

```

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/pc1.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  \
0  1.1  1.4  1.4  1.4  1.3  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
2  91.0  9.0  3.0  2.0  318.0  2089.21  0.04  27.68  75.47
3  109.0  21.0  5.0  18.0  381.0  2547.56  0.04  28.37  89.79
4  505.0  106.0  41.0  82.0  2339.0  20696.93  0.01  75.93  272.58

E  ...  l0Code  l0Comment  locCodeAndComment  l0Blank  uniq_Op  \
0  1.30  ...  2.0  2.0  2.0  2.0  1.2
1  1.00  ...  1.0  1.0  1.0  1.0  1.0
2  57833.24  ...  80.0  44.0  11.0  31.0  29.0
3  72282.68  ...  97.0  41.0  12.0  24.0  28.0
4  1571506.88  ...  457.0  71.0  48.0  49.0  64.0

uniq_Opnd  total_Op  total_Opnd  branchCount  defects
0  1.2  1.2  1.2  1.4  b'false'
1  1.0  1.0  1.0  1.0  b'true'
2  66.0  192.0  126.0  17.0  b'true'
3  75.0  229.0  152.0  38.0  b'true'
4  397.0  1397.0  942.0  178.0  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  -0.631622  -0.459046  -0.245601  -0.300256  -0.588566  -0.462872  7.969369
1  -0.634457  -0.503714  -0.317476  -0.362765  -0.590087  -0.463071  5.926964
2  1.917424  0.389648  0.041900  -0.206493  1.017028  0.920890  -0.608731
3  2.427800  1.729691  0.401277  2.293854  1.336423  1.224661  -0.608731
4  13.656075  11.221663  6.870055  12.295243  11.263018  13.253154  -0.812972

D  I  E  ...  l0Code  l0Comment  locCodeAndComment  \

```

```

0 -0.863205 -0.893495 -0.168975 ... -0.608907 -0.256439 0.315512
1 -0.881576 -0.901977 -0.168977 ... -0.638705 -0.351553 0.016450
2 0.752212 1.203377 0.170082 ... 1.715323 3.738348 3.007069
3 0.794465 1.608220 0.254797 ... 2.221885 3.453006 3.306131
4 3.706872 6.775907 9.044464 ... 12.949100 6.306425 14.072361

```

```

      l0Blank  uniq_Op  uniq_Opnd  total_Op  total_Opnd  branchCount  defects
0 -0.385967 -1.480420 -0.678168 -0.584789 -0.576122 -0.494553 0
1 -0.467296 -1.504874 -0.685056 -0.586581 -0.578440 -0.518747 1
2 1.972589 1.918722 1.553365 1.124079 0.870503 0.449002 1
3 1.403282 1.796451 1.863300 1.455463 1.171883 1.719172 1
4 3.436519 6.198217 12.952091 11.916457 10.329205 10.186974 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

```

```

(1109, 21)
      loc  v(g)  ev(g)  iv(G)      N      V      L      D      I  \
0      1.1    1.4    1.4    1.4      1.3    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
2      91.0    9.0    3.0    2.0    318.0  2089.21  0.04  27.68  75.47
3     109.0   21.0    5.0    18.0   381.0  2547.56  0.04  28.37  89.79
4     505.0  106.0   41.0    82.0  2339.0  20696.93  0.01  75.93  272.58

```

```

      E  ...      T  l0Code  l0Comment  locCodeAndComment  l0Blank  \
0      1.30  ...      1.30    2.0      2.0      2.0    2.0
1      1.00  ...      1.00    1.0      1.0      1.0    1.0
2     57833.24  ...    3212.96    80.0     44.0     11.0   31.0
3     72282.68  ...    4015.70    97.0     41.0     12.0   24.0
4    1571506.88  ...    87305.94   457.0     71.0     48.0   49.0

```

```

      uniq_Op  uniq_Opnd  total_Op  total_Opnd  branchCount
0      1.2      1.2      1.2      1.2      1.4
1      1.0      1.0      1.0      1.0      1.0
2     29.0     66.0    192.0    126.0     17.0
3     28.0     75.0    229.0    152.0     38.0
4     64.0    397.0   1397.0    942.0    178.0

```

```
[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")

