Statistics and Data Science

Mini Project : Income censes

Contributions

- Exploratory data analysis – Y K Ujwal (PES1UG19CS585)
- Graph visualization – Yuvaraj (PES1UG19CS597)
- Normalization and Standardization – Mehul Bhandari(PES2UG19CS909)
- Hypothesis Testing – Pradeep Reddy (PES1UG19CS564)
- Correlation - Yuvaraj (PES1UG19CS597)

Exploratory Data Analysis

- First import the dataset and required libraries.

Data cleaning

In [1]:
```
# Importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:
```
missing_values = ['nan', '?', ' ?']
df = pd.read_csv('dataset.csv', na_values = missing_values)
```

In [3]:
```
df.head()
```

Out[3]:

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

- Add coloumn headers and check for null values

In [4]:
```
# Add headers to the coloumns
column_head = ["Age","WorkClass","fnlwgt","Education","Education Number","marital-status","Ocuupation",
               "RelationShip","race","Gender","Capital-gain","Capital-loss","Hours-per-Week","nativeCountry","Salary"]
df.columns = column_head
```

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

Out[5]:

	Age	WorkClass	fnlwgt	Education	Education Number	marital-status	Ocuupation	RelationShip	race	Gender	Capital-gain	Capital-loss	Hours-per-Week	nativeCountry	Salary
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

Make 2 lists containing different column headers to distinguish between columns that contain integers, and columns that contain strings

In [6]:
```
# List of columns headers that contain integers
integer_cols = ['Age', 'fnlwgt', 'Education Number', 'Capital-gain', 'Capital-loss', 'Hours-per-Week']
non_integer_cols = []

# List of coloumn heads that contain strings
for col in df:
    # If a coloumn in dataset is not in the list of columns that contain integers
    # Add it to list of columns that contain strings
    if col not in integer_cols:
        non_integer_cols.append(col)
```

In [7]:
```
integer_cols
```

Out[7]:
```
['Age',
 'fnlwgt',
 'Education Number',
 'Capital-gain',
 'Capital-loss',
 'Hours-per-Week']
```

In [8]:
```
non_integer_cols
```

Out[8]:
```
['WorkClass',
 'Education',
 'marital-status',
 'Ocuupation',
 'RelationShip',
 'race',
 'Gender',
 'nativeCountry',
 'Salary']
```

Strip white space for all rows

In [9]:
```
# Strip white space for all members
for col in non_integer_cols:
    c = 0
    df[col] = df[col].str.strip()
```

Find the null values

Find the number of null values in the data set

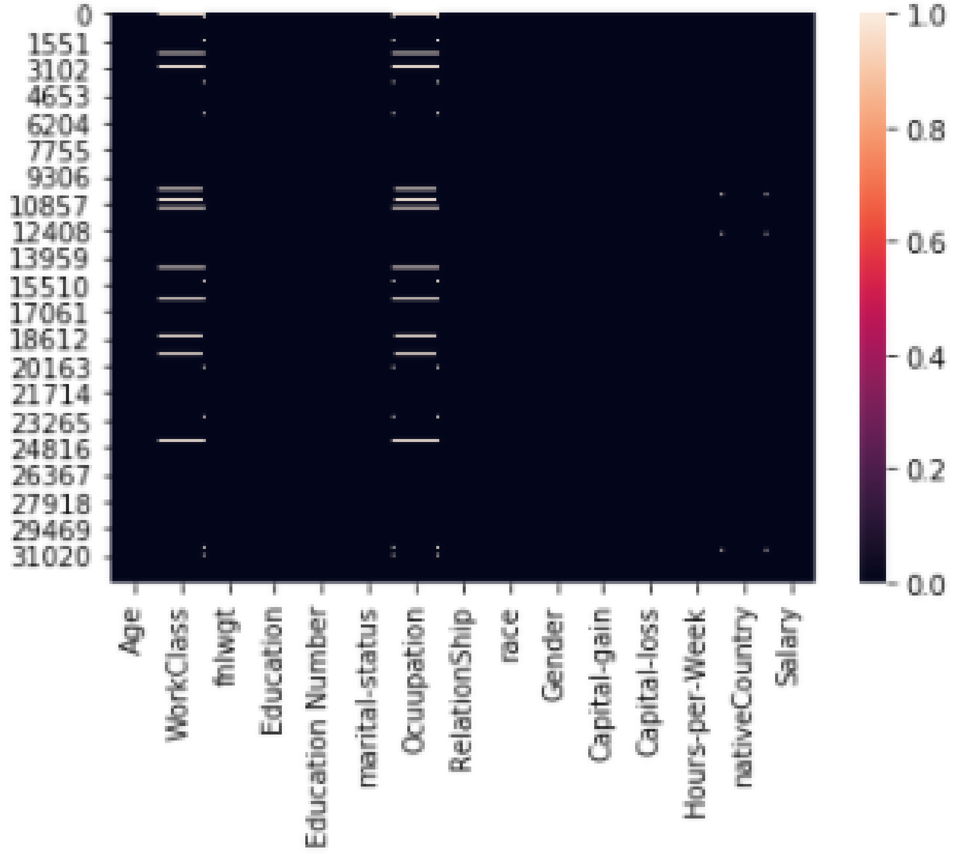
In [10]:
```
# Sum of missing values
df.isnull().sum()
```

Out[10]:
```
Age                  0
WorkClass         1836
fnlwgt               0
Education            0
Education Number     0
marital-status       0
Ocuupation        1843
RelationShip         0
race                 0
Gender               0
Capital-gain         0
Capital-loss         0
Hours-per-Week       0
nativeCountry      583
Salary               0
dtype: int64
```

Visualize null values in a heatmap

In [11]:
```
# Visualize the null elements in the data set
sns.heatmap(df.isnull(),fmt = 'd')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2637556c828>



- Drop null values and duplicates

```
# Drop rows with missing values
df = df.dropna(how = 'any')
```

```
# Drop duplicates
df = df.drop_duplicates()
```

- Check for missing values:

In [14]:
```
# Find rows that contain missing values
missing_vals = df[df.isnull().any(axis=1)]
missing_vals
```

Out[14]:

Age	WorkClass	fnlwgt	Education	Education Number	marital-status	Ocuupation	RelationShip	race	Gender	Capital-gain	Capital-loss	Hours-per-Week	nativeCountry	Salary

- Check for any remaining null values

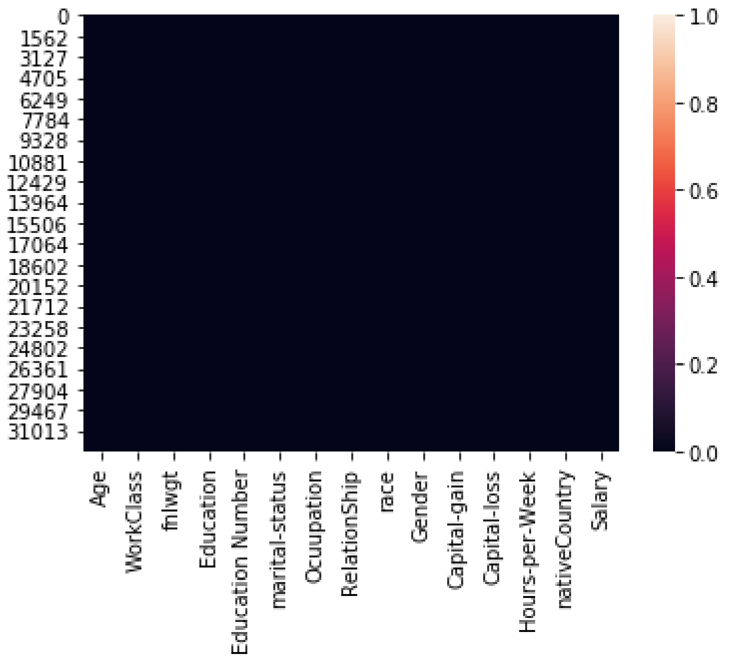
```
df.isnull().sum()
```

```
Age                 0
WorkClass           0
fnlwgt              0
Education           0
Education Number    0
marital-status      0
Ocuupation          0
RelationShip        0
race                0
Gender              0
Capital-gain        0
Capital-loss        0
Hours-per-Week      0
nativeCountry       0
Salary              0
dtype: int64
```

- Using heat map to check for null values

```
# Check again for null values using heat map
sns.heatmap(df.isnull(),fmt = 'd', vmin = 0, vmax = 1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef19e0b5c0>
```



- Convert all integer columns to integer values (in case they are of type float or string)

```
# Check for non-integer values
integer_cols = ['Age', 'fnlwgt', 'Education Number', 'Capital-gain', 'Capital-loss', 'Hours-per-week']
for col in df:
    # If data is supposed to be int
    if col in integer_cols:
        # Convert to int
        df.loc[:,col] = df[col].astype(int)
```

