Binary Salary Classification Using Machine Learning

Prepared by:-

Shridhar Somani	20UCS188
Sanskar Pansari	20DCS012
Pulkit Pandey	20UCC081
Pranav Mehta	20UCS140

1. Abstract:

Central governments of every country take a periodic census. Their motive is to make inferences from the population and predict the standard of living, human development index etc. In this project, we will be analyzing such population data.

We will be given a population census dataset. On it, we will apply Machine learning algorithms to predict if the annual income of a person is more or less than $50k. Some useful things can be known if the government knows the income such as his taxation status if he is above or below the poverty line if he is eligible for various government schemes etc.

The annual income of a person depends on various factors/attributes. We have considered 14 attributes. We have a dataset of 48000 people. We will use classification analysis to predict the annual income of a person with given attributes.

This is a supervised learning-based dataset on which we will apply classification algorithms.

2. Dataset used:

The Data Set we Used for the analysis is Census Income Data Set we took this dataset from the website https://archive.ics.uci.edu/.

This training dataset consists of 48842 instances and of 14 Attributes. The features represent the contributing factors to the annual income of a person and the training set is labelled with the income.

Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	396345

Table 1 Dataset used

2.1 Attributes Considered:

1. **Age**: Age of the person
2. **Workclass**: Working class of a person from the following categories- Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3. **Fnlwgt**: Weight associated to each record(not to be used for model training)continuous.
4. **Education:** The type of education a person has taken like Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
5. **Education-num:** Integer indexed to categories of education. Its continuous.
6. **Marital-status:** Whether a person is married or not. It contain categories as follows Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
7. **Occupation:** Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspect, Adm-clerical,

Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

8. **Relationship:** Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

9. **Race:** White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black. People of certain races do only specific jobs that have a particular return.

10. **Sex:** Male are thought to earn more than females. Female/Male.

11. **Capital-gain:** Income from other sources - continuous.

12. **Capital-loss:** Loss due to other income sources - continuous.

13. **Hours-per-week:** A usual trend in comparison of income of people in the same work field is that a person who works more hours, earns more.

14. **Native-country:** 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, Trinidad & Tobago, Peru, Hong, Holland-Netherlands. Blacks, Asians, Hispanics often do blue-collar jobs that have less return whereas Europeans, Indian Americans do white-collar jobs that pay more.

15. **Income:** It is a class attribute classifying the income of the person in two categories namely, “less than equal to 50k” and “greater than equal to 50k”.

3. Preprocessing

3.1 Libraries Required

1.) Pandas Library - It is a data manipulation library in python.It offers data structures and operations for manipulating numerical tables and time series.

2.) Numpy Library - Numpy is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.

3.) Seaborn Library - Seaborn is a data visualization library based on matplotlib used to plot graphs from data.

4.) Scikit-Learn (sklearn) Library - *Scikit*-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in *Python*.

5.) Matplotlib - It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits

3.2 Reading the dataset file:

First, we get the dataset using the pandas ***read_csv*** function. In this dataset, columns were not assigned any Labels , so they were passed as an argument to the read_csv function:

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

    df_train = pd.read_csv('/content/drive/MyDrive/ids_project/adult.data', names=col_names)
    df_test = pd.read_csv('/content/drive/MyDrive/ids_project/adult.test', names=col_names)
```

3.3 Getting count of instances from training and testing data:

To find the number of instances and features present in both the training and test set we use:

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

Output:

```
(32561,15)      #In Training Set
(16282,15)      #In Testing Set
```

3.4 Checking for null values in our dataset:

Using the head function on our testing dataframe, we display the first five rows in our test data:

```
[ ] df_test.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	|1x3 Cross validator	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K
2	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K
3	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
4	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	>50K

Table 2: First five rows in test dataset

Checking for columns with null values in our test dataset:

```
print(df_test.isnull().sum())
```

```
age              0
workclass        1
fnlwgt           1
education        1
education-num    1
marital-status   1
occupation       1
relationship     1
race             1
sex              1
capital-gain     1
capital-loss     1
hours-per-week   1
native-country   1
income           1
dtype: int64
```

As we can see the first row in the test set contains a record full of NaN values,which we don't require, so we have to drop this row.

```
[ ]  df_test.dropna(axis=0, inplace=True)
```

```
[ ]  print(df_test.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
income            0
dtype: int64
```

Null values are removed in the test dataset.

First five rows in our training dataset look like:

```
[ ]  df_train.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Table 3: First five rows in training dataset

Checking for null values in our training dataset:

```
print(df_train.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
income            0
dtype: int64
```

Training dataset has no null values.

3.5 Resetting the indexes:

Now as the row with index 0 has been dropped so we have to reset the indexes for the test set.

```
[ ] df_test.reset_index(drop=True, inplace=True)
    df_test.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K.
1	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K.
2	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K.
3	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	>50K.
4	18	?	103497.0	Some-college	10.0	Never-married	?	Own-child	White	Female	0.0	0.0	30.0	United-States	<=50K.

Table 4: First 5 records of test dataset after resetting index

3.6 Handling missing values:

As we can see in the above table that there are some rows which contains " ?".These are the missing values, so we replace those "?" values with NaN values in both training and test set. This is done so as python can determine those values as missing. Also the test set's class attribute has error in its class labels which were also handled here:

Looking at the unique values of *workclass* and *income* column:

```
[ ]  df_train['workclass'].unique()

     array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
            ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
            ' Never-worked'], dtype=object)
```

```
[ ]  df_test['workclass'].unique()

     array([' Private', ' Local-gov', ' ?', ' Self-emp-not-inc',
            ' Federal-gov', ' State-gov', ' Self-emp-inc', ' Without-pay',
            ' Never-worked'], dtype=object)
```

```
[ ]  df_test['income'].unique()

     array([' <=50K.', ' >50K.'], dtype=object)
```

There are values like '?' in workclass column. Also ' <=50K.' and ' >50K.' need to be changed to ' <=50K' and ' >50K'. We replace '?' with null value.

```
[ ]  df_train.replace(to_replace=' ?', value=np.nan, inplace=True)
     df_test.replace(to_replace=' ?', value=np.nan, inplace=True)
     df_test.replace(to_replace=' <=50K.', value=' <=50K', inplace=True)

     df_test.replace(to_replace=' >50K.', value=' >50K', inplace=True)
```

There are null values in our dataset that also need to be removed.

```
df_train.isnull().sum()
```

```
age                0
workclass       1836
fnlwgt             0
education          0
education-num      0
marital-status     0
occupation      1843
relationship       0
race               0
sex                0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country   583
income             0
dtype: int64
```

Table 5: Showing the sum of no. of null values for each attribute in training set

```
[ ] df_test.isnull().sum()
```

```
age                0
workclass        963
fnlwgt             0
education          0
education-num      0
marital-status     0
occupation       966
relationship       0
race               0
sex                0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country   274
income             0
dtype: int64
```

Table 6: Showing the sum of no. of null values for each attribute in test set.

Dropping null values in both training and testing dataset:

```
[ ]  df_train.dropna(axis=0, inplace=True)
     df_test.dropna(axis=0, inplace=True)
```

3.8 Checking the number of instances in training dataset:

After doing some changes above in the training set now we are checking the number of instances left in the training set.

```
[ ]  df_train.shape

     (30162, 15)
```

4 Random Sampling:

The following command was used to check the distribution of instances between two classes:

```
[ ]  df_train[df_train['income'] == ' <=50K'].describe()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	22654.000000	2.265400e+04	22654.000000	22654.000000	22654.000000	22654.000000
mean	36.608060	1.903386e+05	9.629116	148.893838	53.448000	39.348592
std	13.464631	1.065713e+05	2.413596	936.392280	310.270263	11.950774
min	17.000000	1.376900e+04	1.000000	0.000000	0.000000	1.000000
25%	26.000000	1.173120e+05	9.000000	0.000000	0.000000	38.000000
50%	34.000000	1.795085e+05	9.000000	0.000000	0.000000	40.000000
75%	45.000000	2.394390e+05	10.000000	0.000000	0.000000	40.000000
max	90.000000	1.484705e+06	16.000000	41310.000000	4356.000000	99.000000

Table 7: This is the to check the description of instances.

As we can see here the dataset contains most of the instances for people having income less than or equal to 50K. So for a better model training, we take a random sample from the instances of records of classes of <=50K and concatenate it with the remaining one.

```
[ ] random_sample1 = df_train[df_train.income == ' <=50K'].sample(n = 8000,replace = False,random_state = 0)
    random_sample2 = df_train[df_train.income == ' >50K'].copy()
    new_train = pd.concat([random_sample1,random_sample2])
```

Hence the Problem of imbalance has been resolved. But since the data is concatenated we might shuffle it as well:

```
[ ] new_train = new_train.sample(frac=1).reset_index(drop=True)
    new_test = df_test.copy()
```

The test set was given an alias just for better readability of the code.

```
[ ] new_train.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	34	Private	173730	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	50	United-States	>50K
1	35	Private	275364	Bachelors	13	Divorced	Tech-support	Unmarried	White	Male	7430	0	40	Germany	>50K
2	22	Private	409230	12th	8	Never-married	Transport-moving	Other-relative	White	Male	0	0	40	United-States	<=50K
3	24	Private	52242	HS-grad	9	Married-civ-spouse	Sales	Wife	White	Female	0	0	40	United-States	>50K
4	40	Local-gov	188436	Some-college	10	Married-civ-spouse	Protective-serv	Husband	White	Male	7298	0	40	United-States	>50K

Table 8: This is the first 5 records from the dataset after shuffling.

4.1 Analyzing the data types:

The data type of each attribute can be viewed using ***.dtypes***. As we can see the data contains lot of attributes with data type as object which is the categorical attribute.

```
[ ]  new_train.dtypes
```

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

Table 8: Showing the types of each attribute.

4.1.1 We analyze different types of attributes by separating them as follows:

```
[ ]  num_attributes = new_train.select_dtypes(include=['int64'])
     num_attributes.hist(figsize=(10,10))
```

Plotting histogram of int64 datatype.

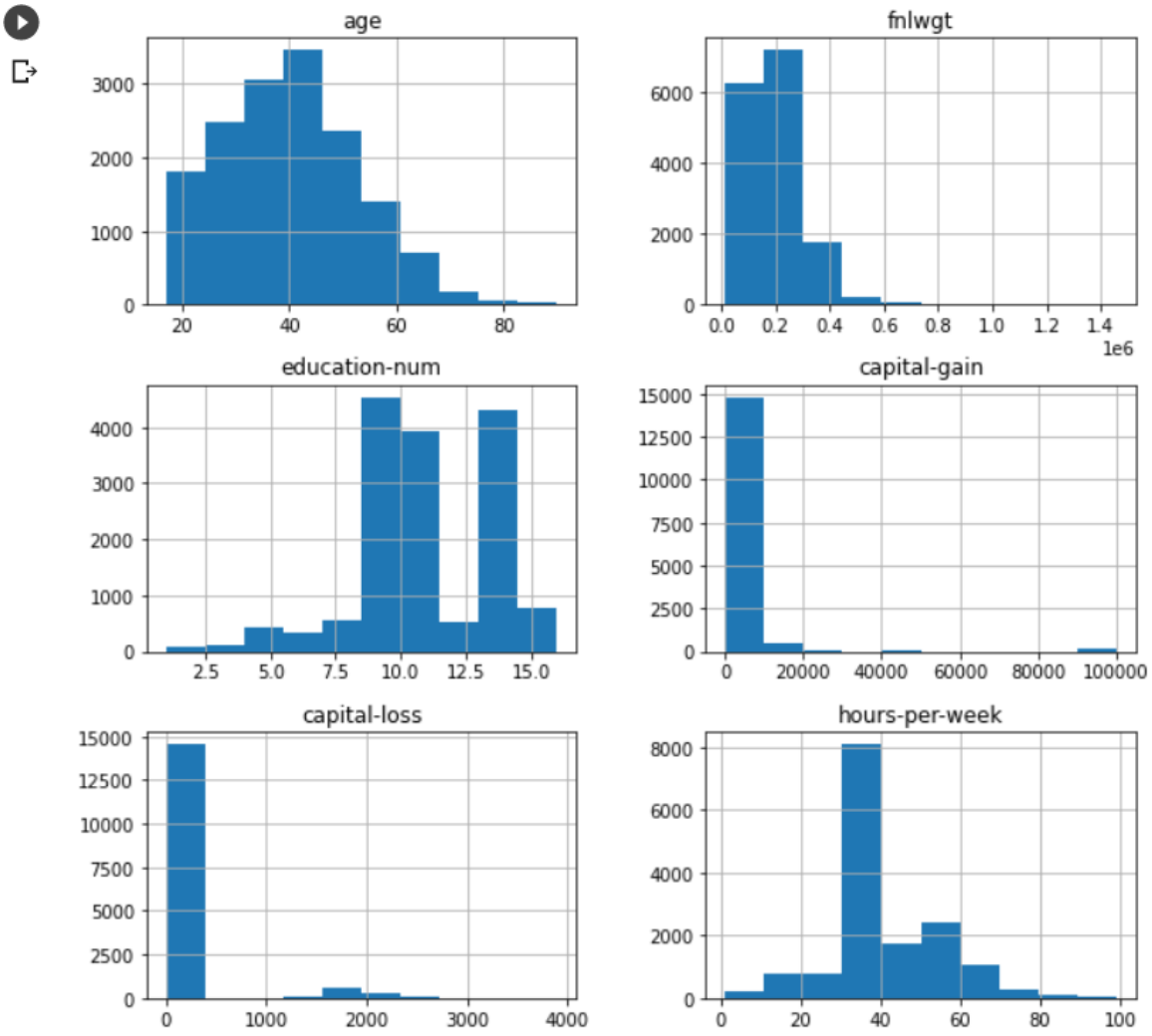


Figure 1: Showing the histogram of int64 data type attributes.

As we can observe from the histograms the data set contains mostly people of age between 20-50, and there aren't many people with capital gain or loss. Also the amount of education - num is mostly between the 8-15.

4.1.2 The Categorical Attributes

Here, we are plotting the annual incomes against the no of years of education.

```
[ ]  cat_attributes = new_train.select_dtypes(include=['object'])
     sns.countplot(y='education', hue='income', data = cat_attributes)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f42a131c590>
```

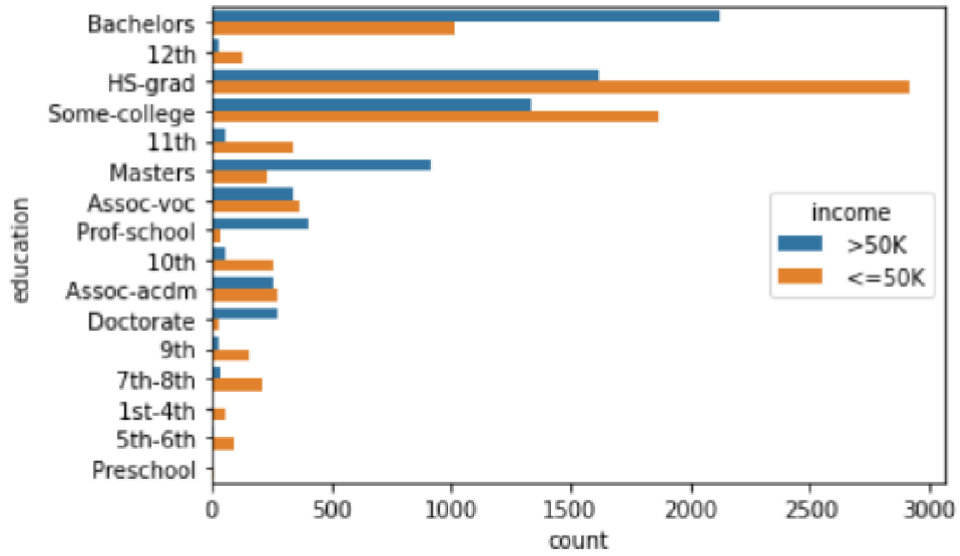


Figure 2: Plot for annual income against the no. of years

As we can deduce from the above graph, people with income >50K are mostly Bachelors and that with <=50K are mostly HS-graduates.

4.2 Encoding the Categorical Attributes:

The categorical attributes need to be mapped with certain numerical numbers so that we can train a model on it. Also the Test Set was encoded too, so that predictions can be made.

This was done using the ***LabelEncoder*** class in the sklearn.

```
from sklearn.preprocessing import LabelEncoder
labels = list(cat_attributes.columns)
for i in labels:
    le = LabelEncoder()
    le.fit(new_train[i])
    new_train[i] = le.transform(new_train[i])
    new_test[i] = le.transform(new_test[i])

new_train.head()
```

The mapped values are as follows for the Training and Test Set respectively.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	34	2	173730	9	13	2	3	0	4	1	0	0	50	37	1
1	35	2	275364	9	13	0	12	4	4	1	7430	0	40	10	1
2	22	2	409230	2	8	4	13	2	4	1	0	0	40	37	0
3	24	2	52242	11	9	2	11	5	4	0	0	0	40	37	1
4	40	1	188436	15	10	2	10	0	4	1	7298	0	40	37	1

Table 9: First 5 records from the dataset after encoding in training set.

```
[31] new_test.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	2	226802.0	1	7.0	4	6	3	2	1	0.0	0.0	40.0	37	0
1	38	2	89814.0	11	9.0	2	4	0	4	1	0.0	0.0	50.0	37	0
2	28	1	336951.0	7	12.0	2	10	0	4	1	0.0	0.0	40.0	37	1
3	44	2	160323.0	15	10.0	2	6	0	2	1	7688.0	0.0	40.0	37	1
5	34	2	198693.0	0	6.0	4	7	1	4	1	0.0	0.0	30.0	37	0

Table 10: First 5 records from the dataset after encoding in test set.

4.3 Splitting the training and testing set:

The Training and Test set were split into two parts each to separate the attributes from the class attribute. But before that we plot a heatmap of the attributes so that we can find the correlation between them and if any are highly correlated we can drop them so as to increase the accuracy and efficiency of our model:

```
X_train = new_train.drop(['income','fnlwgt','education'], axis =1)
Y_train = new_train['income']
X_test = new_test.drop(['income','fnlwgt','education'], axis =1)
Y_test = new_test['income']
```

The education column was dropped as we already had a column containing the numerical value associated with education of a person. The fnlwgt column was also dropped as it was related to the weight of each record.

```
sns.heatmap(new_train.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba03cdb8d0>
```

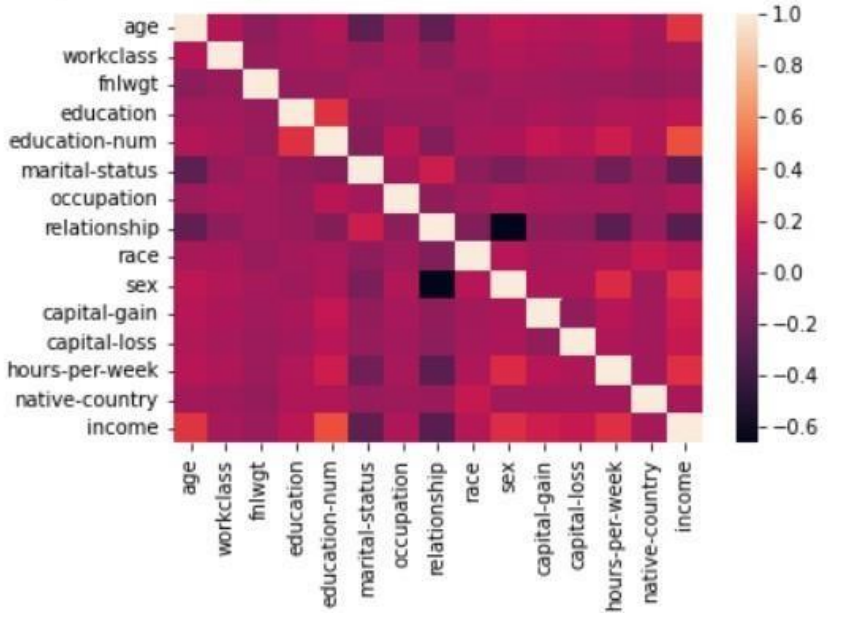


Figure 3: Heatmap of correlation between different features

The Heatmap clearly shows that attributes are not highly correlated to each other so we cannot drop any other attribute.

5. Model Training:

5.1 Applying Random Forest:

Random forest is a collection of decision trees which return mean of all the decision tree's accuracy as the prediction. It is also known as a bagging method. We chose this model because decision trees are good at handling categorical data and random forest further enhances the capabilities of Decision Trees.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
model_1 = RandomForestClassifier(n_estimators=100,bootstrap=True,random_state=0)
model_1.fit(X_train,Y_train)

pred_randfor = model_1.predict(X_test)
print(accuracy_score(pred_randfor, Y_test.values))
```

```
0.8079017264276228
```

After applying random forest model, we get 80.7% (approx) accuracy.

Confusion Matrix:

Confusion matrix here is used to know how many people were predicted to earn more than 50k vs how many actually earn and vice versa. We use the matplotlib and seaborn library to plot the confusion matrix as follows:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
cfm = confusion_matrix(pred_randfor, Y_test.values)
sns.heatmap(cfm, annot=True)
plt.xlabel('Predicted classes')
plt.ylabel('Actual classes')
```

```
Text(33.0, 0.5, 'Actual classes')
```

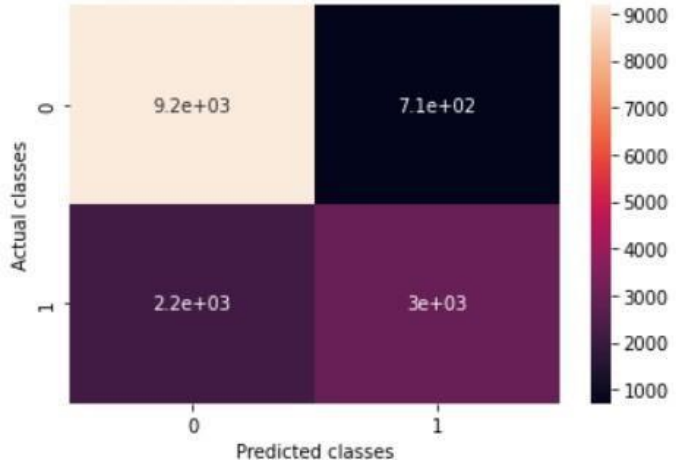


Figure 4: Confusion matrix for Actual classes and Predicted classes

To get the exact numerals of the values in confusion matrix we do the following:

```
[36] confusion_matrix(pred_randfor, Y_test.values)
```

```
array([[9176,  709],
       [2184, 2991]])
```

5.2 Applying Cross-Validation:

```
[37] from sklearn.model_selection import cross_val_score
     scores = cross_val_score(model_1, X_train,
              Y_train, cv=5)
     print(np.mean(scores))
```

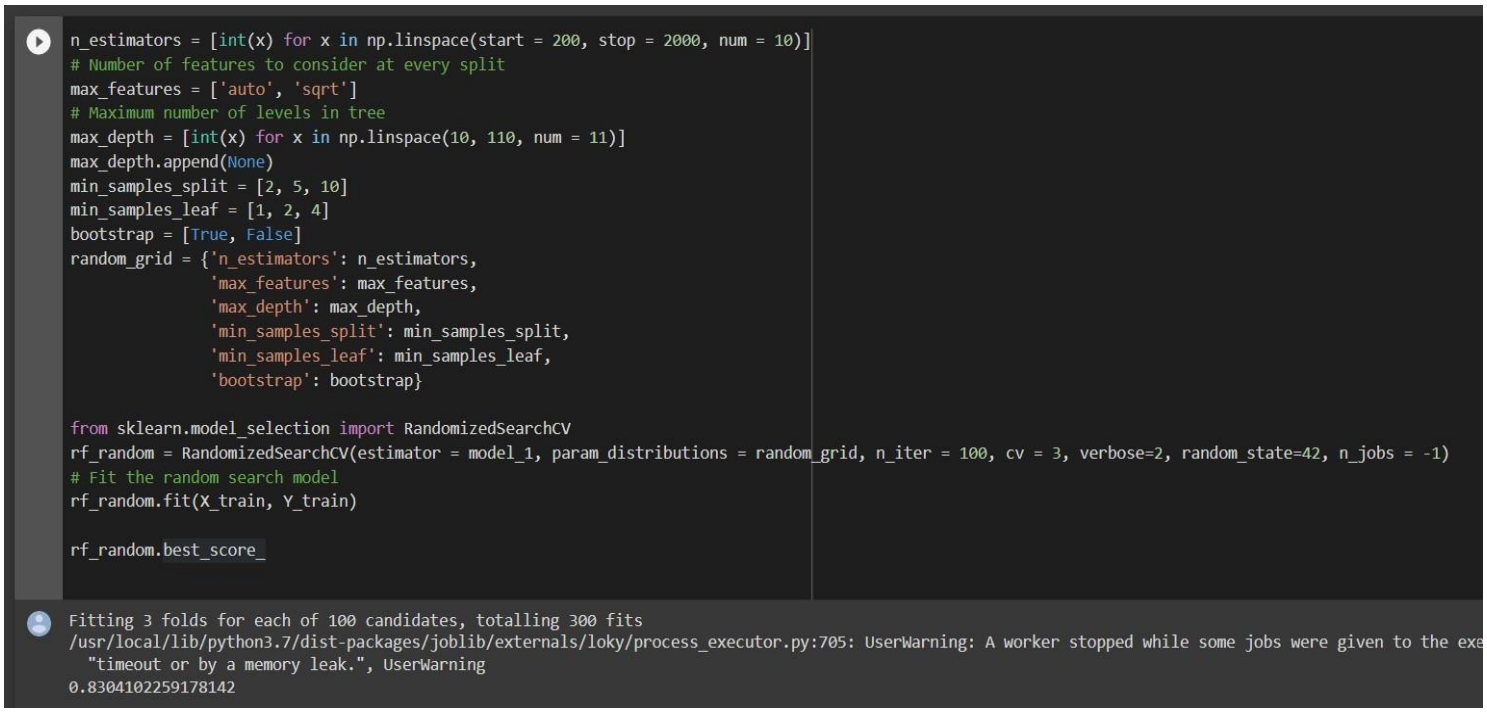
```
0.8168688538939726
```

Here we are using cross-validation technique to train and test the model by dividing the dataset. We will divide the dataset into 5 parts and use one part for testing and the other 4 parts for training. Cross-validation is better than train_test_split() or holdout method because here we have more data for training than in the holdout method.
We then get the mean of all these 5 possibilities.

5.3 Applying Random Search:

Random Search is applied to get the hyperparameters to be used to increase the accuracy of our model. It tries to apply different parameter combinations to our model to train on the provided dataset and returns the parameters that give the best accuracy.

Usually, we only have a vague idea of the best hyperparameters and thus the best approach to narrow our search is to evaluate a wide range of values for each hyperparameter. Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold CV with each combination of values.
Randomized search is the algorithm to select best hyperparameter given in random grid for maximum accuracy.



```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

from sklearn.model_selection import RandomizedSearchCV
rf_random = RandomizedSearchCV(estimator = model_1, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train, Y_train)

rf_random.best_score_
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
/usr/local/lib/python3.7/dist-packages/joblib/externals/loky/process_executor.py:705: UserWarning: A worker stopped while some jobs were given to the exe
  "timeout or by a memory leak.", UserWarning
0.8304102259178142
```

Output=0.8304102259178142
After applying random forest model, we get 83% (approx) accuracy.

6. Inferences

The data set was imbalanced so we performed the Random sampling to make a balanced data set on which our model would train. We plot the heat map of the attributes and found that no two attributes were highly correlated hence we couldn't perform any Feature selection.

Since the data set included a lot of categorical data and Decision trees are good at handling it, so we trained our model based on Decision Tree classifier and to getter better results, we used bagging of Decision Trees known as Random Forest. Initially, we got an accuracy of 80.7% and then used cross-validation to improve the accuracy which led us to an increment of 1%. It is due to the fact that cross-validation trains the model again and again on the same dataset using different pairs of X_train and Y_train.

Then to further improve our accuracy we used Random Search which trains our model on different combinations of Hyperparameters and gives us the best accuracy that could be achieved using the best combination of Hyperparameters. This resulted in achieving an accuracy of about 83%.

So at last, we were able to classify the salaries of the test data and the classification accuracy was 83% which was achieved through Random Search.

The complete code to the project can be found here.