```
import pandas as pd
df = pd.read_excel('CTG.xls','Data2')
df.head()
```

	b	e	AC	FM	UC	DL	DS	DP	DR	Unnamed: 9	...	E	AD	DE	LD	FS	SUSP	Unnamed: 42	CLASS	Unnamed: 44	NSP
0	240.0	357.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	...	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	NaN	9.0	NaN	2.0
1	5.0	632.0	4.0	0.0	4.0	2.0	0.0	0.0	0.0	NaN	...	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	NaN	6.0	NaN	1.0
2	177.0	779.0	2.0	0.0	5.0	2.0	0.0	0.0	0.0	NaN	...	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	NaN	6.0	NaN	1.0
3	411.0	1192.0	2.0	0.0	6.0	2.0	0.0	0.0	0.0	NaN	...	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	NaN	6.0	NaN	1.0
4	533.0	1147.0	4.0	0.0	5.0	0.0	0.0	0.0	0.0	NaN	...	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	NaN	2.0	NaN	1.0

5 rows × 46 columns

```
df = df.drop(df.columns[df.columns.str.contains('Unnamed', na=False)], axis=1)
df = df.drop(df.columns[0:9], axis=1)
```

```
df.head()
```

	LB	AC.1	FM.1	UC.1	DL.1	DS.1	DP.1	ASTV	MSTV	ALTV	...	C	D	E	AD	DE	LD	FS	SUSP	CLASS	NSP
0	120.0	0.000000	0.0	0.000000	0.000000	0.0	0.0	73.0	0.5	43.0	...	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	9.0	2.0
1	132.0	0.006380	0.0	0.006380	0.003190	0.0	0.0	17.0	2.1	0.0	...	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	6.0	1.0
2	133.0	0.003322	0.0	0.008306	0.003322	0.0	0.0	16.0	2.1	0.0	...	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	6.0	1.0
3	134.0	0.002561	0.0	0.007682	0.002561	0.0	0.0	16.0	2.4	0.0	...	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	6.0	1.0
4	132.0	0.006515	0.0	0.008143	0.000000	0.0	0.0	16.0	2.4	0.0	...	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	2.0	1.0

5 rows × 33 columns

```
df.isnull().sum()
```

```
LB          2
AC.1        2
FM.1        2
UC.1        2
DL.1        1
DS.1        1
DP.1        1
ASTV        2
MSTV        2
ALTV        2
MLTV        2
Width       2
Min         2
Max         2
Nmax        2
Nzeros      2
Mode        2
Mean        2
Median      2
Variance    2
Tendency    2
A           1
B           1
C           1
D           1
E           1
AD          1
DE          1
LD          1
FS          1
SUSP        1
CLASS       2
NSP         2
dtype: int64
```

```
df=df.dropna()
df.isnull().sum()
```

