# Data Mining

CLASSIFICATION

LAB 6

# Classification Outline

| NO | Topics |
| --- | --- |
| 1 | Classification Model |
| 2 | K- Nearest Neighbors |
| 3 | Naïve Bays |

# Classification

Unlike regression where you predict a continuous number, you use classification to predict a *category*.

Classification models include linear models like Logistic Regression, SVM, and nonlinear ones like K-NN, Kernel SVM and Random Forests.

# Classification Applications

# K-Nearest Neighbors (KNN)

- ✓ It is a non-parametric, supervised learning classifier
- ✓ It is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

# How does it work?

- ✓ step 1: choose the number of K neighbors.
- ✓ Step 2: Take the K neighbors of the new data point.
- ✓ Step 3: Among these K neighbors you need to count the number of data points in each category.
- ✓ Step 4: Assign the new data point to the category where you counted the most neighbors.

# How does it work?

# Classification Model Steps

1. Splitting dataset into the Training set and Test set
2. Training the KNN model on the Training set
3. Predicting the Test set results
4. Making the Confusion Matrix
5. Visualizing the Confusion Matrix

# Getting Data ready for Classification

| Age | EstimatedSalary | Purchased |
|---|---|---|
| 19 | 19000 | 0 |
| 35 | 20000 | 0 |
| 26 | 43000 | 0 |
| 27 | 57000 | 0 |
| 19 | 76000 | 0 |
| 27 | 58000 | 0 |
| 27 | 84000 | 0 |
| 32 | 150000 | 1 |
| 25 | 33000 | 0 |
| 35 | 65000 | 0 |
| 26 | 80000 | 0 |
| 26 | 52000 | 0 |

**Dataset composed of 1000 row.**

**Age and Salary** are the **independent** variables.

**Purchased** is the **dependent** variable

# Getting Data ready for Classification

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt # visualization lib
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values  # independent variables/ Features
y = dataset.iloc[:, -1].values   # dependent variables ( to be predicted)
```

# Getting Data ready for Classification

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
print(X_train); print(y_train); print(X_test); print(y_test);

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

# Training set after scaling

X_train - NumPy object array

| | 0 | 1 |
|---|---|---|
| 0 | 0.581649 | -0.886707 |
| 1 | -0.606738 | 1.46174 |
| 2 | -0.0125441 | -0.567782 |
| 3 | -0.606738 | 1.89663 |
| 4 | 1.37391 | -1.40858 |
| 5 | 1.47294 | 0.997847 |
| 6 | 0.0864882 | -0.799728 |
| 7 | -0.0125441 | -0.248858 |
| 8 | -0.210609 | -0.567782 |
| 9 | -0.210609 | -0.190872 |
| 10 | -0.309641 | -1.29261 |

# Training the K-NN model on the Training set

| sklearn.neighbors.**KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)** | |
|---|---|
| **neighbors** | is a module implements the k-nearest neighbors algorithm |
| **KNeighborsClassifier** | Classifier implementing the k-nearest neighbors vote.<br>**n_neighbors : *int, default=5***<br>**Metric , *default='minkowski'***<br>**P** *, default=2,* Power parameter for the Minkowski metric. |
| **fit(X, y)** | A method to fit the k-nearest neighbors classifier from the training dataset. |
| **predict(X)** | A Method to predict the class labels for the provided data. |

# Training the K-NN model on the Training set

```python
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5,
                                  metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