- Check to see if any non-integer values remain

```
# Check to see if any non integer values remain
for col in df:
    if col in integer_cols:
        print(col,' : ', df.loc[~df[col].astype(str).str.isdigit(), col].tolist())
```

```
Age  :  []
fnlwgt  :  []
Education Number  :  []
Capital-gain  :  []
Capital-loss  :  []
```

Make a list of all the possible values the data elements can take, for each column

```
: # Check for non string values or ones that do not follow the rules
possible_values = {
    'WorkClass':['Private', 'Self•-emp-•not-•inc', 'Self•-emp-•inc', 'Federal-•gov', 'Local-•gov', 'State•-gov', 'Without•-pay',
                 'Never-•worked'],

    'Education':['Bachelors', 'Some-college', '11th', 'HS-grad', 'Prof-school', 'Assoc-acdm', 'Assoc-voc', '9th', '7th-8th',
                 '12th', 'Masters', '1st-4th', '10th', 'Doctorate', '5th-6th', 'Preschool'],

    'marital-status':[ 'Married-civ-spouse', 'Divorced', 'Never-married', 'Separated', 'Widowed', 'Married-spouse-absent',
                       'Married-AF-spouse'],

    'Ocuupation':['Exec-managerial', 'specialty', 'Handlers-cleaners', 'Machine-op-inspct', 'Adm-clerical', 'Farming-fishing',
                  'Transport-moving', 'Priv-house-serv', 'Protective-serv', 'Armed-Forces'],

    'RelationShip': ['Wife', 'Own-child', 'Husband', 'Not-in-family', 'Other-relative', 'Unmarried'],

    'race':['White', 'Asian-Pac-Islander', 'Amer-Indian-Eskimo','Other', 'Black'],

    'Gender':['Female', 'Male'],

    'nativeCountry':['United-states', 'Cambodia', 'England', 'Puerto-Rico', 'Canada', 'Germany', 'Outlying-US(Guam-USVI-etc)',
                     'India', 'Japan', 'Greece', 'South', 'China', 'Cuba', 'Iran', 'Honduras', 'Philippines', 'Italy', 'Poland',
                     'Jamaica', 'Vietnam', 'Mexico', 'Portugal', 'Ireland', 'France', 'Dominican-Republic', 'Laos', 'Ecuador',
                     'Taiwan', 'Haiti', 'Columbia', 'Hungary', 'Guatemala', 'Nicaragua', 'Scotland', 'Thailand', 'Yugoslavia',
                     'El-Salvador', 'Trinadad&Tobago', 'Peru', 'Hong', 'Holand-Netherlands'],

    'Salary':['<=50k', '>50k']
}

# Replace soft hypens
for key in possible_values:
    possible_values[key] = [word.replace('\xad', '') for word in possible_values[key]]
```

- Find elements that are not in the list of possible values

```
# Find rows in df that are not proper
for col in possible_values:
    c = 0
    for row in df[col]:
        if row not in possible_values[col]:
            print(row, c)
        c+=1
```

- We see that we get no output. This means that all the data already is in the list of possible values, and there are no incorrect observations.

Final step – to save the cleaned data set as an excel file

```
writer = pd.ExcelWriter('CleanDataSet.xlsx',engine='xlsxwriter')
df.to_excel(writer,sheet_name='Lists')
writer.save()
```

Graphical visualization

- Cleaned data frame rows: 30138 , and columns: 14
- Collecting samples from cleaned data frame
- Work class

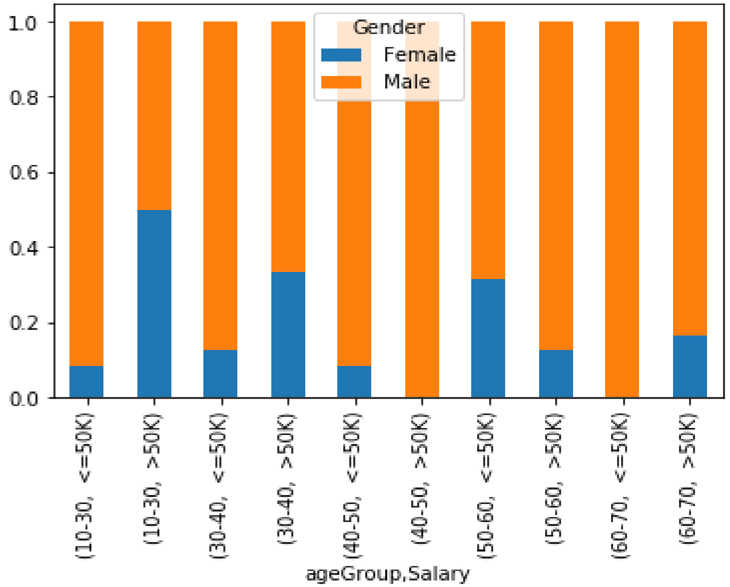
Self-emp-not-inc :	2498	->	100
Private:	22264	->	100
State-gov:	1278	->	100
Federal-gov:	943	->	100
Local-Gov:	2067	->	100
Self-emp-inc:	1074	->	100
without-pay:	14	->	14