```

```

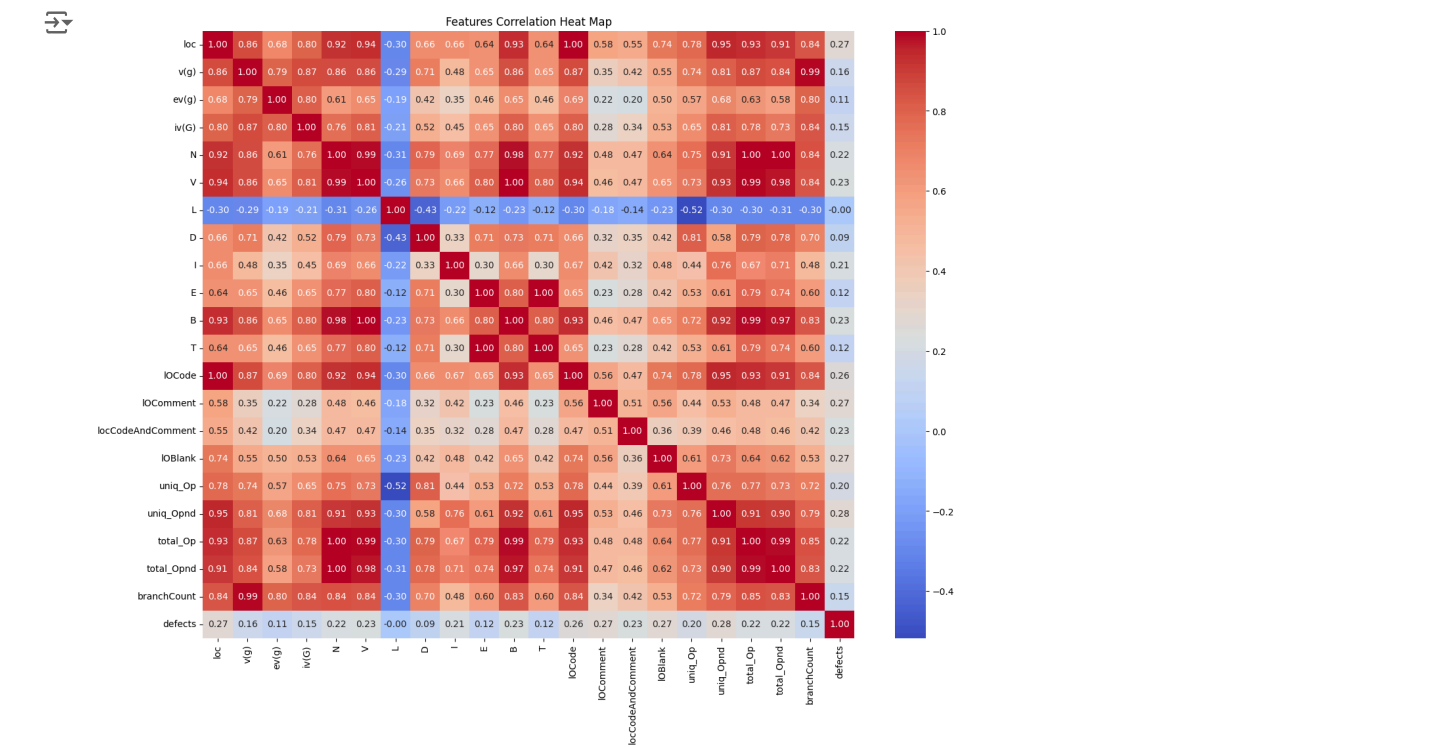
No NaN values remaining in the DataFrame

```

```

# 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()

