Assignment is below at the end

- https://scikit-learn.org/stable/modules/tree.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html

In [1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (20, 6)
plt.rcParams['font.size'] = 14
import pandas as pd
```

In [2]:
```
df = pd.read_csv('../data/adult.data', index_col=False)
```

In [3]:
```
golden = pd.read_csv('../data/adult.test', index_col=False)
```

In [4]:
```
golden.head()
```

Out[4]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female

In [5]:
```
df.head()
```

Out[5]:

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

In [6]:
```
df.columns
```

Out[6]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'salary'],
      dtype='object')
```

In [7]:
```
from sklearn import preprocessing
```

In [8]:
```
enc = preprocessing.OrdinalEncoder()
```

In [9]:
```
transform_columns = ['sex']
non_num_columns = ['workclass', 'education', 'marital-status',
                   'occupation', 'relationship', 'race', 'sex',
                   'native-country']
```

In [10]:
```
pd.get_dummies(df[transform_columns]).head()
```

Out[10]:

	sex_ Female	sex_ Male
0	0	1
1	0	1
2	0	1
3	0	1
4	1	0

In [11]:
```
x = df.copy()

x = pd.concat([x.drop(non_num_columns, axis=1),
               pd.get_dummies(df[transform_columns])], axis=1,)
```

```
x["salary"] = enc.fit_transform(df[["salary"]])
```

In [12]:
```
x.head()
```

Out[12]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	sex_ Female	sex_ Male
0	39	77516	13	2174	0	40	0.0	0	1
1	50	83311	13	0	0	13	0.0	0	1
2	38	215646	9	0	0	40	0.0	0	1
3	53	234721	7	0	0	40	0.0	0	1
4	28	338409	13	0	0	40	0.0	1	0

In [13]:
```
xt = golden.copy()

xt = pd.concat([xt.drop(non_num_columns, axis=1),
                pd.get_dummies(golden[transform_columns])], axis=1,)

xt["salary"] = enc.fit_transform(golden[["salary"]])
```

In [14]:
```
xt.salary.value_counts()
```

Out[14]:
```
0.0    12435
1.0     3846
Name: salary, dtype: int64
```

In [15]:
```
enc.categories_
```

Out[15]:
```
[array([' <=50K.', ' >50K.'], dtype=object)]
```

In [16]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Choose the model of your preference: DecisionTree or RandomForest

In [17]:
```
model = RandomForestClassifier(criterion='entropy')
```

In [18]:
```
model = DecisionTreeClassifier(criterion='entropy', max_depth=None)
```

In [19]:
```
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[19]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [20]:
```
model.tree_.node_count
```

Out[20]:
```
8317
```

In [21]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.feature_importances_))
```

Out[21]:
```
[('age', 0.32367448192304),
 ('education-num', 0.16039957769771152),
 ('capital-gain', 0.22830127920880136),
 ('capital-loss', 0.07800246105037775),
 ('hours-per-week', 0.15405831471575454),
 ('sex_ Female', 0.033582997454121355),
 ('sex_ Male', 0.021980887950193533)]
```

In [22]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.feature_importances_))
```

Out[22]:
```
[('age', 0.32367448192304),
 ('education-num', 0.16039957769771152),
 ('capital-gain', 0.22830127920880136),
 ('capital-loss', 0.07800246105037775),
 ('hours-per-week', 0.15405831471575454),
 ('sex_ Female', 0.033582997454121355),
 ('sex_ Male', 0.021980887950193533)]
```

In [23]:
```
x.drop(['fnlwgt','salary'], axis=1).head()
```

Out[23]:

	age	education-num	capital-gain	capital-loss	hours-per-week	sex_ Female	sex_ Male
0	39	13	2174	0	40	0	1
1	50	13	0	0	13	0	1
2	38	9	0	0	40	0	1
3	53	7	0	0	40	0	1
4	28	13	0	0	40	1	0

In [24]:
```
set(x.columns) - set(xt.columns)
```

Out[24]:
```
set()
```

In [25]:
```
list(x.drop('salary', axis=1).columns)
```

Out[25]:
```
['age',
 'fnlwgt',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'sex_ Female',
 'sex_ Male']
```

In [26]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [27]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [28]:
```
accuracy_score(xt.salary, predictions)
```

Out[28]:
```
0.8205269946563479
```

In [29]:
```
accuracy_score(xt.salary, predictions)
```

Out[29]:
```
0.8205269946563479
```

In [30]:
```
confusion_matrix(xt.salary, predictions)
```

Out[30]:
```
array([[11455,   980],
       [ 1942,  1904]], dtype=int64)
```

In [31]:
```
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.92      0.89     12435
         1.0       0.66      0.50      0.57      3846

