Student ID/Registration no: 220147226

Name: Sudip Subedi

Canvas ID: bi49rb

Module code: CET313 Artificial Intelligence

Protype Development Documentation

Introduction

I plan to create a program that can forecast a person salary. The pay of different people from different jobs and position will be used in then dataset, machine learning methods are applied. Jupyter software was used to create the program.

Data Munging (Process of data cleaning)

Firstly I will import the necessary libraries required to build this protype like numpy, pandas, seaborn and matplotlib and then we will load the dataset to the program and read it to perform operations on it. The description of data is executed using df.describe() command. This will show all features of the dataset as follows:

In [1]:
```
import numpy as np # mathematical calculation
import pandas as pd  # data preprocessing
import matplotlib.pyplot as plt # data visualization
import seaborn as sns  # data visualization
```

```
# Display statical infromation
df.describe()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

This table visualize the statical information of each numerical column and rows. In the table we can see the mean, stander deviation and maximum values of the features.

```
# display information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              32561 non-null  int64 
 1    workclass       32561 non-null  object
 2    fnlwgt          32561 non-null  int64 
 3    education       32561 non-null  object
 4    education-num   32561 non-null  int64 
 5    marital-status  32561 non-null  object
 6    occupation      32561 non-null  object
 7    relationship    32561 non-null  object
 8    race            32561 non-null  object
 9    sex             32561 non-null  object
 10   capital-gain    32561 non-null  int64 
 11   capital-loss    32561 non-null  int64 
 12   hours-per-week  32561 non-null  int64 
 13   native-country  32561 non-null  object
 14   salary          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Checking for missing data in the dataset is critical since processing the data will result in the loss of some column features. Fortunately, my dataset had no missing data to deal with.

```
# display number of rows and column
df.shape
```

```
(32561, 15)
```

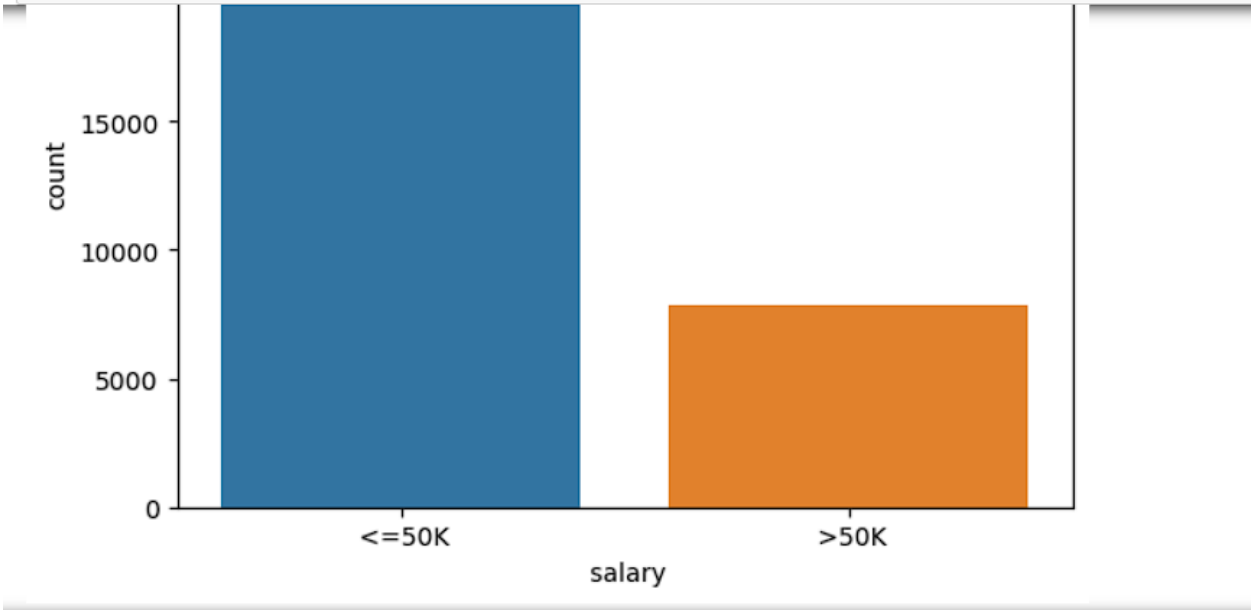
This is an attribute of the data frame object that returns a tuple containing the numbers of rows and column. Here we can see that there is 15 column and 32561 rows present in the dataset.

```
In [9]: #used to obtain the count of unique values
        df[" salary"].value_counts()

Out[9]:  <=50K    24720
         >50K      7841
        Name:  salary, dtype: int64
```

Here we can see there are two types of employees who have salary equal or less than fifty thousand and more than fifty thousand. There are 24720 employee who have less or equal than fifty thousand and 7841 employees have more than fifty thousand.

```
#Displaying count plot based on salary
sns.countplot(x=' salary',data=df )
```



The given bar graph shows the number of people who have less or equal salary than fifty thousand per year.