```



```
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)

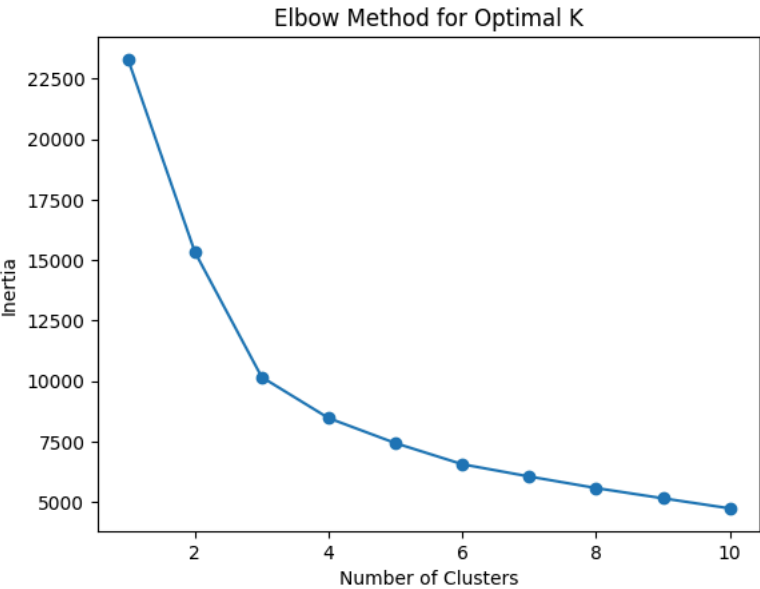
# Add cluster labels to the DataFrame
scaled_df['cluster'] = clusters

# 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(

```

	loc	v(g)	ev(g)	iv(G)	N	V	L	\
0	-0.631622	-0.459046	-0.245601	-0.300256	-0.588566	-0.462872	7.969369	
1	-0.634457	-0.503714	-0.317476	-0.362765	-0.590087	-0.463071	5.926964	
2	1.917424	0.389648	0.041900	-0.206493	1.017028	0.920890	-0.608731	
3	2.427800	1.729691	0.401277	2.293854	1.336423	1.224661	-0.608731	
4	13.656075	11.221663	6.870055	12.295243	11.263018	13.253154	-0.812972	

```

D      I      E      ...      l0Comment      locCodeAndComment      l0Blank      \

```

```

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: 0.9180327868852459
Naive Bayes Accuracy: 0.8743169398907104
Random Forest Accuracy: 0.9262295081967213
Stacking Classifier Accuracy: 0.8797814207650273

