Download this dataset and do the following:

Part 1: Classify the 'Results' column using three models of your choice. At least one must get 90% accuracy with at least 50% precision with ten fold cross validation. Print your confsion matrix.

Read in the dataset:

```
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np

# load data
data = pd.read_csv('homework4.csv')


# separate features and target
X = data.drop(columns=['Results'])
# target
y = data['Results']

# split data into train and test sets for confusion matrix
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# use StratifiedKFold for 10-fold cross-validation
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
scoring = {'accuracy': 'accuracy', 'precision': 'average_precision'}
```

Classify 'Results' column with Model #1:

```
print(f"Evaluating LogisticRegression\n")
model = LogisticRegression(max_iter=1000)

# calc accuracy and precision
accuracy_scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
precision_scores = cross_val_score(model, X, y, cv=kfold, scoring='average_precision')

print(f"Accuracy for LogisticRegression: {np.mean(accuracy_scores) * 100:.2f}%")
print(f"Precision for LogisticRegression: {np.mean(precision_scores) * 100:.2f}%\n")

# train the model on the training set
model.fit(X_train, y_train)
# make predictions on the test set
y_pred = model.predict(X_test)

# calc the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index=model.classes_, columns=model.classes_)
print(f"Confusion Matrix for LogisticRegression:\n{cm_df}\n")

# check if the model meets the accuracy and precision goals
if np.mean(accuracy_scores) >= 0.90 and np.mean(precision_scores) >= 0.50:
    print(f"LogisticRegression has over 90% accuracy and at least 50% precision with 10-fold cv.\n")
```

```
Evaluating LogisticRegression

Accuracy for LogisticRegression: 88.86%
Precision for LogisticRegression: 52.00%

Confusion Matrix for LogisticRegression:
      0   1
0  1682  16
```

```
1   229  73
```

Classify 'Results' column with Model #2:

```
print(f"Evaluating RandomForestClassifier\n")
model = RandomForestClassifier()

# calc accuracy and precision
accuracy_scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
precision_scores = cross_val_score(model, X, y, cv=kfold, scoring='average_precision')

print(f"Accuracy for RandomForestClassifier: {np.mean(accuracy_scores) * 100:.2f}%")
print(f"Precision for RandomForestClassifier: {np.mean(precision_scores) * 100:.2f}%\n")

# train the model on the training set
model.fit(X_train, y_train)
# make predictions on the test set
y_pred = model.predict(X_test)

# calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index=model.classes_, columns=model.classes_)
print(f"Confusion Matrix for RandomForestClassifier:\n{cm_df}\n")

# check if the model meets the accuracy and precision goals
if np.mean(accuracy_scores) >= 0.90 and np.mean(precision_scores) >= 0.50:
    print(f"RandomForestClassifier has over 90% accuracy and at least 50% precision with 10-fold cv.\n")
```

```
Evaluating RandomForestClassifier

Accuracy for RandomForestClassifier: 91.18%
Precision for RandomForestClassifier: 61.82%

Confusion Matrix for RandomForestClassifier:
      0    1
0  1672   26
1   170  132

RandomForestClassifier has over 90% accuracy and at least 50% precision with 10-fold cv.
```

Classify 'Results' column with Model #3:

```
print(f"Evaluating NaiveBayes\n")
model = GaussianNB()

# calc accuracy and precision
accuracy_scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
precision_scores = cross_val_score(model, X, y, cv=kfold, scoring='average_precision')

print(f"Accuracy for NaiveBayes: {np.mean(accuracy_scores) * 100:.2f}%")
print(f"Precision for NaiveBayes: {np.mean(precision_scores) * 100:.2f}%\n")

# train the model on the training set
model.fit(X_train, y_train)
# make predictions on the test set
y_pred = model.predict(X_test)

# calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index=model.classes_, columns=model.classes_)
print(f"Confusion Matrix for NaiveBayes:\n{cm_df}\n")

# check if the model meets the accuracy and precision goals
if np.mean(accuracy_scores) >= 0.90 and np.mean(precision_scores) >= 0.50:
    print(f"NaiveBayes has over 90% accuracy and at least 50% precision with 10-fold cv.\n")
```