→ Final length of sample : 614

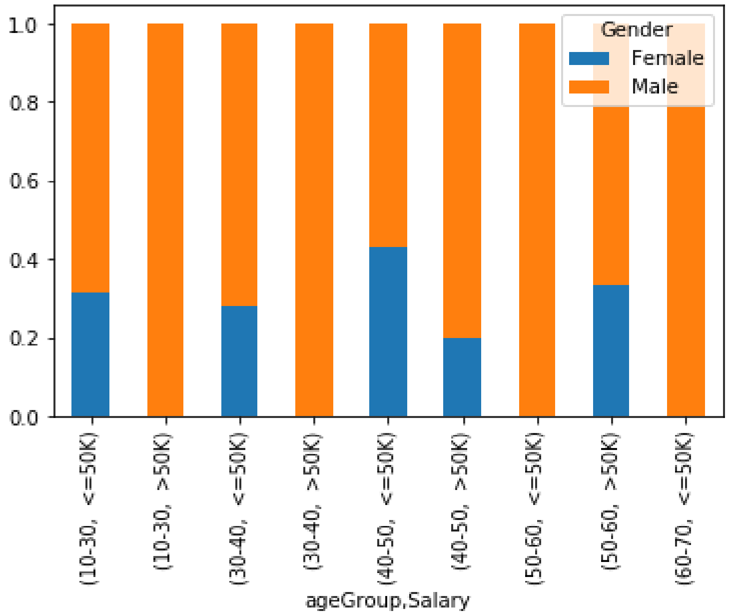
Graphical visualization

Graphs for each Work class

Self-emp-not-inc

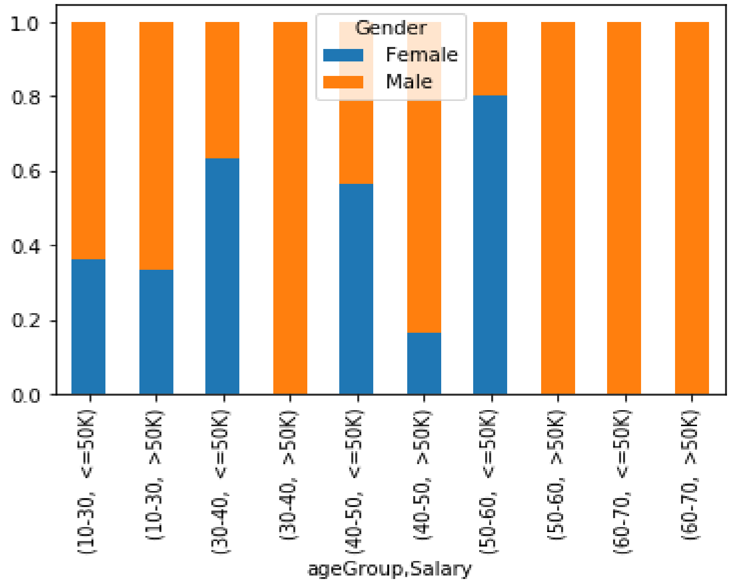


Private



Graphical visualization
Graphs for each Work class

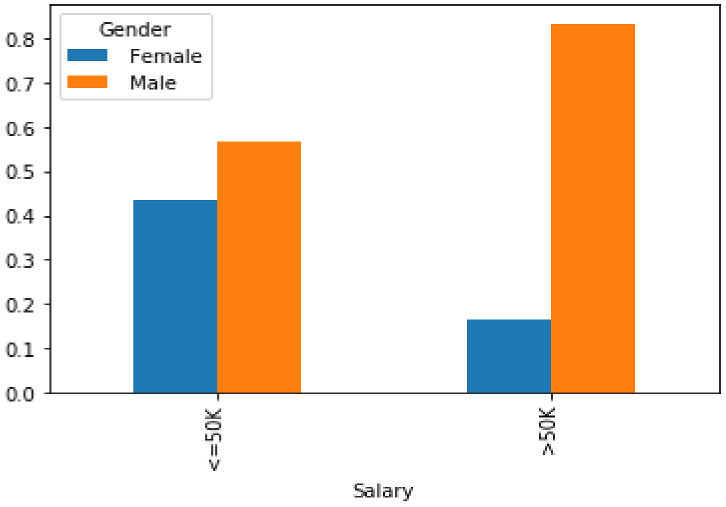
State-gov

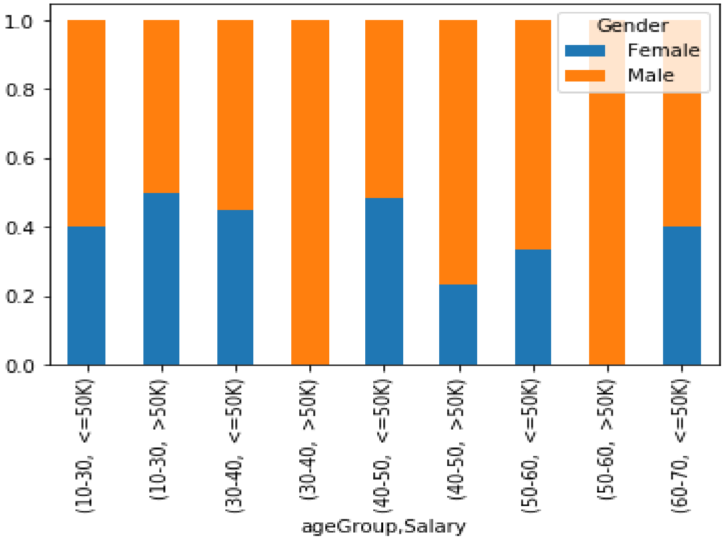


Federal-gov

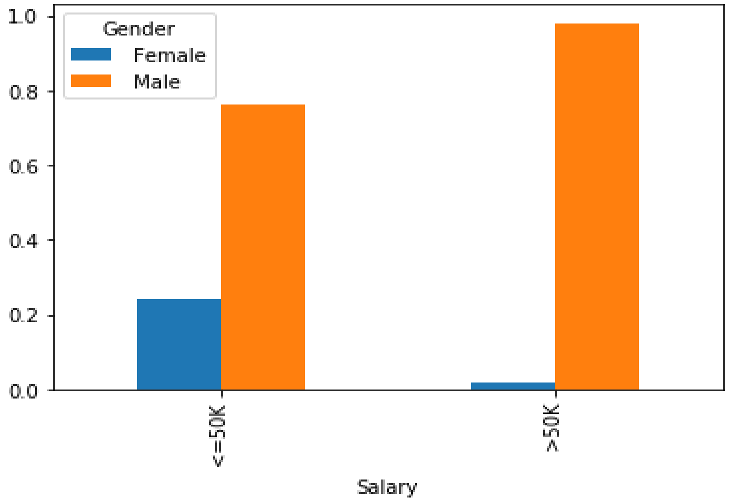
Graphical visualization
Graphs for each Work class

