PCS5024 - Aprendizado Estatístico

Name: Rodrigo da Silva Cunha

NUSP: 7631302

Building Classifiers for Adult Dataset

In this notebook we'll explore the Adult Dataset, available for download at the UCI Repository, and build a few classifiers to predict the class variable ">50k", which defines if a person gets more than US$50k/year or not.

The work reported in this notebook is part of a practice assignment in the discipline PCS5024 - Aprendizado Estatístico. The complete code is avaialable at GitHub.

1. Loading Adult Dataset

The Adult dataset is composed of a three documents:

- **adult.names**: contains general information about the dataset, such as column names, classification benchmarks, information about missing data and the dataset probability for the class labels.
- **adult.train**: random split containing 2/3 of the original dataset
- **adult.test**: random split containing 1/3 of the original dataset

Below we load the train and test datasets treating the '?' characters present in the dataset as missing data.

In [1]:

```
import pandas as pd

column_names = ['age','workclass','fnlwgt','education','education-num','marital-
                'occupation','relationship','race','sex','capital-gain','capital
                'hours-per-week','native-country','income']

train_data = pd.read_csv('../data/01_raw/adult.data',
                         index_col = False,
                         na_values = ' ?',
                         names = column_names)

test_data = pd.read_csv('../data/01_raw/adult.test',
                        index_col = False,
                        skiprows = 1,
                        na_values = ' ?',
                        names = column_names)
```

2. Exploratory Data Analysis (EDA)

In this section we perform some analysis to better describe and understand the Adult dataset. In the following we describe the main aspectes of the Adult dataset highlighted in the EDA task.

2.1

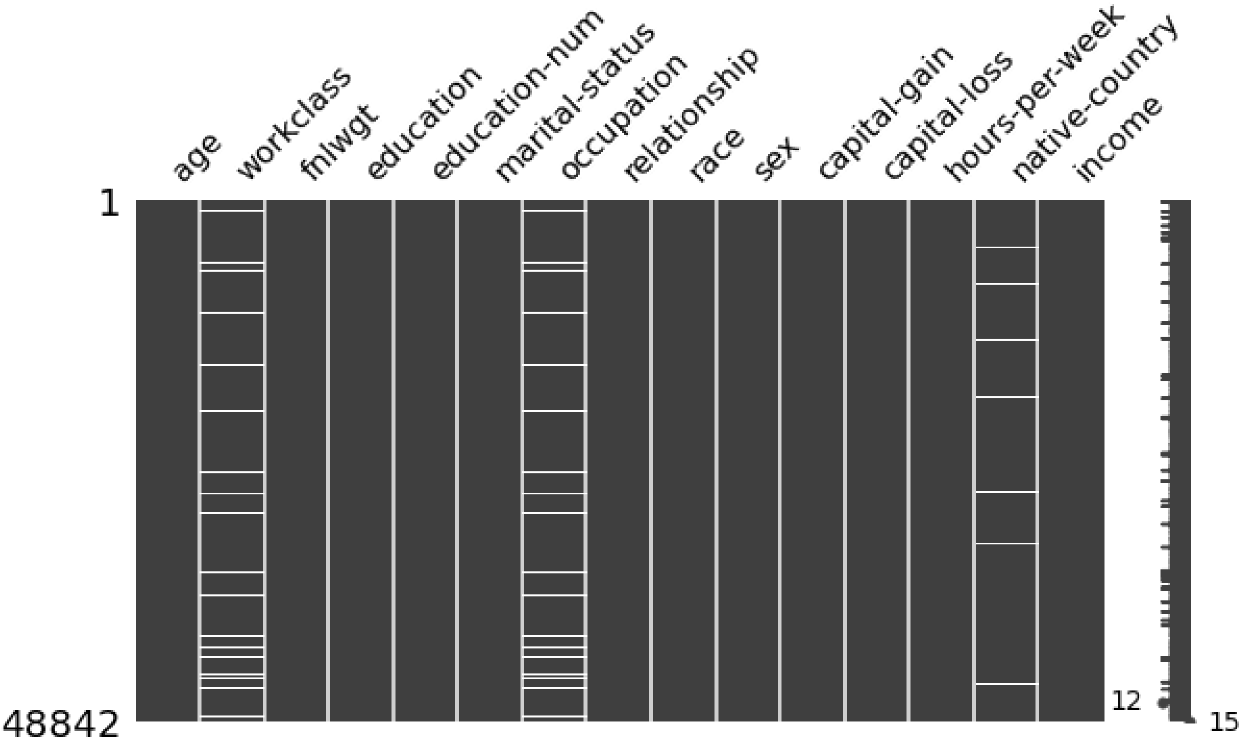
There is missing data present in the Adult dataset. The **missingno** library can be used to plot the missing data in the dataset, what shows how the missing data is distributed. The plot highlights how the missing data is sparse and present on 3 features only. In fact, the rows with missing data correspond to 7.41% of the original dataset (3620 samples out of 48842).

```
Original dataset: 48842
Rows with missing data: 3620
7.41% of missing data.
```