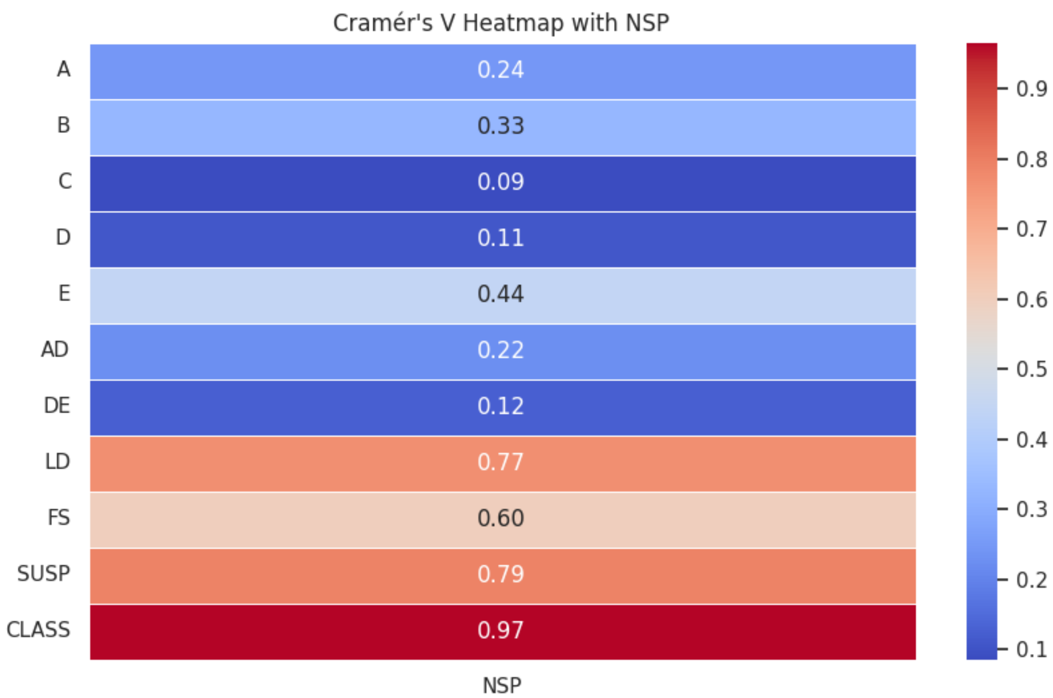
```
LB          0
AC.1        0
FM.1        0
UC.1        0
DL.1        0
DS.1        0
DP.1        0
ASTV        0
MSTV        0
ALTV        0
MLTV        0
```

```
Width       0
Min         0
Max         0
Nmax        0
Nzeros      0
Mode        0
Mean        0
Median      0
Variance    0
Tendency    0
A           0
B           0
C           0
D           0
E           0
AD          0
DE          0
LD          0
FS          0
SUSP        0
CLASS       0
NSP         0
dtype: int64
```

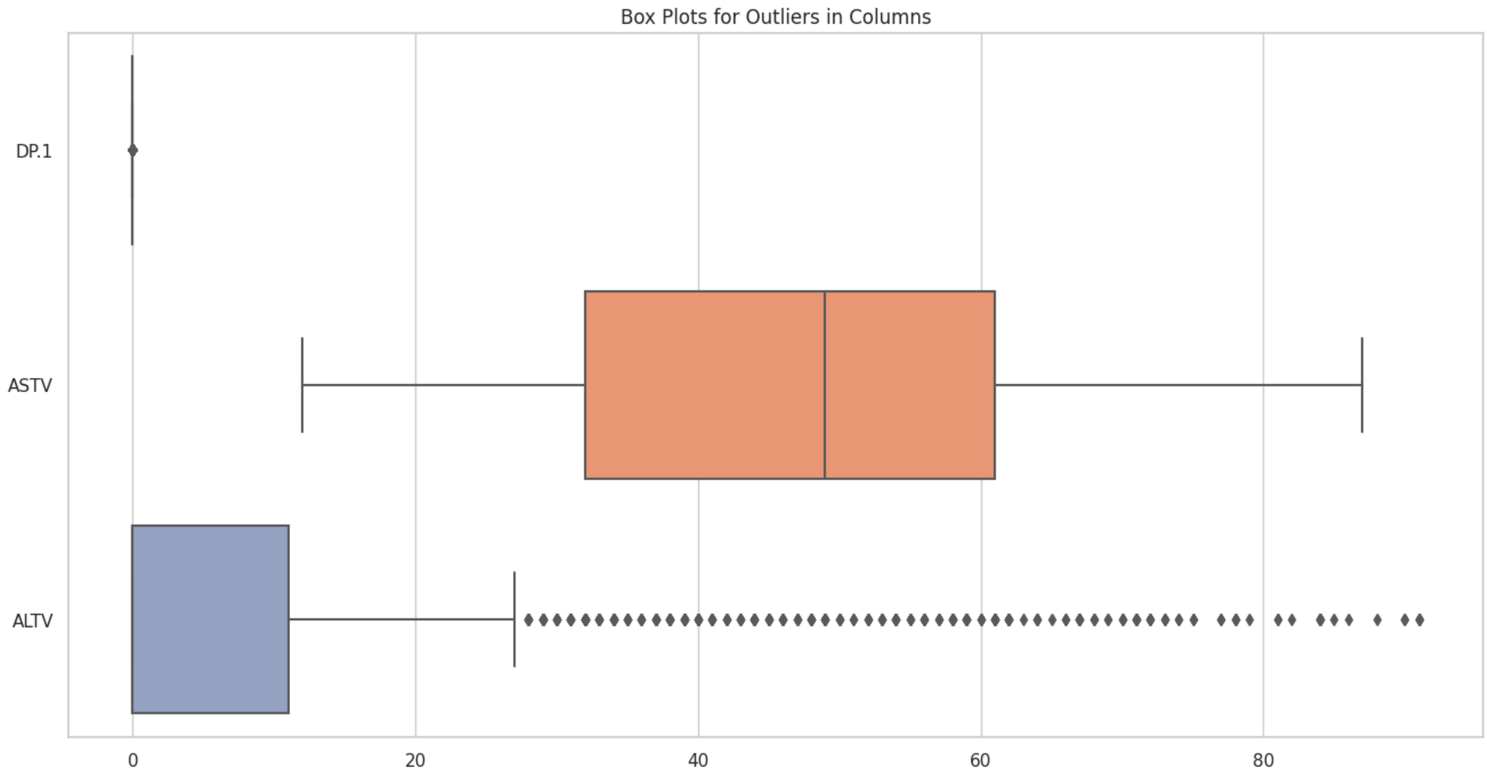
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