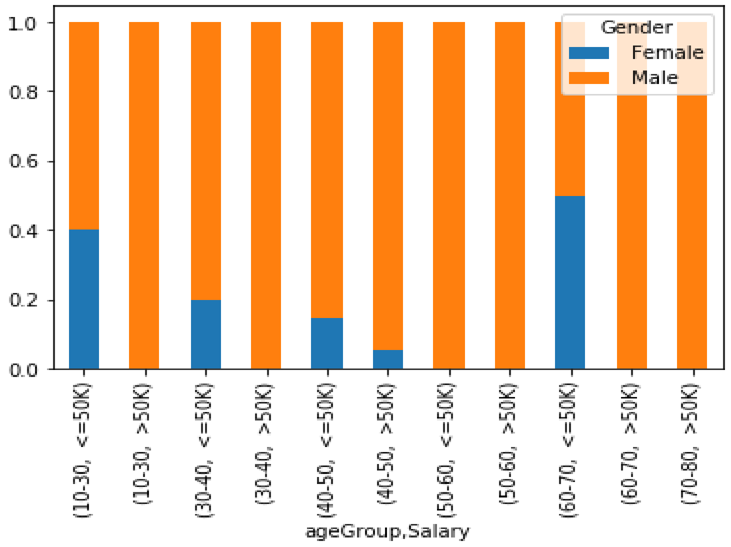
Local-gov





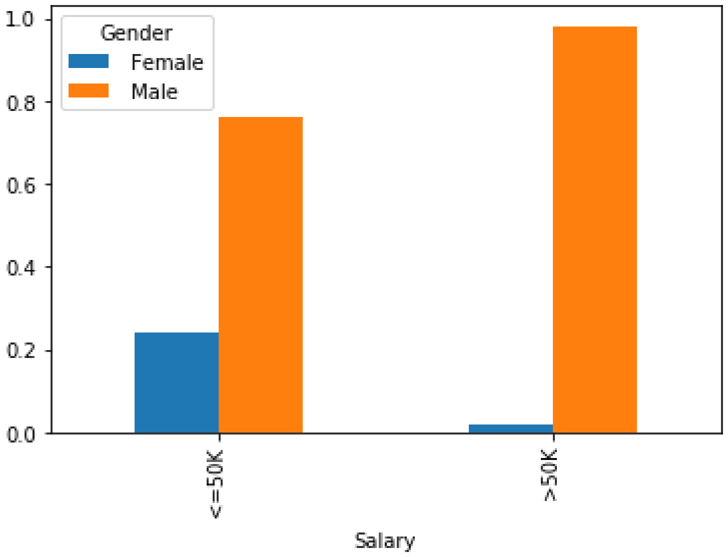
Self-emp-inc

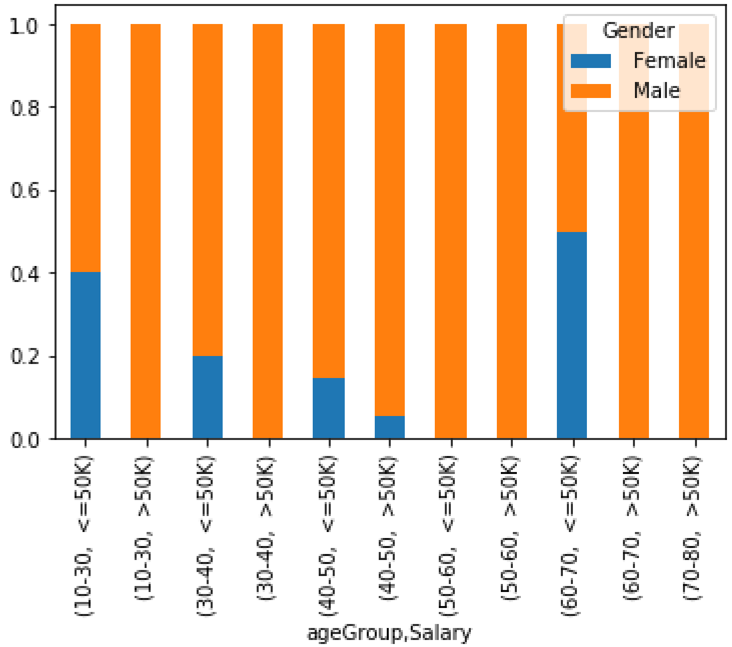




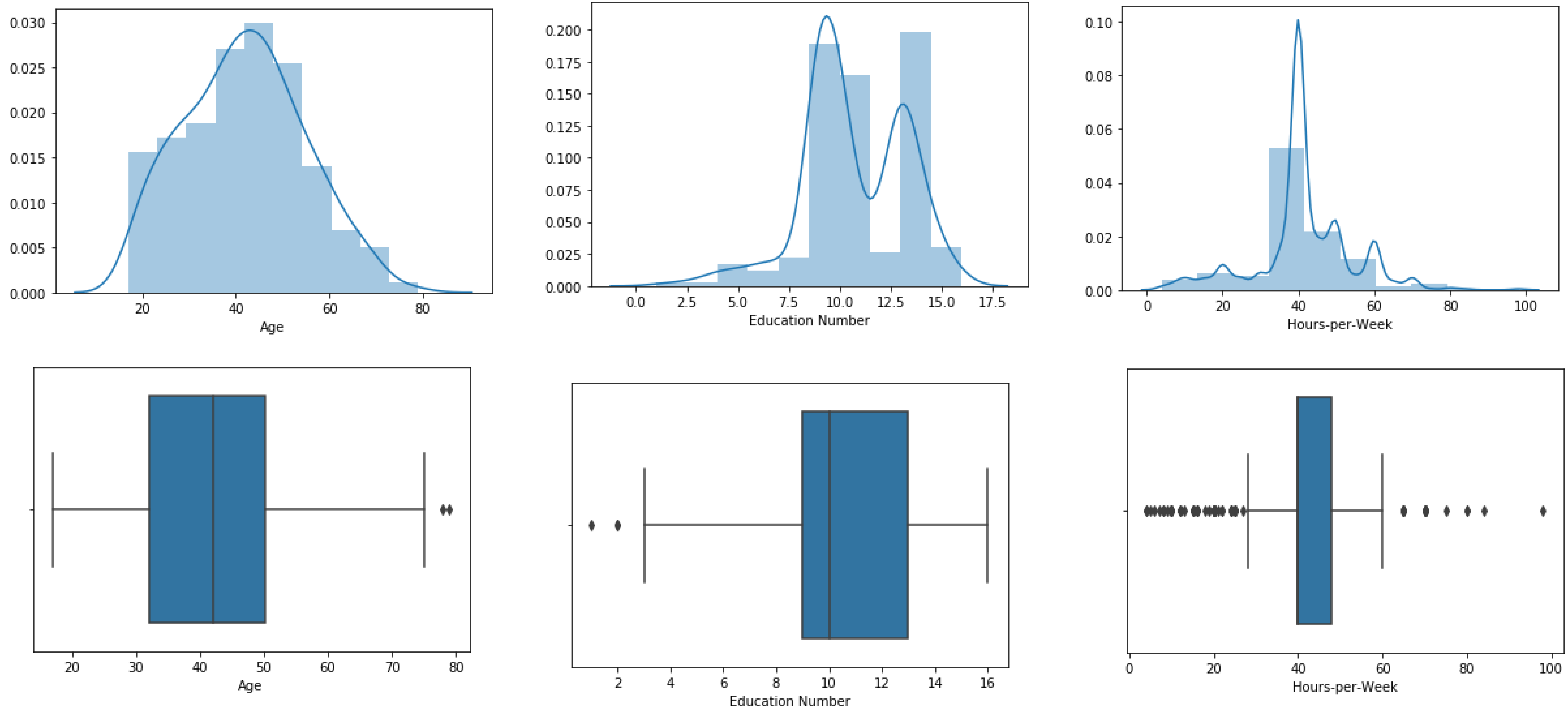
Graphical visualization
Graphs for each Work class

