REPORT-LAB2

Unsupervised Learning and Introduction to Text Mining

Jingye Zhang

TASK1: Clustering

The following experiments all use *housing.csv* as the dataset, and its link is: *https://www.kaggle.com/datasets/camnugent/california-housing-prices*

Task1-1: K-Means

K-Means belongs to the segmentation clustering algorithm, and its core idea is to iteratively divide data objects into different clusters in order to minimize the objective function, so that the generated clusters are as compact and independent as possible. The algorithm steps are as follows:

1) First, randomly select k objects as the centroids of the initial k clusters;

2) Then, assign the remaining objects to the nearest cluster according to their distance from the centroid of each cluster; then find the centroid of the newly formed cluster.

This iterative relocation process is repeated until the objective function is minimized.

The python code is implemented as follows:

1) Import required packages and datasets.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```
df = pd.read_csv("housing.csv")
#df.columns
```

```
df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
...	...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0	INLAND
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0	INLAND
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0	INLAND
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0	INLAND
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0	INLAND

20640 rows × 10 columns

2) Select three columns of data, namely *housing_median_age*, *total_rooms* and *median_house_value*, and normalize the mean variance of the data.

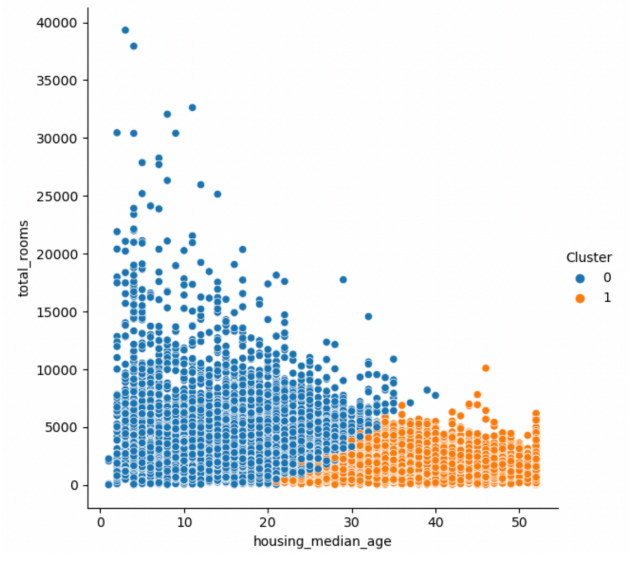
```
X = df[['housing_median_age', 'total_rooms', 'median_house_value']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3) Set the k-value which is the value of n_clusters, now I set n_clusters=2 (I also try to set n_clusters=3, the results will be shown later).

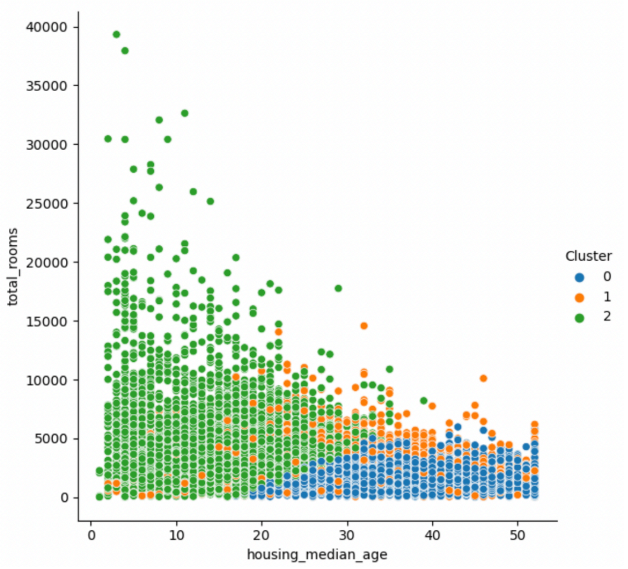
```
kmeans = KMeans(n_clusters=2)
X["Cluster"] = kmeans.fit_predict(X_scaled)
X["Cluster"] = X["Cluster"].astype("category")
#X.head()
```

4) Plot the clustering results.Left figure shows the result when n_clusters=2, right figure shows the result when n_clusters=3.

```
sns.relplot(
    x="housing_median_age", y="total_rooms", hue="Cluster", data=X, height=6,
);
```




```
sns.relplot(
    x="housing_median_age", y="total_rooms", hue="Cluster", data=X, height=6,
);
```



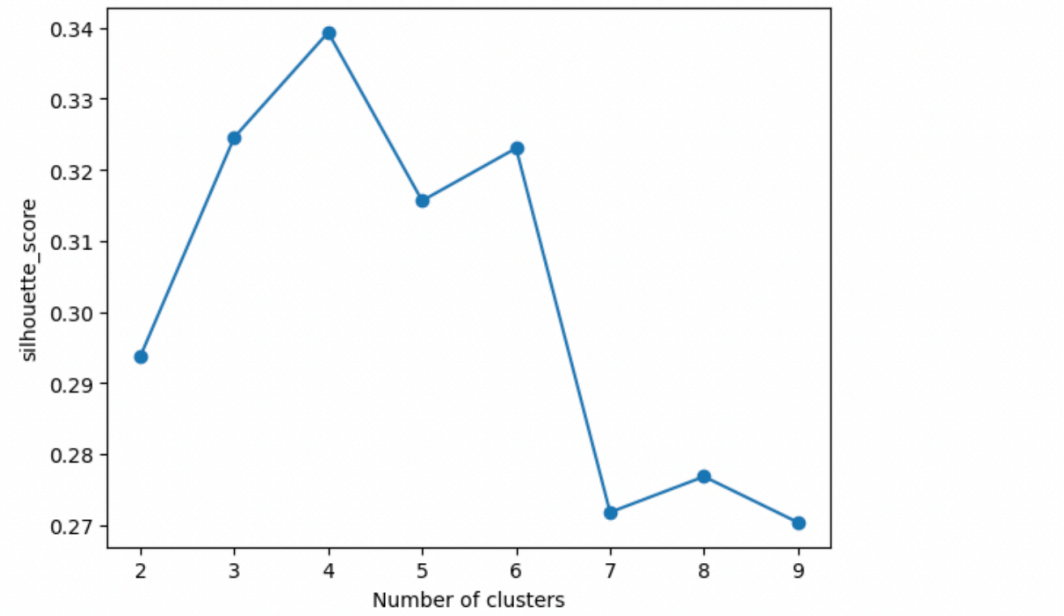
5) I use Silhouette Coefficient to evaluating the performance of this clustering algorithm. Silhouette coefficients can provide a more objective way to determine the optimal number of clusters. This is done by simply computing the silhouette coefficient over the k range and identifying the peak as the best k-value. Performs K-Means clustering over a range of k, finds the best k-value that yields the largest silhouette coefficient, and assigns data points to clusters based on the optimized k-value.

The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

As shown below, the perfect k-value should be 4, because when the number of clusters equals to 4, the silhouette_score reach to the highest which is about 0.34. However, when n_clusters=2, the silhouette_score is about 0.293, and when m_clusters=3, the silhouette_score is about 0.325.

6) Evaluating the running time. The value is about 0.000047584.

