Name: Brian Ryu

Computing ID: kfg2ec

```
# Sources used: ChatGPT to help with graphing and formatting with the models created, as well as the function creatio

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

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import ConfusionMatrixDisplay, precision_score, recall_score, accuracy_score
from sklearn.ensemble import RandomForestClassifier

!pip install lime
from lime import lime_tabular
from sklearn.inspection import permutation_importance

# Below is a new package needed for this lab
!pip install ucimlrepo
from ucimlrepo import fetch_ucirepo
```

```
Requirement already satisfied: lime in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (0.2.0.1)
Requirement already satisfied: matplotlib in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from l
Requirement already satisfied: numpy in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from lime)
Requirement already satisfied: scipy in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from lime)
Requirement already satisfied: tqdm in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from lime) (
Requirement already satisfied: scikit-learn>=0.18 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages
Requirement already satisfied: scikit-image>=0.12 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages
Requirement already satisfied: networkx>=3.0 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (fro
Requirement already satisfied: pillow>=10.1 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-pack
Requirement already satisfied: tifffile>=2022.8.12 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-package
Requirement already satisfied: packaging>=21 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (fro
Requirement already satisfied: lazy-loader>=0.4 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (
Requirement already satisfied: joblib>=1.2.0 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (fro
Requirement already satisfied: threadpoolctl>=3.1.0 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packag
Requirement already satisfied: contourpy>=1.0.1 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (
Requirement already satisfied: cycler>=0.10 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from
Requirement already satisfied: fonttools>=4.22.0 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages
Requirement already satisfied: kiwisolver>=1.3.1 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages
Requirement already satisfied: pyparsing>=3 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from
Requirement already satisfied: python-dateutil>=2.7 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packag
Requirement already satisfied: six>=1.5 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from pyt
Requirement already satisfied: ucimlrepo in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (0.0.7)
Requirement already satisfied: pandas>=1.0.0 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (fro
Requirement already satisfied: certifi>=2020.12.5 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages
Requirement already satisfied: numpy>=1.26.0 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (fro
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-pack
Requirement already satisfied: pytz>=2020.1 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from
Requirement already satisfied: tzdata>=2022.7 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (fr
Requirement already satisfied: six>=1.5 in /Users/brian/.pyenv/versions/3.13.7/lib/python3.13/site-packages (from pyt
```

Lab 3: Decision Trees and Random Forests (100 Points)