Exploratory data analysis and Feature Engineering

```
## Here we will check the missing values in our dataset
df.isnull().sum()
```

```
age               0
 workclass        0
 fnlwgt           0
 education        0
 education-num    0
 marital-status   0
 occupation       0
 relationship     0
 race             0
 sex              0
 capital-gain     0
 capital-loss     0
 hours-per-week   0
 native-country   0
 salary           0
dtype: int64
```

The above code analyze and manipulate dataset and then checks if there is any missing values in the dataset but in our case there are no missing values in the dataset.

```
#display columns form data frame
numerical = [i for i in df.columns if df[i].dtypes != "O"]
```

```
numerical
```

```
['age',
 ' fnlwgt',
 ' education-num',
 ' capital-gain',
 ' capital-loss',
 ' hours-per-week']
```

Here we are going to identify missing values, understand the structure, content and explore the numerical columns in the dataset

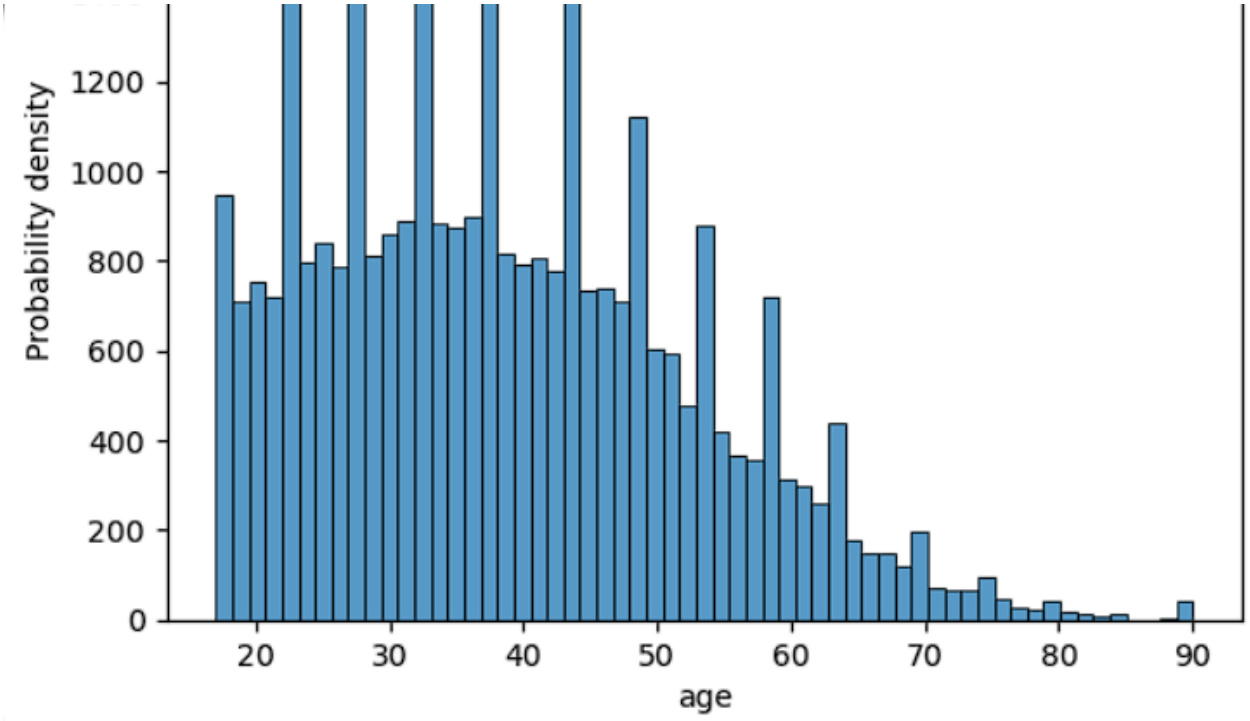
```
for i in numerical:
    print(f" {i}  :  {len(df[i].unique())}")
```

