Comparative Analysis:

Naive Bayes:

- **Introduction:**
  - Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem.
  - It assumes independence between features given the class label.
- **Method/Working:**
  - Calculates probabilities of each class based on the conditional probabilities of features.
  - Fast and simple, making it suitable for large datasets with high dimensionality.

Artificial Neural Network (ANN):

- **Introduction:**
  - ANN is a computational model inspired by the human brain's neural structure.
  - It is widely used for complex tasks, including image recognition and natural language processing.
- **Method/Working:**
  - Composed of interconnected nodes (neurons) organized in layers (input, hidden, output).
  - Employs backpropagation for training by adjusting weights to minimize error.

Random Forest:

- **Introduction:**
  - Random Forest is an ensemble learning method using multiple decision trees.
  - It provides robustness and reduces overfitting compared to individual trees.
- **Method/Working:**
  - Constructs multiple decision trees during training.
  - Makes predictions by aggregating the results of individual trees (voting or averaging).

k-Nearest Neighbors (KNN):

- **Introduction:**
  - KNN is a lazy learning algorithm for classification and regression.
  - It makes predictions based on the majority class or average of k nearest neighbors.
- **Method/Working:**
  - Memorizes the training dataset and performs computations at prediction time.
  - Sensitive to the choice of 'k' (number of neighbors).

Decision Tree (DT):

- **Introduction:**
  - Decision Tree is a tree-like model used for classification and regression.
  - It recursively splits the dataset based on the most significant attributes.
- **Method/Working:**
  - Builds a tree structure where each node represents a decision.
  - Eager learning, constructs the entire tree during the training phase.

Support Vector Machine (SVM):

- **Introduction:**
  - SVM is a powerful classification algorithm.
  - It finds the hyperplane that maximally separates classes in feature space.
- **Method/Working:**
  - Maps data into high-dimensional space to find the optimal hyperplane.
  - Effective in high-dimensional spaces, particularly in tasks with clear class separation.

```
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot  as plt
%matplotlib inline
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
print("Imported!")
```

```
Imported!
```

```
file = pd.read_csv("./heart-missing-classification-dataset.csv")
file.head()
```

	age	sex	chest_pain_type	resting_bp	cholestoral	fasting_blood_sugar	restecg	max_hr	exang	oldpeak	slope	num_major_vessel
0	63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.
1	37.0	1.0	NaN	130.0	250.0	0.0	1.0	187.0	0.0	3.5	0.0	0.
2	41.0	0.0	1.0	130.0	204.0	0.0	0.0	172.0	0.0	1.4	2.0	0.
3	56.0	1.0	1.0	120.0	NaN	0.0	1.0	178.0	0.0	NaN	2.0	Nal
4	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0

```
x = file.drop("target",axis=1)
y = file["target"]
```

```
import random
x.fillna(x.mean(),inplace=True) # x = x.fillna(x.mean())
y.fillna(random.randint(0,1),inplace=True) # y = y.fillna(random.randint(0,1))
y.isna().sum()
```

```
0
```

```
from sklearn.model_selection import train_test_split
# np.random.seed(50)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=50)
x_train
```

	age	sex	chest_pain_type	resting_bp	cholestoral	fasting_blood_sugar	restecg	max_hr	exang	oldpeak	slope	num_majo
168	63.0	1.0	0.0	130.0	254.0	0.0	0.0	147.000000	0.0	1.4	1.000000	
66	51.0	1.0	2.0	100.0	222.0	0.0	1.0	143.000000	1.0	1.2	1.000000	
148	44.0	1.0	2.0	120.0	226.0	0.0	1.0	169.000000	0.0	0.0	2.000000	
290	61.0	1.0	0.0	148.0	203.0	0.0	1.0	161.000000	0.0	0.0	2.000000	
222	65.0	1.0	3.0	138.0	282.0	1.0	0.0	174.000000	0.0	1.4	1.000000	
...	...	...	...	...	...	...	...	...	...	...	...	
70	54.0	1.0	2.0	120.0	258.0	0.0	0.0	147.000000	0.0	0.4	1.000000	
132	42.0	1.0	1.0	120.0	295.0	0.0	1.0	162.000000	0.0	0.0	1.390411	
289	55.0	0.0	0.0	128.0	205.0	0.0	2.0	130.000000	1.0	2.0	1.000000	
109	50.0	0.0	0.0	110.0	254.0	0.0	0.0	149.651568	0.0	0.0	2.000000	
176	60.0	1.0	0.0	117.0	230.0	1.0	1.0	160.000000	1.0	1.4	2.000000	