In [3]:

```
%matplotlib inline
import missingno as msn

msn.matrix(eda_data, figsize = (10,5))
```



Another aspect that can be pointed out is that only categorical features have missing data, therefore one approach to with it could be to set all of them as a new **UNKNOWN** category, in order to keep the data from the other features.

```
Columns with missing data:  ['workclass', 'occupation', 'native-country']
```

2.2

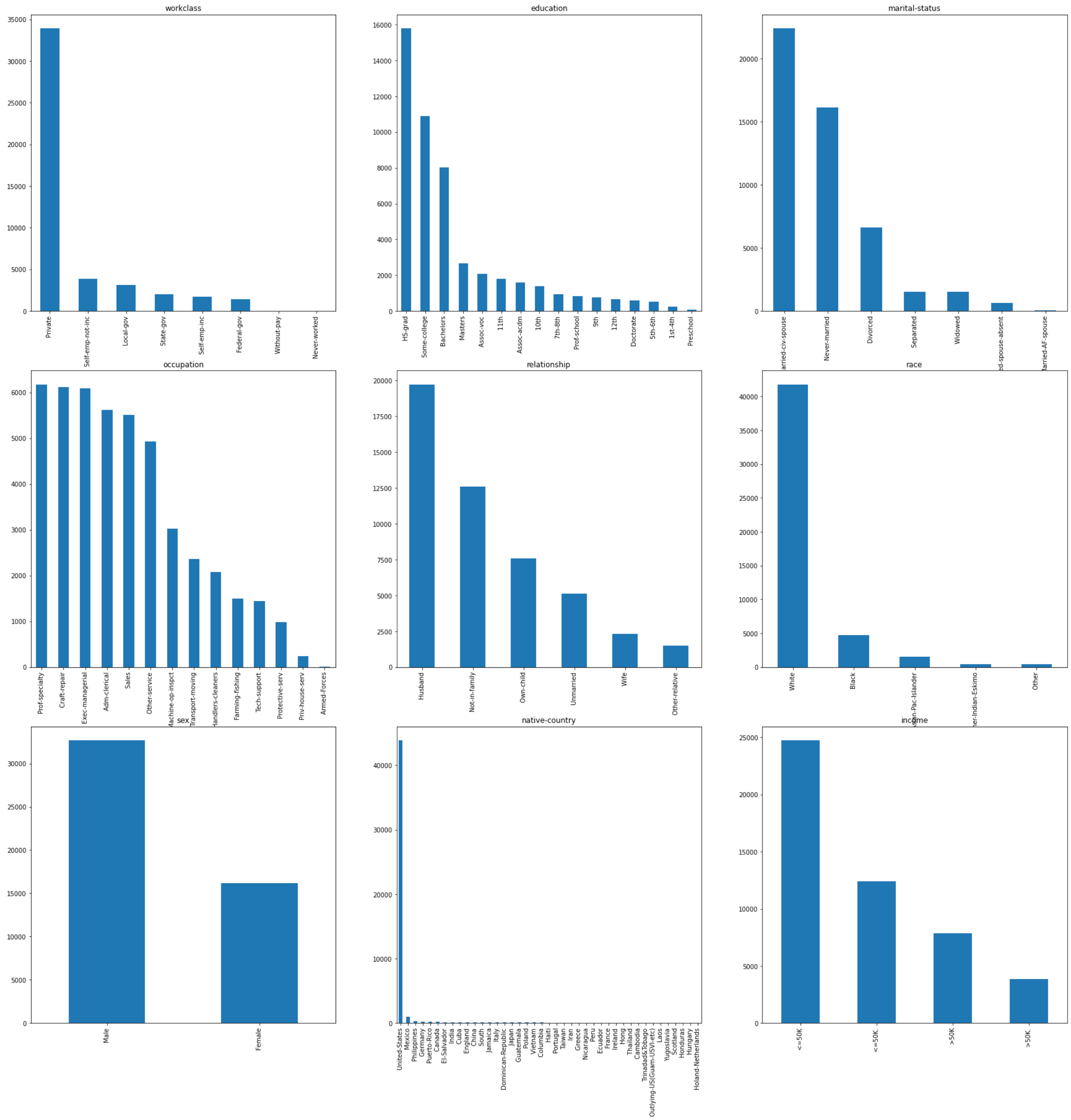
Categorical Features: the plots bellow shows the categorial features present in the Adult dataset. Some of the features are not well balanced, with very few examples for some classes, such as in the **native-country** feature.