```
from sklearn import metrics
scores = []
for i in range(2, 10):
    km = KMeans(        n_clusters=i,
                        init='k-means++',
                        n_init=10,
                        max_iter=30,
                        random_state=0      )
    km.fit(X_scaled)
    scores.append(metrics.silhouette_score(X_scaled, km.labels_ , metric='euclidean'))
plt.plot(range(2,10), scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('silhouette_score')
plt.show()
```



```
from timeit import default_timer
start = default_timer()
duration = default_timer() - start
print(duration)
```

```
4.758400000071106e-05
```

Task1-2: K-Medoids

K-Medoids also belongs to the segmentation clustering algorithm. The K-Medoids (central point) algorithm does not use the average value, but instead uses the most central object in the cluster, that is, the central point (medoids) as a reference point. The algorithm steps are similar to K-means, which is essentially an improvement and optimization of K-means.

The python code is implemented as follows:

1) Import required packages and datasets. Then select the three columns of data, and normalize the mean variance of the data.

```
from pyclust import KMedoids
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
data=pd.read_csv("housing.csv")
X = data[['housing_median_age', 'total_rooms', 'median_house_value']]
X.head()
```

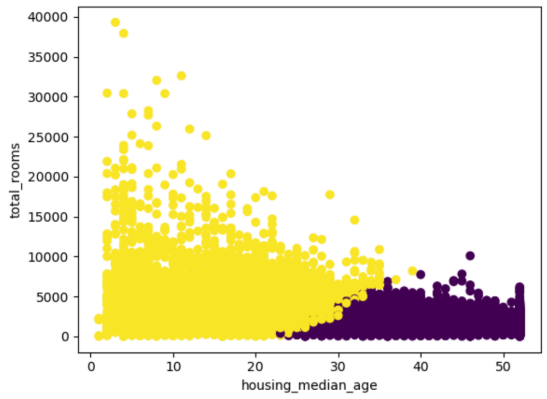
	housing_median_age	total_rooms	median_house_value
0	41.0	880.0	452600.0
1	21.0	7099.0	358500.0
2	52.0	1467.0	352100.0
3	52.0	1274.0	341300.0
4	52.0	1627.0	342200.0

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2) Set the k-value which is the value of n_clusters and plot the clustering results. The left figure shows the result when n_clusters=2, the right figure shows the result when n_cluster=3.

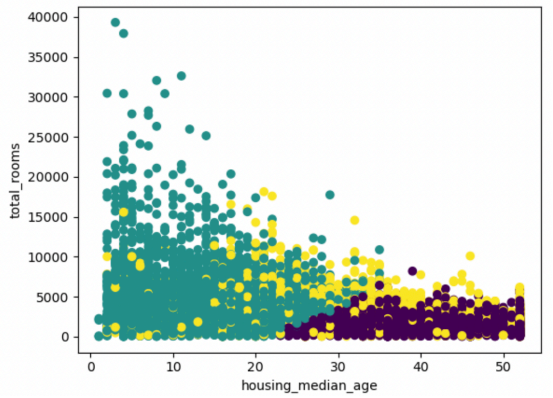
```
kmedoids = KMedoids(n_clusters=2)
kmedoids.fit(X_scaled)
y_kmed = kmedoids.fit_predict(X_scaled)
```

```
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y_kmed)
plt.xlabel('housing_median_age')
plt.ylabel('total_rooms')
plt.show()
```



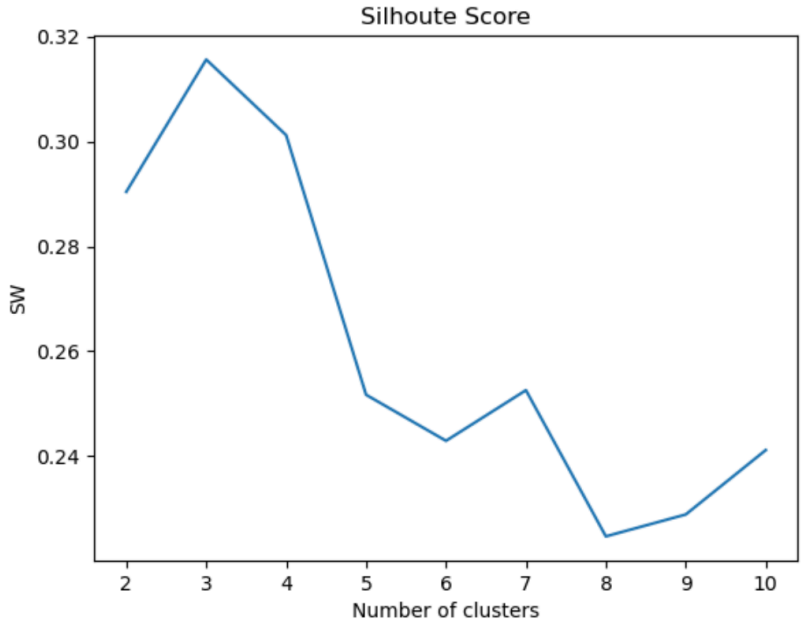

```
kmedoids = KMedoids(n_clusters=3)
kmedoids.fit(X_scaled)
y_kmed = kmedoids.fit_predict(X_scaled)
```

```
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y_kmed)
plt.xlabel('housing_median_age')
plt.ylabel('total_rooms')
plt.show()
```



3) Still using Silhouette Coefficient to evaluate the perfect k-value. As shown below, we should choose 3 for the k-value. Meanwhile, the silhouette_score is about 0.317. And when n_clusters=2, the silhouette_score is about 0.29.

```
sw = []
for i in range(2, 11):
    kMedoids = KMedoids(n_clusters = i, random_state = 0)
    kMedoids.fit(X_scaled)
    y_kmed = kMedoids.fit_predict(X_scaled)
    silhouette_avg = silhouette_score(X_scaled, y_kmed)
    sw.append(silhouette_avg)
plt.plot(range(2, 11), sw)
plt.title('Silhoute Score')
plt.xlabel('Number of clusters')
plt.ylabel('SW')      #within cluster sum of squares
plt.show()
```



4) Evaluating the running time. As shown below, the value is about 0.00020796.

```
from timeit import default_timer
start = default_timer()
duration = default_timer() - start
print(duration)
```

```
0.00020795799999190173
```

Task1-3: BIRCH

BIRCH belongs to hierarchical clustering. Its full name is Balanced Iterative Reducing and Clustering Using Hierarchies, which uses a clustering feature tree (Clustering Feature Tree, CF Tree for short) similar to a B+ tree for fast clustering. Unlike the K-Means algorithm, the BIRCH algorithm does not need to input the k-value of the category number. If we do not input the k-value, the number of groups of the last CF tuple is the final k, otherwise, the CF tuples will be merged according to the distance according to the input k-value. Generally speaking, the BIRCH algorithm is suitable for situations with a large sample size. In addition to clustering, it can also do some outlier detection and preprocessing of data preliminary classification by category.

The python code is implemented as follows:

1) Import required packages and datasets. Then select the three columns of data, and normalize the mean variance of the data.

