Course : Machine Learning

Course Code : 2CS501

Machine Learning Assignment

*Member 1: 19BCE043*

*Name: Yash Darji*

*Member 2: 19BCE069*

*Name: Ayush Gohil*

Dataset number - 4

Adult Data Set

Dataset Description: Predict whether income exceeds $50K/yr based on census data. Also known as "Census Income" dataset.

Link : [http://archive.ics.uci.edu/ml/datasets/Adult](http://archive.ics.uci.edu/ml/datasets/Adult)

1. Download adult.data from [http://archive.ics.uci.edu/ml/machine-learning-databases/adult/](http://archive.ics.uci.edu/ml/machine-learning-databases/adult/)
2. save as csv file
3. read as csv
4. same steps for adult.test file
5. training dataset in dataframe df
6. test dataset in dataframe df_test
7. importing libraries
8. printing dataset head and information
9. But dataset is not contain any column/attribute information
10. So now we are adding attributes name / column's name
11. Now dataframes are with column and headings
12. now for each attribute we will count number of categories
13. we are counting that how many are married, divorcee, widow all that count by maritial status attribute..
14. And this same for all the attributes
15. Now we are comparing two attributes one by one in compare and checking which one to ignore.
16. We can also skip Gain as we see this graph
17. Loss is also not providing much information here
18. So from all this histplots we came to conclusion that we can exclude Gain, Loss, Final Weight, Country , HPR, and race.
19. We can ignore this fetures. So that we can get clean data.
20. So we are dropping some atributes with low information...

21. drop_columns = ['Gain', 'Loss', 'Final-weight', 'Country', 'Hours-per-week', 'Race']
22. we can remove the rows that contain one or more missing values.
23. Splitting of training and testing data
24. Now we are converting >50k in 1 as output
25. and <=50k in 0 as output as we are predicting binary 1 and 0
26. Still our data is in categorical , integer form
27. So our task is to organize it in numerical form
28. So that we can fit different models to it..
29. So using Onehot encoder and pipeline process we converted our data in well organized form successfully
30. So , Finally our data is ready to fit models to it..
31. We will use following models now one by one:

```
1. KNN Classifier
2. Gaussian Naive Bayes
3. Bernouli Naive Bayes
4. SVC
5. SVC with rbf kernel
6. Decision Tree
7. LogisticRegression
```

32. The Model KNN Has Achieved 83.06 Percent Accuracy
33. The Model Gaussian Naive Bayes Has Achieved 59.75 Percent Accuracy, Which is very low..
34. The Model Bernouli Naive Bayes Has Achieved 76.17 Percent Accuracy
35. The Model SVC Naive Bayes Has Achieved 83.22 Percent Accuracy
36. The Model SVC with kerenel rbf Has Achieved 83.15 Percent Accuracy
37. The Model Decision Tree Classifier Has Achieved 82.44 Percent Accuracy
38. The Model Logistic Regression Has Achieved 82.97 Percent Accuracy
39. So at last we are comparing different models accuracy with bar plot..
40. We successfully predicted the Income of adults, which was our dataset's objective...

```
#importing different python Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import csv
import seaborn as sns
```

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

```
from requests import get
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
```

```
from sklearn import metrics,tree,datasets,svm
from sklearn.metrics import classification_report,confusion_matrix
```

**--------------------------------------------------------------------------------------------------------------------------------**

```
#download adult.data from http://archive.ics.uci.edu/ml/machine-learning-databases/adult/
#save as csv file
#read as csv
df = pd.read_csv('adult.data.csv')
```

```
#download adult.test from http://archive.ics.uci.edu/ml/machine-learning-databases/adult/
#save as csv file
#read as csv
df_test = pd.read_csv('adult.test.csv')
```

```
print(df.shape)
#shape
```

```
(17188, 15)
```

```
print(df_test.shape)
```

```
(16281, 1)
```

**--------------------------------------------------------------------------------------------------------------------------------**

```
df.isna().sum()
```

```
39               0
 State-gov       0
 77516           0
 Bachelors       0
 13              0
```

```
Never-married    0
Adm-clerical     0
Not-in-family    0
White            0
Male             0
2174             0
0                0
40               0
United-States    0
<=50K            1
dtype: int64
```

```
df.head()
```

| | 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 |

```
df_test.head()
```

| | | | | | | | | | | | | | | \|1x3 Cross validator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **25** | **Private** | **226802** | **11th** | **7** | **Never-married** | **Machine-op-inspct** | **Own-child** | **Black** | **Male** | **0** | **0** | **40** | **United-States** | <=50K. |
| **38** | **Private** | **89814** | **HS-grad** | **9** | **Married-civ-spouse** | **Farming-fishing** | **Husband** | **White** | **Male** | **0** | **0** | **50** | **United-States** | <=50K. |
| **28** | **Local-gov** | **336951** | **Assoc-acdm** | **12** | **Married-civ-spouse** | **Protective-serv** | **Husband** | **White** | **Male** | **0** | **0** | **40** | **United-States** | >50K. |
| **44** | **Private** | **160323** | **Some-college** | **10** | **Married-civ-spouse** | **Machine-op-inspct** | **Husband** | **Black** | **Male** | **7688** | **0** | **40** | **United-States** | >50K. |
| **18** | **?** | **103497** | **Some-college** | **10** | **Never-married** | **?** | **Own-child** | **White** | **Female** | **0** | **0** | **30** | **United-States** | <=50K. |