# Predicting the Test Set Results

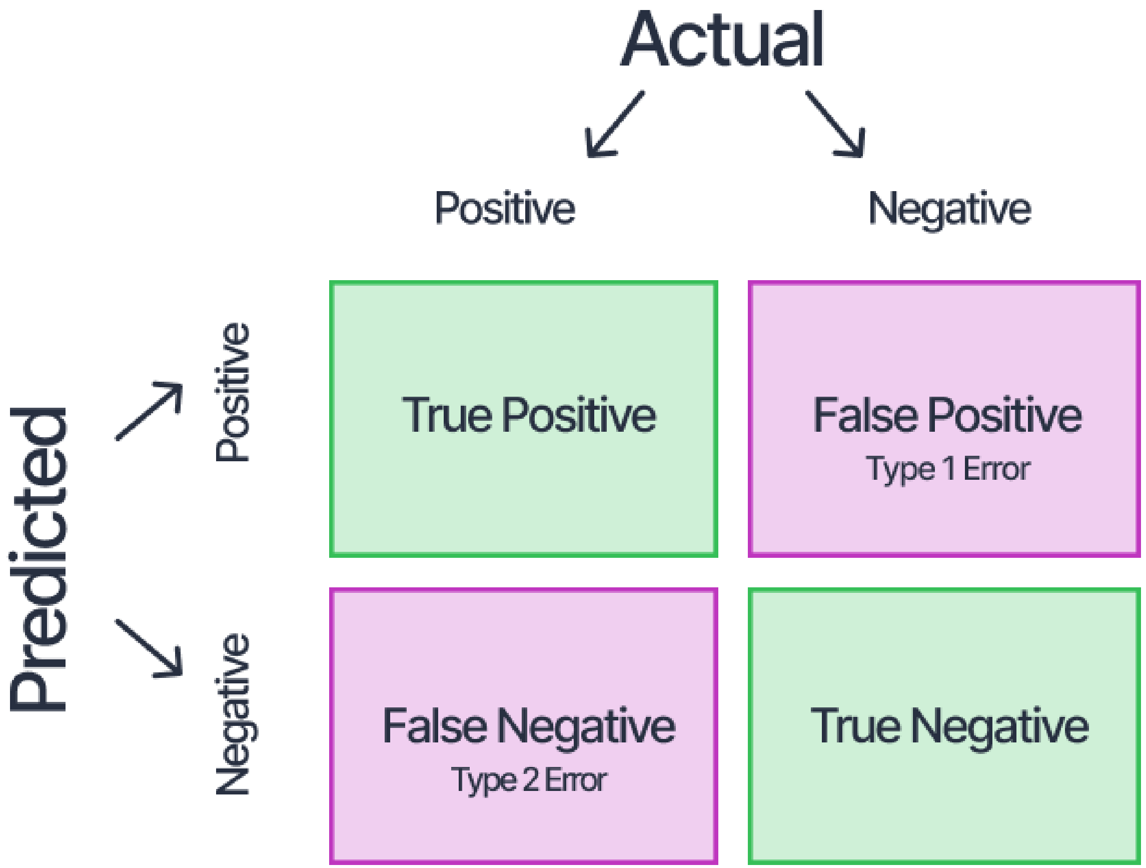
```
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

# What is a Confusion Matrix?

**A confusion matrix** is a summary of prediction results on a classification problem.

- The number of correct and incorrect predictions are summarized with count values and broken down by each class.
- The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

# What is a Confusion Matrix?



# Making the Confusion Matrix

| Name | Description |
|---|---|
| **sklearn.metrics** | Is a module includes score functions, performance metrics and pairwise metrics and distance computations. |
| **confusion_matrix** | Is a function which Computes confusion matrix to evaluate the accuracy of a classification. |
| **accuracy_score** | A function to computes the accuracy $$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$ |

# Making the Confusion Matrix

```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
Out[4]: 0.93
```

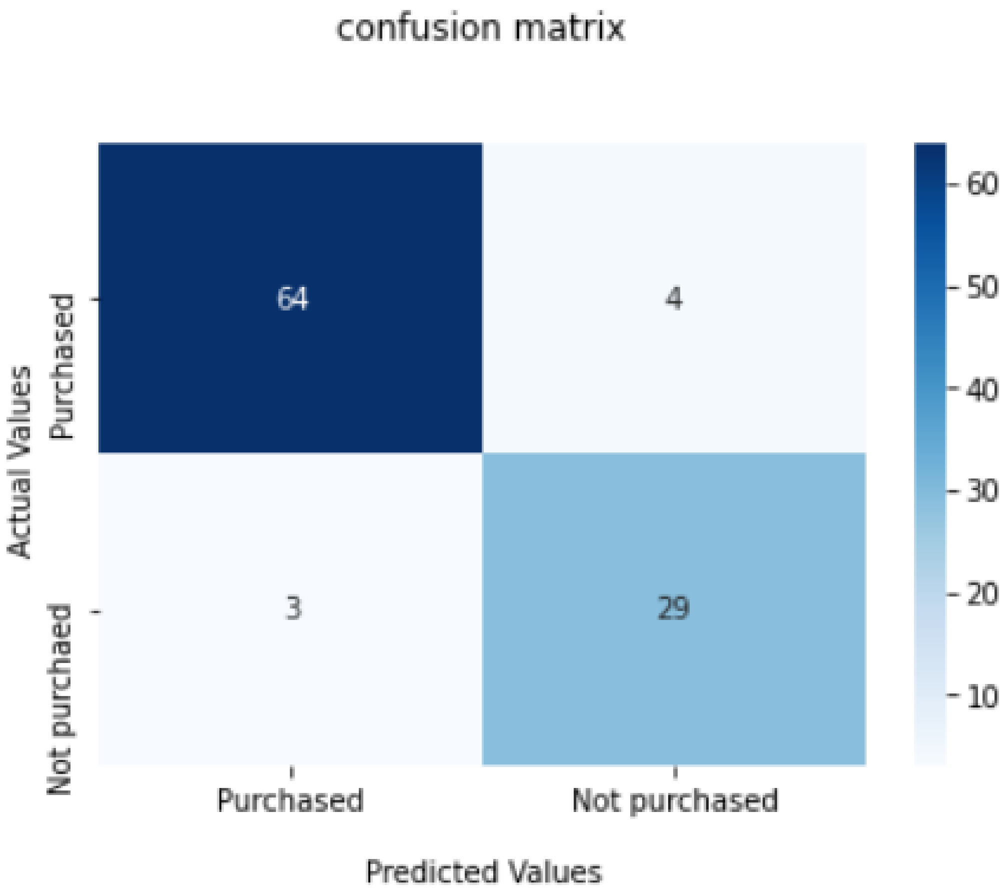
# Visualising the Confusion Matrix

| Name | Description |
| --- | --- |
| **Seaborn** | Is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. |
| **heatmap** | This is an Axes-level function and will draw the heatmap into the currently-active Axes if none is provided to the ax argument. |

# Visualising the Confusion Matrix

