In [1]:
```python
## import packaes
import pandas as pd
from scipy import stats
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

Import Data

In [2]:
```python
## import csv file
og_data = pd.read_csv('F://UM//Data Mining//Assi_2//caravan.csv')
```

In [3]:
```python
og_data.shape
```

Out[3]:
```
(5822, 86)
```

In [4]:
```python
# check missing data
og_data[og_data.isnull().any(axis=1)]
```

Out[4]:

	Customer Subtype	Number of houses	Avg size household	Avg Age	Customer main type	Roman catholic	Protestant	Other religion	No religion	Married	...	Number of private accident insurance policies

0 rows × 86 columns

In [5]:
```python
og_data.head()
```

Out[5]:

	Customer Subtype	Number of houses	Avg size household	Avg Age	Customer main type	Roman catholic	Protestant	Other religion	No religion	Married	...	Number of private accident insurance policies
0	33	1	3	2	8	0	5	1	3	7	...	0
1	37	1	2	2	8	1	4	1	4	6	...	0
2	37	1	2	2	8	0	4	2	4	3	...	0
3	9	1	3	3	3	2	3	2	4	5	...	0
4	40	1	4	2	10	1	4	1	4	7	...	0

5 rows × 86 columns

Explore the data

use descriptive statistics to explore data

In [6]:
```python
## Check whether there is a class unbalanced
```

```
y_column = list(og_data[og_data.columns[-1]]) # the last column of the data

# percentage of 'CARAVAN POLICY'=1
sum(y_column) / len(y_column)

## yes. class balance is needed
```

Out[6]:

```
0.05977327378907592
```

In [7]:

```
## number of independent variables

nb_feature = len(list(og_data.columns.values)) - 1
nb_feature
```

Out[7]:

```
85
```

In [8]:

```
## Before resampling, split the test data at first to aviod overfitting
from sklearn.model_selection import train_test_split

og_y = og_data['CARAVAN POLICY']
og_x = og_data.drop(columns='CARAVAN POLICY')
x_remain, x_test, y_remain, y_test = train_test_split(
                                og_x, og_y, random_state=0, train_size = .9)
```

In [9]:

```
remian_og_data = pd.concat([x_remain, y_remain], axis=1)
```

Resample the data to make it balanced

In [10]:

```
## resample the data with true label to make the dataset balanced

from sklearn.utils import resample

# the data with label 1
y_pos_data = remian_og_data[remian_og_data['CARAVAN POLICY']==1]

# number of pos. data in remain data
num_pos_remain = sum(list(remian_og_data['CARAVAN POLICY']))
num_total_remian = len(list(remian_og_data['CARAVAN POLICY']))

# resample the y_pos_data
y_pos_data_resampled = resample(y_pos_data,
                       n_samples=num_total_remian-num_pos_remain,
                       random_state=1000)

# number of the resampled data, number of data y label = 0
len(y_pos_data_resampled), num_total_remian-num_pos_remain
```

Out[10]:

```
(4935, 4935)
```

In [263…

```
balance_data.shape
```

Out[263]:

```
(9870, 86)
```

In [11]:

```
## create the new balanced data set

y_neg_data = remian_og_data[remian_og_data['CARAVAN POLICY']==0]

balance_data = pd.concat([y_pos_data_resampled, y_neg_data])

balance_data.head()
```

Out[11]:

	Customer Subtype	Number of houses	Avg size household	Avg Age	Customer main type	Roman catholic	Protestant	Other religion	No religion	Married	...	Nun pri acci insura pol
1387	8	1	3	2	2	0	7	2	0	7	...	
4296	33	1	3	3	8	0	7	0	2	5	...	
194	36	1	3	3	8	0	7	0	2	7	...	
2286	39	1	3	3	9	2	7	1	0	7	...	
5307	12	1	3	3	3	0	6	0	3	6	...	

5 rows × 86 columns

Split the data

In [12]:

```
# split the balance_data into x and y

def split_data(input_data):
    y = input_data['CARAVAN POLICY']
    x = input_data.drop(columns='CARAVAN POLICY')

    # name_x: the list of names of x
    name_x = list(x.columns.values)

    return x, y, name_x

x, y, name_x = split_data(balance_data)
```

In [13]:

```
# split the balance_data into train(0.7), validation(0.3)

def split_train_val(input_x, input_y):
    x_train, x_val, y_train, y_val = train_test_split(
                                input_x, input_y, random_state=0, train_size = .7)
    return x_train, x_val, y_train, y_val

x_train, x_val, y_train, y_val = split_train_val(x, y)
```

Model Training and Building

Before Model Building

In [15]:

```
## set the index that interested in
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score


# recrod the choosen model accuracy
# num right prediction / num total data

# how many percents of data that pred. pos. would be right
# precision = ratio tp / (tp + fp)


```

```python
# how many persents of true pos data that would predict right
# recall = tp / (tp + fn)
# num right pos predction / num true pos

def measure(y_pred, true_y):
    acc_test = accuracy_score(true_y, y_pred)
    precision_test = precision_score(true_y, y_pred, average='binary')
    recall_test = recall_score(true_y, y_pred, average='binary')

    return acc_test, precision_test, recall_test
```

In [21]:

```python
## define functions to make the codes efficient

def try_parameter(method, i, x_train, y_train, x_val, y_val, x_test, y_test):
    # For decision tree: i-the choosen depth to choose the depth
    # for KNN: i-the choosen neigbers
    ## method: could be 'dtc' / 'knn'
    if method == 'dtc':
        model = DecisionTreeClassifier(max_depth=i, random_state=1)
    elif method == 'knn':
        model = KNeighborsClassifier(n_neighbors=i)
    else:
        return print('Wrong input of the method')
    model_now = model.fit(x_train, y_train)
    pre_val_now = model_now.predict(x_val)
    pre_test_now = model_now.predict(x_test)

    acc_val = accuracy_score(y_val, pre_val_now)
    precision_val = precision_score(y_val, pre_val_now, average='binary')
    recall_val = recall_score(y_val, pre_val_now, average='binary')

    acc_test = accuracy_score(y_test, pre_test_now)
    precision_test = precision_score(y_test, pre_test_now, average='binary')
    recall_test = recall_score(y_test, pre_test_now, average='binary')

    return acc_val, precision_val, recall_val, acc_test, precision_test, recall_test

def dtc_depth(i, x_train, y_train, x_val, y_val, x_test, y_test):
    # For decision tree: i-the choosen depth
    # to choose the depth
    dtc = DecisionTreeClassifier(max_depth=i, random_state=1)
    dtc_now = dtc.fit(x_train, y_train)
    pre_val_now = dtc_now.predict(x_val)
    pre_test_now = dtc_now.predict(x_test)