Without-pay

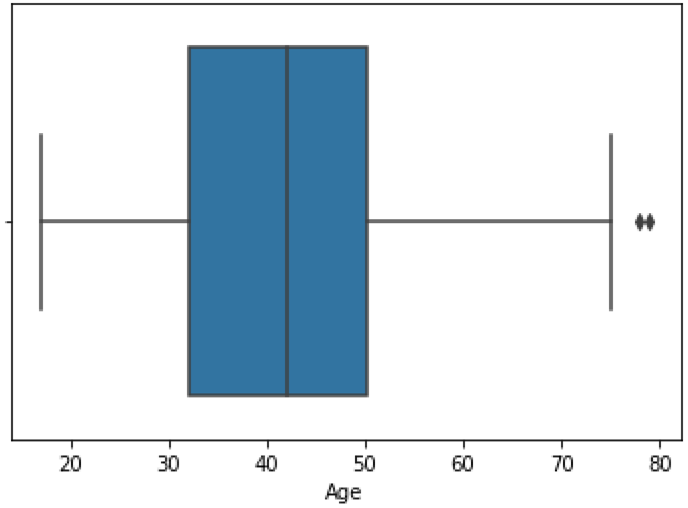


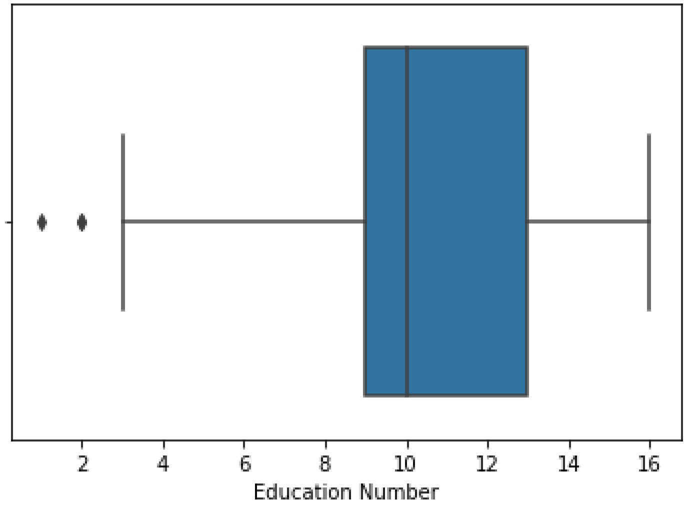


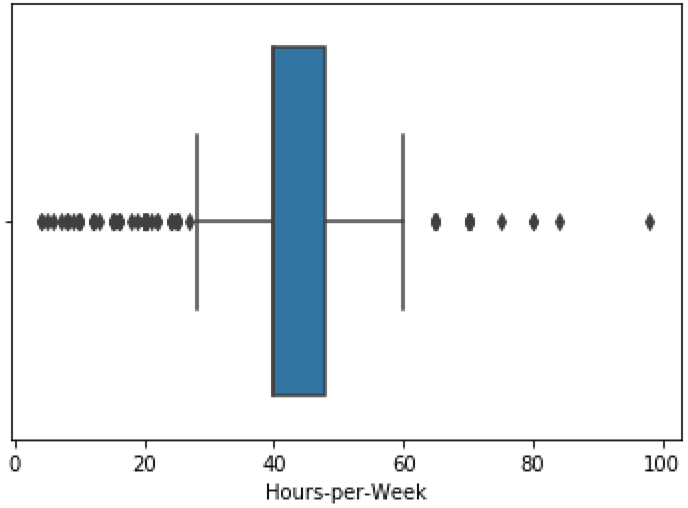
Removing Outliers

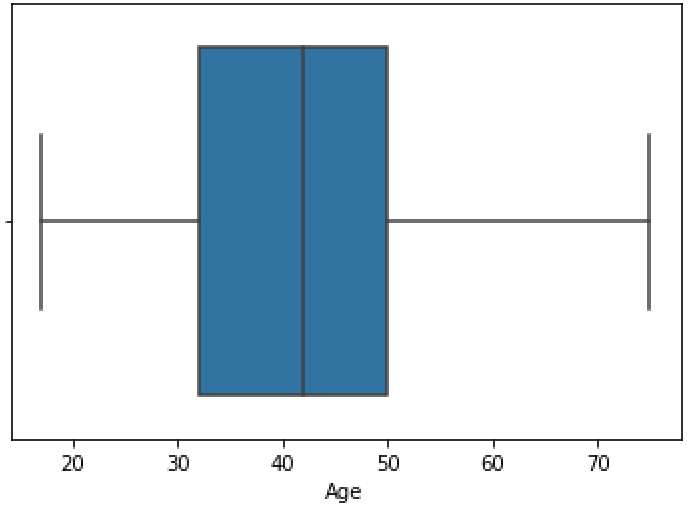


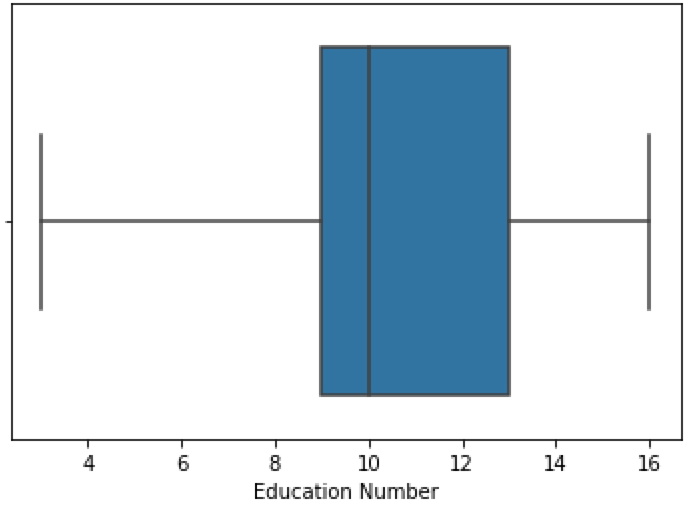
Removing Outliers

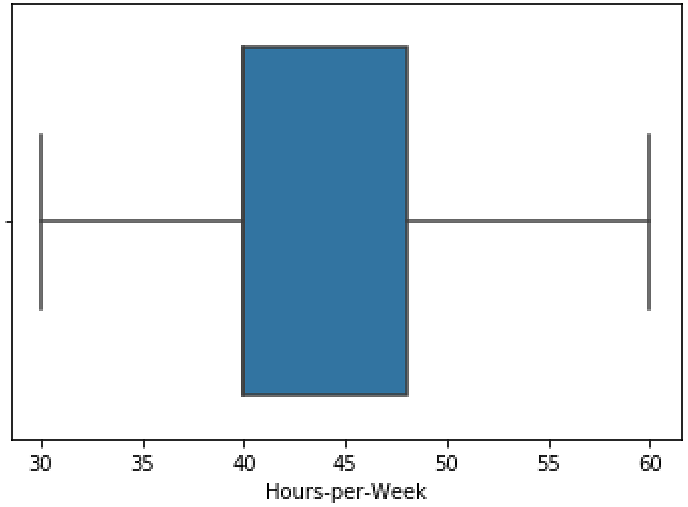






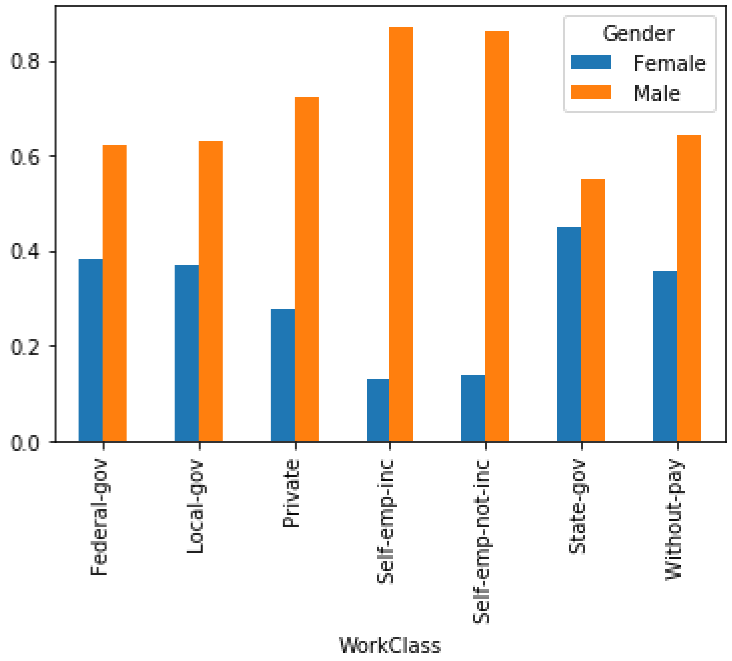


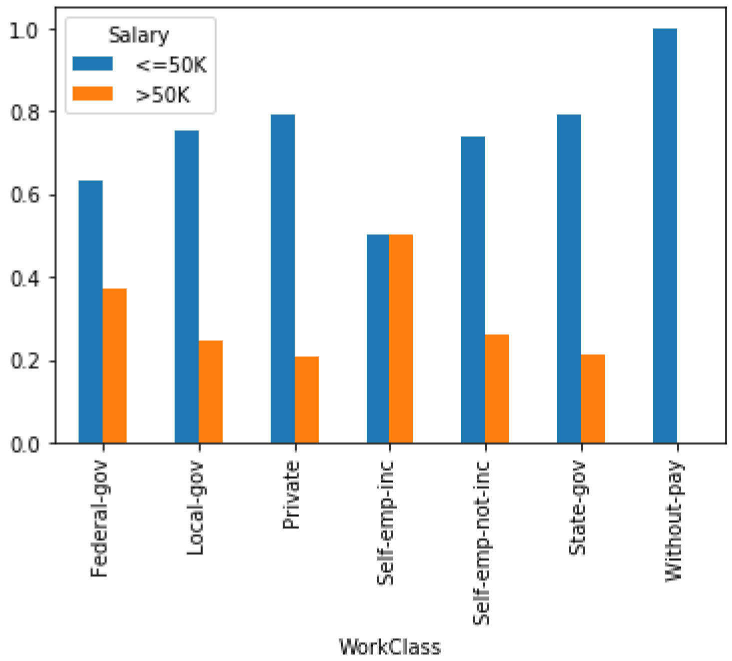


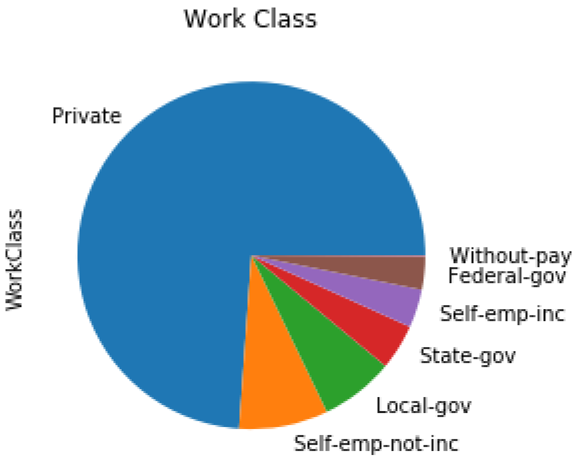


Includes all Work Classes

```
<matplotlib.axes._subplots.AxesSubplot at 0x16ce8322b08>
```







Normalization and standardization

- Calculation of mean and variance of all the columns

```
3]  from sklearn import preprocessing
    import pandas as pd
```

```
5]  clean_df = pd.read_excel('FinalDataSet.xlsx')
```

```
7]  column_head = ["Age","WorkClass","fnlwgt","Education","Education Number","marital-status","Ocuupation",
                   "RelationShip","race","Gender","Capital-gain","Capital-loss","Hours-per-Week","nativeCountry","Salary"]
    integer_cols = ["Age", "fnlwgt", "Education Number", "Capital-gain","Capital-loss","Hours-per-Week",]
    non_integer_cols = []

    for col in clean_df:
      if col not in integer_cols:
        non_integer_cols.append(col)
```

```
[10] for col in integer_cols:
       print('Variance of', col, '=', clean_df[col].var())
```

```
Variance of Age = 166.4007630278331
Variance of fnlwgt = 10872948761.66785
Variance of Education Number = 6.487207903407627
Variance of Capital-gain = 55933110.094337605
Variance of Capital-loss = 162706.14614584157
Variance of Hours-per-Week = 160.0289170207227
```

```
[11] for col in integer_cols:
       print('mean of',col,'=',clean_df[col].mean())
```

```
mean of Age = 41.89473684210526
mean of fnlwgt = 181435.36184210525
mean of Education Number = 10.682565789473685
mean of Capital-gain = 1192.3552631578948
mean of Capital-loss = 86.54605263157895
mean of Hours-per-Week = 41.58552631578947
```

- Normalization of all numeric columns

```
Age_ = []
Fnlwgt_ = []
Education_Number_ = []
Hours_per_Week_ = []

for i in age:
    Age_.append(i)
for i in fnlwgt:
    Fnlwgt_.append(i)
for i in Education_Number:
    Education_Number_.append(i)
for i in Hours_per_Week:
    Hours_per_Week_.append(i)

New_Age = [Age_]
New_fnlwgt = [Fnlwgt_]
New_Education_Number = [Education_Number_]
New_Hours_per_Week = [Hours_per_Week_]
```

```
In [6]: # normalized Columns


normalized_age = preprocessing.normalize(New_Age)
normalized_fnlwgt = preprocessing.normalize(New_fnlwgt)
normalized_Education_Number = preprocessing.normalize(New_Education_Number)
normalized_Hours_per_Week = preprocessing.normalize(New_Hours_per_Week)
```

- **Calculation of mean after normalization**

```
In [13]:
statistics.mean(normalized_age[0])
Out[13]:
0.03876238867029209

In [14]:
statistics.mean(normalized_fnlwgt[0])
Out[14]:
0.03516922047953876

In [15]:
statistics.mean(normalized_Education_Number[0])
Out[15]:
0.03945130615103028

In [16]:
statistics.mean(normalized_Hours_per_Week[0])
Out[16]:
0.038802565722631806
```

- **We can observe that the mean of all columns are zero after normalization.**

•Calculation of variance after normalization

In [26]:

```
df_ = df.drop(columns=['WorkClass','Education','marital-status','Ocuupation','RelationShip'
df
```

In [32]:

```
# finding sigma std
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(df_)
data_scaled
```

Out[32]:

```
array([[-1.46595499,  2.18638818, -3.41174543, -0.12543873],
       [ 1.63745947, -0.71874512, -0.6611509 , -0.12543873],
       [ 0.47367905, -1.24597653,  0.91061741,  0.66571013],
       ...,
       [ 2.02538628, -0.06469444, -0.26820882, -1.31216204],
       [ 1.94780091, -0.42465075, -0.6611509 , -2.34065557],
       [ 1.55987411, -0.20662106, -0.26820882, -2.02419602]])
```

In [33]:

```
print(data_scaled.std(axis=0))
```

```
[1. 1. 1. 1.]
```

StandardScaler results in a distribution with a standard deviation equal to 1. The variance is equal to 1 also, because variance = standard deviation squared. And 1 squared = 1.

