# FOUNDATIONS OF MODERN MACHINE LEARNING, IIIT Hyderabad

MODULE 2: Feature Normalization, Nearest Neighbor Revisited

Project: Binary Classification of Adults

Module Coordinator: Tanvi Kamble

---

This projects requires you to apply the machine learning cocepts that you learnt so far to fill in the #TODO parts so that we can classify which income group an adult lies in.

An adult's income can be determined by a lot of factors like the individual’s education level, age, gender, occupation, and etc. We use a dataset prsent on Kaggle provided by [UCI] to perform KNN and find the income group.

First let's open the dataset stored as a CSV file using pandas dataframe, stored in google drive.

```
from google.colab import files
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving adult.csv to adult (4).csv
{'adult.csv': b'age,workclass,fnlwgt,education,educational-num,marital-status,occupation,relationship,race,gender,capital-gain,
```

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

```
adult = pd.read_csv('/content/adult.csv')
```

```
# Looking at the dataset
adult.head(10)
```

| | age | workclass | fnlwgt | education | educational-num | marital-status | occupation | relationship | race | gender | capital-gain |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | Private | 226802 | 11th | 7 | Never-married | Machine-op-inspct | Own-child | Black | Male | 0 |
| **1** | 38 | Private | 89814 | HS-grad | 9 | Married-civ-spouse | Farming-fishing | Husband | White | Male | 0 |
| **2** | 28 | Local-gov | 336951 | Assoc-acdm | 12 | Married-civ-spouse | Protective-serv | Husband | White | Male | 0 |
| **3** | 44 | Private | 160323 | Some-college | 10 | Married-civ-spouse | Machine-op-inspct | Husband | Black | Male | 7688 |
| **4** | 18 | ? | 103497 | Some-college | 10 | Never-married | ? | Own-child | White | Female | 0 |
| **5** | 34 | Private | 198693 | 10th | 6 | Never-married | Other-service | Not-in-family | White | Male | 0 |
| **6** | 29 | ? | 227026 | HS-grad | 9 | Never-married | ? | Unmarried | Black | Male | 0 |
| **7** | 63 | Self-emp-not-inc | 104626 | Prof-school | 15 | Married-civ-spouse | Prof-specialty | Husband | White | Male | 3103 |
| [illegible] | [illegible] | [illegible] | [illegible] | Some- | [illegible] | Never- | Other- | [illegible] | [illegible] | [illegible] | [illegible] |

```
# Removing duplicate entries
adult=adult.drop_duplicates()
```

```
# Let's get to know the dataset
adult.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48790 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              48790 non-null  int64 
 1   workclass        48790 non-null  object
 2   fnlwgt           48790 non-null  int64 
 3   education        48790 non-null  object
 4   educational-num  48790 non-null  int64 
 5   marital-status   48790 non-null  object
 6   occupation       48790 non-null  object
 7   relationship     48790 non-null  object
 8   race             48790 non-null  object
 9   gender           48790 non-null  object
 10  capital-gain     48790 non-null  int64 
 11  capital-loss     48790 non-null  int64 
 12  hours-per-week   48790 non-null  int64 
 13  native-country   48790 non-null  object
 14  income           48790 non-null  object
dtypes: int64(6), object(9)
memory usage: 6.0+ MB
```

```
adult.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **age** | 48790.0 | 38.652798 | 13.708493 | 17.0 | 28.0 | 37.0 | 48.00 | 90.0 |
| **fnlwgt** | 48790.0 | 189668.999365 | 105617.231232 | 12285.0 | 117555.0 | 178138.5 | 237606.25 | 1490400.0 |
| **capital-gain** | 48790.0 | 1080.217688 | 7455.905921 | 0.0 | 0.0 | 0.0 | 0.00 | 99999.0 |

```
# Adding Index Column so that each entry is identified independently
adult['Index'] = range(1, len(adult) + 1)
```

```
adult = adult.set_index('Index')
```

This dataset has '?' in place of all Null entries. Let's find the total null entries.