    acc_val = accuracy_score(y_val, pre_val_now)
    precision_val = precision_score(y_val, pre_val_now, average='binary')
    recall_val = recall_score(y_val, pre_val_now, average='binary')

    acc_test = accuracy_score(y_test, pre_test_now)
    precision_test = precision_score(y_test, pre_test_now, average='binary')
    recall_test = recall_score(y_test, pre_test_now, average='binary')

    return acc_val, precision_val, recall_val, acc_test, precision_test, recall_test

def knn_k(k, x_train, y_train, x_val, y_val, x_test, y_test):
    # for KNN: k-the choosen neigbers
    knn = KNeighborsClassifier(n_neighbors=k)
    knn_now = knn.fit(x_train, y_train)
    pre_val = knn_now.predict(x_val)
    pre_test = knn_now.predict(x_test)

    acc_val = knn.score(x_val, y_val)
    precision_val = precision_score(y_val, pre_val, average='binary')
    recall_val = recall_score(y_val, pre_val, average='binary')
    acc_test = knn.score(x_test, y_test)
    precision_test = precision_score(y_test, pre_test, average='binary')
    recall_test = recall_score(y_test, pre_test, average='binary')

    return acc_val, precision_val, recall_val, acc_test, precision_test, recall_test
```

Decision Tree

In [17]:

```
# import the packages related to Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import pydot
from IPython.display import Image
```

Find the optimal depth of the decision tree

Decide the depth of decision tree according to the prediction accuracy on the val. data

In [18]:

```
## use all variables to predict y

DTC_all = DecisionTreeClassifier()
dt_whole = DTC_all.fit(x_train, y_train)

## evaluate the Decision tree

# use decision tree to predict the y on val. data
pre_val_tree = dt_whole.predict(x_val)

# calculate the prediction accuracy
acc_val_tree = accuracy_score(y_val, pre_val_tree)

print('The accuracy of Decision tree on val. data is :', acc_val_tree)
### output: 0.96
```

```
The accuracy of Decision tree on val. data is : 0.9608240459304289
```

In [19]:

```
# obtain the max. depth of the decision tree on the given data
max_depth = DTC_all.tree_.max_depth
max_depth
### output: 40
```

Out[19]:

```
40
```

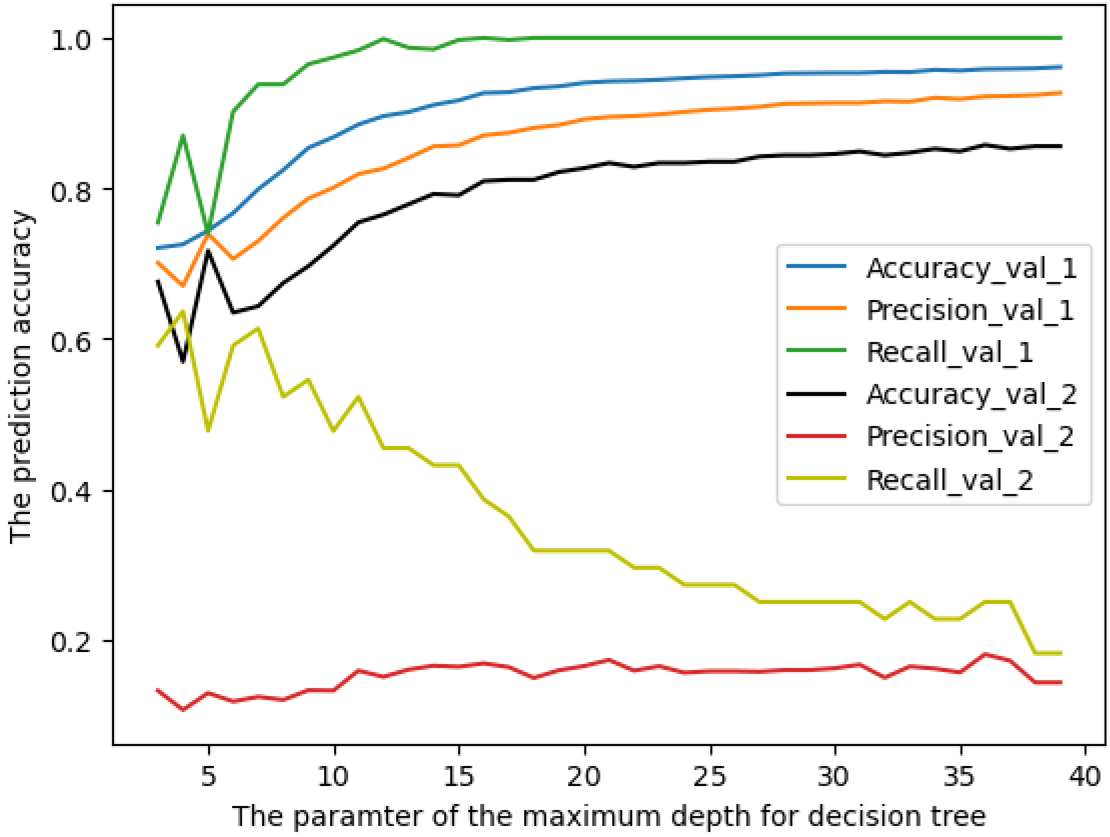
In [37]:

```
## Try different depths on the decison tree
## then calculate its prediction accuracy on the val. data

## Try different depths on the decison tree
## then calculate its prediction accuracy on the val. data

def record_try_parameter(max_para, para, x_train, y_train, x_val, y_val, x_test, y_test)
    # max_para: the maximum od the parameter
    # para: could be 'k' for knn / 'depth' for dtc
    # record the depth of the decision tree & acc & precision & recall
    dict_dtc_index_record = {para:[], 'acc_val':[],
                             'precision_val':[], 'recall_val':[],
                                 'acc_test':[], 'precision_test':[], 'recall_test':[]}
    if para == 'k':
        method = 'knn'
    elif para == 'depth':
        method = 'dtc'
    i = 3
    while i+1 <= max_para:
        acc_val, precision_val, recall_val, acc_test, precision_test, recall_test\
            = try_parameter(method, i, x_train, y_train, x_val, y_val, x_test, y_test)

        dict_dtc_index_record[para].append(i)
        dict_dtc_index_record['acc_val'].append(acc_val)
        dict_dtc_index_record['precision_val'].append(precision_val)
        dict_dtc_index_record['recall_val'].append(recall_val)
        dict_dtc_index_record['acc_test'].append(acc_test)
        dict_dtc_index_record['precision_test'].append(precision_test)
        dict_dtc_index_record['recall_test'].append(recall_test)
        i+=1