correlation_with_target = df.corr()[['NSP']]
print("Correlation with Target Column:")
print(correlation_with_target)

sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.barplot(x=correlation_with_target.index, y=correlation_with_target['NSP'])
plt.xticks(rotation=90)
plt.title(f'Correlation with NSP')
plt.show()
```

```
Correlation with Target Column:
               NSP
LB        0.148151
AC.1     -0.363849
FM.1      0.087933
UC.1     -0.203824
DL.1      0.062702
DS.1      0.135629
DP.1      0.488277
ASTV      0.471191
MSTV     -0.103382
ALTV      0.426146
MLTV     -0.226797
Width    -0.068789
Min       0.063175
Max      -0.045265
Nmax     -0.023666
Nzeros   -0.016682
```

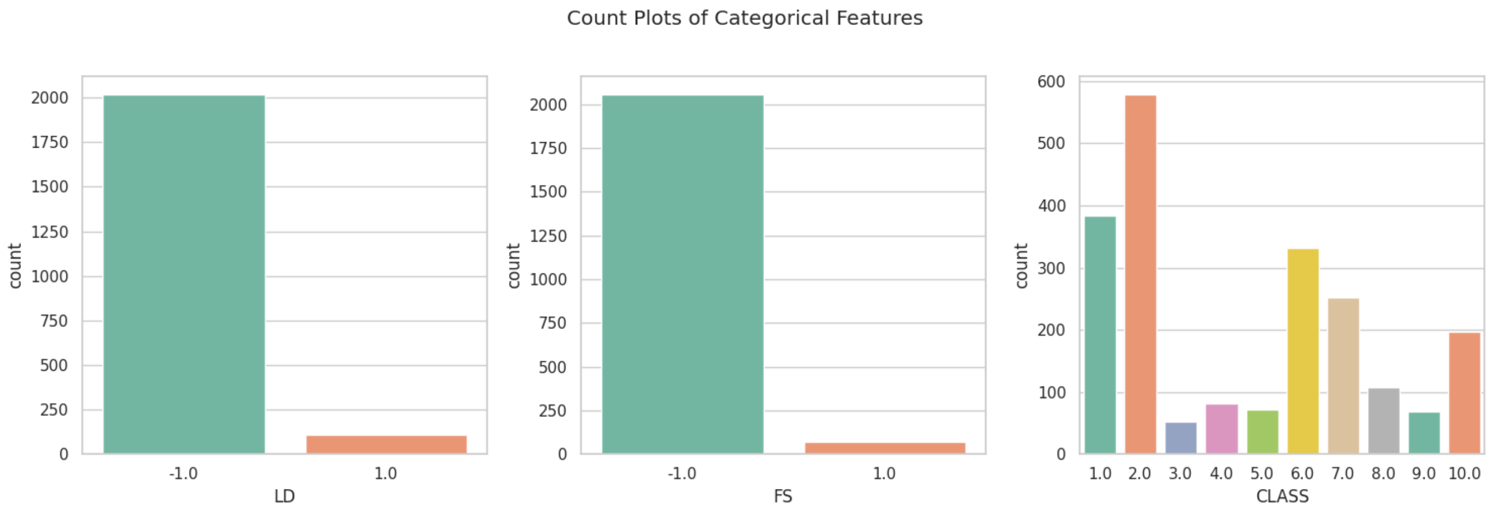
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency


categorical_features = ['A','B','C','D','E', 'AD','DE','LD','FS','SUSP','CLASS']
target_variable = 'NSP'

contingency_tables = pd.DataFrame(index=categorical_features, columns=[target_variable])

for feature in categorical_features:
    contingency_table = pd.crosstab(df[feature], df[target_variable])
    chi2, _, _, _ = chi2_contingency(contingency_table)
    n = np.sum(contingency_table.sum())
    cramers_v = np.sqrt(chi2 / (n * (min(contingency_table.shape) - 1)))
    contingency_tables.at[feature, target_variable] = cramers_v

contingency_tables = contingency_tables.apply(pd.to_numeric)

plt.figure(figsize=(10, 6))
sns.heatmap(contingency_tables, annot=True, cmap="coolwarm", fmt=".2f", linewidths=.5)
plt.title(f'Cramér\'s V Heatmap with {target_variable}')
plt.show()
```