    accuracy                           0.82     16281
   macro avg       0.76      0.71      0.73     16281
weighted avg       0.81      0.82      0.81     16281
```

In [32]:
```
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.92      0.89     12435
         1.0       0.66      0.50      0.57      3846

    accuracy                           0.82     16281
   macro avg       0.76      0.71      0.73     16281
weighted avg       0.81      0.82      0.81     16281
```

In [33]:
```
accuracy_score(x.salary, predictionsx)
```

Out[33]:
```
0.8955806025613464
```

In [34]:
```
confusion_matrix(x.salary, predictionsx)
```

Out[34]:
```
array([[24097,   623],
       [ 2777,  5064]], dtype=int64)
```

In [35]:
```
print(classification_report(x.salary, predictionsx))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.97      0.93     24720
         1.0       0.89      0.65      0.75      7841

    accuracy                           0.90     32561
   macro avg       0.89      0.81      0.84     32561
weighted avg       0.90      0.90      0.89     32561
```

In [36]:
```
print(classification_report(x.salary, predictionsx))
```

	precision	recall	f1-score	support
0.0	0.90	0.97	0.93	24720
1.0	0.89	0.65	0.75	7841
accuracy			0.90	32561
macro avg	0.89	0.81	0.84	32561
weighted avg	0.90	0.90	0.89	32561

For the following use the above `adult` dataset. Start with only numerical features/columns.

In [37]:
```
df_train = df.copy()
df_test = golden.copy()
```

In [38]:
```
non_num_cols = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'sex',
    'native-country'
]
df_train = df.drop(non_num_cols, axis = 1).copy()
x_train = df_train.drop('salary', axis = 1)
y_train = df_train[['salary']]

df_test = golden.drop(non_num_cols, axis = 1).copy()
x_test = df_test.drop('salary', axis = 1)
y_test = df_test[['salary']]
```

1. Show the RandomForest outperforms the DecisionTree for a fixed `max_depth` by training using the train set and `precision`, `recall`, `f1` on golden-test set.

In [39]:
```
y_train = enc.fit_transform(y_train).ravel()
y_test = enc.fit_transform(y_test).ravel()
```

In [40]:
```
rf = RandomForestClassifier(max_depth = 15)
dt = DecisionTreeClassifier(max_depth = 15)
```

In [41]:
```
rf.fit(x_train, y_train)
dt.fit(x_train, y_train)
```

Out[41]:
```
DecisionTreeClassifier(max_depth=15)
```

In [42]:
```
rf_pred = rf.predict(x_test)
dt_pred = dt.predict(x_test)
```

In [43]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [44]:
```
def get_accuracies(y_true, preds):
    acc = accuracy_score(y_true, preds)
    confus = confusion_matrix(y_true, preds)
    classif = classification_report(y_true, preds)

    return [acc, confus, classif]
```

In [45]:
```
acc_dict = {"Random Forest": rf_pred, "Decision Tree": dt_pred}
```

In [46]:
```
for k, v in acc_dict.items():
    accs, confus, classif = get_accuracies(y_test, v)
    print(f"\033[1m{k}\033[0m: \n\nAccuracy: {accs:.3f}%\n\nConfusion Matrix:\n {confu
```

```
Random Forest:

Accuracy: 0.838%

Confusion Matrix:
 [[11882   553]
 [ 2081  1765]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.85      0.96      0.90     12435
         1.0       0.76      0.46      0.57      3846

    accuracy                           0.84     16281
   macro avg       0.81      0.71      0.74     16281
weighted avg       0.83      0.84      0.82     16281

Decision Tree:

Accuracy: 0.820%

Confusion Matrix:
 [[11554   881]
 [ 2053  1793]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.85      0.93      0.89     12435
         1.0       0.67      0.47      0.55      3846

    accuracy                           0.82     16281
   macro avg       0.76      0.70      0.72     16281
weighted avg       0.81      0.82      0.81     16281
```

2. For RandomForest or DecisionTree and using the `adult` dataset, systematically add new columns, one by one, that are non-numerical but converted using the feature-extraction techniques we learned. Show [`precision`, `recall`, `f1`] for each additional feature added.

In [47]:
```
import numpy as np

df_train = df.copy()
df_test = golden.copy()
```

In [48]:
```
def run_model(df_train,df_test,non_num_cols, target_var, col_list = ['sex'], model_typ
    """