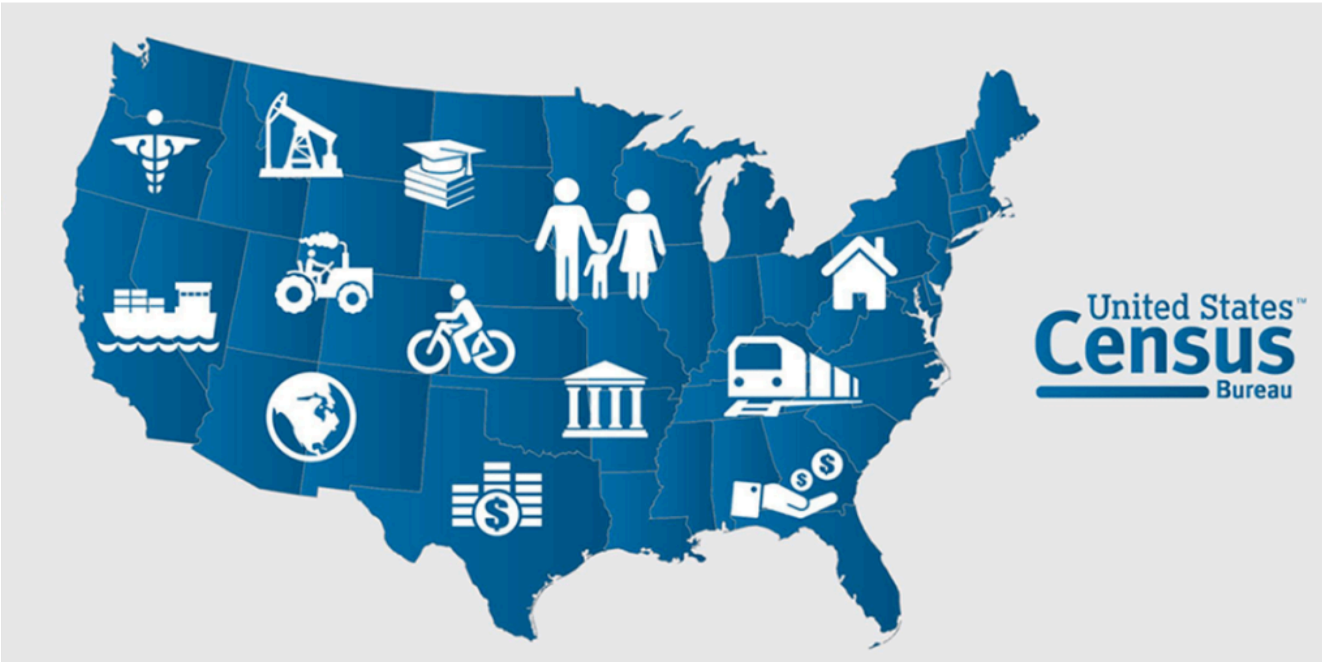
Due November 10th, 2025 at 11:59PM

Open in Colab

The goal of this lab is to optimize Decision Tree and Random Forest models using the provided dataset on census level data. Your goal is to build a Random Forest Classifier to be able to predict income levels above or below 50k.

The guidance this week is less prescriptive in terms of steps, so use the skills you have gained over the semester to build and evaluate your models. You will be graded on your model building, interpretation of the results and explanation of model selection. As always, you are welcome to rely on your classmates but submit your own code. Lastly, there are likely several correct approaches involving a variety of different conclusions, just make sure your conclusions are supported by your approach.

The dataset should be familiar as it's the census data, on 48,000+ individuals with a variety of variables and a target variable for above or below 50k in salary.



Look through the data dictionary at its source link: https://archive.ics.uci.edu/ml/datasets/Adult

Part 1: Data Preparation and EDA (15 points)

In a text cell, answer the following exploratory questions and support your observations with any code, if needed.

Question 1 (2 points):

Read in the features (X) as a Pandas DataFrame. Show the first 5 rows of the features. How many rows do you have?

```
# Fetch dataset
adult = fetch_ucirepo(id=2)

X = adult.data.features
y = adult.data.targets
```

```
X = pd.DataFrame(X)
display(X.head())
print("Rows:", X.shape[0])
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	h
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	
4	28	Private	338409	Bachelors	13	Married-	Prof-	Wife	Black	Female	0	0	

There is a total of... 48,842 rows in features.

Question 2 (2 points):

Are there any potential issues in the data or target that need to be corrected? Why are they issues? What specific method would you use to correct them and why?

Consider using code and reading the data description (https://archive.ics.uci.edu/dataset/2/adult) to explore:

- Assumptions and ranges of collected data
- Missing values (impute? drop?)
- Numerical data types represented as strings
- Encoding categorical data appropriately
- Normalization
- Standardization

You will not need to consider feature imbalances or sampling in part 1 or 2 of the lab.

```
X.info()
display(X.head())

for col in X.columns:
    if X[col].dtype == 'object':
        print(f"\nColumn: {col}")
        print(X[col].value_counts().head)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             48842 non-null  int64 
 1   workclass       47879 non-null  object
 2   fnlwgt          48842 non-null  int64 
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64 
 5   marital-status  48842 non-null  object
 6   occupation      47876 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             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  48568 non-null  object
dtypes: int64(6), object(8)
memory usage: 5.2+ MB
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0

```
Column: workclass
<bound method NDFrame.head of workclass
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
State-gov            1981
?                    1836
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: count, dtype: int64>

Column: education
<bound method NDFrame.head of education
HS-grad         15784
Some-college    10878
Bachelors        8025
Masters          2657
Assoc-voc        2061
11th             1812
Assoc-acdm       1601
10th             1389
7th-8th           955
Prof-school       834
9th               756
12th              657
Doctorate         594
5th-6th           509
1st-4th           247
Preschool          83
Name: count, dtype: int64>

Column: marital-status
<bound method NDFrame.head of marital-status
Married-civ-spouse       22379
Never-married            16117
Divorced                  6633
Separated                 1530
Widowed                   1518
Married-spouse-absent      628
Married-AF-spouse           37
Name: count, dtype: int64>

Column: occupation
```

```
Column: occupation
<bound method NDFrame.head of occupation
Prof-specialty       6172
Craft-repair         6112
Exec-managerial      6086
Adm-clerical         5611
Sales                5504
Other-service        4923
Machine-op-inspct    3022
Transport-moving     2355
Handlers-cleaners    2072
?                    1843
Farming-fishing      1490
Tech-support         1446
Protective-serv       983
Priv-house-serv       242
Armed-Forces           15
Name: count, dtype: int64>

Column: relationship
<bound method NDFrame.head of relationship
Husband           19716
Not-in-family     12583
Own-child          7581
Unmarried          5125
Wife               2331
Other-relative     1506
Name: count, dtype: int64>

Column: race
<bound method NDFrame.head of race
White                 41762
Black                  4685
Asian-Pac-Islander     1519
Amer-Indian-Eskimo      470
Other                   406
Name: count, dtype: int64>

Column: sex
<bound method NDFrame.head of sex
Male      32650
Female    16192
Name: count, dtype: int64>

Column: native-country
<bound method NDFrame.head of native-country
United-States                 43832
Mexico                          951
?                               583
Philippines                     295
Germany                         206
Puerto-Rico                     184
Canada                          182
El-Salvador                     155
India                           151
Cuba                            138
England                         127
China                           122
South                           115
Jamaica                         106
Italy                           105
Dominican-Republic              103
Japan                            92
Guatemala                        88
Poland                           87
Vietnam                          86
Columbia                         85
Haiti                            75
Portugal                         67
Taiwan                           65
Iran                             59
Greece                           49
Nicaragua                        49
Peru                             46
Ecuador                          45
France                           38
Ireland                          37
Hong                             30
Thailand                         30
Cambodia                         28
Trinadad&Tobago                  27
Laos                             23
Yugoslavia                       23
Outlying-US(Guam-USVI-etc)       23
Scotland                         21
```

```
Honduras                          20
Hungary                           10
Holand-Netherlands                 1
Name: count, dtype: int64>
```