```
#visualizing confusion matrix
import seaborn as sns
ax = sns.heatmap(cm, annot=True, cmap='Blues')
ax.set_title('confusion matrix'+'\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Purchased','Not purchased'])
ax.yaxis.set_ticklabels(['Purchased','Not purchaed'])
## Display the visualization of the Confusion Matrix.
plt.show()
```

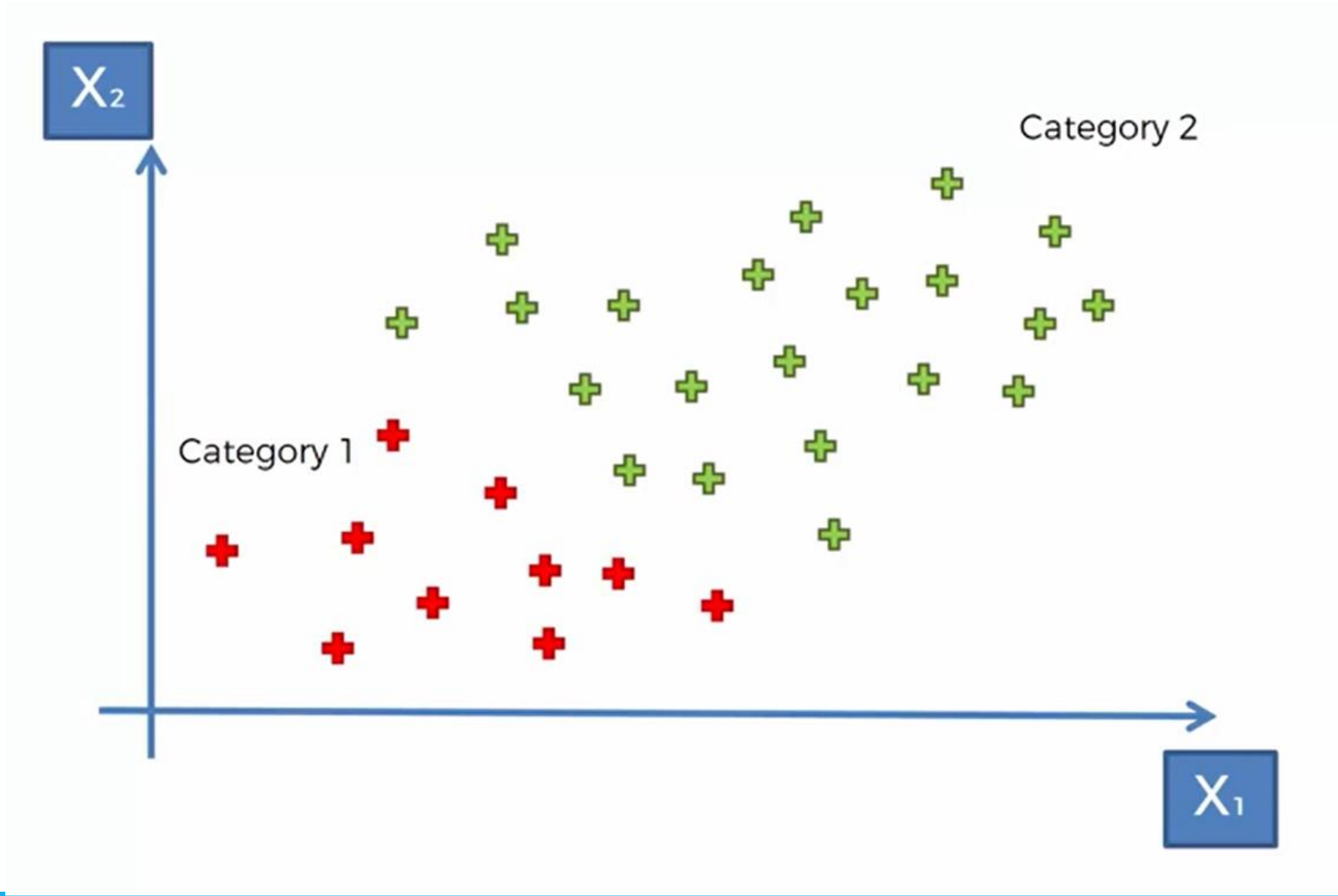
# Visualising the Confusion Matrix



# Naïve bayes

- Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- The naive Bayes classifier assumes that all features in the input data are independent of each other, which is often not true in real-world scenarios.
- However, despite this simplifying assumption, the naive Bayes classifier is widely used because of its efficiency and good performance in many real-world applications.

# Naïve Bias



# How Does Naive Bayes Algorithm Work?

- ➢Step 1: Convert the data set into a frequency table
- ➢Step 2: Create Likelihood table by finding the probabilities
- ➢Step 3: Use Naive Bayesian equation to calculate the posterior probability

$$\underbrace{P(c|x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x|c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

# Example

| Weather | Play |
|---|---|
| Sunny | No |
| Overcast | Yes |
| Rainy | Yes |
| Sunny | Yes |
| Sunny | Yes |
| Overcast | Yes |
| Rainy | No |
| Rainy | No |
| Sunny | Yes |
| Rainy | Yes |
| Sunny | No |
| Overcast | Yes |
| Overcast | Yes |
| Rainy | No |

| Frequency Table | | |
|---|---|---|
| Weather | No | Yes |
| Overcast | | 4 |
| Rainy | 3 | 2 |
| Sunny | 2 | 3 |
| Grand Total | 5 | 9 |

| Likelihood table | | | | |
|---|---|---|---|---|
| Weather | No | Yes | | |
| Overcast | | 4 | =4/14 | 0.29 |
| Rainy | 3 | 2 | =5/14 | 0.36 |
| Sunny | 2 | 3 | =5/14 | 0.36 |
| All | 5 | 9 | | |
| | =5/14 | =9/14 | | |
| | 0.36 | 0.64 | | |

# Example 1

**Problem:** Players will play if the weather is sunny. *Is this statement correct?*

*We can solve it using the above-discussed method of posterior probability.*

P(Yes | Sunny) = P( Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P( Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

# Getting Data ready for Classification

| Age | EstimatedSalary | Purchased |
|---|---|---|
| 19 | 19000 | 0 |
| 35 | 20000 | 0 |
| 26 | 43000 | 0 |
| 27 | 57000 | 0 |
| 19 | 76000 | 0 |
| 27 | 58000 | 0 |
| 27 | 84000 | 0 |
| 32 | 150000 | 1 |
| 25 | 33000 | 0 |
| 35 | 65000 | 0 |
| 26 | 80000 | 0 |
| 26 | 52000 | 0 |
| 20 | 86000 | 0 |
| 32 | 18000 | 0 |
| 18 | 82000 | 0 |
| 29 | 80000 | 0 |

**Dataset composed of 1000 row.**

**Age and Salary** are the **independent** variables.

**Purchased** is the **dependent** variable

# Getting Data ready for Classification

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt # visualization lib
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values  # independent variables/ Features
y = dataset.iloc[:, -1].values   # dependent variables ( to be predicted)
```

