Applying Some Algorithms on our Dataset:

Know we will apply the (KNN, Decision Tree, Naïve Bayes and K-means)

So let's start with splitting our data set which we have cleaned to two section 80% train set and 20% test set as seen in the next image (1)racy

1

```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.model_selection import train_test_split
```

```
# sprate features and target
x=df.drop(['RainTomorrow'],axis=1)
y=df['RainTomorrow']
```

```
#split data to 80% train and 20% test
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y)
```

```
# scaler the data using minmax scaler
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)
```

```
x_train=pd.DataFrame(x_train,columns=X_train.columns)
x_test=pd.DataFrame(x_test,columns=X_test.columns)
```

classification metrics),**confusion_matrix**(Compute confusion matrix to evaluate the accuracy of a classification.) and

Accuracy_score(Accuracy classification score),**train_test_split**(Split arrays or matrices into random train and test subsets.) after that we have import **MinMaxScaler(**Transform features by scaling each feature to a given range)

After that we will use another way to balance data called smote

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

And that we balanced our data in image (2)

23 Model with balanced data

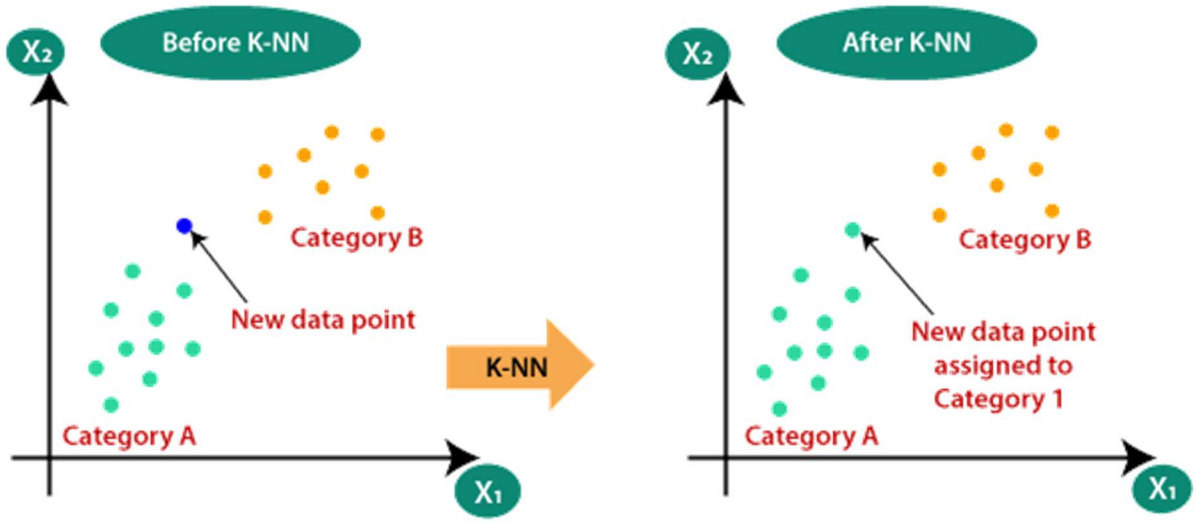
```
# balance data using oversampling
from imblearn.over_sampling import SMOTE
x_res, y_res = SMOTE().fit_resample(x, y)
```

```
#split resample data to 80% train and 20% test
X_res_train,X_res_test,y_res_train,y_res_test=train_test_split(x_res,y_res,test_size=0.
↪2,stratify=y_res)
```

So let's start with knn algorithm

First what is knn ?

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. The K-Nearest Neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph. There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). **This article demonstrates an illustration of K-nearest neighbors on a sample random data using sklearn library.**



For more information see the link below

(https://www.geeksforgeeks.org/k-nearest-neighbors-with-python-ml/#:~:text=The%20K%2DNearest%20Neighbors%20(KNN,are%20near%20to%20each%20other.)