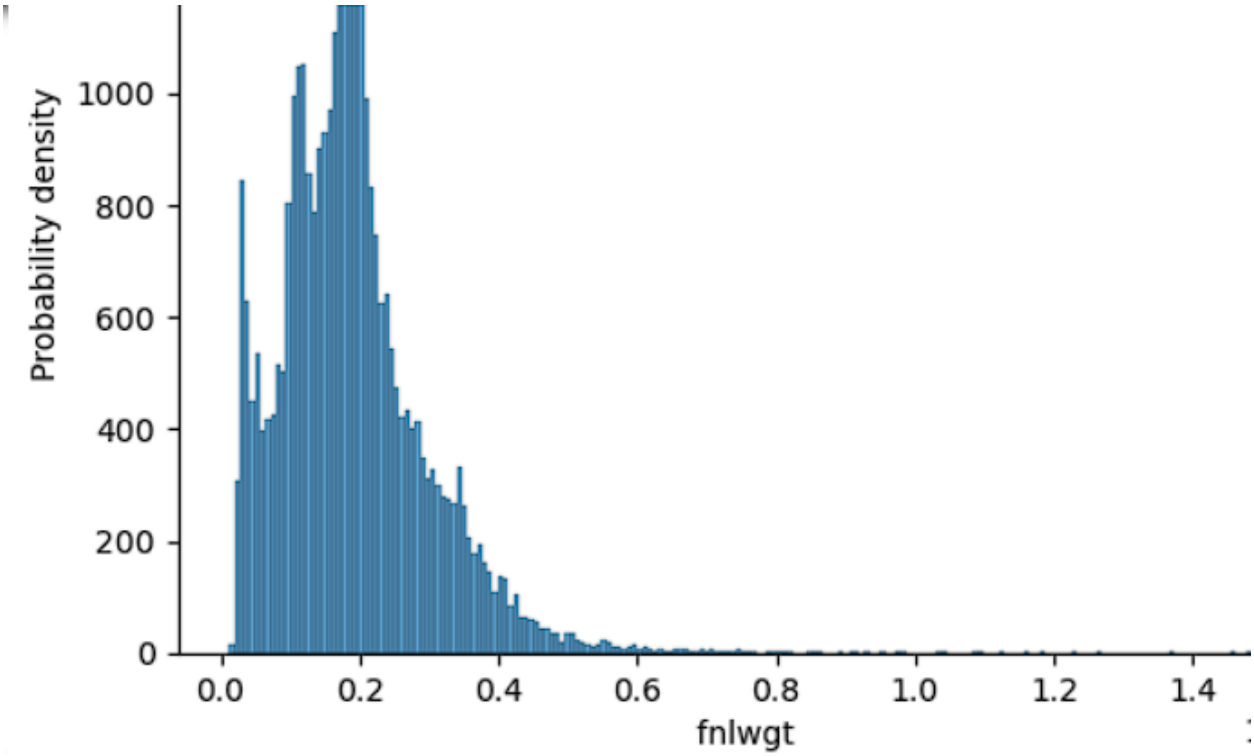
```
 age  :  73
 fnlwgt  :  21648
 education-num  :  16
 capital-gain  :  119
 capital-loss  :  92
 hours-per-week  :  94
```

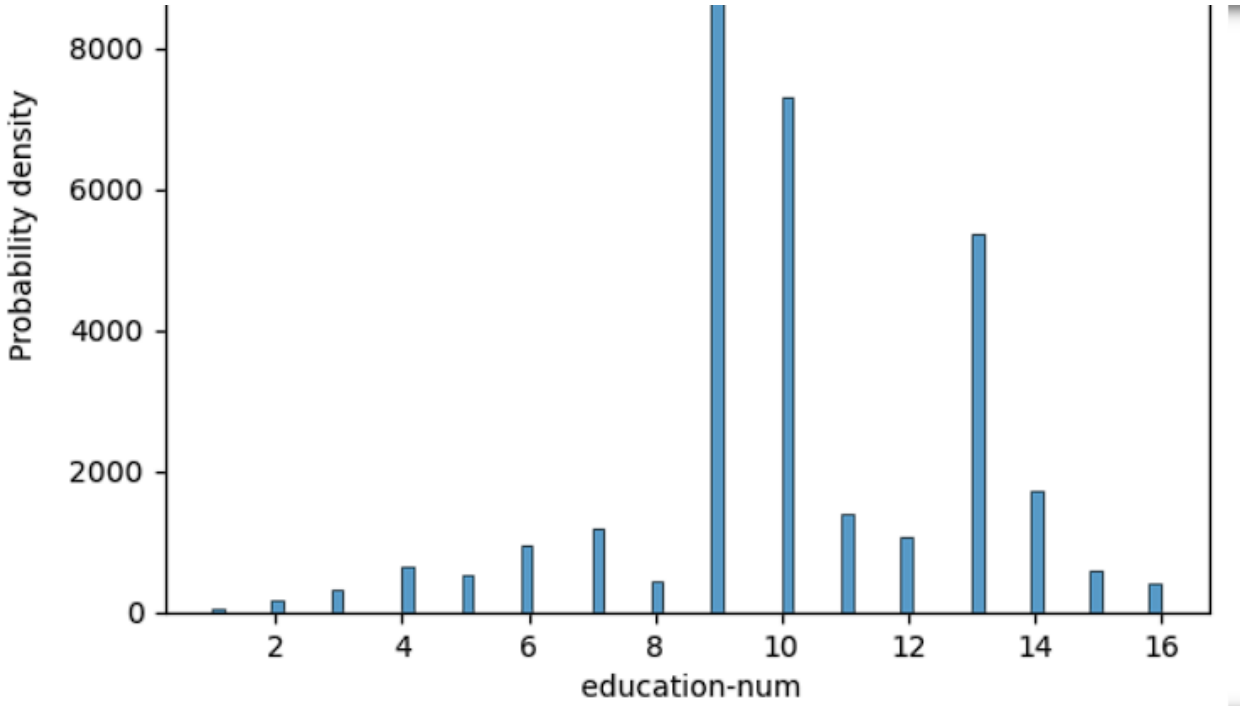
The chart shows that ‘age' column has 73 unique value the ‘fnlwgt’ column has 21648 unique values, ‘education-num’ column has 16 unique values. The ‘capital-gain’ column has 119 uniques values, the ‘capital-loss’ column has 92 unique value and the ‘hours-per-week’ has 94 unique values.

