In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Read and check data

In [ ]:
```
df = pd.read_csv('data.csv')
df.head(5)
```

Out[ ]:

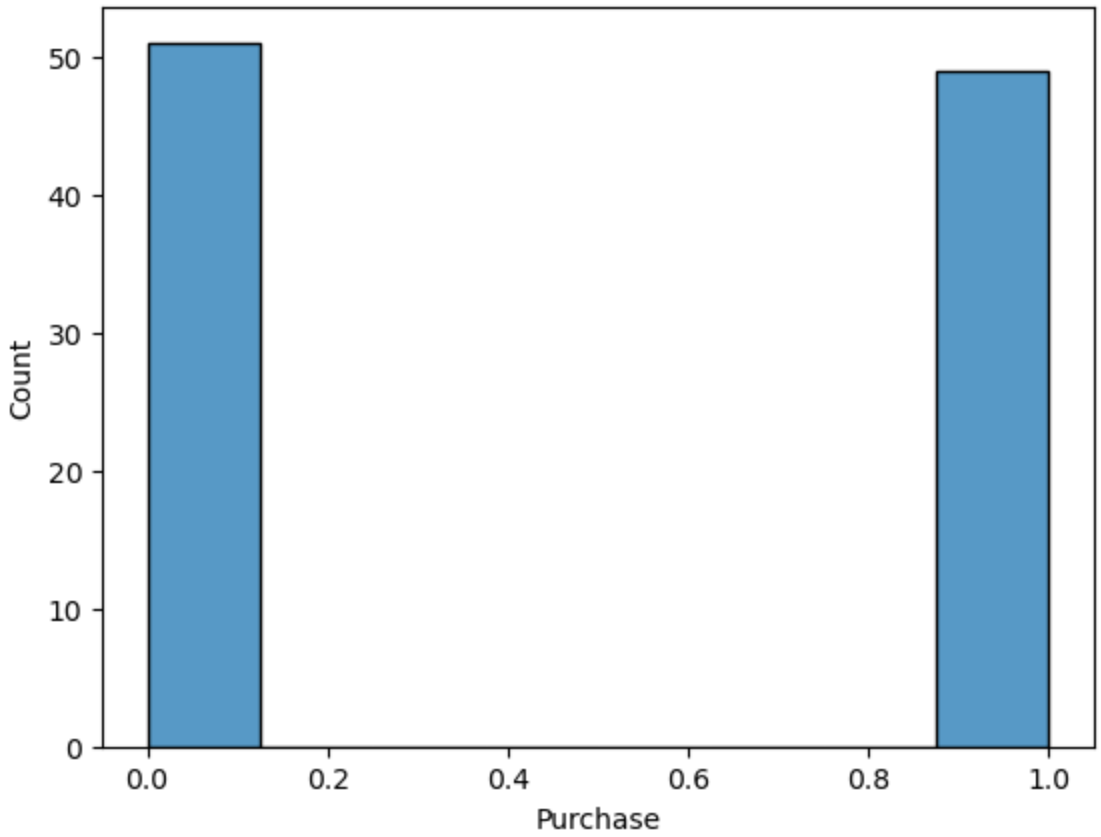
	Purchase	SUS	Duration	Gender	ASR_Error	Intent_Error
0	1	84	254	0	3	2
1	0	58	247	0	6	9
2	0	56	125	1	6	8
3	0	55	22	0	11	7
4	1	95	262	0	2	3

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
Purchase        0
SUS             0
Duration        0
Gender          0
ASR_Error       0
Intent_Error    0
dtype: int64
```

See distribution of Purchases

In [ ]:
```
# histogram of System Usability Survery (SUS) score
sns.histplot(data=df['Purchase'], palette='bright')
plt.show()
```



Prep and label data for model training

In [ ]:
```
# label
y = df['Purchase'].to_numpy()
y
```

Out[ ]:
```
array([1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1])
```

In [ ]:
```
# data
X = df.drop('Purchase', axis = 1).to_numpy()
X
```

Out[ ]:
```
array([[ 84, 254,   0,   3,   2],
       [ 58, 247,   0,   6,   9],
       [ 56, 125,   1,   6,   8],
       [ 55,  22,   0,  11,   7],
       [ 95, 262,   0,   2,   3],
       [ 71, 113,   1,   8,   9],
       [ 76,  91,   1,   6,   3],
       [ 64,  46,   0,   6,  11],
       [ 56, 132,   0,   9,   9],
       [ 96, 190,   0,  11,   4],
       [ 56, 146,   0,   9,   7],
       [ 80, 226,   0,   4,   4],
       [ 72,  54,   1,   7,   9],
       [ 81, 104,   0,   2,   3],
       [ 72, 292,   0,   7,   3],
       [ 66, 126,   0,   8,   9],
       [ 89, 152,   1,   3,   5],
       [ 80, 221,   0,   3,   4],
       [ 66,  51,   0,   9,   9],
       [ 74, 230,   0,   6,   8],
       [ 74,  72,   1,   6,   8],
       [ 68, 284,   0,   6,  11],
       [ 96, 168,   0,   2,   4],
       [ 98, 194,   0,   1,   3],
       [ 86, 130,   1,  10,   1],
       [ 58, 227,   0,   9,   3],
       [ 75, 161,   1,   7,   8],
       [ 86, 262,   1,   2,   3],
       [ 65, 257,   1,   9,  11],
       [ 80,  57,   1,   8,   3],
       [ 68, 138,   0,   8,   9],
       [ 81,  33,   1,   9,   4],
       [ 95, 260,   0,   1,   2],
       [ 82, 178,   1,   3,   4],
       [ 95, 153,   0,   2,   4],
       [ 87, 151,   0,   2,   3],
       [ 69,  67,   0,   8,   9],
       [ 86, 284,   1,   2,   2],
       [ 98,  96,   1,   3,   1],
       [ 98,  31,   0,   1,   3],
       [ 50, 100,   0,  12,   9],
       [ 86, 200,   0,   2,   3],
       [ 98,  24,   1,   1,   2],
       [ 95, 169,   0,   1,   3],
       [ 80, 110,   0,   1,   3],
       [ 86, 184,   1,   2,   3],
       [ 86, 234,   1,   2,   5],
       [ 82,  90,   1,   5,   3],
       [ 87, 202,   1,   3,   3],
       [ 84, 254,   0,   3,   3],
       [ 60, 259,   0,   9,   6],
       [ 92, 193,   1,  11,   5],
       [ 80, 258,   0,   6,   5],
       [ 56, 117,   1,   6,   7],
       [ 81, 297,   1,   3,   4],
       [ 50, 218,   1,  12,   9],
```

