```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.datasets import make_classification

dataset = pd.read_csv('sonar.csv')

X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Split the dataset into Training and Test groups (use 20-80 split, i.e. 20% of data will be used for the Test group and 80% for training).

```python
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier (with nb_trees = 10)
- K- Nearest Neighbors (K-NN)
- Support Vector Machine (SVM) – use the ‘rbf’ kernel

Train each of the above models and make predictions.

```python
logistic_regression = LogisticRegression(random_state=42)
decision_tree = DecisionTreeClassifier(random_state=42)
random_forest = RandomForestClassifier(n_estimators=10,
random_state=42)
knn = KNeighborsClassifier()
naive_bayes = GaussianNB()
svm = SVC(kernel='rbf', random_state=42)

logistic_regression.fit(X_train_scaled, y_train)
decision_tree.fit(X_train_scaled, y_train)
```

```
random_forest.fit(X_train_scaled, y_train)
knn.fit(X_train_scaled, y_train)
naive_bayes.fit(X_train_scaled, y_train)
svm.fit(X_train_scaled, y_train)

y_pred_logistic_regression =
logistic_regression.predict(X_test_scaled)
y_pred_decision_tree = decision_tree.predict(X_test_scaled)
y_pred_random_forest = random_forest.predict(X_test_scaled)
y_pred_knn = knn.predict(X_test_scaled)
y_pred_naive_bayes = naive_bayes.predict(X_test_scaled)
y_pred_svm = svm.predict(X_test_scaled)

results = pd.DataFrame({
    'Actual': y_test,
    'Logistic Regression': y_pred_logistic_regression,
    'Decision Tree': y_pred_decision_tree,
    'Random Forest': y_pred_random_forest,
    'K-NN': y_pred_knn,
    'Naive Bayes': y_pred_naive_bayes,
    'SVM': y_pred_svm
})

print(results)
```

```
    Actual Logistic Regression Decision Tree Random Forest  K-NN Naive
Bayes  \
161   Mine                Mine          Mine          Mine  Mine
Mine
15    Rock                Rock          Mine          Mine  Rock
Rock
73    Rock                Mine          Rock          Rock  Rock
Rock
96    Rock                Rock          Rock          Rock  Rock
Rock
166   Mine                Mine          Mine          Mine  Mine
Mine
9     Rock                Mine          Mine          Rock  Rock
Rock
100   Mine                Mine          Mine          Mine  Mine
Mine
135   Mine                Mine          Mine          Mine  Mine
Mine
18    Rock                Rock          Rock          Rock  Rock
Rock
148   Mine                Rock          Mine          Mine  Rock
Mine
171   Mine                Mine          Mine          Mine  Mine
Mine
30    Rock                Rock          Mine          Rock  Rock
```

```
Rock
155   Mine                Rock          Rock          Rock  Mine
Rock
180   Mine                Mine          Mine          Mine  Mine
Mine
125   Mine                Mine          Mine          Mine  Mine
Mine
197   Mine                Mine          Mine          Mine  Mine
Rock
164   Mine                Mine          Mine          Mine  Mine
Rock
190   Mine                Mine          Mine          Mine  Mine
Rock
84    Rock                Rock          Rock          Mine  Rock
Rock
75    Rock                Rock          Rock          Rock  Rock
Rock
124   Mine                Mine          Mine          Mine  Mine
Mine
170   Mine                Mine          Rock          Rock  Mine
Rock
104   Mine                Rock          Mine          Mine  Mine
Mine
101   Mine                Mine          Mine          Mine  Mine
Mine
69    Rock                Rock          Rock          Rock  Rock
Rock
25    Rock                Rock          Rock          Rock  Rock
Rock
95    Rock                Rock          Mine          Rock  Rock
Rock
16    Rock                Rock          Mine          Mine  Mine
Mine
141   Mine                Mine          Mine          Mine  Mine
Mine
185   Mine                Mine          Mine          Mine  Mine
Mine
154   Mine                Rock          Rock          Mine  Rock
Rock
68    Rock                Rock          Rock          Rock  Rock
Rock
66    Rock                Rock          Rock          Rock  Rock
Rock
120   Mine                Mine          Rock          Mine  Mine
Rock
147   Mine                Mine          Rock          Mine  Mine
Mine
98    Mine                Mine          Mine          Mine  Mine
Mine
```

138	Mine	Mine	Mine	Mine	Mine	Mine
167	Mine	Mine	Rock	Rock	Rock	Rock
45	Rock	Rock	Rock	Rock	Rock	Rock
113	Mine	Rock	Mine	Mine	Mine	Rock
65	Rock	Rock	Rock	Rock	Rock	Rock
178	Mine	Rock	Rock	Rock	Mine	Rock

	SVM
161	Mine
15	Rock
73	Rock
96	Rock
166	Mine
9	Rock
100	Mine
135	Mine
18	Rock
148	Mine
171	Mine
30	Rock
155	Rock
180	Mine
125	Mine
197	Mine
164	Mine
190	Mine
84	Rock
75	Rock
124	Mine
170	Mine
104	Mine
101	Mine
69	Rock
25	Rock
95	Rock
16	Mine
141	Mine
185	Mine
154	Rock
68	Rock
66	Rock
120	Mine
147	Mine

```
98   Mine
138  Mine
167  Rock
45   Rock
113  Mine
65   Rock
178  Rock
```