Now let's see the code in image(3)

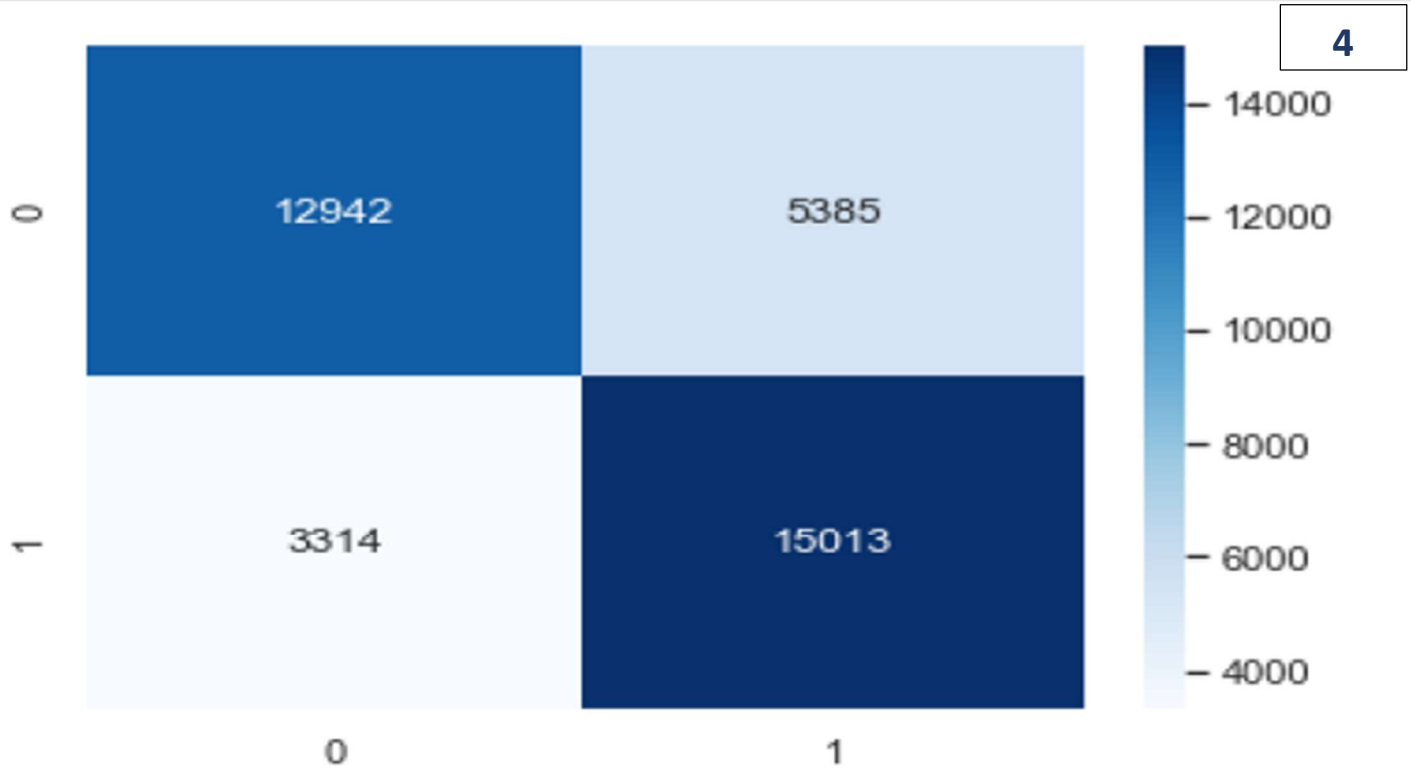
24 KNN Model balanced

3

```
knn = KNeighborsClassifier(int((len(x_res_train)**(0.5))/2))
# fit the KNN model in our resample train data
knn.fit(x_res_train,y_res_train)
# predict the test data
pr=knn.predict(x_res_test)
# show confusion matrix
mat=confusion_matrix(y_res_test, pr)
sns.heatmap(mat,annot=True,fmt="g",cmap='Blues')
plt.show()
print(classification_report(y_res_test, pr))
# show Accuracy of KNN model
print("KNN Accuracy: ",accuracy_score(y_res_test, pr))
```

