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

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

In [4]:
```
golden = pd.read_csv('../data/adult.test', index_col=False)
```

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

Out[5]:

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

In [6]:
```
df.head()
```

Out[6]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

In [7]:
```
df.columns
```

Out[7]:
```
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 [8]:
```
from sklearn import preprocessing
```

In [9]:
```
# 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 [10]:
```
dummies = pd.get_dummies(df[transform_columns])
dummies
```

Out[10]:

	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 [11]:
```
dummies.shape
```

Out[11]:
```
(32561, 2)
```

sklearn has a similar process for OneHot Encoding features

In [12]:
```
onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist", sparse_output=False)
onehot.fit(df[transform_columns])
```

Out[12]:
```
OneHotEncoder(handle_unknown='infrequent_if_exist', sparse_output=False)
```

In [13]:
```
onehot.categories_
```

Out[13]:
```
[array([' Female', ' Male'], dtype=object)]
```

In [14]:
```
sex = onehot.transform(df[transform_columns])
sex
```

Out[14]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       ...,
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

In [15]:
```
sex.shape
```

Out[15]:
```
(32561, 2)
```

In addition to OneHot encoding there is Ordinal Encoding

In [16]:
```
enc = preprocessing.OrdinalEncoder()
enc.fit(df[["salary"]])
salary = enc.transform(df[["salary"]])
salary
```

Out[16]:
```
array([[0.],
       [0.],
       [0.],
       ...,
       [0.],
       [0.],
       [1.]])
```

In [17]:
```
enc.categories_[0]
```

Out[17]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [18]:
```
x = df.copy()

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


onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist", sparse_output=False).fit(df[transform_columns])

enc = preprocessing.OrdinalEncoder()

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


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


x = pd.concat(
    [
        x.drop(non_num_columns, axis=1),
        df_trans
    ],
    axis=1,)


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

In [19]:
```
x.head()
```

Out[19]:

	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

In [20]:
```
xt = golden.copy()

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

xt = pd.concat(
    [
        xt.drop(non_num_columns, axis=1),
        df_trans
    ],
    axis=1,)

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

In [21]:
```
xt.salary.value_counts()
```

Out[21]:
```
salary
0.0    12435
1.0     3846
Name: count, dtype: int64
```

In [22]:
```
enc.categories_
```

Out[22]:
```
[array([' <=50K.', ' >50K.'], dtype=object)]
```

In [23]:
```
xt.head()
```

Out[23]:

	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 [24]:
```
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 [25]:
```
model = RandomForestClassifier(criterion='entropy')
```

In [26]:
```
model = DecisionTreeClassifier(criterion='entropy', max_depth=None)
```

In [27]:
```
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[27]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [28]:
```
model.tree_.node_count
```

Out[28]:
```
8335
```

In [29]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.feature_importances_))
```

Out[29]:
```
[('age', 0.3220820489868926),
 ('education-num', 0.1611791386086972),
 ('capital-gain', 0.2276201906115405),
 ('capital-loss', 0.07816785519359692),
 ('hours-per-week', 0.15573221153915023),
 (' Female', 0.02144509828139056),
 (' Male', 0.03377345677873193)]
```

In [30]:
```
list(zip(xt.drop(['fnlwgt','salary'], axis=1).columns, model.feature_importances_))
```

Out[30]:
```
[('age', 0.3220820489868926),
 ('education-num', 0.1611791386086972),
 ('capital-gain', 0.2276201906115405),
 ('capital-loss', 0.07816785519359692),
 ('hours-per-week', 0.15573221153915023),
 (' Female', 0.02144509828139056),
 (' Male', 0.03377345677873193)]
```

In [31]:
```
x.drop(['fnlwgt','salary'], axis=1).head()
```

Out[31]:

	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 [32]:
```
xt.drop(['fnlwgt','salary'], axis=1).head()
```

Out[32]:

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

In [33]:
```
set(x.columns) - set(xt.columns)
```

Out[33]:
```
set()
```

In [34]:
```
list(x.drop('salary', axis=1).columns)
```

Out[34]:
```
['age',
 'fnlwgt',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 ' Female',
 ' Male']
```

In [35]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [36]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [37]:
```
accuracy_score(xt.salary, predictions)
```

Out[37]:
```
0.8206498372335852
```

In [38]:
```
accuracy_score(xt.salary, predictions)
```

Out[38]:
```
0.8206498372335852
```

In [39]:
```
confusion_matrix(xt.salary, predictions)
```

Out[39]:
```
array([[11457,   978],
       [ 1942,  1904]], dtype=int64)
```

In [40]:
```
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 [41]:
```
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 [42]:
```
accuracy_score(x.salary, predictionsx)
```

Out[42]:
```
0.8955806025613464
```

In [43]:
```
confusion_matrix(x.salary, predictionsx)
```

Out[43]:
```
array([[24097,   623],
       [ 2777,  5064]], dtype=int64)
```

In [44]:
```
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 [45]:
```
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 [289…
```
train = df.copy()
train = train[(train['native-country'] != " Holand-Netherlands")]
train = train.dropna()
train = train.reset_index(drop=True)
test = golden.copy()
```

In [290…
```
train = train.replace(" ?", np.NaN)
test = test.replace(" ?", np.NaN)
```

In [291…
```
onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist", sparse_output=False).fit(df[transform_columns])
enc = preprocessing.OrdinalEncoder()