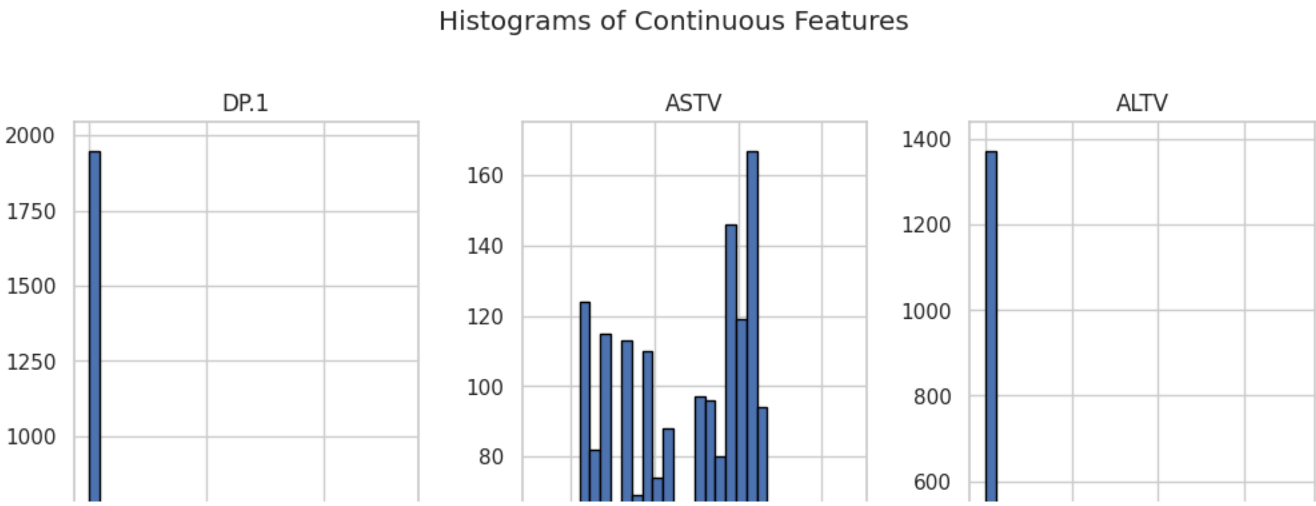

```
plt.figure(figsize=(16, 8))
cols = [ 'DP.1', 'ASTV','ALTV']
sns.boxplot(data=df[cols], orient="h", palette="Set2")
plt.title('Box Plots for Outliers in Columns')
plt.show()
```




```
sns.set(style="whitegrid")
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
sns.countplot(x='LD', data=df, ax=axes[0], palette='Set2')
sns.countplot(x='FS', data=df, ax=axes[1], palette='Set2')
sns.countplot(x='CLASS', data=df, ax=axes[2], palette='Set2')
plt.suptitle('Count Plots of Categorical Features', y=1.02)
plt.tight_layout()
plt.show()
```




```
sns.set(style="whitegrid")
df[['DP.1', 'ASTV', 'ALTV']].hist(bins=30, figsize=(12, 6), edgecolor='black', layout=(1, 3))
plt.suptitle('Histograms of Continuous Features', y=1.02)
plt.show()
```




```
print(df['DP.1'].skew())
print(df.ASTV.skew())
print(df.ALTV.skew())
```