In [5]:

```
%matplotlib inline
import matplotlib.pyplot as plt


# selecting categorical features
not_num = eda_data[['workclass','education','marital-status','occupation',
                      'relationship','race','sex','native-country','income']]

# ploting categorial features
ncols = 3
nrows = 3
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(10*nrows, 10*ncols))
for column, (i, j) in zip(list(not_num.columns), ((i,j) for i in range(nrows) fo
    not_num[column].value_counts().plot(kind="bar", ax=axes[i,j])
    axes[i,j].set_title(column)
```



It is also possible to see that there is more than one category (' >50K',' <=50K') in the **income** feature. That happens because in the test dataset the category string ends with a dot (' >50K.',' <=50K.'). Therefore it is necessary to normalize the income feature replacing the dot on the test dataset.

2.3

Numerical Features: to analyze the numerical data, we will map the **income** categorical feature to a new boolean feature called **50K_plus**, which is equal to 1 when income is higher than 50 thousand per year, else equals 0. The correlation matrix heatmap below show how the numerical features are correlated with the label **50K_plus**: the numerical features indeed shows some correlation with the label, but the **fnlwgt** label shows no correlation, indicating that this feature could be ignored.

In [7]:

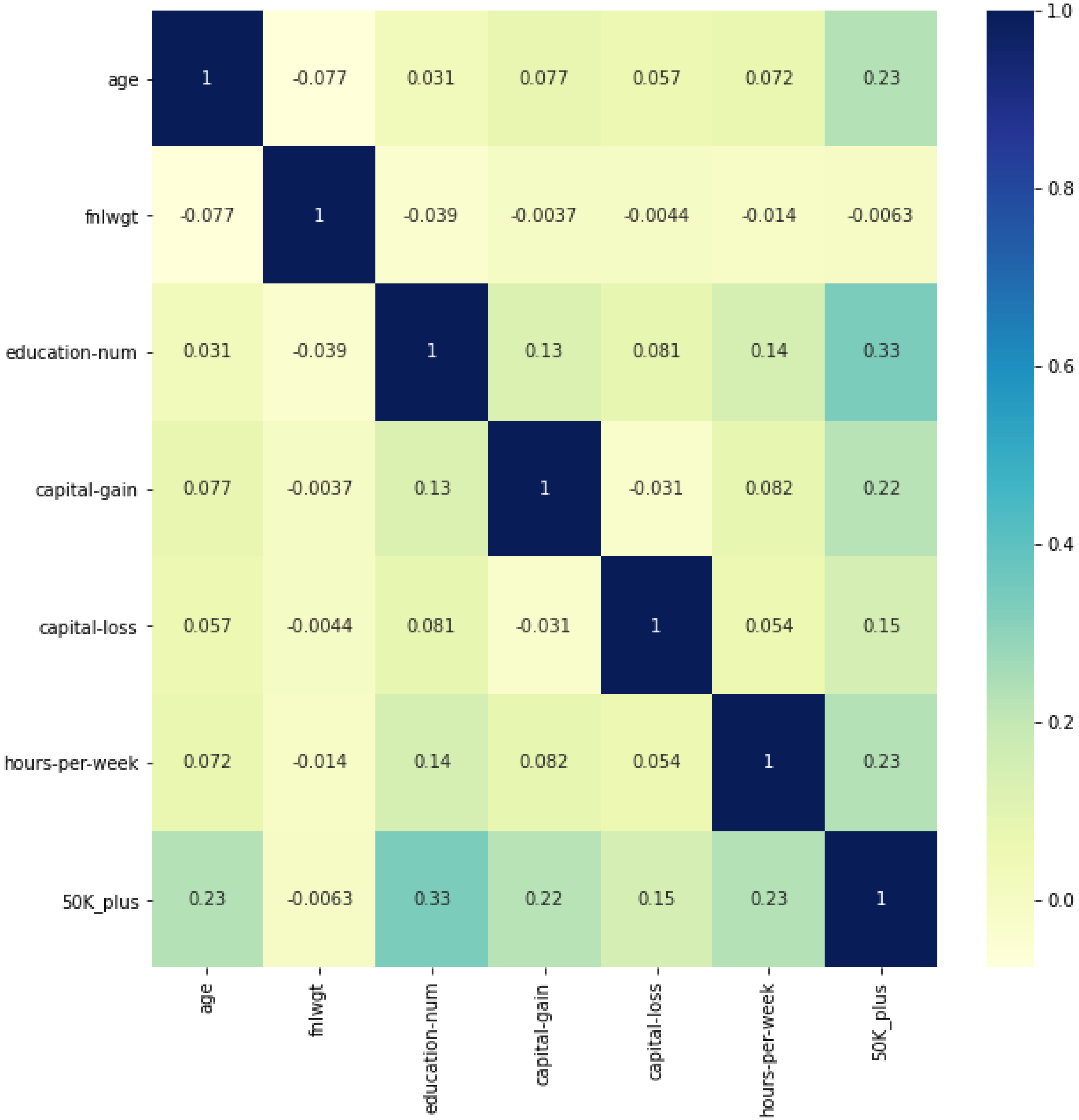
```
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt


# Normalizing the income feature (remove dots and spaces)
eda_data['50K_plus'] = eda_data['income'].str.replace('.','')
eda_data['50K_plus'] = eda_data['50K_plus'].str.replace(' ','')

# Select numerical features
num = eda_data[['age','fnlwgt','education-num','capital-gain','capital-loss','hc

# Mapping classes to 0 and 1
num = num.replace(to_replace=['<=50K', '>50K',], value=[0,1])

# Plot correlation matrix
corrMatrix = pd.DataFrame(np.corrcoef(num.values, rowvar=False), columns=num.col
fig, ax = plt.subplots(figsize=(10,10))
sn.heatmap(corrMatrix,ax=ax, annot=True, cmap="YlGnBu")
plt.yticks(rotation=0)
plt.show()
```

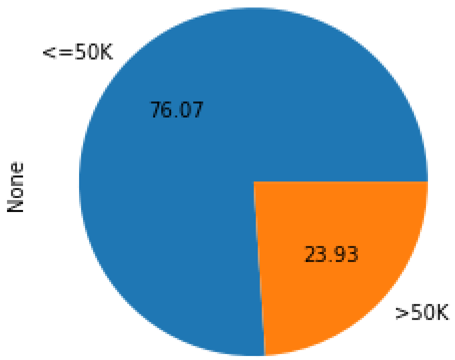


2.4

The estimated probability of the dataset for the income feature is:

- **'>50K'**: 23.93%
- **'<=50K'**: 76.07%

Hence the baseline accuracy for any model trained on this dataset is 76.07%, what could be acchieved by guessing all possible inputs as **<=50K**.