```
[ 56, 119,   0,   0,   8],
[ 70,  54,   1,   8,   6],
[ 80, 143,   1,   0,   4],
[ 58, 300,   1,   9,  11],
[ 88,  43,   0,   3,   4],
[ 40,  28,   0,   7,   6],
[ 57, 357,   1,   9,   4],
[ 70,  21,   0,  13,   7],
[ 65, 100,   0,   7,   9],
[ 76, 303,   1,   9,   8],
[ 55,  56,   1,   6,  10],
[ 73, 205,   0,   5,  12],
[ 87, 104,   0,   4,   1],
[ 90, 342,   0,   1,   3],
[ 78,  78,   0,   3,   2],
[ 75,  56,   0,   6,   5],
[ 80, 297,   1,   2,   1],
[ 90,  56,   0,   5,   4],
[ 85, 365,   0,   4,   3],
[ 75, 275,   0,  11,   6],
[ 43, 198,   1,   9,   9],
[ 57, 224,   0,  12,  12],
[ 65, 326,   0,   7,   7],
[ 55,  23,   0,  14,   9],
[ 71,  78,   1,   9,  10],
[ 67, 108,   0,   6,  11],
[ 43, 326,   0,  13,   7],
[ 59,  61,   0,  10,   8],
[ 70,  48,   1,   1,   5],
[ 63,  90,   0,  11,   4],
[ 89,  87,   1,   0,   1],
[ 90,  65,   1,   2,   2],
[ 92,  22,   1,   3,   0],
[ 64,  10,   1,   9,   9],
[ 66,  45,   0,   5,   8],
[ 83,  69,   1,   4,   0],
[ 80,  57,   0,   1,   1],
[ 92, 208,   1,   0,   4],
[ 65, 197,   0,  11,  10],
[ 57, 358,   0,  13,   7],
[ 93,  71,   0,   3,   0],
[ 80,  34,   1,   0,   9],
[ 82,  49,   1,   4,   1],
[ 78, 213,   0,   1,   4]])
```

Data Scaling

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.model_selection import train_test_split

scale = StandardScaler()
scaled_X = scale.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size =
```

Import Classifiers and Metrics from Sklearn

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix, roc_cu

import matplotlib.pyplot as plt
```

In [ ]:
```
lc = LogisticRegression()
knn = KNeighborsClassifier(n_neighbors=3)
svc = SVC(probability=True)
nbc = GaussianNB()
rfc = RandomForestClassifier()
```

Train ML model with training dataset with model.fit() function

In [ ]:
```
lc.fit(X_train, y_train)
knn.fit(X_train, y_train)
svc.fit(X_train, y_train)
nbc.fit(X_train, y_train)
rfc.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

Test model with test dataset

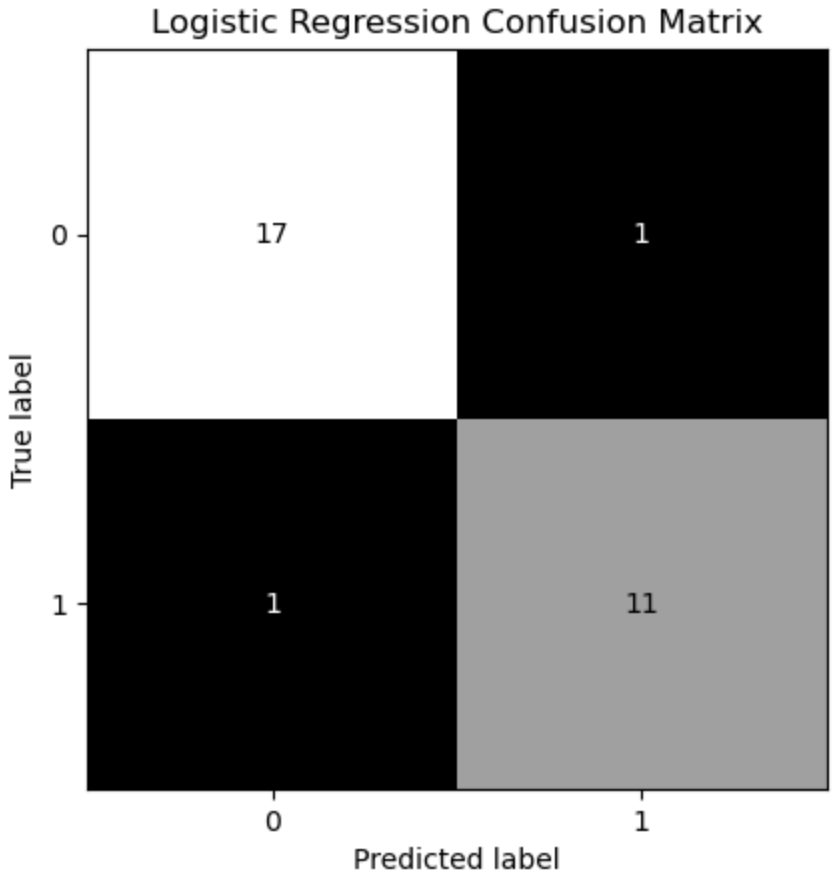
In [ ]:
```
y_lc_predicted = lc.predict(X_test)
y_lc_pred_proba = lc.predict_proba(X_test)

y_knn_predicted = knn.predict(X_test)
y_knn_pred_proba = knn.predict_proba(X_test)

y_svc_predicted = svc.predict(X_test)
y_svc_pred_proba = svc.predict_proba(X_test)

y_nbc_predicted = nbc.predict(X_test)
y_nbc_pred_proba = nbc.predict_proba(X_test)
```