So we have import the classifier of knn given it its training set then fit the knn with our data after that we have predicted our new data then we have drown confusion matrix to which evaluate the accuracy of algorithm and showed it by heatmap as shown in image (4)

4



And then we will see the accuracy, precision, recall, fi-score, and support

Precision quantifies the number of positive class predictions that belong to the positive class.
Recall quantifies the number of positive class predictions made from all positive examples in the dataset.
F-Measure provides a single score that balances both the concerns of precision and recall in one number

As seen in the image (5)

5

33

	precision	recall	f1-score	support
0	0.80	0.71	0.75	18327
1	0.74	0.82	0.78	18327
accuracy			0.76	36654
macro avg	0.77	0.76	0.76	36654
weighted avg	0.77	0.76	0.76	36654

```
KNN Accuracy:  0.7626725596115022
```

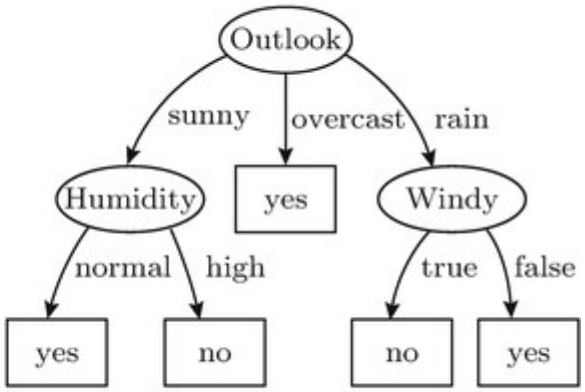
Now let's apply Decision Tree algorithm

What is Decision Tree?

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Construction of Decision Tree:
A tree can be *"learned"* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Outlook	*Temp*	*Humidity*	*Windy*	*Golf?*
rainy	hot	high	false	no
rainy	hot	high	true	no
overcast	hot	high	false	yes
sunny	mild	high	false	yes
sunny	cool	normal	false	yes
sunny	cool	normal	true	no
overcast	cool	normal	true	yes
rainy	mild	high	false	no
rainy	cool	normal	false	yes
sunny	mild	normal	false	yes
rainy	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
sunny	mild	high	true	no



For more information about decision tree see the link below

(www.geeksforgeeks.org/decision-tree)

And now with the code in image(6)