3. Data Engineering

In this section the raw Adult dataset will be processed in order to enalble supervised learning.

3.1. Handling missing data

As noted on section **2.1**, 7.41% of the dataset rows have missing data, which occurs only on three categorical features: **workclass**, **occupation**, **native-country**. Basically, we have two options:

1. Remove every row with missing data on at least one column, corresponding to less 7.41% of data in our dataset.
2. Treat the missing data as a new **UNKNOWN** class, in order to keep the whole dataset.

Here we will choose the option **2**. The main reason for this particular decision is to use the biggest dataset possible.

In [11]:

```
def handle_missing_data(df):
    return df.fillna('UNKNOWN')

no_missing_data = handle_missing_data(pd.concat([train_data, test_data]))
```

3.2. Cleaning the data

As observed with the EDA, the income feature need to be normalized and mapped to a numerical feature (0 or 1). Here we will create a new numerical feature named **50K_plus**.

In [12]:

```
def clean_data(df):

    # Normalizing the income feature (remove dots and spaces)
    df['50K_plus'] = df['income'].str.replace('.','')
    df['50K_plus'] = df['50K_plus'].str.replace(' ','')

    # Mapping classes to 0 and 1
    df['50K_plus'] = df['50K_plus'].replace(to_replace=['<=50K', '>50K',], value

    # Select the interest features to form our clean dataset
    return df[['age', 'workclass', 'education', 'education-num',
        'marital-status', 'occupation', 'relationship', 'race', 'sex',
        'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', '50K_

cleaned_data = clean_data(no_missing_data)
```

3.3. Dealing with categorical data

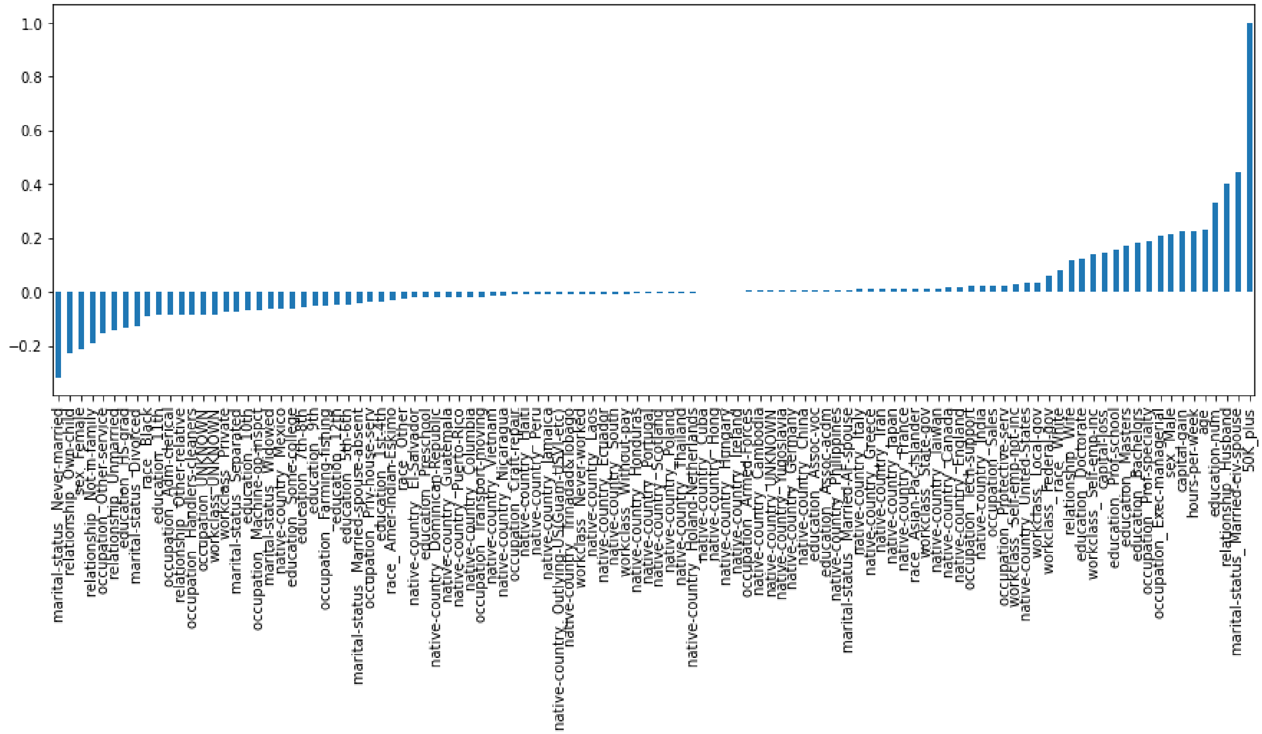
There are many ways to deal with the categorical data. The most canonical and simple method is the **one-hot-encoding**, which is equivalent to create a new column for each class of every categorical feature and map them to zeros and ones. This method leads to a very wide dataset, with many features that might not be meaningful for our purpose of prediciting the **50K_pluss** class.