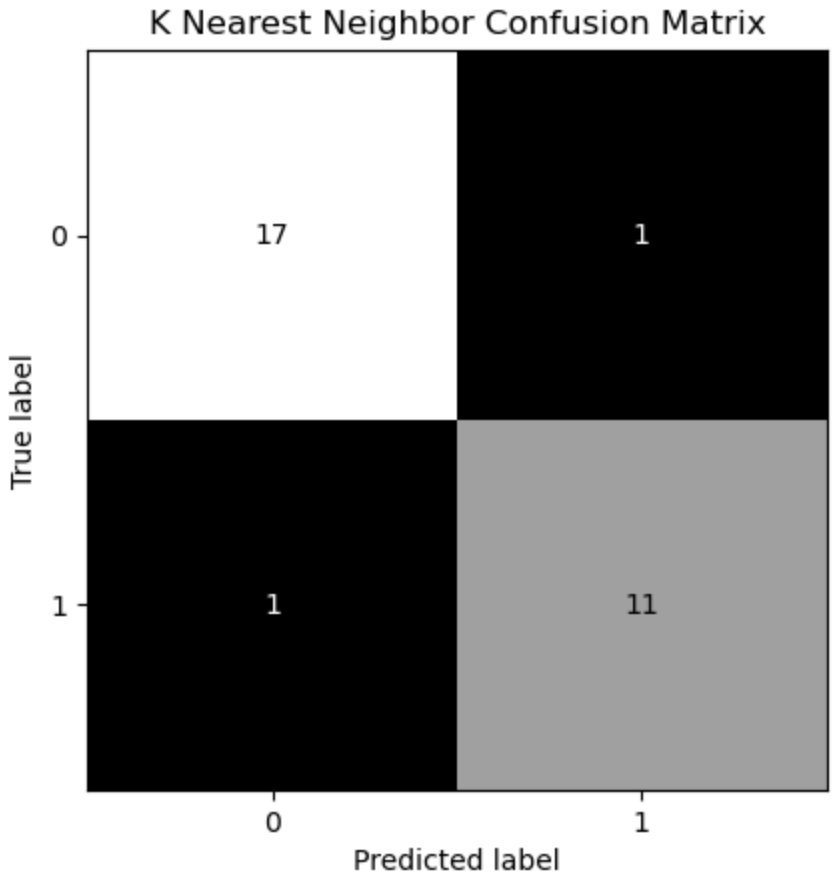
```
y_rfc_predicted = rfc.predict(X_test)
y_rfc_pred_proba = rfc.predict_proba(X_test)
```

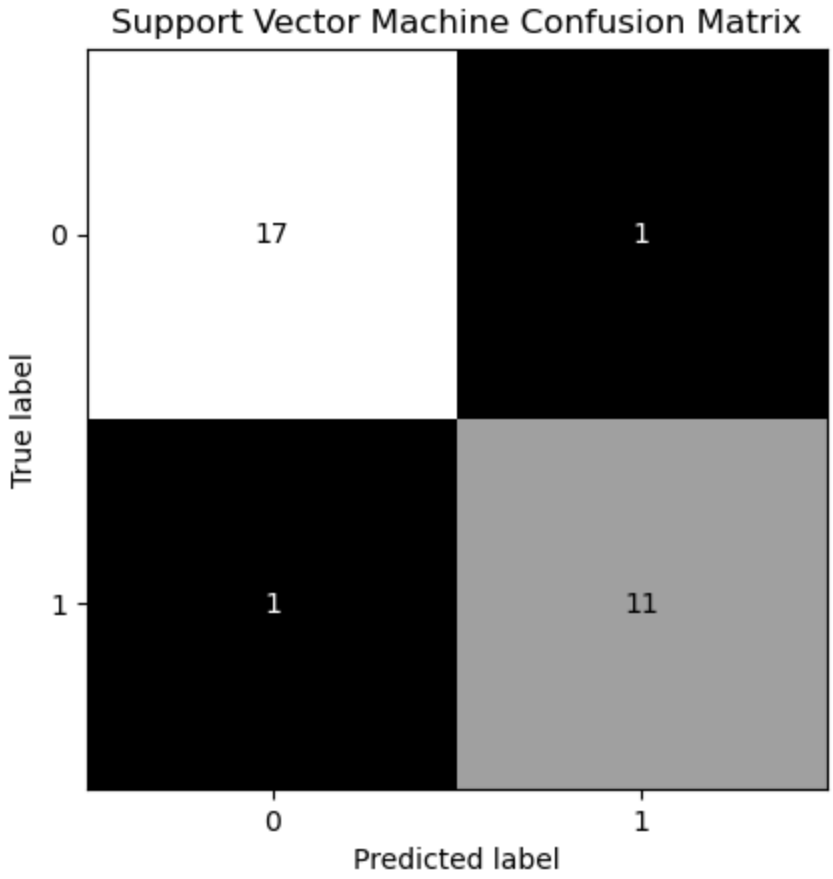
In [ ]:

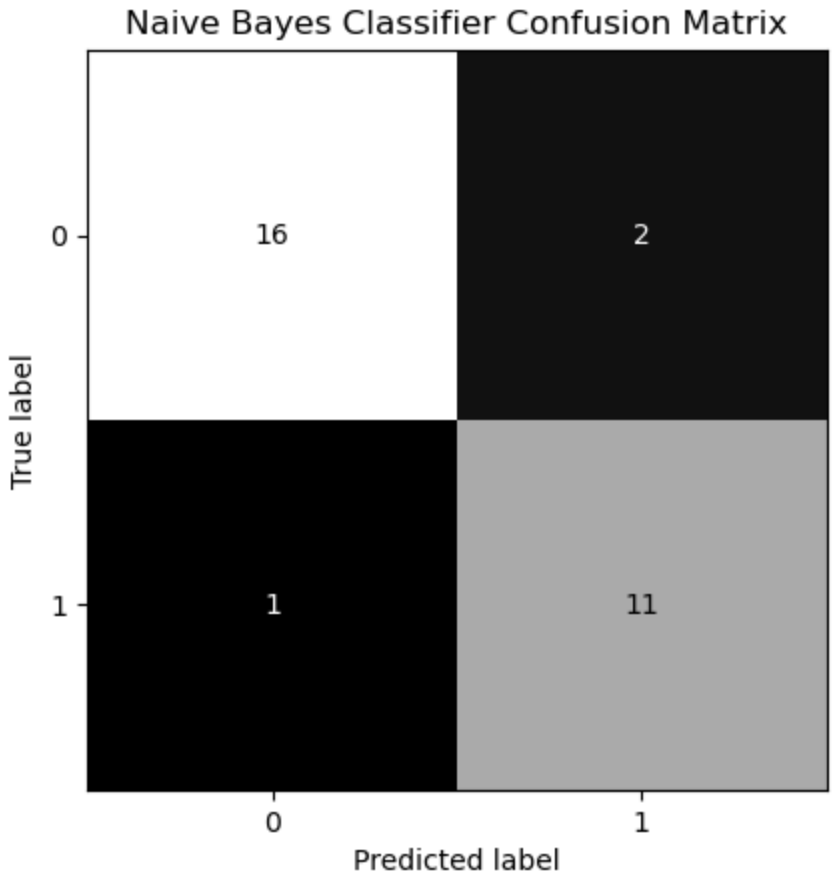
```
# model evaluation with confusion matrix and ROC curve

print(classification_report(y_test, y_lc_predicted))
print(classification_report(y_test, y_knn_predicted))
print(classification_report(y_test, y_svc_predicted))
print(classification_report(y_test, y_nbc_predicted))
print(classification_report(y_test, y_rfc_predicted))

models = ['Logistic Regression', 'K Nearest Neighbor', 'Support Vector Machi
predictions = [y_lc_predicted, y_knn_predicted, y_svc_predicted, y_nbc_predi
pred_probabilities = [y_lc_pred_proba, y_knn_pred_proba, y_svc_pred_proba, y

plot = 1

for model, prediction, pred_proba in zip(models, predictions, pred_probabili
    disp = ConfusionMatrixDisplay(confusion_matrix(y_test.ravel(), predictic
    disp.plot(
        include_values=True,
        cmap='gray',
        colorbar=False
    )
    disp.ax_.set_title(f"{model} Confusion Matrix")

plt.figure(figsize=(30, 15))
plt.suptitle("ROC Curves")
plot_index = 1

for model, prediction, pred_proba in zip(models, predictions, pred_probabili
    fpr, tpr, thresholds = roc_curve(y_test, pred_proba[:, 1])
    auc_score = auc(fpr, tpr)
    plt.subplot(3, 2, plot_index)
    plt.plot(fpr, tpr, 'r', label='ROC curve')
    # pyplot.figure(figsize=(5, 5))
    plt.title(f'Roc Curve - {model} - [AUC - {auc_score}]', fontsize=14)
    plt.xlabel('FPR', fontsize=12)
    plt.ylabel('TPR', fontsize=12)
    plt.legend()
    plot_index += 1
plt.show()
```