# Getting Data ready for Classification

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
print(X_train); print(y_train); print(X_test); print(y_test);

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

# Training set after scaling

X_train - NumPy object array

| | 0 | 1 |
|---|---|---|
| 0 | 0.581649 | -0.886707 |
| 1 | -0.606738 | 1.46174 |
| 2 | -0.0125441 | -0.567782 |
| 3 | -0.606738 | 1.89663 |
| 4 | 1.37391 | -1.40858 |
| 5 | 1.47294 | 0.997847 |
| 6 | 0.0864882 | -0.799728 |
| 7 | -0.0125441 | -0.248858 |
| 8 | -0.210609 | -0.567782 |
| 9 | -0.210609 | -0.190872 |
| 10 | -0.309641 | -1.29261 |

# Training the Naïve bayes model on the Training set

| class sklearn.naive_bayes.GaussianNB() | |
|---|---|
| **naive_bayes** | is a module implements Naive Bayes algorithms. |
| **GaussianNB()** | Classifier implementing the gaussian naïve bayes. |
| **fit(X, y)** | A method to fit the Naïve bayes classifier from the training dataset. |
| **predict(X)** | A Method to predict the class labels for the provided data. |

# Training the Naïve bayes model on the Training set

```
# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

# Predicting the Test Set Results