    ## Transform the data into df structure
    df_depth_index_dtc = pd.DataFrame.from_dict(dict_dtc_index_record)

    # print the head data accroding to 'recall' column
    print(df_depth_index_dtc.sort_values(
        by=['recall_test', 'precision_test'], ascending=False).head())

    # return the acc & precision & recall on different depths
    return df_depth_index_dtc, dict_dtc_index_record

df_depth_index_dtc, dict_dtc_index_record  = record_try_parameter(
                    max_depth, 'depth', x_train, y_train, x_val, y_val, x_test, y_test)
```

```
   depth   acc_val  precision_val  recall_val  acc_test  precision_test  \
1      4  0.725431       0.670190    0.870281  0.569468        0.106464
4      7  0.798717       0.729845    0.938229  0.643225        0.123853
0      3  0.720702       0.700893    0.754290  0.675815        0.131980
3      6  0.766971       0.706072    0.901853  0.634648        0.117647
6      9  0.853766       0.786353    0.964997  0.696398        0.132597

   recall_test
1     0.636364
4     0.613636
0     0.590909
3     0.590909
6     0.545455
```

In [38]:

```
## Visualize the relationship between three indexes and depth
## on the validation data

def plot_diff_para(dict_dtc_index_record, xlab, ylab, special_para):
    plt.plot(dict_dtc_index_record[special_para],
             dict_dtc_index_record['acc_val'], label='Accuracy_val_1')
    plt.plot(dict_dtc_index_record[special_para],
             dict_dtc_index_record['precision_val'], label='Precision_val_1')
    plt.plot(dict_dtc_index_record[special_para],
             dict_dtc_index_record['recall_val'], label='Recall_val_1')
    plt.plot(dict_dtc_index_record[special_para],
             dict_dtc_index_record['acc_test'], label='Accuracy_val_2',color='k')
    plt.plot(dict_dtc_index_record[special_para],
             dict_dtc_index_record['precision_test'], label='Precision_val_2')
    plt.plot(dict_dtc_index_record[special_para],
             dict_dtc_index_record['recall_test'], label='Recall_val_2', color='y')
    plt.xlabel(xlab)
    plt.ylabel(ylab)
    plt.legend()

xlab = 'The paramter of the maximum depth for decision tree'
ylab = "The prediction accuracy"
plot_diff_para(dict_dtc_index_record, xlab, ylab, 'depth')
```



Choose the 'best' decision tree

In [39]:

```
## As we can see, when the parameter 'max_depth'=4, the indexes are good enough

# see the specific performance

optimal_depth_allvar = 4
df_depth_index_dtc.loc[df_depth_index_dtc['depth'] == optimal_depth_allvar]
```

Out[39]:

	depth	acc_val	precision_val	recall_val	acc_test	precision_test	recall_test
1	4	0.725431	0.67019	0.870281	0.569468	0.106464	0.636364

In [114…

```
choose_dtc_allvar = DecisionTreeClassifier(
        max_depth=optimal_depth_allvar, random_state=1)

# use the all balanced data to train
best_dtc_allvar = choose_dtc_allvar.fit(x_train, y_train)

pred_best_dtc_allvar = best_dtc_allvar.predict(x_test)

metric_dtc = measure(pred_best_dtc_allvar, y_test)
metric_dtc
```

Out[114]:

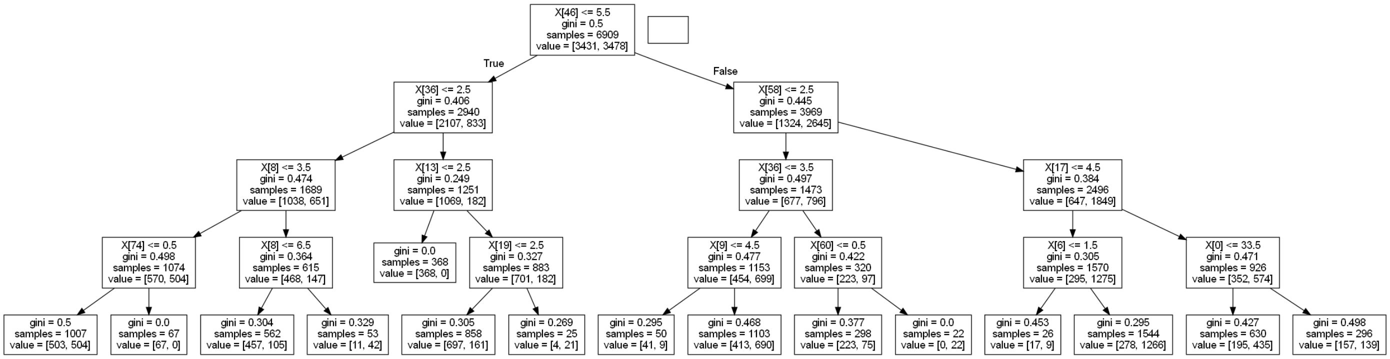
```
(0.5694682675814752, 0.10646387832699619, 0.6363636363636364)
```

In [86]:

```
## Visualize the 'best' decision tree Model
## consider both the complexity and accuracy

from six import StringIO
import pydotplus


# train the data on the choosen model
dtc_fix_dep = DecisionTreeClassifier(max_depth=4, random_state=1)
```

```
dtc_choose = dtc_fix_dep.fit(x_train, y_train)
pre_val_dtc_choose = dtc_choose.predict(x_val)

dot_data = StringIO()
tree.export_graphviz(dtc_choose, out_file=dot_data)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

# save figure as pdf file
graph.write_pdf("best_dtc.pdf")
# show the figure
Image(graph.create_png())
```

Out[86]:



In [87]:

```
def show_dtc(dtc_choose, name_x, name_file):
    dot_data = StringIO()
    tree.export_graphviz(dtc_choose, out_file=dot_data,
                        feature_names=name_x, class_names=['0','1'],impurity=False)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

    graph.write_pdf(name_file)
    Image(graph.create_png())

name_file = "best_dtc_name.pdf"
show_dtc(dtc_choose, name_x, name_file)
```

Nearest Neighbor

In [41]:

```
## import the related packages
from sklearn.neighbors import KNeighborsClassifier