25 Decision tree model balanced

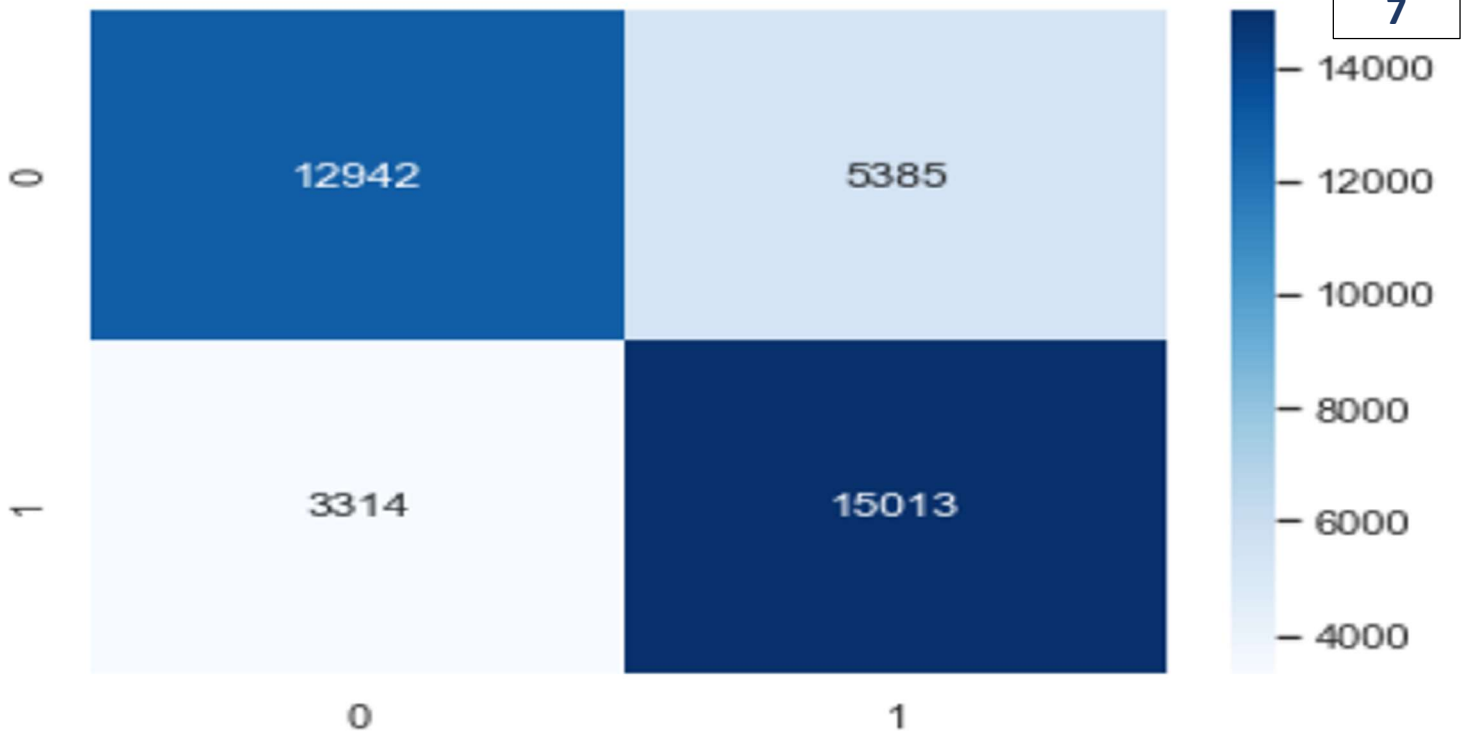
6

```
dt = DecisionTreeClassifier()
# fit the Decision tree model in our resample train data
dt.fit(x_res_train,y_res_train)
# predict the test data
predictions = dt.predict(x_res_test)
dat=confusion_matrix(y_res_test, predictions)
sns.heatmap(dat,annot=True,fmt="g",cmap='Blues')
plt.show()
print(classification_report(y_res_test, predictions))
# show Accuracy Decision tree model
print("Decision tree Accuracy: ",accuracy_score(y_res_test, predictions))
```

First we import DecisionTreeClassifier() then we have fitted data on our training data then making new prediction and drawn confusion matrix applying it on heatmap then print the report of precision, recall, fi-score and accuracy

As seen in image (7) and image (8)

7



8

	precision	recall	f1-score	support
				33
0	0.80	0.71	0.75	18327
1	0.74	0.82	0.78	18327
accuracy			0.76	36654
macro avg	0.77	0.76	0.76	36654
weighted avg	0.77	0.76	0.76	36654

