- https://scikit-learn.org/stable/modules/tree.html (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.DecisionTreeClassifier.html)
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.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	se
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Mal
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Mal
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Mal
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Mal
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Femal

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]:
```
# Encode categorical features as an integer array.
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

- Copy dummy values back to df
- delete non numerical features

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

x = x.drop(non_num_columns, axis=1)

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

In [12]:
```
x.salary.value_counts()
```

Out[12]:
```
0.0    24720
1.0     7841
Name: salary, dtype: int64
```

In [13]:
```
x.head()
```

Out[13]:

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

In [14]:
```
xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
```

In [15]:
```
xt.salary.value_counts()
```

Out[15]:
```
0.0    12435
1.0     3846
Name: salary, dtype: int64
```

In [16]:
```
enc.categories_
```

Out[16]:
```
[array([' <=50K.', ' >50K.'], dtype=object)]
```

In [17]:
```
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 [18]:
```
model = DecisionTreeClassifier(criterion='entropy', max_depth=None)
```

In [19]:
```
# dropping features from training dataset
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

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

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

Out[20]:
```
7465
```

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

Out[21]:
```
[('age', 0.32665352638151884),
 ('education-num', 0.16621219584906177),
 ('capital-gain', 0.24827550894767372),
 ('capital-loss', 0.0982141790631074),
 ('hours-per-week', 0.16064458975863827)]
```

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

Out[22]:
```
[('age', 0.32665352638151884),
 ('education-num', 0.16621219584906177),
 ('capital-gain', 0.24827550894767372),
 ('capital-loss', 0.0982141790631074),
 ('hours-per-week', 0.16064458975863827)]
```

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

Out[23]:

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

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']
```

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]:
```
0.8165960321847552
```

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

Out[29]:
```
0.8165960321847552
```

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

Out[30]:
```
array([[11510,   925],
       [ 2061,  1785]], dtype=int64)
```

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

Out[31]:
```
array([[11510,   925],
       [ 2061,  1785]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

         0.0       0.85      0.93      0.89     12435
         1.0       0.66      0.46      0.54      3846

    accuracy                           0.82     16281
   macro avg       0.75      0.69      0.71     16281
weighted avg       0.80      0.82      0.80     16281
```

In [33]:
```
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.93      0.89     12435
         1.0       0.66      0.46      0.54      3846

    accuracy                           0.82     16281
   macro avg       0.75      0.69      0.71     16281
weighted avg       0.80      0.82      0.80     16281
```

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

Out[34]:
```
0.8841558920180584
```

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

Out[35]:
```
array([[24136,   584],
       [ 3188,  4653]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

         0.0       0.88      0.98      0.93     24720
         1.0       0.89      0.59      0.71      7841

    accuracy                           0.88     32561
   macro avg       0.89      0.78      0.82     32561
weighted avg       0.88      0.88      0.88     32561
```

In [37]:
```
print(classification_report(x.salary, predictionsx))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.98      0.93     24720
         1.0       0.89      0.59      0.71      7841

    accuracy                           0.88     32561
   macro avg       0.89      0.78      0.82     32561
weighted avg       0.88      0.88      0.88     32561
```

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

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 [38]:
```
# see 12:40 of lecture
model = RandomForestClassifier(criterion='entropy')
```

In [39]:
```
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[39]:
```
RandomForestClassifier(criterion='entropy')
```

In [40]:
```
# Will only work for one tree.  this will not work for RandomForest because there
# model.tree_.node_count
```

In [41]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.feature_importances_)
```

Out[41]:
```
[('age', 0.34395059292891417),
 ('education-num', 0.17390557988677033),
 ('capital-gain', 0.2100042038246609),
 ('capital-loss', 0.08326587176902904),
 ('hours-per-week', 0.18887375159062547)]
```

In [42]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [43]:
```
accuracy_score(xt.salary, predictions)
```

Out[43]:
```
0.8249493274368896
```

In [44]:
```
confusion_matrix(xt.salary, predictions)
```

Out[44]:
```
array([[11610,   825],
       [ 2025,  1821]], dtype=int64)
```

In [45]:
```
Rpt_RndFrst_00 = classification_report(xt.salary, predictions)
print(Rpt_RndFrst_00)
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.93      0.89     12435
         1.0       0.69      0.47      0.56      3846

    accuracy                           0.82     16281
   macro avg       0.77      0.70      0.73     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.

Add 'workclass' column

In [46]:
```
non_num_columns = ['education', 'marital-status', 'sex', 'occupation', 'relations
```

In [47]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
```

In [48]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[48]:
```
RandomForestClassifier(criterion='entropy')
```

In [49]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [50]:
```
Rpt_RndFrst_01 = (classification_report(xt.salary, predictions))
```

Add 'marital-status' column

In [51]:
```
non_num_columns = ['education', 'sex', 'occupation', 'relationship', 'race', 'nat
```

