Business Problem 1:

Prepare a classification model using Naive Bayes for salary data

Data Description:

age -- age of a person

workclass -- A work class is a grouping of work

education -- Education of an individuals

maritalstatus -- Marital status of an individuals

occupation -- occupation of an individuals

relationship --

race -- Race of an Individual

sex -- Gender of an Individual

capitalgain -- profit received from the sale of an investment

capitalloss -- A decrease in the value of a capital asset

hoursperweek -- number of hours work per week

native -- Native of an individual

Salary -- salary of an individual

Solution: In the given business problem, we need to classify the data given based on that the salary which is lesser than or greater than fifty thousand is classified.

Importing essential libraries:

Codes:

```
import pandas as pd

import numpy as np

from sklearn.naive_bayes import GaussianNB

from sklearn.naive_bayes import MultinomialNB

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix
```

Data Loading:

Codes:

```
Salary_test=pd.read_csv("E:\Data\Assignments\imade\Naivebayes\SalaryData_Test.cs")

Salary_train=pd.read_csv("E:\Data\Assignments\imade\Naivebayes\SalaryData_Train.csv")


colnames_test = list(Salary_test.columns)

colnames_train = list(Salary_train.columns)
```

Getting Dummies:

As the data consists categorical data, we need to convert them into numerical, so that we create dummies for the categorical data.

Codes:

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder

le = LabelEncoder()


Salary_test['workclass'] = le.fit_transform(Salary_test['workclass'])
Salary_test['education'] = le.fit_transform(Salary_test['education'])
Salary_test['maritalstatus'] = le.fit_transform(Salary_test['maritalstatus'])
Salary_test['occupation'] = le.fit_transform(Salary_test['occupation'])
Salary_test['relationship'] = le.fit_transform(Salary_test['relationship'])
Salary_test['race'] = le.fit_transform(Salary_test['race'])
Salary_test['sex'] = le.fit_transform(Salary_test['sex'])
Salary_test['Salary'] = le.fit_transform(Salary_test['Salary'])
Salary_test['native'] = le.fit_transform(Salary_test['native'])
```

```
Salary_train['workclass'] = le.fit_transform(Salary_train['workclass'])
Salary_train['education'] = le.fit_transform(Salary_train['education'])
Salary_train['maritalstatus'] = le.fit_transform(Salary_train['maritalstatus'])
Salary_train['occupation'] = le.fit_transform(Salary_train['occupation'])
Salary_train['relationship'] = le.fit_transform(Salary_train['relationship'])
Salary_train['race'] = le.fit_transform(Salary_train['race'])
Salary_train['sex'] = le.fit_transform(Salary_train['sex'])
Salary_train['Salary'] = le.fit_transform(Salary_train['Salary'])
Salary_train['native'] = le.fit_transform(Salary_train['native'])
```

Defining input and outputs:

In this problem we need to classify the salaries which are lesser and greater than 50 thousand, so the salary variables becomes the output or target variable and the rest becomes the input variables.

Codes:

```
test_predictors = colnames_test[:13]
test_target = colnames_test[13]

train_predictors = colnames_train[:13]
train_target = colnames_train[13]
```

Splitting:

To get the confusion matrix, we need to split the data into Train and Test.

Splitting the test data set:

Codes:

```
DXtrain,DXtest,Dytrain,Dytest=train_test_split(Salary_test[test_predictors],Salary_test[test_target],test_size=0.3, random_state=0)

Dgnb = GaussianNB()

Dmnb = MultinomialNB()
```

Test data Prediction:

Codes:

```
Dpred_gnb = Dgnb.fit(DXtrain,Dytrain).predict(DXtest)

Dpred_mnb = Dmnb.fit(DXtrain,Dytrain).predict(DXtest)
```

Getting confusion matrix:

To check the accuracies we need to form the confusion matrix.

Test data Confusion Matrix:

Codes:

Gaussian Naive Bayes:

```
confusion_matrix(Dytest,Dpred_gnb)

print ("Accuracy",(3223+370)/(3223+370+162+763))
```

Accuracy = 79.53

Multinomial Naive Bayes:

```
confusion_matrix(Dytest,Dpred_mnb)

print ("Accuracy",(3258+232)/(3258+232+2584+127+901))
```

Accuracy = 49.14

Splitting the test data set:

Codes:

```
DXtrain,DXtest,Dytrain,Dytest=train_test_split(Salary_train[train_predictors],Salary_train[train_target],test_size=0.3, random_state=0)

Dgnb = GaussianNB()