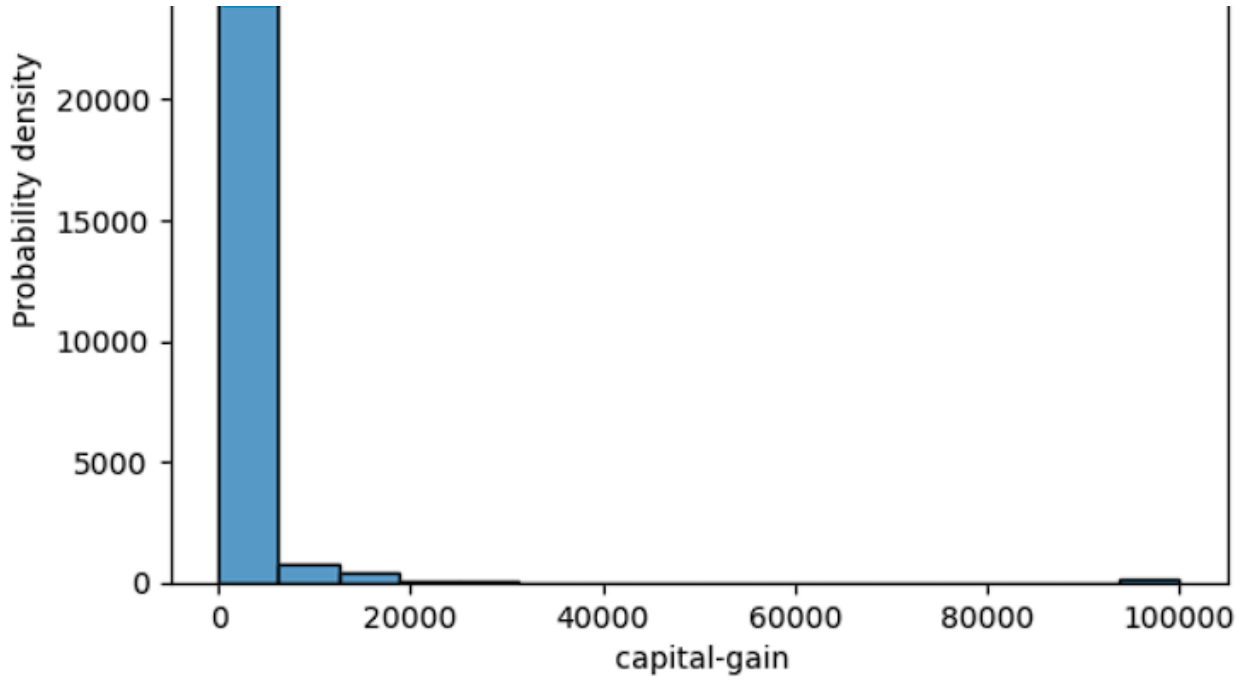
Data Visualization

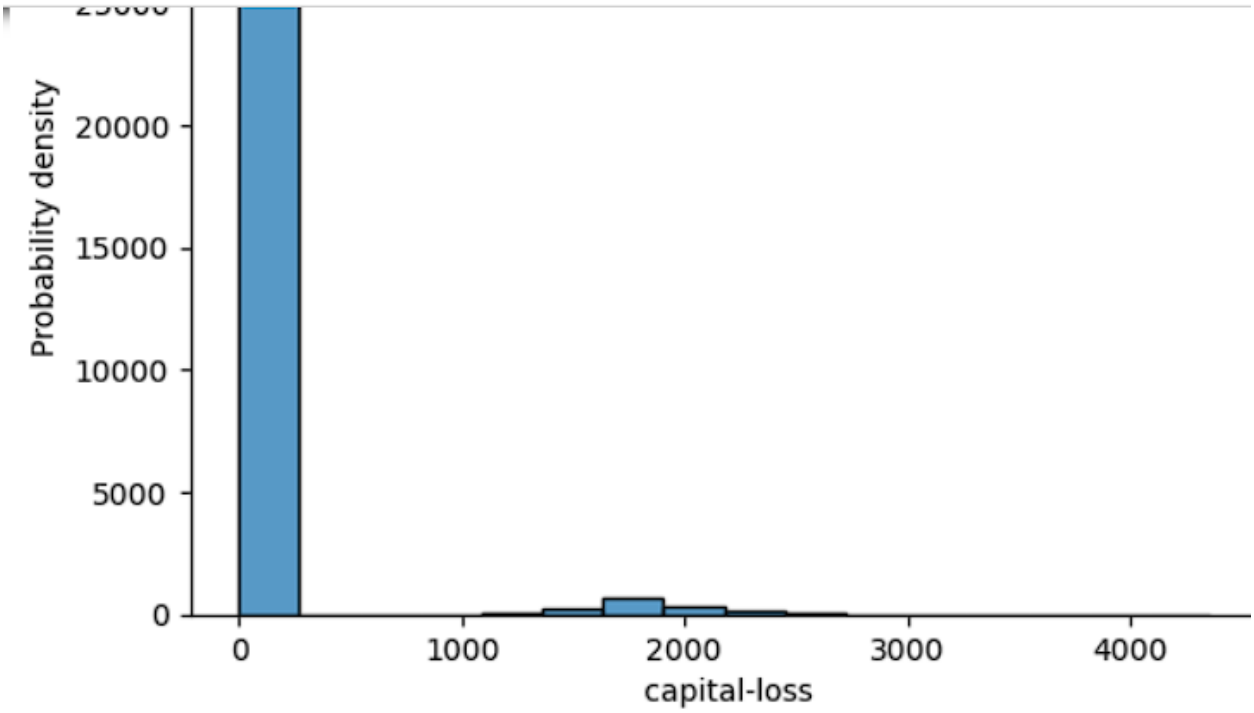
```
# DIsplay the histrogram for each numeric features.
for feature in numerical:
    bar = sns.histplot(df[feature] , kde_kws = {'bw' : 1})
    bar.legend(["Skewness: {:0.2f}".format(df[feature].skew())])
    plt.xlabel(feature)
    plt.ylabel("Probability density")
