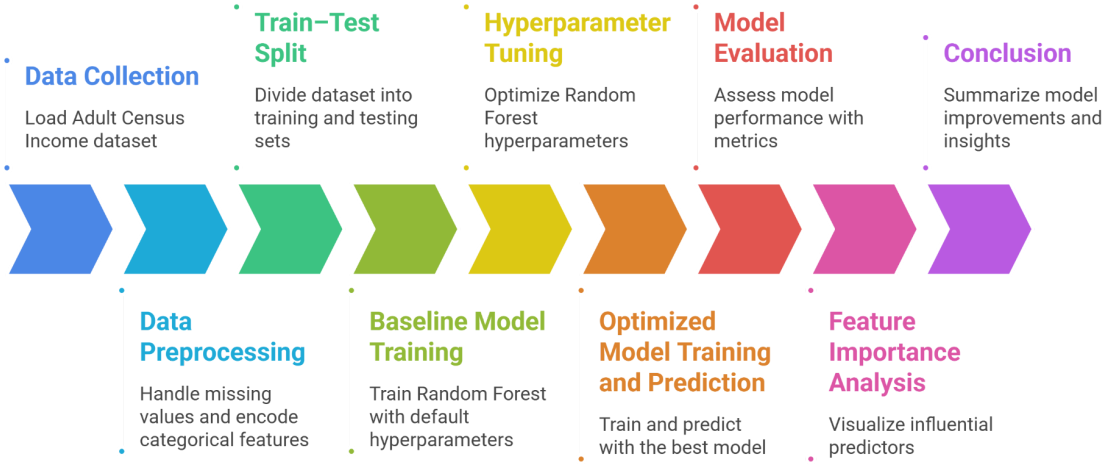
$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

Limitations:

1. **Reduced Interpretability (Black-Box Nature):** While a single Decision Tree is easy to visualize and interpret, a Random Forest combines the predictions of hundreds or even thousands of trees. This makes it difficult to clearly understand why the model made a particular prediction. The overall decision logic is distributed across many trees, so extracting simple, human-readable rules becomes impractical, which is a drawback in domains where explainability is critical.
2. **Higher Computational Cost (Time and Memory):** Random Forests are significantly more computationally expensive than individual Decision Trees. Training involves building many trees on different bootstrap samples, which increases both training time and memory usage. This can be inefficient for very large datasets or environments with limited computational resources, especially compared to simpler models.
3. **Bias Toward Features with More Levels:** Random Forests can favor features that have many unique values (such as continuous variables or high-cardinality categorical features). These features are more likely to produce splits that reduce impurity, even if they are not truly more informative, which can introduce subtle bias in feature importance rankings.
4. **Limited Extrapolation Ability:** Like Decision Trees, Random Forests are poor at extrapolating beyond the range of the training data. Since predictions are based on averages of observed outcomes in leaf nodes, the model cannot reliably predict values outside the patterns it has already seen, making it less suitable for tasks requiring strong extrapolation behavior.

Workflow:

1. **Data Collection:** The Adult Census Income dataset is loaded from a CSV file into a Pandas DataFrame. It contains demographic and socioeconomic features used to predict whether an individual earns more than $50K annually.
2. **Data Preprocessing:** Missing values represented by '?' are replaced with NaN and removed. All categorical features are encoded into numerical form using Label Encoding to make them compatible with machine learning algorithms.
3. **Train–Test Split:** The dataset is split into training (80%) and testing (20%) sets using a fixed random state to ensure reproducibility.
4. **Baseline Model Training:** A baseline Random Forest Classifier is trained using default hyperparameters. Its accuracy on the test set is computed to serve as a reference point for model improvement.
5. **Hyperparameter Tuning:** RandomizedSearchCV is applied to tune key Random Forest hyperparameters such as the number of trees, maximum depth, minimum samples for splits and leaves, and bootstrap strategy. Three-fold cross-validation is used to identify the best model configuration.
6. **Optimized Model Training and Prediction:** The best Random Forest model obtained from hyperparameter tuning is trained on the full training set and used to generate predictions on the test data.
7. **Model Evaluation:** Model performance is evaluated using accuracy, a detailed classification report (precision, recall, F1-score), and a confusion matrix to analyze prediction errors across income classes.
8. **Feature Importance Analysis:** Feature importance scores from the optimized Random Forest model are extracted and visualized using a bar plot to identify the most influential predictors of income.
9. **Conclusion:** The optimized Random Forest model outperforms the baseline by leveraging hyperparameter tuning, achieving improved classification performance while also providing insights into the most important features influencing income prediction.



