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 [ ]:
```
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 [ ]:
```
df = pd.read_csv('../mlnn-main/data/adult.data', index_col=False)
```

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

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

Out[ ]:

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

Out[ ]:

	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 [ ]:
```
df.columns, golden.columns
```

Out[ ]:
```
(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'),
 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 [ ]:
```
from sklearn import preprocessing
```

In [ ]:
```
# Columns we want to transform
transform_columns = ['sex']

#Columns we can't use because non-numerical
non_num_columns = ['workclass', 'education', 'marital-status',
                   'occupation', 'relationship', 'race', 'sex',
                   'native-country']
```

First let's try using `pandas.get_dummies()` to transform columns

In [ ]:
```
dummies = pd.get_dummies(df[transform_columns])
dummies
```

Out[ ]:

	sex_ Female	sex_ Male
0	False	True
1	False	True
2	False	True
3	False	True
4	True	False
...	...	...
32556	True	False
32557	False	True
32558	True	False
32559	False	True
32560	True	False

32561 rows × 2 columns

In [ ]:
```
dummies.shape
```

Out[ ]:
```
(32561, 2)
```

sklearn has a similar process for OneHot Encoding features

In [ ]:
```
onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist", sparse=Fals
onehot.fit(df[transform_columns])
```

```
c:\Users\jomors\AppData\Local\anaconda3\Lib\site-packages\sklearn\preprocessing\_enco
ders.py:972: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 an
d will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
```

Out[ ]:
```
OneHotEncoder(handle_unknown='infrequent_if_exist', sparse=False,
              sparse_output=False)
```

In [ ]:
```
onehot.categories_
```

Out[ ]:
```
[array([' Female', ' Male'], dtype=object)]
```

In [ ]:
```
sex = onehot.transform(df[transform_columns])
sex
```

Out[ ]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       ...,
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

In [ ]:
```
sex.shape
```

Out[ ]:
```
(32561, 2)
```

The variable *sex* now has the sex variable from the *training* dataset one hot encoded.

In addition to OneHot encoding there is Ordinal Encoding

In [ ]:
```
enc = preprocessing.OrdinalEncoder()
enc.fit(df[["salary"]])
salary = enc.transform(df[["salary"]])
salary
```

Out[ ]:
```
array([[0.],
       [0.],
       [0.],
       ...,
       [0.],
       [0.],
       [1.]])
```

In [ ]:
```
enc.categories_[0]
```

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

This next code block uses OneHotEncoder and OrdinalEncoder to produce training data and store it in x. Notice that in this process we are dropping ALL of the non-numeric values and reattaching the columns that were transformed.

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

# transformed = pd.get_dummies(df[transform_columns])


onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist", sparse=Fals

enc = preprocessing.OrdinalEncoder()

enc.fit(df[["salary"]])


transformed = onehot.transform(df[transform_columns])   # Get the OHE values from the
new_cols = list(onehot.categories_[0].flatten())        # Get the column names of thos
df_trans = pd.DataFrame(transformed, columns=new_cols)  # Create a dataframe with just

# Concatenate the OHE columns form the datframe to the x training data,
# but drop all non-numeric values along the way.
```

```
x = pd.concat(
    [
        x.drop(non_num_columns, axis=1),    # drop non-numeric values
        df_trans                            # Add on the OHE columns
    ],
    axis=1,)


# At this point, salary is still left as a categorical variable,
# so we need to change it via the Ordinal Encoder
x["salary"] = enc.transform(df[["salary"]])
```

```
c:\Users\jomors\AppData\Local\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

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

Out[ ]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	Female	Male
0	39	77516	13	2174	0	40	0.0	0.0	1.0
1	50	83311	13	0	0	13	0.0	0.0	1.0
2	38	215646	9	0	0	40	0.0	0.0	1.0
3	53	234721	7	0	0	40	0.0	0.0	1.0
4	28	338409	13	0	0	40	0.0	1.0	0.0

Now we have updates the training data, we need to do the same thing to the test data based on the *golden* dataset