```
adult.isin(['?']).sum()
```

```
age                   0
workclass          2795
fnlwgt                0
education             0
educational-num       0
marital-status        0
occupation         2805
relationship          0
race                  0
gender                0
capital-gain          0
capital-loss          0
hours-per-week        0
native-country      856
income                0
dtype: int64
```

```
df = adult.copy()
```

```
df['income'] = df['income'].replace('nan', np.nan)
df = df[df['income'].isin([np.nan]) == False]
```

```
#  Three classes called Workclass, Occupation and Native-Country have null values so we first replace it with np.nan.
df['workclass']=df['workclass'].replace('?',np.nan)
df['occupation']=df['occupation'].replace('?',np.nan)
df['native-country']=df['native-country'].replace('?',np.nan)
```

These three features are categorical in nature so performing Imputation based KNN will be the best option to find out the missing features.

```
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
df.education=lb.fit_transform(df.education)
df['marital-status']=lb.fit_transform(df['marital-status'])
df.relationship=lb.fit_transform(df.relationship)
df.race=lb.fit_transform(df.race)
df.gender=lb.fit_transform(df.gender)
df.income=lb.fit_transform(df.income)
```

```
df.isin([np.nan]).sum()
```

```
age                   0
workclass          2795
fnlwgt                0
education             0
educational-num       0
marital-status        0
occupation         2805
relationship          0
race                  0
gender                0
capital-gain          0
capital-loss          0
hours-per-week        0
native-country      856
income                0
dtype: int64
```

```
# For the NULL values of capital loss and hours per week feature perfrom imputation by mean.
df['capital-loss'] = lb.fit_transform(df['capital-loss'])
df['hours-per-week'] = lb.fit_transform(df['hours-per-week'])


