In [1]:
```
import numpy as np # linear algebra
import pandas as pd
```

In [2]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [1]:
```
import numpy as np
import pandas as pd
```

In [24]:
```
pwd
```

Out[24]:
```
'C:\\Users\\admin'
```

In [27]:
```
df = pd.read_csv(r'C:\Users\admin\SMSSpamCollection',sep='\t',names=['label','text'
```

In [28]:
```
df .shape
```

Out[28]:
```
(5572, 2)
```

In [29]:
```
import nltk
```

In [30]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[30]:
```
True
```

In [31]:
```
sent = 'How are you friends?'
```

In [32]:
```
from nltk.tokenize import word_tokenize
word_tokenize(sent)
```

Out[32]:
```
['How', 'are', 'you', 'friends', '?']
```

In [33]:
```
from nltk.corpus import stopwords
swords = stopwords.words('english')
```

In [34]:
```
clean = [word for word in word_tokenize(sent) if word not in swords]
```

In [35]:
```
clean
```

Out[35]:
```
['How', 'friends', '?']
```

In [36]:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
clean = [ps.stem(word) for word in word_tokenize(sent)
         if word not in swords]
clean
```

Out[36]:
```
['how', 'friend', '?']
```

In [37]:
```
sent = 'Hello friends! How are you? We will learning python today'
```

In [38]:
```
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 [39]:
```
clean_text(sent)
```

Out[39]:
```
['hello', 'friend', 'how', 'we', 'learn', 'python', 'today']
```

In [40]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Pre-processing
```

In [41]:
```
tfidf = TfidfVectorizer(analyzer=clean_text)
```

In [42]:
```
x = df['text']
y = df['label']
```

In [43]:
```
x_new = tfidf.fit_transform(x)
```

In [44]:
```
x.shape
```

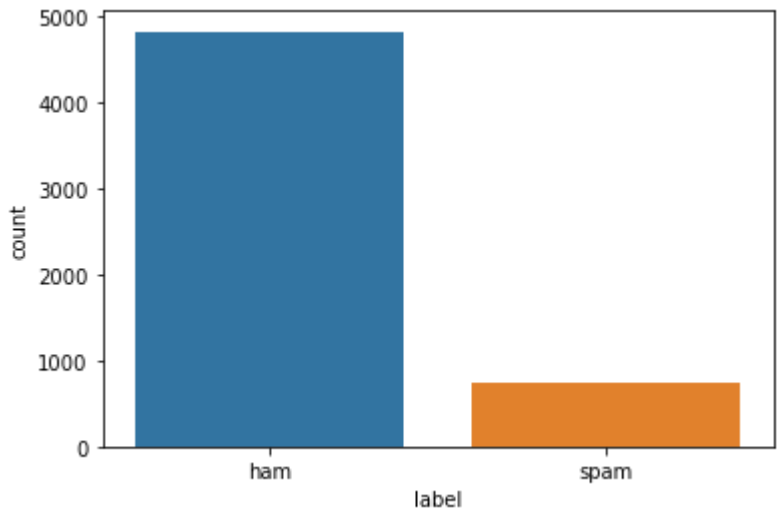
Out[44]:
```
(5572,)
```

In [45]:
```
x_new.shape
```

Out[45]:
```
(5572, 6513)
```

In [47]:
```
import seaborn as sns
sns.countplot(x=y)
```

Out[47]:
```
<AxesSubplot:xlabel='label', ylabel='count'>
```



In [48]:
```
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_stat

#cross validation
```

In [49]:
```
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}")
```

```
Size of splitted data
x_train (4179, 6513)
y_train (4179,)
y_test (1393, 6513)
y_test (1393,)
```

In [50]:
```
from sklearn.naive_bayes import GaussianNB
```

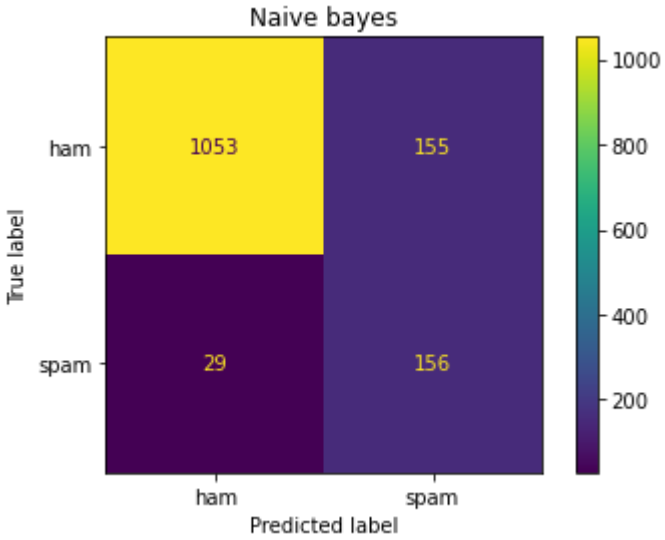
In [51]:
```
nb = GaussianNB()
nb.fit(x_train.toarray(),y_train)
y_pred_nb = nb.predict(x_test.toarray())
```

In [52]:
```
y_test.value_counts()
```

Out[52]:
```
ham     1208
spam     185
Name: label, dtype: int64
```

In [53]:
```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