Dmnb = MultinomialNB()
```

Train data Prediction:

Codes:

```
Dpred_gnb = Dgnb.fit(DXtrain,Dytrain).predict(DXtest)

Dpred_mnb = Dmnb.fit(DXtrain,Dytrain).predict(DXtest)
```

Test data Confusion Matrix:

Codes:

Gaussian Naive Bayes:

```
confusion_matrix(Dytest,Dpred_gnb)

print ("Accuracy",(6463+720)/(6463+720+1531+335))
```

Accuracy = 79.37

Multinomial Naive Bayes:

```
confusion_matrix(Dytest,Dpred_mnb)

print ("Accuracy",(6526+471)/(6526+471+272+1780))
```

Accuracy = 77.32

So we can conclude that the Accuracy of the Guassian Naive Bayes of both Test data and Train data are more accurate than Multinomial Naive Bayes.

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

Salary_test = pd.read_csv("E:\\Data\\Assignments\\i made\\Naivebayes\\SalaryData_Test.csv")
Salary_train = pd.read_csv("E:\\Data\\Assignments\\i made\\Naivebayes\\SalaryData_Train.csv")

colnames_test = list(Salary_test.columns)
colnames_train = list(Salary_train.columns)

#.......Get Dummies.....
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
le = LabelEncoder()

Salary_test['workclass'] = le.fit_transform(Salary_test['workclass'])
Salary_test['education'] = le.fit_transform(Salary_test['education'])
Salary_test['maritalstatus'] = le.fit_transform(Salary_test['maritalstatus'])
Salary_test['occupation'] = le.fit_transform(Salary_test['occupation'])
Salary_test['relationship'] = le.fit_transform(Salary_test['relationship'])
Salary_test['race'] = le.fit_transform(Salary_test['race'])
Salary_test['sex'] = le.fit_transform(Salary_test['sex'])
Salary_test['Salary'] = le.fit_transform(Salary_test['Salary'])
Salary_test['native'] = le.fit_transform(Salary_test['native'])


Salary_train['workclass'] = le.fit_transform(Salary_train['workclass'])
Salary_train['education'] = le.fit_transform(Salary_train['education'])
Salary_train['maritalstatus'] = le.fit_transform(Salary_train['maritalstatus'])
Salary_train['occupation'] = le.fit_transform(Salary_train['occupation'])
Salary_train['relationship'] = le.fit_transform(Salary_train['relationship'])
Salary_train['race'] = le.fit_transform(Salary_train['race'])
Salary_train['sex'] = le.fit_transform(Salary_train['sex'])
Salary_train['Salary'] = le.fit_transform(Salary_train['Salary'])
Salary_train['native'] = le.fit_transform(Salary_train['native'])
```

```
# Definig input and output
test_predictors = colnames_test[:13]
test_target = colnames_test[13]

train_predictors = colnames_train[:13]
train_target = colnames_train[13]

# Splitting
DXtrain,DXtest,Dytrain,Dytest = train_test_split(Salary_test[test_predictors],Salary_test[test_target],test_size=0.3, random_state=0)

Dgnb = GaussianNB()
Dmnb = MultinomialNB()

# Prediction
Dpred_gnb = Dgnb.fit(DXtrain,Dytrain).predict(DXtest)
Dpred_mnb = Dmnb.fit(DXtrain,Dytrain).predict(DXtest)

# Confusion Matrix
# GaussianNB
confusion_matrix(Dytest,Dpred_gnb)
print ("Accuracy",(3223+370)/(3223+370+162+763)) # 79.53

# MultinomialNB
confusion_matrix(Dytest,Dpred_mnb)
print ("Accuracy",(3258+232)/(3258+232+2584+127+901)) # 49.14


DXtrain,DXtest,Dytrain,Dytest = train_test_split(Salary_train[train_predictors],Salary_train[train_target],test_size=0.3, random_state=0)

Dgnb = GaussianNB()
Dmnb = MultinomialNB()

# Prediction

Dpred_gnb = Dgnb.fit(DXtrain,Dytrain).predict(DXtest)
Dpred_mnb = Dmnb.fit(DXtrain,Dytrain).predict(DXtest)
```

```
# Confusion Matrix
# GaussianNB
confusion_matrix(Dytest,Dpred_gnb)
print ("Accuracy",(6463+720)/(6463+720+1531+335)) # 79.37

# MultinomialNB
confusion_matrix(Dytest,Dpred_mnb)
print ("Accuracy",(6526+471)/(6526+471+272+1780)) # 77.32

```

Business Problem 2:

Build a naive Bayes model on the data set for classifying the ham and spam

Solution: In the given business problem, we need to classify before the classification we need to clean the data, since the given data is raw or in unstructured format and consists of unwanted data.