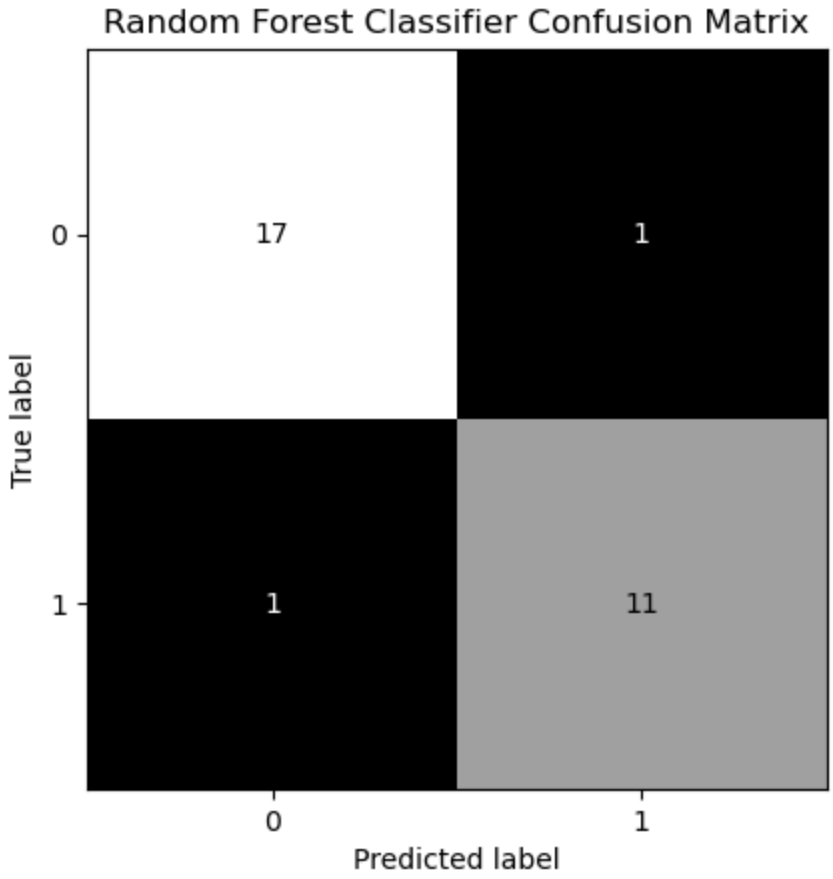
```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

              precision    recall  f1-score   support

           0       0.94      0.89      0.91        18
           1       0.85      0.92      0.88        12

    accuracy                           0.90        30
   macro avg       0.89      0.90      0.90        30
weighted avg       0.90      0.90      0.90        30

              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

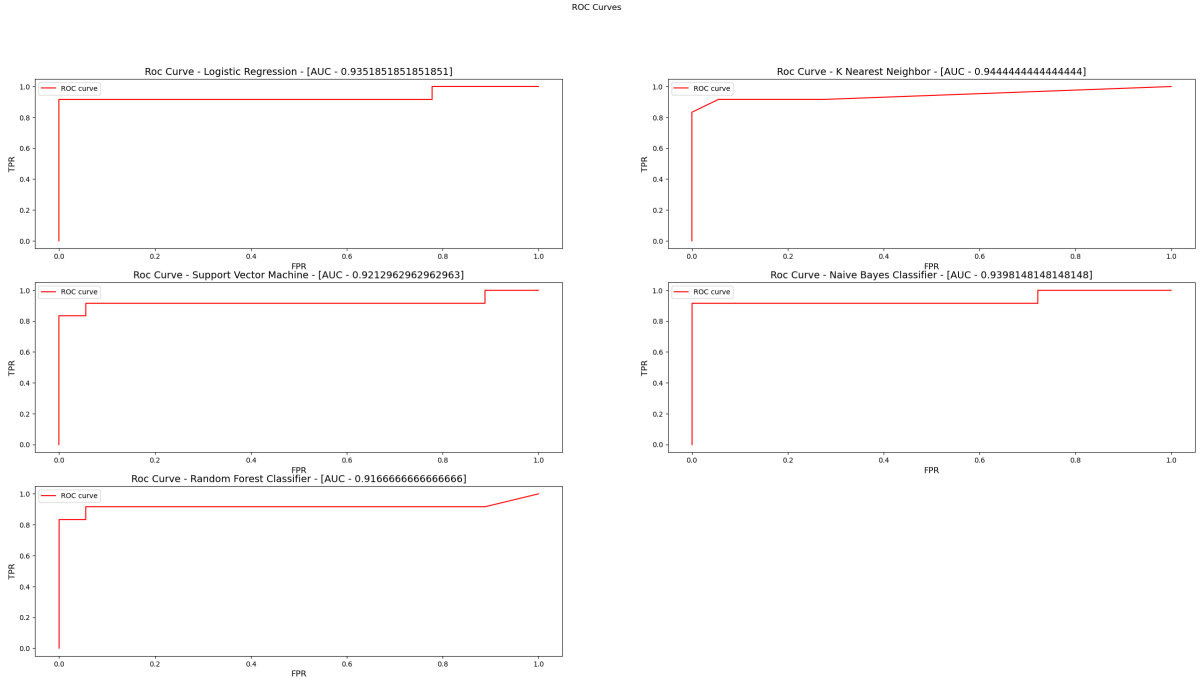

Logistic Regression Confusion Matrix
17
1
1
11
0
1
True label
0
1
Predicted label


K Nearest Neighbor Confusion Matrix
17
1
1
11
0
1
True label
0
1
Predicted label


Support Vector Machine Confusion Matrix
17
1
1
11
0
1
True label
0
1
Predicted label


Naive Bayes Classifier Confusion Matrix
16
2
1
11
0
1
True label
0
1
Predicted label





In [ ]:
```
# use SMOTE for imbalanced classification