    plt.title(feature)
    plt.show()
```

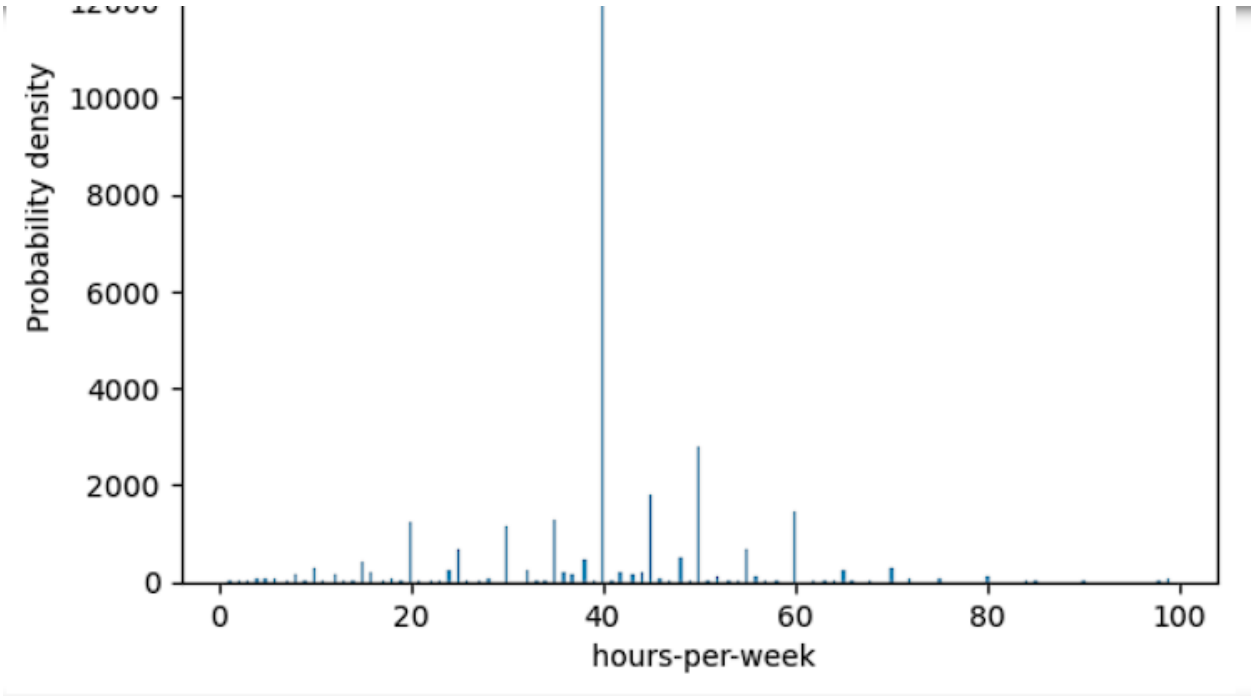

Probability density
1200
1000
800
600
400
200
0
20
30
40
50
60
70
80
90
age


Probability density
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
fnlwgt


Probability density
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16
education-num