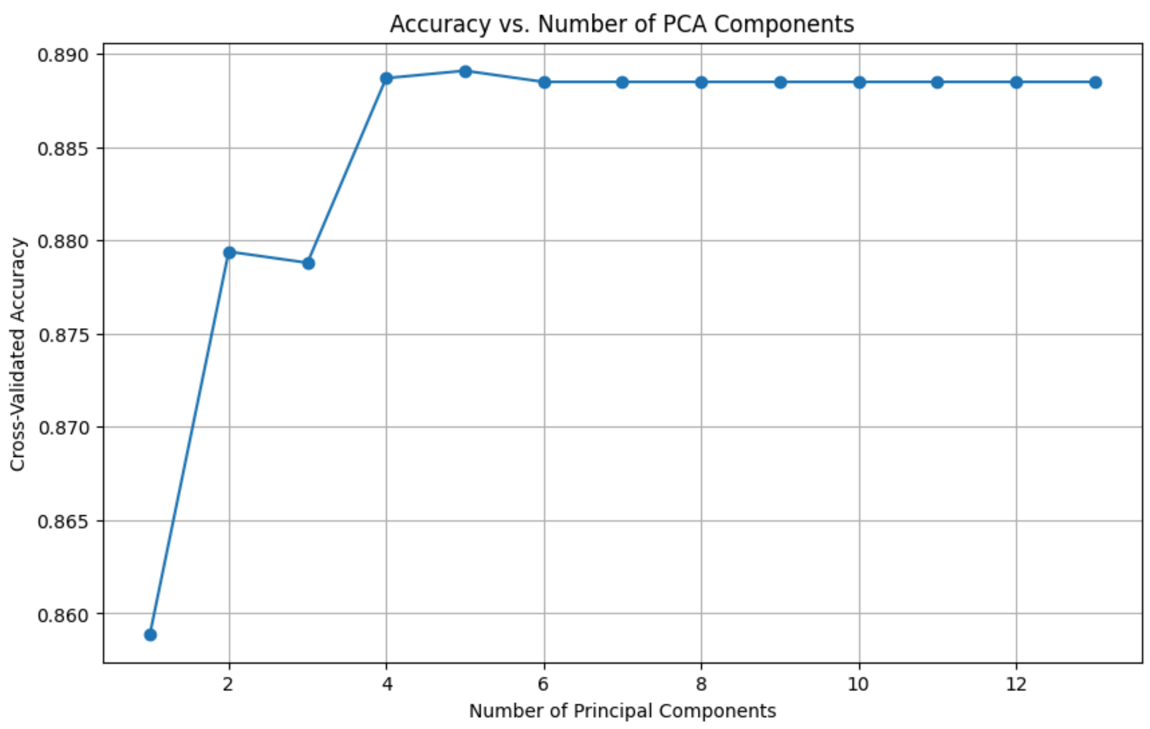
```
Evaluating NaiveBayes

10-Fold CV Accuracy for NaiveBayes: 85.15%
10-Fold CV Precision for NaiveBayes: 48.23%
```

```
Confusion Matrix for NaiveBayes:
      0    1
0  1548  150
1   181  121
```

Part 2: Run PCA and discover how many dimensions you can reduce the problem to before you start seeing significant decreases in accuracy.