## Ignore Warnings
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [44]:

```
# record the k of knn & acc & precision & recall

# try mex. neighbor=50
dict_knn_index_record =  record_try_parameter(50, 'k', x_train,
                            y_train, x_val, y_val, x_test, y_test)
```

```
     k   acc_val  precision_val  recall_val  acc_test  precision_test  \
40  43  0.688281       0.646542    0.808511  0.567753        0.109023
29  32  0.730496       0.691014    0.818119  0.610635        0.117155
30  33  0.719352       0.675056    0.828415  0.598628        0.113821
28  31  0.727457       0.683202    0.831846  0.591767        0.112000
41  44  0.692334       0.654412    0.794097  0.578045        0.108527

    recall_test
40     0.659091
```

```
29      0.636364
30      0.636364
28      0.636364
41      0.636364
```

In [51]:

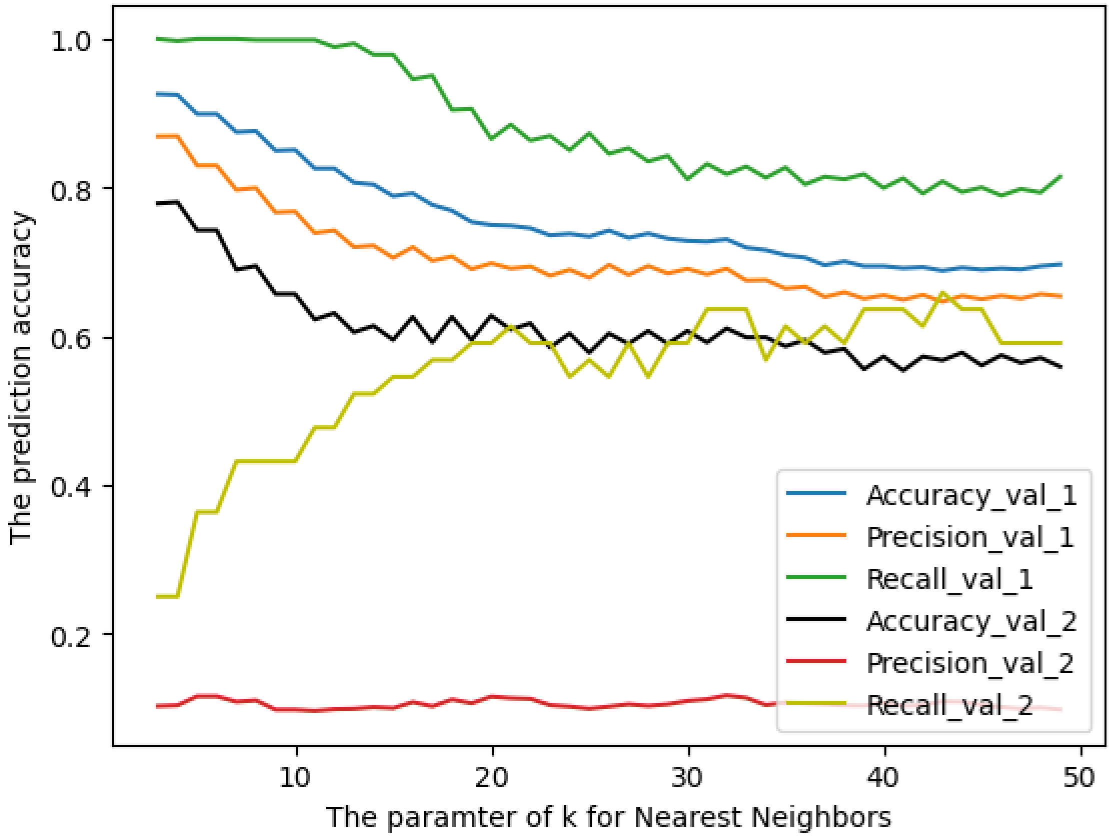
```
## Transform the data into df structure
# df_k_index_knn = pd.DataFrame.from_tuple(dict_knn_index_record)
df_k_index_knn = dict_knn_index_record[0]

df_k_index_knn.sort_values(
    by=['recall_test', 'precision_test'], ascending=False).head()
```

Out[51]:

	k	acc_val	precision_val	recall_val	acc_test	precision_test	recall_test
40	43	0.688281	0.646542	0.808511	0.567753	0.109023	0.659091
29	32	0.730496	0.691014	0.818119	0.610635	0.117155	0.636364
30	33	0.719352	0.675056	0.828415	0.598628	0.113821	0.636364
28	31	0.727457	0.683202	0.831846	0.591767	0.112000	0.636364
41	44	0.692334	0.654412	0.794097	0.578045	0.108527	0.636364

In [55]:

```
xlab = 'The paramter of k for Nearest Neighbors'
ylab = "The prediction accuracy"
# plot_dtc_depths(dict_knn_index_record, xlab, ylab, 'k')
plot_diff_para(df_k_index_knn, xlab, ylab, 'k')
```



In [56]:

```
## Choosen k: 43
df_k_index_knn.loc[df_k_index_knn['k'] == 43]
```

Out[56]:

	k	acc_val	precision_val	recall_val	acc_test	precision_test	recall_test
40	43	0.688281	0.646542	0.808511	0.567753	0.109023	0.659091

In [110…

```
## Build the best nearest neighbor model

best_k_allvar = 43
choose_knn_allvar = KNeighborsClassifier(n_neighbors=best_k_allvar)
best_knn_allvar = choose_knn_allvar.fit(x_train, y_train)

pre_best_knn_allvar = choose_knn_allvar.predict(x_test)

metric_knn = measure(pre_best_knn_allvar, y_test)
metric_knn
```

Out[110]:

```
(0.5677530017152659, 0.10902255639097744, 0.6590909090909091)
```

Logistic Regression

In [60]:

```
## import the related packages
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

In [69]:

```
logreg = LogisticRegression()
# class_weight={1:0.55, 0:0.45}

# fit the model with data
logreg.fit(x, y)

pre_test_logreg = logreg.predict(x_test)