Potential Issues: Potential issues I've identified are.... missing values that have been substituted by "?". When looking into columns such as workclass, there is a ? with 1836, which makes it difficult to see what this value is representing (does not use NaN). The way I would handle this would be to put NaN instead of "?" and then dropping the missing rows if the values are minimal such as less than 5%. Another issue could be to one-hot encode categorical variables because tree models require this type of action. Another potential issue to put down could be to look for possible outliers such as having an age of 1000 which is not realistic at all. For y, we can also clean this by making sure that there are no trailing peroids for target labels such as '>50K.'.

Question 3 (6 points):

Preprocess the data according to the issues and correction methods you've identified. Save the new features and target variable (if necessary) as X_clean and y_clean.

```
X = X.replace("?", np.nan)

X_clean = X.dropna()

X_clean = pd.get_dummies(X_clean, drop_first=True)

y_clean = y.apply(lambda s: s.str.strip().replace('.', ''))

display(y_clean.head())

display(X_clean.head())
```

	income
0	<=50K
1	<=50K
2	<=50K
3	<=50K
4	<=50K

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-no
0	39	77516	13	2174	0	40	False	False	False	
1	50	83311	13	0	0	13	False	False	False	
2	38	215646	9	0	0	40	False	True	False	
3	53	234721	7	0	0	40	False	True	False	
4	28	338409	13	0	0	40	False	True	False	

5 rows × 96 columns

Question 4 (5 points):

Create 2 versions of y_clean to create a new target response of whether income is above or below $50,000 for classification.

1. **y_clean_binary:** Recode y_clean to be 1 if the target is over or equal to 50,000
2. **y_clean_string:** Recode y_clean to be "Above or Equal" if the target is over or equal to 50,000 and "Below" if under.

```
if isinstance(y_clean, pd.DataFrame):
    y_clean = y_clean.iloc[:, 0]

y_clean = y_clean[X_clean.index]

y_clean_binary = y_clean.apply(lambda s: 1 if s == '>50K' else 0)
y_clean_string = y_clean.apply(lambda s: "Above or Equal" if s == '>50K' else "Below")
```

```
print(y_clean_binary.value_counts())

print(y_clean_string.value_counts())
```

```
income
0    37714
1     7508
Name: count, dtype: int64
income
Below            37714
Above or Equal     7508
Name: count, dtype: int64
```

Part 2: Decision Tree Pruning, Tuning and Evaluation (30 Points)

Question 1 (5 points):