Probability density
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000
capital-gain





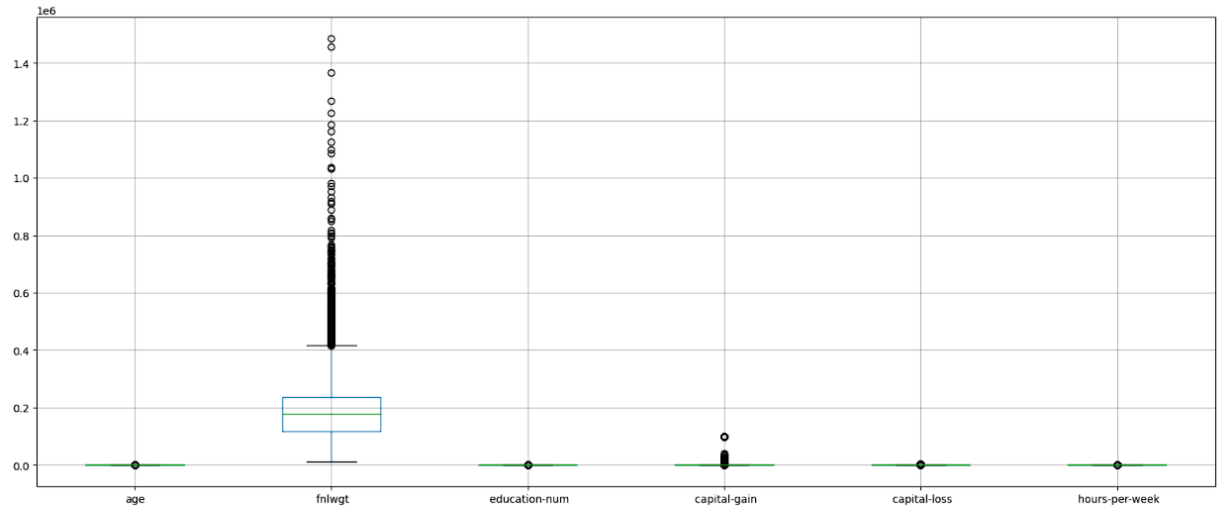
The image is a visual representation of the probability function for each numerical features in the dataset. The data set contain numerical features such as ‘age’,’fnlwgt’,’education-num’,’capital-gain’,’capital-loss ,and ‘hours-per-week’. For each features the image displays a histogram with a

kernel density estimate curve overlaid on top. The x-axis represents the numerical values of the features while the y-axis represents the probability density.

Additionally, the skewness of each feature is displayed in the legend of histogram. Skewness is a measure of the asymmetry of the probability distribution of a real valued random variable about its mean. In this case skewness is represented as a decimal value.

Overall, the image provides a comprehensive visual understanding of the probability density of each numerical features in the dataset, which can be useful for identifying potential outlier or anomalies in the data.

```
#measuring outlier
import matplotlib.pyplot as plt
plt.figure(figsize=(20,8))
df.boxplot()
plt.show()
```



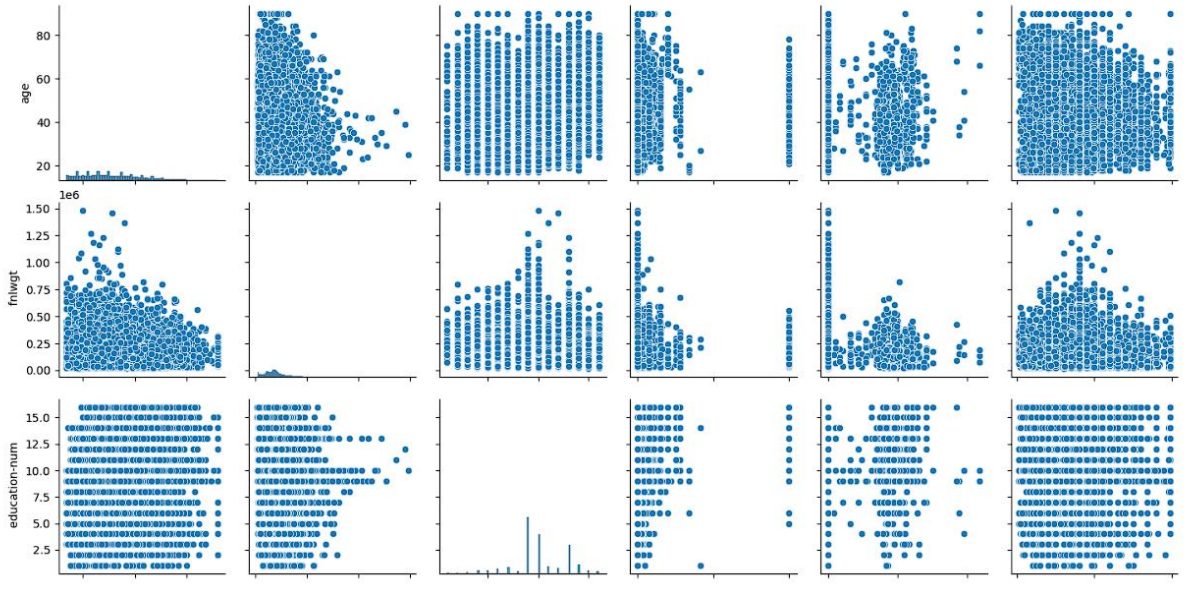
Here I have checked the outlier of the columns using matplotlib. Pyplot. Here we can see the fnlnwgt has the higher upper values 992 and education num also contain some outlier in lower values i.e. 1198. Upper value in capital gain, hours per week has 2712, 1519 respectively. While lower value in hours per week is 5516.

```
#Display the lower and upper outlier values
for i in df:
  if(df[i].dtypes!='O'):
      q1=df[i].quantile(0.25)
      q3=df[i].quantile(0.75)
      iqr=q3-q1
      upper=q3+1.5*iqr
      lower=q1-1.5*iqr
      totalUpper=len(df[df[i]>upper])
      totalLower=len(df[df[i]<lower])
      print(f'Lower values in {i} {totalLower}')
      print(f'Upper values in {i} {totalUpper}')
```

```
Lower values in age 0
Upper values in age 143
Lower values in  fnlwgt 0
Upper values in  fnlwgt 992
Lower values in  education-num 1198
Upper values in  education-num 0
Lower values in  capital-gain 0
Upper values in  capital-gain 2712
Lower values in  capital-loss 0
Upper values in  capital-loss 1519
Lower values in  hours-per-week 5516
Upper values in  hours-per-week 3492
```

```
#Ploting the pair plots of the dataset
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x224bcf5ad40>
```