```
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

# Making the Confusion Matrix

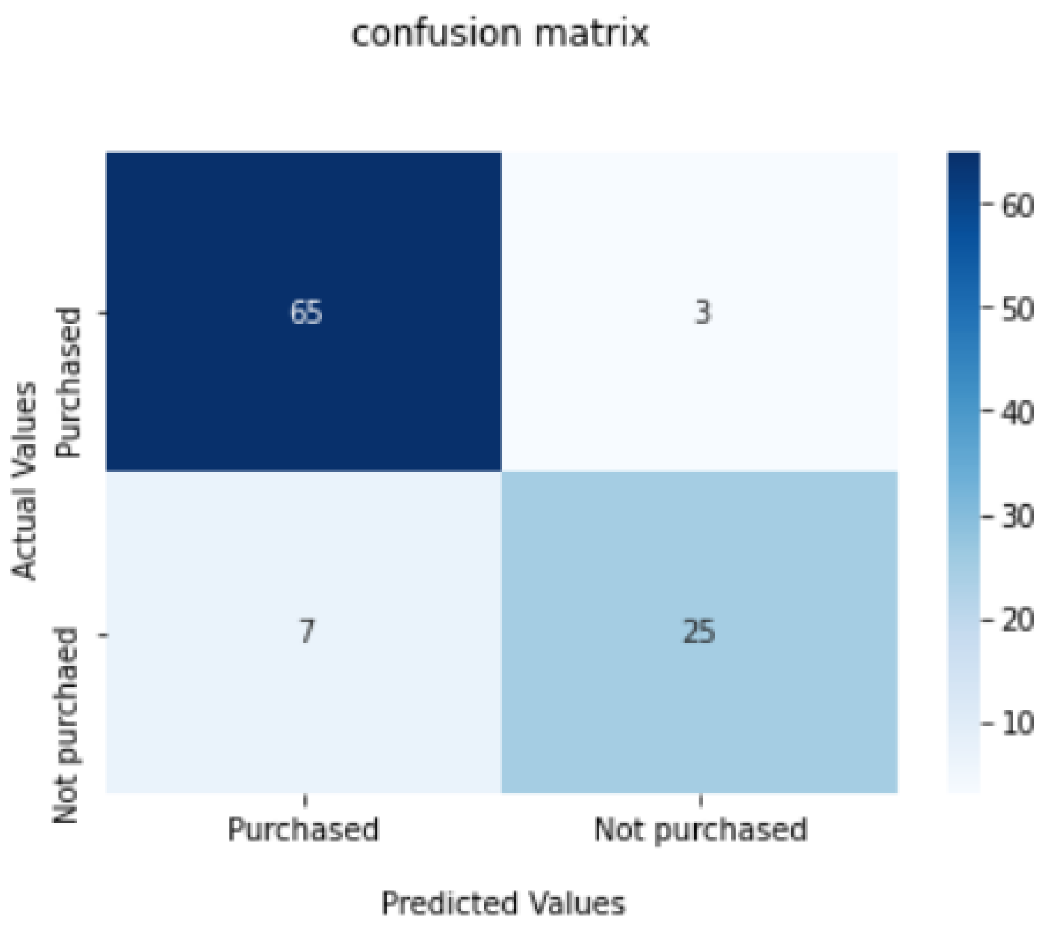
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 7 25]]
Out[19]: 0.9
```

# Visualising the Confusion Matrix

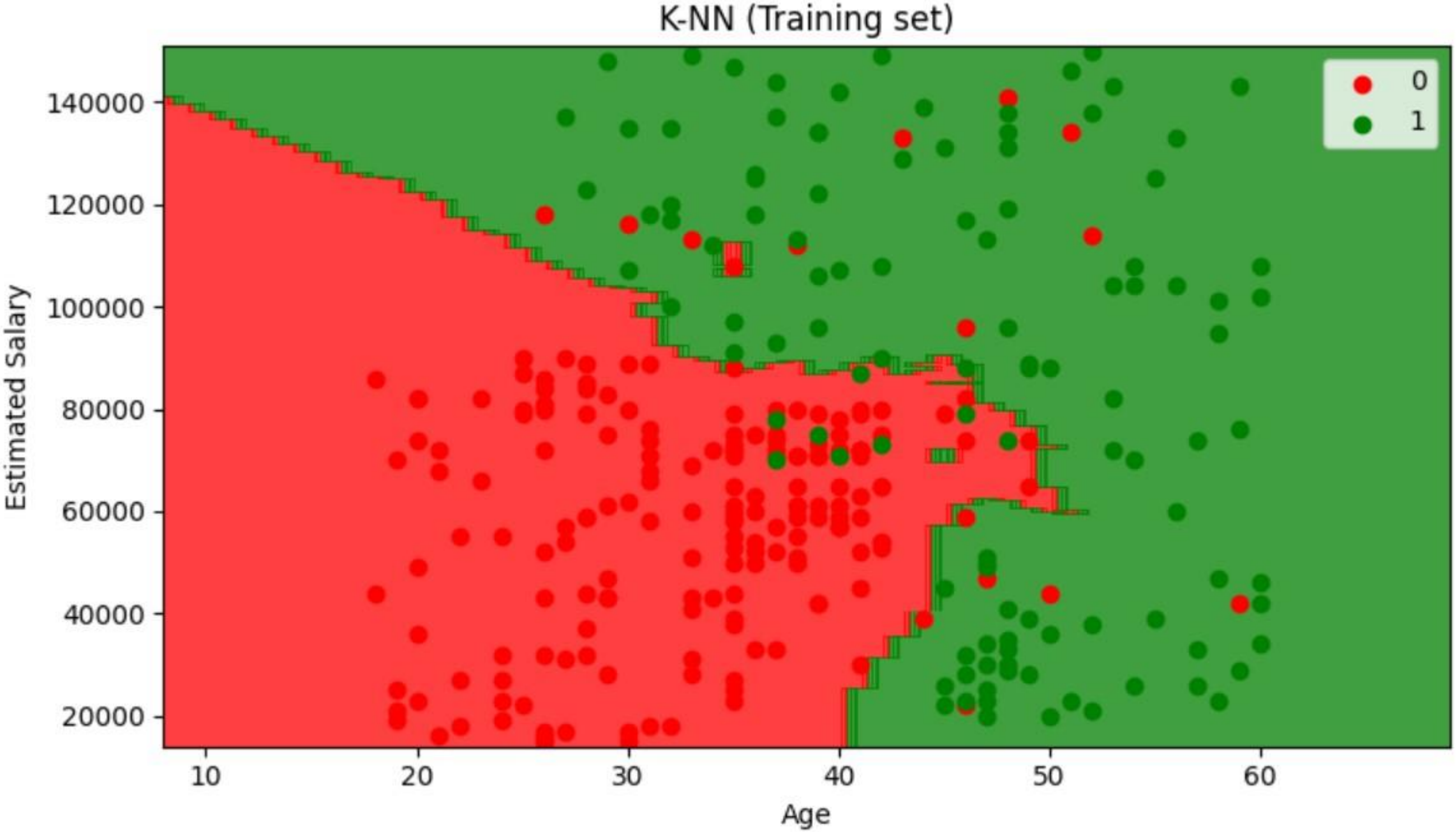
```
#visualizing confusion matrix
import seaborn as sns
ax = sns.heatmap(cm, annot=True, cmap='Blues')
ax.set_title('confusion matrix'+'\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Purchased','Not purchased'])
ax.yaxis.set_ticklabels(['Purchased','Not purchaed'])
## Display the visualization of the Confusion Matrix.
plt.show()
```

# Visualising the Confusion Matrix



# Visualization

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
#plt.show()
```


K-NN (Training set)
Estimated Salary
Age
0
1
140000
120000
100000
80000
60000
40000
20000
10
20
30
40
50
60

# Thank you!