    Function that runs classification models and outputs all accuracy metrics

    Variables
    --------------
    df_train: dataframe
        dataframe used to train the model
    df_test: dataframe
        dataframe used to test the model's accuracy
    non_num_cols: list
        list of columns that are non-numeric
    target_var: str
        name of predictor column
    col_list: list
        list of columns that are intended to be encoded and concatenated to the traini
    model_type: str
        name of model used as well as report
            opts: 'Random Forest', 'Decision Tree'
    verbose: bool
        boolean to either print or not print the output of the accuracy report
    """

    dftrain = df_train.drop(non_num_cols, axis = 1).copy()
    dftest = df_test.drop(non_num_cols, axis = 1).copy()

    xtrain = pd.concat([dftrain, pd.get_dummies(df_train[col_list])], axis = 1)
    xtest = pd.concat([dftest, pd.get_dummies(df_test[col_list])], axis = 1)

    diff_cols = set(xtrain.columns) - set(xtest.columns)

    # Create zero column for columns that differ between the test and training sets
    if diff_cols:
        difference_df = pd.DataFrame(data=np.zeros((xtest.shape[0], len(diff_cols))),
                                     columns=list(diff_cols))
        xtest = pd.concat([xtest, difference_df], axis = 1)
        xtest = xtest[xtrain.columns]


    ytrain = enc.fit_transform(df_train[[target_var]]).ravel()
    ytest = enc.fit_transform(df_test[[target_var]]).ravel()

    if model_type == 'Random Forest':
        model = RandomForestClassifier()
    elif model_type == 'Decision Tree':
        model = DecisionTreeClassifier()

    model.fit(xtrain, ytrain)
    preds = model.predict(xtest)

    acc_list = get_accuracies(ytest, preds)

    if verbose:
        print(f"\033[1m{model_type}\033[0m: \n\nNon-Numerical Columns: {col_list} \n\r

    return acc_list
```

In [49]:
```
def run_model_addcat(df_train, df_test, target_var = 'salary'):
```

```
    """
    Function that systematically adds encoded categorical variables, trains a model, a

    Variables
    ----------
    df_train: dataframe
        dataframe used to train the model
    df_test: dataframe
        dataframe used to test the model
    target_var: str
        name of predictor column

    """


    cat_vars = list(df_train.select_dtypes(include=['object']).columns)
    cat_vars.remove(target_var)
    cat_vars_uniquedict = {i:len(df[i].unique()) for i in cat_vars}

    non_num_cols = [i[0] for i in sorted(cat_vars_uniquedict.items(), key = lambda x:

    cat_vars = list(df_train.select_dtypes(include=['object']).columns)
    non_num_cols = cat_vars.copy()
    cat_vars.remove(target_var)
    cat_vars_uniquedict = {i:len(df[i].unique()) for i in cat_vars}
    cat_vars = [i[0] for i in sorted(cat_vars_uniquedict.items(), key = lambda x: x[1]
    print("---Starting Report---\n\n\n")
    for i in range(1,len(cat_vars) + 1):
        run_model(df_train, df_test, non_num_cols, 'salary', cat_vars[0:i], 'Random Fo
        run_model(df_train, df_test, non_num_cols, 'salary', cat_vars[0:i], 'Decision
    print("\n\n\n---Finishing Report---")
```

In [50]:
```
run_model_addcat(df, golden)
```

```
---Starting Report---
```

Random Forest:

```
Non-Numerical Columns: ['sex']

Accuracy: 81.98%

Confusion Matrix:
 [[11287  1148]
 [ 1786  2060]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.86      0.91      0.88     12435
         1.0       0.64      0.54      0.58      3846

    accuracy                           0.82     16281
   macro avg       0.75      0.72      0.73     16281
weighted avg       0.81      0.82      0.81     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex']

Accuracy: 78.48%

Confusion Matrix:
 [[10652  1783]
 [ 1720  2126]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.86      0.86      0.86     12435
         1.0       0.54      0.55      0.55      3846

    accuracy                           0.78     16281
   macro avg       0.70      0.70      0.70     16281
weighted avg       0.79      0.78      0.79     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race']

Accuracy: 82.03%

Confusion Matrix:
 [[11313  1122]
 [ 1803  2043]]
```

```
Classification Report:

              precision    recall  f1-score   support

         0.0       0.86      0.91      0.89     12435
         1.0       0.65      0.53      0.58      3846

    accuracy                           0.82     16281
   macro avg       0.75      0.72      0.73     16281
weighted avg       0.81      0.82      0.81     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race']

Accuracy: 78.20%

Confusion Matrix:
 [[10610  1825]
 [ 1724  2122]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.86      0.85      0.86     12435
         1.0       0.54      0.55      0.54      3846

    accuracy                           0.78     16281
   macro avg       0.70      0.70      0.70     16281
weighted avg       0.78      0.78      0.78     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race', 'relationship']

Accuracy: 84.61%

Confusion Matrix:
 [[11431  1004]
 [ 1502  2344]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.92      0.90     12435
         1.0       0.70      0.61      0.65      3846

    accuracy                           0.85     16281
   macro avg       0.79      0.76      0.78     16281
weighted avg       0.84      0.85      0.84     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race', 'relationship']