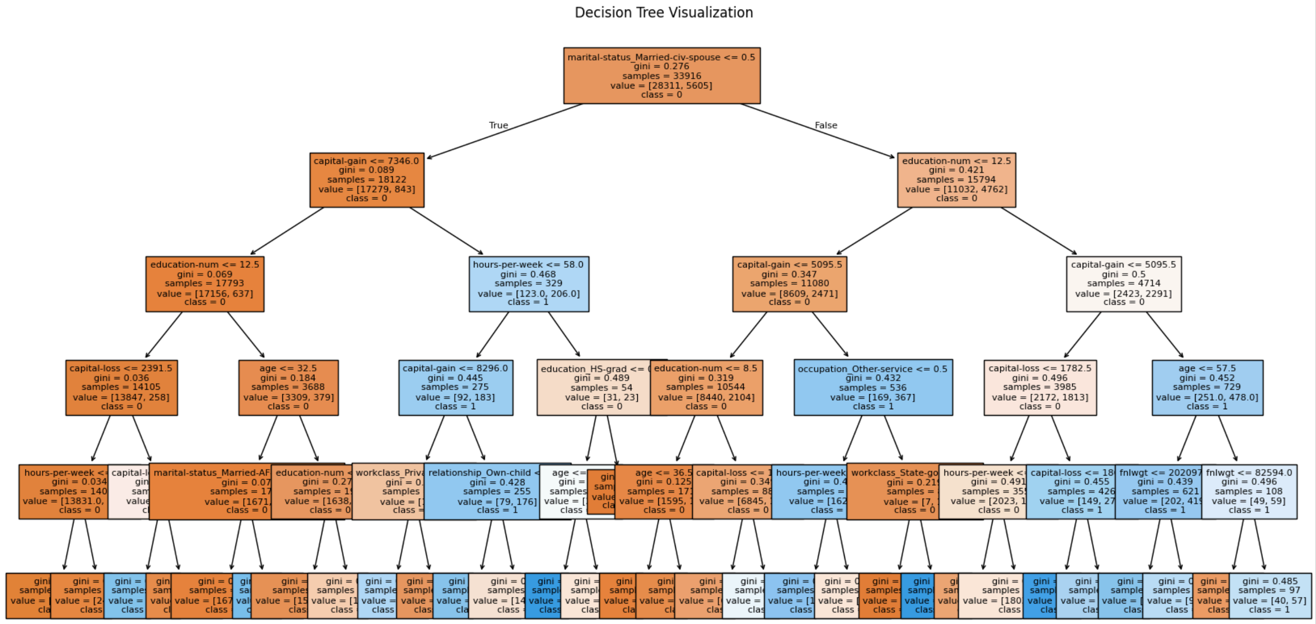
Create a function to take in a feature variable (X) and (y). In this function, do the following:

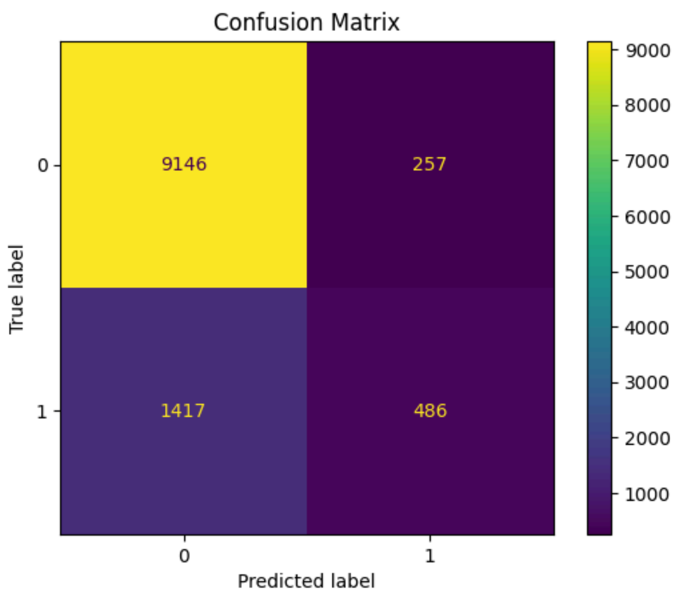
- Create a train test split with a random seed of 3001.
- Use a vanilla decision tree model to fit the model on the train set and predict on the test set.
- Print the precision, recall, and accuracy of the model after prediction.

Test your function on both y_clean_binary and y_clean_string. For any of the following questions, you may use whichever y_clean variable you'd like.