Handling categorical data

```
#Display the names of all columns in the DataFrame df that have an object or categorical data type.
categorical = [i for i in df.columns if df[i].dtypes == "O"]
```

```
#Display the each categorcial features and unique value
for feature in categorical:
    print(f" {feature}  :  {len(df[feature].unique())}")
```

```
 workclass  :  9
 education  :  16
 marital-status  :  7
 occupation  :  15
 relationship  :  6
 race  :  5
 sex  :  2
 native-country  :  42
 salary  :  2
```

Now we are going to display the categorical data that have an object. We can see work class education martial-status occupation relationship race sex native-country and salary are unique value. We can see education and occupation has the highest number of unique value.

Handling missing values in categorical features

```
# Display the percentage of occurrences of specific values in categorical columns
print(f"workclass : {round(2093 / 32561 , 4) *100}%")
print(f"occupation : {round(1843 / 32561 , 4) *100}%")
print(f"native-country : {round(583 / 32561 , 4) *100}%")
```

```
workclass : 6.43%
occupation : 5.66%
native-country : 1.79%
```

```
df[" occupation"].mode()[0]
```

```
' Prof-specialty'
```

Now we are going to handle the missing value in categorical value. Here we can see work class, occupation native-country have missing value so we are going to fill missing value with mode value.

here we have less than 6 percent missing values so we can fill it with mode value

```
df[" workclass"] = df[' workclass'].str.replace('?', 'Private' )
df[' occupation'] = df[' occupation'].str.replace('?', 'Prof-specialty' )
df[' native-country'] = df[' native-country'].str.replace('?', 'United-States' )
```

```
# education Category
df[" education"].replace(['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th','10th', '11th', '12th'], 'school' ,
                         inplace = True , regex = True)
df[" education"].replace(['Assoc-voc', 'Assoc-acdm', 'Prof-school', 'Some-college'], 'higher' , inplace = True , regex =
```

```
#martial status
df[' marital-status'].replace(['Married-civ-spouse', 'Married-AF-spouse'], 'married' , inplace = True , regex = True)
df[' marital-status'].replace(['Divorced', 'Separated','Widowed',
                                            'Married-spouse-absent'], 'other' , inplace = True , regex = True)
```