from imblearn.over_sampling import SMOTE

oversample = SMOTE()
over_sampled_X_train, over_sampled_y_train = oversample.fit_resample(X_train
```

In [ ]:
```
lc = LogisticRegression()
knn = KNeighborsClassifier(n_neighbors=3)
svc = SVC(probability=True)
nbc = GaussianNB()
rfc = RandomForestClassifier()
```

In [ ]:
```
lc.fit(over_sampled_X_train, over_sampled_y_train)
knn.fit(over_sampled_X_train, over_sampled_y_train)
svc.fit(over_sampled_X_train, over_sampled_y_train)
nbc.fit(over_sampled_X_train, over_sampled_y_train)
rfc.fit(over_sampled_X_train, over_sampled_y_train)
```

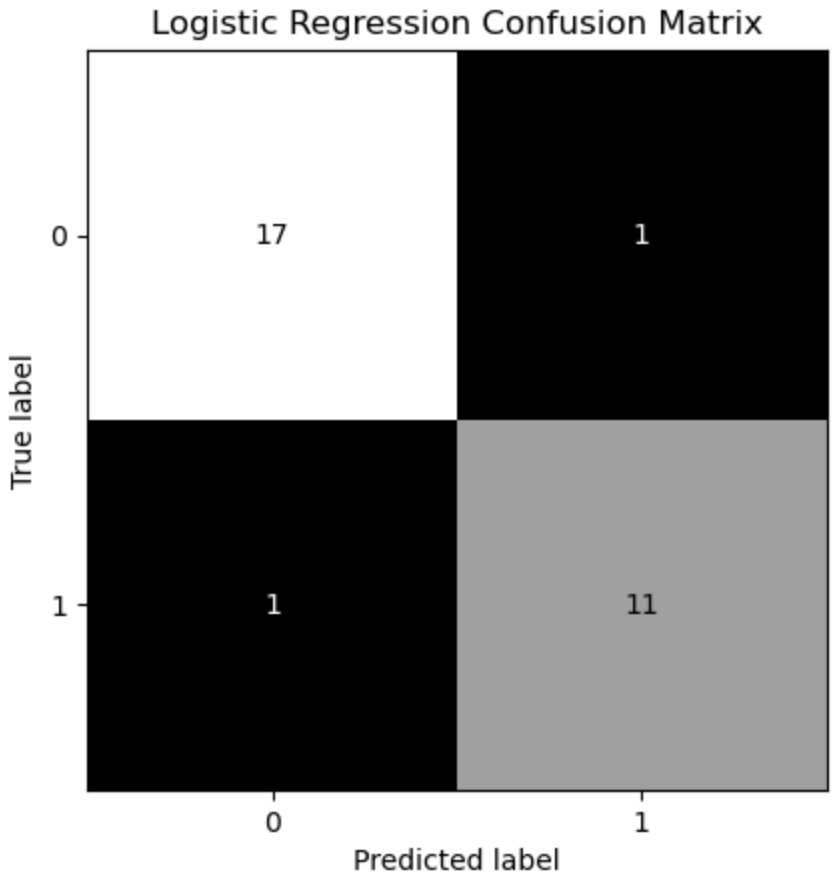
Out[ ]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

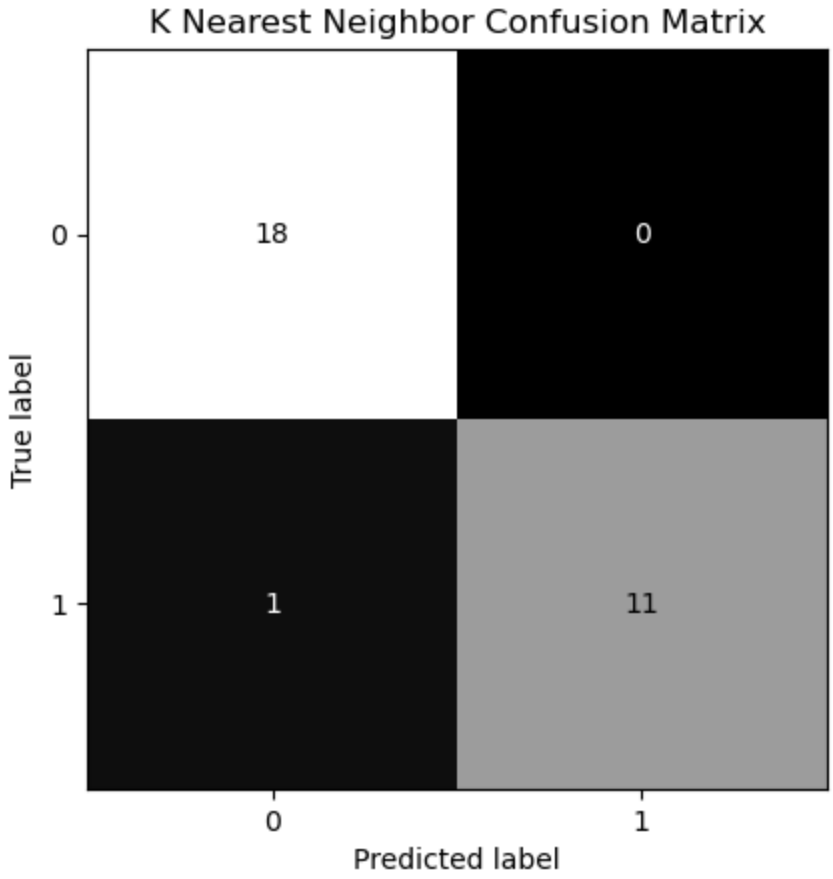
In [ ]:
```
y_lc_predicted = lc.predict(X_test)
y_lc_pred_proba = lc.predict_proba(X_test)

y_knn_predicted = neigh.predict(X_test)
y_knn_pred_proba = neigh.predict_proba(X_test)

y_svc_predicted = svc.predict(X_test)
y_svc_pred_proba = svc.predict_proba(X_test)

y_nbc_predicted = nbc.predict(X_test)
y_nbc_pred_proba = nbc.predict_proba(X_test)

y_rfc_predicted = rfc.predict(X_test)
y_rfc_pred_proba = rfc.predict_proba(X_test)