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

transformed = onehot.transform(xt[transform_columns])
new_cols = list(onehot.categories_[0].flatten())
df_trans = pd.DataFrame(transformed, columns=new_cols)

# !!!! Note the original file assigned this to x not xt, which was an error.
xt = pd.concat(
    [
        xt.drop(non_num_columns, axis=1),
        df_trans
    ],
    axis=1,)

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

In [ ]:
```
xt.head()
```

Out[ ]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	Female	Male
0	25	226802	7	0	0	40	0.0	0.0	1.0
1	38	89814	9	0	0	50	0.0	0.0	1.0
2	28	336951	12	0	0	40	1.0	0.0	1.0
3	44	160323	10	7688	0	40	1.0	0.0	1.0
4	18	103497	10	0	0	30	0.0	1.0	0.0

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

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

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

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

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

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

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

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

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

Out[ ]:
```
8327
```

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

Out[ ]:
```
[('age', 0.3254844062448),
 ('education-num', 0.16067436634608706),
 ('capital-gain', 0.22725795380400066),
 ('capital-loss', 0.07753432242607652),
 ('hours-per-week', 0.15357252211923253),
 (' Female', 0.03410189123506942),
 (' Male', 0.02137453782473386)]
```

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

Out[ ]:
```
[('age', 0.3254844062448),
 ('education-num', 0.16067436634608706),
 ('capital-gain', 0.22725795380400066),
 ('capital-loss', 0.07753432242607652),
 ('hours-per-week', 0.15357252211923253),
 (' Female', 0.03410189123506942),
 (' Male', 0.02137453782473386)]
```

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

Out[ ]:

	age	education-num	capital-gain	capital-loss	hours-per-week	Female	Male
0	39	13	2174	0	40	0.0	1.0
1	50	13	0	0	13	0.0	1.0
2	38	9	0	0	40	0.0	1.0
3	53	7	0	0	40	0.0	1.0
4	28	13	0	0	40	1.0	0.0

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

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

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

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

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

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

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

Out[ ]:
```
0.8210797862539156
```

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

Out[ ]:
```
0.8210797862539156
```

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

Out[ ]:
```
array([[11467,   968],
       [ 1945,  1901]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

         0.0       0.85      0.92      0.89     12435
         1.0       0.66      0.49      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 [ ]:
```
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.92      0.89     12435
         1.0       0.66      0.49      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 [ ]:
```
accuracy_score(x.salary, predictionsx)
```

Out[ ]:
```
0.8955806025613464
```

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

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

In [ ]:
```
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 [ ]:
```
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.

1. Show the RandomForest outperforms the DecisionTree for a fixed `max_depth` by training using the train set and calculate `precision`, `recall`, `f1`, `confusion matrix` on golden-test set. Start with only numerical features/columns. (age, education-num, capital-gain, capital-loss, hours-per-week)

In [ ]:
```
# Training data comes from the original training dataframe (adult.data)
xtrain = x.drop(['fnlwgt','salary'], axis=1)
ytrain = x.salary

# Test data comes form the original golden data set (adult.test)
xtest = xt.drop(['fnlwgt','salary'], axis=1)
ytest = xt.salary
```

In [ ]:
```
# Create a Decision Tree model with a fixed max_depth,
# and fit the training data.
model_tree = DecisionTreeClassifier(criterion='entropy', max_depth=10)
model_tree.fit(xtrain, ytrain)

# How many nodes did we get?
model_tree.tree_.node_count
```

Out[ ]:
```
459
```

In [ ]:
```
# Predict results from the Decision Tree using the test data
pred_tree = model_tree.predict(xtest)
accuracy_score(ytest, pred_tree)
```