Accuracy: 81.09%

Confusion Matrix:
 [[10883  1552]
 [ 1526  2320]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88     12435
         1.0       0.60      0.60      0.60      3846

    accuracy                           0.81     16281
   macro avg       0.74      0.74      0.74     16281
weighted avg       0.81      0.81      0.81     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status']

Accuracy: 84.35%

Confusion Matrix:
 [[11399  1036]
 [ 1512  2334]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.92      0.90     12435
         1.0       0.69      0.61      0.65      3846

    accuracy                           0.84     16281
   macro avg       0.79      0.76      0.77     16281
weighted avg       0.84      0.84      0.84     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status']

Accuracy: 81.31%

Confusion Matrix:
 [[10896  1539]
 [ 1504  2342]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88     12435
```

```
         1.0       0.60      0.61      0.61      3846

    accuracy                           0.81     16281
   macro avg       0.74      0.74      0.74     16281
weighted avg       0.81      0.81      0.81     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass']

Accuracy: 84.42%

Confusion Matrix:
 [[11456   979]
 [ 1557  2289]]

Classification Report:

               precision    recall  f1-score   support

         0.0       0.88      0.92      0.90     12435
         1.0       0.70      0.60      0.64      3846

    accuracy                           0.84     16281
   macro avg       0.79      0.76      0.77     16281
weighted avg       0.84      0.84      0.84     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass']

Accuracy: 81.03%

Confusion Matrix:
 [[10879  1556]
 [ 1533  2313]]

Classification Report:

               precision    recall  f1-score   support

         0.0       0.88      0.87      0.88     12435
         1.0       0.60      0.60      0.60      3846

    accuracy                           0.81     16281
   macro avg       0.74      0.74      0.74     16281
weighted avg       0.81      0.81      0.81     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass',
'occupation']

Accuracy: 85.39%
```

```
Confusion Matrix:
 [[11544   891]
 [ 1487  2359]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.89      0.93      0.91     12435
         1.0       0.73      0.61      0.66      3846

    accuracy                           0.85     16281
   macro avg       0.81      0.77      0.79     16281
weighted avg       0.85      0.85      0.85     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass',
'occupation']

Accuracy: 81.13%

Confusion Matrix:
 [[10822  1613]
 [ 1460  2386]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.87      0.88     12435
         1.0       0.60      0.62      0.61      3846

    accuracy                           0.81     16281
   macro avg       0.74      0.75      0.74     16281
weighted avg       0.81      0.81      0.81     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass',
'occupation', 'education']

Accuracy: 85.06%

Confusion Matrix:
 [[11506   929]
 [ 1503  2343]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.93      0.90     12435
         1.0       0.72      0.61      0.66      3846
```

```
    accuracy                           0.85     16281
   macro avg       0.80      0.77      0.78     16281
weighted avg       0.84      0.85      0.85     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass',
'occupation', 'education']

Accuracy: 80.93%

Confusion Matrix:
 [[10779  1656]
 [ 1448  2398]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.87      0.87     12435
         1.0       0.59      0.62      0.61      3846

    accuracy                           0.81     16281
   macro avg       0.74      0.75      0.74     16281
weighted avg       0.81      0.81      0.81     16281
```

Random Forest:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass',
'occupation', 'education', 'native-country']

Accuracy: 85.05%

Confusion Matrix:
 [[11513   922]
 [ 1512  2334]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.93      0.90     12435
         1.0       0.72      0.61      0.66      3846

    accuracy                           0.85     16281
   macro avg       0.80      0.77      0.78     16281
weighted avg       0.84      0.85      0.85     16281
```

Decision Tree:

```
Non-Numerical Columns: ['sex', 'race', 'relationship', 'marital-status', 'workclass',
'occupation', 'education', 'native-country']

Accuracy: 81.08%
```

```
Confusion Matrix:
 [[10817  1618]
 [ 1462  2384]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.88      0.87      0.88     12435
         1.0       0.60      0.62      0.61      3846

    accuracy                           0.81     16281
   macro avg       0.74      0.74      0.74     16281
weighted avg       0.81      0.81      0.81     16281
```

```
---Finishing Report---
```

3. Optional: Using gridSearch find the most optimal parameters for your model

Warning: this can be computationally intensive and may take some time.