The **one-hot-encoding** will be done with the **pandas.get_dummies()** function from the Pandas library. The bar plot below shows the correlation value for each one-hot-encoded feature.

In [40]:

```
one_hot_encoded = pd.get_dummies(cleaned_data)
feature_corr = one_hot_encoded.corr().loc[:,'50K_plus'].sort_values(ascending=Tr

feature_corr.plot(kind='bar', figsize=(15,5))
```



3.4. Feature Engineering

To avoid unecessary features, we will select only the dummie features above the median when sorted by the absolute value of its correlation with the target variable. We will also remove the dummie features that have the **UNKNOWN** class, as they would not have practical meaning. At the end, we keep 51 features out of a total of 107.

In [14]:

```
def feature_engineering(df):

    # One hot encode data
    one_hot_encoded = pd.get_dummies(df)
    feature_corr = one_hot_encoded.corr().loc[:,'50K_plus'].sort_values(ascendir

    # Select features with correlation above the median
    features = feature_corr[feature_corr.abs() >= np.quantile(feature_corr.abs()
    feature_list = list(features.index)

    # Remove UNKNOWN classes out of the selected features
    feature_list = [ x for x in feature_list if "UNKNOWN" not in x ]


    return one_hot_encoded[feature_list]

master_dataset = feature_engineering(cleaned_data)
```

3.5. Data Split

Finally, after preparing the raw data, we can split the dataset into train and test sets, in which a model for predicting if someone's yearly income is higher than 50 thousand dollars will be trained using supervised learning. We will use 1/3 of the dataset for test and 2/3 for training.

In [15]:

```
from sklearn.model_selection import train_test_split

def split_data(master_dataset):

    X = master_dataset.drop('50K_plus', axis=1)
    y = master_dataset['50K_plus']

    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.33,
        random_state=5024,
        shuffle=True
    )

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = split_data(master_dataset)
```

4. Model Fitting

4.1. KNN Classifier

In this section we will train a KNN Classifier. The hyperparameters will be chosen using cross validation on the training set with 5 folds and Grid Search. We will evaluate the number of neighbors and the value of p (Power parameter for the Minkowski metric).

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

# Defining hyperparamters space
k_range = list(range(1,35))
```

```
weights = ['uniform', 'distance']
p_range = list(range(1,3))
param = dict(n_neighbors=k_range, p=p_range)

knn = KNeighborsClassifier()

# Run Grid Search with cross validation
grid = GridSearchCV(knn, param, cv=5, scoring='accuracy', n_jobs = -2)
grid.fit(X_train, y_train)
```

Evaluating our best model on the test dataset (holdout data) and printing the final metrics.

In [31]:

```
from sklearn.metrics import confusion_matrix

def print_metrics(grid,X_test,y_test, model_name):

    y_hat = grid.best_estimator_.predict(X_test)

    tn, fp, fn, tp = confusion_matrix(y_test, y_hat).ravel()

    print(15*'-')
    print('Best {} Classifier: '.format(model_name), grid.best_estimator_)
    print('Accuracy on Training Set: {:.2f}%'.format(100*grid.best_score_))
    print(15*'-')
    print("Test Metrics".format(model_name))
    print('Accuracy: {:.2f}'.format((tn+tp)*100/(tp+tn+fp+fn)))
    print('Precision: {:.2f}'.format((tp)*100/(tp+fp)))
    print('Recall: {:.2f}'.format((tp)*100/(tp+fn)))

print_metrics(grid, X_test, y_test, 'KNN')
```

```
---------------
Best KNN Classifier:  KNeighborsClassifier(n_neighbors=24, p=1)
Accuracy on Training Set: 85.92%
---------------
Test Metrics
Accuracy: 86.28
Precision: 76.45
Recall: 60.26
```

4.2. Random Forest Classifier

In this section we will train a Random Forest Classifier. The hyperparameters will be chosen using cross validation on the training set with 5 folds and Grid Search. We will evaluate different values for the number of estimators and the criterion (entropy or gini) used by the trees.

In [35]:

```
from sklearn.ensemble import RandomForestClassifier

# Defining hyperparamters space
n_range = list(range(100,400,50))
c_list = ['gini', 'entropy']
param = dict(n_estimators=n_range, criterion=c_list)

RF = RandomForestClassifier()

# Run Grid Search with cross validation
grid = GridSearchCV(RF, param, cv=5, scoring='accuracy', n_jobs = -2)
grid.fit(X_train, y_train)
```

Out[35]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-2,
             param_grid={'criterion': ['gini', 'entropy'],
                         'n_estimators': [100, 150, 200, 250, 300, 350]},
             scoring='accuracy')
```

Evaluating our best model on the test dataset (holdout data) and printing the final metrics.

In [37]:
```
print_metrics(grid, X_test, y_test, 'Random Forest')
```

```
---------------
Best Random Forest Classifier:  RandomForestClassifier(criterion='entropy', n_es
timators=200)
Accuracy on Training Set: 84.68%
---------------
Test Metrics
Accuracy: 85.02
Precision: 71.04
Recall: 61.34
```

4.3. MLP Classifier

In this section we will train a Random Forest Classifier. The hyperparameters will be chosen using cross validation on the training set with 5 folds and Grid Search.

In [38]:
```
from sklearn.neural_network import MLPClassifier

# Defining hyperparamters space
h_layers_range = [(500,200,100),(50,20,10),(250,100,50)]
act_list = ['logistic', 'relu']
param = dict(hidden_layer_sizes=h_layers_range, activation=act_list)

RF = MLPClassifier()

# Run Grid Search with cross validation
grid = GridSearchCV(RF, param, cv=5, scoring='accuracy', n_jobs = -2)
grid.fit(X_train, y_train)
```

Out[38]:
```
GridSearchCV(cv=5, estimator=MLPClassifier(), n_jobs=-2,
             param_grid={'activation': ['logistic', 'relu'],
                         'hidden_layer_sizes': [(500, 200, 100), (50, 20, 10),
                                                (250, 100, 50)]},
             scoring='accuracy')
```

Evaluating our best model on the test dataset (holdout data) and printing the final metrics.

In [39]:
```
print_metrics(grid, X_test, y_test, 'MLP')
```

```
---------------
Best MLP Classifier:  MLPClassifier(activation='logistic', hidden_layer_sizes=(5
00, 200, 100))
Accuracy on Training Set: 84.81%
---------------
Test Metrics
Accuracy: 83.66
Precision: 67.66
Recall: 58.54
```

5. Conclusions

The final results shows that the KNN classifier is the best model for classification in the Adult dataset considering the Data Engineering steps applied in this notebook. In this case, the simple of all models did the best results. That could be explained by the size of the dataset, which is not big enough to train a complex model such as an MLP, for instance. Nonetheless, the results for both of the three models employed here are very similar.

Model Type	Accuracy	Precision	Recall
KNN	**86.28%**	**76.45%**	60.26%
Random Forest	85.02%	71.04%	**61.34%**
MLP	83.66%	67.66%	58.54