Out[ ]:
```
0.8398747005712179
```

In [ ]:
```
confusion_matrix(ytest, pred_tree)
```

Out[ ]:
```
array([[11867,   568],
       [ 2039,  1807]], dtype=int64)
```

In [ ]:
```
print(classification_report(ytest, pred_tree))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.95      0.90     12435
         1.0       0.76      0.47      0.58      3846

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

In [ ]:
```
# Create a RandomForest model and fit the training data
model_forest = RandomForestClassifier(criterion='entropy')
model_forest.fit(xtrain, ytrain)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy')
```

In [ ]:
```
# Predict results from the Random Forest using the test data
pred_forest = model_forest.predict(xtest)
```

In [ ]:
```
# What is the accuracy of the Random Forest on the test data?
accuracy_score(ytest, pred_forest)
```

Out[ ]:
```
0.82808181315644
```

In [ ]:
```
# Confustion matrix for the Random Forest
confusion_matrix(ytest, pred_forest)
```

Out[ ]:
```
array([[11491,   944],
       [ 1855,  1991]], dtype=int64)
```

In [ ]:
```
print(classification_report(ytest, pred_forest))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.92      0.89     12435
         1.0       0.68      0.52      0.59      3846

    accuracy                           0.83     16281
   macro avg       0.77      0.72      0.74     16281
weighted avg       0.82      0.83      0.82     16281
```

Conclusion

I could not, in fact, show that the RandomForest outperforms the DecisionTree for a fixed **max_depth**. Setting the **max_depth** to 10, a relatively low number, resulted in better accuracy, and mixed results for precision, recall, and the f1-score. Consider, first the accuracy scores below:

In [ ]:
```
results = {
    "Accuracy": [accuracy_score(xt.salary, predictions),
                 accuracy_score(ytest, pred_tree),
                 accuracy_score(ytest, pred_forest)]
```

```
}
pd.DataFrame(results, index=["Open Tree", "Fixed Tree (10)", "Random Forest"])
```

Out[ ]:

	Accuracy
Open Tree	0.821080
Fixed Tree (10)	0.839875
Random Forest	0.828082

Based on the above, we can see that the fixed depth Decision Tree outperformed both the open depth Decision Tree and the Random Forest!

We can also compare the confusion matrices for each model.

In [ ]:
```
confusion_matrix(ytest, pred_tree), confusion_matrix(ytest, pred_forest)
```

Out[ ]:
```
(array([[11867,   568],
        [ 2039,  1807]], dtype=int64),
 array([[11491,   944],
        [ 1855,  1991]], dtype=int64))
```

In the first matrix, for the Decision Tree, the True Negatives come across better than for the Random Forest, while the True Positives is lower than the Random Forest. We see similar mixed results for the False Negatives (fewer than the Random Forest), and the False Positives (more than the Random Forest). These mixed results beteen the two models carries through with the *porecision*, *recall*, and *f1-scores* as we can see in the following tables. First, the results for **salary <= $50k** (value = 0):

In [ ]:
```
report_tree = classification_report(ytest, pred_tree, output_dict=True)
report_forest = classification_report(ytest, pred_forest, output_dict=True)

results = {
    "Precision": [report_tree['0.0']['precision'], report_forest['0.0']['precision']],
    "Recall": [report_tree['0.0']['recall'], report_forest['0.0']['recall']],
    "F1-Score": [report_tree['0.0']['f1-score'], report_forest['0.0']['f1-score']]
}
pd.DataFrame(results, index = ["Decision Tree", "Random Forest"])
```

Out[ ]:

	Precision	Recall	F1-Score
Decision Tree	0.853373	0.954322	0.901029
Random Forest	0.861007	0.924085	0.891432

Here, the Decision Tree shows better results for both Recall and F1-Scores, and only slightly worse results for the Precision. Now, consider **salary > $50k** (value 1):

In [ ]:
```
results = {
    "Precision": [report_tree['1.0']['precision'], report_forest['1.0']['precision']],
    "Recall": [report_tree['1.0']['recall'], report_forest['1.0']['recall']],
    "F1-Score": [report_tree['1.0']['f1-score'], report_forest['1.0']['f1-score']]
}
pd.DataFrame(results, index = ["Decision Tree", "Random Forest"])
```

Out[ ]:

	Precision	Recall	F1-Score
Decision Tree	0.760842	0.469839	0.580936
Random Forest	0.678365	0.517681	0.587229

Here we see the situation reversed, with the Decision Tree far outperforming the Random Forest on Precision, while underperforming on Recall and F1-Score.

All of this leads me to conclude that neither of these models is significantly better than the other, and that the fixed-depth Decision Tree may be sligtly *better* than the Random Forest.

2. Use a RandomForest or DecisionTree and the `adult` dataset, systematically add new columns, one by one, that are non-numerical but converted using the feature-extraction techniques we learned. Using the golden-test set show [`precision`, `recall`, `f1`, `confusion matrix`] for each additional feature added.

Start with existing dataset based on previous exercise.

In [ ]:
```
xtrain1 = xtrain.copy()
xtest1 = xtest.copy()
```

Run our initial model to get results.

In [ ]:
```
model1 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model1.fit(xtrain1, ytrain)

