```
import requests
from io import StringIO
import pandas as pd
from scipy.io import arff
```

```
from scipy.io import arff
import pandas as pd
import requests
from io import StringIO

# URL of the dataset in ARFF format
url = 'http://promise.site.uottawa.ca/SERepository/datasets/kc1.arff'

# Download the dataset
response = requests.get(url)

# Load the dataset from the response content
data = arff.loadarff(StringIO(response.content.decode('utf-8')))
df = pd.DataFrame(data[0])

# Print the first few rows of the DataFrame
print(df.head())
```

```
    loc  v(g)  ev(g)  iv(g)      n       v     l      d      i         e  ...  \
0   1.1   1.4    1.4    1.4    1.3    1.30  1.30   1.30   1.30      1.30  ...
1   1.0   1.0    1.0    1.0    1.0    1.00  1.00   1.00   1.00      1.00  ...
2  83.0  11.0    1.0   11.0  171.0  927.89  0.04  23.04  40.27  21378.61  ...
3  46.0   8.0    6.0    8.0  141.0  769.78  0.07  14.86  51.81  11436.73  ...
4  25.0   3.0    1.0    3.0   58.0  254.75  0.11   9.35  27.25   2381.95  ...

   lOCode  lOComment  lOBlank  locCodeAndComment  uniq_Op  uniq_Opnd  \
0     2.0        2.0      2.0                2.0      1.2        1.2
1     1.0        1.0      1.0                1.0      1.0        1.0
2    65.0       10.0      6.0                0.0     18.0       25.0
3    37.0        2.0      5.0                0.0     16.0       28.0
4    21.0        0.0      2.0                0.0     11.0       10.0

   total_Op  total_Opnd  branchCount   defects
0       1.2         1.2          1.4  b'false'
1       1.0         1.0          1.0   b'true'
2     107.0        64.0         21.0   b'true'
3      89.0        52.0         15.0   b'true'
4      41.0        17.0          5.0   b'true'

[5 rows x 22 columns]
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode the 'defects' column
df['defects'] = label_encoder.fit_transform(df['defects'])

# Separate the 'defects' column from the rest of the data
defects_column = df['defects']

# Drop the 'defects' column from the DataFrame
df = df.drop(columns=['defects'])

# Initialize StandardScaler
scaler = StandardScaler()

# Fit scaler to your data and transform it
scaled_data = scaler.fit_transform(df)

# Concatenate the scaled features with the 'defects' column
scaled_df = pd.DataFrame(scaled_data, columns=df.columns)
scaled_df['defects'] = defects_column

# Print the first few rows of the scaled DataFrame
print(scaled_df.head())
```

```
        loc      v(g)     ev(g)     iv(g)         n         v         l  \
0 -0.647864 -0.368740 -0.124739 -0.339674 -0.580634 -0.498642  3.093249
1 -0.651226 -0.471309 -0.306546 -0.458191 -0.584224 -0.499223  2.146739
2  2.105319  2.092900 -0.306546  2.504721  1.449754  1.296396 -0.882093
3  0.861512  1.323638  1.966040  1.615847  1.090817  0.990097 -0.787442
4  0.155568  0.041533 -0.306546  0.134392  0.097757 -0.007646 -0.661241

          d         i         e  ...    lOCode  lOComment   lOBlank  \
```

```
0 -0.695929 -0.927649 -0.300506  ... -0.517950   0.341722  0.062345
1 -0.734088 -0.941606 -0.300524  ... -0.559302   0.017524 -0.196995
2  2.069348  0.885308  0.925197  ...  2.087232   2.935302  1.099707
3  1.028871  1.422170  0.355163  ...  0.929373   0.341722  0.840366
4  0.328012  0.279593 -0.164008  ...  0.267740  -0.306673  0.062345

   locCodeAndComment   uniq_Op  uniq_Opnd  total_Op  total_Opnd  branchCount  \
0           2.652867 -1.122654  -0.683788 -0.576537   -0.548440    -0.419224
1           1.232121 -1.157565  -0.700191 -0.580400   -0.554677    -0.470570
2          -0.188624  1.809801   1.268178  1.467364    1.409972     2.096706
3          -0.188624  1.460699   1.514224  1.119630    1.035753     1.326523
4          -0.188624  0.587944   0.037947  0.192341   -0.055719     0.042885

   defects
0        0
1        1
2        1
3        1
4        1

[5 rows x 22 columns]
```

```
print(df.shape)  # Check the dimensions of the DataFrame
print(df.head())  # Display the first few rows of the DataFrame
```