metric_log = measure(pre_test_logreg, y_test)
metric_log
```

Out[69]:

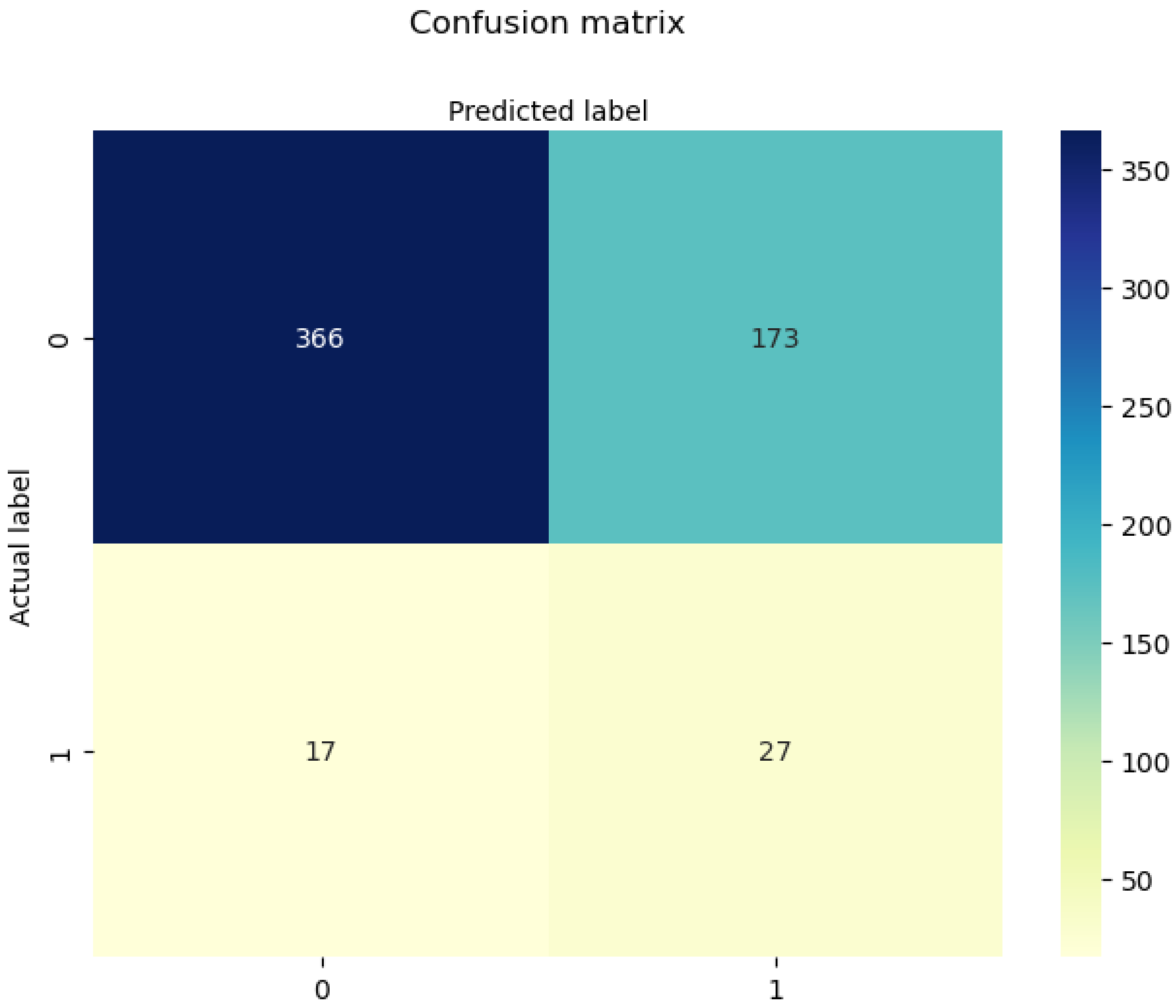
```
(0.6740994854202401, 0.135, 0.6136363636363636)
```

In [68]:

```
## Use confusion matrix to evaluate the log reg model
from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(y_test, pre_test_logreg)
# cnf_matrix

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

#Text(0.5,257.44,'Predicted label');
```

Out[68]:

```
Text(0.5, 427.9555555555555, 'Predicted label')
```



SVM

In [106…

```
from sklearn.svm import SVC

svm = SVC(random_state = 1)

svm.fit(x, y)
pre_test_svm = svm.predict(x_test)

metric_svm = measure(pre_test_svm, y_test)
metric_svm
```

Out[106]:

```
(0.6861063464837049, 0.13227513227513227, 0.5681818181818182)
```

Bayes Model

In [72]:

```
from sklearn.naive_bayes import GaussianNB

# Build a Gaussian Classifier
bayes_model = GaussianNB()

# Model training
bayes_model.fit(x, y)

# Predict Output
pre_test_bayes = bayes_model.predict(x_test)
```

```
# acc_bayes_total = accuracy_score(y_val, pre_bayes_val)
# acc_bayes_precision = precision_score(y_val, pre_bayes_val, average='binary')
# acc_bayes_recall = recall_score(y_val, pre_bayes_val, average='binary')

metric_baye = measure(pre_test_bayes, y_test)
metric_baye
```

Out[72]:

```
(0.20411663807890223, 0.08661417322834646, 1.0)
```

Feature Selection

In [73]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# apply SelectKBest class to extract top 15 best features
bestfeatures = SelectKBest(score_func=chi2, k=15)
fit = bestfeatures.fit(x, y) # balanced data

dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(name_x)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Specs','Score']  # naming the dataframe columns
# print(featureScores.nlargest(15,'Score'))  #print 10 best features
```

In [74]:

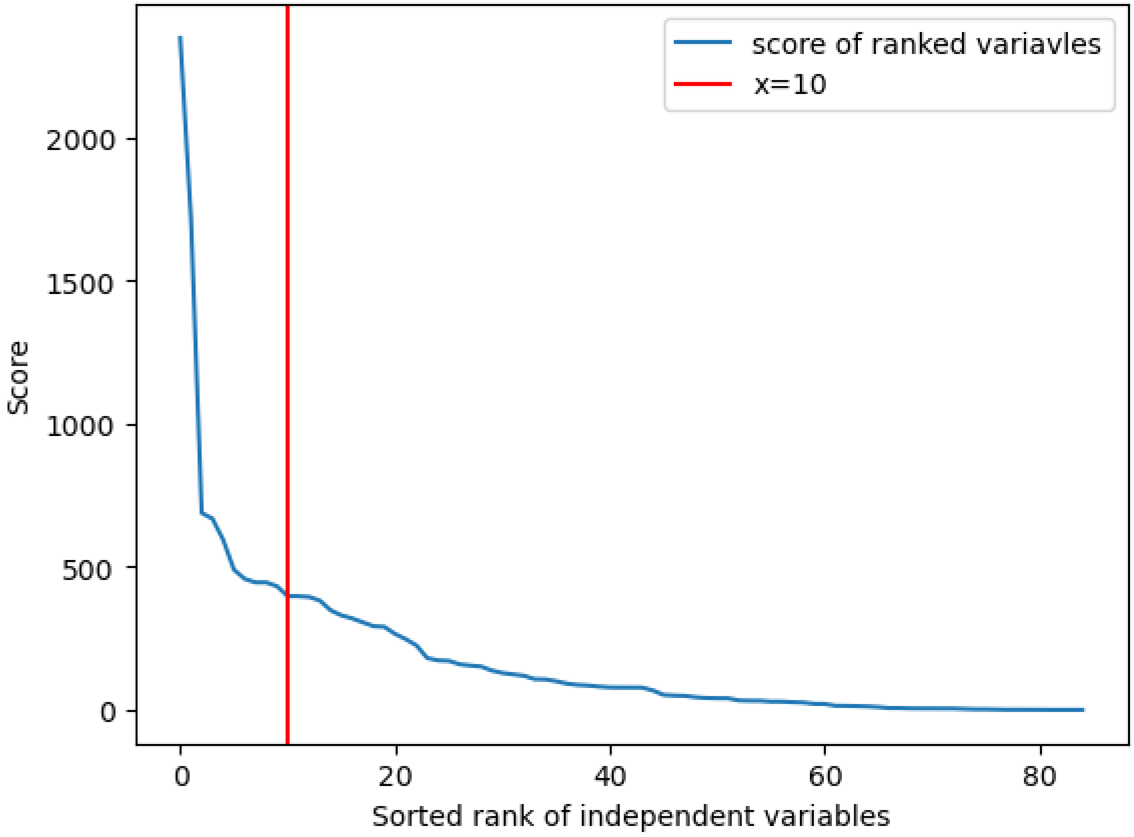
```
# sort the dataframe according to the values of score
sort_feature_score = featureScores.sort_values('Score', ascending=False)
```