Importing essential libraries:

Codes:

```
import pandas as pd

import numpy as np

from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
```

Loading the data:

Codes:

```
email_data=pd.read_csv("E:\\Data\\Assignments\\imade\\Naivebayes\\ham_spam.csv",en
gine = "python")
```

cleaning data:

Codes:

```
import re

stop_words = []

with open("E:\\Data\\Assignments\\i made\\TextMining\\stop.txt") as f:

        stop_words = f.read()
```

Splitting the entire string by giving separator as "\n" to get list of all stop words:

Codes:

```
stop_words = stop_words.split("\n")
```

Transform a count matrix to a normalized tf or tf-idf representation:

Creating a matrix of token counts for the entire text document:

Codes:

```
def split_into_words(i):
    return [word for word in i.split(" ")]
```

splitting data into train and test data sets:

Codes:

```
from sklearn.model_selection import train_test_split
email_train,email_test = train_test_split(email_data,test_size=0.3)
```

Preparing email texts into word count matrix format:

Codes:

```
emails_bow = CountVectorizer(analyzer=split_into_words).fit(email_data.text)
```

For all messages:

Codes:

```
all_emails_matrix = emails_bow.transform(email_data.text)
all_emails_matrix.shape
```

(5559, 6661)

For training messages:

Codes:

```
train_emails_matrix = emails_bow.transform(email_train.text)

train_emails_matrix.shape
```

(3891, 6661)

For testing messages:

Codes:

```
test_emails_matrix = emails_bow.transform(email_test.text)

test_emails_matrix.shape
```

(1668, 6661)

Preparing a naive bayes model on training data set without TFIDF matrices:

Codes:

```
from sklearn.naive_bayes import MultinomialNB as MB

from sklearn.naive_bayes import GaussianNB as GB
```

Multinomial Naive Bayes

Codes:

```
classifier_mb = MB()

classifier_mb.fit(train_emails_matrix,email_train.type)

train_pred_m = classifier_mb.predict(train_emails_matrix)

accuracy_train_m = np.mean(train_pred_m==email_train.type)
```

Accuracy = 98%

test_pred_m = classifier_mb.predict(test_emails_matrix)

accuracy_test_m = np.mean(test_pred_m==email_test.type)

Accuracy = 96%

Gaussian Naive Bayes

classifier_gb = GB()

classifier_gb.fit(train_emails_matrix.toarray(),email_train.type.values) # we need to convert tfidf into array format which is compatible for gaussian naive bayes

train_pred_g = classifier_gb.predict(train_emails_matrix.toarray())

accuracy_train_g = np.mean(train_pred_g==email_train.type)

Accuracy = 90%

test_pred_g = classifier_gb.predict(test_emails_matrix.toarray())

accuracy_test_g = np.mean(test_pred_g==email_test.type)

Accuracy = 83%

Learning Term weighting and normalizing on entire emails:

tfidf_transformer = TfidfTransformer().fit(all_emails_matrix)

Preparing TFIDF for train emails:

train_tfidf = tfidf_transformer.transform(train_emails_matrix)

train_tfidf.shape

(3891, 6661)

Preparing TFIDF for test emails:

test_tfidf = tfidf_transformer.transform(test_emails_matrix)

test_tfidf.shape

(1668, 6661)

Preparing a naive bayes model on training data set:

from sklearn.naive_bayes import MultinomialNB as MB

from sklearn.naive_bayes import GaussianNB as GB

Multinomial Naive Bayes:

classifier_mb = MB()

classifier_mb.fit(train_tfidf,email_train.type)

train_pred_m = classifier_mb.predict(train_tfidf)

accuracy_train_m = np.mean(train_pred_m==email_train.type)

Accuracy = 96%

test_pred_m = classifier_mb.predict(test_tfidf)

accuracy_test_m = np.mean(test_pred_m==email_test.type)

Accuracy = 95%

Gaussian Naive Bayes:

classifier_gb = GB()

classifier_gb.fit(train_tfidf.toarray(),email_train.type.values)

[we need to convert tfidf into array format which is compatible for gaussian naive bayes]

train_pred_g = classifier_gb.predict(train_tfidf.toarray())

accuracy_train_g = np.mean(train_pred_g==email_train.type)

Accuracy = 91%

test_pred_g = classifier_gb.predict(test_tfidf.toarray())

accuracy_test_g = np.mean(test_pred_g==email_test.type)