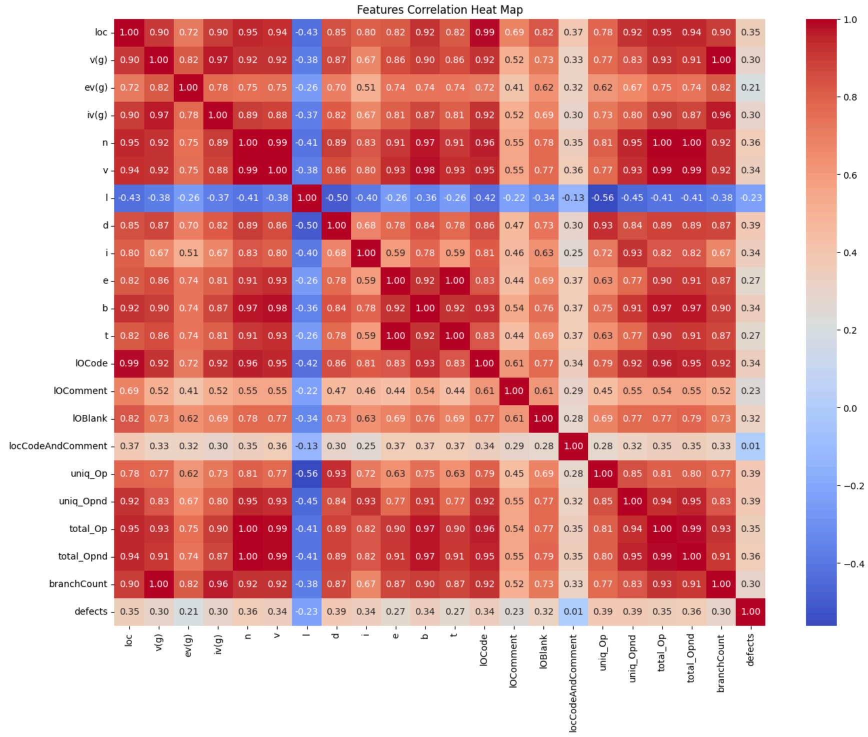
```
(2109, 21)
    loc  v(g)  ev(g)  iv(g)      n       v     l      d      i         e  ...  \
0   1.1   1.4    1.4    1.4    1.3    1.30  1.30   1.30   1.30      1.30  ...
1   1.0   1.0    1.0    1.0    1.0    1.00  1.00   1.00   1.00      1.00  ...
2  83.0  11.0    1.0   11.0  171.0  927.89  0.04  23.04  40.27  21378.61  ...
3  46.0   8.0    6.0    8.0  141.0  769.78  0.07  14.86  51.81  11436.73  ...
4  25.0   3.0    1.0    3.0   58.0  254.75  0.11   9.35  27.25   2381.95  ...

         t  lOCode  lOComment  lOBlank  locCodeAndComment  uniq_Op  uniq_Opnd  \
0     1.30     2.0        2.0      2.0                2.0      1.2        1.2
1     1.00     1.0        1.0      1.0                1.0      1.0        1.0
2  1187.70    65.0       10.0      6.0                0.0     18.0       25.0
3   635.37    37.0        2.0      5.0                0.0     16.0       28.0
4   132.33    21.0        0.0      2.0                0.0     11.0       10.0

   total_Op  total_Opnd  branchCount
0       1.2         1.2          1.4
1       1.0         1.0          1.0
2     107.0        64.0         21.0
3      89.0        52.0         15.0
4      41.0        17.0          5.0

[5 rows x 21 columns]
```

```
# Remove rows with null values
df.dropna(inplace=True)
```

```
# Check for NaN values in each column
nan_counts = df.isna().sum()

# Display columns with NaN values, if any
columns_with_nan = nan_counts[nan_counts > 0].index.tolist()
if columns_with_nan:
    print("Columns with NaN values:", columns_with_nan)
else:
    print("No NaN values remaining in the DataFrame")
```