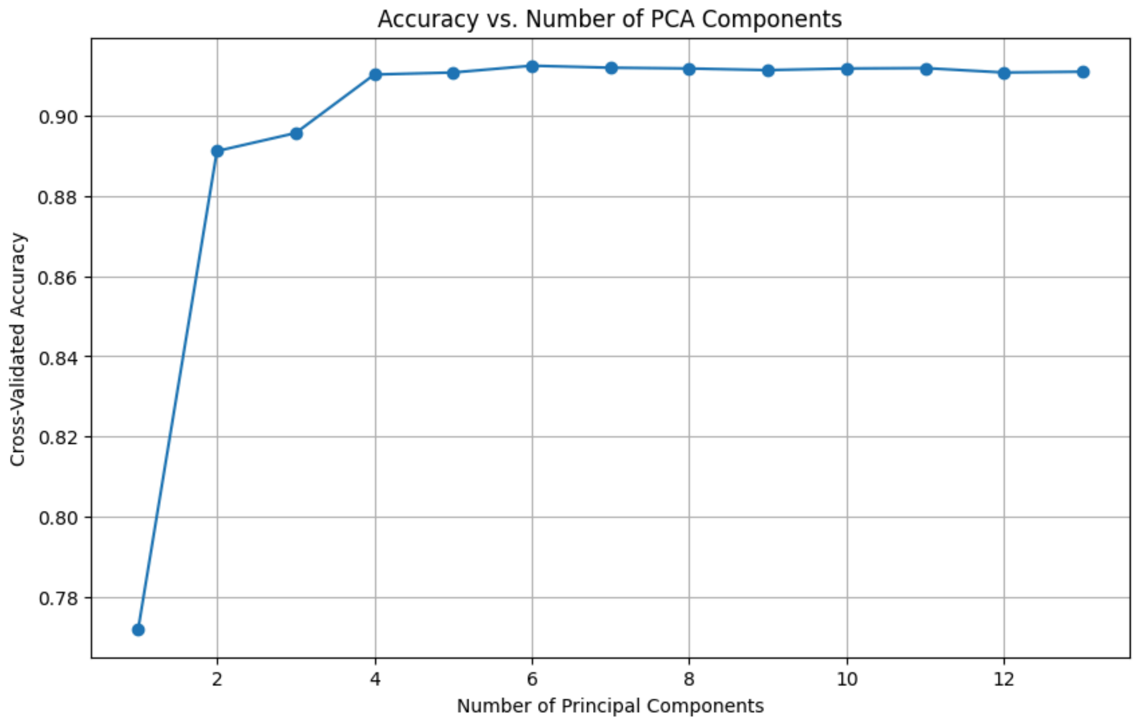
Run PCA:

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# load dataset
df = pd.read_csv('homework4.csv')


# separate features and target
X = df.drop(columns=['Results'])
y = df['Results']

# standardize the data for PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# model
model = LogisticRegression(max_iter=1000)

# PCA and test
accuracies = []
# range from 1 to the total number of features
components = range(1, X.shape[1] + 1)

for n_components in components:
    # PCA
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X_scaled)

    # 10-fold cv and calc accuracy
    scores = cross_val_score(model, X_pca, y, cv=10, scoring='accuracy')
    accuracies.append(scores.mean())

# accuracy vs. number of components
plt.figure(figsize=(10, 6))
plt.plot(components, accuracies, marker='o', linestyle='-')
plt.title('Accuracy vs. Number of PCA Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cross-Validated Accuracy')
plt.grid()
plt.show()

# find number of components where accuracy drops
best_accuracy = max(accuracies)
optimal_components = accuracies.index(best_accuracy) + 1

print(f"Optimal number of components: {optimal_components}")
print(f"Best accuracy achieved: {best_accuracy:.2f}")
```



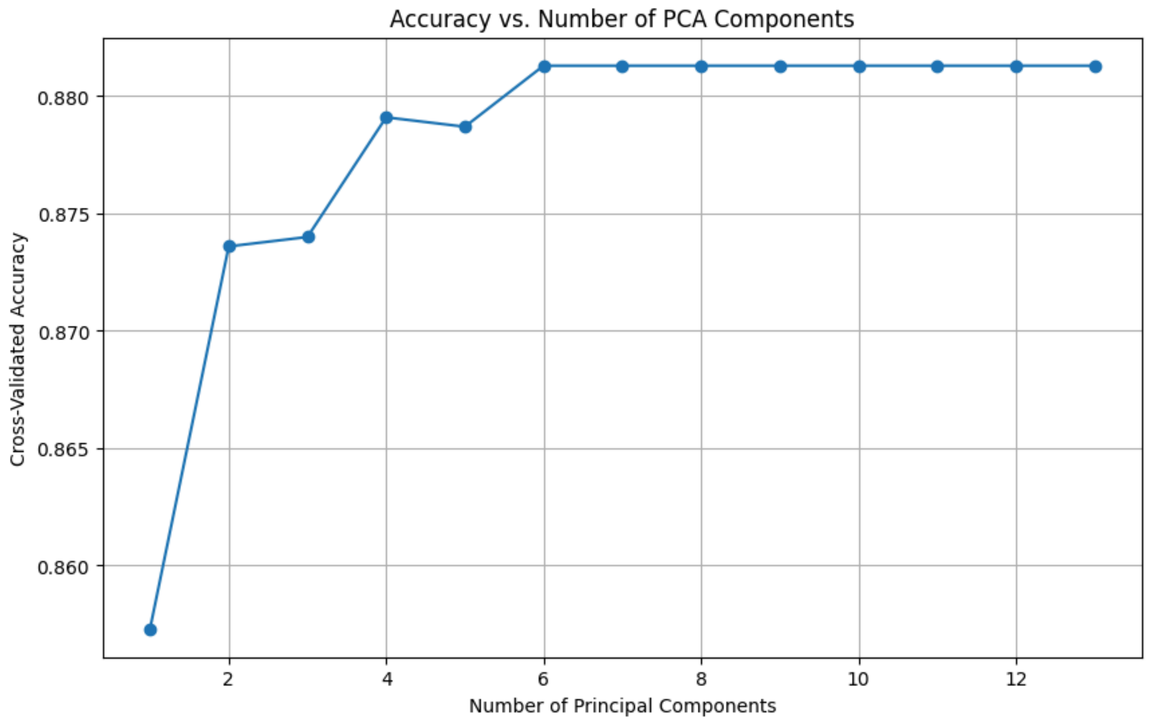
```
Optimal number of components: 5
Best accuracy achieved: 0.89
```