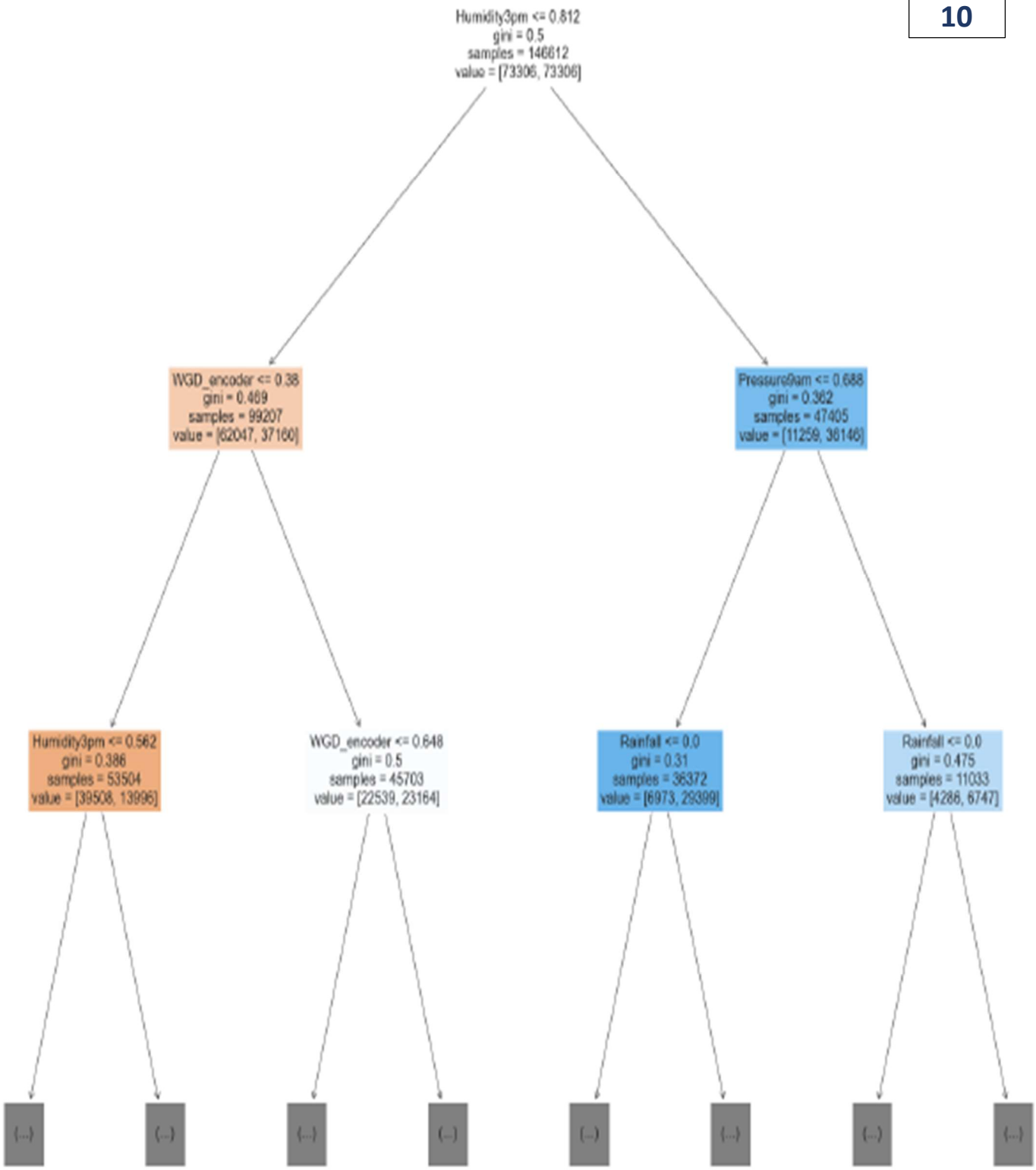
KNN Accuracy: 0.7626725596115022

And here is a code for drawing the tree and its output in images (9) and (10)

9

```
# plot 2 depth of Decision tree
plt.figure(figsize=(30,30))
tree.plot_tree(dt,max_depth=2,feature_names=x_res_train.columns,filled=True)
plt.show()
```

10



Now let's apply Naïve Bayes algorithm

What is Naïve Bayes?