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://scikit-learn.org/stable/modules/grid_search.html

In [395…

```
from sklearn.model_selection import GridSearchCV
parameters_rf = {
    'n_estimators' : [10, 50, 100, 200, 300, 500],
    'criterion' : ['gini', 'entropy'],
    'max_depth': [None, 10, 15, 25, 50]
}
rf = RandomForestClassifier()
clf = GridSearchCV(rf, parameters_rf)
clf.fit(x_train, y_train)
```

Out[395]:

```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 10, 15, 25, 50],
                         'n_estimators': [10, 50, 100, 200, 300, 500]})
```

In [397…

```
clf.cv_results_
```

Out[397]:
```
{'mean_fit_time': array([ 0.26607652,  1.49843245,  3.07443242,  5.42180214,  8.64762
654,
        14.51657195,  0.14425912,  0.71538653,  1.42572818,  2.84674644,
         4.27355132,  7.39134669,  0.19362202,  1.06073928,  2.05134807,
         3.65817304,  5.48727622,  9.12941923,  0.24207854,  1.15445013,
         2.29742756,  5.3503171 ,  8.94251018, 15.09877682,  0.37479377,
         1.72970343,  3.06913753,  6.01469517,  9.01518703, 15.24403338,
         0.38510656,  2.0672318 ,  4.09541273,  8.10080194, 13.41103654,
        23.85574636,  0.21171923,  0.96369548,  2.60740027,  5.95415359,
         8.08663545, 11.94553466,  0.33672209,  1.4965764 ,  3.04027114,
         6.44706473,  8.87330565, 15.21995511,  0.34397197,  1.73710818,
         3.44171743,  7.7625483 , 11.62421522, 20.41408758,  0.40847578,
         2.03247933,  4.49909558,  7.94361548, 13.54872761, 22.33676162]),
 'std_fit_time': array([6.75443030e-02, 1.23954929e-01, 2.61542535e-01, 4.46269218e-0
1,
        3.43647564e-01, 8.92577828e-01, 5.53860254e-03, 1.96575079e-02,
        2.93837213e-02, 1.40392686e-02, 4.48388137e-02, 4.05256354e-01,
        1.58320708e-02, 1.16645855e-01, 1.62370242e-01, 2.93967142e-02,
        3.34056994e-02, 2.63363177e-02, 1.36739914e-02, 1.85746992e-02,
        1.43904137e-02, 2.52713649e-01, 6.65413520e-01, 7.05152146e-01,
        5.49848038e-02, 3.08303539e-01, 1.15612837e-01, 2.49943091e-01,
        8.01281693e-02, 1.69074781e-01, 1.84416658e-02, 1.36483168e-01,
        2.63284169e-01, 2.73536152e-01, 2.14270172e+00, 2.63310852e+00,
        1.26576820e-02, 3.61383343e-02, 4.26649953e-01, 4.96613446e-01,
        1.59563853e+00, 1.59897479e+00, 2.08440263e-02, 9.71952274e-02,
        2.10912221e-01, 5.81342456e-01, 4.09483788e-01, 1.16687762e+00,
        2.47785674e-03, 5.77697638e-02, 7.74454004e-02, 6.94507058e-01,
        3.35104615e-01, 1.48757919e+00, 1.34741632e-02, 1.84216461e-01,
        4.14596791e-01, 6.41281096e-01, 2.51809170e+00, 3.01951312e+00]),
 'mean_score_time': array([0.01962352, 0.09836264, 0.20806046, 0.37410049, 0.5319016
,
        1.02558203, 0.0103703 , 0.04104562, 0.08036871, 0.15942698,
        0.23838816, 0.39551568, 0.01358857, 0.06196446, 0.11278629,
        0.20237041, 0.30895376, 0.53273554, 0.01722789, 0.07253556,
        0.14885025, 0.31497359, 0.54915042, 0.85707278, 0.03570647,
        0.10074358, 0.20768161, 0.40445223, 0.58216748, 0.94904275,
        0.02137151, 0.0949461 , 0.20518732, 0.37055998, 0.76928658,
        0.95595131, 0.01220379, 0.04849496, 0.11816721, 0.25336456,
        0.32691298, 0.50702114, 0.02497644, 0.06899943, 0.13270693,
        0.29823918, 0.37813659, 0.66077771, 0.01909366, 0.08517518,
        0.15417237, 0.32889748, 0.5491363 , 1.02477174, 0.02231531,
        0.11787529, 0.22240648, 0.38394308, 0.72043104, 0.99918385]),
 'std_score_time': array([0.00433416, 0.00978219, 0.0652977 , 0.07075258, 0.07213458,
        0.27274129, 0.00082672, 0.00175432, 0.00161673, 0.00300204,
        0.01314208, 0.0108666 , 0.00136289, 0.00635717, 0.01297358,
        0.00209548, 0.00362666, 0.01978794, 0.00201604, 0.00306782,
        0.01651311, 0.01759621, 0.01759177, 0.05444235, 0.02116327,
        0.01209304, 0.03194997, 0.05616422, 0.05988733, 0.04714396,
        0.00209167, 0.00604295, 0.03974779, 0.00973101, 0.40152135,
        0.12806161, 0.00126395, 0.00419959, 0.02465119, 0.03734734,
        0.08113808, 0.06457283, 0.01427815, 0.00385778, 0.00159236,
        0.07290611, 0.00964049, 0.05785028, 0.00205581, 0.00815546,
        0.00385392, 0.00843533, 0.0310248 , 0.12101545, 0.00192016,
        0.02798329, 0.05177499, 0.03710497, 0.16381312, 0.20491679]),
 'param_criterion': masked_array(data=['gini', 'gini', 'gini', 'gini', 'gini', 'gin
i', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'entropy', 'entropy', 'entropy',
```

```
                       'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                       'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                       'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                       'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                       'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                       'entropy', 'entropy'],
                 mask=[False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False],
            fill_value='?',