# IMPUTATION USING K-NN
# Workclass
x_train_workclass = df.loc[df['workclass'].isin([np.nan]) == False].drop(['workclass', 'occupation', 'native-country'], axis = 1)
y_train_workclass = df.loc[df['workclass'].isin([np.nan]) == False].workclass
y_train_workclass = lb.fit_transform(y_train_workclass)
for itr, ind in enumerate(x_train_workclass.index):
  df['workclass'][ind] = y_train_workclass[itr]
x_test_workclass = df.loc[df['workclass'].isin([np.nan])].drop(['workclass', 'occupation', 'native-country'], axis = 1)
# Occupation
x_train_occupation = df.loc[df['occupation'].isin([np.nan]) == False].drop(['workclass', 'occupation', 'native-country'], axis = 1)
y_train_occupation = df.loc[df['occupation'].isin([np.nan]) == False].occupation
y_train_occupation = lb.fit_transform(y_train_occupation)
for itr, ind in enumerate(x_train_occupation.index):
  df['occupation'][ind] = y_train_occupation[itr]
x_test_occupation = df.loc[df['occupation'].isin([np.nan])].drop(['workclass', 'occupation', 'native-country'], axis = 1)
# Native Country
x_train_country = df.loc[df['native-country'].isin([np.nan]) == False].drop(['workclass', 'occupation', 'native-country'], axis = 1)
y_train_country = df.loc[df['native-country'].isin([np.nan]) == False]['native-country']
y_train_country = lb.fit_transform(y_train_country)
for itr, ind in enumerate(x_train_country.index):
  df['native-country'][ind] = y_train_country[itr]
x_test_country = df.loc[df['native-country'].isin([np.nan])].drop(['workclass', 'occupation', 'native-country'], axis = 1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
```

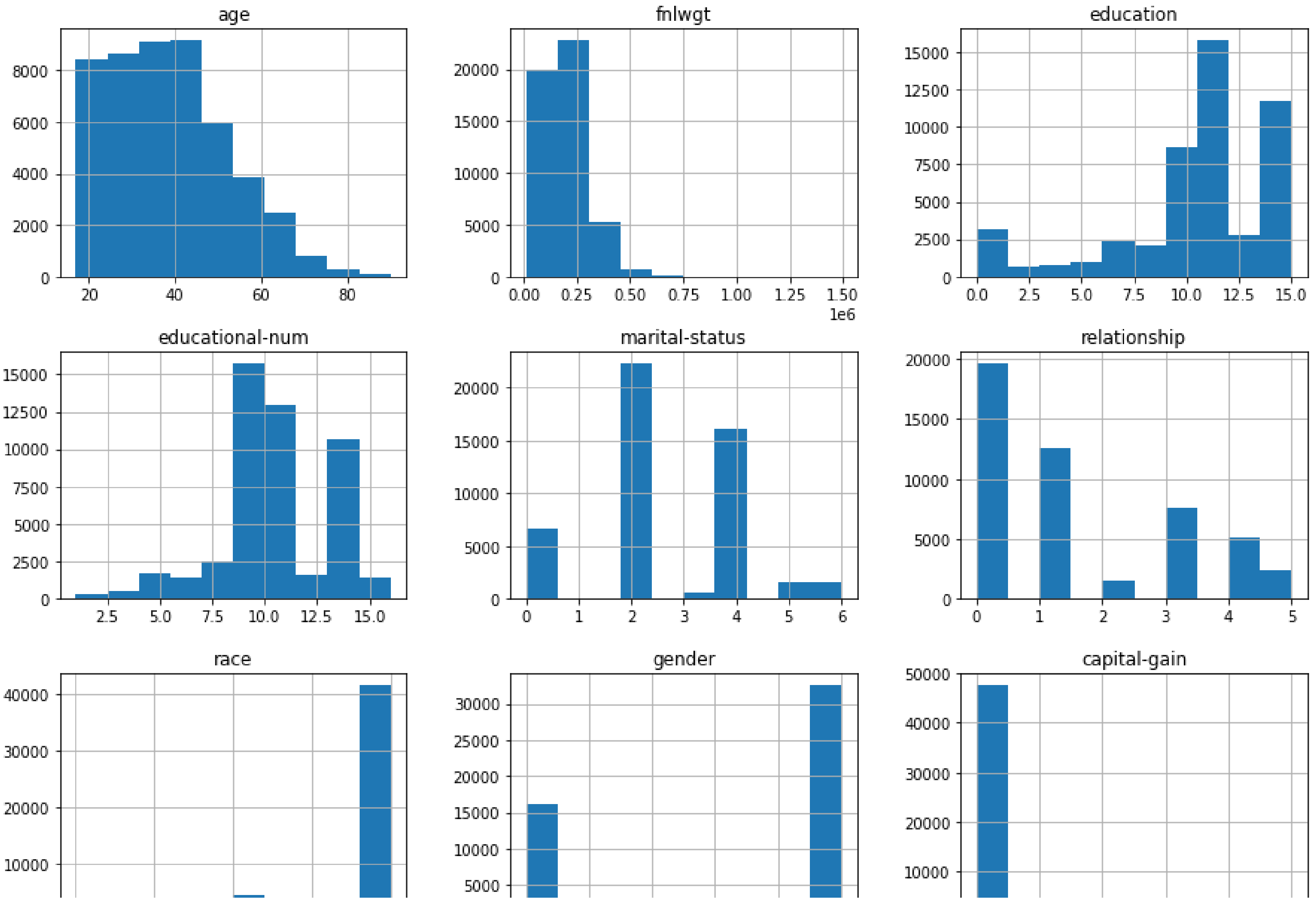
```
from sklearn.neighbors import KNeighborsClassifier


# calculating predictions for all the features
#  Use the KNeighborsClassifier with neighbours = 7 and all the other entries as default to find the missing values.
 x_train_workclass, x_test_workclass, y_train_workplace, y_test = train_test_split(x, y, test_size=0.33, random_state=17)
 sklearn_knn = KNeighborsClassifier(n_neighbors=7)
 sklearn_knn.fit(x_train_workclass,y_train_workclass)
 workplace_pred = sklearn_knn.predict(x_test_workclass)

 x_train_occupation, x_test_occupation, y_train_occupation, y_test = train_test_split(x, y, test_size=0.33, random_state=17)
 sklearn_knn.fit(x_train_occupation,y_train_occupation)
 occupation_pred = sklearn_knn.predict(x_test_occupation)

 x_train_country, x_test_country, y_train_country, y_test = train_test_split(x, y, test_size=0.33, random_state=17)
 sklearn_knn.fit(x_train_country,y_train_country)
 country_pred = sklearn_knn.predict(x_test_country)