Now we will give column heading to our dataframe

**----------------------------------------------------------------------------------------------------------------------------**

```
#column names missing

columns = ['Age', 'Working class', 'Final-weight', 'Education', 'Education-num', 'Marital-status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Gain', 'Loss', 'Hours-per-week', 'Country', 'Inco
```

Training dataframe column names

```
filename='adult.data.csv'
first_row = False
with open(filename, 'r') as file:
        reader = csv.reader(file, delimiter=',')

        with open('temp.csv', 'w') as temp:
            writer = csv.writer(temp, delimiter=',')

            line_count = 0
            for row in reader:
                if line_count == 0:
                    writer.writerow(columns)
                    if first_row:
                        writer.writerow(row)
                else:
                    writer.writerow(row)
                line_count += 1
os.remove(filename)
os.rename('temp.csv', filename)
```

Testing dataframe column names

```
filename='adult.test.csv'
first_row = False
with open(filename, 'r') as file:
        reader = csv.reader(file, delimiter=',')

        with open('temp.csv', 'w') as temp:
            writer = csv.writer(temp, delimiter=',')

            line_count = 0
            for row in reader:
                if line_count == 0:
                    writer.writerow(columns)
                    if first_row:
                        writer.writerow(row)
                else:
                    writer.writerow(row)
                line_count += 1
os.remove(filename)
os.rename('temp.csv', filename)
```

```
df = pd.read_csv('adult.data.csv')
df_test = pd.read_csv('adult.test.csv')
```

```
df.head()
```

| | Age | Working class | Final-weight | Education | Education-num | Marital-status | Occupation | Relationship | Race | Sex | Gain | Loss | Hours-per-week | Country | Income |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |

```
df_test.head()
```

| | Age | Working class | Final-weight | Education | Education-num | Marital-status | Occupation | Relationship | Race | Sex | Gain | Loss | Hours-per-week | Country | Income |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | Private | 226802 | 11th | 7 | Never-married | Machine-op-inspct | Own-child | Black | Male | 0 | 0 | 40 | United-States | <=50K. |
| **1** | 38 | Private | 89814 | HS-grad | 9 | Married-civ-spouse | Farming-fishing | Husband | White | Male | 0 | 0 | 50 | United-States | <=50K. |
| **2** | 28 | Local-gov | 336951 | Assoc-acdm | 12 | Married-civ-spouse | Protective-serv | Husband | White | Male | 0 | 0 | 40 | United-States | >50K. |
| **3** | 44 | Private | 160323 | Some-college | 10 | Married-civ-spouse | Machine-op-inspct | Husband | Black | Male | 7688 | 0 | 40 | United-States | >50K. |
| **4** | 18 | ? | 103497 | Some-college | 10 | Never-married | ? | Own-child | White | Female | 0 | 0 | 30 | United-States | <=50K. |

Now dataframes are with column and headings

now for each attribute we will count number of categories

```
#Age continues values
df['Working class'].value_counts()
#Final weights also continues values
```

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: Working class, dtype: int64
```

```
df['Education'].value_counts()
```

```
 HS-grad         10501
 Some-college     7291
 Bachelors        5354
```

```
 Masters          1723
 Assoc-voc        1382
 11th             1175
 Assoc-acdm       1067
 10th              933
 7th-8th           646
 Prof-school       576
 9th               514
 12th              433
 Doctorate         413
 5th-6th           333
 1st-4th           168
 Preschool          51
Name: Education, dtype: int64
```

```
df['Education-num'].value_counts()
```

```
9     10501
10     7291
13     5354
14     1723
11     1382
7      1175
12     1067
6       933
4       646
15      576
5       514
8       433
16      413
3       333
2       168
1        51
Name: Education-num, dtype: int64
```

```
df['Marital-status'].value_counts()
```

```
 Married-civ-spouse       14976
 Never-married            10682
 Divorced                  4443
 Separated                 1025
 Widowed                    993
 Married-spouse-absent      418
 Married-AF-spouse           23
Name: Marital-status, dtype: int64
```

```
df['Occupation'].value_counts()
```

```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
```

```
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: Occupation, dtype: int64
```

```python
df['Relationship'].value_counts()
```

```
 Husband           13193
 Not-in-family      8304
 Own-child          5068
 Unmarried          3446
 Wife               1568
 Other-relative      981
Name: Relationship, dtype: int64
```

```python
df['Race'].value_counts()
```

```
 White                 27815
 Black                  3124
 Asian-Pac-Islander     1039
 Amer-Indian-Eskimo      311
 Other                   271
Name: Race, dtype: int64
```

```python
df['Sex'].value_counts()
```

```
 Male      21789
 Female    10771
Name: Sex, dtype: int64
```

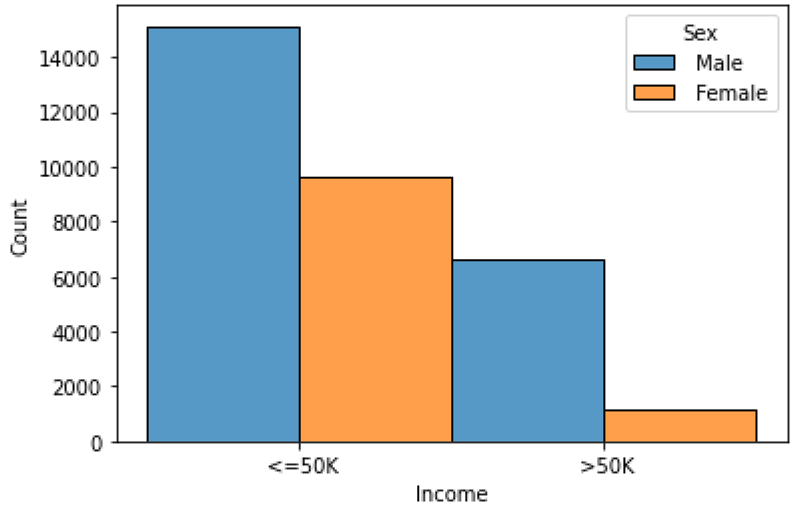
```python
df['Country'].value_counts()
```