```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
sns.set()
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.cluster import Birch
```

```
df = pd.read_csv("housing.csv")
```

```
X = df[['housing_median_age', 'total_rooms', 'median_house_value']]
X
```

	housing_median_age	total_rooms	median_house_value
0	41.0	880.0	452600.0
1	21.0	7099.0	358500.0
2	52.0	1467.0	352100.0
3	52.0	1274.0	341300.0
4	52.0	1627.0	342200.0
...	...	...	...
20635	25.0	1665.0	78100.0
20636	18.0	697.0	77100.0
20637	17.0	2254.0	92300.0
20638	18.0	1860.0	84700.0
20639	16.0	2785.0	89400.0

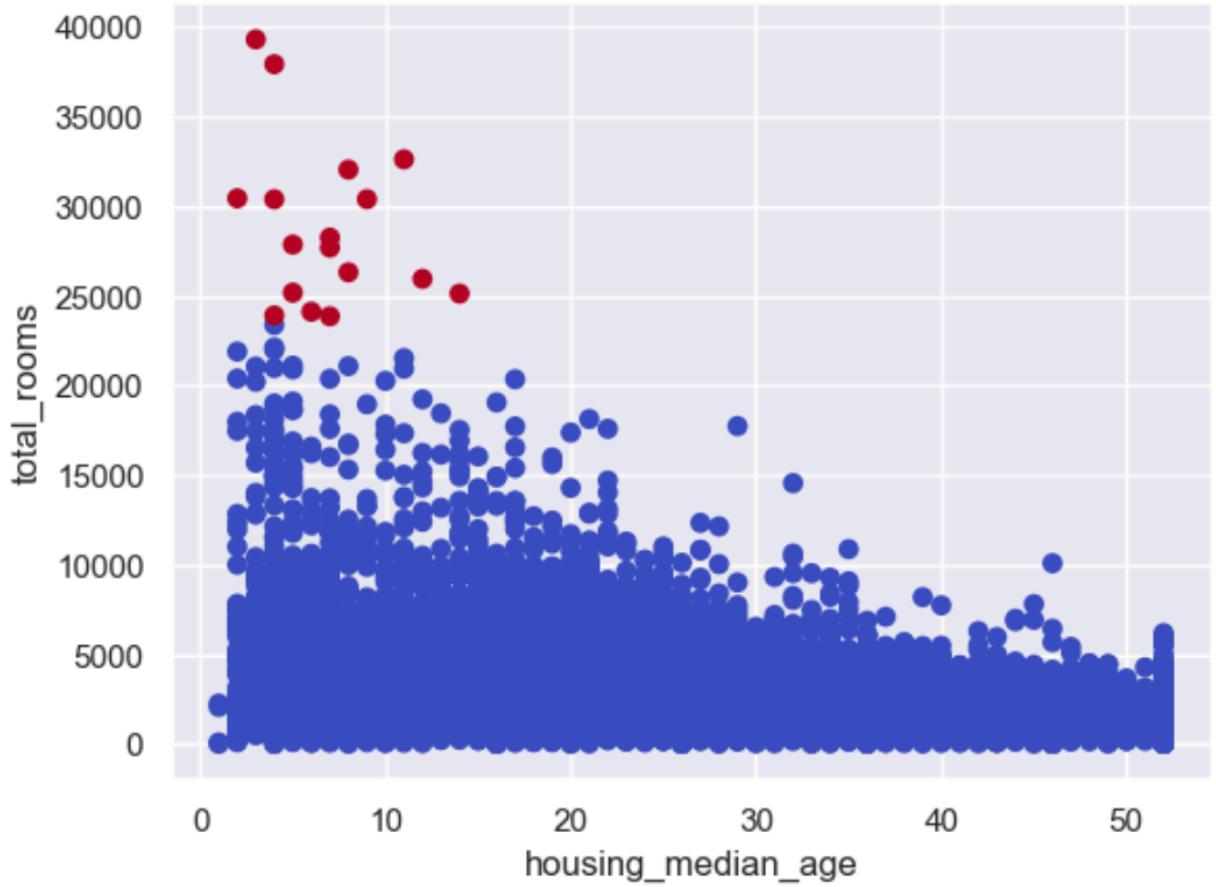
20640 rows × 3 columns

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2) Set the k-value which is the value of n_clusters and plot the clustering results. Now I set n_cluster=2, and calculate the Silhouette Coefficient. As shown below, the silhouette_score is about 0.61134.

```
clusters = make_blobs(centers = 8, cluster_std = 0.75, random_state = 0)
model = Birch(branching_factor = 50, n_clusters = 2, threshold = 1.5)
model.fit(X)
y_birch = model.fit_predict(X_scaled)
```

```
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y_birch, cmap='coolwarm')
plt.xlabel('housing_median_age')
plt.ylabel('total_rooms')
plt.show()
```



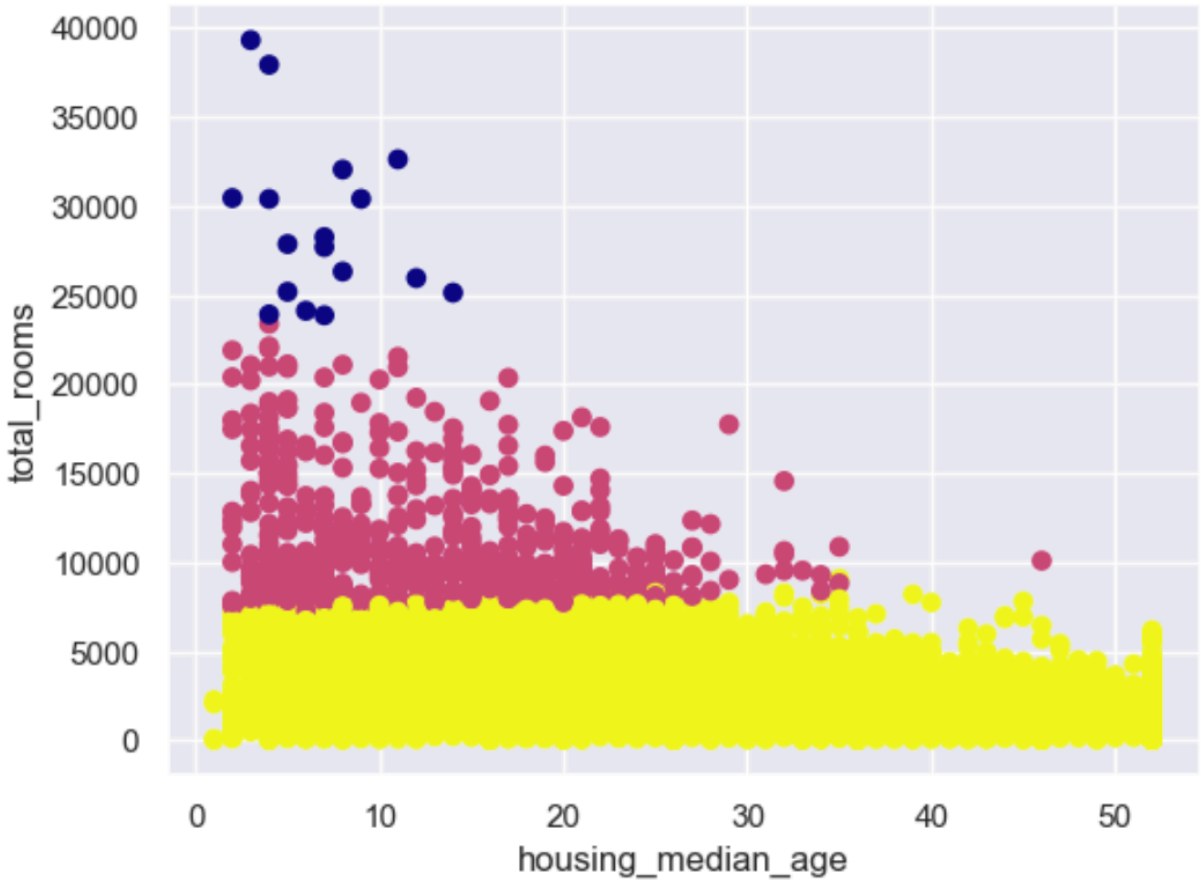

```
model = Birch(branching_factor = 50, n_clusters = 2, threshold = 1.5)
model.fit(X)
labels = model.labels_
metrics.silhouette_score(X, labels, metric='euclidean')
```

```
0.6113372359105868
```

3)Now I try to set the n_clusters=3, and calculate the Silhouette Coefficient. As shown below, the silhouette_score is about 0.581046.