print(classification_report(y_test, y_lc_predicted))
print(classification_report(y_test, y_knn_predicted))
print(classification_report(y_test, y_svc_predicted))
print(classification_report(y_test, y_nbc_predicted))
print(classification_report(y_test, y_rfc_predicted))
```

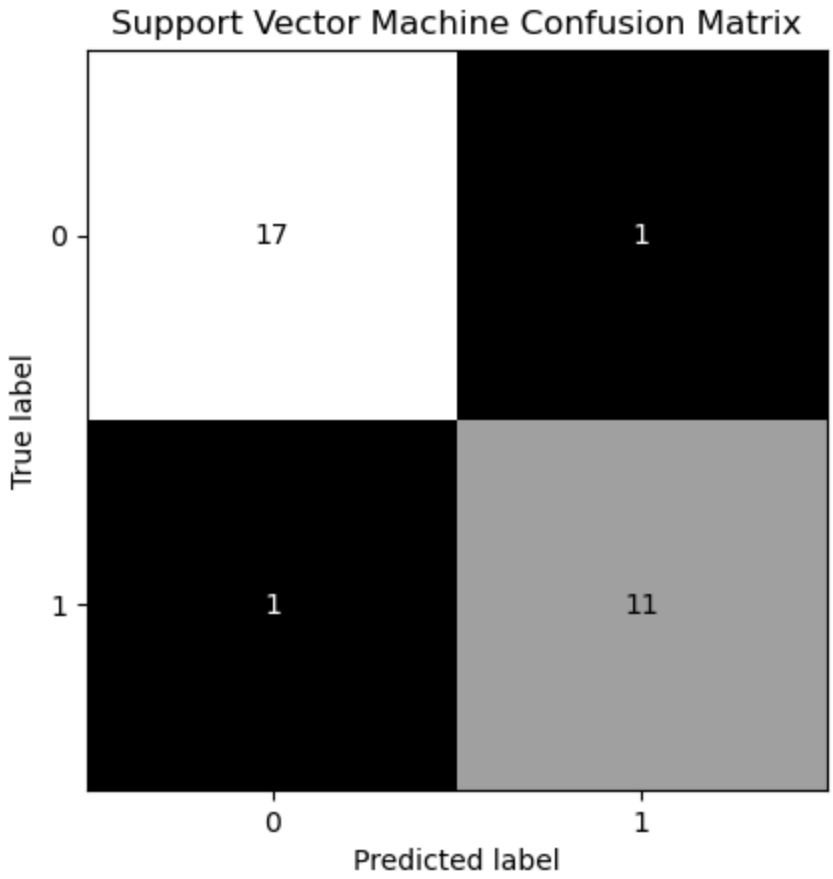
```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

              precision    recall  f1-score   support

           0       0.95      1.00      0.97        18
           1       1.00      0.92      0.96        12

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30

              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

              precision    recall  f1-score   support

           0       0.94      0.89      0.91        18
           1       0.85      0.92      0.88        12

    accuracy                           0.90        30
   macro avg       0.89      0.90      0.90        30
weighted avg       0.90      0.90      0.90        30

              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

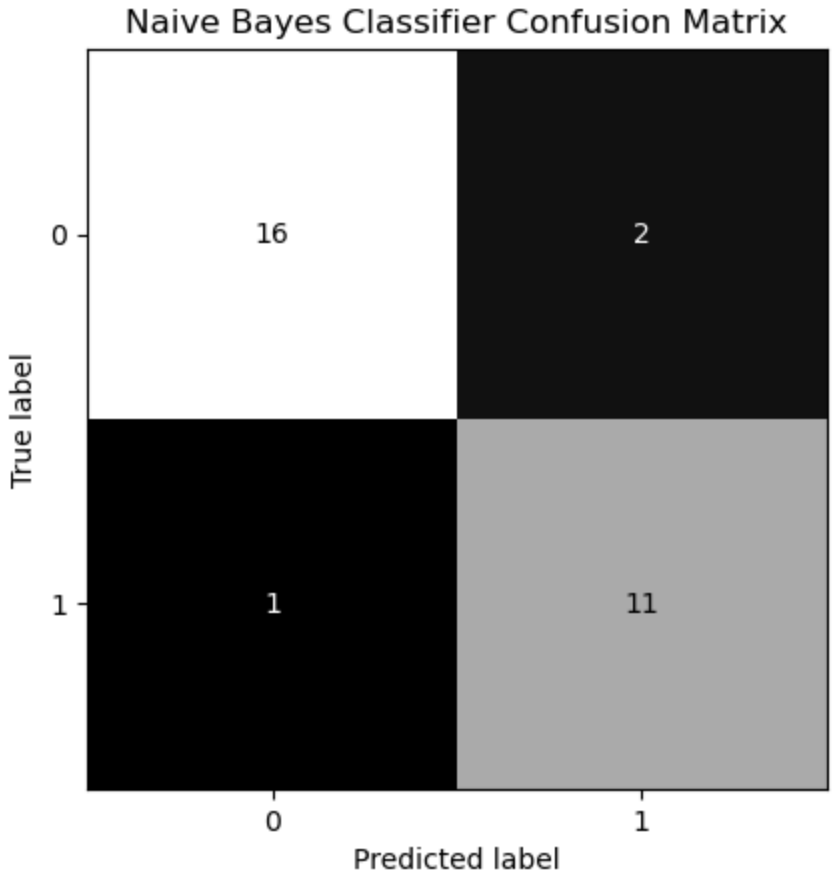
In [ ]:
```
models = ['Logistic Regression', 'K Nearest Neighbor', 'Support Vector Machi
predictions = [y_lc_predicted, y_knn_predicted, y_svc_predicted, y_nbc_predi
pred_probabilities = [y_lc_pred_proba, y_knn_pred_proba, y_svc_pred_proba, y

plot = 1

for model, prediction, pred_proba in zip(models, predictions, pred_probabili
    disp = ConfusionMatrixDisplay(confusion_matrix(y_test.ravel(), predictic
    disp.plot(
        include_values=True,
```