```
 United-States                 29169
 Mexico                          643
 ?                               583
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
 Poland                           60
 Columbia                         59
 Taiwan                           51
```

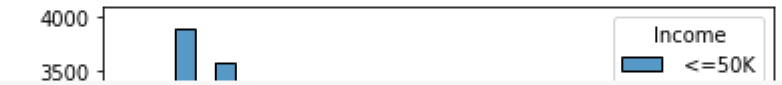
```
    Haiti                             44
    Iran                              43
    Portugal                          37
    Nicaragua                         34
    Peru                              31
    France                            29
    Greece                            29
    Ecuador                           28
    Ireland                           24
    Hong                              20
    Cambodia                          19
    Trinadad&Tobago                   19
    Laos                              18
    Thailand                          18
    Yugoslavia                        16
    Outlying-US(Guam-USVI-etc)        14
    Hungary                           13
    Honduras                          13
    Scotland                          12
    Holand-Netherlands                 1
   Name: Country, dtype: int64
```

Now we are comparing two attributes one by one in compare and checking which one to ignore.

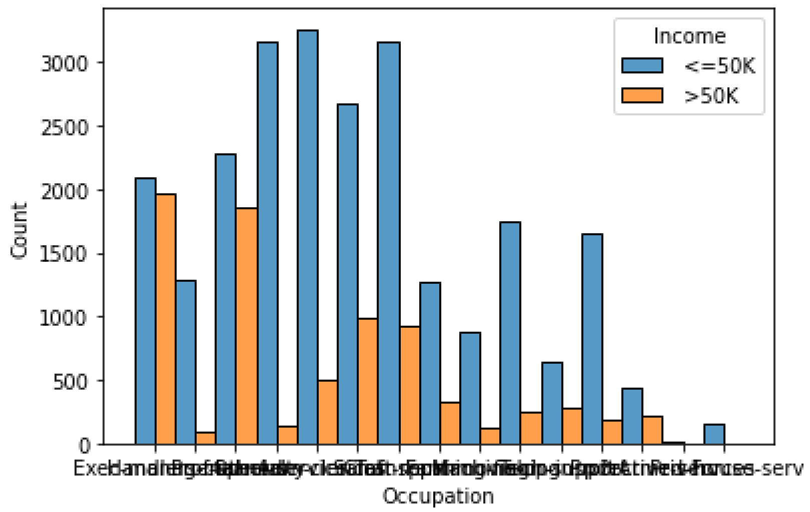
```
sns.histplot(df, x = df['Income'], hue = df['Sex'], multiple = 'dodge');
```



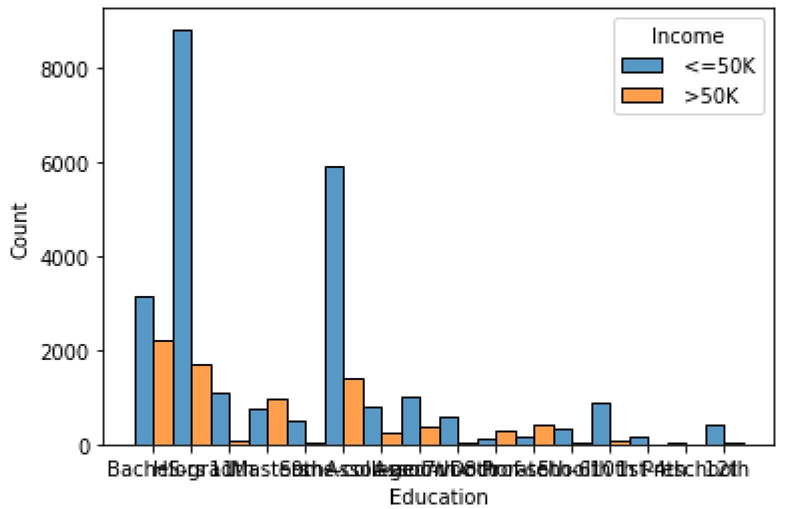