                 dtype=object),
  'param_max_depth': masked_array(data=[None, None, None, None, None, None, 10, 10, 1
0, 10, 10,
                       10, 15, 15, 15, 15, 15, 15, 25, 25, 25, 25, 25, 25, 50,
                       50, 50, 50, 50, 50, None, None, None, None, None, None,
                       10, 10, 10, 10, 10, 10, 15, 15, 15, 15, 15, 15, 25, 25,
                       25, 25, 25, 25, 50, 50, 50, 50, 50, 50],
                 mask=[False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False],
            fill_value='?',
                 dtype=object),
  'param_n_estimators': masked_array(data=[10, 50, 100, 200, 300, 500, 10, 50, 100, 20
0, 300, 500,
                       10, 50, 100, 200, 300, 500, 10, 50, 100, 200, 300, 500,
                       10, 50, 100, 200, 300, 500, 10, 50, 100, 200, 300, 500,
                       10, 50, 100, 200, 300, 500, 10, 50, 100, 200, 300, 500,
                       10, 50, 100, 200, 300, 500, 10, 50, 100, 200, 300, 500],
                 mask=[False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False],
            fill_value='?',
                 dtype=object),
  'params': [{'criterion': 'gini', 'max_depth': None, 'n_estimators': 10},
   {'criterion': 'gini', 'max_depth': None, 'n_estimators': 50},
   {'criterion': 'gini', 'max_depth': None, 'n_estimators': 100},
   {'criterion': 'gini', 'max_depth': None, 'n_estimators': 200},
   {'criterion': 'gini', 'max_depth': None, 'n_estimators': 300},
   {'criterion': 'gini', 'max_depth': None, 'n_estimators': 500},
   {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 10},
   {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 50},
   {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 100},
   {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 200},
   {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 300},
   {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 500},
```

```
  {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 10},
  {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 50},
  {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 500},
  {'criterion': 'gini', 'max_depth': 25, 'n_estimators': 10},
  {'criterion': 'gini', 'max_depth': 25, 'n_estimators': 50},
  {'criterion': 'gini', 'max_depth': 25, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 25, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 25, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 25, 'n_estimators': 500},
  {'criterion': 'gini', 'max_depth': 50, 'n_estimators': 10},
  {'criterion': 'gini', 'max_depth': 50, 'n_estimators': 50},
  {'criterion': 'gini', 'max_depth': 50, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 50, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 50, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 50, 'n_estimators': 500},
  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 10},
  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 50},
  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 500},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 10},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 50},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 500},
  {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 10},
  {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 50},
  {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 500},
  {'criterion': 'entropy', 'max_depth': 25, 'n_estimators': 10},
  {'criterion': 'entropy', 'max_depth': 25, 'n_estimators': 50},
  {'criterion': 'entropy', 'max_depth': 25, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 25, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 25, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 25, 'n_estimators': 500},
  {'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 10},
  {'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 50},
  {'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 500}],
 'split0_test_score': array([0.80853677, 0.80377706, 0.80869031, 0.80761554, 0.806080
15,
        0.80546599, 0.83187471, 0.83387072, 0.83356364, 0.83325656,
        0.83387072, 0.83264241, 0.83356364, 0.83279595, 0.83233533,
        0.83279595, 0.83264241, 0.83233533, 0.80915093, 0.81621373,
        0.81867035, 0.81882389, 0.82128052, 0.81897743, 0.80408414,
        0.80454476, 0.80546599, 0.80608015, 0.80730846, 0.80761554,
        0.81329648, 0.80976509, 0.80654076, 0.80776908, 0.80485183,
        0.80577307, 0.83387072, 0.83356364, 0.8334101 , 0.83356364,
        0.83387072, 0.83448488, 0.83448488, 0.83126056, 0.83279595,
        0.83371718, 0.83294949, 0.83310302, 0.81682788, 0.82542607,
        0.82250883, 0.82389068, 0.82450484, 0.82419776, 0.80761554,
```

```
       0.80838323, 0.80730846, 0.80577307, 0.80608015, 0.8046983 ]),
 'split1_test_score': array([0.8034398 , 0.80036855, 0.7997543 , 0.80052211, 0.798832
92,
       0.7972973 , 0.83445946, 0.83522727, 0.83568796, 0.83399877,
       0.83323096, 0.83369165, 0.83077396, 0.83445946, 0.83507371,
