In [ ]:

```
import numpy as np
import pandas as pd
```

In [ ]:

```
df = pd.read_csv('../input/sms-spam-collection-data-set/SMSSpamCollection',sep='\t',names=[
```

In [ ]:

```
df
```

Data Pre-processing

In [ ]:

```
df.shape
```

In [ ]:

```
import nltk #!pip install nltk
```

In [ ]:

```
nltk.download('stopwords')
```

In [ ]:

```
sent = 'How are you friends?'
```

In [ ]:

```
from nltk.tokenize import word_tokenize
word_tokenize(sent)
```

In [ ]:

```
from nltk.corpus import stopwords
swords = stopwords.words('english')
```

In [ ]:

```
clean = [word for word in word_tokenize(sent) if word not in swords]
```

In [ ]:

```
clean
```

In [ ]:

```
# Stemming words with NLTK
from nltk.stem import PorterStemmer
ps = PorterStemmer()
clean = [ps.stem(word) for word in word_tokenize(sent)
         if word not in swords]
clean
```

In [ ]:

```
sent = 'Hello friends! How are you? We will learning python today'
```

In [ ]:

```
def clean_text(sent):
    tokens = word_tokenize(sent)
    clean = [word for word in tokens if word.isdigit() or word.isalpha()]
    clean = [ps.stem(word) for word in clean
         if word not in swords]
    return clean
```

In [ ]:

```
clean_text(sent)
```

In [ ]:

```
# Pre-processing
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [ ]:

```
tfidf = TfidfVectorizer(analyzer=clean_text)
```

In [ ]:

```
x = df['text']
y = df['label']
```

In [ ]:

```
x_new = tfidf.fit_transform(x)
```

In [ ]:

```
x.shape
```

In [ ]:

```
x_new.shape
```

In [ ]:

```
# tfidf.get_feature_names()
```

In [ ]:

```
import seaborn as sns
sns.countplot(x=y)
```

In [ ]:

```
#cross validation
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_new,y,test_size=0.25,random_state=1)
```

In [ ]:

```
print(f"Size of splitted data")
print(f"x_train {x_train.shape}")
print(f"y_train {y_train.shape}")
print(f"y_test {x_test.shape}")
print(f"y_test {y_test.shape}")
```

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
```

In [ ]:

```
nb = GaussianNB()
nb.fit(x_train.toarray(),y_train)
y_pred_nb = nb.predict(x_test.toarray())
```

In [ ]:

```
y_test.value_counts()
```

In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

In [ ]:

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_nb)
plt.title('Naive bayes')
plt.show()
print(f" Accuracy is {accuracy_score(y_test,y_pred_nb)}")
print(classification_report(y_test,y_pred_nb))
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(random_state=1)
model_rf.fit(x_train,y_train)
```

In [ ]:

```
y_pred_rf = model_rf.predict(x_test) #float
```

In [ ]:

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_rf)
plt.title('Random Forest')
plt.show()
print(f" Accuracy is {accuracy_score(y_test,y_pred_rf)}")
print(classification_report(y_test,y_pred_rf))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression(random_state=1)

model_lr.fit(x_train,y_train)
y_pred_lr = model_lr.predict(x_test)
```

In [ ]:

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_lr)
plt.title('Logistic regression')
plt.show()
print(f" Accuracy is {accuracy_score(y_test,y_pred_lr)}")
print(classification_report(y_test,y_pred_lr))
```

Hyper parameter tunning

In [ ]:

```
from sklearn.model_selection import GridSearchCV
```

In [ ]:

```
para = {

    'criterion':['gini', 'entropy','log_loss'],
#     'max_features': ['sqrt','log2'],
#     'random_state': [0,1,2,3,4],
    'class_weight':['balanced','balanced_subsample']
}
```

In [ ]:

```
grid = GridSearchCV(model_rf, param_grid=para, cv=5, scoring='accuracy')
```

In [ ]:

```
grid.fit(x_train,y_train)
```

In [ ]:

```
rf = grid.best_estimator_
```

In [ ]:

```
y_pred_grid = rf.predict(x_test)
```

In [ ]:

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_grid)
plt.title('Gride Search')
plt.show()
print(f" Accuracy is {accuracy_score(y_test,y_pred_grid)}")
print(classification_report(y_test,y_pred_grid))
```

In [ ]:

```

```