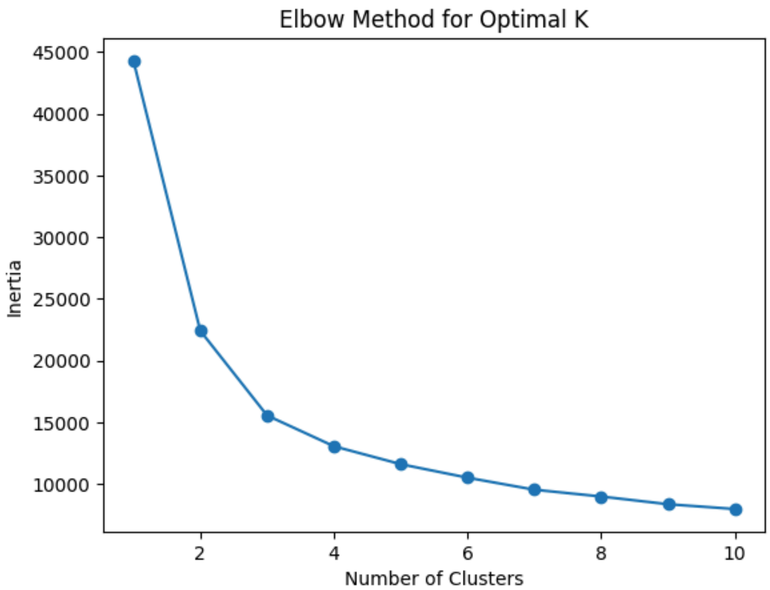
```
No NaN values remaining in the DataFrame
```

```
# Plotting the features correlation heat map
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 12))
correlation_matrix = scaled_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Features Correlation Heat Map')
plt.show()
```


Features Correlation Heat Map
loc
v(g)
ev(g)
iv(g)
n
v
l
d
i
e
b
t
lOCode
lOComment
lOBlank
locCodeAndComment
uniq_Op
uniq_Opnd
total_Op
total_Opnd
branchCount
defects
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Drop the 'defects' column for clustering
clustering_data = scaled_df.drop(columns=['defects'])

# Determine the optimal number of clusters using the elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(clustering_data)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()

# From the elbow curve, choose the optimal number of clusters
optimal_k = 2  # Example: Based on the plot, choose the optimal number of clusters

# Apply K-means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(clustering_data)

# Add cluster labels to the DataFrame
scaled_df['cluster'] = clusters

# Print the first few rows of the DataFrame with cluster labels
print(scaled_df.head())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
        loc      v(g)     ev(g)     iv(g)         n         v         l  \
0 -0.647864 -0.368740 -0.124739 -0.339674 -0.580634 -0.498642  3.093249
1 -0.651226 -0.471309 -0.306546 -0.458191 -0.584224 -0.499223  2.146739
2  2.105319  2.092900 -0.306546  2.504721  1.449754  1.296396 -0.882093
3  0.861512  1.323638  1.966040  1.615847  1.090817  0.990097 -0.787442
4  0.155568  0.041533 -0.306546  0.134392  0.097757 -0.007646 -0.661241

          d         i         e  ...  lOComment   lOBlank  locCodeAndComment  \
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.metrics import accuracy_score

# Split the data into training and testing sets
X = scaled_df.drop(columns=['defects', 'cluster'])  # Features
y = scaled_df['defects']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Initialize classifiers
svm_classifier = SVC(kernel='linear', random_state=42)
nb_classifier = GaussianNB()
rf_classifier = RandomForestClassifier(n_estimators=1000, random_state=42)

# Initialize stacking classifier
estimators = [('svm', svm_classifier), ('nb', nb_classifier), ('rf', rf_classifier)]
stacking_classifier = StackingClassifier(estimators=estimators, final_estimator=nb_classifier)

# Train the classifiers
svm_classifier.fit(X_train, y_train)
nb_classifier.fit(X_train, y_train)
rf_classifier.fit(X_train, y_train)
stacking_classifier.fit(X_train, y_train)