```
sns.histplot(df, x = df['Age'], hue = df['Income'], multiple = 'dodge', binwidth=5);
```



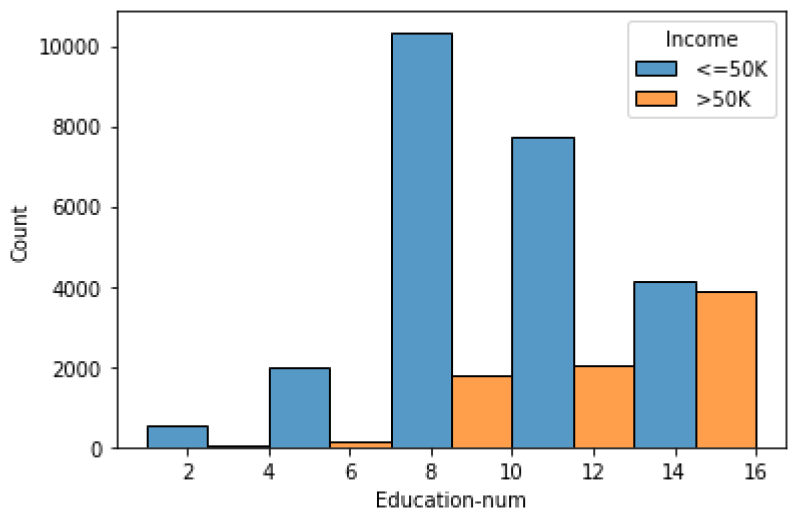

```
sns.histplot(df, x = df['Occupation'], hue = df['Income'], multiple = 'dodge');
```



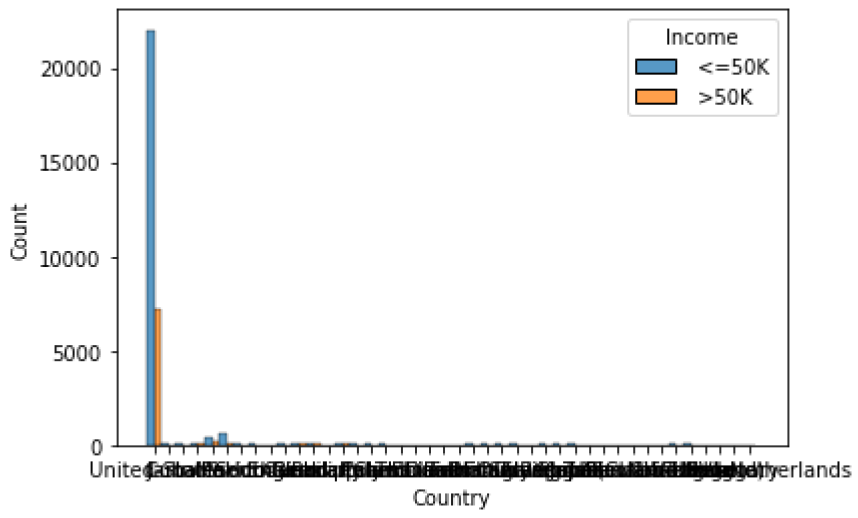

```
sns.histplot(df, x = df['Education'], hue = df['Income'], multiple = 'dodge');
```



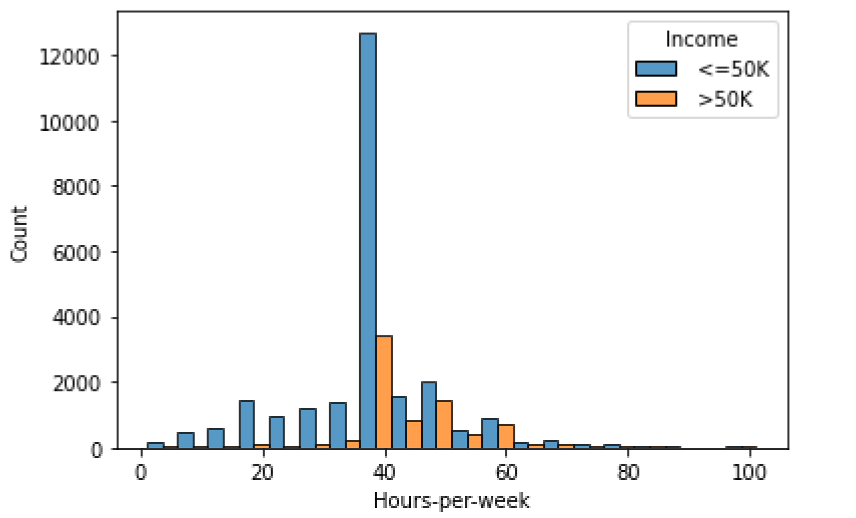