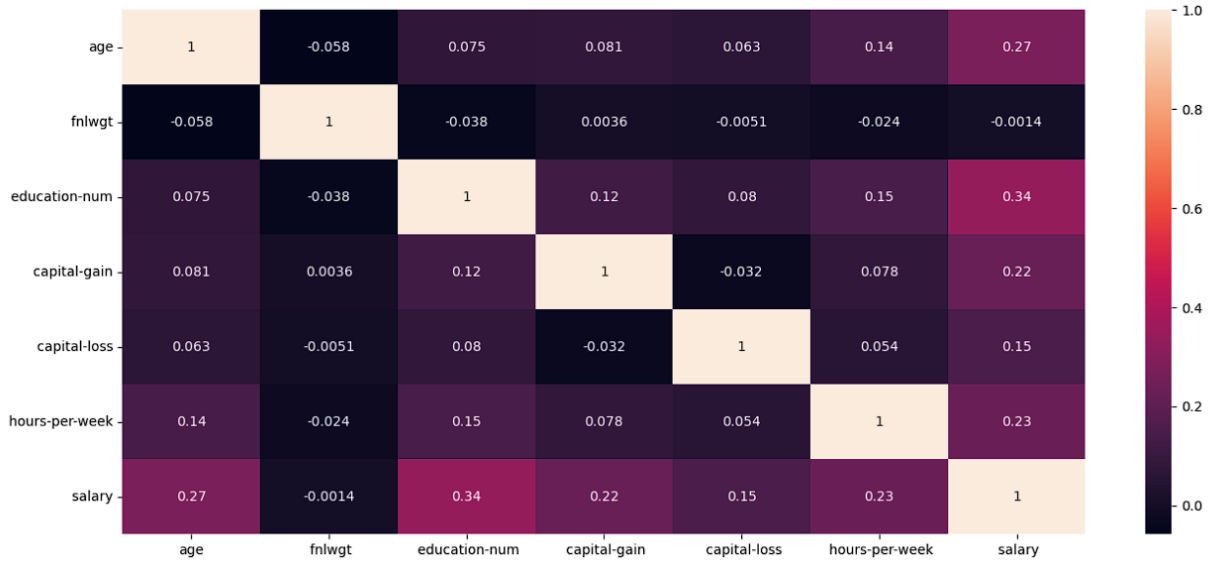
```
# income
df[" salary"] = df[" salary"].replace({'<=50K' : 0 , ">50K" : 1 } , regex = True)
```

```
df.head()
```

So, we have replaced the missing value in categorical data with mode value. As you can see we have replaced the work class with private and occupation with Prof-specialty and for native-country it is replaced with United-States.

For salary we replace the <50k to 0 and >50k to 1

```
# Display the  heatmap using Seaborn to visualize the correlation matrix
plt.figure(figsize = (16 , 7))
sns.heatmap(df.corr(), annot=True);
```



The heatmap shows the correlation between different features in the dataset. From the above heatmap we can say that the education num is positively correlated to salary which indicates higher the education num higher the salary. Also, fnlwgt is negatively correlated to salary which means that the lower the salary higher fnlwgt.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(class_weight="balanced")
lr.fit(X_train, y_train)
prediction = lr.predict(X_test)
accuracy_score(y_test, prediction)
```

```
0.7670904735581352
```

```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.90      0.78      0.84     12435
           1       0.50      0.74      0.60      3846

    accuracy                           0.77     16281
   macro avg       0.70      0.76      0.72     16281
weighted avg       0.81      0.77      0.78     16281
```

As we can see the accuracy score of Logistic regression is 0.77, indicating that model correctly predicted the outcome for 77% of the test data. The classification report analysis the model's performance un further depth. It features precision, recall, f1-score and class suppo0rt. These measures aid in assessing the model's performance in many areas, such as precision, recall, and

f1-score. In this case, models' performance is relatively good, with a high accuracy score and good precision, recall, and f1-score values for both classes.

Random Forest

```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 20, criterion = 'entropy' ,random_state = 51)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.8474295190713101
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.90     12435
           1       0.74      0.55      0.63      3846

    accuracy                           0.85     16281
   macro avg       0.80      0.75      0.77     16281
weighted avg       0.84      0.85      0.84     16281
```

The random forest is a popular machine learning algorithm used for classification tasks. As we can see the random forest classifier is trained on the training data(x-train, y-train) and used to predicate the outcomes for the test data(x-test).

The classification report provides a detailed analysis of the model's performance and accuracy. We can see the accuracy score from random forest is 0.85 which is 85% of the test data. In this case the models performance is relatively good with a high accuracy score.

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
knn_model =KNeighborsClassifier().fit(X_train,y_train)
y_pred = knn_model.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.89     12435
           1       0.67      0.59      0.63      3846

    accuracy                           0.83     16281
   macro avg       0.77      0.75      0.76     16281
weighted avg       0.83      0.83      0.83     16281
```

To categorize the given dataset, the KNN model from sklearn.neighbour was utilized. The model was trained on the training dataset before being used to predict the labels on the test dataset (test). The accuracy of the models predicating was calculated using the sklearn.metrics accuracy score function.

The classification report provides a detailed analysis of the model's performance and accuracy. We can see the accuracy score from K-Nearest-Neighbor is 0.83 which is 83% of the test data. In this case the model's performance is relatively good with a high accuracy score.

Form all the model random forest has the highest accuracy score. So for this protype random forest is the best model

Instructions for code Execution

- **Launch Jupyter application**
- **Then upload the python csv file in the ide**
- **After uploading the file then mount the dataset.**
- **Import all the necessary libraries for the code execution.**
- **Read the dataset, by giving the dataset that will be used for the remainder of the program the variable name.**
- **Everything will be in order at this time to carry out the remaining portions of the program.**