# Predictions
svm_pred = svm_classifier.predict(X_test)
nb_pred = nb_classifier.predict(X_test)
rf_pred = rf_classifier.predict(X_test)
stacking_pred = stacking_classifier.predict(X_test)

# Evaluate accuracy
svm_accuracy = accuracy_score(y_test, svm_pred)
nb_accuracy = accuracy_score(y_test, nb_pred)
rf_accuracy = accuracy_score(y_test, rf_pred)
stacking_accuracy = accuracy_score(y_test, stacking_pred)

# Print accuracies
print("SVM Accuracy:", svm_accuracy)
print("Naive Bayes Accuracy:", nb_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
print("Stacking Classifier Accuracy:", stacking_accuracy)
```

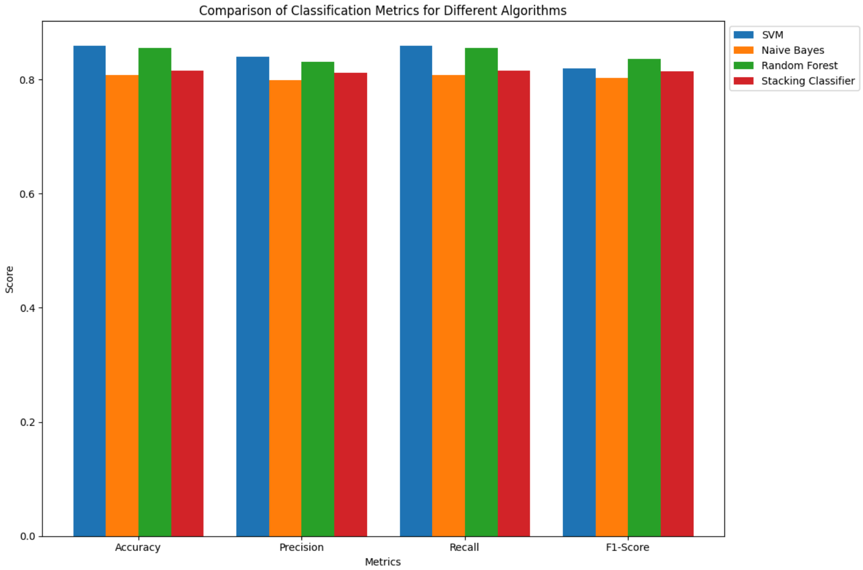
```
SVM Accuracy: 0.8591954022988506
Naive Bayes Accuracy: 0.8074712643678161
Random Forest Accuracy: 0.8548850574712644
Stacking Classifier Accuracy: 0.8160919540229885
```

```
from sklearn.metrics import classification_report

# Calculate metrics for SVM
svm_report = classification_report(y_test, svm_pred)
print("SVM Metrics:")
print(svm_report)

# Calculate metrics for Naive Bayes
nb_report = classification_report(y_test, nb_pred)
print("\nNaive Bayes Metrics:")
print(nb_report)

# Calculate metrics for Random Forest
rf_report = classification_report(y_test, rf_pred)
print("\nRandom Forest Metrics:")
print(rf_report)

# Calculate metrics for Stacking Classifier
stacking_report = classification_report(y_test, stacking_pred)
print("\nStacking Classifier Metrics:")
print(stacking_report)
```

```
SVM Metrics:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       588
           1       0.71      0.16      0.26       108

    accuracy                           0.86       696
   macro avg       0.79      0.57      0.59       696
weighted avg       0.84      0.86      0.82       696
```

```
Naive Bayes Metrics:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       588
           1       0.36      0.32      0.34       108

    accuracy                           0.81       696
   macro avg       0.62      0.61      0.62       696
weighted avg       0.80      0.81      0.80       696


Random Forest Metrics:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       588
           1       0.56      0.30      0.39       108

    accuracy                           0.85       696
   macro avg       0.72      0.63      0.65       696
weighted avg       0.83      0.85      0.84       696


Stacking Classifier Metrics:
              precision    recall  f1-score   support

           0       0.89      0.90      0.89       588
           1       0.40      0.38      0.39       108

    accuracy                           0.82       696
   macro avg       0.64      0.64      0.64       696
weighted avg       0.81      0.82      0.81       696
```