```
sns.histplot(df, x = df['Education-num'], hue = df['Income'], multiple = 'dodge', binwidth=3);
```



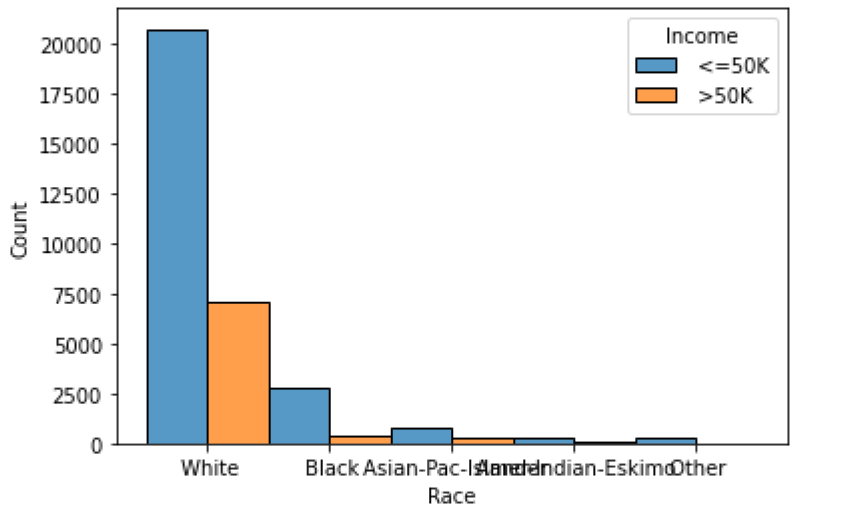
```
sns.histplot(df, x = df['Country'], hue = df['Income'], multiple = 'dodge', binwidth=3);
#we can exclude country attribute
```



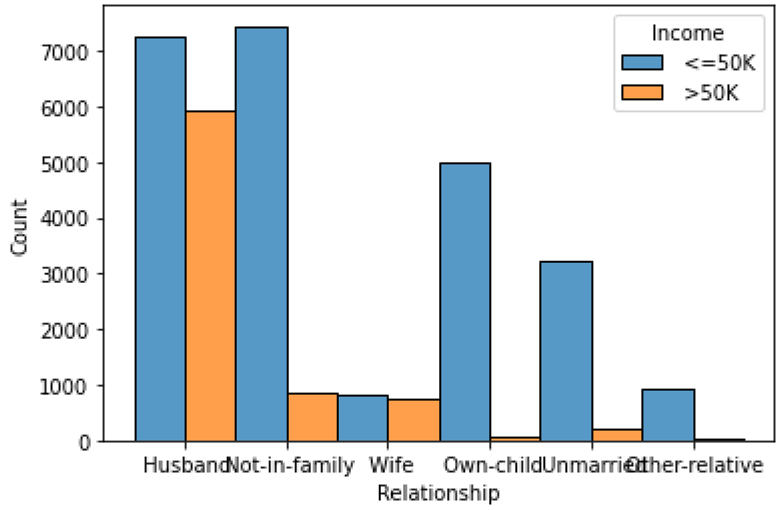

```
sns.histplot(df, x = df['Hours-per-week'], hue = df['Income'], binwidth = 5, multiple = 'dodge');
```



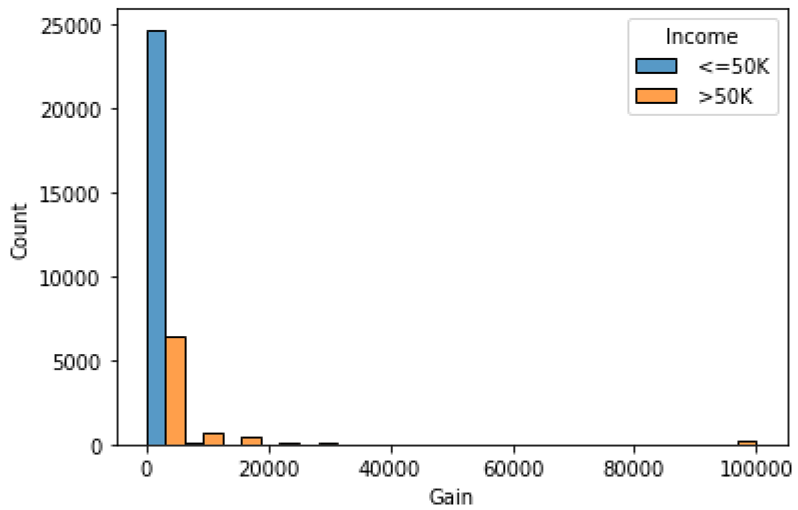

```
sns.histplot(df, x = df['Race'], hue = df['Income'], multiple = 'dodge');
#we can skip Race
```



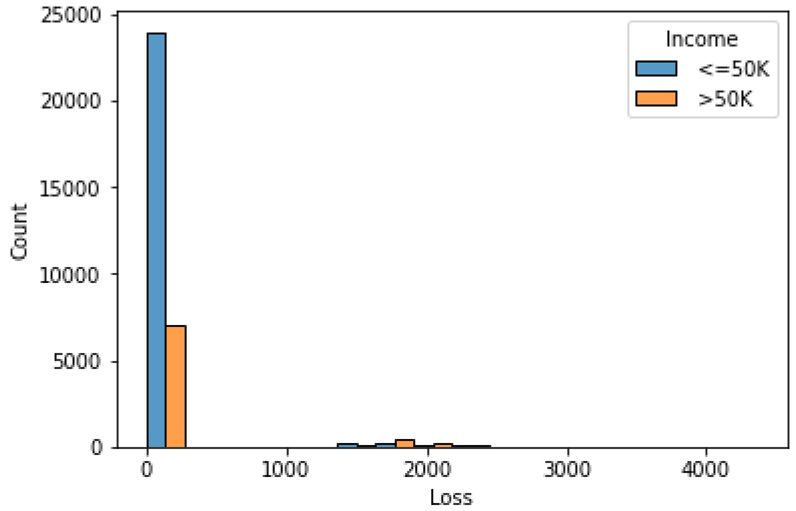
```
sns.histplot(df, x = df['Relationship'], hue = df['Income'], multiple = 'dodge');
```




```
sns.histplot(df, x = df['Gain'], hue = df['Income'], multiple = 'dodge');
#We can also skip Gain as we see this graph
```




```
sns.histplot(df, x = df['Loss'], hue = df['Income'], multiple = 'dodge');
#Loss is not providing much information here
#We can also skip Loss as we see this graph
```



So from all this histplots we came to conclusion that we can exclude Gain, Loss, Final Weight, Country , HPR, and race. We can ignore this fetures. So that we can get clean data.

So we are dropping some atributes with low information...

---