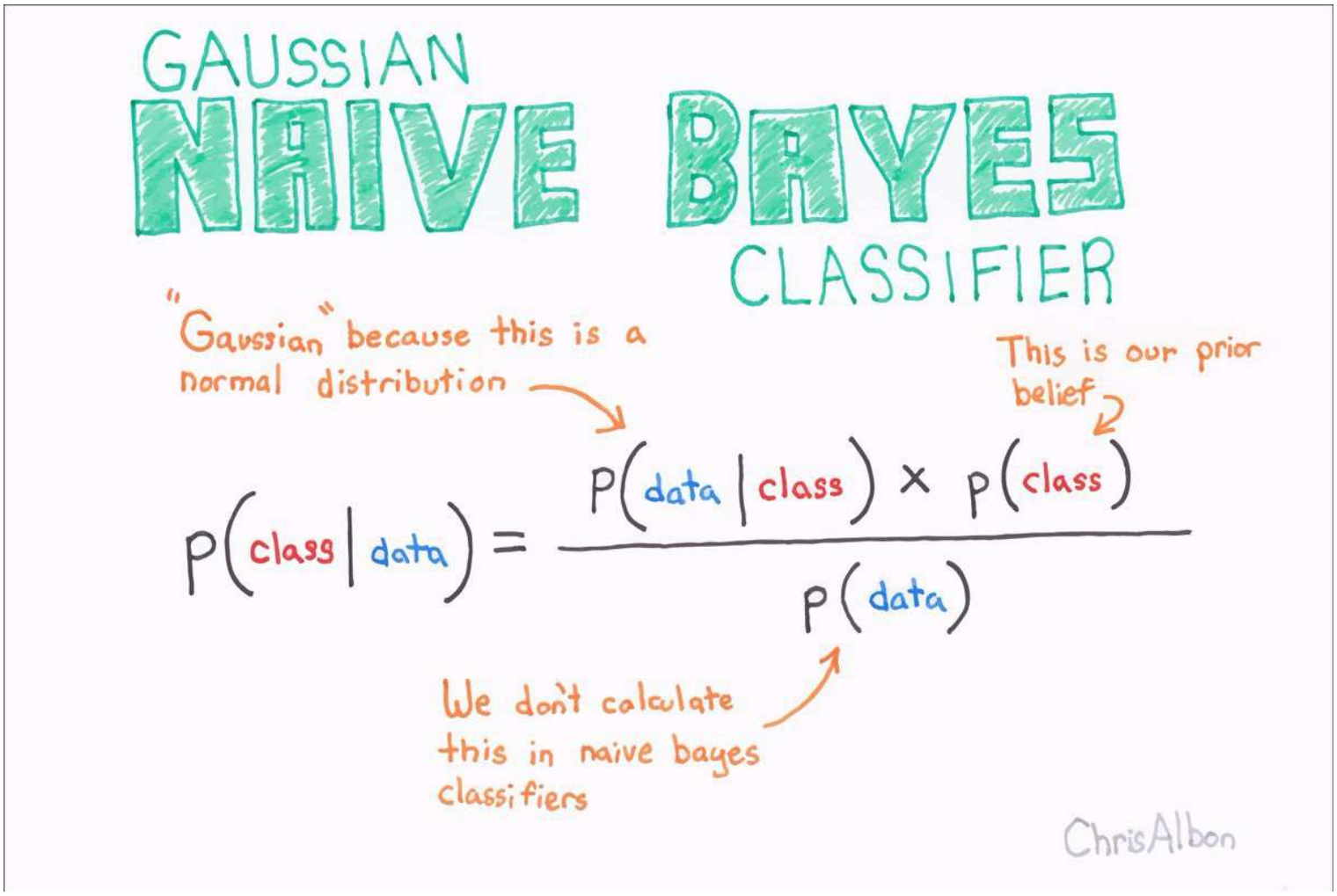
It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

To start with, let us consider a dataset.

Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for playing golf.

