OPT Machine Learning Final Project

YIJING TAN

In [27]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE
from scipy.optimize import fmin_tnc  # compute the minimum for func
tion
from sklearn import svm
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
import itertools
#pip install imblearn
#pip install spicy
```

1. Read excel and import data sheet

In [28]:
```
xls = pd.ExcelFile('patient.xlsx')
original_data = pd.read_excel(xls, 'original')
original_data.shape
original_data
```

Out[28]:

	patient_id	sex	age	province	infection_case	state
0	1000000001	male	50s	Seoul	overseas inflow	released
1	1000000002	male	30s	Seoul	overseas inflow	released
2	1000000003	male	50s	Seoul	contact with patient	released
3	1000000004	male	20s	Seoul	overseas inflow	released
4	1000000005	female	20s	Seoul	contact with patient	released
...	...	...	...	...	...	...
5160	7000000015	female	30s	Jeju-do	overseas inflow	released
5161	7000000016	NaN	NaN	Jeju-do	overseas inflow	released
5162	7000000017	NaN	NaN	Jeju-do	overseas inflow	isolated
5163	7000000018	NaN	NaN	Jeju-do	overseas inflow	isolated
5164	7000000019	NaN	NaN	Jeju-do	overseas inflow	isolated

5165 rows × 6 columns

2. Descriptive analysis

We performed descriptive analyses of the predictors by respective stratification groups and present the results as numbers

In [3]:
```
gp1=original_data.groupby(by=['sex'])
gp1.size()
```

Out[3]:
```
sex
female    2218
male      1825
dtype: int64
```

In [4]:
```
gp2=original_data.groupby(by=['age'])
gp2.size()
```

Out[4]:
```
age
0s        66
100s       1
10s      178
20s      899
30s      523
40s      518
50s      667
60s      482
70s      232
80s      170
90s       49
dtype: int64
```

In [5]:
```
gp3=original_data.groupby(by=['province'])
gp3.size()
```

Out[5]:
```
province
Busan                 151
Chungcheongbuk-do      56
Chungcheongnam-do     168
Daegu                 137
Daejeon               119
Gangwon-do             63
Gwangju                44
Gyeonggi-do          1208
Gyeongsangbuk-do     1254
Gyeongsangnam-do      133
Incheon               343
Jeju-do                19
Jeollabuk-do           27
Jeollanam-do           25
Sejong                 51
Seoul                1312
Ulsan                  55
dtype: int64
```

In [6]:
```
gp4=original_data.groupby(by=['infection_case'])
gp4.size()
```

Out[6]:
```
infection_case
Anyang Gunpo Pastors Group                          1
Biblical Language study meeting                     3
Bonghwa Pureun Nursing Home                        31
Changnyeong Coin Karaoke                            4
Cheongdo Daenam Hospital                           21
Coupang Logistics Center                           80
Daejeon door-to-door sales                          1
Daezayeon Korea                                     3
Day Care Center                                    43
Dongan Church                                      17
Dunsan Electronics Town                            13
Eunpyeong St. Mary's Hospital                      16
Gangnam Dongin Church                               1
Gangnam Yeoksam-dong gathering                      6
Geochang Church                                     6
Geumcheon-gu rice milling machine manufacture       6
Guri Collective Infection                           5
Guro-gu Call Center                               112
Gyeongsan Cham Joeun Community Center              10
Gyeongsan Jeil Silver Town                         12
Gyeongsan Seorin Nursing Home                      15
Itaewon Clubs                                     162
KB Life Insurance                                  13
Korea Campus Crusade of Christ                      7
Milal Shelter                                      11
Ministry of Oceans and Fisheries                   28
Onchun Church                                      33
Orange Life                                         1
Orange Town                                         7
Pilgrimage to Israel                                2
Richway                                           128
River of Grace Community Church                     1
SMR Newly Planted Churches Group                   36
Samsung Fire & Marine Insurance                     4
Samsung Medical Center                              7
Seocho Family                                       5
Seongdong-gu APT                                   13
Seoul City Hall Station safety worker               3
Shincheonji Church                                107
Suyeong-gu Kindergarten                             3
Uiwang Logistics Center                             2
Wangsung Church                                    24
Yangcheon Table Tennis Club                        44
Yeonana News Class                                  5
Yeongdeungpo Learning Institute                     3
Yongin Brothers                                     4
contact with patient                             1610
etc                                               703
gym facility in Cheonan                            30
gym facility in Sejong                              4
overseas inflow                                   840
dtype: int64
```

In [7]:
```
gp5=original_data.groupby(by=['state'])
gp5.size()
```

Out[7]:
```
state
deceased      78
isolated    2158
released    2929
dtype: int64
```

3. Data prepocessing

As we can see above, the infection_case variable has too many types, so we wanna regroup it to fewer categories. Specifically, we followed prior study to recategory it to 9 groups, which are: contact with patient,overseas inflow,etc,Nursing home,Hospital ,Religious gathering, Call center, CSA(Community center, shelter and apartment),Gym facility. Also, since our goal is to perdict the mortality,we need to catergorize both the released and isolated cases to survived group. Then we deleted some cases that have missing values. This step was performed via excel and we attached the spreedsheet which demonstrates our procedures.