In [75]:

```
sort_score = sort_feature_score['Score']
plt.plot(np.arange(len(sort_score)), sort_score, label='score of ranked variavles')
plt.axvline(x = 10, color='r', linestyle='-', label='x=10')
plt.xlabel('Sorted rank of independent variables')
plt.ylabel('Score')
plt.legend()
```

Out[75]:

```
<matplotlib.legend.Legend at 0x21fa90e6310>
```



In [76]:

```
head_feature_10 = sort_feature_score.head(10)
head_feature_10
```

Out[76]:

	Specs	Score
46	Contribution car policies	2347.500426
0	Customer Subtype	1718.351005
29	Rented house	687.890156
36	Income < 30.000	667.826813
58	Contribution fire policies	594.033645
30	Home owners	489.694486
67	Number of car policies	457.567749
17	Lower level education	445.286804
15	High level education	445.203311
60	Contribution boat policies	432.015491

use selected feature to train the models

In [77]:

```
# selected features' name
select_name_x = list(head_feature_10['Specs'])

select_col_name = select_name_x + ['CARAVAN POLICY']
select_feature_df = balance_data.loc[:, select_col_name]

select_feature_df.shape
```

Out[77]:

```
(9870, 11)
```

In [78]:

```
## Split the select_feature_df

select_x, select_y, select_name_x = split_data(select_feature_df)

select_x_train, select_x_val, select_y_train, select_y_val = split_train_val(select_x, s
```

In [79]:

```
## renew the test data according to the selected features
test_data = pd.concat([x_test, y_test], axis=1)
select_test = test_data.loc[:, select_col_name]

select_y_test = select_test['CARAVAN POLICY']
select_x_test = select_test.drop(columns='CARAVAN POLICY')
```

Try selected features on decision tree

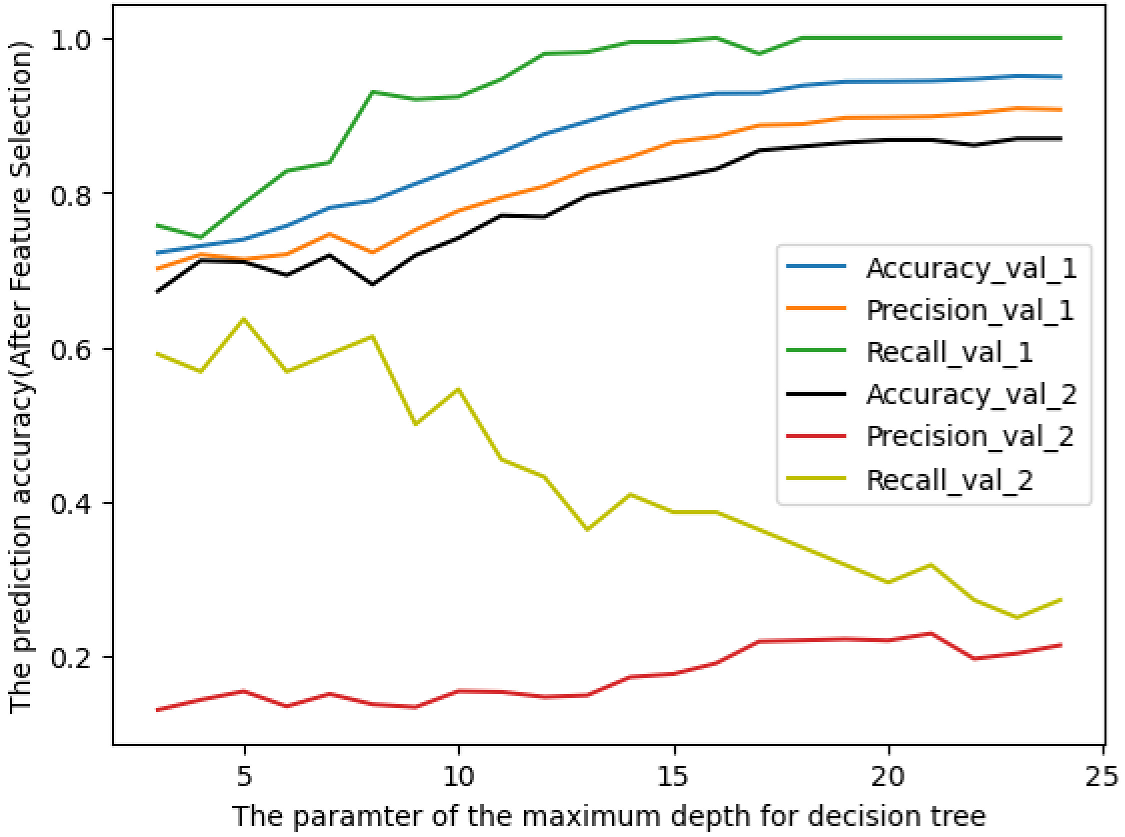
In [80]:

```
df_select_depth_index_dtc, dict_select_dtc_index_record  = record_try_parameter(
                          25, 'depth', select_x_train, select_y_train,
                          select_x_val, select_y_val, select_x_test, select_y_test)
```

```
   depth    acc_val  precision_val  recall_val  acc_test  precision_test  \
2      5  0.739277       0.713396    0.785861  0.710120        0.154696   
5      8  0.789598       0.722281    0.929993  0.680961        0.137755   
4      7  0.780142       0.746032    0.838710  0.718696        0.151163   
0      3  0.722053       0.701654    0.757035  0.672384        0.130653   
1      4  0.730834       0.719707    0.741935  0.711835        0.143678   

   recall_test  
2     0.636364  
5     0.613636  
4     0.590909  
0     0.590909  
1     0.568182  
```