```
drop_columns = ['Gain', 'Loss', 'Final-weight', 'Country', 'Hours-per-week', 'Race']
df.drop(axis = 0, columns = drop_columns, inplace = True)
df_test.drop(columns = drop_columns, inplace = True)
```

```
df.head()
```

| | Age | Working class | Education | Education-num | Marital-status | Occupation | Relationship | Sex | Income |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | Self-emp-not-inc | Bachelors | 13 | Married-civ-spouse | Exec-managerial | Husband | Male | <=50K |
| **1** | 38 | Private | HS-grad | 9 | Divorced | Handlers-cleaners | Not-in-family | Male | <=50K |
| **2** | 53 | Private | 11th | 7 | Married-civ-spouse | Handlers-cleaners | Husband | Male | <=50K |
| **3** | 28 | Private | Bachelors | 13 | Married-civ-spouse | Prof-specialty | Wife | Female | <=50K |
| **4** | 37 | Private | Masters | 14 | Married-civ-spouse | Exec-managerial | Wife | Female | <=50K |

```
df.describe()
```

| | Age | Education-num |
|---|---|---|
| **count** | 32560.000000 | 32560.000000 |
| **mean** | 38.581634 | 10.080590 |
| **std** | 13.640642 | 2.572709 |
| **min** | 17.000000 | 1.000000 |
| **25%** | 28.000000 | 9.000000 |
| **50%** | 37.000000 | 10.000000 |
| **75%** | 48.000000 | 12.000000 |
| **max** | 90.000000 | 16.000000 |

```
df_test.head()
```

| | Age | Working class | Education | Education-num | Marital-status | Occupation | Relationship | Sex | Income |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | Private | 11th | 7 | Never-married | Machine-op-inspct | Own-child | Male | <=50K. |
| **1** | 38 | Private | HS-grad | 9 | Married-civ-spouse | Farming-fishing | Husband | Male | <=50K. |

```
df_test.describe()
```

| | Age | Education-num |
|---|---|---|
| **count** | 16281.000000 | 16281.000000 |
| **mean** | 38.767459 | 10.072907 |
| **std** | 13.849187 | 2.567545 |
| **min** | 17.000000 | 1.000000 |
| **25%** | 28.000000 | 9.000000 |
| **50%** | 37.000000 | 10.000000 |
| **75%** | 48.000000 | 12.000000 |
| **max** | 90.000000 | 16.000000 |

```
print(df.shape)
print(df_test.shape)
```

```
(32560, 9)
(16281, 9)
```

```
#we can remove the rows that contain one or more missing values.
df = df.dropna()
df_test = df_test.dropna()
```

```
print(df.shape)
print(df_test.shape)
#0 rows with missing values here
```

```
(32560, 9)
(16281, 9)
```

Splitting of training and testing data

```
#Splitting of training and testing data
train_data = df.drop(columns = ['Income'])
y_train = df['Income']

test_data = df_test.drop(columns = ['Income'])
y_test = df_test['Income']
```

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16281 entries, 0 to 16280
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             16281 non-null  int64
 1   Working class   16281 non-null  object
 2   Education       16281 non-null  object
 3   Education-num   16281 non-null  int64
 4   Marital-status  16281 non-null  object
 5   Occupation      16281 non-null  object
 6   Relationship    16281 non-null  object
 7   Sex             16281 non-null  object
dtypes: int64(2), object(6)
memory usage: 1017.7+ KB
```

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32560 entries, 0 to 32559
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             32560 non-null  int64
 1   Working class   32560 non-null  object
 2   Education       32560 non-null  object
 3   Education-num   32560 non-null  int64
 4   Marital-status  32560 non-null  object
 5   Occupation      32560 non-null  object
 6   Relationship    32560 non-null  object
 7   Sex             32560 non-null  object
dtypes: int64(2), object(6)
memory usage: 2.2+ MB
```

Now we will convert income column in 1 or 0

```
y_train_n = np.array(y_train)
y_test = np.array(y_test)

y_train_n[y_train_n == ' >50K'] = 1
y_test[y_test == ' >50K.'] = 1

y_train_n[y_train_n == ' <=50K'] = 0
y_test[y_test == ' <=50K.'] = 0

y_train_n = np.array(y_train_n, dtype = np.uint8)
y_test = np.array(y_test, dtype = np.uint8)

y_test
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise compariso
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise compariso
```

```
array([0, 0, 1, ..., 0, 0, 1], dtype=uint8)
```

```
np.count_nonzero(y_train)
```

```
32560
```

Now our data is in categorical and integer form..

So our task is to transform it into organized way so that we can fit different models to it.

Some content of the next portion of code is taken from another resourse for transforming dataset in organized way..

```
class MostFrequentImputer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.mf = {}
        for col in X.columns:
            self.mf[col] = X[col].value_counts().index[0]
            self.data = X
        return self
    def transform(self, X, y=None):
        for col in X.columns:
            X[col] = X[col].replace([' ?'], self.mf[col])
        return X
```

```
Categorical_Pipeline = Pipeline(
    [
        ('Replace "?" values', MostFrequentImputer()),
        ('Encode Values', OneHotEncoder(sparse = False)),
    ]
)


Full_Pipeline = ColumnTransformer(
    [
        ('Numerical Data', StandardScaler(), ['Age', 'Education-num']),
        ('Categorical Data', Categorical_Pipeline, ['Working class', 'Education', 'Marital-status', 'Occupation',
                                                    'Relationship', 'Sex']),
    ]
)
```

```
train_data_processed = Full_Pipeline.fit_transform(train_data)
X_test = Full_Pipeline.fit_transform(test_data)
```

```
y_test
```

```
array([0, 0, 1, ..., 0, 0, 1], dtype=uint8)
```

```
X_train, X_valid, y_train, y_valid = train_test_split(train_data_processed, y_train_n, test_size = 0.4)
```

```
len(X_train), len(X_valid)
```

```
(19536, 13024)
```

So , Finally our data is ready to fit models to it..