```

```

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	0.92	1.00	0.96	336
1	0.00	0.00	0.00	30
accuracy			0.92	366
macro avg	0.46	0.50	0.48	366
weighted avg	0.84	0.92	0.88	366

Naive Bayes Metrics:				
	precision	recall	f1-score	support
0	0.93	0.93	0.93	336
1	0.23	0.23	0.23	30
accuracy			0.87	366
macro avg	0.58	0.58	0.58	366
weighted avg	0.87	0.87	0.87	366

Random Forest Metrics:				
	precision	recall	f1-score	support
0	0.93	0.99	0.96	336
1	0.71	0.17	0.27	30
accuracy			0.93	366
macro avg	0.82	0.58	0.62	366
weighted avg	0.91	0.93	0.90	366

Stacking Classifier Metrics:				
	precision	recall	f1-score	support
0	0.94	0.93	0.93	336
1	0.31	0.37	0.33	30
accuracy			0.88	366
macro avg	0.62	0.65	0.63	366
weighted avg	0.89	0.88	0.88	366

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
_warn_prf(average, modifier, msg_start, len(result))
```

```

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()

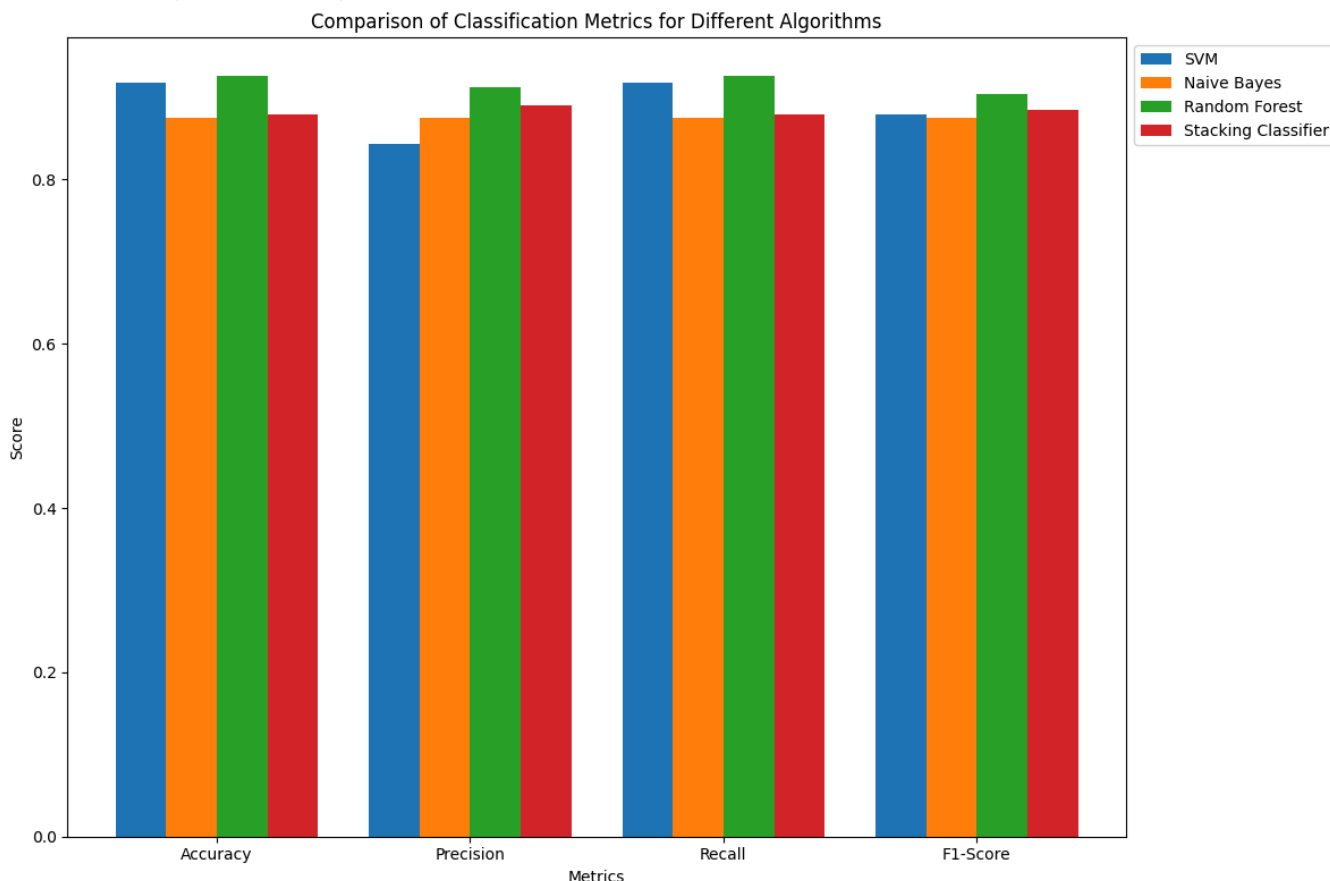
```



```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
_warn_prf(average, modifier, msg_start, len(result))