```
clusters = make_blobs(centers = 8, cluster_std = 0.75, random_state = 0)
model = Birch(branching_factor = 50, n_clusters = 3, threshold = 1.5)
model.fit(X)
y_birch = model.fit_predict(X_scaled)
```

```
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y_birch, cmap='plasma')
plt.xlabel('housing_median_age')
plt.ylabel('total_rooms')
plt.show()
```



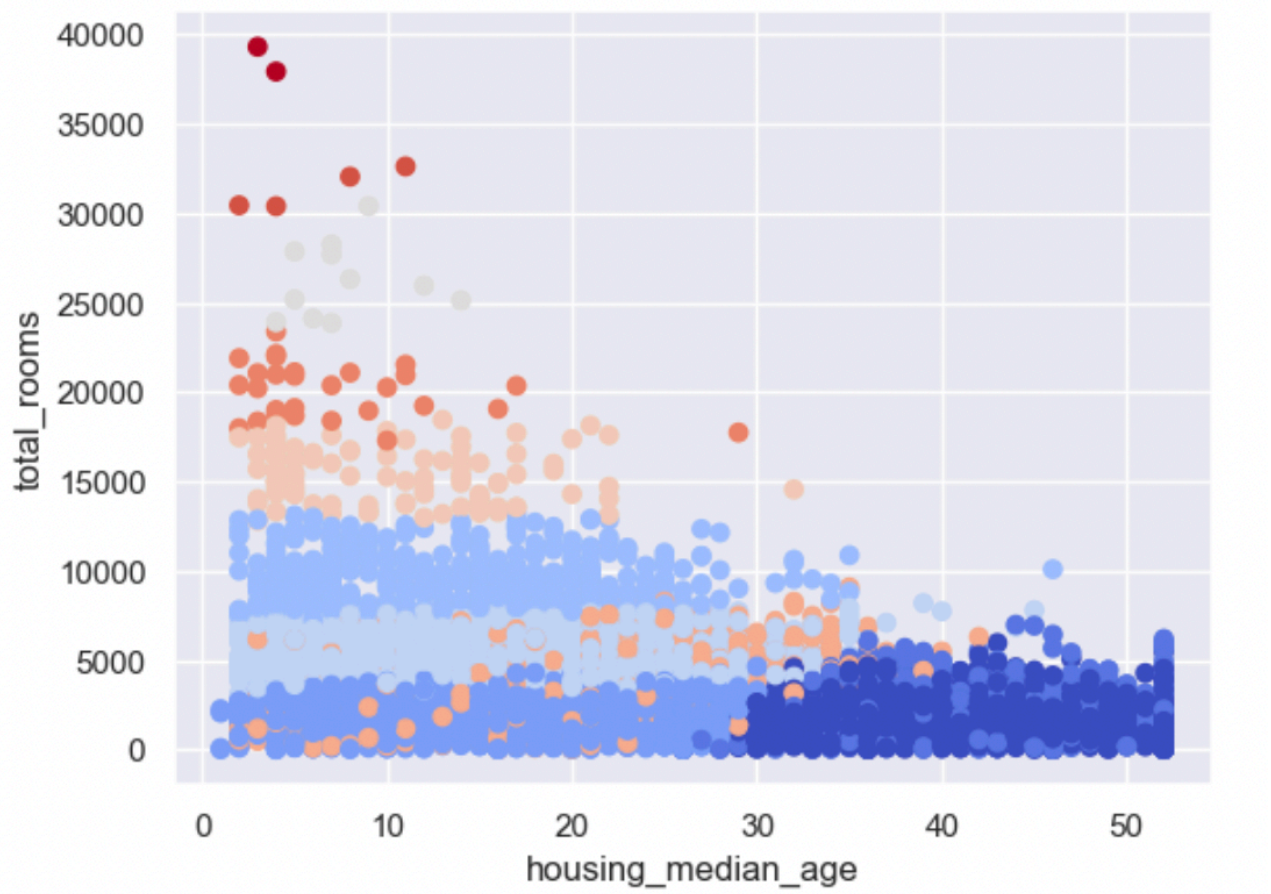

```
model = Birch(branching_factor = 50, n_clusters = 3, threshold = 1.5)
model.fit(X)
labels = model.labels_
metrics.silhouette_score(X, labels, metric='euclidean')
```

```
0.5810458725777637
```

4) Then I try to skip setting the k-value, which means I set n_clusters=None, and the result and the silhouette_score are shown below. As we can see, when we do not input the k-value, the silhouette_score is about 0.00480476.

```
clusters = make_blobs(centers = 8, cluster_std = 0.75, random_state = 0)
model = Birch(branching_factor = 50, n_clusters = None, threshold = 1.5)
model.fit(X)
y_birch = model.fit_predict(X_scaled)
```

```
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y_birch, cmap='coolwarm')
plt.xlabel('housing_median_age')
plt.ylabel('total_rooms')
plt.show()
```




```
model = Birch(branching_factor = 50, n_clusters = None, threshold = 1.5)
model.fit(X)
labels = model.labels_
metrics.silhouette_score(X, labels, metric='euclidean')
```

```
0.004804758002310301
```

5) Evaluating the running time. As shown below, the value is about 0.00002538.

```
from timeit import default_timer
start = default_timer()
duration = default_timer() - start
print(duration)
```

```
2.537500040489249e-05
```

Task1-4: Conclusion and Compare

1) K-Means: It can efficiently classify large datasets with a computational complexity of O(tkmn), however it usually terminates when a local optimum is obtained.

2) K-medoids: It is more robust than K-means in the presence of noise and outliers. However, K-medoids work well for small datasets, but not well for large datasets, and the step time complexity of calculating the centroid is O(n^2), which runs slower.

3) BIRCH: The clustering speed is fast, and the CF Tree can be established by only scanning the training set once, and the addition, deletion, and modification of the CF Tree are very fast. However, the data clustering effect on high-dimensional features is not good. At this time, we can choose Mini Batch K-Means

	k=2 -> silhouette_score	k=3 -> silhouette_score	running time
K-Means	0.293	0.325	0.000047584
K-Medoids	0.29	0.317	0.00020796
BIRCH	0.61134	0.581046	0.00002538

TASK2: Text Mining

For most of the experiments in task2 I used the dataset named *spam.csv*, and its link is: *https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset*

I used this dataset for classifying the type of email, which is labeled as *ham* and *spam.* Including the machine learning techniques for auto-predict the type of the email and print the confusion matrix for each models. I have already explain in annotations some of the important techniques I have used with the code.

Task2-1: Data Processing + NLP

The python code is implemented as follows:

1) Import some required libraries and packages.