       0.83384521, 0.83568796, 0.83507371, 0.80712531, 0.81296069,
       0.81434275, 0.81649263, 0.81464988, 0.81572482, 0.80205774,
       0.80052211, 0.80113636, 0.79914005, 0.79990786, 0.80082924,
       0.80082924, 0.80021499, 0.79914005, 0.7972973 , 0.79545455,
       0.7985258 , 0.83246314, 0.83476658, 0.83399877, 0.83338452,
       0.83338452, 0.83399877, 0.83261671, 0.83599509, 0.8355344 ,
       0.83584152, 0.83476658, 0.83369165, 0.81326781, 0.8161855 ,
       0.81879607, 0.81710688, 0.82186732, 0.81879607, 0.80236486,
       0.8009828 , 0.7997543 , 0.79668305, 0.79668305, 0.79760442]),
 'split2_test_score': array([0.80773956, 0.81388206, 0.81188575, 0.81280713, 0.813421
38,
       0.81219287, 0.83599509, 0.83707002, 0.83722359, 0.83768428,
       0.83691646, 0.83722359, 0.83722359, 0.83845209, 0.83860565,
       0.83875921, 0.83937346, 0.83875921, 0.81633907, 0.82493857,
       0.82632064, 0.82555283, 0.82708845, 0.82693489, 0.81035012,
       0.8125    , 0.81142506, 0.8125    , 0.81449631, 0.81265356,
       0.81188575, 0.81280713, 0.81480344, 0.81096437, 0.80850737,
       0.81157862, 0.83246314, 0.83476658, 0.83522727, 0.83645577,
       0.83630221, 0.83691646, 0.83445946, 0.83707002, 0.83722359,
       0.83722359, 0.83737715, 0.83783784, 0.82202088, 0.82754914,
       0.8264742 , 0.83000614, 0.82908477, 0.82785627, 0.80850737,
       0.81219287, 0.81203931, 0.81050369, 0.81280713, 0.81219287]),
 'split3_test_score': array([0.810043  , 0.80850737, 0.80420762, 0.80589681, 0.805896
81,
       0.80420762, 0.83691646, 0.83845209, 0.84060197, 0.84090909,
       0.84014128, 0.84090909, 0.83492015, 0.8379914 , 0.83737715,
       0.83722359, 0.83753071, 0.8379914 , 0.81557125, 0.82186732,
       0.82094595, 0.82125307, 0.82248157, 0.82217445, 0.81050369,
       0.80390049, 0.80574324, 0.80482187, 0.80620393, 0.80497543,
       0.807586  , 0.80681818, 0.80482187, 0.80697174, 0.80420762,
       0.80497543, 0.83630221, 0.8404484 , 0.8404484 , 0.83983415,
       0.83937346, 0.83968059, 0.83323096, 0.83768428, 0.83783784,
       0.83737715, 0.83753071, 0.83814496, 0.81864251, 0.82355651,
       0.82693489, 0.8264742 , 0.82816339, 0.82632064, 0.80451474,
       0.80558968, 0.80789312, 0.80605037, 0.80451474, 0.80528256]),
 'split4_test_score': array([0.80727887, 0.80743243, 0.80927518, 0.807586  , 0.808353
81,
       0.80958231, 0.83369165, 0.83599509, 0.83599509, 0.83522727,
       0.83691646, 0.83522727, 0.83323096, 0.83660934, 0.83737715,
       0.83630221, 0.83691646, 0.83707002, 0.814957  , 0.82263514,
       0.82048526, 0.82386364, 0.82263514, 0.82232801, 0.81065725,
       0.80743243, 0.80927518, 0.80988943, 0.80835381, 0.80896806,
       0.80651106, 0.80635749, 0.80635749, 0.80912162, 0.80866093,
       0.80850737, 0.83522727, 0.8343059 , 0.83538084, 0.83492015,
       0.83507371, 0.83492015, 0.83154177, 0.83599509, 0.8355344 ,
       0.83660934, 0.83691646, 0.83722359, 0.82263514, 0.82324939,
       0.82570639, 0.82632064, 0.82739558, 0.82601351, 0.81142506,
       0.80743243, 0.80528256, 0.80835381, 0.80712531, 0.80773956]),
 'mean_test_score': array([0.8074076 , 0.8067935 , 0.80676263, 0.80688552, 0.8065170
1,
       0.80574922, 0.83458747, 0.83612304, 0.83661445, 0.83621519,
       0.83621518, 0.8359388 , 0.83394246, 0.83606165, 0.8361538 ,
       0.83578523, 0.8364302 , 0.83624594, 0.81262871, 0.81972309,
       0.82015299, 0.82119721, 0.82162711, 0.82122792, 0.80753059,
       0.80577996, 0.80660917, 0.8064863 , 0.80725408, 0.80700837,
```

```
        0.8080217 , 0.80719257, 0.80633272, 0.80642482, 0.80433646,
        0.80587206, 0.8340653 , 0.83557022, 0.83569308, 0.83563165,
        0.83560093, 0.83600017, 0.83326675, 0.83560101, 0.83578523,
        0.83615376, 0.83590808, 0.83600021, 0.81867884, 0.82319332,
        0.82408408, 0.82475971, 0.82620318, 0.82463685, 0.80688552,
        0.8069162 , 0.80645555, 0.8054728 , 0.80544207, 0.80550354]),
 'std_test_score': array([0.00219502, 0.00456009, 0.00428766, 0.00393852, 0.00470457,
        0.00510108, 0.00176542, 0.00156288, 0.00231606, 0.00278686,
        0.00248193, 0.00292495, 0.00211711, 0.00214336, 0.00222495,
        0.00218542, 0.00223646, 0.00231125, 0.00374776, 0.00443452,
        0.00386616, 0.00328177, 0.00400935, 0.00374231, 0.00369852,
        0.00401479, 0.00353012, 0.0045749 , 0.0046633 , 0.00395959,
        0.00440358, 0.00418725, 0.00501626, 0.00476021, 0.00480142,
        0.0043426 , 0.00151826, 0.00247843, 0.00249039, 0.00237372,
        0.00214046, 0.00209061, 0.00112301, 0.00226487, 0.00175178,
        0.0013329 , 0.00178125, 0.00215391, 0.00345045, 0.0038263 ,
        0.00306321, 0.0042947 , 0.00265525, 0.00314333, 0.00315682,
        0.00366679, 0.00400637, 0.00471791, 0.0051974 , 0.00475058]),
 'rank_test_score': array([39, 46, 47, 44, 49, 56, 21,  8,  1,  4,  5, 12, 23,  9,
6, 14,  2,
         3, 36, 34, 33, 32, 30, 31, 38, 55, 48, 50, 40, 42, 37, 41, 53, 52,
        60, 54, 22, 20, 16, 17, 19, 11, 24, 18, 14,  7, 13, 10, 35, 29, 28,
        26, 25, 27, 44, 43, 51, 58, 59, 57])}
```