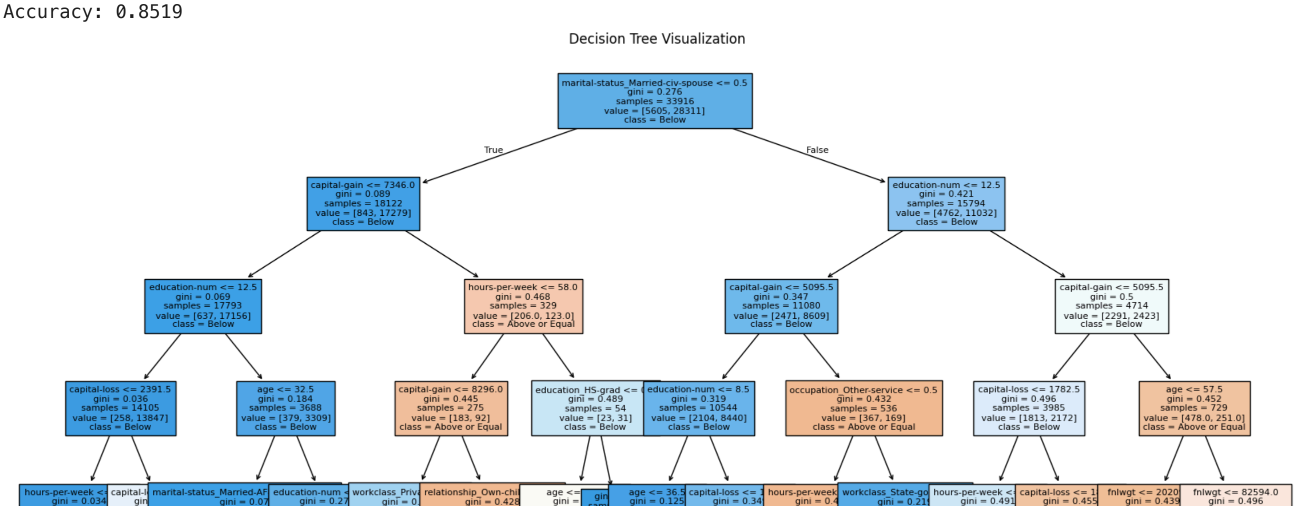
```
def eval_dtree(X, y, max_depth=None):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=3001)

    dt = DecisionTreeClassifier(random_state=3001, max_depth=max_depth)
    dt.fit(X_train, y_train)

    y_pred = dt.predict(X_test)

    if y.dtype.kind in ['O', 'U', 'S']:
        pos_label = "Above or Equal"
    else:
        pos_label = 1

    precision = precision_score(y_test, y_pred, pos_label=pos_label)
    recall = recall_score(y_test, y_pred, pos_label=pos_label)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"Accuracy: {accuracy:.4f}")

    plt.figure(figsize=(20,10))
    plot_tree(dt, feature_names=X.columns, class_names=[str(c) for c in dt.classes_],
              filled=True, fontsize=8, max_depth=5)
    plt.title("Decision Tree Visualization")
    plt.show()

    cm_display = ConfusionMatrixDisplay.from_estimator(dt, X_test, y_test, display_labels=[str(c) for c in dt.classe
    cm_display.ax_.set_title("Confusion Matrix")
    plt.show()

    return dt

print("Using y_clean_binary:")
eval_dtree(X_clean, y_clean_binary, max_depth=5)

print("\nUsing y_clean_string:")
eval_dtree(X_clean, y_clean_string, max_depth=5)

dt = eval_dtree(X_clean, y_clean_binary, max_depth=5)
```

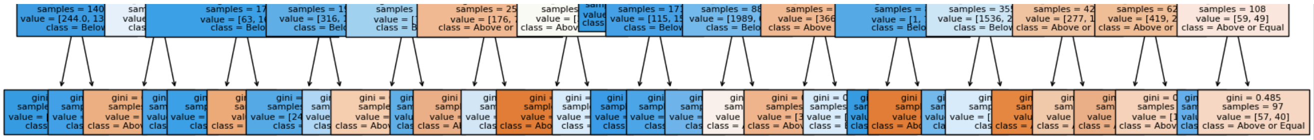
```
Using y_clean_binary:
Precision: 0.6541
Recall: 0.2554
Accuracy: 0.8519
```