```



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

```
from sklearn.metrics import mean_absolute_error, mean_squared_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: 0.08196721311475409
Naive Bayes MAE: 0.12568306010928962
Random Forest MAE: 0.07377049180327869
Stacking Classifier MAE: 0.12021857923497267

SVM RMSE: 0.2862991671569341
Naive Bayes RMSE: 0.35451806739472336
Random Forest RMSE: 0.2716072381275556
Stacking Classifier RMSE: 0.3467255099282034

import matplotlib.pyplot as plt
import numpy as np

# Define the algorithms and their corresponding metrics
algorithms = ['SVM', 'Naive Bayes', 'Random Forest', 'Stacking Classifier']
mae_values = [svm_mae, nb_mae, rf_mae, stacking_mae]
rmse_values = [svm_rmse, nb_rmse, rf_rmse, stacking_rmse]

# Set the width of the bars
bar_width = 0.35

# Set positions of the bars on the x-axis
r1 = np.arange(len(algorithms))
r2 = [x + bar_width for x in r1]

# Plotting
plt.figure(figsize=(12, 6))

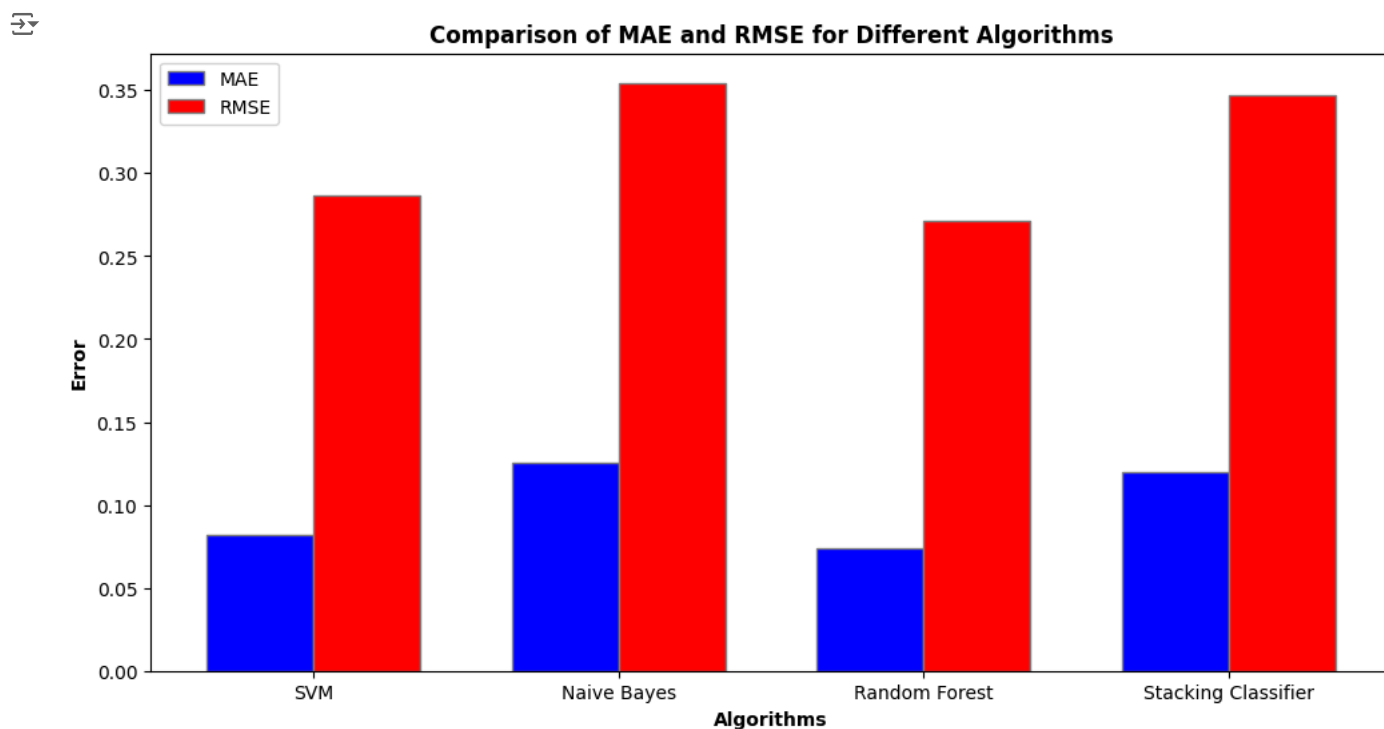
# Create bars
plt.bar(r1, mae_values, color='b', width=bar_width, edgecolor='grey', label='MAE')
plt.bar(r2, rmse_values, color='r', width=bar_width, edgecolor='grey', label='RMSE')

# Add labels
plt.xlabel('Algorithms', fontweight='bold')
plt.ylabel('Error', fontweight='bold')
plt.title('Comparison of MAE and RMSE for Different Algorithms', fontweight='bold')

# Add xticks on the middle of the group bars
plt.xticks([r + bar_width/2 for r in range(len(algorithms))], algorithms)

# Create legend
plt.legend()

# Show plot
plt.show()
```



```
pip install lazypredict
```

↗ Collecting lazypredict
Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)

```
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from lazypredict) (8.1.7)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from lazypredict) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from lazypredict) (2.0.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from lazypredict) (4.66.4)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from lazypredict) (1.4.2)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.10/dist-packages (from lazypredict) (4.1.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (from lazypredict) (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lightgbm->lazypredict) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lightgbm->lazypredict) (1.11.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->lazypredict) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->lazypredict) (1.16.0)
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

```
import lazypredict
from lazypredict.Supervised import LazyClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# X = data.data
# y = data.target