Accuracy = 85%

Hence we can conclude that the Accuracy of the Multinomial Naïve Bayes model for both training and testing data sets gives a better results or accuracies than Guassian Naive Bayes model.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer

# Loading the data set

email_data = pd.read_csv("E:\\Data\\Assignments\\i made\\Naivebayes\\ham_spam.csv",engine = "python")


# cleaning data
import re
stop_words = []
with open("E:\\Data\\Assignments\\i made\\TextMining\\stop.txt") as f:
    stop_words = f.read()


# splitting the entire string by giving separator as "\n" to get list of
# all stop words
stop_words = stop_words.split("\n")

```

```
email_data.text = email_data.text.apply(cleaning_text)

# removing empty rows
email_data.shape
email_data = email_data.loc[email_data.text != " ",:]


# CountVectorizer
# Convert a collection of text documents to a matrix of token counts

# TfidfTransformer
# Transform a count matrix to a normalized tf or tf-idf representation

# creating a matrix of token counts for the entire text document

def split_into_words(i):
    return [word for word in i.split(" ")]


# splitting data into train and test data sets
from sklearn.model_selection import train_test_split

email_train,email_test = train_test_split(email_data,test_size=0.3)


```

```
# Preparing email texts into word count matrix format
emails_bow = CountVectorizer(analyzer=split_into_words).fit(email_data.text)


# For all messages
all_emails_matrix = emails_bow.transform(email_data.text)
all_emails_matrix.shape # (5559,6661)
# For training messages
train_emails_matrix = emails_bow.transform(email_train.text)
train_emails_matrix.shape # (3891,6661)



# For testing messages
test_emails_matrix = emails_bow.transform(email_test.text)
test_emails_matrix.shape # (1668,6661)

####### Without TFIDF matrices ########################
# Preparing a naive bayes model on training data set

from sklearn.naive_bayes import MultinomialNB as MB
from sklearn.naive_bayes import GaussianNB as GB

# Multinomial Naive Bayes
classifier_mb = MB()
classifier_mb.fit(train_emails_matrix,email_train.type)
train_pred_m = classifier_mb.predict(train_emails_matrix)
accuracy_train_m = np.mean(train_pred_m==email_train.type) # 98%

test_pred_m = classifier_mb.predict(test_emails_matrix)
accuracy_test_m = np.mean(test_pred_m==email_test.type) # 96%
```

```
# Gaussian Naive Bayes
classifier_gb = GB()
classifier_gb.fit(train_emails_matrix.toarray(),email_train.type.values) # we need to convert tfidf into array format
train_pred_g = classifier_gb.predict(train_emails_matrix.toarray())
accuracy_train_g = np.mean(train_pred_g==email_train.type) # 90%

test_pred_g = classifier_gb.predict(test_emails_matrix.toarray())
accuracy_test_g = np.mean(test_pred_g==email_test.type) # 83%


#########################################################3

# Learning Term weighting and normalizing on entire emails
tfidf_transformer = TfidfTransformer().fit(all_emails_matrix)

# Preparing TFIDF for train emails
train_tfidf = tfidf_transformer.transform(train_emails_matrix)

train_tfidf.shape # (3891, 6661)

# Preparing TFIDF for test emails
test_tfidf = tfidf_transformer.transform(test_emails_matrix)

test_tfidf.shape #  (1668, 6661)

# Preparing a naive bayes model on training data set

from sklearn.naive_bayes import MultinomialNB as MB
from sklearn.naive_bayes import GaussianNB as GB

```

```
# Multinomial Naive Bayes
classifier_mb = MB()
classifier_mb.fit(train_tfidf,email_train.type)
train_pred_m = classifier_mb.predict(train_tfidf)
accuracy_train_m = np.mean(train_pred_m==email_train.type) # 96%

test_pred_m = classifier_mb.predict(test_tfidf)
accuracy_test_m = np.mean(test_pred_m==email_test.type) # 95%

# Gaussian Naive Bayes
classifier_gb = GB()
classifier_gb.fit(train_tfidf.toarray(),email_train.type.values) # we need to convert tfidf into array format i
train_pred_g = classifier_gb.predict(train_tfidf.toarray())
accuracy_train_g = np.mean(train_pred_g==email_train.type) # 91%
test_pred_g = classifier_gb.predict(test_tfidf.toarray())
accuracy_test_g = np.mean(test_pred_g==email_test.type) # 85%

# inplace of tfidf we can also use train_emails_matrix and test_emails_matrix instead of term inverse document

```