In [82]:

```
xlab = 'The paramter of the maximum depth for decision tree'
ylab = "The prediction accuracy(After Feature Selection)"
plot_diff_para(dict_select_dtc_index_record, xlab, ylab, 'depth')
```



In [103…

```
# train the data on the choosen model
dtc_select_fix_dep = DecisionTreeClassifier(max_depth=5, random_state=1)
dtc_select_choose = dtc_select_fix_dep.fit(select_x_train, select_y_train)

pre_dtc_select_choose = dtc_select_choose.predict(select_x_test)

metric_dtc_select = measure(pre_dtc_select_choose, select_y_test)
print(metric_dtc_select)

name_file = "best_select_dtc_name.pdf"
show_dtc(dtc_select_choose, select_name_x, name_file)
```

```
(0.7101200686106347, 0.15469613259668508, 0.6363636363636364)
```

In [264…

```
name_file = "select_explain_dtc_name.pdf"
show_dtc(dtc_select_choose, select_name_x, name_file)
```

Nearest Neighbor

In [90]:

```
# try mex. neighbor=50
select_dict_knn_index_record =  record_try_parameter(50, 'k', select_x_train,
                                   select_y_train, select_x_val, select_y_val,
                                    select_x_test, select_y_test)
```

```
     k   acc_val  precision_val  recall_val  acc_test  precision_test  \
10  13  0.815603       0.729240    0.994509  0.631218        0.123348   
37  40  0.720365       0.676785    0.826356  0.619211        0.119658   
39  42  0.719352       0.674276    0.831160  0.617496        0.119149   
41  44  0.723067       0.676650    0.837337  0.617496        0.119149   
34  37  0.719352       0.673889    0.832533  0.615780        0.118644   

    recall_test  
10     0.636364  
37     0.636364  
39     0.636364  
41     0.636364  
34     0.636364  
```

In [91]:

```
select_df_k_index_knn = select_dict_knn_index_record[0]

select_df_k_index_knn.sort_values(
    by=['recall_test', 'precision_test'], ascending=False).head()
```

Out[91]:

	k	acc_val	precision_val	recall_val	acc_test	precision_test	recall_test
10	13	0.815603	0.729240	0.994509	0.631218	0.123348	0.636364
37	40	0.720365	0.676785	0.826356	0.619211	0.119658	0.636364
39	42	0.719352	0.674276	0.831160	0.617496	0.119149	0.636364
41	44	0.723067	0.676650	0.837337	0.617496	0.119149	0.636364
34	37	0.719352	0.673889	0.832533	0.615780	0.118644	0.636364

In [129…

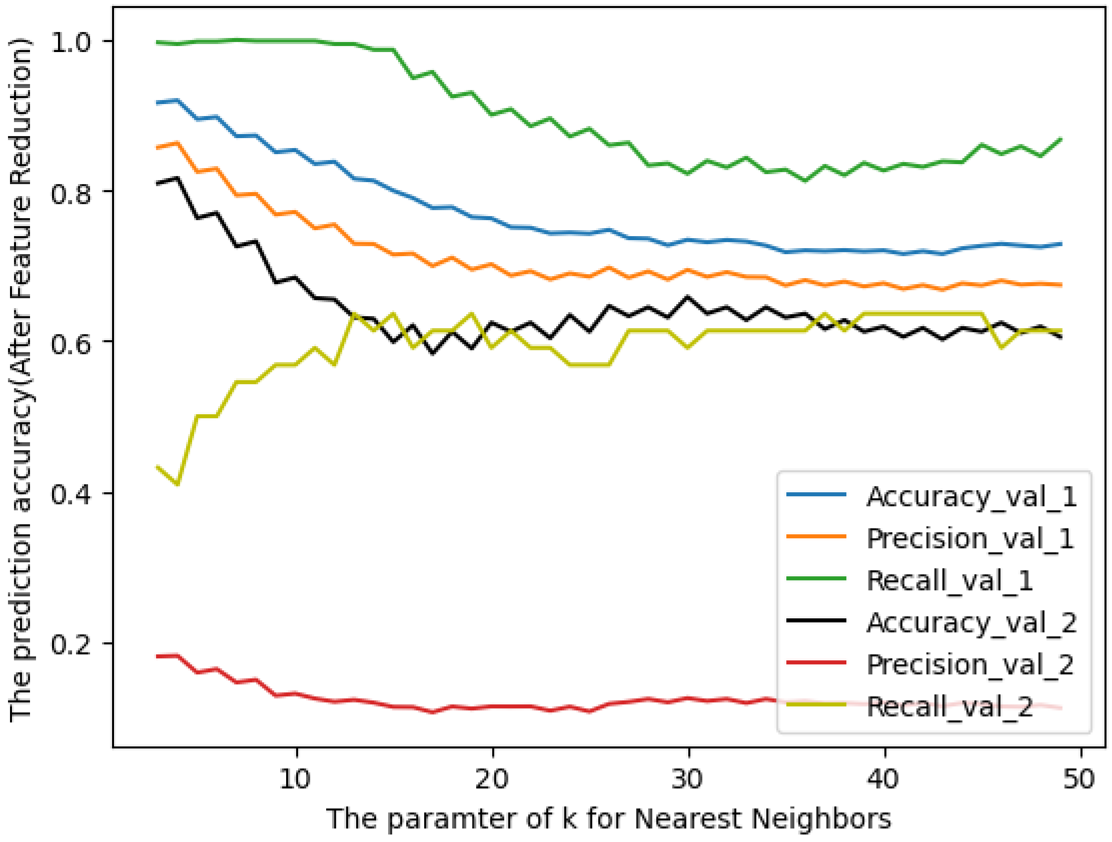
```
select_knn = KNeighborsClassifier(n_neighbors=13)
select_knn_choose = select_knn.fit(select_x, select_y)

select_pre_knn = select_knn_choose.predict(select_x_test)

metric_knn_select = measure(select_pre_knn, select_y_test)
metric_knn_select
```

Out[129]:

```
(0.6672384219554031, 0.125, 0.5681818181818182)
```

In [93]:

```
xlab = 'The paramter of k for Nearest Neighbors'
ylab = "The prediction accuracy(After Feature Reduction)"
# plot_dtc_depths(select_dict_knn_index_record, xlab, ylab, 'k')
plot_diff_para(select_df_k_index_knn, xlab, ylab, 'k')
```