# Predict results from the Decision Tree using the test data
pred1 = model1.predict(xtest1)
accuracy_score(ytest, pred1)
```

Out[ ]:
```
0.8211412075425343
```

Note that throughout, I will collect the output from the confusion matrix and the classification report to review at the end, in the **Results** section.

In [ ]:
```
# Store results of the confusion_matrix and classification_report in variables
# for future reference.
conf1 = confusion_matrix(ytest, pred1)
report1 = classification_report(ytest, pred1, output_dict=True)
```

Transform an additional column: **workclass**.

In [ ]:
```
# First the training dataset:
# Fit the encoder to the workclass column from the original data,
# and transform it into numerical values.
enc.fit(df[['workclass']])
transformed = enc.transform(df[['workclass']])
#enc.categories_[0]
# Change the output to a dataframe
df_trans = pd.DataFrame(transformed, columns=["workclass"])
# Concatenate on to the end of the previous training data
# to get a new set of training data.
xtrain2 = pd.concat([xtrain1, df_trans], axis=1)
#enc.categories_[0]
```

In [ ]:
```
# Now the testing dataset
enc.fit(golden[['workclass']])
transformed = enc.transform(golden[['workclass']])
#enc.categories_[0]
df_trans = pd.DataFrame(transformed, columns=["workclass"])

xtest2 = pd.concat([xtest1, df_trans], axis=1)

#enc.categories_[0]
```

Run a new Classification test and get results.

In [ ]:
```
model2 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model2.fit(xtrain2, ytrain)

# Predict results from the Decision Tree using the test data
pred2 = model2.predict(xtest2)
accuracy_score(ytest, pred2)
```

Out[ ]:
```
0.814262023217247
```

In [ ]:
```
conf2 = confusion_matrix(ytest, pred2)
report2 = classification_report(ytest, pred2, output_dict=True)
```

Repeat with **education**.

In [ ]:
```
# Training
enc.fit(df[['education']])
transformed = enc.transform(df[['education']])
df_trans = pd.DataFrame(transformed, columns=["education"])
xtrain3 = pd.concat([xtrain2, df_trans], axis=1)

# Testing
enc.fit(golden[['education']])
transformed = enc.transform(golden[['education']])
df_trans = pd.DataFrame(transformed, columns=["education"])
xtest3 = pd.concat([xtest2, df_trans], axis=1)

model3 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model3.fit(xtrain3, ytrain)
```

```
pred3 = model3.predict(xtest3)
accuracy_score(ytest, pred3)
```

Out[ ]:
```
0.8146919722375775
```

In [ ]:
```
conf3 = confusion_matrix(ytest, pred3)
report3 = classification_report(ytest, pred3, output_dict=True)
```

Repeat with **marital-status**.

In [ ]:
```
# Training
enc.fit(df[['marital-status']])
transformed = enc.transform(df[['marital-status']])
df_trans = pd.DataFrame(transformed, columns=["marital-status"])
xtrain4 = pd.concat([xtrain3, df_trans], axis=1)