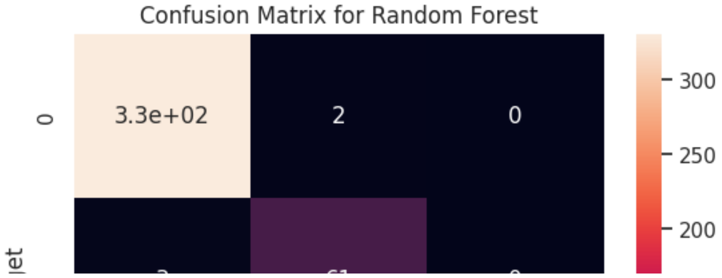
```
4.27880069004074
-0.011828576215617885
2.195075308589335
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
X = df[['LD', 'FS', 'CLASS',
        'DP.1', 'ASTV', 'ALTV']]
y = df['NSP']

X = pd.get_dummies(X, columns=['LD', 'FS', 'CLASS'], drop_first=True)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf_rf = RandomForestClassifier(n_estimators=100, random_state=42)
clf_rf.fit(X_train, y_train)
y_pred = clf_rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy_rf:.2f}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
Accuracy: 0.98

Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       333
           1       0.95      0.94      0.94        64
           2       1.00      0.97      0.98        29

    accuracy                           0.98       426
   macro avg       0.98      0.97      0.97       426
weighted avg       0.98      0.98      0.98       426
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cf= confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix for Random Forest')
```

```
Text(0.5, 1.0, 'Confusion Matrix for Random Forest')
```

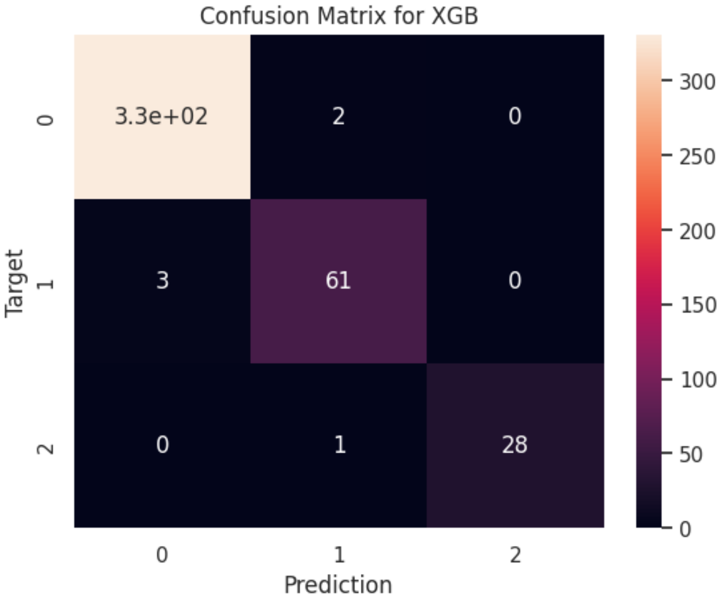



```
from xgboost import XGBClassifier
clf_xgb = XGBClassifier(n_estimators=100, random_state=42)
clf_xgb.fit(X_train, y_train)
y_pred = clf_xgb.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy_xgb:.2f}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
Accuracy: 0.99

Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       333
           1       0.95      0.95      0.95        64
           2       1.00      0.97      0.98        29

    accuracy                           0.99       426
   macro avg       0.98      0.97      0.98       426
weighted avg       0.99      0.99      0.99       426
```

```
cf= confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix for XGB')
```

```
Text(0.5, 1.0, 'Confusion Matrix for XGB')
```