```
# separate features and target
X = df.drop(columns=['Results'])
y = df['Results']

# standardize the data for PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# model
model = RandomForestClassifier()

# PCA and test
accuracies = []
# range from 1 to the total number of features
components = range(1, X.shape[1] + 1)

for n_components in components:
    # PCA
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X_scaled)

    # 10-fold cv and calc accuracy
    scores = cross_val_score(model, X_pca, y, cv=10, scoring='accuracy')
    accuracies.append(scores.mean())

# accuracy vs. number of components
plt.figure(figsize=(10, 6))
plt.plot(components, accuracies, marker='o', linestyle='-')
plt.title('Accuracy vs. Number of PCA Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cross-Validated Accuracy')
plt.grid()
plt.show()

# find number of components where accuracy drops
best_accuracy = max(accuracies)
optimal_components = accuracies.index(best_accuracy) + 1

print(f"Optimal number of components: {optimal_components}")
print(f"Best accuracy achieved: {best_accuracy:.2f}")
```



```
Optimal number of components: 6
Best accuracy achieved: 0.91
```

```
# separate features and target
X = df.drop(columns=['Results'])
y = df['Results']

# standardize the data for PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# model
model = GaussianNB()

# PCA and test
accuracies = []
# range from 1 to the total number of features
components = range(1, X.shape[1] + 1)

for n_components in components:
    # PCA
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X_scaled)

    # 10-fold cv and calc accuracy
    scores = cross_val_score(model, X_pca, y, cv=10, scoring='accuracy')
    accuracies.append(scores.mean())

# accuracy vs. number of components
plt.figure(figsize=(10, 6))
plt.plot(components, accuracies, marker='o', linestyle='-')
plt.title('Accuracy vs. Number of PCA Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cross-Validated Accuracy')
plt.grid()
plt.show()

# find number of components where accuracy drops
best_accuracy = max(accuracies)
optimal_components = accuracies.index(best_accuracy) + 1

print(f"Optimal number of components: {optimal_components}")
print(f"Best accuracy achieved: {best_accuracy:.2f}")
```



```
Optimal number of components: 6
Best accuracy achieved: 0.88
```

How many dimensions can you reduce? (Explain your answer)

For logistic regression, looking at the graph, it shows that 5 is the optimal number of components before there is a significant decrease in accuracy, so 5 is the dimensions that this specific model can be reduced to before seeing decreases.

For random forest classifier, looking at the graph, it shows that 4 is the optimal number of components before there is a significant decrease in accuracy, even though in the graph it says that 6 is the most optimal. However, lower than 4 components is where there is the most significant decrease in accuracy, so 4 is the dimensions that this specific model can be reduced to before seeing decreases.

For naive bayes, looking at the graph, it shows that 6 is the optimal number of components before there is a significant decrease in accuracy, so 6 is the dimensions that this specific model can be reduced to before seeing decreases.