# Testing
enc.fit(golden[['marital-status']])
transformed = enc.transform(golden[['marital-status']])
df_trans = pd.DataFrame(transformed, columns=["marital-status"])
xtest4 = pd.concat([xtest3, df_trans], axis=1)

model4 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model4.fit(xtrain4, ytrain)
pred4 = model4.predict(xtest4)
accuracy_score(ytest, pred4)
```

Out[ ]:
```
0.8284503408881518
```

In [ ]:
```
conf4 = confusion_matrix(ytest, pred4)
report4 = classification_report(ytest, pred4, output_dict=True)
```

Repeat with **occupation**.

In [ ]:
```
# Training
enc.fit(df[['occupation']])
transformed = enc.transform(df[['occupation']])
df_trans = pd.DataFrame(transformed, columns=["occupation"])
xtrain5 = pd.concat([xtrain4, df_trans], axis=1)

# Testing
enc.fit(golden[['occupation']])
transformed = enc.transform(golden[['occupation']])
df_trans = pd.DataFrame(transformed, columns=["occupation"])
xtest5 = pd.concat([xtest4, df_trans], axis=1)

model5 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model5.fit(xtrain5, ytrain)
pred5 = model5.predict(xtest5)
accuracy_score(ytest, pred5)
```

Out[ ]:
```
0.8175173515140348
```

In [ ]:
```
conf5 = confusion_matrix(ytest, pred5)
report5 = classification_report(ytest, pred5, output_dict=True)
```

Repeat with **relationship**.

In [ ]:
```
# Training
enc.fit(df[['relationship']])
transformed = enc.transform(df[['relationship']])
df_trans = pd.DataFrame(transformed, columns=["relationship"])
xtrain6 = pd.concat([xtrain5, df_trans], axis=1)

# Testing
enc.fit(golden[['relationship']])
transformed = enc.transform(golden[['relationship']])
df_trans = pd.DataFrame(transformed, columns=["relationship"])
xtest6 = pd.concat([xtest5, df_trans], axis=1)

model6 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model6.fit(xtrain6, ytrain)
pred6 = model6.predict(xtest6)
accuracy_score(ytest, pred6)
```

Out[ ]:
```
0.8194214114612125
```

In [ ]:
```
conf6 = confusion_matrix(ytest, pred6)
report6 = classification_report(ytest, pred6, output_dict=True)
```

Repeat with **race**.

In [ ]:
```
# Training
enc.fit(df[['race']])
transformed = enc.transform(df[['race']])
df_trans = pd.DataFrame(transformed, columns=["race"])
xtrain7 = pd.concat([xtrain6, df_trans], axis=1)

# Testing
enc.fit(golden[['race']])
transformed = enc.transform(golden[['race']])
df_trans = pd.DataFrame(transformed, columns=["race"])
xtest7 = pd.concat([xtest6, df_trans], axis=1)

model7 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model7.fit(xtrain7, ytrain)
pred7 = model7.predict(xtest7)
accuracy_score(ytest, pred7)
```

Out[ ]:
```
0.8197285179043057
```

In [ ]:
```
conf7 = confusion_matrix(ytest, pred7)
report7 = classification_report(ytest, pred7, output_dict=True)
```

Repeat with **native-country**.

In [ ]:
```
# Training
enc.fit(df[['native-country']])
transformed = enc.transform(df[['native-country']])
df_trans = pd.DataFrame(transformed, columns=["native-country"])
xtrain8 = pd.concat([xtrain7, df_trans], axis=1)
```