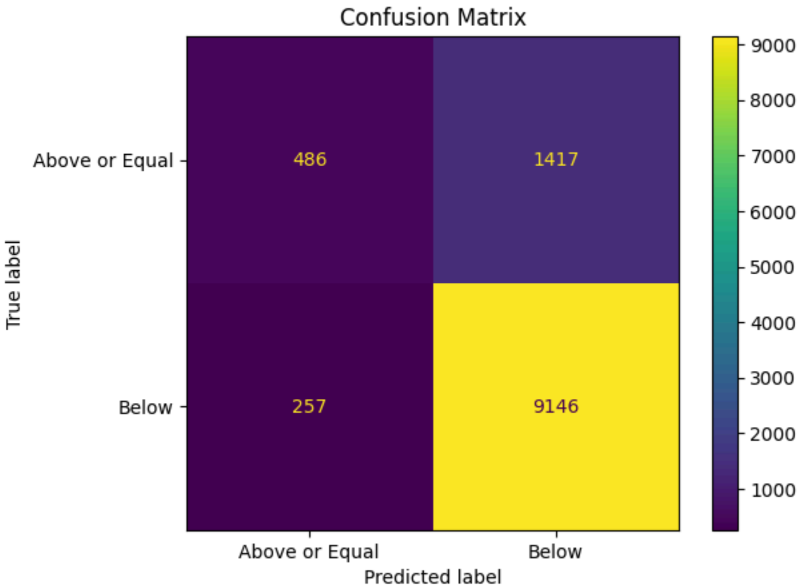




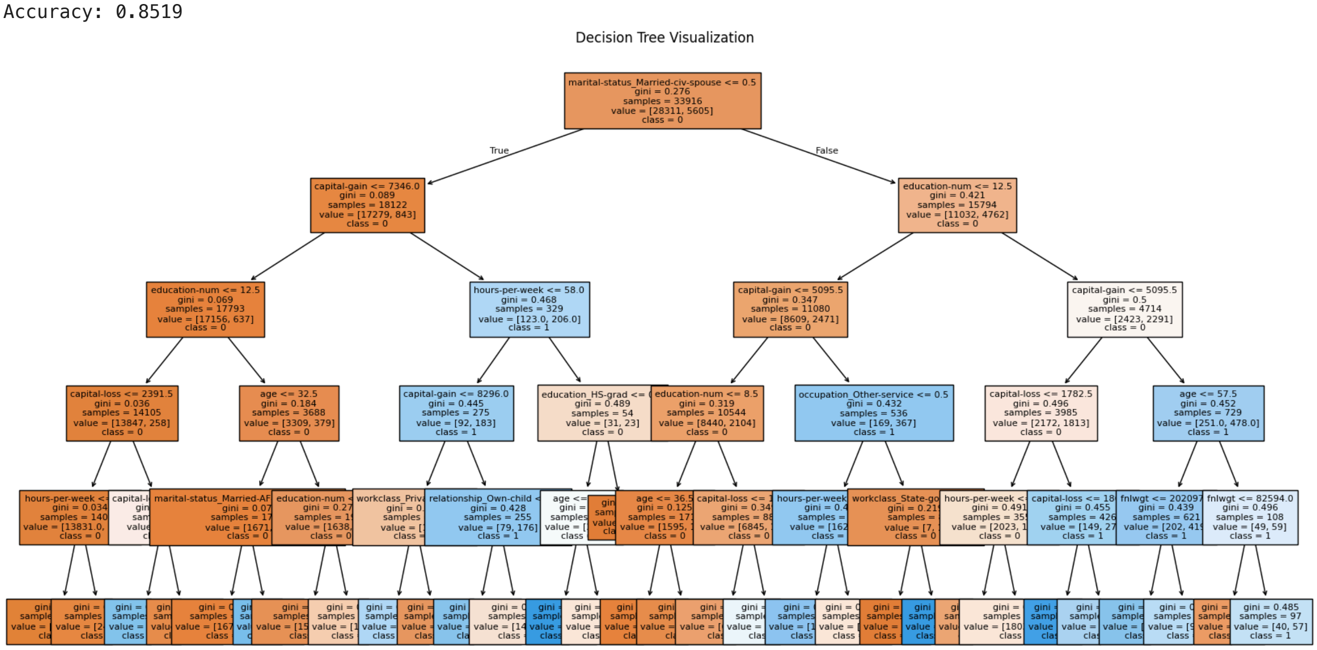

```
Using y_clean_string:
Precision: 0.6541
Recall: 0.2554
Accuracy: 0.8519
```

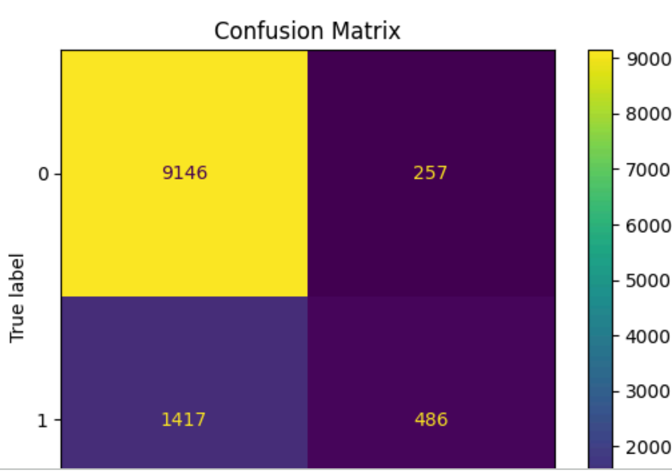







```
Precision: 0.6541
Recall: 0.2554
Accuracy: 0.8519
```





Question 2 (5 points):

Adjust your function to include some plotting features. After your prediction code, plot:

1. A visualization of the resulting Decision Tree
2. A confusion matrix of the results

Your tree might be overwhelming or very large! If it is too large to be interpreted, constrain the max_depth parameter manually to 5 or less.