In [54]:
```
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))
```



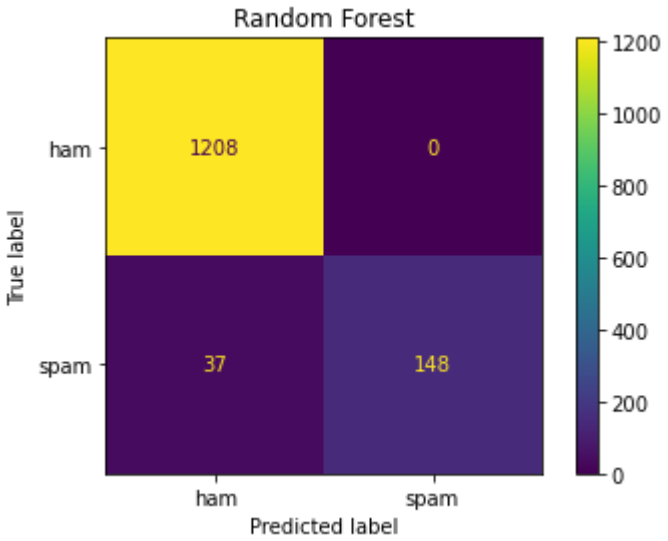

```
 Accuracy is 0.867910983488873
              precision    recall  f1-score   support

         ham       0.97      0.87      0.92      1208
        spam       0.50      0.84      0.63       185

    accuracy                           0.87      1393
   macro avg       0.74      0.86      0.77      1393
weighted avg       0.91      0.87      0.88      1393
```

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(random_state=1)
model_rf.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [57]:
```
y_pred_rf = model_rf.predict(x_test)

#float
```

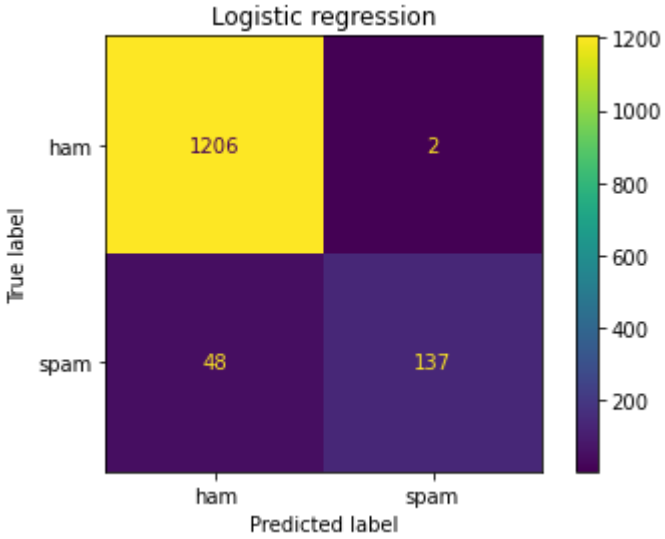
In [58]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_rf)
plt.title('Random Forest')
plt.show()
print(f" Accuracy is {accuracy_score(y_test,y_pred_rf)}")
print(classification_report(y_test,y_pred_rf))
```




```
 Accuracy is 0.9734386216798278
              precision    recall  f1-score   support

         ham       0.97      1.00      0.98      1208
        spam       1.00      0.80      0.89       185

    accuracy                           0.97      1393
   macro avg       0.99      0.90      0.94      1393
weighted avg       0.97      0.97      0.97      1393
```

In [59]:
```
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 [60]:
```
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))
```




```
Accuracy is 0.9641062455132807
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98      1208
        spam       0.99      0.74      0.85       185

    accuracy                           0.96      1393
   macro avg       0.97      0.87      0.91      1393
weighted avg       0.96      0.96      0.96      1393
```

In [61]:
```
from sklearn.model_selection import GridSearchCV

# Hyper parameter tunning
```

In [62]:
```
para = {

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

In [63]:
```
grid = GridSearchCV(model_rf, param_grid=para, cv=5, scoring='accuracy')
```

In [64]:
```
grid.fit(x_train,y_train)
```

Out[64]:
```
GridSearchCV
best_estimator_: RandomForestClassifier
RandomForestClassifier
```

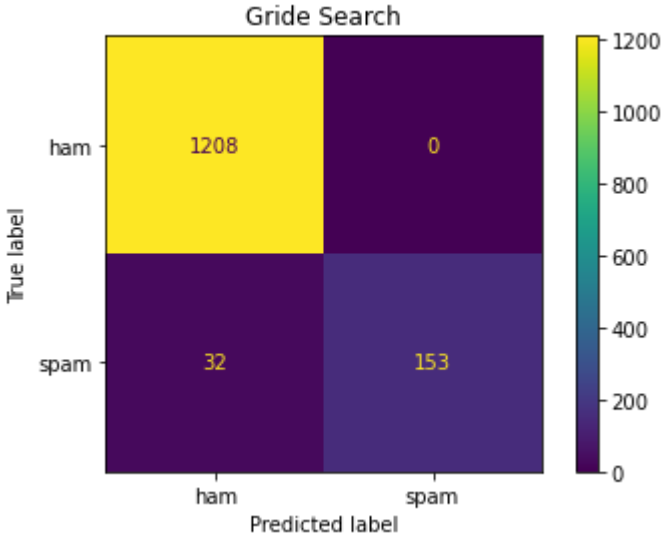
In [65]:
```
grid.fit(x_train,y_train)
```

Out[65]:
```
GridSearchCV
best_estimator_: RandomForestClassifier
RandomForestClassifier
```

In [68]:
```
rf = grid.best_estimator_
```

In [69]:
```
y_pred_grid = rf.predict(x_test)
```

In [70]:
```
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))
```




```
Accuracy is 0.9770279971284996
              precision    recall  f1-score   support

         ham       0.97      1.00      0.99      1208
        spam       1.00      0.83      0.91       185

    accuracy                           0.98      1393
   macro avg       0.99      0.91      0.95      1393
weighted avg       0.98      0.98      0.98      1393
```

In [ ]:
```

```