242 rows × 13 columns

```
#for naive bayes
from sklearn import naive_bayes
model_NB = naive_bayes.BernoulliNB()
model_NB.fit(x_train,y_train)
```

```
▾ BernoulliNB
BernoulliNB()
```

```
print(f"The model's Accuracy with Naive Bayes algorithm is: {model_NB.score(x_test,y_test)*100:.2f}%")
nb = model_NB.score(x_test,y_test)*100
```

```
The model's Accuracy with Naive Bayes algorithm is: 75.41%
```

```
from sklearn import svm

# Set the 'dual' parameter explicitly and increase 'max_iter'
model_SVM = svm.LinearSVC(dual=False, max_iter=100)  # You can adjust max_iter as needed

model_SVM.fit(x_train, y_train)
```

```
LinearSVC
```

```
print(f"The model's Accuracy with SVM algorithm is: {model_SVM.score(x_test,y_test)*100:.2f}%")
svm =model_SVM.score(x_test,y_test)*100
```

```
The model's Accuracy with SVM algorithm is: 72.13%
```

```
# using KNN

from sklearn import neighbors

model_KNN = neighbors.KNeighborsClassifier(n_neighbors=7) #this(n_neighbors) number can vary choose your number.

model_KNN.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
print(f"The model's Accuracy with KNN algorithm is: {model_KNN.score(x_test,y_test)*100:.2f}%")
knn = model_KNN.score(x_test,y_test)*100
```

```
The model's Accuracy with KNN algorithm is: 72.13%
```

```
# using Decision Tree

from sklearn import tree

model_DT = tree.DecisionTreeClassifier(max_depth=5,max_leaf_nodes=2) #number can vary, and is optional

model_DT.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, max_leaf_nodes=2)
```

```
print(f"The model's Accuracy with Decision Tree algorithm is: {model_DT.score(x_test,y_test)*100:.2f}%")
dt = model_DT.score(x_test,y_test)*100
```

```
The model's Accuracy with Decision Tree algorithm is: 68.85%
```

```
# Random forest

from sklearn.ensemble import RandomForestClassifier

model_RF = RandomForestClassifier(n_estimators=200) #optional can vary.

model_RF.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

```
print(f"The model's Accuracy with RandomForest Classifier algorithm is: {model_RF.score(x_test,y_test)*100:.2f}%")
rf = model_RF.score(x_test,y_test)*100
```

```
The model's Accuracy with RandomForest Classifier algorithm is: 70.49%
```

```
# ANN

np.random.seed(50)
# Build an ANN model
model = keras.Sequential([
    layers.Input(shape=(x_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
ann = test_accuracy*100
```