Graphs of the resulting decision trees and confusion matrices are displayed above.

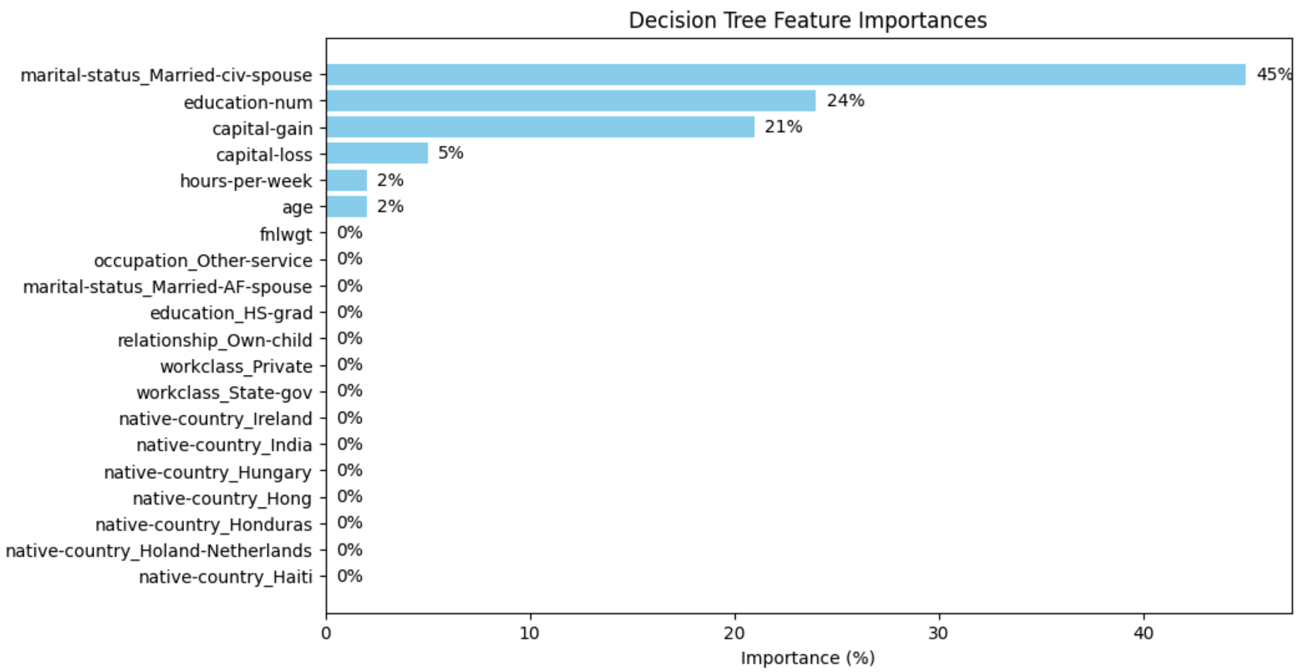
Question 3 (5 points):

Create a sorted list of feature importances and comment on the top features. Plot your feature importances in a horizontal or vertical bar chart from most to least important. Label each bar with its feature importance rounded to the nearest integer (ie: 30%).

Are there a few that seem to be more important than the others?

```
feature_importances = dt.feature_importances_
features = X_clean.columns

importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)


importance_df = importance_df.head(20)
importance_df['Importance (%)'] = (importance_df['Importance'] * 100).round(0)

print("Sorted Feature Importance")
print(importance_df)

plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance (%)'], color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel('Importance (%)')
plt.title('Decision Tree Feature Importances')

for i, v in enumerate(importance_df['Importance (%)']):
    plt.text(v + 0.5, i, f"{int(v)}%", va='center')

plt.show()
```

```
Sorted Feature Importance
                                Feature  Importance  Importance (%)
28      marital-status_Married-civ-spouse    0.450597            45.0
2                         education-num    0.242256            24.0
3                          capital-gain    0.213196            21.0
4                          capital-loss    0.051415             5.0
5                        hours-per-week    0.016153             2.0
0                                   age    0.015335             2.0
1                                fnlwgt    0.002703             0.0
39             occupation_Other-service    0.002089             0.0
27       marital-status_Married-AF-spouse    0.001677             0.0
22                     education_HS-grad    0.001609             0.0
48                relationship_Own-child    0.001425             0.0
7                     workclass_Private    0.000828             0.0
10                  workclass_State-gov    0.000719             0.0
75                native-country_Ireland    0.000000             0.0
73                  native-country_India    0.000000             0.0
72                native-country_Hungary    0.000000             0.0
71                   native-country_Hong    0.000000             0.0
70               native-country_Honduras    0.000000             0.0
69     native-country_Holand-Netherlands    0.000000             0.0
68                  native-country_Haiti    0.000000             0.0
```



Yes, the ones that seem to be more important than the others consist of a few such as the martial-status_Married-civ-spouse, education-num, capital-gain as they all have above 20%. The rest of the data are either less than 5% or just 0%.