In [399…
```
clf.best_params_
```

Out[399]:
```
{'criterion': 'gini', 'max_depth': 10, 'n_estimators': 100}
```

In [51]:
```
cat_vars = list(df.select_dtypes(include = ['object']).columns)
non_num_cols = cat_vars.copy()
cat_vars.remove('salary')
```

In [52]:
```
rf = RandomForestClassifier(criterion = 'gini', max_depth = 10, n_estimators = 100)

xtrain = pd.concat([df_train.drop(non_num_cols, axis = 1), pd.get_dummies(df[cat_vars]
# xtrain = xtrain.drop('native-country_ Holand-Netherlands', axis = 1)
xtest = pd.concat([df_test.drop(non_num_cols, axis = 1), pd.get_dummies(golden[cat_var
diff_cols = set(xtrain.columns) - set(xtest.columns)
diff_df = pd.DataFrame(np.zeros((xtest.shape[0], 1)), columns = diff_cols)
xtest = pd.concat([xtest, diff_df], axis = 1)
xtest = xtest[xtrain.columns]

ytrain = df[['salary']]
ytrain = enc.fit_transform(ytrain).ravel()
ytest = golden[['salary']]
ytest = enc.fit_transform(ytest).ravel()

rf.fit(xtrain, ytrain)
pred = rf.predict(xtest)

accs = get_accuracies(ytest, pred)
```

In [54]:
```
print(f"\033[1m{'Random Forest'}\033[0m: \n\nAccuracy: {accs[0] * 100:.2f}%\n\nConfusi
```

Random Forest:

```
Accuracy: 85.81%

Confusion Matrix:
 [[11888   547]
 [ 1763  2083]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.87      0.96      0.91     12435
         1.0       0.79      0.54      0.64      3846

    accuracy                           0.86     16281
   macro avg       0.83      0.75      0.78     16281
weighted avg       0.85      0.86      0.85     16281
```

In [ ]: