Project name : Adult census income prediction

In [1]:

```
# import libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import imblearn
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearch
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import roc_curve, roc_auc_score, plot_confusion_matrix
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
```

problem statement

The Goal is to predict whether a person has an income of more than 50K a year or not.This is basically a binary classification problem where a person is classified into the

>50K group or <=50K group.

```
    Over the last two decades, humans have grown a lot of dependence on data and
information in society and with this advent growth, technologies have evolved for
their storage,analysis and processing on a huge scale. The fields of Data Mining
and Machine Learning have not only exploited them for knowledge and discovery but
also to explore certain hidden patterns and concepts which led to the prediction
of future events, not easy to obtain.
    The problem of income inequality has been of great concern in the recent
years. Making the poor better off does not seem to be the sole criteria to be
inquest for eradicating this issue. People of the United States believe that the
advent of economic inequality is unacceptable and demands a fair share of wealth
in the society. This model actually aims to conduct a comprehensive analysis to
highlight the key factors that are necessary in improving an individual's income.
Such an analysis helps to set focus on the important areas which can significantly
improve the income levels of individuals.
```

In [2]:

```
!python --version
```

```
Python 3.10.0
```

Data Gathering

In [3]:

```
df = pd.read_csv('adult.csv')
df
```

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	r
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	W
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	W
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	W
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Bl
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Bl
...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	W
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	W
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	W
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	W
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	W

32561 rows × 15 columns

Exploratory data analysis

In [4]:

```
1 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  country         32561 non-null  object
 14  salary          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [5]:

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

Out[5]:

```
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
country           0
salary            0
dtype: int64
```

Here,we see that dataset didn't consist null values, so we didn't clean up our data according to missing values or null values

In [6]:

```
1 df.describe()
```

Out[6]:

	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

In [7]:

```
1 df.dtypes
```

Out[7]:

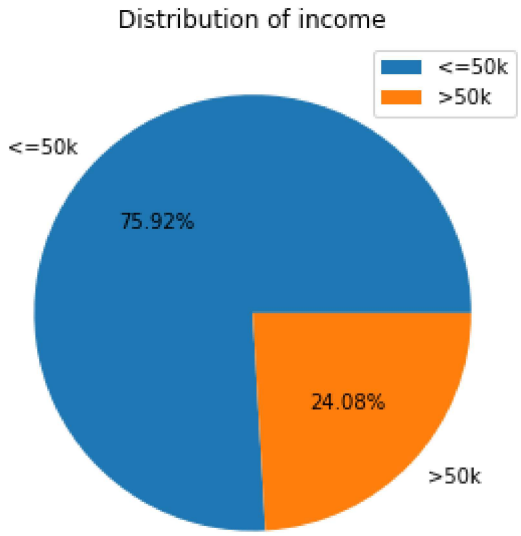
```
age              int64
workclass       object
fnlwgt           int64
education       object
education-num    int64
marital-status  object
occupation      object
relationship    object
race            object
sex             object
capital-gain     int64
capital-loss     int64
hours-per-week   int64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

The target column in our dataset is binary, so we apply classification algorithm on dataset. but some of the input or independent variables are in the form of object data type, so we have to do feature engineering on our dataset so that We need to convert all variables to the same datatype for classification.. Also, We can observe that education-num and education are the same variables with different data types. We can drop any one of them.

i) To check the income proportion in dataset (<=50k and >50k)

In [8]:

```
# pie chart for income prediction

df1 = plt.subplots(figsize=(8,5))
plt.pie(df['salary'].value_counts(),labels=['<=50k','>50k'],autopct='%.2f%%')
plt.legend()
plt.title('Distribution of income')
plt.show()
```

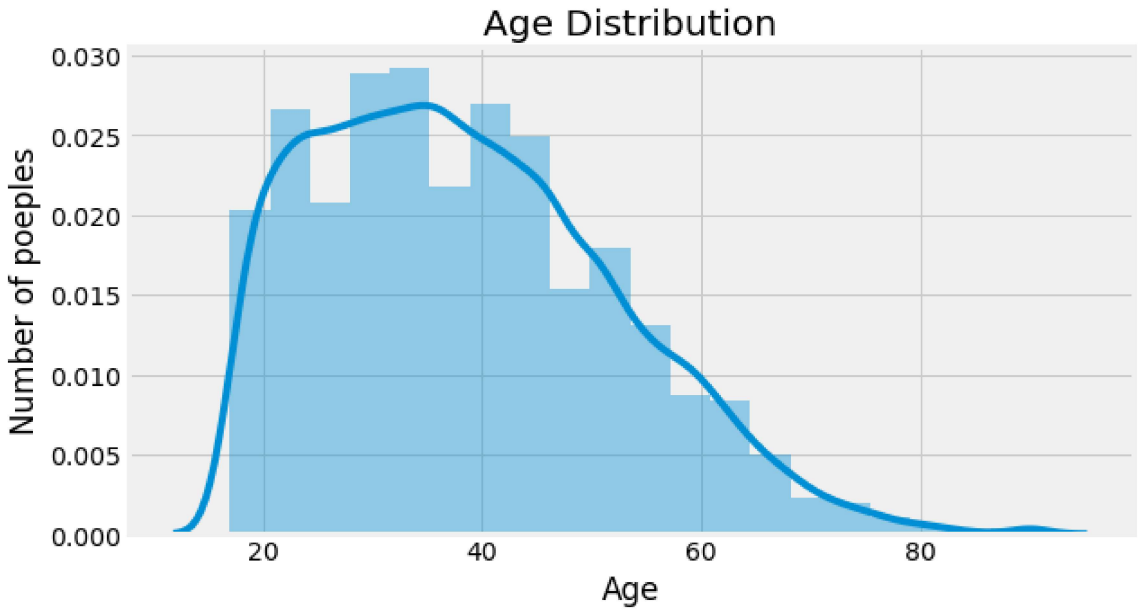


From above diagram we clearly conclude that the income is not distributed properly.

ii) To create a distribution plot for 'AGE'

In [9]:

```
age = df['age'].value_counts()
plt.figure(figsize=(10,5))
plt.style.use('fivethirtyeight')
sns.distplot(df['age'],bins=20)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Number of poeples')
plt.show()
```



Sample population contain more number of people in the age group of 20-40

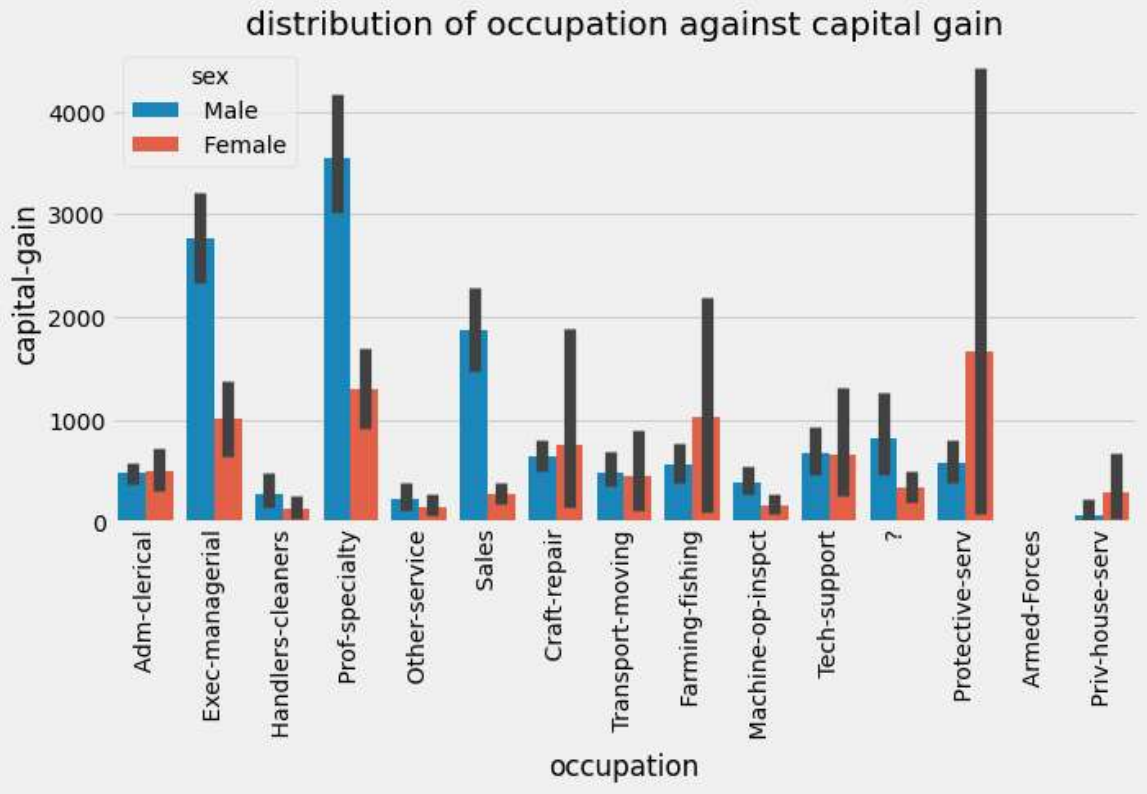
iii) To check the distribution of capital-gain and occupation as per gender

In [10]:

```
plt.figure(figsize=(10,5))
sns.barplot(x='occupation',y='capital-gain',data=df,hue='sex')
plt.xticks(rotation=90)
plt.title('distribution of occupation against capital gain')
```

Out[10]:

```
Text(0.5, 1.0, 'distribution of occupation against capital gain')
```



Capital gain is totally depend on occupation rather than income

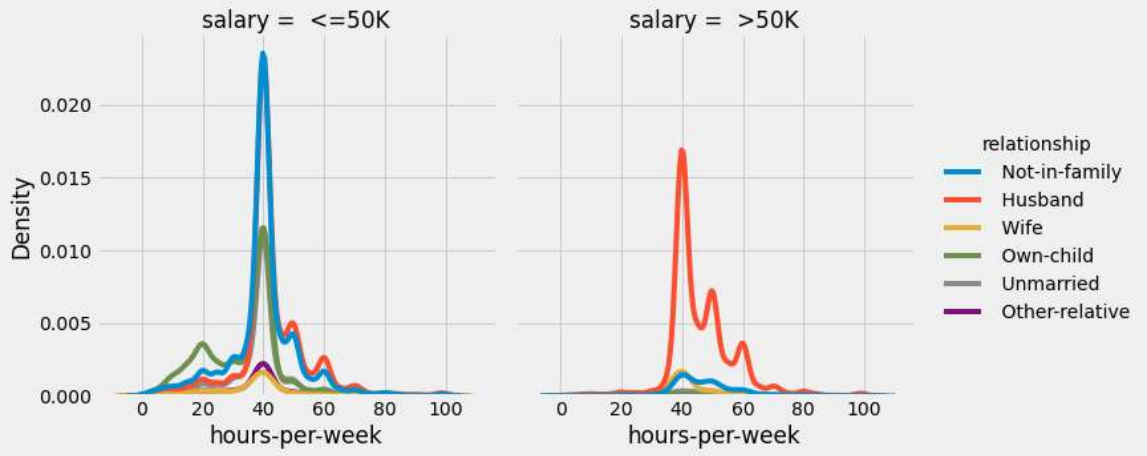
iv) To create a distribution plot of hours-per-week, relationship and salary

In [11]:

```
1 sns.displot(data=df, x=df["hours-per-week"], hue=df["relationship"], col=df["salary"
```

Out[11]:

```
<seaborn.axisgrid.FacetGrid at 0x15c3dc18eb0>
```



peoples who are not in the family are worked for 40 hours and their salary is <=50k.

People who have a relationship of husband work for around 40 hours per week and recieve a salary of >50 k.

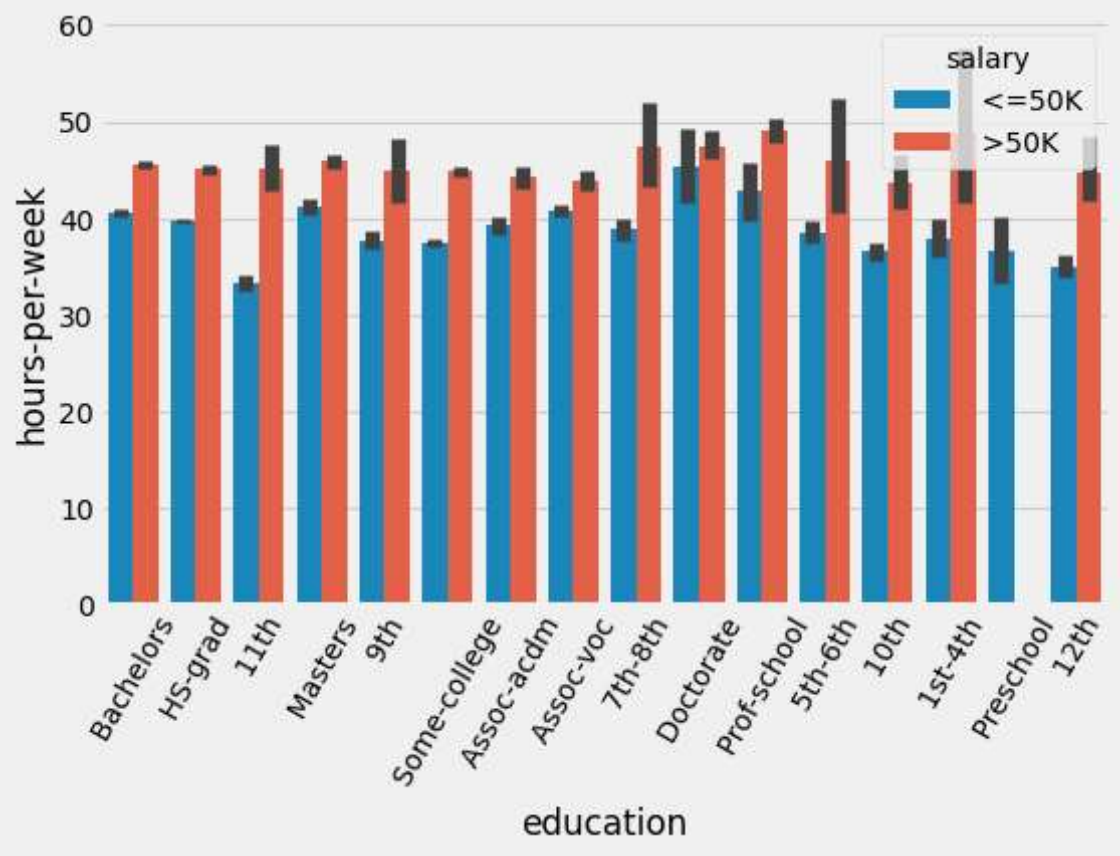
v) To create barplot of hours-per-week, education and salary

In [12]:

```
plt.figure(figsize=(8,5))
sns.barplot(x='education',y='hours-per-week',data=df,hue='salary')
plt.xticks(rotation=60)
```

Out[12]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, ' Bachelors'),
  Text(1, 0, ' HS-grad'),
  Text(2, 0, ' 11th'),
  Text(3, 0, ' Masters'),
  Text(4, 0, ' 9th'),
  Text(5, 0, ' Some-college'),
  Text(6, 0, ' Assoc-acdm'),
  Text(7, 0, ' Assoc-voc'),
  Text(8, 0, ' 7th-8th'),
  Text(9, 0, ' Doctorate'),
  Text(10, 0, ' Prof-school'),
  Text(11, 0, ' 5th-6th'),
  Text(12, 0, ' 10th'),
  Text(13, 0, ' 1st-4th'),
  Text(14, 0, ' Preschool'),
  Text(15, 0, ' 12th')])
```



Here we clearly see that the people with more hours of work has more salary

Feature Engineering

In [13]:

```
for dataset in [df]:
    dataset.loc[dataset['country'] != ' United-States', 'country'] = 0
    dataset.loc[dataset['country'] == ' United-States', 'country'] = 1
    dataset.loc[dataset['race'] != ' White', 'race'] = 0
    dataset.loc[dataset['race'] == ' White', 'race'] = 1
    dataset.loc[dataset['workclass'] != ' Private', 'workclass'] = 0
    dataset.loc[dataset['workclass'] == ' Private', 'workclass'] = 1
    dataset.loc[dataset['hours-per-week'] <= 40, 'hours-per-week'] = 0
    dataset.loc[dataset['hours-per-week'] > 40, 'hours-per-week'] = 1
for col in df[df.columns]:
    if df[col].dtypes == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
df = df.astype(int)
df=df.drop(["education"],axis=1)
df.head()
```

Out[13]:

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	sex	capital-gain
0	39	0	77516	13	4	1	1	1	1	2174
1	50	0	83311	13	2	4	0	1	1	0
2	38	1	215646	9	0	6	1	1	1	0
3	53	1	234721	7	2	6	0	0	1	0
4	28	1	338409	13	2	10	5	0	0	0

train_test_split

In [14]:

```
x = df.drop('salary',axis=1)
y = df['salary']
```

In [15]:

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

Out[15]:

```
0    24720
1     7841
Name: salary, dtype: int64
```

As we see here, our data is imbalanced.Imbalanced data is a common problem in machine learning, which brings challenges to feature correlation, class separation and evaluation, and results in poor model performance.Classes that make up a large proportion of the data set are called majority

classes. Those that make up a smaller proportion are minority classes. so by using randomoversampler method we make our data balanced.

In [16]:

```
random_sampler = RandomOverSampler(random_state=30,sampling_strategy=0.95)
random_sampler.fit(x,y)
```

Out[16]:

```
RandomOverSampler(random_state=30, sampling_strategy=0.95)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [17]:

```
x_new,y_new = random_sampler.fit_resample(x, y)
```

In [18]:

```
y_new.value_counts()
```

Out[18]:

```
0    24720
1    23484
Name: salary, dtype: int64
```

In [19]:

```
x_new_train, x_new_test, y_new_train,y_new_test = train_test_split(x_new,y_new,test_
x_new_train.shape, x_new_test.shape, y_new_train.shape, y_new_test.shape
```

Out[19]:

```
((33742, 13), (14462, 13), (33742,), (14462,))
```

Model training

Random_forest

```
random forest doesn't make any assumption about the data or its distribution.
Hence it generally requires minimal data transformations. Random forest algorithm
makes use of random subsets of features and hence it can perform quite well with a
high dimensional dataset (a dataset with a large number of features).
--------------------------------------------------------------------------------
While building a machine learning model, our aim is to generalize the model
properly for giving predictions on unseen data.
```

The problem of overfitting takes place when we have a flexible model. A flexible model is having high variance because the learned parameters like the structure of the decision tree, etc will vary with the training data. On the contrary, an inflexible model is said to have a high bias as it makes assumptions about the training data and an inflexible model may not have the capacity to fit even the training data and in both situations, the model has high variance, and high bias implies the model is not able to generalize new and unseen data points properly. So, we have to build a model carefully by keeping the bias-variance tradeoff in mind.

The main reason for the overfitting of the decision tree due to not put the limit on the maximum depth of the tree is because it has unlimited flexibility, which means it keeps growing unless, for every single observation, there is one leaf node present.

Moreover, instead of limiting the depth of the tree which results in reduced variance and an increase in bias, we can combine many decision trees that eventually convert into a forest, known as a single ensemble model (known as the random forest).

The steps that are included while performing the random forest algorithm are as follows:

Step-1: Pick K random records from the dataset having a total of N records.

Step-2: Build and train a decision tree model on these K records.

Step-3: Choose the number of trees you want in your algorithm and repeat steps 1 and 2.

Step-4: In the case of a regression problem, for an unseen data point, each tree in the forest predicts a value for output. The final value can be calculated by taking the mean or average of all the values predicted by all the trees in the forest.

and, in the case of a classification problem, each tree in the forest predicts the class to which the new data point belongs. Finally, the new data point is assigned to the class that has the maximum votes among them i.e, wins the majority vote.

---

n_estimators: We know that a random forest is nothing but a group of many decision trees, the n_estimator parameter controls the number of trees inside the classifier. We may think that using many trees to fit a model will help us to get a more generalized result, but this is not always the case. However, it will not cause any overfitting but can certainly increase the time complexity of the model. The default number of estimators is 100 in scikit-learn.

max_depth: It governs the maximum height upto which the trees inside the forest can grow. It is one of the most important hyperparameters when it comes to increasing the accuracy of the model, as we increase the depth of the tree the model accuracy increases upto a certain limit but then it will start to decrease gradually because of overfitting in the model. It is important to set its value appropriately to avoid overfitting. The default value is set to None, None specifies that the nodes inside the tree will continue to grow until all leaves become pure or all leaves contain less than

min_samples_split (another hyperparameter).

```
min_samples_split: It specifies the minimum amount of samples an internal node must hold in order to split into further nodes. If we have a very low value of min_samples_splits then, in this case, our tree will continue to grow and start overfitting. By increasing the value of min_samples_splits we can decrease the total number of splits thus limiting the number of parameters in the model and thus can aid in reducing the overfitting in the model. However, the value should not be kept very large that a number of parameters drop extremely causing the model to underfit. We generally keep

min_samples_split value between 2 and 6. However, the default value is set to 2.
min_samples_leaf:  It specifies the minimum amount of samples that a node must hold after getting split. It also helps to reduce overfitting when we have ample amount of parameters. Less number of parameters can lead to overfitting also, we should keep in mind that increasing the value to a large number can lead to less number of parameters and in this case model can underfit also. The default value is set to 1.

max_features: Random forest takes random subsets of features and tries to find the best split.  max_features helps to find the number of features to take into account in order to make the best split. It can take four values “auto“, “sqrt“, “log2” and None.
In case of auto: considers max_features = sqrt(n_features)
In case of sqrt: considers max_features = sqrt(n_features), it is same as auto
In case of log2: considers max_features = log2(n_features)
In case of None: considers max_features = n_features

max_leaf_nodes: It sets a limit on the splitting of the node and thus helps to reduce the depth of the tree, and effectively helps in reducing overfitting. If the value is set to None, the tree continues to grow infinitely.

max_samples: This hyperparameter helps to choose maximum number of samples from the training dataset to train each individual tree.
--------------------------------------------------------------------------------
Advantages:

-- Random Forest is unbiased as we train multiple decision trees and each tree is trained on a subset of the same training data.
-- It is very stable since if we introduce the new data points in the dataset, then it does not affect much as the new data point impacts one tree, and is pretty hard to impact all the trees.
-- Also, it works well when you have both categorical and numerical features in the problem statement.
-- It performs very well, with missing values in the dataset.
--------------------------------------------------------------------------------
Real life applications of Random Forests :

-- Fraud detection for bank accounts, credit card.

-- Detect and predict the drug sensitivity of a medicine.

-- Identify a patient's disease by analyzing their medical records.

-- Predict estimated loss or profit while purchasing a particular stock.
```

In [20]:

```
rf_model = RandomForestClassifier(random_state=15)
rf_model.fit(x_new_train,y_new_train)
```

Out[20]:

```
RandomForestClassifier(random_state=15)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [21]:

```
y_pred = rf_model.predict(x_new_test)
```

Model evaluation

In [22]:

```
# Training data Evalution
def model_eval(model,x,y):
    y_pred = model.predict(x)

    cm = confusion_matrix(y, y_pred)
    print(f"Confusion Matrix = \n{cm}")
    print("*"*50)

    ac = accuracy_score(y, y_pred)
    print(f"Accuracy Score = {ac}")
    print("*"*50)

    cr = classification_report(y, y_pred)
    print(f"Classification Report = \n{cr}")

    pr = precision_score(y,y_pred)
    print(f'precision_score = {pr}')

    rc = recall_score(y,y_pred)
    print(f'recall_score = {rc}')

    f1 = f1_score(y,y_pred)
    print(f'f1_score = {f1}')

    return "Model Eval Success"
```

```
>> Evaluation matrix :

Confusion Matrix :
    - It is a N * N matrix used to evaluate performance of Classification model.
    - It compares actual values and predicted values

|-----------------------------------------------|
|           |                  Actual Values     |
|-----------|------------------------------------|
|           |            | Positive |   Negative |
```

```
|Predicted  | Positive   |    TP    |       FP   |
|Values     | Negative   |    FN    |       TN   |
|-----------|------------|----------|------------|

    ex. Positive, Negative
    - True Positive (TP):
        -Actual values is Positive and predicted as Positive.
        -the pred(Positive) matches with actual (Positive) value.

    - True Negative (TN):
        -Actual values is Negative and predicted as Negative.
        -the pred(Negative) matches with actual (Negative) value.

    - False Positive (FP): Type-1 Error
        -Actual values is Negative and predicted as Positive.
        -the pred(Positive) not matches with actual (Negative) value.
        -the pred value is falsely predicted.

    - False Negative (FN): Type-2 Error
        -Actual values is Positive and predicted as Negative.
        -the pred(Negative) not matches with actual (Positive) value.
        -the pred value is falsely predicted.

--------------------------------------------------------------------------------
----------

We Define classification report using Precision,Recall,F1-Score and Support

1.Precision:

    Precision = TP/TP+FP

    We will focus on FP
    It ranges from 0 to 1

    Out of total predicted positive results(TP+FP) how many are actually positive
(TP)

    When FP is important then Precision is the appropriate Metric
    When money is on stake, Stock market, e-commerce, Finance

2.Recall:

    Recall = TP/TP+FN

    We will focus on FN
    It ranges from 0 to 1

    Out of total actual positive results(TP+FN) how many are predicted positive
(TP)

    When FN is important then Recall is the appropriate Metric
    When lives are on stake, Medical, Pharma

3.F1-Score:
    f1-score = 2PR/P+R

    It is harmonic mean of Precision and Recall
    When Precision is low and Recall is high or vice-versa, its difficult to
compare
    in such cases f1-score is a good metric to use.
```

```
    It strikes a balance between Precision and Recall.

4.Support: It is the total entries of each class in the actual dataset. It is simply the sum of rows for every class

5. Macro and weighted average of precision and recall:

   -Precision Weighted avg= precision class0*(Support*Total)+Precision class1*(Support*Total)
   -Recall Weighted avg= Recall class0*(Support*Total)+Recall class1*(Support*Total)
```

In [23]:

```
model_eval(rf_model,x_new_train,y_new_train)
```

```
Confusion Matrix =
[[17302     2]
 [    0 16438]]
**************************************************
Accuracy Score = 0.9999407266907712
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17304
           1       1.00      1.00      1.00     16438

    accuracy                           1.00     33742
   macro avg       1.00      1.00      1.00     33742
weighted avg       1.00      1.00      1.00     33742

precision_score = 0.9998783454987834
recall_score = 1.0
f1_score = 0.9999391690492122
```

Out[23]:

```
'Model Eval Success'
```

In [24]:

```
model_eval(rf_model,x_new_test,y_new_test)
```

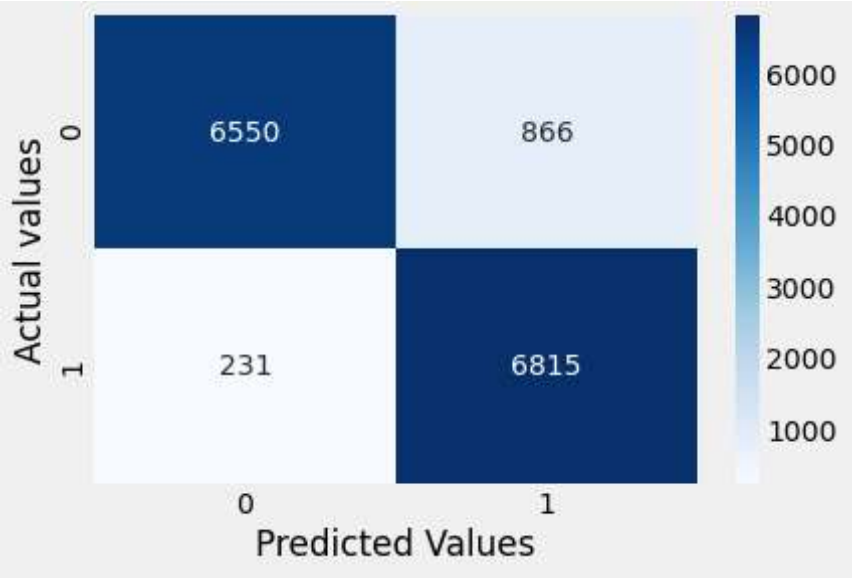
```
Confusion Matrix =
[[6550  866]
 [ 231 6815]]
**************************************************
Accuracy Score = 0.9241460378924077
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.97      0.88      0.92      7416
           1       0.89      0.97      0.93      7046

    accuracy                           0.92     14462
   macro avg       0.93      0.93      0.92     14462
weighted avg       0.93      0.92      0.92     14462

precision_score = 0.8872542637677385
recall_score = 0.9672154413851831
f1_score = 0.9255109662524613
```

Out[24]:

```
'Model Eval Success'
```

In [25]:

```
cm = confusion_matrix(y_new_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm,annot=True,fmt='d',cmap="Blues")
plt.xlabel("Predicted Values")
plt.ylabel("Actual values")
plt.show()
```



In random forest algorithm, it consist low bias and low variance, so it is generalised model. it doesn't need of hyperparamter tuning

In [26]:

```
model_eval_test = pd.DataFrame({'recall':[recall_score(y_new_test,y_pred)],'precisic
model_eval_test
```

Out[26]:

	recall	precision	f1_score	accuracy
0	0.967215	0.887254	0.925511	0.924146

Feature_selection

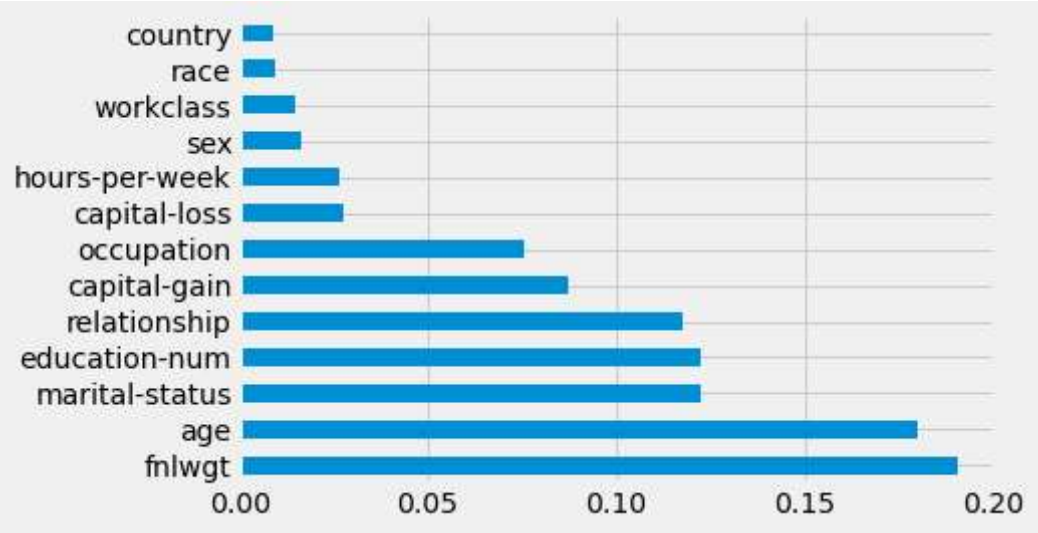
Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data. It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve.

In [27]:

```
s1 = pd.Series(rf_model.feature_importances_, index = x_new.columns)
s1.sort_values(ascending=False).plot(kind="barh")
```

Out[27]:

```
<AxesSubplot:>
```



As we see the diagram, every input variable has their own impact on dataset, so we didn't ignore or neglect any of the variable. we will work by including all the variables.

Decision_tree

```
A decision tree is a type of supervised learning algorithm that is commonly used
in machine learning to model and predict outcomes based on input data. It is a
tree-like structure where each internal node represents a decision or test on a
specific feature or attribute, each branch represents the outcome of that
decision, and each leaf node represents the final decision or prediction.

-- Example of a decision tree
```

Suppose we want to build a decision tree to predict whether a person is likely to buy a new car based on their demographic and behavior data. The decision tree starts with the root node, which represents the entire dataset. The root node splits the dataset based on the “income” attribute. If the person’s income is less than or equal to $50,000, the decision tree follows the left branch, and if the income is greater than $50,000, the decision tree follows the right branch.

The left branch leads to a node that represents the “age” attribute. If the person’s age is less than or equal to 30, the decision tree follows the left branch, and if the age is greater than 30, the decision tree follows the right branch. The right branch leads to a leaf node that predicts that the person is unlikely to buy a new car.

The left branch leads to another node that represents the “education” attribute. If the person’s education level is less than or equal to high school, the decision tree follows the left branch, and if the education level is greater than high school, the decision tree follows the right branch. The left branch leads to a leaf node that predicts that the person is unlikely to buy a new car. The right branch leads to another node that represents the “credit score” attribute. If the person’s credit score is less than or equal to 650, the decision tree follows the left branch, and if the credit score is greater than 650, the decision tree follows the right branch. The left branch leads to a leaf node that predicts that the person is unlikely to buy a new car. The right branch leads to a leaf node that predicts that the person is likely to buy a new car.

In summary, a decision tree is a graphical representation of all the possible outcomes of a decision based on the input data. It is a powerful tool for modeling and predicting outcomes in a wide range of domains, including business, finance, healthcare, and more.

-- Below are some assumptions that we made while using the decision tree:

i) At the beginning, we consider the whole training set as the root.
ii) Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
iii) On the basis of attribute values, records are distributed recursively.
iv) iWe use statistical methods for ordering attributes as root or the internal node.

In the Decision Tree, the major challenge is the identification of the attribute for the root node at each level. This process is known as attribute selection. We have two popular attribute selection measures:

i) Information Gain
ii) Gini Index

-- 1. Information Gain :

When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy. Definition: Suppose S is a set of instances, A is an attribute, Sv is the subset of S with A = v, and Values (A) is the set of all possible values of A, then Gain(S, A) = Entropy(S) - \sum_{v \epsilon Values(A)}\frac{\left | S_{v} \right |}{\left | S \right |}. Entropy(S_{v}) Entropy Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content. Definition: Suppose S is a set of instances, A is an attribute, Sv is the subset of S with A = v, and Values (A) is the set of all possible values of A, then

-- Information_gain = E(S) - sum(weighted_average * entropy of each feature)

```
30
31 -- Building Decision Tree using Information Gain The essentials:
32
33 i) Start with all training instances associated with the root node
34 ii) Use info gain to choose which attribute to label each node with
35 iii) Note: No root-to-leaf path should contain the same discrete attribute twice
36 iv) Recursively construct each subtree on the subset of training instances that
   would be classified down that path in the tree.
37 v) If all positive or all negative training instances remain, the label that node
   “yes” or “no” accordingly
38 vi) If no attributes remain, label with a majority vote of training instances left
   at that node
39 vii) If no instances remain, label with a majority vote of the parent’s training
   instances.
40
41 -- Gini Index :
42
43 i) Gini Index is a metric to measure how often a randomly chosen element would be
   incorrectly identified.
44 ii) It means an attribute with a lower Gini index should be preferred.
45 iii) Sklearn supports “Gini” criteria for Gini Index and by default, it takes
   “gini” value.
46 iv) The Formula for the calculation of the Gini Index is given below.
47 v) The Gini Index is a measure of the inequality or impurity of a distribution,
   commonly used in decision trees and other machine learning algorithms. It ranges
   from 0 to 1, where 0 represents perfect equality (all values are the same) and 1
   represents perfect inequality (all values are different).
48
49 -- Some additional features and characteristics of the Gini Index are:
50
51 i) It is calculated by summing the squared probabilities of each outcome in a
   distribution and subtracting the result from 1.
52 ii) A lower Gini Index indicates a more homogeneous or pure distribution, while a
   higher Gini Index indicates a more heterogeneous or impure distribution.
53 iii) In decision trees, the Gini Index is used to evaluate the quality of a split
   by measuring the difference between the impurity of the parent node and the
   weighted impurity of the child nodes.
54 Compared to other impurity measures like entropy, the Gini Index is faster to
   compute and more sensitive to changes in class probabilities.
55 iv) One disadvantage of the Gini Index is that it tends to favor splits that
   create equally sized child nodes, even if they are not optimal for classification
   accuracy.
56 v) In practice, the choice between using the Gini Index or other impurity measures
   depends on the specific problem and dataset, and often requires experimentation
   and tuning
```

In [28]:

```
tree = DecisionTreeClassifier(criterion='entropy',max_depth= 6,min_samples_leaf= 9,m
tree.fit(x_new_train,y_new_train)
```

Out[28]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=6, min_samples_leaf=
9,
                       min_samples_split=7)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [29]:

```
y_pred = tree.predict(x_new_test)
```

In [30]:

```
print(model_eval(tree,x_new_train,y_new_train))
```

```
Confusion Matrix =
[[13617  3687]
 [ 2716 13722]]
**************************************************
Accuracy Score = 0.8102365005038231
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.83      0.79      0.81     17304
           1       0.79      0.83      0.81     16438

    accuracy                           0.81     33742
   macro avg       0.81      0.81      0.81     33742
weighted avg       0.81      0.81      0.81     33742

precision_score = 0.7882129932793382
recall_score = 0.8347730867502129
f1_score = 0.8108251839158567
Model Eval Success
```

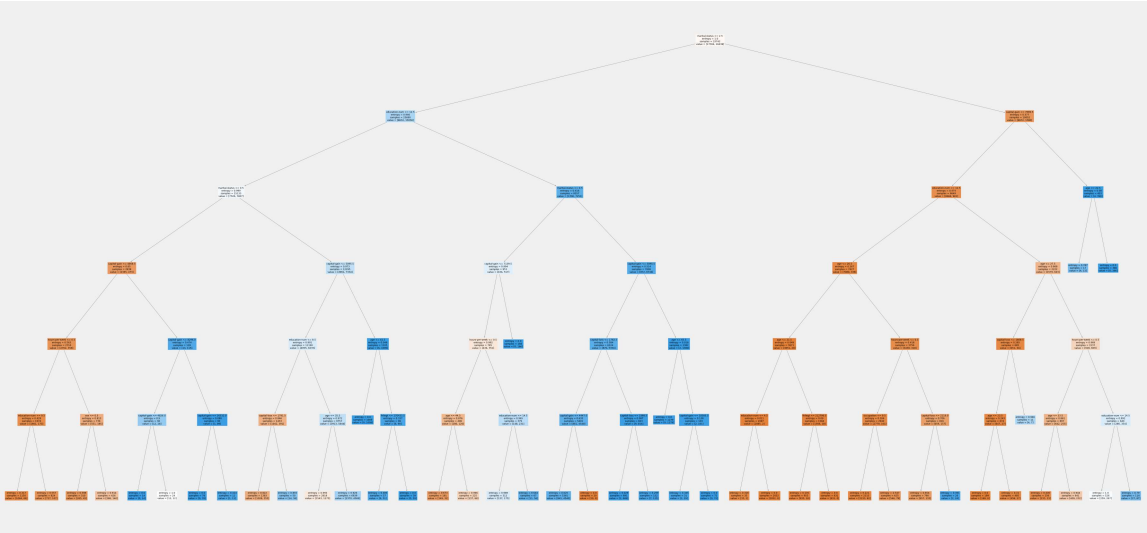
In [31]:

```
print(model_eval(tree,x_new_test,y_new_test))
```

```
Confusion Matrix =
[[5831 1585]
 [1144 5902]]
**************************************************
Accuracy Score = 0.8112985755773752
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.84      0.79      0.81      7416
           1       0.79      0.84      0.81      7046

    accuracy                           0.81     14462
   macro avg       0.81      0.81      0.81     14462
weighted avg       0.81      0.81      0.81     14462

precision_score = 0.7882997195138239
recall_score = 0.8376383763837638
f1_score = 0.8122204637721049
Model Eval Success
```

In [32]:

```
plt.figure(figsize=(100,50))
decision_tree_ = plot_tree(tree,feature_names=x_new.columns,filled=True)
plt.savefig('decision_tree.png')
```



In [33]:

```
tree = DecisionTreeClassifier()
hype = {'criterion':['entropy','gini index'],
        'min_samples_split':np.arange(2,10),
        'min_samples_leaf':np.arange(1,10),
        'max_depth':[4,5,6,7]}
rscv = RandomizedSearchCV(tree,hype,cv=7)
rscv.fit(x_new_train,y_new_train)
rscv.best_params_
```

Out[33]:

```
{'min_samples_split': 8,
 'min_samples_leaf': 1,
 'max_depth': 7,
 'criterion': 'entropy'}
```

In [34]:

```
tree = DecisionTreeClassifier(criterion='entropy',max_depth= 7,min_samples_leaf= 7,m
tree.fit(x_new_train,y_new_train)
```

Out[34]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=7, min_samples_leaf=
7,
                       min_samples_split=8)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [35]:

```
y_pred = tree.predict(x_new_test)
```

In [36]:

```
1 print(model_eval(tree,x_new_train,y_new_train))
```

```
Confusion Matrix =
[[13969  3335]
 [ 2977 13461]]
**************************************************
Accuracy Score = 0.812933436073736
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.82      0.81      0.82     17304
           1       0.80      0.82      0.81     16438

    accuracy                           0.81     33742
   macro avg       0.81      0.81      0.81     33742
weighted avg       0.81      0.81      0.81     33742

precision_score = 0.8014408192426768
recall_score = 0.8188952427302592
f1_score = 0.8100740205813323
Model Eval Success
```

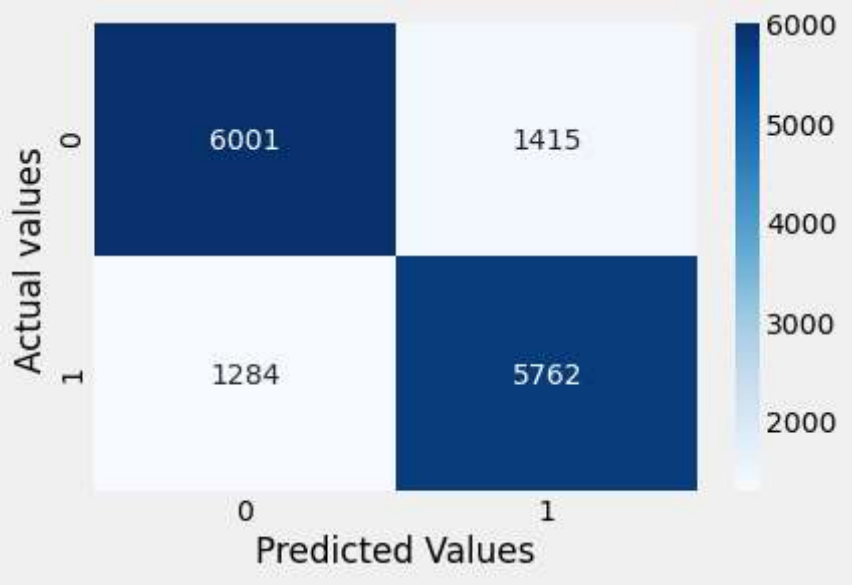
In [37]:

```
1 print(model_eval(tree,x_new_test,y_new_test))
```

```
Confusion Matrix =
[[6001 1415]
 [1284 5762]]
**************************************************
Accuracy Score = 0.8133729774581663
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.82      0.81      0.82      7416
           1       0.80      0.82      0.81      7046

    accuracy                           0.81     14462
   macro avg       0.81      0.81      0.81     14462
weighted avg       0.81      0.81      0.81     14462

precision_score = 0.8028424132645953
recall_score = 0.8177689469202384
f1_score = 0.8102369401673346
Model Eval Success
```

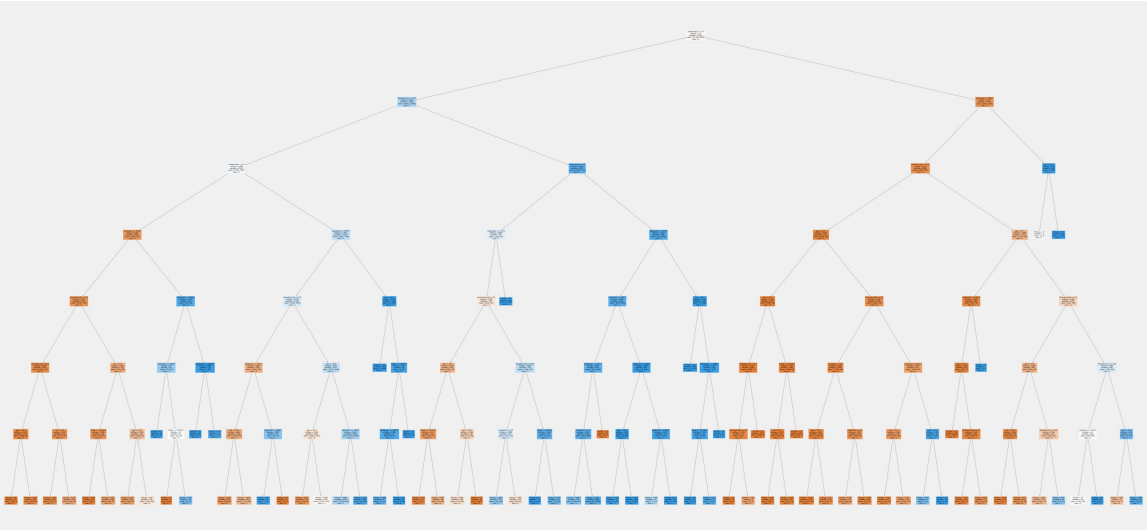
In [38]:

```
cm = confusion_matrix(y_new_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm,annot=True,fmt='d',cmap="Blues")
plt.xlabel("Predicted Values")
plt.ylabel("Actual values")
plt.show()
```



In [39]:

```
plt.figure(figsize=(100,50))
plot_tree(tree,feature_names=x_new.columns,filled=True,class_names=['0','1'])
plt.savefig('dec_tree.png')
```



In [40]:

```
model_eval_test.loc[1] = [recall_score(y_new_test,y_pred),precision_score(y_new_test
model_eval_test
```

Out[40]:

	recall	precision	f1_score	accuracy
0	0.967215	0.887254	0.925511	0.924146
1	0.817769	0.802842	0.810237	0.813373

Logistic_regression

```
Logistic Regression:

>> It is a Supervised ML algorithm which predicts categorical values based on probabilities of target class/event.
>> It is a probabilistic algorithm(0 to 1)
>> It is a Classification algorithm.
>> It is a parametric algorithm
-------------------------------------------------------------------------
>> Logistic regression is a simple and more efficient method for binary and linear classification problems.
>> It is a classification model which is very easy to realize and achieves very good performance with linearly separable
  classes.
>> It is an extensively employed algorithm for classification in industry.
>> Logistic regression is famous because it can convert the values of logits (log-odds), which can range from -∞ to +∞ to
   a range between 0 and 1.
>> As logistic functions output the probability of occurrence of an event it can be applied to many real-life scenarios.
>> It is for this reason that the logistic regression model is very popular. Another reason why logistic fairs in comparison
   to linear regression is that it is able to handle the categorical variables.
-------------------------------------------------------------------------

Assumptions of logostic regression :

1. Target var shoulf be categorical in nature (binary, ordinal , multiclass)
2. Constructs Linear Boundaries
3. Requires large datasets

>> Some others:
4. No Multicolinearity
5. Independence

-------------------------------------------------------------------------

Logistic Function (Sigmoid Function):

>>The sigmoid function is a special form of the logistic function and is usually denoted by σ(x) or sig(x).
  It is given by: σ(x) = 1/(1+exp(-x))

>> A few  properties include:
```

```
-Domain: (-∞, +∞)
-Range: (0, +1)
-σ(0) = 0.5
-The function is monotonically increasing.
-The function is continuous everywhere.
-The function is differentiable everywhere in its domain.
-Numerically, it is enough to compute this function’s value over a small range of numbers,
 e.g., [-10, +10].
 For values less than -10, the function’s value is almost zero.
 For values greater than 10, the function’s values are almost one.

>> The sigmoid function is a mathematical function used to map the predicted values to probabilities.
>> It maps any real value into another value within a range of 0 and 1.
>> The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve
   like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
>> In logistic regression we use the concept of the threshold value which defines the probability of either 0 or 1.
   Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.
>> With the help of this sigmoid function, we can successfully predict the output in terms of a probability
```

In [41]:

```
log_reg = LogisticRegression()
log_reg.fit(x_new_train,y_new_train)
```

Out[41]:

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [42]:

```
y_pred = log_reg.predict(x_new_test)
```

In [43]:

```
print(model_eval(log_reg,x_new_train,y_new_train))
```

```
Confusion Matrix =
[[16108  1196]
 [11268  5170]]
**************************************************
Accuracy Score = 0.6306087368857803
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.59      0.93      0.72     17304
           1       0.81      0.31      0.45     16438

    accuracy                           0.63     33742
   macro avg       0.70      0.62      0.59     33742
weighted avg       0.70      0.63      0.59     33742

precision_score = 0.8121269242852655
recall_score = 0.31451514782820295
f1_score = 0.4534292229433432
Model Eval Success
```

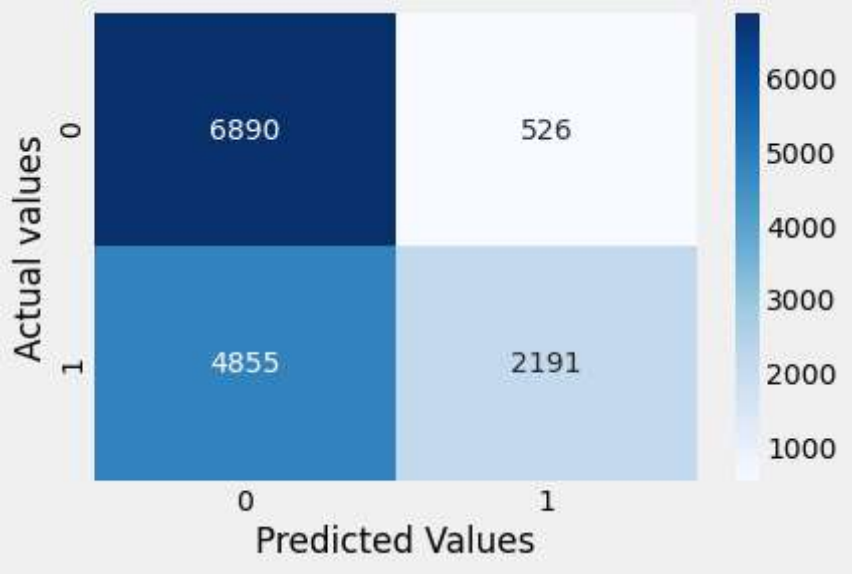
In [44]:

```
print(model_eval(log_reg,x_new_test,y_new_test))
```

```
Confusion Matrix =
[[6890  526]
 [4855 2191]]
**************************************************
Accuracy Score = 0.6279214493154474
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.59      0.93      0.72      7416
           1       0.81      0.31      0.45      7046

    accuracy                           0.63     14462
   macro avg       0.70      0.62      0.58     14462
weighted avg       0.69      0.63      0.59     14462

precision_score = 0.8064041221935959
recall_score = 0.3109565711041726
f1_score = 0.44883744750588955
Model Eval Success
```

In [45]:

```
cm = confusion_matrix(y_new_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm,annot=True,fmt='d',cmap="Blues")
plt.xlabel("Predicted Values")
plt.ylabel("Actual values")
plt.show()
```



In [46]:

```
y_pred_train_prob = log_reg.predict_proba(x_new_train)
y_pred_train_prob
```

Out[46]:

```
array([[0.5443506 , 0.4556494 ],
       [0.51773696, 0.48226304],
       [0.54235258, 0.45764742],
       ...,
       [0.52969392, 0.47030608],
       [0.5674072 , 0.4325928 ],
       [0.59543042, 0.40456958]])
```

In [47]:

```
y_pred_train_prob_class1 = y_pred_train_prob[:,1]
y_pred_train_prob_class1
```

Out[47]:

```
array([0.4556494 , 0.48226304, 0.45764742, ..., 0.47030608, 0.4325928 ,
       0.40456958])
```

AUC-ROC curve

```
ROC :
    >>The Receiver Operator Characteristic (ROC) curve is an evaluation metric for
binary classification problems.
    >>It is a probability curve that plots the TPR against FPR at various
threshold values and essentially separates
```

```
     the ‘signal’ from the ‘noise.’
   >>In other words, it shows the performance of a classification model at all classification thresholds.
--------------------------------------------------------------------------------
Importance of ROC curve :

>> A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve,
   or ROC curve.
>> The curves of different models can be compared directly in general or for different thresholds.
>> The area under the curve (AUC) can be used as a summary of the model skill.
>> The shape of the curve contains a lot of information, including what we might care about most for a problem,
   the expected false positive rate, and the false negative rate
>> An operator may plot the ROC curve for the final model and choose a threshold that gives a desirable balance
   between the false positives and false negatives.
```

In [48]:

```
fpr,tpr,threshold = roc_curve(y_new_train, y_pred_train_prob_class1)
np.around(threshold,3)
```

Out[48]:

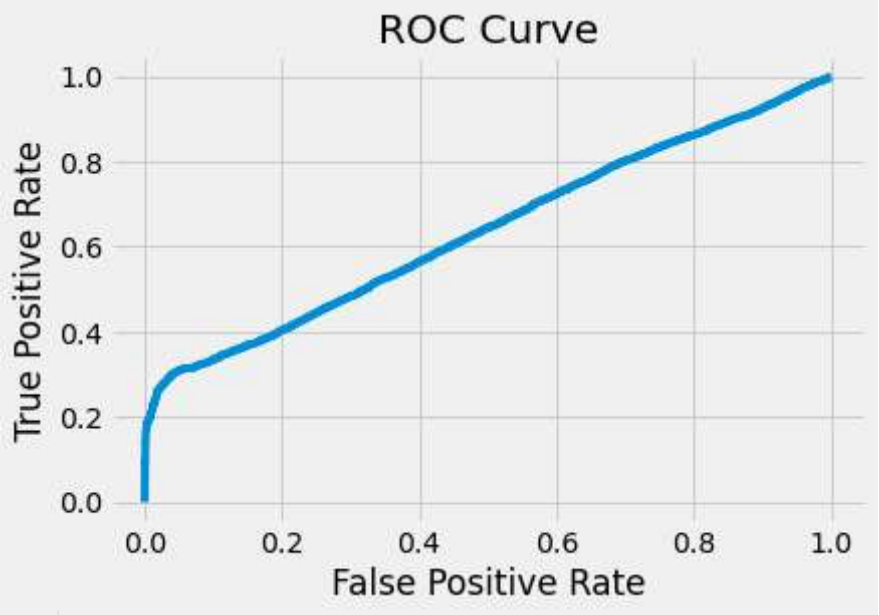
```
array([2.   , 1.   , 1.   , ..., 0.132, 0.124, 0.086])
```

In [49]:

```
plt.plot(fpr,tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
```

Out[49]:

```
Text(0.5, 1.0, 'ROC Curve')
```




```
>>The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems.
```

```
>>It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the ‘signal’
  from the ‘noise.’ In other words, it shows the performance of a classification model at all classification thresholds.
>>The Area Under the Curve (AUC) is the measure of the ability of a binary classifier to distinguish between classes and is
  used as a summary of the ROC curve.
>>The higher the AUC, the better the model’s performance at distinguishing between the positive and negative classes.
>>When AUC = 1, the classifier can correctly distinguish between all the Positive and the Negative class points.
  If, however, the AUC had been 0, then the classifier would predict all Negatives as Positives and all Positives as Negatives.
>>When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from
 the negative ones. This is so because the classifier is able to detect more numbers of True positives and True negatives
 than False negatives and False positives
>>When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points.
  Meaning that the classifier either predicts a random class or a constant class for all the data points.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.
```

In [50]:

```
roc_auc_score(y_new,log_reg.predict_proba(x_new)[:,1])
```

Out[50]:

```
0.6353182858025213
```

In [51]:

```
roc_auc_score(y_new,log_reg.decision_function(x_new))
```

Out[51]:

```
0.6353182858025213
```

Here the value for roc_auc_curve is not much better, is far from 1. so regarding to assumptions of roc_auc the classifier is not able to distinguish between Positive and Negative class points. Meaning that the classifier either predicts a random class or a constant class for all the data points.

In [52]:

```
model_eval_test.loc[2] = [recall_score(y_new_test,y_pred),precision_score(y_new_test
model_eval_test
```

Out[52]:

	recall	precision	f1_score	accuracy
0	0.967215	0.887254	0.925511	0.924146
1	0.817769	0.802842	0.810237	0.813373
2	0.310957	0.806404	0.448837	0.627921

K-Nearest Neighbors

```
The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

The K-Nearest Neighbor algorithm is a supervised learning algorithm that can be used for both classification and regression tasks. The algorithm works by finding the K nearest neighbors to a given data point, and then using those neighbors to predict the class or value of the data point.

It uses data in which there is a target column present i.e, labelled data to model a function to produce an output for the unseen data. It uses the euclidean distance formula to compute the distance between the data points for classification or prediction.

The main objective of this algorithm is that similar data points must be close to each other so it uses the distance to calculate the similar points that are close to each other.

>>>> The following operations have happened during each iteration of the algorithm. For each of the unseen or test data point, the kNN classifier must:

Step-1: Calculate the distances of test point to all points in the training set and store them

Step-2: Sort the calculated distances in increasing order

Step-3: Store the K nearest points from our training dataset

Step-4: Calculate the proportions of each class

Step-5: Assign the class with the highest proportion

>>>> Advantages:

1. No Training Period: It does not learn anything during the training period since it does not find any discriminative function with the help of the training data. In simple words, actually, there is no training period for the KNN algorithm. It stores the training dataset and learns from it only when we use the algorithm for making the real-time predictions on the test dataset.As a result, the KNN algorithm is much faster than other algorithms which require training. For Example, SupportVector Machines(SVMs), Linear Regression, etc.Moreover, since the KNN algorithm does not require any training before making predictions as a result new data can be added seamlessly without impacting the accuracy of the algorithm.

2. Easy to implement and understand: To implement the KNN algorithm, we need only two
parameters i.e. the value of K and the distance metric(e.g. Euclidean or Manhattan, etc.).
Since both the parameters are easily interpretable therefore they are easy to understand.

Applications of KNN >>>

1. Text mining
2. Agriculture
3. Finance
```

```
4. Medical
5. Facial recognition
6. Recommendation systems (Amazon, Hulu, Netflix, etc)

The various real-life applications of the KNN Algorithm includes:

1. KNN allows the calculation of the credit rating. By collecting the financial characteristics vs. comparing people having similar financial features to a database we can calculate the same. Moreover, the very nature of a credit rating where people who have similar financial details would be given similar credit ratings also plays an important role. Hence the existing database can then be used to predict a new customer’s credit rating, without having to perform all the calculations.

2. In political science: KNN can also be used to predict whether a potential voter “will vote” or “will not vote”, or to “vote Democrat” or “vote Republican” in an election.Apart from the above-mentioned use cases, KNN algorithms are also used for handwriting detection (like OCR), Image recognition, and video recognition.

```

In [53]:

```
x_new_train, x_new_test, y_new_train,y_new_test = train_test_split(x_new,y_new,test_
x_new_train.shape, x_new_test.shape, y_new_train.shape, y_new_test.shape
```

Out[53]:

```
((33742, 13), (14462, 13), (33742,), (14462,))
```

In [54]:

```
normal_scaler = MinMaxScaler() # x_norm = x - x_min / x_max-x_min
normal_scaler.fit(x_new) ###   x_min && x_max, Normalization
normalized_x = normal_scaler.transform(x_new) ### Apply the transformed values and r
x_norm = pd.DataFrame(normalized_x,columns=x_new.columns)
x_norm
```

Out[54]:

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	s
0	0.301370	0.0	0.044302	0.800000	0.666667	0.071429	0.2	1.0	
1	0.452055	0.0	0.048238	0.800000	0.333333	0.285714	0.0	1.0	
2	0.287671	1.0	0.138113	0.533333	0.000000	0.428571	0.2	1.0	
3	0.493151	1.0	0.151068	0.400000	0.333333	0.428571	0.0	0.0	
4	0.150685	1.0	0.221488	0.800000	0.333333	0.714286	1.0	0.0	
...	...	...	...	...	...	...	...	...	
48199	0.410959	1.0	0.099157	0.866667	0.333333	0.285714	0.0	1.0	
48200	0.438356	0.0	0.077157	0.800000	0.333333	0.285714	0.0	1.0	
48201	0.383562	1.0	0.075655	0.533333	0.333333	0.857143	0.0	1.0	
48202	0.232877	1.0	0.098791	0.800000	0.333333	0.285714	0.0	1.0	
48203	0.602740	0.0	0.020818	0.800000	0.666667	0.000000	0.2	1.0	

48204 rows × 13 columns

In [55]:

```
x_norm_train, x_norm_test, y_new_train,y_new_test = train_test_split(x_norm,y_new,te
```

In [56]:

```
knn_clf_model = KNeighborsClassifier(n_neighbors=5, p=2)
knn_clf_model.fit(x_new_train,y_new_train) ## data will be loaded ### LAZY Learner
```

Out[56]:

```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [57]:

```
# Training Data model Evaluation
model_eval(knn_clf_model,x_new_train,y_new_train)
```

```
Confusion Matrix =
[[13054  4250]
 [ 1956 14482]]
**************************************************
Accuracy Score = 0.8160749214628653
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.87      0.75      0.81     17304
           1       0.77      0.88      0.82     16438

    accuracy                           0.82     33742
   macro avg       0.82      0.82      0.82     33742
weighted avg       0.82      0.82      0.82     33742

precision_score = 0.7731155242366005
recall_score = 0.8810074218274729
f1_score = 0.8235427921524027
```

Out[57]:

```
'Model Eval Success'
```

In [58]:

```
# Testing Data model Evaluation
model_eval(knn_clf_model,x_new_test,y_new_test)
```

```
Confusion Matrix =
[[4859 2557]
 [1575 5471]]
**************************************************
Accuracy Score = 0.7142857142857143
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.76      0.66      0.70      7416
           1       0.68      0.78      0.73      7046

    accuracy                           0.71     14462
   macro avg       0.72      0.72      0.71     14462
weighted avg       0.72      0.71      0.71     14462

precision_score = 0.6814897857498754
recall_score = 0.7764689185353392
f1_score = 0.7258856308876209
```

Out[58]:

```
'Model Eval Success'
```

Hyperparameter tuning :

```
The two best strategies for Hyperparameter tuning are: GridSearchCV.
RandomizedSearchCV.

>>>> GridSearchCV :

In GridSearchCV approach, the machine learning model is evaluated for a range of
hyperparameter values. This approach is called GridSearchCV, because it searches
for the best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set two hyperparameters C and Alpha of the Logistic
Regression Classifier model, with different sets of values. The grid search
technique will construct many versions of the model with all possible combinations
of hyperparameters and will return the best one.

>>>> RandomizedSearchCV :

RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a
fixed number of hyperparameter settings. It moves within the grid in a random
fashion to find the best set of hyperparameters. This approach reduces unnecessary
computation.
```

In [59]:

```
knn_model = KNeighborsClassifier(n_neighbors=5, p = 2)

hyp = {
    "n_neighbors":np.arange(3,20),
    "p":[1,2]
}

gscv_knn = GridSearchCV(knn_model,hyp,cv= 5)
gscv_knn.fit(x_new_train,y_new_train)
gscv_knn.best_estimator_
```

Out[59]:

```
KNeighborsClassifier(n_neighbors=3, p=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [60]:

```
knn_hyp = gscv_knn.best_estimator_
knn_hyp
```

Out[60]:

```
KNeighborsClassifier(n_neighbors=3, p=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:

```
y_pred = knn_hyp.predict(x_new_test)
```

In [62]:

```
# Training Data evalaution
model_eval(knn_hyp,x_new_train,y_new_train)
```

```
Confusion Matrix =
[[14148  3156]
 [  701 15737]]
**************************************************
Accuracy Score = 0.8856914231521545
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.95      0.82      0.88     17304
           1       0.83      0.96      0.89     16438

    accuracy                           0.89     33742
   macro avg       0.89      0.89      0.89     33742
weighted avg       0.89      0.89      0.89     33742

precision_score = 0.8329540041285132
recall_score = 0.9573549093563694
f1_score = 0.8908324134612663
```

Out[62]:

```
'Model Eval Success'
```

In [63]:

```
# Testing data evaluation
model_eval(knn_hyp,x_new_test,y_new_test)
```

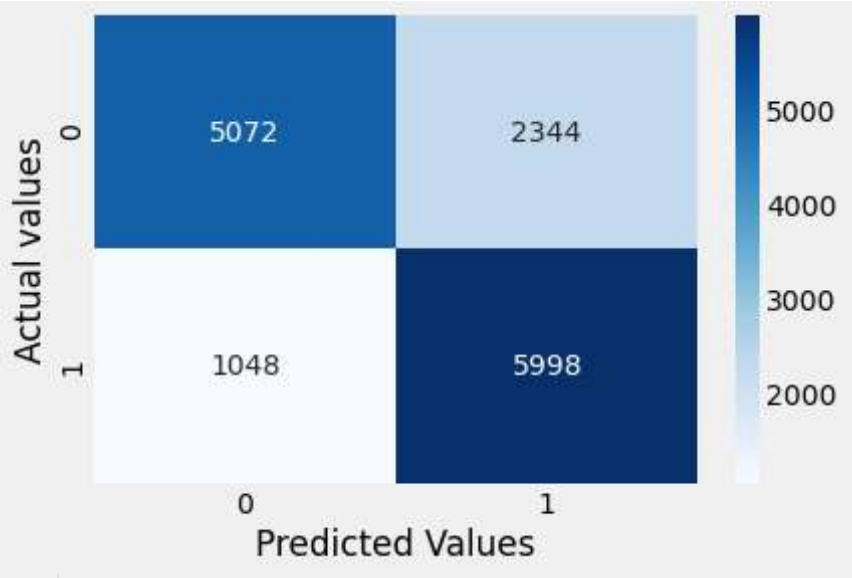
```
Confusion Matrix =
[[5072 2344]
 [1048 5998]]
**************************************************
Accuracy Score = 0.7654542940118932
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.83      0.68      0.75      7416
           1       0.72      0.85      0.78      7046

    accuracy                           0.77     14462
   macro avg       0.77      0.77      0.76     14462
weighted avg       0.78      0.77      0.76     14462

precision_score = 0.719012227283625
recall_score = 0.8512631280158955
f1_score = 0.7795684949311151
```

Out[63]:

```
'Model Eval Success'
```

In [64]:

```
cm = confusion_matrix(y_new_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm,annot=True,fmt='d',cmap="Blues")
plt.xlabel("Predicted Values")
plt.ylabel("Actual values")
plt.show()
```




```
four different distance functions, which are
Euclidean distance,
cosine similarity measure,
Minkowsky,
correlation and
```

```
Chi square are used in the k-NN classifier respectively
```

To finding the different values of k and p

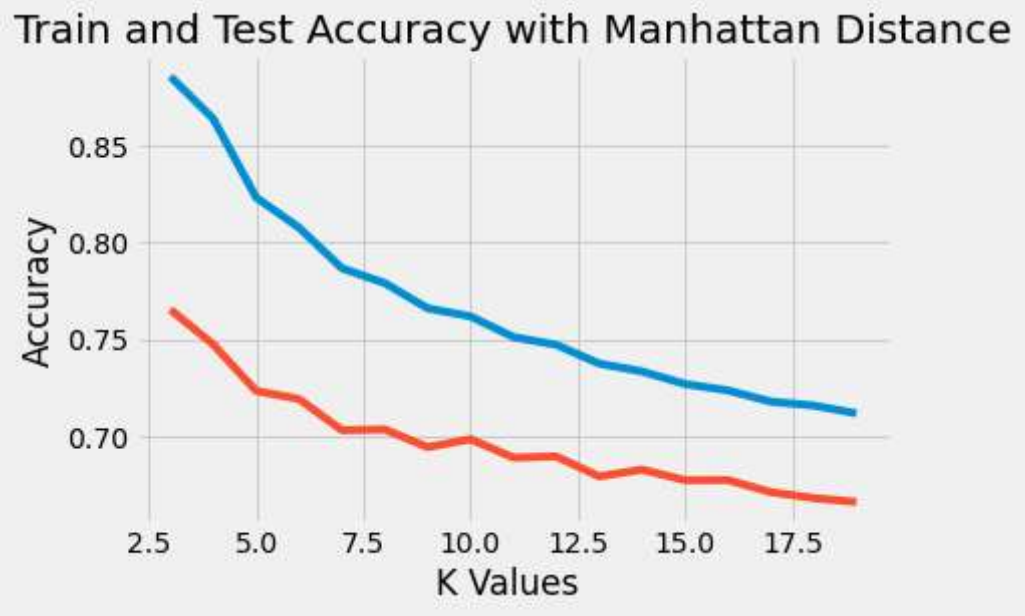
In [65]:

```
distanc_param=1
train_acc_list = []
test_acc_list = []
k_range = np.arange(3,20)
for k in k_range:
    model = KNeighborsClassifier(n_neighbors=k,p=distanc_param)
    model.fit(x_new_train,y_new_train)
    # Training Acc
    train_acc = model.score(x_new_train,y_new_train)
    train_acc_list.append(train_acc)
    # Testing Acc
    test_acc_list.append(model.score(x_new_test,y_new_test))
print(train_acc_list)
print(test_acc_list)
```

```
[0.8856914231521545, 0.8638195720467073, 0.8232173552249422, 0.80768774820
69824, 0.7867346333945824, 0.7790587398494458, 0.7660482484737123, 0.76189
91168276925, 0.7512299211664987, 0.747228972793551, 0.737508150080019, 0.7
334775650524569, 0.7269871376918974, 0.7236974690296959, 0.717918321379882
7, 0.7159030288661016, 0.7118428071839251]
[0.7654542940118932, 0.7474069976490112, 0.7234130825611949, 0.71926427879
96127, 0.7031530908588024, 0.7035679712349606, 0.6943714562301203, 0.69852
02599917024, 0.6889780113400636, 0.6896003319043009, 0.6792974692297055,
0.6828239524270502, 0.6773613608076338, 0.6774305075369935, 0.671207301894
6204, 0.6680956990734338, 0.6660904439220025]
```

In [66]:

```
plt.plot(k_range,train_acc_list)
plt.plot(k_range,test_acc_list)
plt.xlabel("K Values ")
plt.ylabel("Accuracy")
plt.title("Train and Test Accuracy with Manhattan Distance")
plt.savefig("Accuracy_MD.png")
```



In [67]:

```
print(train_acc_list[13])
test_acc_list[13]
```

```
0.7236974690296959
```

Out[67]:

```
0.6774305075369935
```

In [68]:

```
model_eval_test.loc[3] = [recall_score(y_new_test,y_pred),precision_score(y_new_test
model_eval_test
```

Out[68]:

	recall	precision	f1_score	accuracy
0	0.967215	0.887254	0.925511	0.924146
1	0.817769	0.802842	0.810237	0.813373
2	0.310957	0.806404	0.448837	0.627921
3	0.851263	0.719012	0.779568	0.765454

Adaboost

```
AdaBoost is the first designed boosting algorithm with a particular loss function.
```

```
---------------------------------------------------------------------------
-------------

boosting also creates an ensemble of classifiers by resampling the data, which are
then combined by majority voting. However, in boosting, resampling is
strategically geared to provide the most informative training data for each
consecutive classifier. In essence, each iteration of boosting creates three weak
classifiers:

the first classifier C1 is trained with a random subset of the available training
data. The training data subset for the second classifier C2 is chosen as the most
informative subset, given C1 .

Specifically, C2 is trained on a training data only half of which is correctly
classified by C1 , and the other half is misclassified.

The third classifier C3 is trained with instances on which C1 and C2 disagree. The
three classifiers are combined through a three-way majority vote.

---------------------------------------------------------------------------

Boosting has been a prevalent technique for tackling binary classification
problems. These algorithms improve the prediction power by converting a number of
weak learners to strong learners.

---------------------------------------------------------------------------

AdaBoost is an ensemble learning method (also known as “meta-learning”) which was
initially created to increase the efficiency of binary classifiers. AdaBoost uses
an iterative approach to learn from the mistakes of weak classifiers, and turn
them into strong ones.

1. Assisgn The sample weights to all datapoints
     Sample weights = 1/ Total no of samples

2. Construct Decision Stump

3. Calculate TOTAL ERROR
    TE = no of misclassified samples / total no of samples

4. Calculate the performance
    performacnce = 1/2 * log((1-TE)/TE)

    importance, Amount of Say, Siginificance, influence etc

5. Calculate New Sample Weights
    NSW = old Sample Weight * e^(+-performance)

    NSW correct = old Sample Weight * e ^(-performace)
    NSW Incorrect = old Sample Weight * e ^(+performace)

6. TOTAL weight = (NSW correct * No correct Samples) + (NSW Incorrect * No
incorrect Samples)

7. Normalization of NSW
    Norm_NSW = NSW / Total weight

    Norm_NSW_correct = NSW correct / Total weight
    Norm_NSW_Incorrect = NSW Incorrect / Total weight

```

```
8. Create a buckets using normlized weights. [0 to 1]

9. create a random array between 0 to 1, create new dataset by picking values from buckets

10. Repeat the process 1 to 9 until same result previous stump or we reach to n_estimator = max_limit
    n_estimator = 50

we go through this procedure, and if classification problem is arise then by using accuracy_score, confusion_matrix, classification report is used for evaluation and for regression mean_squared_error, mean_absolute_error, r2_score is used for evaluation.
```

In [69]:

```
ada_model = AdaBoostClassifier(base_estimator=None,
                           n_estimators=50,
                           learning_rate=1.0,
                           random_state=5)
ada_model.fit(x_new_train,y_new_train)
```

Out[69]:

```
AdaBoostClassifier(random_state=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [70]:

```
y_pred = ada_model.predict(x_new_test)
```

In [71]:

```
# training data Evalation

model_eval(ada_model,x_new_train,y_new_train)
```

```
Confusion Matrix =
[[14081  3223]
 [ 2692 13746]]
**************************************************
Accuracy Score = 0.8246991879556635
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.84      0.81      0.83     17304
           1       0.81      0.84      0.82     16438

    accuracy                           0.82     33742
   macro avg       0.82      0.82      0.82     33742
weighted avg       0.83      0.82      0.82     33742

precision_score = 0.8100654134009075
recall_score = 0.8362331183842316
f1_score = 0.8229412997276019
```

Out[71]:

```
'Model Eval Success'
```

In [72]:

```
# testing data Evalation

model_eval(ada_model,x_new_test,y_new_test)
```

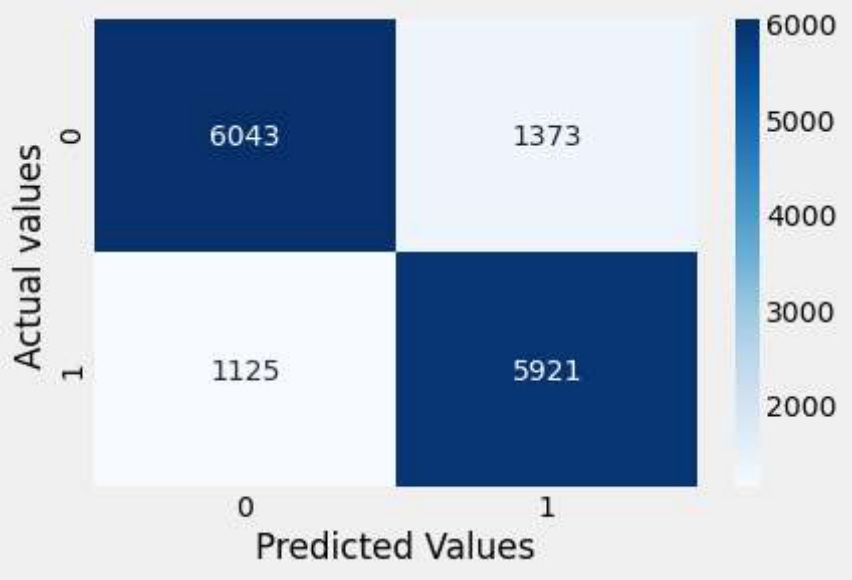
```
Confusion Matrix =
[[6043 1373]
 [1125 5921]]
**************************************************
Accuracy Score = 0.8272714700594662
**************************************************
Classification Report =
              precision    recall  f1-score   support

           0       0.84      0.81      0.83      7416
           1       0.81      0.84      0.83      7046

    accuracy                           0.83     14462
   macro avg       0.83      0.83      0.83     14462
weighted avg       0.83      0.83      0.83     14462

precision_score = 0.8117630929531121
recall_score = 0.8403349418109566
f1_score = 0.8258019525801954
```

Out[72]:

```
'Model Eval Success'
```

In [73]:

```
cm = confusion_matrix(y_new_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm,annot=True,fmt='d',cmap="Blues")
plt.xlabel("Predicted Values")
plt.ylabel("Actual values")
plt.show()
```



Adaboost with different max_depth Decision tree

In [74]:

```
train_acc_list = []
test_acc_list = []
for i in range(1,10):
    base_model = DecisionTreeClassifier()
    base_model = DecisionTreeClassifier(max_depth=i,random_state=7)
    model = AdaBoostClassifier(base_estimator=base_model,
                               n_estimators=50,
                              learning_rate=1.0,
                              random_state=10)

    model.fit(x_new_train,y_new_train)

    train_acc = model.score(x_new_train,y_new_train)
    train_acc_list.append(train_acc)

    test_acc = model.score(x_new_test,y_new_test)
    test_acc_list.append(test_acc)

    print(f"Iteration >> {i}, Training Acc = {train_acc}, Testing Acc = {test_acc}")
```

```
Iteration >> 1, Training Acc = 0.8246991879556635, Testing Acc = 0.8272714
700594662
Iteration >> 2, Training Acc = 0.8399027917728646, Testing Acc = 0.8417922
832250034
Iteration >> 3, Training Acc = 0.8508683539802027, Testing Acc = 0.8467708
47738902
Iteration >> 4, Training Acc = 0.862545195898287, Testing Acc = 0.84843036
92435348
Iteration >> 5, Training Acc = 0.883498310710687, Testing Acc = 0.85562162
90969437
Iteration >> 6, Training Acc = 0.9184102898464821, Testing Acc = 0.8715945
235790347
Iteration >> 7, Training Acc = 0.9494102305731729, Testing Acc = 0.8857696
030977735
Iteration >> 8, Training Acc = 0.9836109299982218, Testing Acc = 0.9058221
546120868
Iteration >> 9, Training Acc = 0.9999110900361567, Testing Acc = 0.9131517
079242152
```

Rather than being a model in itself, AdaBoost can be applied on top of any classifier to learn from its shortcomings and propose a more accurate model. It is usually called the "best out-of-the-box classifier" for this reason

In [75]:

```
model_eval_test.loc[4] = [recall_score(y_new_test,y_pred),precision_score(y_new_test
model_eval_test
```

Out[75]:

	recall	precision	f1_score	accuracy
0	0.967215	0.887254	0.925511	0.924146
1	0.817769	0.802842	0.810237	0.813373
2	0.310957	0.806404	0.448837	0.627921
3	0.851263	0.719012	0.779568	0.765454
4	0.840335	0.811763	0.825802	0.827271

In [76]:

```
model_eval_test.index = ['Random_forest','Decision_tree','Logistic_regression','KNN'
```

In [77]:

```
model_eval_test
```

Out[77]:

	recall	precision	f1_score	accuracy
Random_forest	0.967215	0.887254	0.925511	0.924146
Decision_tree	0.817769	0.802842	0.810237	0.813373
Logistic_regression	0.310957	0.806404	0.448837	0.627921
KNN	0.851263	0.719012	0.779568	0.765454
Adaboost	0.840335	0.811763	0.825802	0.827271

Our dataset is balanced, so we conclude our final assumption by considering accuracies of algorithms.

Here, random forest algoritm gives better accuracy rather than remaining algorithms. so we select random forest algorithm as a final model with 92.41% accuracy.