For more information about Naïve Bayes see the link (www.geeksforgeeks.org/naive-bayes-classifiers)

And this data set will be used widely in our code



And here is our code let's start:

In the following image(11)

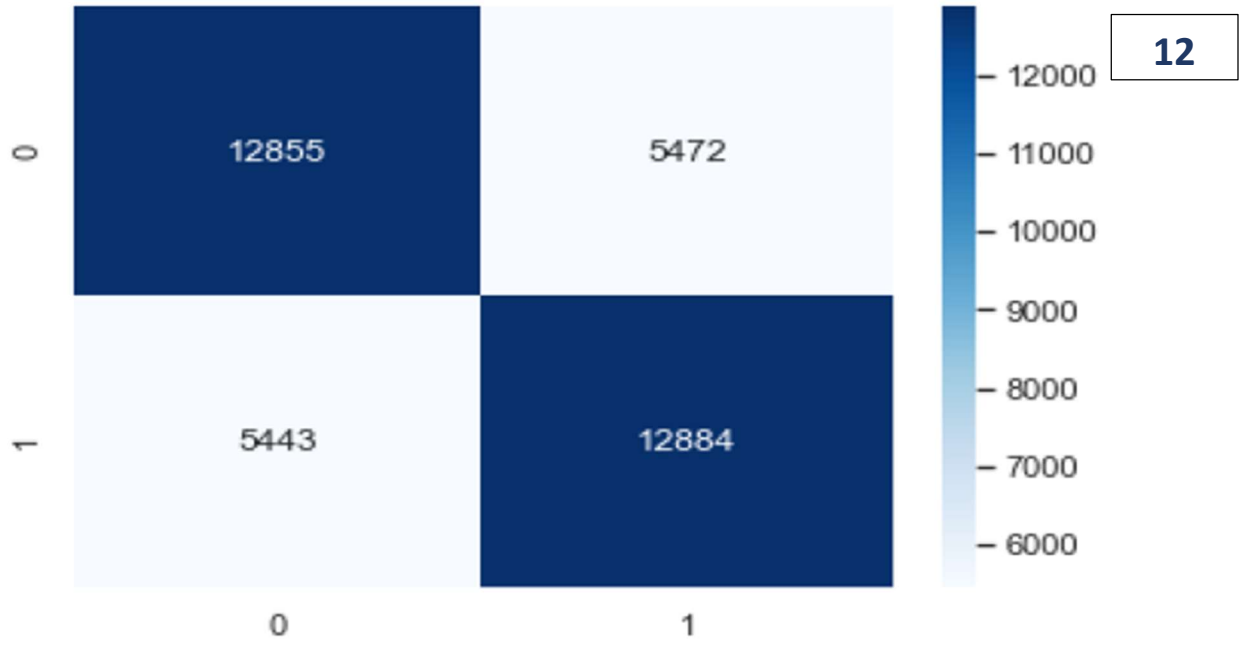
26 Naive Bayes Model balanced

11

```
nav = GaussianNB()
# fit the model in our train data
nav.fit(x_res_train,y_res_train)
# predict the test data
y_pr=nav.predict(x_res_test)
# show confusion matrix
nv=confusion_matrix(y_res_test, y_pr)
sns.heatmap(nv,annot=True,fmt="g",cmap='Blues')
plt.show()
print(classification_report(y_res_test, y_pr))
# show Accuracy of Naive Bayes model
print("Naive Bayes Accuracy: ",accuracy_score(y_res_test, y_pr))
```

We have imported Gaussian Naïve Bayes then applied it to our data by fitting the train sets then we have made our predictions for to test the accuracy later then we implemented the confusion matrix to see how accurate our predicated data is and apply it on a heat map to make it clear as a visualization then we have printed report of precision, recall, fi-score and accuracy as shown in images(12,13)

12



13

	precision	recall	f1-score	support
0	0.70	0.70	0.70	18327
1	0.70	0.70	0.70	18327
accuracy			0.70	36654
macro avg	0.70	0.70	0.70	36654
weighted avg	0.70	0.70	0.70	36654

Naive Bayes Accuracy: 0.7022153107437115