Question 4 (5 points):

Write at least 5 sentences interpreting the results of your decision tree, confusion matrix, and feature importance visualizations.

Is there any aspect of your results that you are uncertain or unsure of?

My results show... that the decision tree achieved a pretty good level of accuracy in predicting the individual's income of if it is above or below $50,000. From the confusion matrix, the model does seem to perform well overall but sometimes misclassify some individuals with lower incomes as higher, possibly showing an imbalance or overlap in features between two classes. The top features that were possibly identified was education level, capital gain, and hours worked per week, which makes realistic sense. In general, individuals with higher education, more capital gains, and longer working hours lead to more money being made to higher income.

For an aspect of my result that I am uncertain or unsure of is the fact that the results are from the one-hot encoded categorical variables. This is because doing so may lead to some being more dominant in importance just because they had more representation or unique splits. There is also a vulnerability of decision trees being overfitted when it is not pruned or max_depth is set high. This makes it so that the deeper the trees are, the more unreliable it can become. The values from the precision and recall may hint at something like a Random Forest being better in generalizing.

Question 5 (5 points):

Finally, we will create a new function to tune your decision tree to get more accurate and efficient results. Update your function to take in several new parameters with these default values:

- criterion_val ='gini'
- splitter_val ='best'
- max_depth_val = None
- min_samples_split_val = 2
- min_samples_leaf_val =1

Pass your own variable into the decision tree by specifying what sklearn parameter you are trying to tune. This will simply be the parameter without the "_val" suffix.

For example, if your vanilla decision tree variable is called clf, you would adjust it like this:

`clf = DecisionTreeClassifier(criterion=criterion_val, splitter=splitter_val, ...)`

```
def eval_dtree_tuned(X, y,
    criterion_val='gini',
    splitter_val='best',
    max_depth_val=None,
    min_samples_split_val=2,
    min_samples_leaf_val=1
):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=3001
    )

    clf = DecisionTreeClassifier(
        criterion=criterion_val,
        splitter=splitter_val,
        max_depth=max_depth_val,
        min_samples_split=min_samples_split_val,
        min_samples_leaf=min_samples_leaf_val,
        random_state=3001
    )

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    if y.dtype == 'object':
        pos_label = "Above or Equal"
    else:
        pos_label = 1

    precision = precision_score(y_test, y_pred, pos_label=pos_label, zero_division=0)
    recall = recall_score(y_test, y_pred, pos_label=pos_label, zero_division=0)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"Accuracy: {accuracy:.4f}")

    plt.figure(figsize=(18, 8))
    plot_tree(clf, filled=True, max_depth=3, fontsize=6)
    plt.title("Decision Tree Visualization (max_depth=3 for readability)")
    plt.show()

    ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap='Blues')
    plt.title("Confusion Matrix")
```

```
        plt.show()

        return clf
```

Question 6 (5 points):

Call your new function with either clean y variable at least 3 times. Each time, vary the values for all the parameters and examine its effects on your tree, confusion matrix, and metrics.

You will likely want to look at documentation to see accepted values: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Why did you pick the values you did? What combination had the best effect on accuracy? Were you surprised by any of the results?

```
print("Model 1: Default")
dt1 = eval_dtree_tuned(
    X_clean, y_clean_binary,
    criterion_val='gini',
    splitter_val='best',
    max_depth_val=None,
    min_samples_split_val=2,
    min_samples_leaf_val=1
)

print("Model 2: changed to entropy, moderate pruning")
dt2 = eval_dtree_tuned(
    X_clean, y_clean_binary,
    criterion_val='entropy',
    splitter_val='best',
    max_depth_val=5,
    min_samples_split_val=10,
    min_samples_leaf_val=4
)

print("\nModel 3: random split, more and more and more pruning")
dt3 = eval_dtree_tuned(
    X_clean, y_clean_binary,
    criterion_val='gini',
    splitter_val='random',
    max_depth_val=3,
    min_samples_split_val=20,
    min_samples_leaf_val=8
)
```