Normalization:

Similarly, the goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

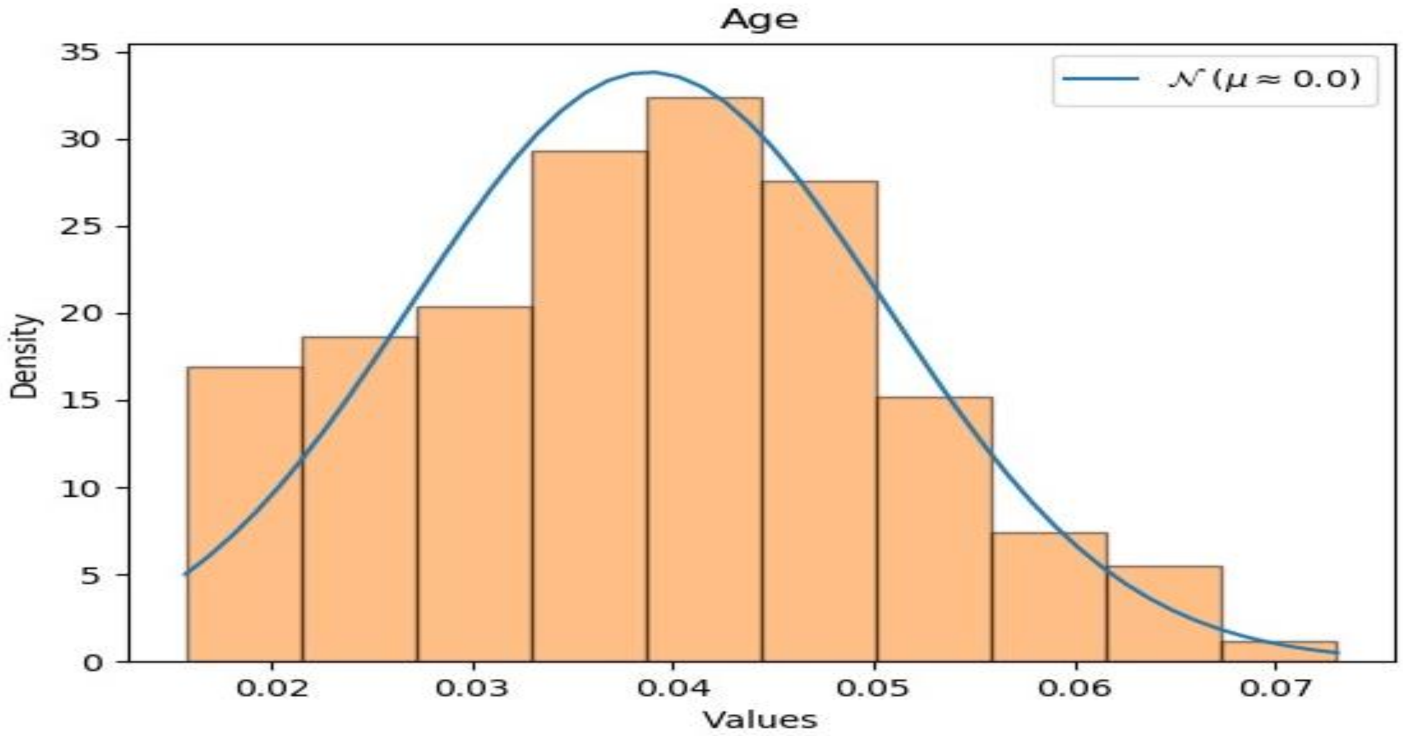
For example, consider a data set containing two features, age, and income(x2). Where age ranges from 0–100, while income ranges from 0–100,000 and higher. Income is about 1,000 times larger than age. So, these two features are in very different ranges. When we do further analysis, like multivariate linear regression, for example, the attributed income will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor. So we normalize the data to bring all the variables to the same range.

Normalization is a good technique to use when you do not know the distribution of your data or when you know the distribution is not Gaussian (a bell curve). Normalization is useful when your data has varying scales and the algorithm you are using does not make assumptions about the distribution of your data, such as k-nearest neighbors and artificial neural networks.

- Graphs after normalization
  - Age

```
In [13]: std = np.std(normalized_age[0],ddof=1)
         mean = np.mean(normalized_age[0])
```

```
In [38]: domain = np.linspace(np.min(normalized_age[0]),np.max(normalized_age[0]))
         plt.plot(domain,norm.pdf(domain,mean,std),
                  label='$\mathcal{N}$ '+f'$( \mu \\approx {round(mean)} )$')
         plt.hist(normalized_age[0],edgecolor ='black',alpha=.5,density=True)
         plt.title("Age")
         plt.xlabel("Values")
         plt.ylabel("Density")
         plt.legend()
         plt.show()
```



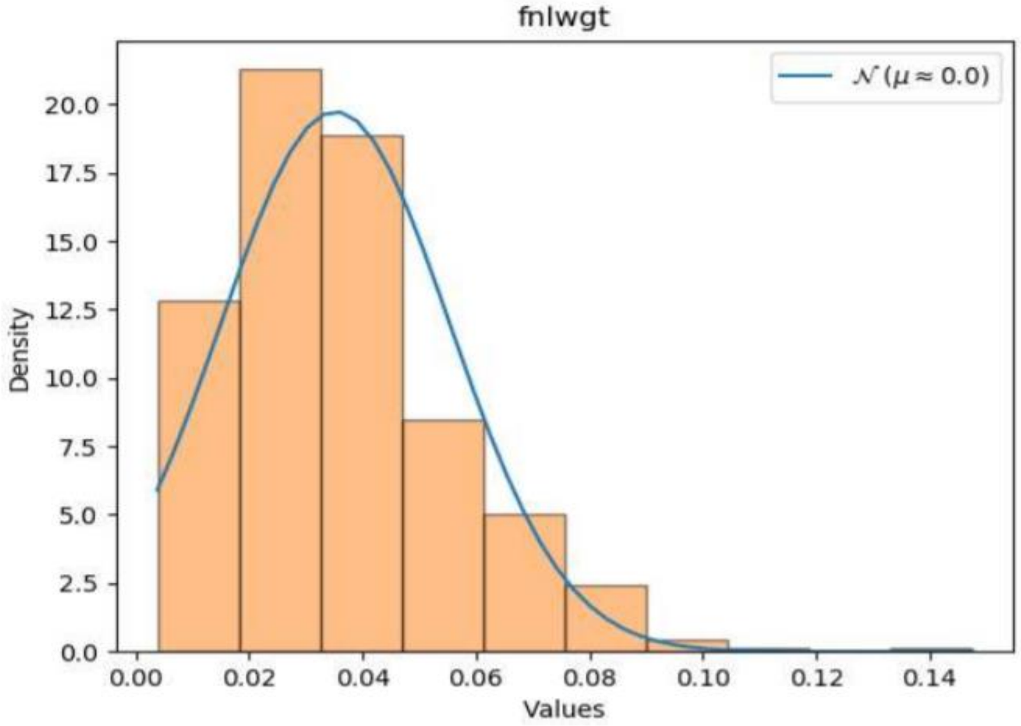
- fnlwgt

In [18]:

```
std = np.std(normalized_fnlwgt[0],ddof=1)
mean = np.mean(normalized_fnlwgt[0])
```

In [19]:

```
domain = np.linspace(np.min(normalized_fnlwgt[0]),np.max(normalized_fnlwgt[0]))
plt.plot(domain,norm.pdf(domain,mean,std),
         label='$\mathcal{N}$ '+f'$( \mu \\approx {round(mean)} )$')
plt.hist(normalized_fnlwgt[0],edgecolor ='black',alpha=.5,density=True)
plt.title("fnlwgt")
plt.xlabel("Values")
plt.ylabel("Density")
plt.legend()
plt.show()
```



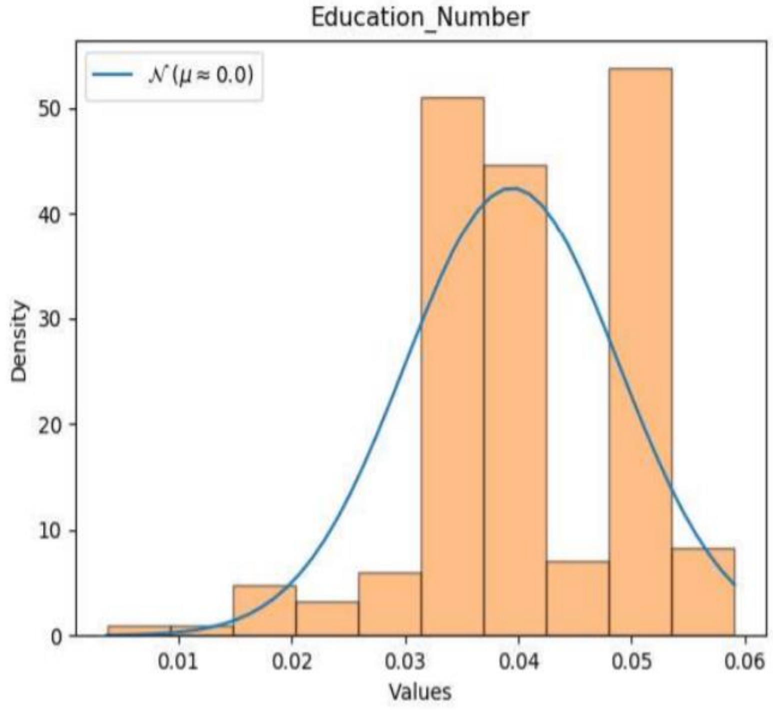
- Education Number

In [21]:

```
std = np.std(normalized_Education_Number[0],ddof=1)
mean = np.mean(normalized_Education_Number[0])
```