Performance Analysis:

The performance of the implemented Random Forest classification model is evaluated using Accuracy, Precision, Recall, and the F1-Score. These metrics together assess the model's ability to correctly categorize individuals into income brackets (<=50K or >50K) based on demographic and socio-economic

features. While accuracy provides a general overview, the class-specific metrics reveal how the model handles the inherent imbalance in the census data.

- **Accuracy Analysis:** The model achieved a tuned accuracy of approximately 0.8589 (85.9%). This indicates that the Random Forest correctly predicted the income category for nearly 86 out of every 100 individuals in the test set. Given the real-world complexity and "noise" of the 1994 Census data, this performance is quite high, significantly outperforming a baseline "majority-class" predictor which would only achieve around 75% accuracy.
- **Precision and Recall Analysis:** The model shows a high precision for the majority class (0.88), but more importantly, a precision of 0.77 for the high-income class (1). This means when the model predicts someone earns >$50K, it is correct 77% of the time. However, the Recall for class 1 is lower at 0.62, indicating that the model captures about 62% of all actual high earners in the dataset. This suggests the model is somewhat "conservative"—it prefers to be sure before labeling someone as a high earner, likely due to the class imbalance where high earners are less frequent.
- **F1-Score Analysis:** The F1-Score for the >$50K class is 0.69, which is the harmonic mean of precision and recall. This score is a more reliable indicator of success for the minority class than simple accuracy. A score of 0.69 demonstrates that while the model has a strong grasp of the features that lead to higher income, there is still some difficulty in distinguishing "borderline" cases where individuals have high education or age but may not yet have reached the $50K threshold.

Hyperparameter Tuning:

Hyperparameter tuning was performed to optimize the Random Forest model and ensure it achieves the best possible balance between capturing complex patterns and generalizing to new data. Instead of relying on the default settings, which are often prone to overfitting on high-depth trees, RandomizedSearchCV was utilized to search through a predefined grid of values. This approach employs k-fold cross-validation to evaluate different combinations of settings, ensuring that the final model is robust and not just tailored to the specific noise within the training set. Hyperparameters Tuned: Several key parameters that govern the growth and diversity of the forest were evaluated:

- n_estimators: The number of trees in the forest. More trees generally increase stability but also increase computational time.
- max_depth: The maximum number of levels in each tree. Limiting this prevents the model from becoming too complex and over-learning the training data.
- min_samples_split: The minimum number of samples required to split an internal node, which helps control the sensitivity of the tree splits.
- min_samples_leaf: The minimum number of samples required to be at a leaf node, providing a smoothing effect on the decision boundaries.
- bootstrap: Whether to use random subsets of the data for each tree, which is essential for reducing variance in the ensemble.

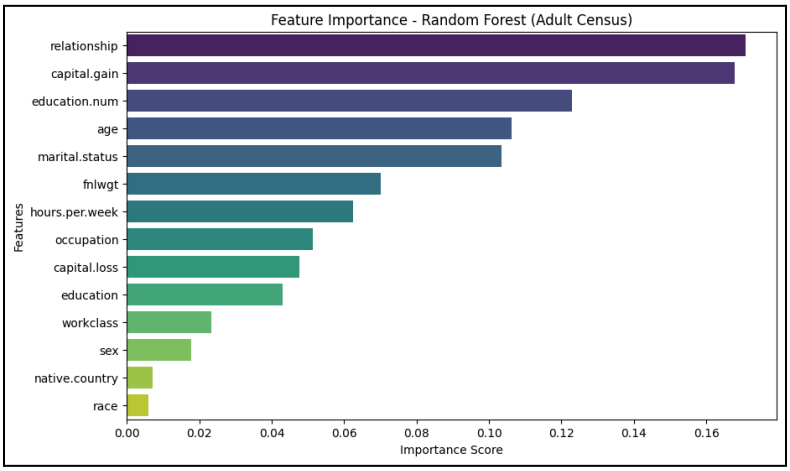
A search across these parameters was conducted using 3-fold cross-validation to identify the most effective configuration.