Create and print the Confusion Matrix and the accuracy scores for each model.

```
def print_evaluation(model_name, y_true, y_pred):
    print(f"===== {model_name} =====")
    cm = confusion_matrix(y_true, y_pred)
    print(f"Confusion Matrix:\n{cm}")

    acc_score = accuracy_score(y_true, y_pred)
    print(f"Accuracy Score: {acc_score:.4f}\n")

print_evaluation("Logistic Regression", y_test,
y_pred_logistic_regression)
print_evaluation("Decision Tree", y_test, y_pred_decision_tree)
print_evaluation("Random Forest", y_test, y_pred_random_forest)
print_evaluation("K-NN", y_test, y_pred_knn)
print_evaluation("Naive Bayes", y_test, y_pred_naive_bayes)
print_evaluation("SVM", y_test, y_pred_svm)

===== Logistic Regression =====
Confusion Matrix:
[[20  6]
 [ 2 14]]
Accuracy Score: 0.8095

===== Decision Tree =====
Confusion Matrix:
[[19  7]
 [ 5 11]]
Accuracy Score: 0.7143

===== Random Forest =====
Confusion Matrix:
[[22  4]
 [ 3 13]]
Accuracy Score: 0.8333

===== K-NN =====
Confusion Matrix:
[[23  3]
 [ 1 15]]
Accuracy Score: 0.9048
```

```
===== Naive Bayes =====
Confusion Matrix:
[[16 10]
 [ 1 15]]
Accuracy Score: 0.7381

===== SVM =====
Confusion Matrix:
[[22  4]
 [ 1 15]]
Accuracy Score: 0.8810
```

For each model, use the K-fold cross-validation (use K=10; in python - cv=10).

```
def cross_val_and_print(model, model_name):
    scores = cross_val_score(model, X_train_scaled, y_train, cv=10,
scoring='accuracy')

    print(f"===== {model_name} (K-fold Cross-Validation) =====")
    print(f"Accuracy Scores: {scores}")
    print(f"Mean Accuracy: {scores.mean():.4f}")
    print(f"Standard Deviation: {scores.std():.4f}\n")

cross_val_and_print(logistic_regression, "Logistic Regression")
cross_val_and_print(decision_tree, "Decision Tree")
cross_val_and_print(random_forest, "Random Forest")
cross_val_and_print(knn, "K-NN")
cross_val_and_print(naive_bayes, "Naive Bayes")
cross_val_and_print(svm, "SVM")

===== Logistic Regression (K-fold Cross-Validation) =====
Accuracy Scores: [0.76470588 0.52941176 1.         0.82352941
0.76470588 0.70588235
 0.8125     0.5625     0.875      0.8125    ]
Mean Accuracy: 0.7651
Standard Deviation: 0.1324

===== Decision Tree (K-fold Cross-Validation) =====
Accuracy Scores: [0.88235294 0.70588235 0.64705882 0.70588235
0.76470588 0.70588235
 0.875      0.6875     0.75       0.8125    ]
Mean Accuracy: 0.7537
Standard Deviation: 0.0758

===== Random Forest (K-fold Cross-Validation) =====
Accuracy Scores: [0.82352941 0.76470588 0.70588235 0.82352941
0.88235294 0.76470588
 0.6875     0.8125     0.75       0.8125    ]
Mean Accuracy: 0.7827
```

```
Standard Deviation: 0.0563

===== K-NN (K-fold Cross-Validation) =====
Accuracy Scores: [0.94117647 0.58823529 0.70588235 0.82352941
0.64705882 0.76470588
 0.8125     0.8125     0.875      0.75      ]
Mean Accuracy: 0.7721
Standard Deviation: 0.0997

===== Naive Bayes (K-fold Cross-Validation) =====
Accuracy Scores: [0.82352941 0.64705882 0.88235294 0.58823529
0.58823529 0.76470588
 0.625      0.625      0.8125     0.5625    ]
Mean Accuracy: 0.6919
Standard Deviation: 0.1107

===== SVM (K-fold Cross-Validation) =====
Accuracy Scores: [0.94117647 0.76470588 0.94117647 0.82352941
0.82352941 0.88235294
 0.8125     0.8125     0.8125     0.8125    ]
Mean Accuracy: 0.8426
Standard Deviation: 0.0561
```

For the chosen best performing model select two hyperparameters you would like to tune.

```
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

gnb = GaussianNB()

param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}

grid_search = GridSearchCV(estimator=gnb, param_grid=param_grid, cv=5,
scoring='accuracy')

grid_search.fit(X_train, y_train)

print("Best Parameters: ", grid_search.best_params_)
print("Best Accuracy: {:.2f}".format(grid_search.best_score_))


Best Parameters:  {'var_smoothing': 1e-09}
Best Accuracy: 0.85
```

After tuning the hyperparameter, we can see that the model performed slightly better, however the value of var_smoothing is the same as the default one used - 1e-09