# Replacing the predicted values in the original dataframe
for itr, ind in enumerate(x_test_workclass.index):
  df['workclass'][ind] = workplace_pred[itr]

for itr, ind in enumerate(x_test_occupation.index):
  df['occupation'][ind] = occupation_pred[itr]

for itr, ind in enumerate(x_test_country.index):
  df['native-country'][ind] = country_pred[itr]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  if __name__ == '__main__':
```

```
df['workclass'] = df['workclass'].astype(str).astype(str)
df['occupation'] = df['occupation'].astype(str).astype(str)
df['native-country'] = df['native-country'].astype(str).astype(str)
```

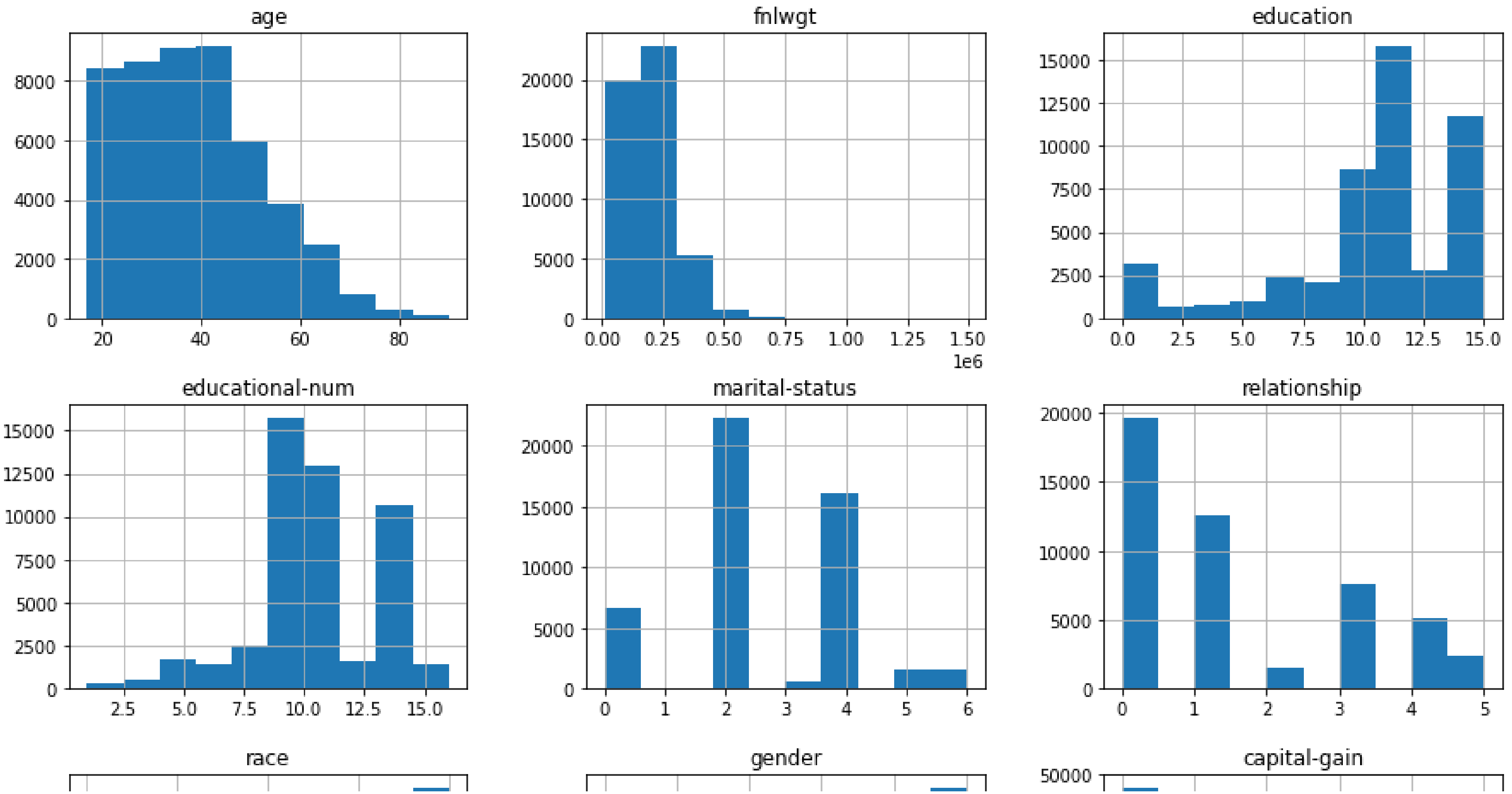
```
p = df.hist(figsize = (15,15))
```



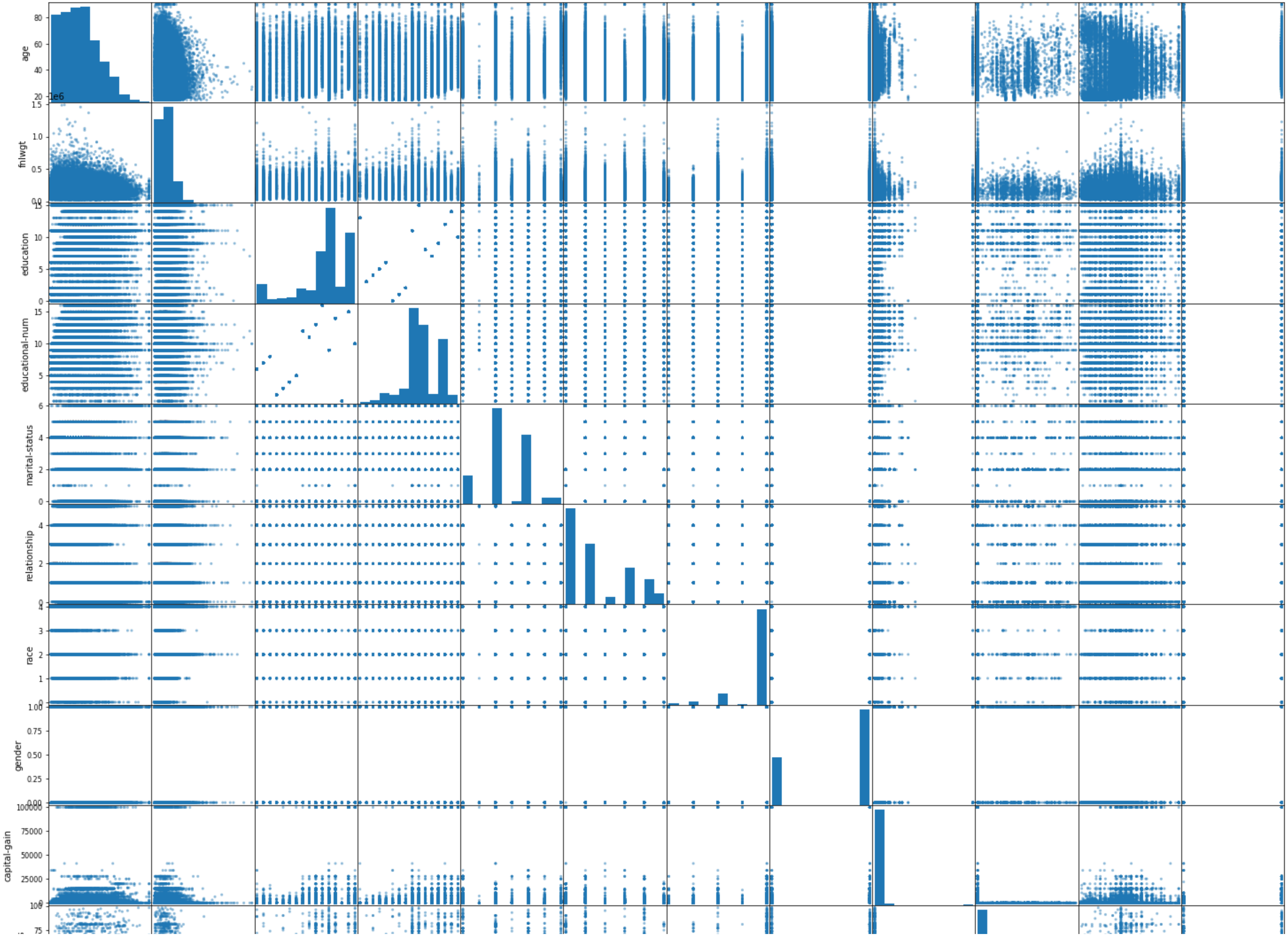

After plotting the figures we can see that there is some scope for clipping fnlwgt, capital loss and capital gain with vmax.

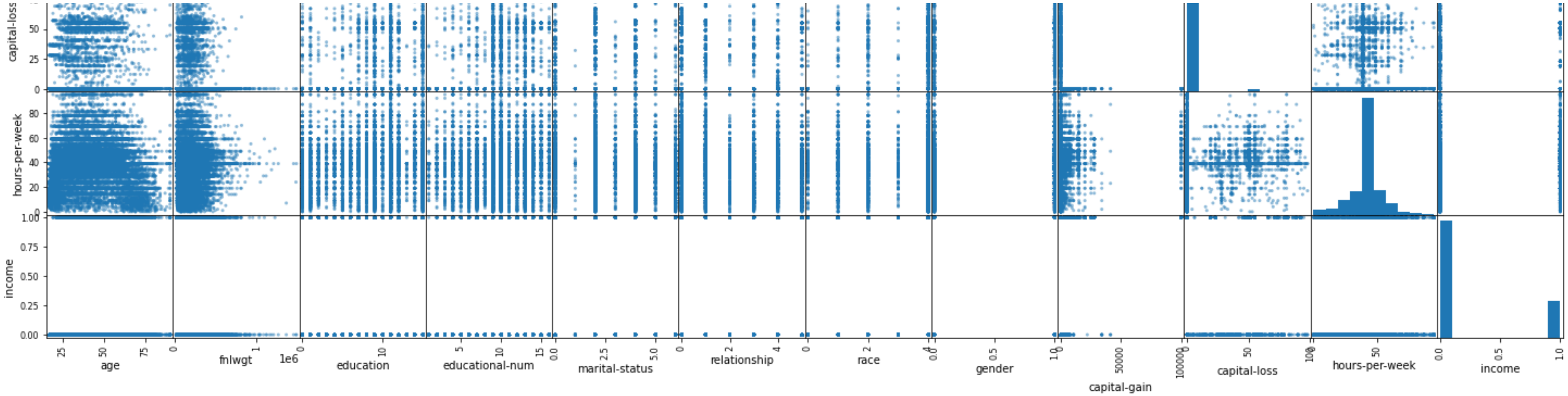
```
'''
TODO
Choose an appropriate maximum value to clip Capital Loss and Capital Gain values to and clip them accordingly
```