```
# basics
import random
import string
import pandas as pd

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly_express as px
from wordcloud import WordCloud

# Natural Language Processing
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
# nltk.download('stopwords')
# nltk.download('wordnet')
# nltk.download('punkt')
# nltk.download('omw-1.4')
#nltk.download('averaged_perceptron_tagger')

# Machine Learning
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score, f1_score, recall_score
```

2) Import the dataset. Since I got some errors about type of encoding, so I use *result = chardet.detect(rawdata.read(100000))* for checking the type.

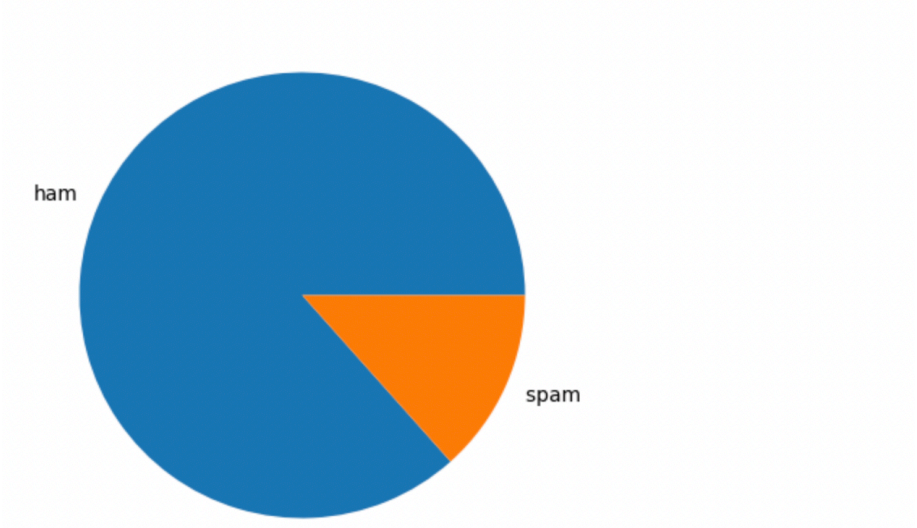
```
#search the type of encoding ---> utf-8
# import chardet
# with open("spam.csv", 'rb') as rawdata:
#     result = chardet.detect(rawdata.read(100000))
# result
```

```
data = pd.read_csv("spam.csv", encoding='utf-8')
data.head()
data = data.rename(columns={'v1':'label'})
data = data.rename(columns={'v2':'text'})
data.head()
```

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

3) Output the pie chart to see the percentage of each label.

```
#output a pie chart and see the percentage of each label----->"ham"&"spam"
fig = plt.figure(figsize=(8, 5))
plt.pie(data['label'].value_counts(), labels = ['ham', 'spam '])
plt.show()
```



4) Add new features named Len and words, which means the length of the text and the amount of the word in each text.

```
#add a new feature named "len" to the dataframe, means the length of each text.
data['len'] = data['text'].apply(len)
data.head()
```

	label	text	len	words	final_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	29	8	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	nah i think goe usf live around though

```
#add a new feature named "words", means the amount of words in each sentence(text)
# split strings into section is tokenization
data['words'] = data['text'].apply(lambda x: len(nltk.word_tokenize(x))) # to produce a list of words
round(data['words'].describe(), 2)
```

```
count    5572.00
mean       18.70
std        13.74
min         1.00
25%         9.00
50%        15.00
75%        27.00
max       220.00
Name: words, dtype: float64
```

```
data.head()
```

	label	text	len	words
0	ham	Go until jurong point, crazy.. Available only ...	111	24
1	ham	Ok lar... Joking wif u oni...	29	8
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	37
3	ham	U dun say so early hor... U c already then say...	49	13
4	ham	Nah I don't think he goes to usf, he lives aro...	61	15

5) Data Preprocessing + NLP. For this part I took some simple sentences as the examples to show how these methods works.

i) Sentence Splitting

```
#take this sentence as an example:
ex = 'I am a student of Computer Science and Sustainable Development at HKR!!:) I would like to learn more about NLP...
#step1: remove all the punctuation marks and make this ex to nopunct_ex.
punct = list(string.punctuation)
#print(punct)
nopunct_ex = [letter for letter in ex if letter not in punct]
nopunct_ex = ''.join(nopunct_ex)
print(nopunct_ex)
print(nopunct_ex.split())
print(type(nopunct_ex))
```

```
I am a student of Computer Science and Sustainable Development at HKR I would like to learn more about NLP
['I', 'am', 'a', 'student', 'of', 'Computer', 'Science', 'and', 'Sustainable', 'Development', 'at', 'HKR', 'I', 'woul
d', 'like', 'to', 'learn', 'more', 'about', 'NLP']
<class 'str'>
```

```
#words into characters
char_ex = list(nopunct_ex)
print(char_ex)
```

```
['I', ' ', 'a', 'm', ' ', 'a', ' ', 's', 't', 'u', 'd', 'e', 'n', 't', ' ', 'o', 'f', ' ', 'C', 'o', 'm', 'p', 'u',
't', 'e', 'r', ' ', 'S', 'c', 'i', 'e', 'n', 'c', 'e', ' ', 'a', 'n', 'd', ' ', 'S', 'u', 's', 't', 'a', 'i', 'n',
'a', 'b', 'l', 'e', ' ', 'D', 'e', 'v', 'e', 'l', 'o', 'p', 'm', 'e', 'n', 't', ' ', 'a', 't', ' ', 'H', 'K', 'R', '
', 'I', ' ', 'w', 'o', 'u', 'l', 'd', ' ', 'l', 'i', 'k', 'e', ' ', 't', 'o', ' ', 'l', 'e', 'a', 'r', 'n', ' ', 'm',
'o', 'r', 'e', ' ', 'a', 'b', 'o', 'u', 't', ' ', 'N', 'L', 'P']
```

ii) Tokenization + Stop words removal

```
#now we try to remove all the stopwords, since stopwords will not increase any info.
#NLTK includes stopwords so we can use it directly.
#print(random.sample(stopwords.words('english'), 10))
#print(random.sample(stopwords.words('german'), 10))
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(nopunct_ex)
nostop_ex = [word for word in word_tokens if word not in stop_words]
print(word_tokens)
print(nostop_ex)
```

```
['I', 'am', 'a', 'student', 'of', 'Computer', 'Science', 'and', 'Sustainable', 'Development', 'at', 'HKR', 'I', 'woul
d', 'like', 'to', 'learn', 'more', 'about', 'NLP']
['I', 'student', 'Computer', 'Science', 'Sustainable', 'Development', 'HKR', 'I', 'would', 'like', 'learn', 'NLP']
```

```
#step2:
#a.unify all the letters to lower_letter:
nocaps_ex = [word.lower() for word in nostop_ex]
print(nocaps_ex)
```

```
['i', 'student', 'computer', 'science', 'sustainable', 'development', 'hkr', 'i', 'would', 'like', 'learn', 'nlp']
```

iii) Stemming and Lemmatization

```
#b.transform the words to the oringal form.
ps = nltk.stem.PorterStemmer()
lemma = nltk.wordnet.WordNetLemmatizer()
#take word 'walking' as an example: here is for present participle
print(ps.stem('walking'))
print(lemma.lemmatize('walking', nltk.corpus.wordnet.VERB))
```

```
walk
walk
```

```
#take word 'went' as an example: here is for past participle
print(ps.stem('went'))
print(lemma.lemmatize('went', nltk.corpus.wordnet.VERB))
```

```
went
go
```

```
# start lemmatize the nostop_ex sentence
print('stemming:')
norm_ex = [ps.stem(word) for word in nostop_ex]
print(norm_ex)

print('lemmatize:')
norm_ex = [lemma.lemmatize(word, nltk.corpus.wordnet.VERB) for word in nostop_ex]
print(norm_ex) # using dictionary words
```