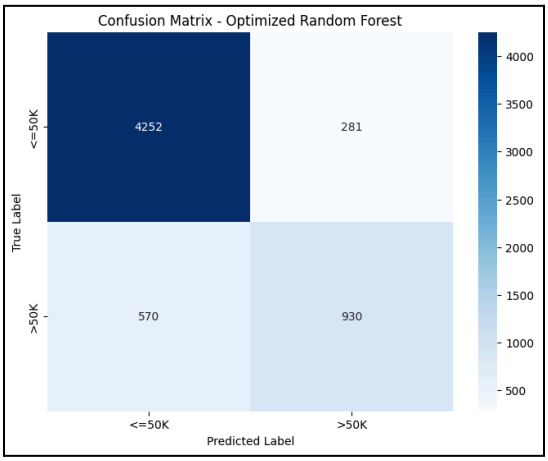
Code & Output:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split, RandomizedSearchCV
from sklearn.ensemble import
RandomForestClassifier
```

```
from sklearn.metrics import
classification_report,
accuracy_score
from sklearn.preprocessing import
LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import
confusion_matrix

df = pd.read_csv('adult.csv')

df.replace('?', np.nan,
inplace=True)
df.dropna(inplace=True)

le = LabelEncoder()
for col in
df.select_dtypes(include=['object'])
.columns:
    df[col] =
le.fit_transform(df[col])

X = df.drop('income', axis=1)
y = df['income']
X_train, X_test, y_train, y_test =
train_test_split(X, y,
test_size=0.2, random_state=42)

rf_base =
RandomForestClassifier(random_state=
42)
rf_base.fit(X_train, y_train)
print(f"Baseline RF Accuracy:
{accuracy_score(y_test,
rf_base.predict(X_test)):.4f}")

param_dist = {
    'n_estimators': [100, 200, 300],
# Number of trees in the forest
    'max_depth': [10, 20, 30, None],
# Max depth of each tree
    'min_samples_split': [2, 5, 10],
# Min samples to split a node
    'min_samples_leaf': [1, 2, 4],
# Min samples at a leaf node
    'bootstrap': [True, False]
# Method of selecting samples for
training
}

print("\nTuning Random Forest (this
may take a minute)...")
rf_random = RandomizedSearchCV(

estimator=RandomForestClassifier(ran
dom_state=42),
    param_distributions=param_dist,
    n_iter=10,        # Number of
parameter settings sampled
    cv=3,             # 3-fold
cross-validation
    random_state=42,
    n_jobs=-1
)

rf_random.fit(X_train, y_train)

best_rf = rf_random.best_estimator_
y_pred = best_rf.predict(X_test)

print("\n--- Optimized Random Forest
Results ---")
print(f"Best Parameters:
{rf_random.best_params_}")
print(f"Tuned Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test,
y_pred))

importances =
best_rf.feature_importances_
feature_names = X.columns
feature_importance_df =
pd.DataFrame({'Feature':
feature_names, 'Importance':
importances})
feature_importance_df =
feature_importance_df.sort_values(by
='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance',
y='Feature',
data=feature_importance_df,
palette='viridis')
```

```
plt.title('Feature Importance -
Random Forest (Adult Census)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()

cm = confusion_matrix(y_test,
y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',
                xticklabels=['≤50K',
'>50K'],
                yticklabels=['≤50K',
'>50K'])
plt.title('Confusion Matrix -
Optimized Random Forest')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```





Conclusion:

The Random Forest classifier demonstrates strong performance in predicting income categories, achieving a tuned accuracy of approximately 86% on the Adult Census dataset. Hyperparameter tuning using RandomizedSearchCV significantly improves generalization by controlling model complexity and reducing overfitting. The model performs particularly well for the majority class, while maintaining reasonable precision–recall balance for the minority high-income class. Feature importance analysis further enhances interpretability by highlighting the most influential socioeconomic factors. Overall, Random Forest proves to be a robust and effective model for income classification on real-world census data.