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5, random_state =123)

clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)

print(models)
```

```
97%|██████████| 28/29 [00:03<00:00, 5.30it/s][LightGBM] [Info] Number of positive: 47, number of negative: 696
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000387 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2125
[LightGBM] [Info] Number of data points in the train set: 743, number of used features: 21
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.063257 -> initscore=-2.695202
[LightGBM] [Info] Start training from score -2.695202
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
100%|██████████| 29/29 [00:03<00:00, 8.11it/s][LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

Accuracy Balanced Accuracy ROC AUC F1 Score \
Model
DecisionTreeClassifier      0.91      0.63      0.63      0.90
ExtraTreeClassifier         0.91      0.63      0.63      0.90
XGBClassifier               0.93      0.61      0.61      0.91
AdaBoostClassifier          0.93      0.60      0.60      0.91
RandomForestClassifier      0.93      0.60      0.60      0.91
KNeighborsClassifier        0.93      0.60      0.60      0.91
BaggingClassifier           0.93      0.60      0.60      0.91
ExtraTreesClassifier        0.93      0.60      0.60      0.91
LinearDiscriminantAnalysis  0.92      0.59      0.59      0.90
LGBMClassifier              0.92      0.59      0.59      0.90
GaussianNB                  0.87      0.58      0.58      0.87
BernoulliNB                 0.70      0.58      0.58      0.77
LabelPropagation            0.91      0.57      0.57      0.89
LabelSpreading              0.91      0.57      0.57      0.89
SGDClassifier               0.91      0.57      0.57      0.89
LinearSVC                   0.92      0.55      0.55      0.90
LogisticRegression          0.92      0.55      0.55      0.89
NearestCentroid             0.80      0.54      0.54      0.83
QuadraticDiscriminantAnalysis 0.90      0.52      0.52      0.88
SVC                         0.92      0.52      0.52      0.89
DummyClassifier             0.92      0.50      0.50      0.88
Perceptron                  0.92      0.50      0.50      0.88
CalibratedClassifierCV       0.92      0.50      0.50      0.88
RidgeClassifier             0.92      0.50      0.50      0.88
RidgeClassifierCV           0.92      0.50      0.50      0.88
PassiveAggressiveClassifier  0.90      0.49      0.49      0.87

Time Taken
Model
DecisionTreeClassifier      0.02
ExtraTreeClassifier         0.01
```

```
XGBClassifier          1.21
AdaBoostClassifier     0.16
RandomForestClassifier 0.35
```

```
from sklearn.feature_selection import SelectKBest, f_classif
```

```
# Separate features (X) and target variable (y)
```

```
X = scaled_df.drop(columns=['defects'])
```

```
y = scaled_df['defects']
```

```
# Perform feature selection using SelectKBest with ANOVA F-value
```

```
k = 10 # Number of features to select
```

```
selector = SelectKBest(score_func=f_classif, k=k)
```

```
X_selected = selector.fit_transform(X, y)
```

```
# Get indices of selected features
```

```
selected_indices = selector.get_support(indices=True)
```

```
# Get names of selected features
```

```
selected_features = X.columns[selected_indices]
```

```
# Print selected features
```

```
print("Selected Features:")
```

```
print(selected_features)
```

```
Selected Features:
Index(['loc', 'n', 'v', 'i', 'b', 'lOComment', 'uniq_Op', 'uniq_Opnd',
      'total_Op', 'total_Opnd'],
      dtype='object')
```

```
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
# Split the data into training and testing sets
```

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

```
# Initialize the MLPClassifier
```

```
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', random_state=42)
```

```
# Train the classifier
```

```
mlp_classifier.fit(X_train, y_train)
```

```
# Predict on the test set
```

```
y_pred = mlp_classifier.predict(X_test)
```

```
# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
```

```
Accuracy: 0.87
```

```
from sklearn.metrics import confusion_matrix
```

```
# Calculate confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
print("Confusion Matrix:")
```

```
print(conf_matrix)
```

```
Confusion Matrix:
[[87  1]
 [12  0]]
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Plot heatmap
```