```
from sklearn import metrics,tree,datasets,svm
from sklearn.metrics import classification_report,confusion_matrix
```

We will use following models now one by one:

1. *KNN Classifier*

2. *Gaussian Naive Bayes*

3. *Bernouli Naive Bayes*

4. *SVC*

5. *SVC with rbf kernel*

6. *Decision Tree*

7. *LogisticRegression*

## 1. **KNN Classifier**

I have used Grid search for getting best result among different values of hyperparameters

```
# Parameter Space of the KNN Classifier
para = [{'n_neighbors': [5, 7, 9, 10,11, 15,17 ,21,23,25]}]
knn_ = KNeighborsClassifier()
rknn = GridSearchCV(knn_, para)
rknn.fit(X_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': [5, 7, 9, 10, 11, 15, 17, 21, 23,
                                          25]}],
```

```
            pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
            scoring=None, verbose=0)
```

```
rknn.best_estimator_   # Best Hyperparameters
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=25, p=2,
                     weights='uniform')
```

```
model_knn = KNeighborsClassifier(n_neighbors=25)   # Using the best parameters
model_knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=25, p=2,
                     weights='uniform')
```

```
print(model_knn.score(X_train, y_train))

print(model_knn.score(X_valid, y_valid))

knn_accuracy = 100*model_knn.score(X_valid, y_valid)
```

```
0.8429054054054054
0.8306203931203932
```

```
print('The Model KNN Has Achieved %.2f Percent Accuracy'%(knn_accuracy))
```

```
The Model KNN Has Achieved 83.06 Percent Accuracy
```

*The Model KNN Has Achieved 83.06 Percent Accuracy*

## 2. **Gaussian Naive Bayes**

```
model_nb = GaussianNB()
model_nb.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
Gnb_accuracy=100*model_nb.score(X_valid, y_valid)
```

```
print('The Model Gaussian Naive Bayes Has Achieved %.2f Percent Accuracy'%(Gnb_accuracy))
```

```
The Model Gaussian Naive Bayes Has Achieved 59.75 Percent Accuracy
```

*The Model Gaussian Naive Bayes Has Achieved 59.75 Percent Accuracy, Which is very low..*

### 3. **Bernouli Naive Bayes**

```
from sklearn.naive_bayes import BernoulliNB

model_nb = BernoulliNB()
model_nb.fit(X_train, y_train)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

```
model_nb.score(X_valid, y_valid)
```

```
0.7616707616707616
```

```
bnb_accuracy=100*model_nb.score(X_valid, y_valid)
```

```
print('The Model Bernouli Naive Bayes Has Achieved %.2f Percent Accuracy'%(bnb_accuracy))
```

```
The Model Bernouli Naive Bayes Has Achieved 76.17 Percent Accuracy
```

*The Model Bernouli Naive Bayes Has Achieved 76.17 Percent Accuracy*

### 4. **SVC**

```
model=SVC()
model.fit(X_train,y_train)
pred=model.predict(X_valid)
```

```
print('The Model Has Achieved %.2f Percent Accuracy'%(100*metrics.accuracy_score(y_valid,pred)))
SVM_accurancy = (100*metrics.accuracy_score(y_valid,pred))
print(SVM_accurancy)
```

```
The Model Has Achieved 83.22 Percent Accuracy
83.21560196560198
```

*The Model SVC Naive Bayes Has Achieved 83.22 Percent Accuracy*

### 5. **SVC [kernel RBF]**

```
model_sc = SVC(kernel='rbf', gamma = 0.1, C = 1)  # Radial Basis Function for non-linear decision boundary
model_sc.fit(X_train, y_train)
```

```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
model_sc.score(X_valid, y_valid)
```

```
0.831541769041769
```

```
svc_rbf_accuracy=100*model_sc.score(X_valid, y_valid)
```

```
print('The Model SVC with kerenel rbf Has Achieved %.2f Percent Accuracy'%(svc_rbf_accuracy))
```

```
The Model SVC with kerenel rbf Has Achieved 83.15 Percent Accuracy
```

### *The Model SVC with kerenel rbf Has Achieved 83.15 Percent Accuracy*

## __ 6. Decision Tree Classifier__

```
# Parameter Space of the Decision Tree Classifier
param_dt = [
            {'max_depth': [2, 4, 7, 8, 10, 12, 15, 20, 30, 50, 100] },
            {'max_leaf_nodes': [4, 5, 7, 10, 20, 25, 30, 50, 70, 100, 150]},
]

model_dt_temp = DecisionTreeClassifier()
rsdt = GridSearchCV(model_dt_temp, param_dt, cv = 5, n_jobs = -1, return_train_score = True)
rsdt.fit(X_train, y_train)
rsdt.best_estimator_  # Best Hyperparameters
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=50,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
model_dt = DecisionTreeClassifier(criterion='gini', max_leaf_nodes = 100)  # Using the best hyperparameters
model_dt.fit(X_train, y_train)
model_dt.score(X_valid, y_valid)
```

```
0.8244011056511057
```

```
dt_accuracy=100*model_dt.score(X_valid, y_valid)
```

```
print('The Model Decision Tree Classifier Has Achieved %.2f Percent Accuracy'%(dt_accuracy))
```

```
The Model Decision Tree Classifier Has Achieved 82.44 Percent Accuracy
```

*The Model Decision Tree Classifier Has Achieved 82.44 Percent Accuracy*

### 7. Logistic Regression

```
model_lr = LogisticRegression(n_jobs = -1)
model_lr.fit(X_train, y_train)
model_lr.score(X_valid, y_valid)
```

```
0.8296990171990172
```

```
lr_accuracy=100*model_lr.score(X_valid, y_valid)
```

```
print('The Model Logistic Regression Has Achieved %.2f Percent Accuracy'%(lr_accuracy))
```

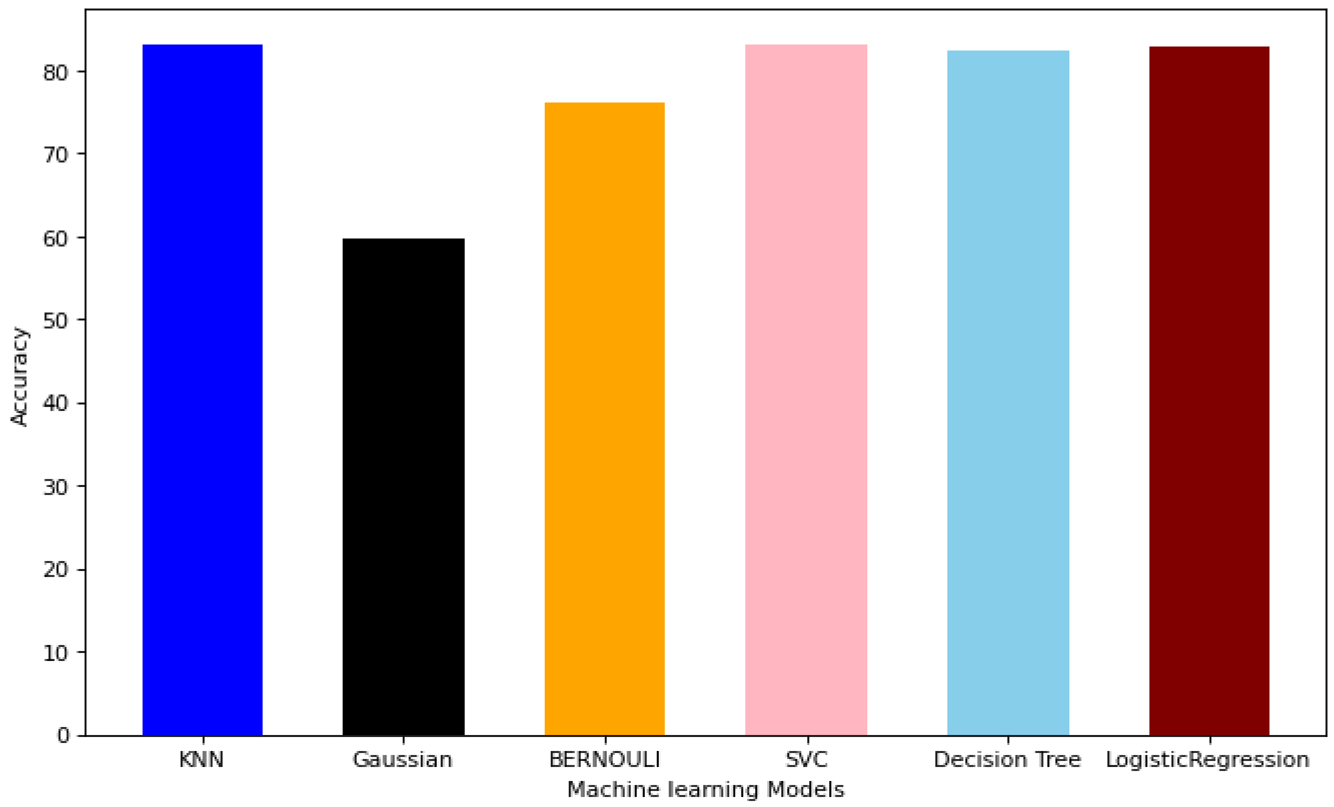
```
The Model Logistic Regression Has Achieved 82.97 Percent Accuracy
```

*The Model Logistic Regression Has Achieved 82.97 Percent Accuracy*

## Comparision of different Model's accuracy

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10,6),dpi=80)
List=[knn_accuracy,Gnb_accuracy,bnb_accuracy,SVM_accurancy,dt_accuracy,lr_accuracy]
l = ['KNN', 'Gaussian', 'BERNOULI','SVC', 'Decision Tree','LogisticRegression']

c=['b','black','orange','lightpink','skyblue','maroon']
plt.bar(l,List, width = 0.6,color=c)
plt.xlabel('Machine learning Models')
plt.ylabel('Accuracy')
plt.show()
```



Accuracy
0
10
20
30
40
50
60
70
80
KNN
Gaussian
BERNOULI
SVC
Decision Tree
LogisticRegression
Machine learning Models