```
Epoch 1/50
8/8 [==============================] - 1s 27ms/step - loss: 50.9611 - accuracy: 0.5041 - val_loss: 35.1187 - val_accuracy: 0.508
Epoch 2/50
8/8 [==============================] - 0s 8ms/step - loss: 24.4259 - accuracy: 0.5041 - val_loss: 9.9913 - val_accuracy: 0.4754
Epoch 3/50
8/8 [==============================] - 0s 7ms/step - loss: 7.1887 - accuracy: 0.3967 - val_loss: 9.7982 - val_accuracy: 0.4590
Epoch 4/50
8/8 [==============================] - 0s 8ms/step - loss: 8.6073 - accuracy: 0.4463 - val_loss: 3.9155 - val_accuracy: 0.3934
Epoch 5/50
8/8 [==============================] - 0s 8ms/step - loss: 4.9363 - accuracy: 0.3760 - val_loss: 4.7325 - val_accuracy: 0.4918
Epoch 6/50
8/8 [==============================] - 0s 8ms/step - loss: 4.2742 - accuracy: 0.3802 - val_loss: 3.3513 - val_accuracy: 0.4262
Epoch 7/50
8/8 [==============================] - 0s 8ms/step - loss: 2.7712 - accuracy: 0.4132 - val_loss: 2.5979 - val_accuracy: 0.5082
Epoch 8/50
8/8 [==============================] - 0s 7ms/step - loss: 2.0394 - accuracy: 0.4587 - val_loss: 1.9012 - val_accuracy: 0.4590
Epoch 9/50
8/8 [==============================] - 0s 8ms/step - loss: 1.2753 - accuracy: 0.5041 - val_loss: 1.0193 - val_accuracy: 0.5082
Epoch 10/50
8/8 [==============================] - 0s 8ms/step - loss: 0.8588 - accuracy: 0.5702 - val_loss: 1.1028 - val_accuracy: 0.5738
Epoch 11/50
8/8 [==============================] - 0s 8ms/step - loss: 0.7817 - accuracy: 0.6198 - val_loss: 0.7749 - val_accuracy: 0.6885
Epoch 12/50
8/8 [==============================] - 0s 8ms/step - loss: 0.6499 - accuracy: 0.7107 - val_loss: 0.8232 - val_accuracy: 0.6230
Epoch 13/50
8/8 [==============================] - 0s 8ms/step - loss: 0.6002 - accuracy: 0.7314 - val_loss: 0.8059 - val_accuracy: 0.6230
Epoch 14/50
8/8 [==============================] - 0s 8ms/step - loss: 0.5705 - accuracy: 0.7231 - val_loss: 0.7261 - val_accuracy: 0.7541
Epoch 15/50
8/8 [==============================] - 0s 7ms/step - loss: 0.5622 - accuracy: 0.7149 - val_loss: 0.6914 - val_accuracy: 0.6393
Epoch 16/50
8/8 [==============================] - 0s 8ms/step - loss: 0.5436 - accuracy: 0.7686 - val_loss: 0.7711 - val_accuracy: 0.6230
Epoch 17/50
8/8 [==============================] - 0s 8ms/step - loss: 0.5380 - accuracy: 0.7397 - val_loss: 0.6869 - val_accuracy: 0.6230
Epoch 18/50
8/8 [==============================] - 0s 8ms/step - loss: 0.5511 - accuracy: 0.7562 - val_loss: 0.6907 - val_accuracy: 0.6393
Epoch 19/50
8/8 [==============================] - 0s 11ms/step - loss: 0.5152 - accuracy: 0.7686 - val_loss: 0.6854 - val_accuracy: 0.7377
Epoch 20/50
8/8 [==============================] - 0s 7ms/step - loss: 0.5088 - accuracy: 0.7769 - val_loss: 0.7223 - val_accuracy: 0.6393
Epoch 21/50
8/8 [==============================] - 0s 10ms/step - loss: 0.5747 - accuracy: 0.7314 - val_loss: 0.7568 - val_accuracy: 0.6230
Epoch 22/50
8/8 [==============================] - 0s 9ms/step - loss: 0.5470 - accuracy: 0.7190 - val_loss: 0.6629 - val_accuracy: 0.6885
Epoch 23/50
8/8 [==============================] - 0s 9ms/step - loss: 0.5388 - accuracy: 0.7397 - val_loss: 0.6759 - val_accuracy: 0.7377
Epoch 24/50
8/8 [==============================] - 0s 7ms/step - loss: 0.5005 - accuracy: 0.7603 - val_loss: 0.6719 - val_accuracy: 0.6885
Epoch 25/50
8/8 [==============================] - 0s 8ms/step - loss: 0.5017 - accuracy: 0.7686 - val_loss: 0.6555 - val_accuracy: 0.7049
Epoch 26/50
8/8 [==============================] - 0s 9ms/step - loss: 0.4956 - accuracy: 0.7645 - val_loss: 0.6627 - val_accuracy: 0.7049
Epoch 27/50
8/8 [==============================] - 0s 9ms/step - loss: 0.4960 - accuracy: 0.7645 - val_loss: 0.6507 - val_accuracy: 0.6885
Epoch 28/50
8/8 [==============================] - 0s 8ms/step - loss: 0.5008 - accuracy: 0.7603 - val_loss: 0.6947 - val_accuracy: 0.6885
Epoch 29/50
```

```
model_scores = {
    'Model': ['Naive Bayes', 'SVM', 'KNN', 'Decision Tree', 'Random Forest','ANN'],
    'Score': [nb, svm, knn, dt, rf,ann]
}

scores_df = pd.DataFrame(model_scores)

scores_df
```

	Model	Score
0	Naive Bayes	75.409836
1	SVM	72.131148
2	KNN	72.131148

```
plt.figure(figsize=(10, 6))
plt.bar(scores_df['Model'], scores_df['Score'], color='darkred')
plt.xlabel('Models')
plt.ylabel('Accuracy Score')
plt.title('Comparison of Model Scores')
plt.ylim(0, 100)  # Assuming scores are between 0 and 1 (accuracy)
plt.show()
```