```
plt.figure(figsize=(8, 6))
```

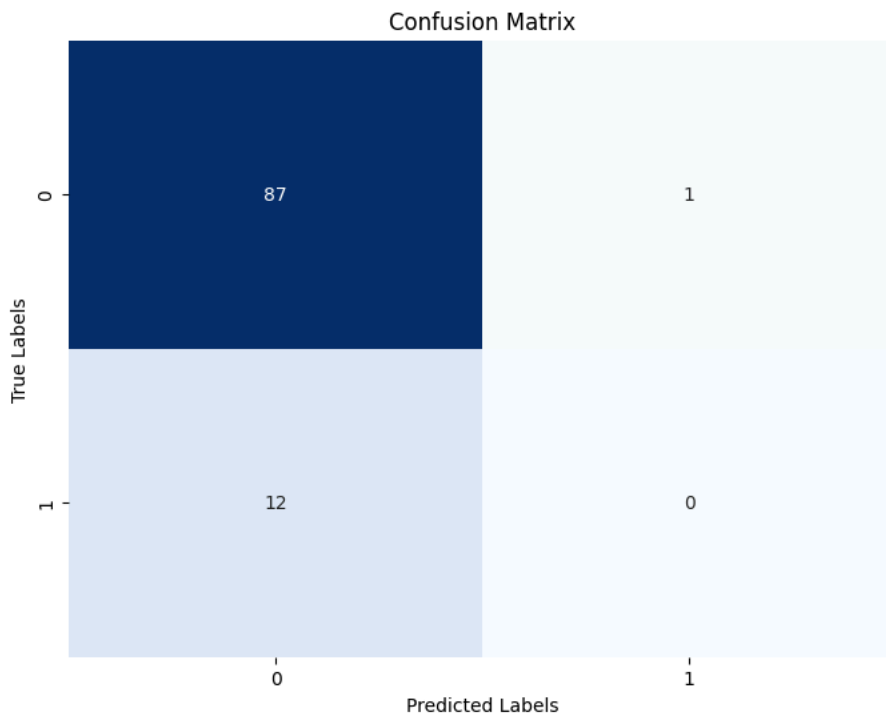
```
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
```

```
plt.title("Confusion Matrix")
```

```
plt.xlabel("Predicted Labels")
```

```
plt.ylabel("True Labels")
```

```
plt.show()
```



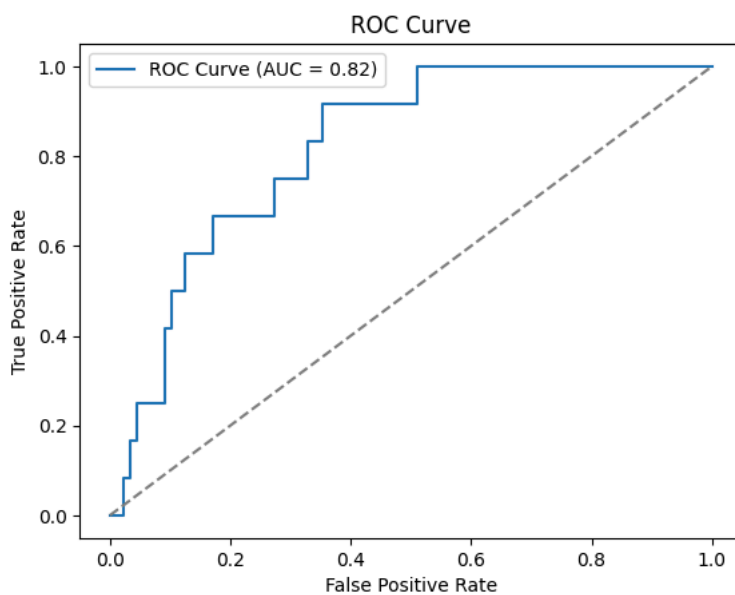
```
from sklearn.metrics import roc_auc_score, roc_curve

# Calculate AUC
auc = roc_auc_score(y_test, mlp_classifier.predict_proba(X_test)[:, 1])
print("AUC:", auc)

# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, mlp_classifier.predict_proba(X_test)[:, 1])
plt.plot(fpr, tpr, label="ROC Curve (AUC = {:.2f})".format(auc))
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



AUC: 0.8210227272727273



```
from sklearn.svm import SVC

# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear', random_state=42)

# Calculate accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("SVM Accuracy:", accuracy_svm)
```