```
import matplotlib.pyplot as plt

# Extracting metrics for SVM
svm_metrics = classification_report(y_test, svm_pred, output_dict=True)
svm_accuracy = svm_metrics['accuracy']
svm_precision = svm_metrics['weighted avg']['precision']
svm_recall = svm_metrics['weighted avg']['recall']
svm_f1_score = svm_metrics['weighted avg']['f1-score']

# Extracting metrics for Naive Bayes
nb_metrics = classification_report(y_test, nb_pred, output_dict=True)
nb_accuracy = nb_metrics['accuracy']
nb_precision = nb_metrics['weighted avg']['precision']
nb_recall = nb_metrics['weighted avg']['recall']
nb_f1_score = nb_metrics['weighted avg']['f1-score']

# Extracting metrics for Random Forest
rf_metrics = classification_report(y_test, rf_pred, output_dict=True)
rf_accuracy = rf_metrics['accuracy']
rf_precision = rf_metrics['weighted avg']['precision']
rf_recall = rf_metrics['weighted avg']['recall']
rf_f1_score = rf_metrics['weighted avg']['f1-score']

# Extracting metrics for Stacking Classifier
stacking_metrics = classification_report(y_test, stacking_pred, output_dict=True)
stacking_accuracy = stacking_metrics['accuracy']
stacking_precision = stacking_metrics['weighted avg']['precision']
stacking_recall = stacking_metrics['weighted avg']['recall']
stacking_f1_score = stacking_metrics['weighted avg']['f1-score']

# Plotting
labels = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
svm_values = [svm_accuracy, svm_precision, svm_recall, svm_f1_score]
nb_values = [nb_accuracy, nb_precision, nb_recall, nb_f1_score]
rf_values = [rf_accuracy, rf_precision, rf_recall, rf_f1_score]
stacking_values = [stacking_accuracy, stacking_precision, stacking_recall, stacking_f1_score]

x = range(len(labels))

plt.figure(figsize=(12, 8))  # Increase the figure size

plt.bar(x, svm_values, width=0.2, label='SVM', align='center')
plt.bar([i + 0.2 for i in x], nb_values, width=0.2, label='Naive Bayes', align='center')
plt.bar([i + 0.4 for i in x], rf_values, width=0.2, label='Random Forest', align='center')
plt.bar([i + 0.6 for i in x], stacking_values, width=0.2, label='Stacking Classifier', align='center')

plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Comparison of Classification Metrics for Different Algorithms')
plt.xticks([i + 0.3 for i in x], labels)
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))  # Move legend outside the plot

plt.tight_layout()  # Adjust subplots to fit into figure area.
plt.show()
```




```
# prompt: i want to calculate MAE, RMSE, and MAPE, of applied algorithms

from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error

# Calculate mean absolute error (MAE)
svm_mae = mean_absolute_error(y_test, svm_pred)
nb_mae = mean_absolute_error(y_test, nb_pred)
rf_mae = mean_absolute_error(y_test, rf_pred)
stacking_mae = mean_absolute_error(y_test, stacking_pred)

# Calculate root mean squared error (RMSE)
svm_rmse = mean_squared_error(y_test, svm_pred, squared=False)
nb_rmse = mean_squared_error(y_test, nb_pred, squared=False)
rf_rmse = mean_squared_error(y_test, rf_pred, squared=False)
stacking_rmse = mean_squared_error(y_test, stacking_pred, squared=False)


# Print the calculated metrics
print("SVM MAE:", svm_mae)
print("Naive Bayes MAE:", nb_mae)
print("Random Forest MAE:", rf_mae)
print("Stacking Classifier MAE:", stacking_mae)

print("\nSVM RMSE:", svm_rmse)
print("Naive Bayes RMSE:", nb_rmse)
print("Random Forest RMSE:", rf_rmse)
print("Stacking Classifier RMSE:", stacking_rmse)
```

```
SVM MAE: 0.14080459770114942
Naive Bayes MAE: 0.1925287356321839
Random Forest MAE: 0.14511494252873564
Stacking Classifier MAE: 0.1839080459770115

SVM RMSE: 0.3752393871932282
Naive Bayes RMSE: 0.4387809654396871
Random Forest RMSE: 0.38093955232915316
Stacking Classifier RMSE: 0.4288450139351179
```