In [22]:

```
domain = np.linspace(np.min(normalized_Education_Number[0]),np.max(normalized_Education_Num
plt.plot(domain,norm.pdf(domain,mean,std),
         label='$\mathcal{N}$ '+f'$( \mu \\approx {round(mean)} )$')
plt.hist(normalized_Education_Number[0],edgecolor ='black',alpha=.5,density=True)
plt.title("Education_Number")
plt.xlabel("Values")
plt.ylabel("Density")
plt.legend()
plt.show()
```



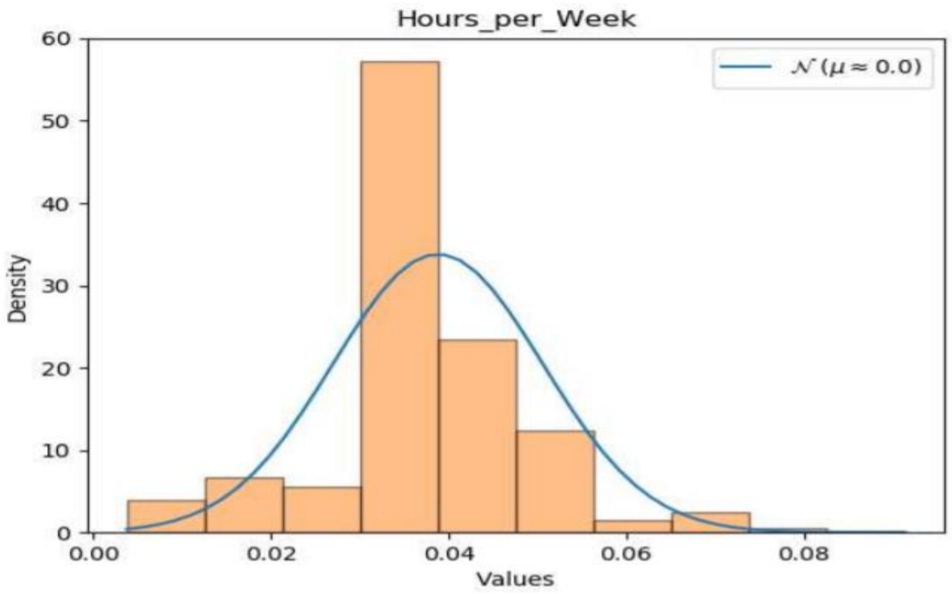
- hours per week

In [24]:

```
std = np.std(normalized_Hours_per_Week[0],ddof=1)
mean = np.mean(normalized_Hours_per_Week[0])
```

In [25]:

```
domain = np.linspace(np.min(normalized_Hours_per_Week[0]),np.max(normalized_Hours_per_Week[
plt.plot(domain,norm.pdf(domain,mean,std),
        label='$\mathcal{N}$ '+f'$( \mu \\approx {round(mean)} )$')
plt.hist(normalized_Hours_per_Week[0],edgecolor ='black',alpha=.5,density=True)
plt.title("Hours_per_Week")
plt.xlabel("Values")
plt.ylabel("Density")
plt.legend()
plt.show()
```



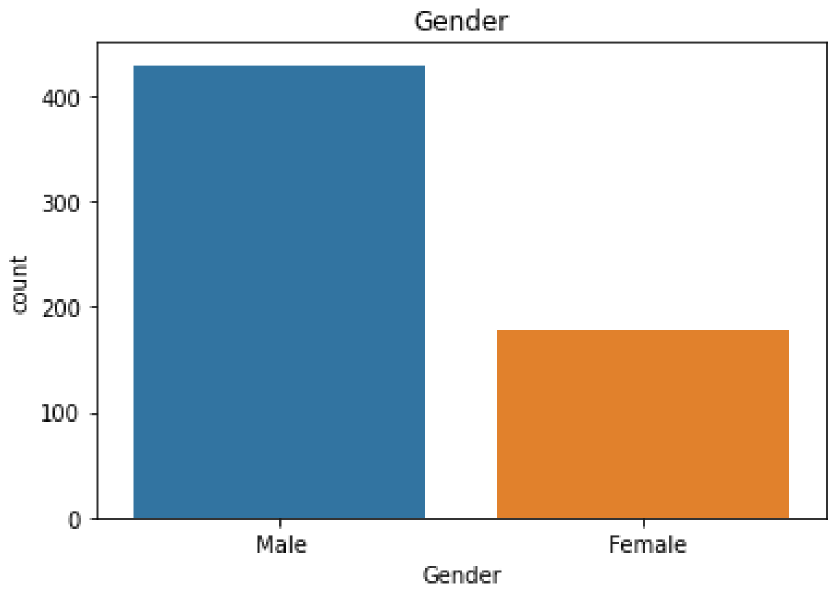
- **As we can observe that all graph are bell shaped in nature, we can conclude that they are normal.**

Hypothesis testing

Hypothesis based on the columns:
1)There is any relationship between Gender and income. (Hypothesis)
2)If there is any association, then is it positive or negative.
3)Check the fact that men are earning more money than female.

```
sns.countplot(x = 'Gender',data = df)
plt.title("Gender")
```

```
Text(0.5, 1.0, 'Gender')
```




```
df9 = pd.crosstab(df.Gender , df.Salary)
print("Following is contigency table")
df9
```

Following is contigency table

Salary	<=50K	>50K
Gender		
Female	154	24
Male	276	154

```
df_chi = pd.read_csv('dataset.csv')
contingency_table=pd.crosstab(df_chi["Gender"],df_chi["Salary"])
print('contingency_table :-\n',contingency_table)
#Observed Values
Observed_Values = contingency_table.values
print("Observed Values :-\n",Observed_Values)
b=stats.chi2_contingency(contingency_table)
Expected_Values = b[3]
print("Expected Values :-\n",Expected_Values)
no_of_rows=len(contingency_table.iloc[0:2,0])
no_of_columns=len(contingency_table.iloc[0,0:2])
ddof=(no_of_rows-1)*(no_of_columns-1)
print("Degree of Freedom:-",ddof)
alpha = 0.05
```

```
contingency_table :-
 Salary    <=50K   >50K
Gender
 Female     154     24
 Male       276    154
Observed Values :-
 [[154  24]
 [276 154]]
Expected Values :-
 [[125.88815789  52.11184211]
 [304.11184211 125.88815789]]
Degree of Freedom:- 1
```

Doing statistical chi square test to check association between Gender and income

Hypothesis

Null hypothesis Ho = There is no association between Gender and income

Alternative hypothesis H1 = There is association

```
a1 = [154,24]
a2 = [276,154]
a3 = np.array([a1,a2])
from scipy import stats
stats.chi2_contingency(a3)
print('Significance level: ',alpha)
critical_value=chi2.ppf(q=1-alpha,df=ddof)
print('critical_value:',critical_value)
chi2_stat, p_val, dof, ex = stats.chi2_contingency(a3)
print("Chisquare test value is : ",chi2_stat)
print("\nP-Value is : ",p_val)
print()
if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")
```

```
Significance level:  0.05
critical_value: 3.841458820694124
Chisquare test value is :  29.249913397777643

P-Value is :  6.361906975325 25e-08

Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
```

Since from above result , Chi square value is greater than P- value so we reject H0 and conclude that there is association between Gender and income

Now second objective, whether it is positive or negative association.
To check Positive or negative association, We use concepts as follow,
Gender has two level a1 = Female and a2 = Male. Income has two level b1 = '<=50K' and b2 = '>50k'
then following is formula is used to find association

```
x,y,z = a3[0][1]+a3[0][0],a3[1][1]+a3[1][0],a3[0][0]+a3[1][0]+a3[0][1]+a3[1][1]
print('Number of female earning less than <=50K is ',a3[0][0])
print('Number of female observation is ',a3[0][1]+a3[0][0])
print('Number of male ',a3[1][1]+a3[1][0])
print('Total observation is ',a3[0][0]+a3[1][0]+a3[0][1]+a3[1][1])
print("Value of evaluation metric is ",((x*y)/z))
```

```
Number of female earning less than <=50K is  154
Number of female observation is  178
Number of male  430
Total observation is  608
Value of evaluation metric is  125.88815789473684
```

```
df10 = (df.groupby(['Gender','Salary']).WorkClass.count()/df.groupby(['Gender']).WorkClass.count())*100
df10
```

```
Gender   Salary
 Female   <=50K     86.516854
          >50K      13.483146
 Male     <=50K     64.186047
          >50K      35.813953
Name: WorkClass, dtype: float64
```