```
stemming:
['i', 'student', 'comput', 'scienc', 'sustain', 'develop', 'hkr', 'i', 'would', 'like', 'learn', 'nlp']
lemmatize:
['I', 'student', 'Computer', 'Science', 'Sustainable', 'Development', 'HKR', 'I', 'would', 'like', 'learn', 'NLP']
```

iv) Part-of-speech (POS) Tagging

```
#For example, we gonna deal with two words: "like" & "hate"
words=word_tokenize('i hate study on monday. Jim like rabbit.')
nltk.pos_tag(words)
```

```
[('i', 'NN'),
 ('hate', 'VBP'),
 ('study', 'NN'),
 ('on', 'IN'),
 ('monday', 'NN'),
 ('.', '.'),
 ('Jim', 'NNP'),
 ('like', 'IN'),
 ('rabbit', 'NN'),
 ('.', '.')]
```

v) Parsing the grammatical structures

```
#Take this is an example for parsing the grammatical structures
import nltk
from nltk import CFG
grammar1 = nltk.CFG.fromstring("""
  S -> NP VP
  VP -> V NP | V NP PP
  PP -> P NP
  V -> "saw" | "ate" | "walked"
  NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
  Det -> "a" | "an" | "the" | "my"
  N -> "man" | "dog" | "cat" | "telescope" | "park"
  P -> "in" | "on" | "by" | "with"
  """)
rd_parser = nltk.RecursiveDescentParser(grammar1)
sent = 'Mary saw a dog'.split()
for t in rd_parser.parse(sent):
    print(t)
```

```
(S (NP Mary) (VP (V saw) (NP (Det a) (N dog))))
```

6) Since my idea is to processing the data and make this dataset useful, so I put the necessary actions into one function, and add the new feature named

final_text into the data frame.

```
#now we can do the 4 actions above to a dataset, and we can set a function to do this.
def preproc(message):
    nostop = " ".join([word for word in message.split(' ') if word not in stopwords.words('english')])
    nopunct = ''.join([lett for lett in nostop if lett not in list(string.punctuation)])
    nocaps = [word.lower() for word in nopunct.split(' ')]
    norm = [ps.stem(word) for word in nocaps]
    return ' '.join(norm)
data['final_text'] = data['text'].apply(preproc)
#print(data.head())
data.head()
```

	label	text	len	words	final_text
0	ham	Go until jurong point, crazy.. Available only ...	111	24	go jurong point crazi avail bugi n great world...
1	ham	Ok lar... Joking wif u oni...	29	8	ok lar joke wif u oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	free entri 2 wkli comp win fa cup final tkt 21...
3	ham	U dun say so early hor... U c already then say...	49	13	u dun say earli hor u c alreadi say
4	ham	Nah I don't think he goes to usf, he lives aro...	61	15	nah i think goe usf live around though

Till now, I basically finish the data processing part.

And after this, I try to visualise the most frequent words and use word cloud to show the data.

Task2-2: Data Visualisation

The python code is implemented as follows:

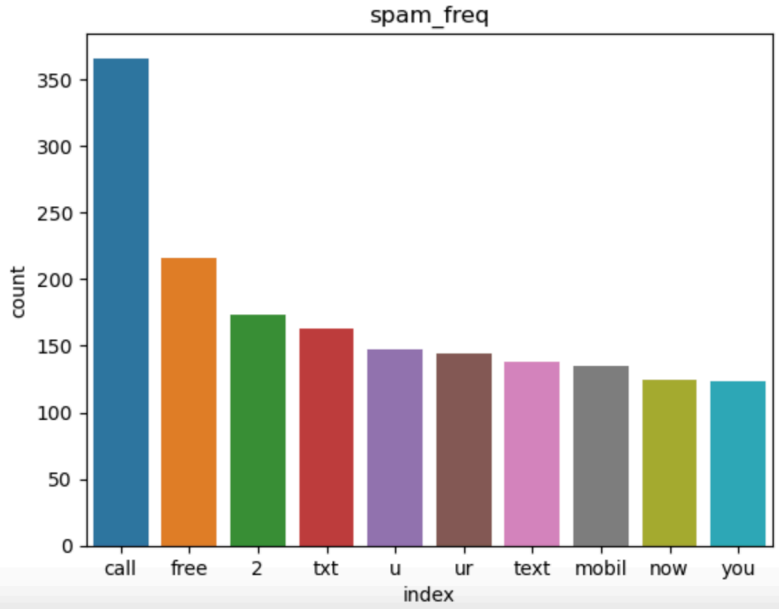
1) Calculate the frequency of *ham* and *spam*.

```
#visualize the data and try to output the most frequent word.
ham_text = " ".join(data[data.label == 'ham']['final_text'])
spam_text = " ".join(data[data['label'] == 'spam']['final_text'])
# print('------ham_text------\n' + ham_text)
# print('------spam_text------\n' + spam_text)
# calculate the frequency of each word.
ham_freq = nltk.FreqDist(nltk.word_tokenize(ham_text))
spam_freq = nltk.FreqDist(nltk.word_tokenize(spam_text))
print('------ham_freq------')
print(ham_freq)
print('------spam_freq------')
print(spam_freq)
```

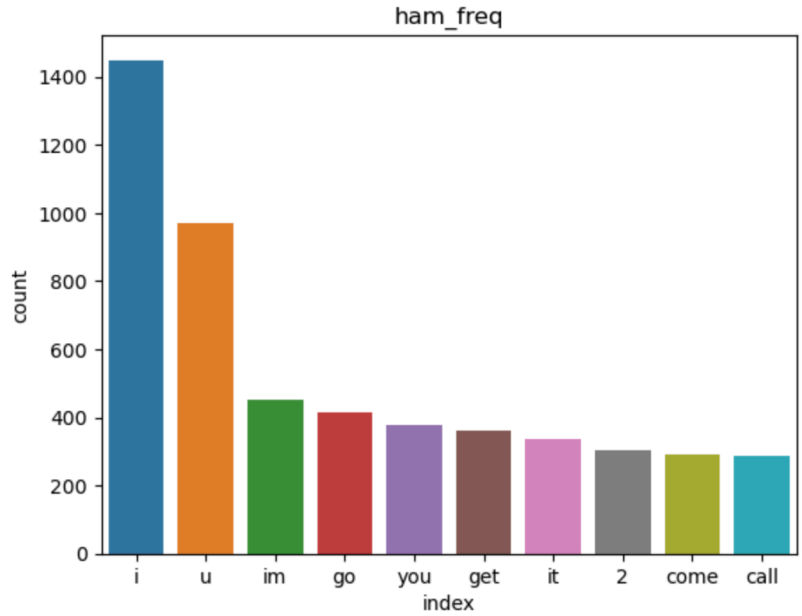
```
------ham_freq------
<FreqDist with 6339 samples and 46019 outcomes>
------spam_freq------
<FreqDist with 2781 samples and 13599 outcomes>
```

2) Visualise the frequency of these 2 data_freq.