```
# Testing
enc.fit(golden[['native-country']])
transformed = enc.transform(golden[['native-country']])
df_trans = pd.DataFrame(transformed, columns=["native-country"])
xtest8 = pd.concat([xtest7, df_trans], axis=1)

model8 = DecisionTreeClassifier(criterion='entropy', max_depth=None)
model8.fit(xtrain8, ytrain)
pred8 = model8.predict(xtest8)
accuracy_score(ytest, pred8)
```

Out[ ]:
```
0.8189914624408821
```

In [ ]:
```
conf8 = confusion_matrix(ytest, pred8)
report8 = classification_report(ytest, pred8, output_dict=True)
```

Results

Now we can look at the results of all of the models built and compare. We start with the accuracy scores.

In [ ]:
```
colnames = ["sex", "workclass", "education", "marital-status",
            "occupation", "relationship", "race", "native-country"]

results = {
    "Accuracy": [accuracy_score(ytest, pred1),
                 accuracy_score(ytest, pred2),
                 accuracy_score(ytest, pred3),
                 accuracy_score(ytest, pred4),
                 accuracy_score(ytest, pred5),
                 accuracy_score(ytest, pred6),
                 accuracy_score(ytest, pred7),
                 accuracy_score(ytest, pred8)]
}
pd.DataFrame(results, index=colnames)
```

Out[ ]:

	Accuracy
sex	0.821141
workclass	0.814262
education	0.814692
marital-status	0.828450
occupation	0.817517
relationship	0.819421
race	0.819729
native-country	0.818991

In the above table, each accuracy score is related to the model that added that particular column with an encoded value. It starts with our original model that only encoded (through OneHotEncoder), the **sex** column. After that, we added the **workclass**column with encoding,

added on to the model prior, and so on. As we build on each model, adding an encoded column, the peak accuracy appears to be when we encode **marital-status** with a near *83% accuracy*.

Next we can compare the results of the confusion matrices produced. IN the table below, I ahve flattened the matrices and labeled each column according to what it is showing: True Negatives, Fales Negatives, False Positives, and True Positives. Note that *Negative* refers to a salary value of 0, or <=*50k* and *Positive* refers to a salary value of 1, or >*50k*.

In [ ]:
```
confs = {
    "True Neg": [conf1[0,0], conf2[0,0], conf3[0,0], conf4[0,0],
                 conf5[0,0], conf6[0,0], conf7[0,0], conf8[0,0]],
    "False Neg": [conf1[0,1], conf2[0,1], conf3[0,1], conf4[0,1],
                  conf5[0,1], conf6[0,1], conf7[0,1], conf8[0,1]],
    "False Pos": [conf1[1,0], conf2[1,0], conf3[1,0], conf4[1,0],
                  conf5[1,0], conf6[1,0], conf7[1,0], conf8[1,0]],
    "True Pos": [conf1[1,1], conf2[1,1], conf3[1,1], conf4[1,1],
                 conf5[1,1], conf6[1,1], conf7[1,1], conf8[1,1]]
}
pd.DataFrame(confs, index=colnames)
```

Out[ ]:

	True Neg	False Neg	False Pos	True Pos
sex	11461	974	1938	1908
workclass	11307	1128	1896	1950
education	11318	1117	1900	1946
marital-status	11274	1161	1632	2214
occupation	11038	1397	1574	2272
relationship	11045	1390	1550	2296
race	11031	1404	1531	2315
native-country	11021	1414	1533	2313

Based on the confusion matrices, the number of True Negatives and True Positives continues to improve beyond the encoding of **marital-status**, peaking with **relationship** or **race**. The values drop again with **native-country**, which may indicate it is not a valuable column for inclusion in the model. We can check quickly based on the feature importances.

In [ ]:
```
list(zip(xtrain8.columns, model8.feature_importances_))
```

Out[ ]:
```
[('age', 0.19163345370620719),
 ('education-num', 0.11446515403377595),
 ('capital-gain', 0.11838688780352544),
 ('capital-loss', 0.040561997586636166),
 ('hours-per-week', 0.10681717288176383),
 (' Female', 0.0029634529231619845),
 (' Male', 0.002998991060058452),
 ('workclass', 0.049573828527634325),
 ('education', 0.01843854666765985),
 ('marital-status', 0.015307516338233958),
 ('occupation', 0.08307936116243719),
 ('relationship', 0.2141965260343729),
 ('race', 0.020678372869407734),
 ('native-country', 0.020898738405124868)]
```