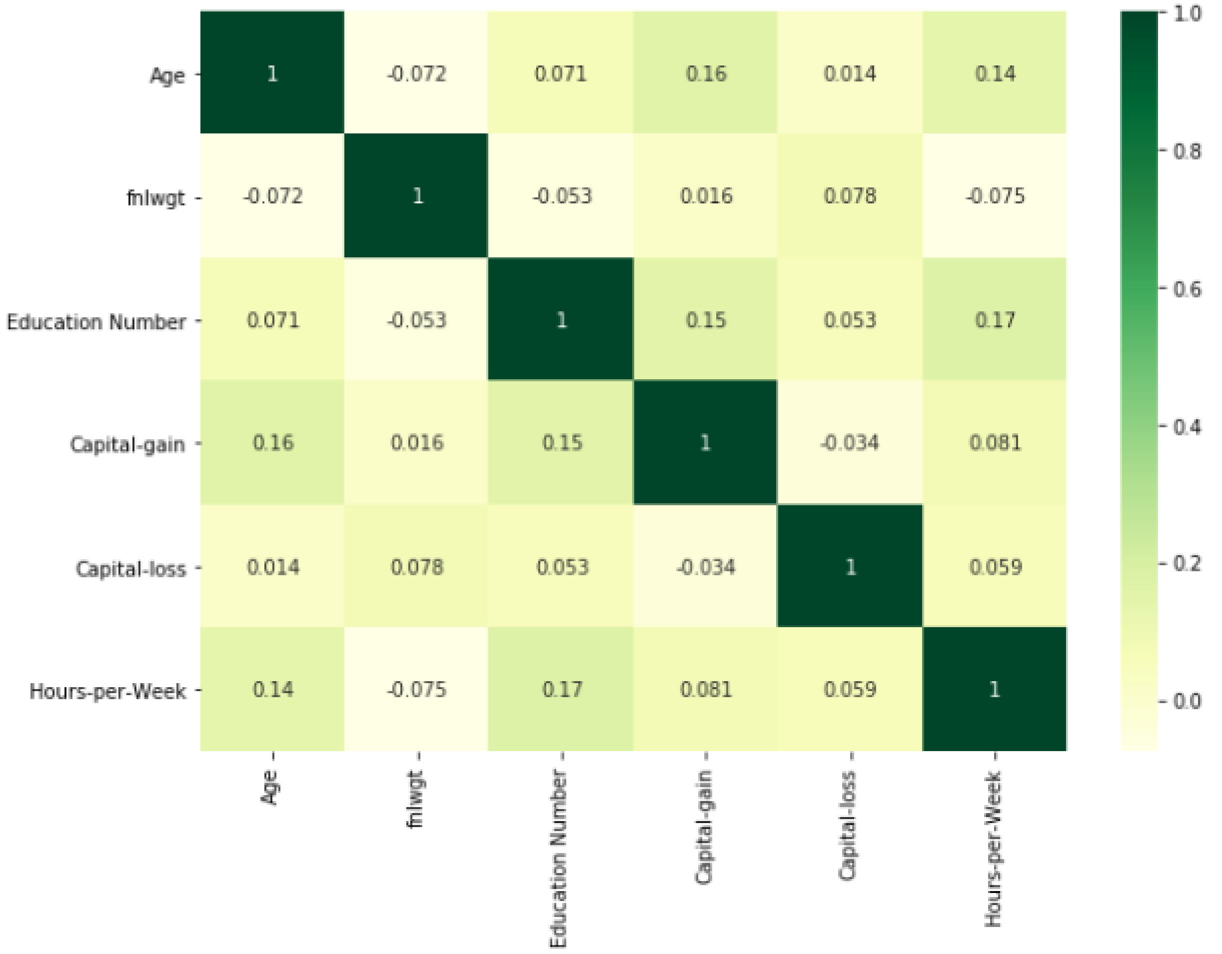
for positive association (a1b1) > ((a1)(b1))/N
for negative association (a1b1) < ((a1)(b1))/N
where,
(a1b1) = Number of Female earning less than <=50K (a1) = Total number of Female observation (b1) = Total number of observation earning less than <=50K N = Total number of observations

From above result ,
20% more female is earning <=50K than male, while 20% more male is earning >50K than female.
This revealed the fact that Male make more money than female if income >50K while Female make more money if income <=50K.

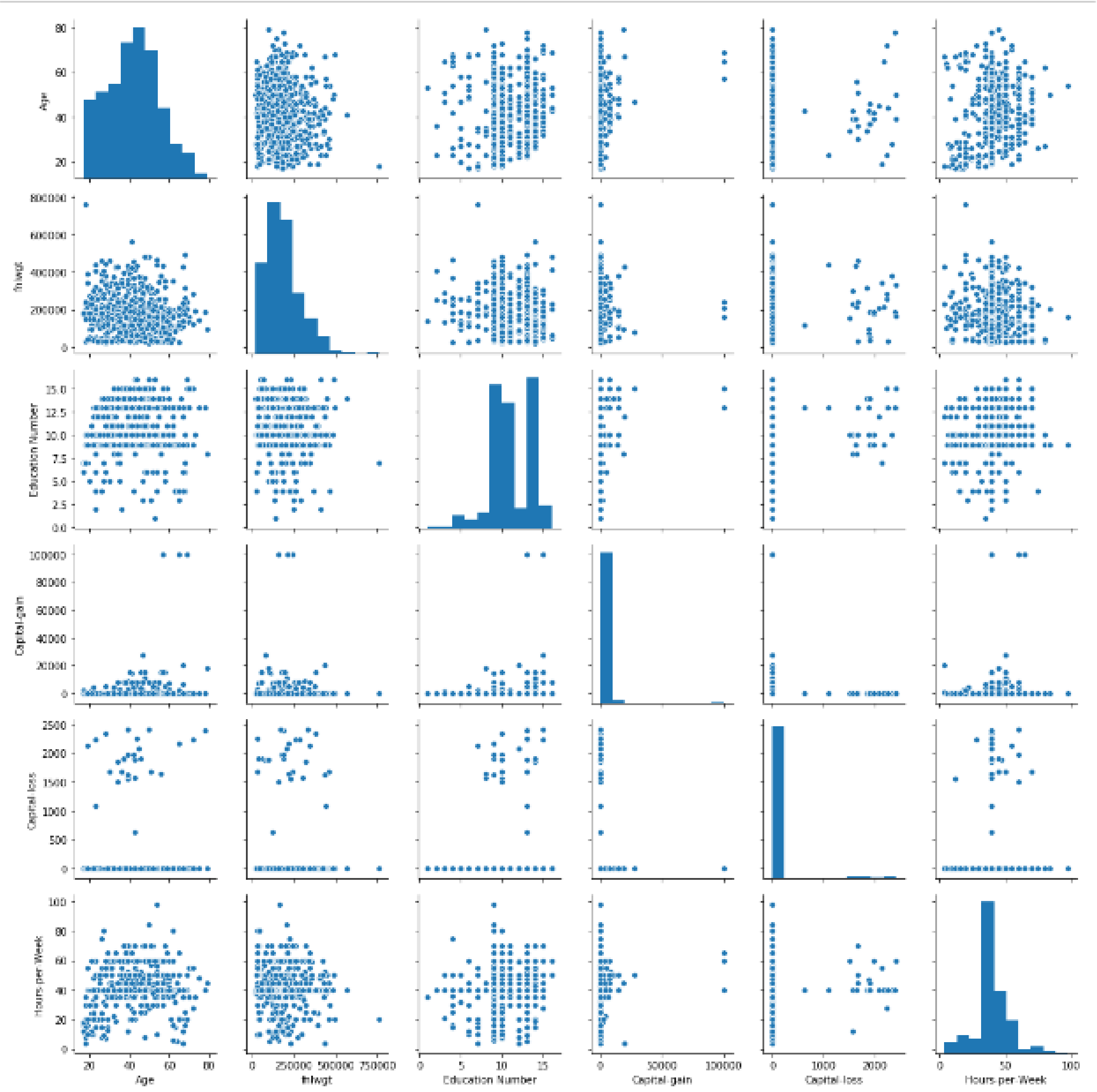
Correlation

- We got 15 relations ,4 – negative and 11 – positive

	Age	fnlwgt	Education Number	Capital-gain	Capital-loss	Hours-per-Week
Age	1.000000	-0.072332	0.071136	0.160688	0.014409	0.137800
fnlwgt	-0.072332	1.000000	-0.052771	0.015825	0.077863	-0.074558
Education Number	0.071136	-0.052771	1.000000	0.149951	0.052677	0.170368
Capital-gain	0.160688	0.015825	0.149951	1.000000	-0.034263	0.081109
Capital-loss	0.014409	0.077863	0.052677	-0.034263	1.000000	0.058592
Hours-per-Week	0.137800	-0.074558	0.170368	0.081109	0.058592	1.000000



Correlation



Correlation

correlation coefficient (r)

0.15 0.17 0.16 0.14