```
'''
df_standard = df.copy()
vmax_cap_gain = 0
vmax_cap_loss = 0
vmax_fnlwgt = 0
df_standard['capital-loss'] = df_standard['capital-loss']
df_standard['capital-gain'] = df_standard['capital-gain']
df_standard['fnlwgt'] = df_standard['fnlwgt']


df_standard = df_standard.dropna(how = 'all')


# Let's plot the same graph for standardized data
p = df_standard.hist(figsize = (15,15))
```




```
# Let's plot the various features now and see if we can find any useless features not required for KNN
from pandas.plotting import scatter_matrix
p = scatter_matrix(df,figsize=(25, 25))
```


age
fnlwgt
1e6
education
educational-num
marital-status
relationship
race
gender
capital-gain




```
# Let's analyse the same using correlation map.
df.corr()
```

| | age | fnlwgt | education | educational-num | marital-status | relationship | race | gender | capital-gain | capital-loss | hc per- |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **age** | 1.000000 | -0.076451 | -0.015142 | 0.030635 | -0.263594 | -0.263395 | 0.028803 | 0.088043 | 0.077185 | 0.062129 | 0.07 |
| **fnlwgt** | -0.076451 | 1.000000 | -0.022539 | -0.038727 | 0.029779 | 0.009017 | -0.027165 | 0.027879 | -0.003715 | -0.005698 | -0.01 |
| **education** | -0.015142 | -0.022539 | 1.000000 | 0.359825 | -0.037449 | -0.010861 | 0.013387 | -0.027120 | 0.028958 | 0.016976 | 0.05 |
| **educational-num** | 0.030635 | -0.038727 | 0.359825 | 1.000000 | -0.069859 | -0.090697 | 0.029331 | 0.009364 | 0.125219 | 0.084203 | 0.14 |
| **gender** | 0.088043 | 0.027879 | -0.027120 | 0.009364 | -0.127505 | -0.579955 | 0.086959 | 1.000000 | 0.047127 | 0.049019 | 0.22 |

## Observations

For the income column it is clear that no column directly affects the Income. We can safely assume that there no feature will completely overpower and determine the outcome. Hence, no need for regularization.