Based on the above, both **race** and **native-country** seem to have a low influence, around 2%. Compare this to **relationship** with an influence of 21%! Based on this, and the results of the confusion matrices, I would stop at the encoding of **relationship**. I think it is also important to note the extremely low influence of the OneHotEncoded **sex** columns, at around 0.3%. One concern I have here is the multicollinearity resulting from the dummy variable trap, and at least one of these two columns should be excluded from the model.

Finally, let's look at the results of the classification reports. In this case, we will compare the results for **salary=0.0** (<=50k) in the first table, followed by **salary=1.0** (>50k) in the second table. This separation is simply for ease of comparison across models.

In [ ]:
```
# Compare Classification reports for salary=0.0
reports = {
    "Precision": [report1['0.0']['precision'], report2['0.0']['precision'], report3['0
                  report4['0.0']['precision'], report5['0.0']['precision'], report6['0
                  report7['0.0']['precision'], report8['0.0']['precision']],
    "Recall": [report1['0.0']['recall'], report2['0.0']['recall'], report3['0.0']['rec
               report4['0.0']['recall'], report5['0.0']['recall'], report6['0.0']['rec
               report7['0.0']['recall'], report8['0.0']['recall']],
    "F1-Score": [report1['0.0']['f1-score'], report2['0.0']['f1-score'], report3['0.0'
                 report4['0.0']['f1-score'], report5['0.0']['f1-score'], report6['0.0'
                 report7['0.0']['f1-score'], report8['0.0']['f1-score']]
}
pd.DataFrame(reports, index = colnames)
```

Out[ ]:

	Precision	Recall	F1-Score
sex	0.855362	0.921673	0.887280
workclass	0.856396	0.909288	0.882050
education	0.856257	0.910173	0.882392
marital-status	0.873547	0.906634	0.889783
occupation	0.875198	0.887656	0.881383
relationship	0.876935	0.888219	0.882541
race	0.878125	0.887093	0.882586
native-country	0.877888	0.886289	0.882068

For the salary values of **0.0**, the Precision appears to top out with the inclusion of **race**, much like the confusion matrices. However, the Recall peaks with **education**, and the F1-Score peaks at the inclusion of **marital-status**, aligning with the Accuracy scores we looked at earlier.

Let's compare this to the results for salary values of **1.0**.

In [ ]:
```
# Compare Classification reports for salary=0.0
reports = {
    "Precision": [report1['1.0']['precision'], report2['1.0']['precision'], report3['1
                  report4['1.0']['precision'], report5['1.0']['precision'], report6['1
                  report7['1.0']['precision'], report8['1.0']['precision']],
    "Recall": [report1['1.0']['recall'], report2['1.0']['recall'], report3['1.0']['rec
               report4['1.0']['recall'], report5['1.0']['recall'], report6['1.0']['rec
               report7['1.0']['recall'], report8['1.0']['recall']],
    "F1-Score": [report1['1.0']['f1-score'], report2['1.0']['f1-score'], report3['1.0'
                 report4['1.0']['f1-score'], report5['1.0']['f1-score'], report6['1.0'
                 report7['1.0']['f1-score'], report8['1.0']['f1-score']]
}
pd.DataFrame(reports, index = colnames)
```

Out[ ]:

	Precision	Recall	F1-Score
sex	0.662040	0.496100	0.567182
workclass	0.633528	0.507020	0.563258
education	0.635325	0.505980	0.563323
marital-status	0.656000	0.575663	0.613211
occupation	0.619242	0.590744	0.604657
relationship	0.622897	0.596984	0.609665
race	0.622479	0.601924	0.612029
native-country	0.620606	0.601404	0.610854

Here, Precision and the F1-Score are best with the includion of **marital-status**, but Recall continues to improve untill we reach the inclusion of **race**.

If we consider all of these results together - Accuracy, Precision, Recall, the F1-Score, and the Confusion Matrices, we might conclude that our best model is where we have included the encoding of **marital-status**. However, our *feature influences* also indicated a strong influence from **relationship** and little influence from the OneHotEncoded **sex** variables. Further refinement of this model may improve results, though not drastically.