enc.fit(train[["salary"]])
train["salary"] = enc.transform(train[["salary"]])

enc.fit(test[["salary"]])
test["salary"] = enc.transform(test[["salary"]])
```

In [292…
```
train_small = train[["age","education-num","capital-gain","capital-loss","hours-per-week"]]
test_small = test[["age","education-num","capital-gain","capital-loss","hours-per-week"]]
```

Random Forest Model

In [293…
```
model_rand = RandomForestClassifier(criterion='entropy', max_depth=7)
model_rand.fit(train_small, train.salary)
rand_pred = model_rand.predict(test_small)
```

In [294…
```
    confusion_matrix(test.salary, rand_pred)
```

Out[294]:
```
array([[11950,   485],
       [ 2188,  1658]], dtype=int64)
```

In [295…
```
print(classification_report(test.salary, rand_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.96      0.90     12435
         1.0       0.77      0.43      0.55      3846

    accuracy                           0.84     16281
   macro avg       0.81      0.70      0.73     16281
weighted avg       0.83      0.84      0.82     16281
```

Decision Tree Model

In [296…
```
model_dec = DecisionTreeClassifier(criterion='entropy', max_depth=7)
model_dec.fit(train_small, train.salary)
dec_pred = model_dec.predict(test_small)
```

In [297…
```
confusion_matrix(test.salary, dec_pred)
```

Out[297]:
```
array([[12005,   430],
       [ 2307,  1539]], dtype=int64)
```

In [298…
```
print(classification_report(test.salary, dec_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.97      0.90     12435
         1.0       0.78      0.40      0.53      3846

    accuracy                           0.83     16281
   macro avg       0.81      0.68      0.71     16281
weighted avg       0.83      0.83      0.81     16281
```

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.

In [288…
```
for variable in non_num_columns:
    new_column = [variable]
    onehot.fit(train[new_column])
    transformed = onehot.transform(train[new_column])
    new_cols = list(onehot.categories_[0].flatten())
    df_trans = pd.DataFrame(transformed, columns=new_cols)

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

    onehot.fit(test[new_column])
    transformed = onehot.transform(test[new_column])
    new_cols = list(onehot.categories_[0].flatten())
    df_trans = pd.DataFrame(transformed, columns=new_cols)

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

    model_rand = RandomForestClassifier(criterion='entropy', max_depth=7)
    model_rand.fit(train_small, train.salary)
    rand_pred = model_rand.predict(test_small)

    print(f"Confusion Matrix and Classification Report with {variable} variables added")
    print(confusion_matrix(test.salary, rand_pred), classification_report(test.salary, rand_pred))
```

```
Confusion Matrix and Classification Report with workclass variables added
[[12121   314]
 [ 2358  1488]]               precision    recall  f1-score   support

         0.0       0.84      0.97      0.90     12435
         1.0       0.83      0.39      0.53      3846

    accuracy                           0.84     16281
   macro avg       0.83      0.68      0.71     16281
weighted avg       0.83      0.84      0.81     16281

Confusion Matrix and Classification Report with education variables added
[[12219   216]
 [ 2580  1266]]               precision    recall  f1-score   support

         0.0       0.83      0.98      0.90     12435
         1.0       0.85      0.33      0.48      3846

    accuracy                           0.83     16281
   macro avg       0.84      0.66      0.69     16281
weighted avg       0.83      0.83      0.80     16281

Confusion Matrix and Classification Report with marital-status variables added
[[11874   561]
 [ 1806  2040]]               precision    recall  f1-score   support

         0.0       0.87      0.95      0.91     12435
         1.0       0.78      0.53      0.63      3846

    accuracy                           0.85     16281
   macro avg       0.83      0.74      0.77     16281
weighted avg       0.85      0.85      0.84     16281

Confusion Matrix and Classification Report with occupation variables added
[[11965   470]
 [ 1902  1944]]               precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     12435
         1.0       0.81      0.51      0.62      3846

    accuracy                           0.85     16281
   macro avg       0.83      0.73      0.77     16281
weighted avg       0.85      0.85      0.84     16281

Confusion Matrix and Classification Report with relationship variables added
[[11901   534]
 [ 1858  1988]]               precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     12435
         1.0       0.79      0.52      0.62      3846

    accuracy                           0.85     16281
   macro avg       0.83      0.74      0.77     16281
weighted avg       0.85      0.85      0.84     16281

Confusion Matrix and Classification Report with race variables added
[[11926   509]
 [ 1922  1924]]               precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     12435
         1.0       0.79      0.50      0.61      3846

    accuracy                           0.85     16281
   macro avg       0.83      0.73      0.76     16281
weighted avg       0.84      0.85      0.84     16281

Confusion Matrix and Classification Report with sex variables added
[[11903   532]
 [ 1863  1983]]               precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     12435
         1.0       0.79      0.52      0.62      3846

    accuracy                           0.85     16281
   macro avg       0.83      0.74      0.77     16281
weighted avg       0.85      0.85      0.84     16281

Confusion Matrix and Classification Report with native-country variables added
[[11933   502]
 [ 1986  1860]]               precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     12435
         1.0       0.79      0.48      0.60      3846

    accuracy                           0.85     16281
   macro avg       0.82      0.72      0.75     16281
```

weighted avg	0.84	0.85	0.83	16281