```
spam_freq_df = pd.DataFrame.from_dict(dict(spam_freq), 'index', columns=['count']).reset_index()
#output the fisr 10 words of heigh frequency.
ax = sns.barplot(data = spam_freq_df.sort_values('count', ascending=False)[: 10], x = 'index', y = 'count')
ax.set_title("spam_freq")
plt.show()
```



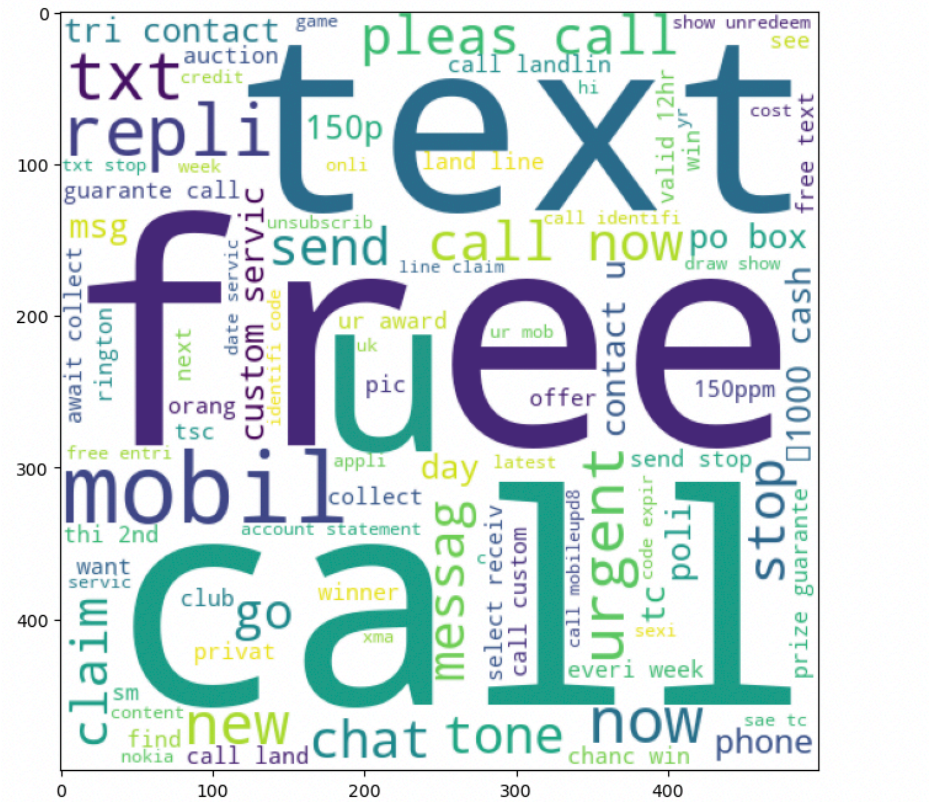

```
ham_freq_df = pd.DataFrame.from_dict(dict(ham_freq), 'index', columns=['count']).reset_index()
ax = sns.barplot(data = ham_freq_df.sort_values('count', ascending=False)[: 10], x = 'index', y = 'count')
ax.set_title('ham_freq')
plt.show()
```



3) Draw the word_cloud chart.

```
wc = WordCloud(width = 500, height = 500, min_font_size = 10, background_color = 'white')
ham_wc = wc.generate(data[data['label'] == 'ham']['final_text'].str.cat(sep = " "))
plt.figure(figsize = (20, 8))
plt.imshow(ham_wc)
plt.show()
```


```
# try to draw the word_cloud
wc = WordCloud(width = 500, height = 500, min_font_size = 10, background_color = 'white')
spam_wc = wc.generate(data[data['label'] == 'spam']['final_text'].str.cat(sep = " "))
plt.figure(figsize = (20, 8))
plt.imshow(spam_wc)
plt.show()
```



Task2-3: Machine Learning

Now I use the TF-IDF algorithm for predicting the category.

TF-IDF is a commonly used weighting technology for information retrieval and data mining. It is often used to mine keywords in articles, and the algorithm is simple and efficient. It is often used by industry for initial text data cleaning.

It consists of two parts, TF and IDF.

TF = the total number of times a word appears in the article / the total number of words in the article

IDF = log(total number of documents in the corpus / (number of documents containing the word + 1))

TF-IDF is proportional to the number of occurrences of a word in the document and inversely proportional to the number of occurrences of the word in the entire corpus.

The steps of the TF-IDF algorithm:

1) Calculate the word frequency

2) Calculate the inverse document frequency

3) Calculate TF-IDF

The python code is implemented as follows:

1) Set a TF-IDF matrix.

```
#set a TF-IDF matrix
tfidf = TfidfVectorizer(max_features = 3000)
x = tfidf.fit_transform(data['final_text']).toarray()
print(x)
print(data.label)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
0        ham
1        ham
2       spam
3        ham
4        ham
        ...
5567    spam
5568     ham
5569     ham
5570     ham
5571     ham
Name: label, Length: 5572, dtype: object
```

2) Transfer the label, which means ”ham"-->"0" & “spam”-->"1".

```
# transfer the label to "ham"-->"0" & "spam"-->"1"
encoder = LabelEncoder()
data['label'] = encoder.fit_transform(data['label'])
print(data.label)
```

```
0        0
1        0
2        1
3        0
4        0
       ..
5567     1
5568     0
5569     0
5570     0
5571     0
Name: label, Length: 5572, dtype: int64
```

3) Split the dataset into train and test data set.

```
#split the dataset into train and test data set
#use 30% for testing and 70% for training.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state = 100)
```

4) Classify and predict by using logistic regression model and naive bayes model.

```
# classify and predict by using logistic regression model and naive bayes model
lr = LogisticRegression(solver='liblinear', penalty='l1')
mnb = MultinomialNB()  # multi nomial naives bayes: feature matrix represents the frequencies
```

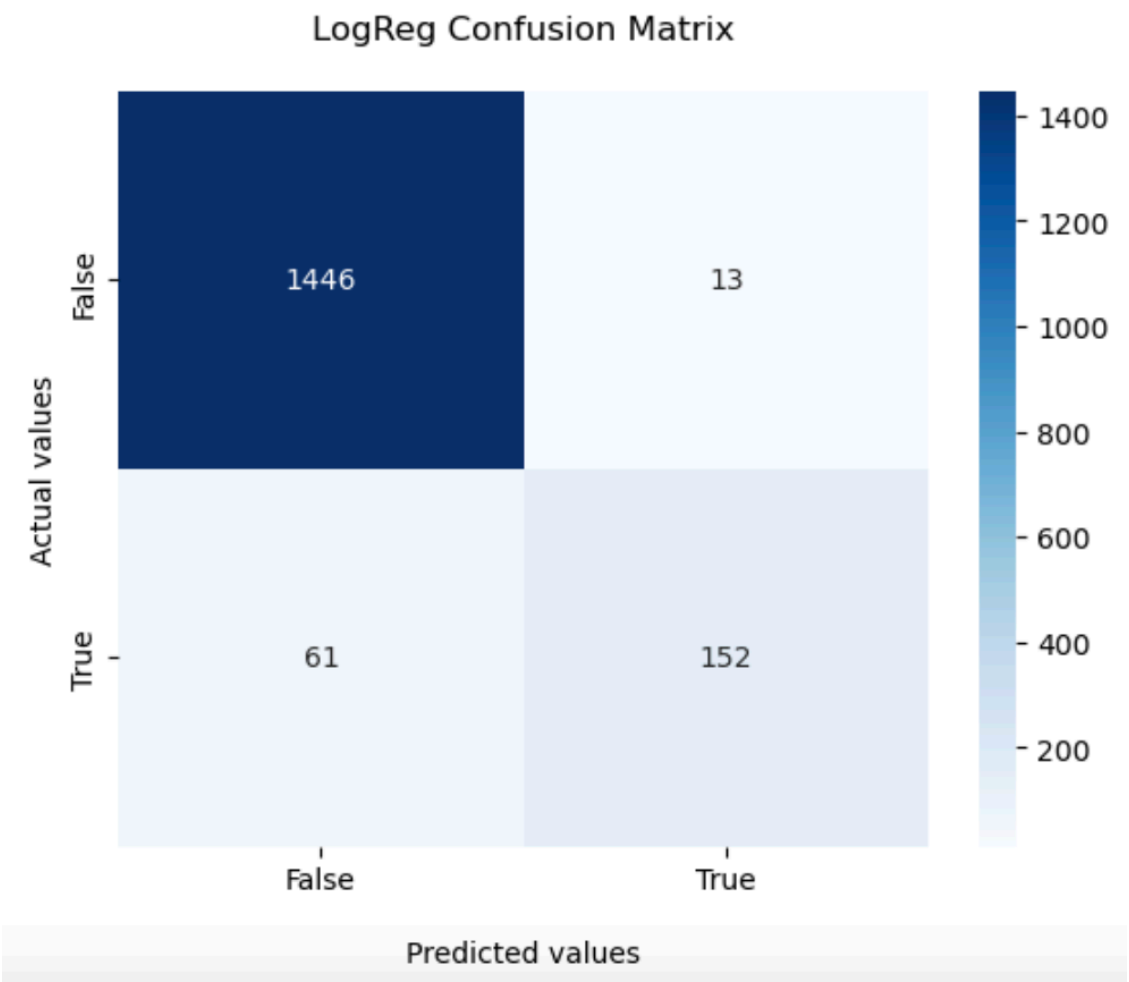
1) Using Logistic Regression