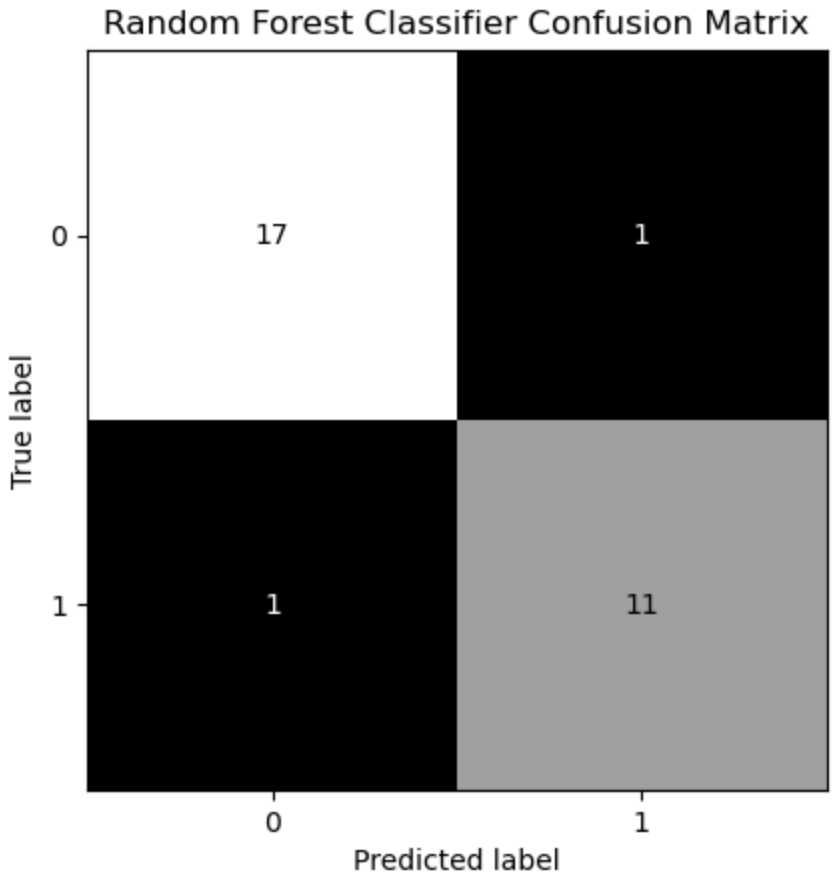
```
        cmap='gray',
        colorbar=False
    )
    disp.ax_.set_title(f"{model} Confusion Matrix")

plt.figure(figsize=(30, 15))
plt.suptitle("ROC Curves")
plot_index = 1

for model, prediction, pred_proba in zip(models, predictions, pred_probabili
    fpr, tpr, thresholds = roc_curve(y_test, pred_proba[:, 1])
    auc_score = auc(fpr, tpr)
    plt.subplot(3, 2, plot_index)
    plt.plot(fpr, tpr, 'r', label='ROC curve')
    # pyplot.figure(figsize=(5, 5))
    plt.title(f'Roc Curve - {model} - [AUC - {auc_score}]', fontsize=14)
    plt.xlabel('FPR', fontsize=12)
    plt.ylabel('TPR', fontsize=12)
    plt.legend()
    plot_index += 1
plt.show()
```



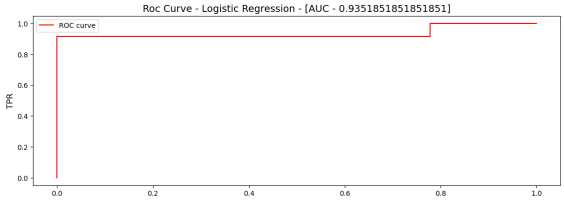

K Nearest Neighbor Confusion Matrix
18
0
1
11
0
1
True label
0
1
Predicted label

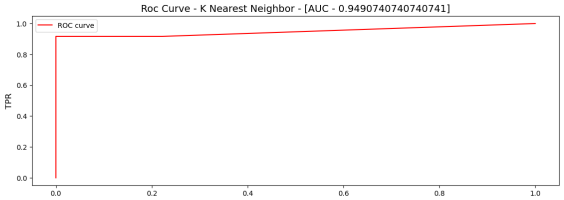

Support Vector Machine Confusion Matrix
17
1
1
11
0
1
True label
0
1
Predicted label

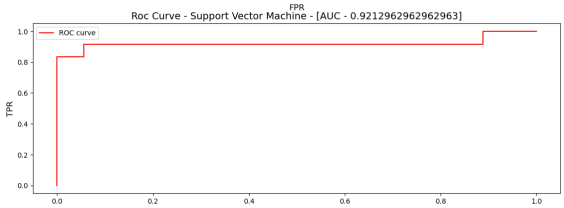

Naive Bayes Classifier Confusion Matrix
16
2
1
11
0
1
True label
0
1
Predicted label

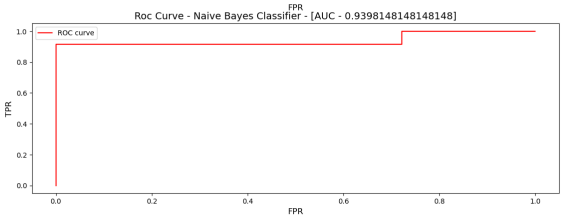

Random Forest Classifier Confusion Matrix
17
1
1
11
0
1
True label
0
1
Predicted label

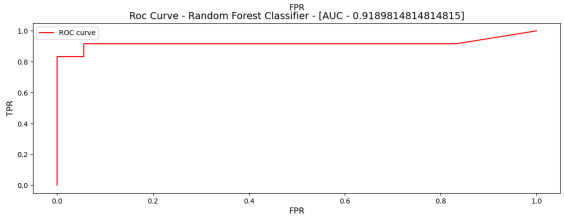
ROC Curves


Roc Curve - Logistic Regression - [AUC - 0.9351851851851851]
ROC curve
TPR
FPR


Roc Curve - K Nearest Neighbor - [AUC - 0.9490740740740741]
ROC curve
TPR
FPR


Roc Curve - Support Vector Machine - [AUC - 0.9212962962962963]
ROC curve
TPR
FPR


Roc Curve - Naive Bayes Classifier - [AUC - 0.9398148148148148]
ROC curve
TPR
FPR


Roc Curve - Random Forest Classifier - [AUC - 0.9189814814814815]
ROC curve
TPR
FPR