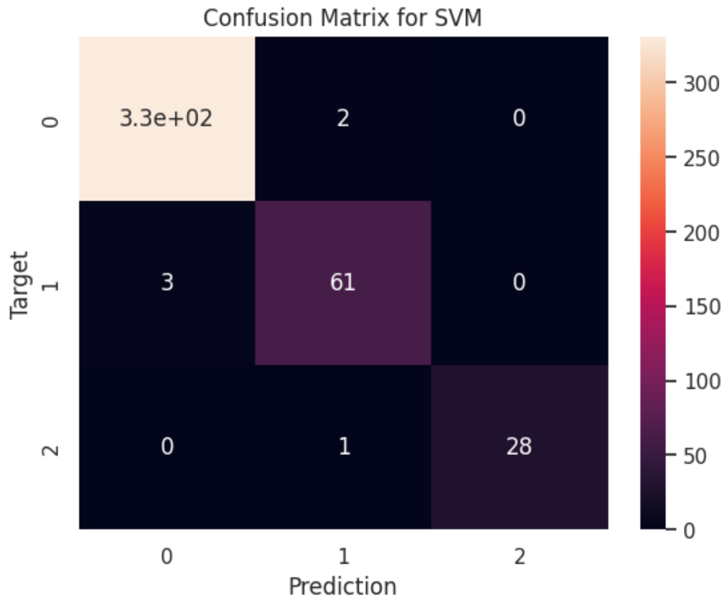

```
from sklearn.svm import SVC
clf_svm = SVC(kernel='linear', C=1.0, random_state=42)
clf_svm.fit(X_train, y_train)
y_pred_svm = clf_svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'SVM Accuracy: {accuracy_svm:.2f}')
print('\nSVM Classification Report:\n', classification_report(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.98

SVM Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       333
           1       0.95      0.92      0.94        64
           2       1.00      0.97      0.98        29

    accuracy                           0.98       426
```

```
    macro avg       0.98      0.96      0.97       426
 weighted avg       0.98      0.98      0.98       426
```

```
cf= confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix for SVM')
```

```
Text(0.5, 1.0, 'Confusion Matrix for SVM')
```




```
from sklearn.neighbors import KNeighborsClassifier
clf_knn = KNeighborsClassifier(n_neighbors=5)
clf_knn.fit(X_train, y_train)
y_pred_knn = clf_knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'KNN Accuracy: {accuracy_knn:.2f}')
print('\nKNN Classification Report:\n', classification_report(y_test, y_pred_knn))
```

```
KNN Accuracy: 0.92

KNN Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.97      0.95       333
           1       0.83      0.67      0.74        64
           2       0.96      0.93      0.95        29

    accuracy                           0.92       426
   macro avg       0.91      0.86      0.88       426
weighted avg       0.92      0.92      0.92       426
```

```
cf= confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix for KNN')
```

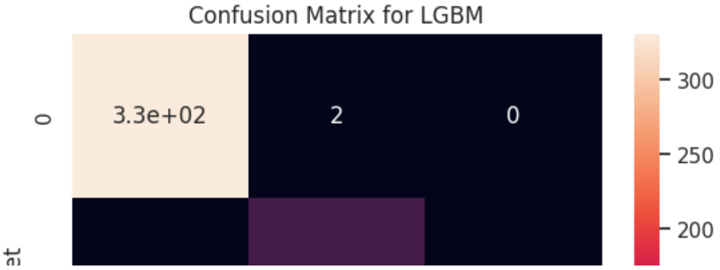
```
Text(0.5, 1.0, 'Confusion Matrix for KNN')
```


```
from lightgbm import LGBMClassifier
clf_lgbm = LGBMClassifier(n_estimators=100, random_state=42)
clf_lgbm.fit(X_train, y_train)
y_pred_lgbm = clf_lgbm.predict(X_test)
accuracy_lgbm = accuracy_score(y_test, y_pred_lgbm)
print(f'LGBM Accuracy: {accuracy_lgbm:.2f}')
print('\nLGBM Classification Report:\n', classification_report(y_test, y_pred_lgbm))
```

```
[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 214
[LightGBM] [Info] Number of data points in the train set: 1700, number of used features: 14
[LightGBM] [Info] Start training from score -0.251483
[LightGBM] [Info] Start training from score -1.995966
[LightGBM] [Info] Start training from score -2.447951
[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
[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
[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
LGBM Accuracy: 0.98

LGBM Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       333
           1       0.95      0.94      0.94        64
           2       1.00      0.97      0.98        29

    accuracy                           0.98       426
   macro avg       0.98      0.97      0.97       426
weighted avg       0.98      0.98      0.98       426
```

```
cf= confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix for LGBM')
```