```
# Checking if the data is biased
print(df['income'].value_counts())
plt.bar([0,1],df['income'].value_counts())
```

```
0    37109
1    11681
```

## Observations

There is a clear bias in the data and our model might end up giving income as 0 for most cases


```
# Next we want to Z-Scale the data
from sklearn.preprocessing import StandardScaler
```

```
# TODO
# Write the code for z-transforming the data using StandardScalar class of sklearn which we have imported in the last cell.
scaled_data = None
```

```
df_scaled = pd.DataFrame(scaled_data,columns=df_standard.columns[:-1])
df_scaled.head()
```

| | age | workclass | fnlwgt | education | educational-num | marital-status | occupation | relationship | race | gender | capital-gain | c |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.997775 | -0.096959 | 0.359442 | -2.385847 | -1.198164 | 0.905590 | 0.023073 | 0.976802 | -1.980601 | 0.704627 | -0.295996 | -( |
| **1** | -0.056145 | -0.096959 | -0.955908 | 0.188776 | -0.418706 | -0.417454 | -0.473019 | -0.902069 | 0.391828 | 0.704627 | -0.295996 | -( |
| **2** | -0.780475 | -1.885749 | 1.417085 | -0.841073 | 0.750481 | -0.417454 | 1.015255 | -0.902069 | 0.391828 | 0.704627 | -0.295996 | -( |
| **3** | 0.378454 | -0.096959 | -0.278885 | 1.218625 | -0.028977 | -0.417454 | 0.023073 | -0.902069 | -1.980601 | 0.704627 | 3.428012 | -( |
| **4** | -1.504806 | -0.096959 | -0.824525 | 1.218625 | -0.028977 | 0.905590 | -0.224973 | 0.976802 | 0.391828 | -1.419191 | -0.295996 | -( |

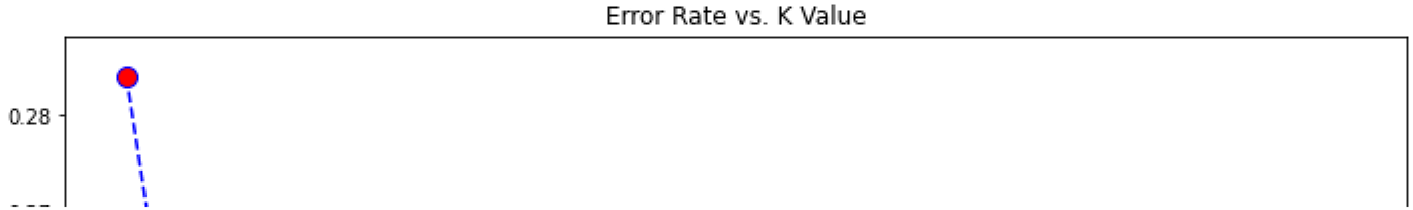
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
```

```
# Let's create a K-NN and compare the performances of scaled vs unscaled data.
# We first create a function for performing KNN
```

```
##########################
##########################

## TODO : Complete the lines of code wherever marked as [REQUIRED] in this cell.

##########################
##########################


def plot_KNN_error_rate(xdata,ydata):
  error_rate = []
  test_scores = []
  train_scores = []

  ## [REQUIRED] Split the data into train and test sets in a 70:30 ratio (70% train, 30% test)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, train_size=0.70 random_state = 17)


  for i in range(1,15):
      ## [REQUIRED] Complete the code in the next three lines
      knn = sklearn_knn = KNeighborsClassifier(n_neighbors=i)
      sklearn_knn.fit(X_train,y_train)
      pred_i = sklearn_knn.predict(X_test)

      error_rate.append(np.mean(pred_i != y_test))
      train_scores.append(knn.score(X_train,y_train))
      test_scores.append(knn.score(X_test,y_test))

  plt.figure(figsize=(12,8))
  plt.plot(range(1,15),error_rate,color='blue', linestyle='dashed', marker='o',
          markerfacecolor='red', markersize=10)
  plt.title('Error Rate vs. K Value')
  plt.xlabel('K')
  plt.ylabel('Error Rate')
  print()
  ## score that comes from testing on the same datapoints that were used for training
  max_train_score = max(train_scores)
```