```
# firstly, using lr
lr.fit(x_train, y_train)
y_pred_lr = lr.predict(x_test)
```

```
test_acc_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
print(f'Test ACC LR: {round(test_acc_lr, 3)}')
print(f'Precision LR: {round(precision_lr, 3)}')
print(f'F1_Score LR: {round(f1_lr, 3)}')
print(f'Recall LR: {round(recall_lr, 3)}')
```

```
Test ACC LR: 0.956
Precision LR: 0.921
F1_Score LR: 0.804
Recall LR: 0.714
```

```
cf_matrix = confusion_matrix(y_test, y_pred_lr)
ax = sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='.5g')
ax.set_title('LogReg Confusion Matrix\n')
ax.set_xlabel('\nPredicted values')
ax.set_ylabel('Actual values')
ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True'])
plt.show()
```



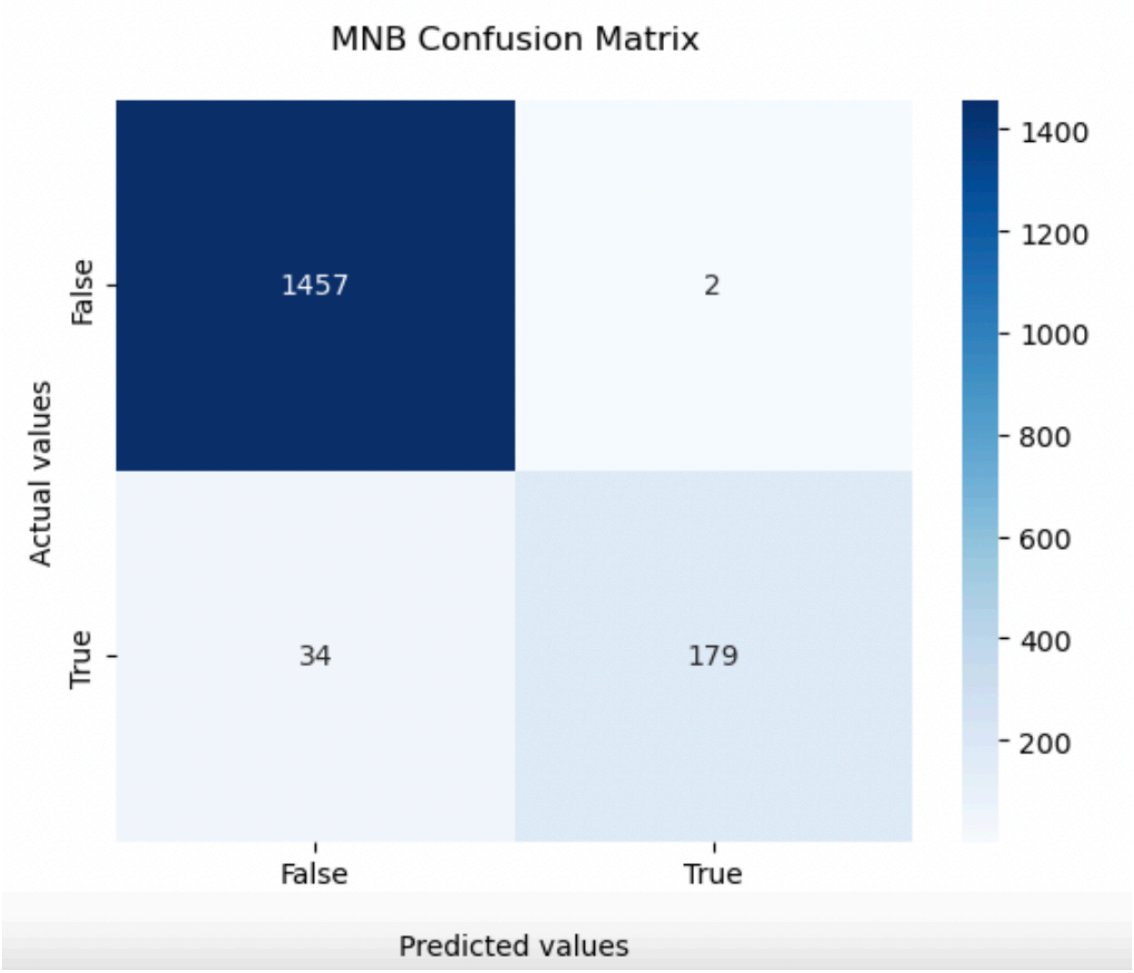
ii) Using Multinomial Naives Bayes

```
# secondly, using mnb
mnb.fit(x_train, y_train)
y_pred_mnb = mnb.predict(x_test)
test_acc_mnb = accuracy_score(y_test, y_pred_mnb)
precision_mnb = precision_score(y_test, y_pred_mnb)
f1_mnb = f1_score(y_test, y_pred_mnb)
recall_mnb = recall_score(y_test, y_pred_mnb)
print(f'Test ACC MNB: {round(test_acc_mnb, 3)}')
print(f'Precision MNB: {round(precision_mnb, 3)}')
print(f'F1_Score MNB: {round(f1_mnb, 3)}')
print(f'Recall MNB: {round(recall_mnb, 3)}')
```

```
Test ACC MNB: 0.978
Precision MNB: 0.989
F1_Score MNB: 0.909
Recall MNB: 0.84
```

```
cf_matrix = confusion_matrix(y_test, y_pred_mnb)
ax = sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='.5g')
ax.set_title('MNB Confusion Matrix\n')
ax.set_xlabel('\nPredicted values')
ax.set_ylabel('Actual values')
ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True'])
```

```
[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]
```



Task2-4: Conclusion

I have learned lots of methods about data preprocessing and mining, and some basic natural language processing techniques. I have got a lot of errors during the lab, but after searching informations on the Internet, I have found some good ways to solve the problems. As I mentioned, we can apply so many other techniques with data processing and NLP, such as data visualisation and machine learning.

By reading other people's study notes about NLTK toolkit on the Internet, I found that NLTK does not have a good parser for parsing the grammatical

structures. And found that the NLTK library can only perform word segmentation processing for English at present, and cannot perform this operation for Chinese.