Text(0.5, 1.0, 'Confusion Matrix for LGBM')

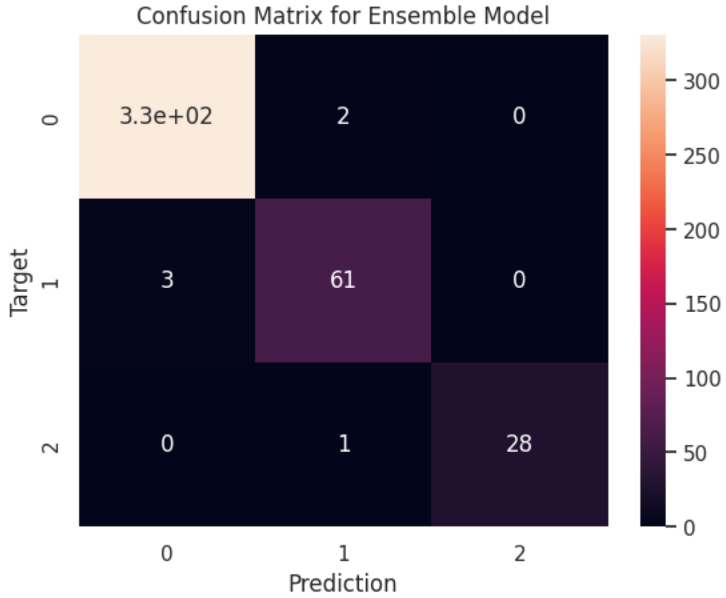



```
from sklearn.ensemble import VotingClassifier
svm_clf = SVC(kernel='linear', C=1.0, probability=True, random_state=42)
xgb_clf = XGBClassifier(n_estimators=100, random_state=42)
lgbm_clf = LGBMClassifier(n_estimators=100, random_state=42)
knn_clf = KNeighborsClassifier(n_neighbors=5)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
ensemble_clf = VotingClassifier(estimators=[
    ('svm', svm_clf),
    ('xgb', xgb_clf),
    ('lgbm', lgbm_clf),
    ('knn', knn_clf),
    ('rf', rf_clf)
], voting='soft')
ensemble_clf.fit(X_train, y_train)
y_pred_ensemble = ensemble_clf.predict(X_test)
accuracy_ensemble = accuracy_score(y_test, y_pred_ensemble)
print(f'Ensemble Model Accuracy: {accuracy_ensemble:.2f}')
print('\nEnsemble Model Classification Report:\n', classification_report(y_test, y_pred_ensemble))
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000201 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 214
[LightGBM] [Info] Number of data points in the train set: 1700, number of used features: 14
[LightGBM] [Info] Start training from score -0.251483
[LightGBM] [Info] Start training from score -1.995966
[LightGBM] [Info] Start training from score -2.447951
[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
[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
[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
Ensemble Model Accuracy: 0.98

Ensemble Model Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       333
           1       0.95      0.92      0.94        64
           2       1.00      0.97      0.98        29

    accuracy                           0.98       426
   macro avg       0.98      0.96      0.97       426
weighted avg       0.98      0.98      0.98       426
```

```
cf= confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix for Ensemble Model')
```

```
Text(0.5, 1.0, 'Confusion Matrix for Ensemble Model')
```




```
accuracies = [
    ('Random Forest', accuracy_rf),
    ('XGBoost', accuracy_xgb),
    ('SVM', accuracy_svm),
    ('KNN', accuracy_knn),
    ('LGBM', accuracy_lgbm),
    ('Ensemble', accuracy_ensemble)
]

accuracies_df = pd.DataFrame(accuracies, columns=['Classifier', 'Accuracy'])
plt.figure(figsize=(10, 6))
sns.set(style="whitegrid")
sns.barplot(x='Classifier', y='Accuracy', data=accuracies_df, palette="viridis")
plt.title('Classifier Accuracies Comparison')
plt.yticks([i/10 for i in range(11)])  # Set y-axis ticks in intervals of 0.1
plt.ylim(0, 1)
plt.show()
```