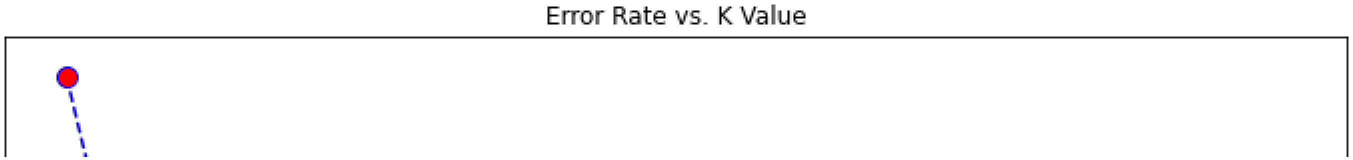
```
  train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score]
  print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambda x: x+1, train_scores_ind))))
  print()
  ## score that comes from testing on the datapoints that were split in the beginning to be used for testing solely
  max_test_score = max(test_scores)
  test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
  print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))

  return test_scores



# Unchanged dataset
orig_X = df.drop('income', axis = 1)
orig_y = df.income
unchanged_test_scores = plot_KNN_error_rate(orig_X, orig_y)
```

```
Max train score 99.99257223501449 % and k = [1]

Max test score 79.77469670710572 % and k = [14]
```



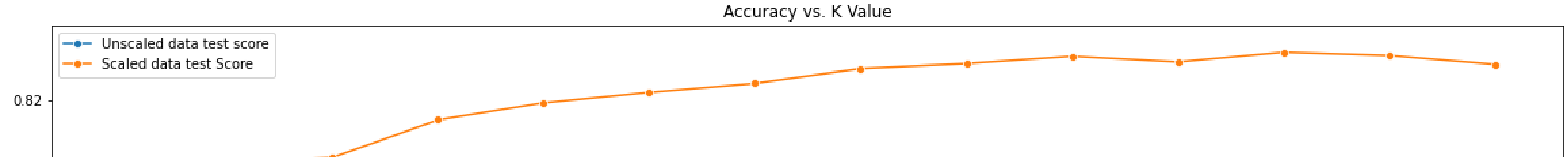

```
# Standardized Dataset
scaled_X = df_scaled
scaled_y = df_standard.income
scaled_test_scores = plot_KNN_error_rate(scaled_X, scaled_y)
```

```
Max train score 99.99257223501449 % and k = [1]

Max test score 83.0155979202773 % and k = [12]
```




```
# Comparing the two accuracies
import seaborn as sns
plt.figure(figsize=(20,8))
plt.title('Accuracy vs. K Value')
sns.lineplot(range(1,15),unchanged_test_scores,marker='o',label='Unscaled data test score')
sns.lineplot(range(1,15),scaled_test_scores,marker='o',label='Scaled data test Score')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args:
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args:
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f86df302d10>
```




```
# TODO
# Use Weighted KNN and compare the results of both the datasets
```

```
# TODO
# Refer to MinMax Scaler provided in scikit-learn.
## Use MinMax scaling on the dataset, and see the performance of KNN on this minmax-scaled dataset.
```

```
## TASK-8: Use K-Fold cross validation on all the above classification experiments and present an analysis of the results you obtain.
```

# Conclusion

We carried out data analysis which helped us realise the missing values and helped us check if there is any visible bias in the data.

As for the classification tasks, the standardized data yields much better results than the unscaled data over most of the K-values considered, thus indicating the importance of standardizing data in Machine Learning problems.

## References

https://www.kaggle.com/wenruliu/adult-income-dataset