In [52]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])
x["marital-status"] = enc.fit_transform(df[["marital-status"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
xt["marital-status"] = enc.fit_transform(golden[["marital-status"]])
```

In [53]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[53]:
```
RandomForestClassifier(criterion='entropy')
```

In [54]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [55]:
```
Rpt_RndFrst_02 = classification_report(xt.salary, predictions)
```

Add 'sex' column

In [56]:
```
non_num_columns = ['education', 'occupation', 'relationship', 'race', 'native-cou
```

In [57]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])
x["marital-status"] = enc.fit_transform(df[["marital-status"]])
x["sex"] = enc.fit_transform(df[["sex"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
xt["marital-status"] = enc.fit_transform(golden[["marital-status"]])
xt["sex"] = enc.fit_transform(golden[["sex"]])
```

In [58]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[58]:
```
RandomForestClassifier(criterion='entropy')
```

In [59]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [60]:
```
Rpt_RndFrst_03 = classification_report(xt.salary, predictions)
```

Add 'occupation' column

In [61]:
```
non_num_columns = ['education', 'relationship', 'race', 'native-country']
```

In [62]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])
x["marital-status"] = enc.fit_transform(df[["marital-status"]])
x["sex"] = enc.fit_transform(df[["sex"]])
x["occupation"] = enc.fit_transform(df[["occupation"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
xt["marital-status"] = enc.fit_transform(golden[["marital-status"]])
xt["sex"] = enc.fit_transform(golden[["sex"]])
xt["occupation"] = enc.fit_transform(golden[["occupation"]])
```

In [63]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[63]:
```
RandomForestClassifier(criterion='entropy')
```

In [64]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [65]:
```
Rpt_RndFrst_04 = classification_report(xt.salary, predictions)
```

Add 'relationship' column

In [66]:
```
non_num_columns = ['education', 'race', 'native-country']
```

In [67]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])
x["marital-status"] = enc.fit_transform(df[["marital-status"]])
x["sex"] = enc.fit_transform(df[["sex"]])
x["occupation"] = enc.fit_transform(df[["occupation"]])
x["relationship"] = enc.fit_transform(df[["relationship"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
xt["marital-status"] = enc.fit_transform(golden[["marital-status"]])
xt["sex"] = enc.fit_transform(golden[["sex"]])
xt["occupation"] = enc.fit_transform(golden[["occupation"]])
xt["relationship"] = enc.fit_transform(golden[["relationship"]])
```

In [68]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[68]:
```
RandomForestClassifier(criterion='entropy')
```

In [69]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [70]:
```
Rpt_RndFrst_05 = classification_report(xt.salary, predictions)
```

Add 'race' column

In [71]:
```
non_num_columns = ['education', 'native-country']
```

In [72]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])
x["marital-status"] = enc.fit_transform(df[["marital-status"]])
x["sex"] = enc.fit_transform(df[["sex"]])
x["occupation"] = enc.fit_transform(df[["occupation"]])
x["relationship"] = enc.fit_transform(df[["relationship"]])
x["race"] = enc.fit_transform(df[["race"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
xt["marital-status"] = enc.fit_transform(golden[["marital-status"]])
xt["sex"] = enc.fit_transform(golden[["sex"]])
xt["occupation"] = enc.fit_transform(golden[["occupation"]])
xt["relationship"] = enc.fit_transform(golden[["relationship"]])
xt["race"] = enc.fit_transform(golden[["race"]])
```

In [73]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[73]:
```
RandomForestClassifier(criterion='entropy')
```

In [74]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [75]:
```
Rpt_RndFrst_06 = classification_report(xt.salary, predictions)
```

In [76]:
```
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.92      0.90     12435
         1.0       0.71      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
```

Add 'native-country' column

In [77]:
```
non_num_columns = ['education'] # Note that education already encoded under colum
```

In [86]:
```
x = df.copy()
x = x.drop(non_num_columns, axis=1)
x["salary"] = enc.fit_transform(df[["salary"]])
x["workclass"] = enc.fit_transform(df[["workclass"]])
x["marital-status"] = enc.fit_transform(df[["marital-status"]])
x["sex"] = enc.fit_transform(df[["sex"]])
x["occupation"] = enc.fit_transform(df[["occupation"]])
x["relationship"] = enc.fit_transform(df[["relationship"]])
x["race"] = enc.fit_transform(df[["race"]])
x["native-country"] = enc.fit_transform(df[["native-country"]])

xt = golden.copy()
xt = xt.drop(non_num_columns, axis=1)
xt["salary"] = enc.fit_transform(golden[["salary"]])
xt["workclass"] = enc.fit_transform(golden[["workclass"]])
xt["marital-status"] = enc.fit_transform(golden[["marital-status"]])
xt["sex"] = enc.fit_transform(golden[["sex"]])
xt["occupation"] = enc.fit_transform(golden[["occupation"]])
xt["relationship"] = enc.fit_transform(golden[["relationship"]])
xt["race"] = enc.fit_transform(golden[["race"]])
xt["native-country"] = enc.fit_transform(golden[["native-country"]])
```

In [87]:
```
model = RandomForestClassifier(criterion='entropy')
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