Logistic

In [101…

```
logreg_select = LogisticRegression()
logreg_select.fit(select_x, select_y)

select_pre_test_logreg = logreg_select.predict(select_x_test)

metric_log_select = measure(select_pre_test_logreg, select_y_test)
print(metric_log_select)
```

```
(0.660377358490566, 0.13333333333333333, 0.6363636363636364)
```

SVM

In [100…

```
svm_select = SVC(random_state = 1)
svm_select.fit(select_x, select_y)


select_pre_test_svm = svm_select.predict(select_x_test)

metric_svm_select = measure(select_pre_test_svm, select_y_test)
print(metric_svm_select)
```

```
(0.6037735849056604, 0.11836734693877551, 0.6590909090909091)
```

Bayes

In [130…

```
bayes_select = GaussianNB()
```

```
bayes_select.fit(select_x, select_y)

select_pre_test_bay = bayes_select.predict(select_x_test)

metric_baye_select = measure(select_pre_test_bay, select_y_test)
print(metric_baye_select)
```

```
(0.7924528301886793, 0.15315315315315314, 0.38636363636363635)
```

Model Validation

In [127…

```
dict_metric_combine = {'dtc_allvar': metric_dtc,
                       'dtc_select': metric_dtc_select,
                       'knn_allvar': metric_knn,
                       'knn_select': metric_knn_select,
                       'logistics_allvar': metric_log,
                       'logistics_select': metric_log_select,
                       'svm_allvar': metric_svm,
                       'svm_select': metric_svm_select,
                       'bayes_allvar' : metric_baye,
                       'bayes_select' : metric_baye_select}

df_metric_combine = pd.DataFrame.from_dict(dict_metric_combine, orient='index')
df_metric_combine.columns = ['Accuracy', 'Precision', 'Recall']
df_metric_combine
```

Out[127]:

	Accuracy	Precision	Recall
dtc_allvar	0.569468	0.106464	0.636364
dtc_select	0.710120	0.154696	0.636364
knn_allvar	0.567753	0.109023	0.659091
knn_select	0.631218	0.123348	0.636364
logistics_allvar	0.674099	0.135000	0.613636
logistics_select	0.660377	0.133333	0.636364
svm_allvar	0.686106	0.132275	0.568182
svm_select	0.603774	0.118367	0.659091
bayes_allvar	0.204117	0.086614	1.000000
bayes_select	0.792453	0.153153	0.386364

Build Finial Model

In [249…

```
def combine_model(model_list, model_name, test_x, test_y):
    dict_record_pred_test = {}


    for i in range(len(model_name)):
        model = model_list[i]
        pred = model.predict(test_x)
        dict_record_pred_test[model_name[i]] = pred

        if model_name[i] == 'log':
            result = model.predict_proba(test_x)
            prob = []
            for aa in result:
                prob.append(aa[1])
            dict_record_pred_test['prob'] = prob

    dict_record_pred_test['true_y'] = test_y

    return dict_record_pred_test
```

In [250…

```
model_list = [dtc_select_choose, select_knn_choose, logreg_select, svm_select, bayes_sel
model_name = ['dtc', 'knn', 'log', 'svm', 'bayes']

dict_record_pred_test = combine_model(model_list, model_name, select_x_test, select_y_te

df_record_pred_test = pd.DataFrame.from_dict(dict_record_pred_test)
df_record_pred_test.head()
```

Out[250]:

	dtc	knn	log	prob	svm	bayes	true_y
840	0	0	0	0.234913	0	0	0
3338	0	0	0	0.215518	0	0	0
2976	0	0	1	0.576011	1	0	0
5114	1	0	1	0.563683	1	0	0
527	1	0	1	0.570594	1	0	0

In [251…

```
df_record_pred_test['sum_vote'] = df_record_pred_test[model_name].sum(axis=1)
```

In [252…

```
sort_df_record_pred_test = df_record_pred_test.sort_values(by=['sum_vote', 'prob'], asce
sort_df_record_pred_test.head()
```

Out[252]:

	dtc	knn	log	prob	svm	bayes	true_y	sum_vote
3509	1	1	1	0.984121	1	1	1	5
5172	1	1	1	0.837080	1	1	0	5
2986	1	1	1	0.833328	1	1	0	5
4754	1	1	1	0.822062	1	1	0	5
394	1	1	1	0.818030	1	1	0	5

In [253…

```
def majority_vote_model(test_x, test_y, model_list, model_name, num_pred):
    dict_record_pred_test = combine_model(model_list, model_name, test_x, test_y)

    df_record_pred_test = pd.DataFrame.from_dict(dict_record_pred_test)
#     print(df_record_pred_test.head())
    df_record_pred_test['sum_vote'] = df_record_pred_test[model_name].sum(axis=1)

    sort_df_record_pred_test = df_record_pred_test.sort_values(
                by=['sum_vote', 'prob'], ascending=False)

    if num_pred != 'None':
        first_num_pred = sort_df_record_pred_test[:num_pred]

        tp = sum(first_num_pred['true_y'])

        print('Number of right predicted target:', tp)
        print('Totoal true target:', sum(test_y))
```

In [254…

```
len(y_test), sum(y_test)
```

Out[254]:

```
(583, 44)
```

In [255…

```
majority_vote_model(select_x_test, select_y_test, model_list, model_name, 88)
```

```
Number of right predicted target: 17
Totoal true target: 44
```

Customer Selection

Use single method

In [256…

```
## import csv file
og_test = pd.read_csv('F://UM//Data Mining//Assi_2//caravanTest.csv')

## split the test data into x and y
last_y = og_test['CARAVAN POLICY']
last_x = og_test.drop(columns='CARAVAN POLICY')
```

In [257…

```
select_test_df = og_test.loc[:, select_col_name]

select_last_y = select_test_df['CARAVAN POLICY']
select_last_x = select_test_df.drop(columns='CARAVAN POLICY')
```

In [258…

```
model_list = [dtc_select_choose, select_knn_choose, logreg_select, svm_select, bayes_sel
model_name = ['dtc', 'knn', 'log', 'svm', 'bayes']

majority_vote_model(select_last_x, select_last_y,
                    model_list, model_name, 800)
```

```
Number of right predicted target: 108
Totoal true target: 238
```

In [260…

```
model_list = [dtc_choose, choose_knn_allvar, logreg, svm, bayes_model]
model_name = ['dtc', 'knn', 'log', 'svm', 'bayes']

majority_vote_model(last_x, last_y,
                    model_list, model_name, 800)
```

```
Number of right predicted target: 118
Totoal true target: 238
```