Out[87]:
```
RandomForestClassifier(criterion='entropy')
```

In [88]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

In [89]:
```
Rpt_RndFrst_07 = classification_report(xt.salary, predictions)
```

Print all classifcation reports

In [90]:
```
print(Rpt_RndFrst_00)
print(Rpt_RndFrst_01)
print(Rpt_RndFrst_02)
print(Rpt_RndFrst_03)
print(Rpt_RndFrst_04)
print(Rpt_RndFrst_05)
print(Rpt_RndFrst_06)
print(Rpt_RndFrst_07)
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.93      0.89     12435
         1.0       0.69      0.47      0.56      3846

    accuracy                           0.82     16281
   macro avg       0.77      0.70      0.73     16281
weighted avg       0.81      0.82      0.81     16281

              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

              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

              precision    recall  f1-score   support

         0.0       0.88      0.92      0.90     12435
         1.0       0.70      0.60      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

              precision    recall  f1-score   support

         0.0       0.89      0.92      0.90     12435
         1.0       0.71      0.62      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

              precision    recall  f1-score   support

         0.0       0.89      0.92      0.90     12435
```

```
           1.0       0.70      0.62      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

              precision    recall  f1-score   support

         0.0       0.89      0.92      0.90     12435
         1.0       0.71      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

              precision    recall  f1-score   support

         0.0       0.88      0.93      0.91     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       0.85      0.85      0.85     16281
```

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

In [141]:
```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
```

In [155]:
```
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
```

In [156]:
```
clf = GridSearchCV(model, parameters)
```

In [160]:
```
clf.fit(x.drop(['salary'], axis=1), x.salary)
sorted(clf.cv_results_.keys())
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-160-c0b0011d2d6b> in <module>
----> 1 clf.fit(x.drop(['salary'], axis=1), x.salary)
      2 sorted(clf.cv_results_.keys())

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, *
*kwargs)
     71                           FutureWarning)
     72         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)
})
---> 73         return f(**kwargs)
     74     return inner_f
     75

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self,
X, y, groups, **fit_params)
    734                 return results
    735
--> 736             self._run_search(evaluate_candidates)
    737
    738         # For multi-metric evaluation, store the best_index_, best_pa
rams_ and

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in _run_sear
ch(self, evaluate_candidates)
   1186     def _run_search(self, evaluate_candidates):
   1187         """Search all candidates in param_grid"""
-> 1188         evaluate_candidates(ParameterGrid(self.param_grid))
   1189
   1190

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in evaluate_
candidates(candidate_params)
    713                                for parameters, (train, test)
    714                                in product(candidate_params,
--> 715                                           cv.split(X, y, groups)))
    716
    717                 if len(out) < 1:

~\anaconda3\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1002             # remaining jobs.
   1003             self._iterating = False
-> 1004             if self.dispatch_one_batch(iterator):
   1005                 self._iterating = self._original_iterator is not None
   1006

~\anaconda3\lib\site-packages\joblib\parallel.py in dispatch_one_batch(self,
 iterator)
    833                 return False
    834             else:
--> 835                 self._dispatch(tasks)
    836                 return True
    837
```

```
~\anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self, batch)
    752         with self._lock:
    753             job_idx = len(self._jobs)
--> 754             job = self._backend.apply_async(batch, callback=cb)
    755             # A job can complete so quickly than its callback is
    756             # called before we get here, causing self._jobs to

~\anaconda3\lib\site-packages\joblib\_parallel_backends.py in apply_async(sel
f, func, callback)
    207     def apply_async(self, func, callback=None):
    208         """Schedule a func to be run"""
--> 209         result = ImmediateResult(func)
    210         if callback:
    211             callback(result)

~\anaconda3\lib\site-packages\joblib\_parallel_backends.py in __init__(self,
 batch)
    588         # Don't delay the application, to avoid keeping the input
    589         # arguments in memory
--> 590         self.results = batch()
    591
    592     def get(self):

~\anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    254         with parallel_backend(self._backend, n_jobs=self._n_jobs):
    255             return [func(*args, **kwargs)
--> 256                     for func, args, kwargs in self.items]
    257
    258     def __len__(self):

~\anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    254         with parallel_backend(self._backend, n_jobs=self._n_jobs):
    255             return [func(*args, **kwargs)
--> 256                     for func, args, kwargs in self.items]
    257
    258     def __len__(self):

~\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py in _fit_
and_score(estimator, X, y, scorer, train, test, verbose, parameters, fit_para
ms, return_train_score, return_parameters, return_n_test_samples, return_time
s, return_estimator, error_score)
    518             cloned_parameters[k] = clone(v, safe=False)
    519
--> 520         estimator = estimator.set_params(**cloned_parameters)
    521
    522     start_time = time.time()

~\anaconda3\lib\site-packages\sklearn\base.py in set_params(self, **params)
    250                                  'Check the list of available paramet
ers '
    251                                  'with `estimator.get_params().keys()
`.' %
--> 252                                  (key, self))
    253
    254             if delim:
```

```
ValueError: Invalid parameter C for estimator RandomForestClassifier(criterio
n='entropy'). Check the list of available parameters with `estimator.get_para
ms().keys()`.
```