4. Import prepocessed data

In [29]:
```
data = pd.read_excel(xls, 'DP(without missing)')
data = data.drop('patient_id', 1)
data
```

Out[29]:

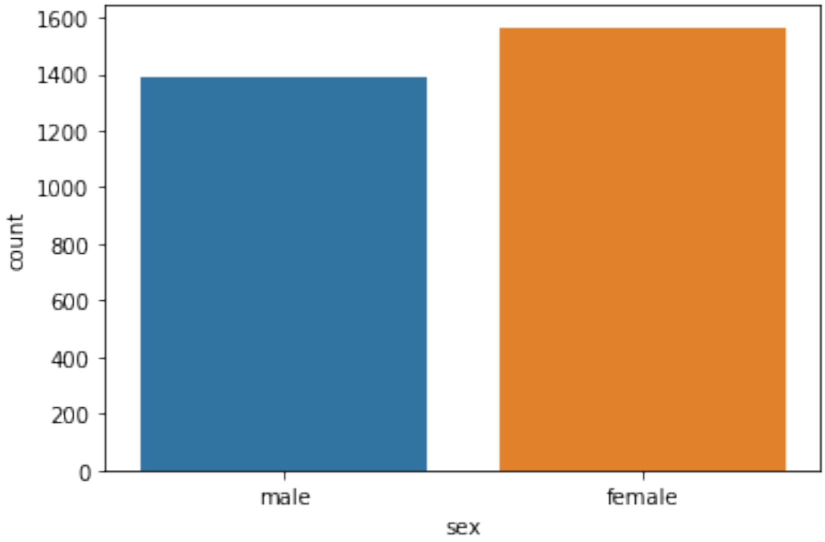
	sex	age	province	infection_case	state
0	male	50s	Seoul	overseas inflow	Survived
1	male	30s	Seoul	overseas inflow	Survived
2	male	50s	Seoul	contact with patient	Survived
3	male	20s	Seoul	overseas inflow	Survived
4	female	20s	Seoul	contact with patient	Survived
...	...	...	...	...	...
2950	male	30s	Jeju-do	contact with patient	Survived
2951	female	20s	Jeju-do	overseas inflow	Survived
2952	female	10s	Jeju-do	overseas inflow	Survived
2953	female	30s	Jeju-do	CSA	Survived
2954	female	30s	Jeju-do	overseas inflow	Survived

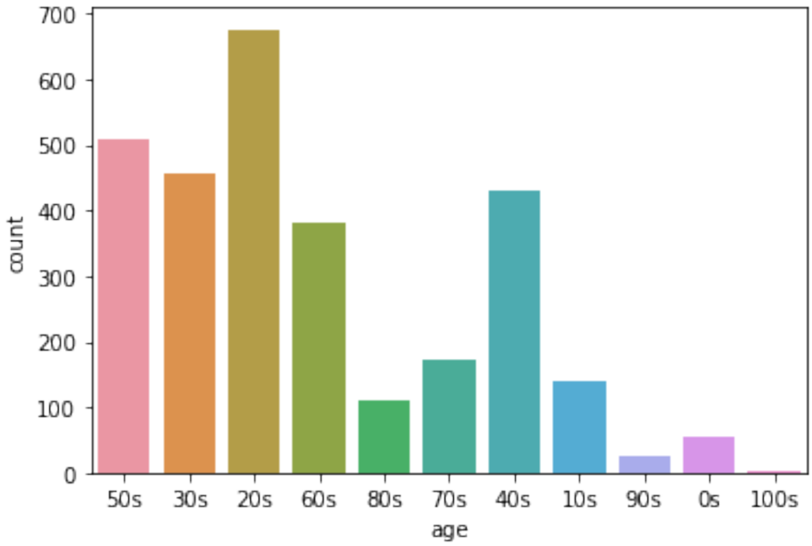
2955 rows × 5 columns

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

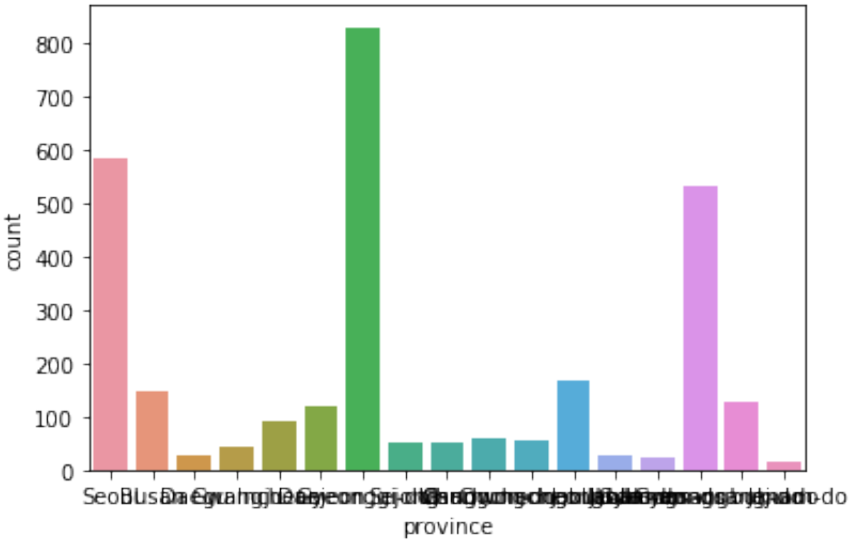
Out[28]: (2955, 31)

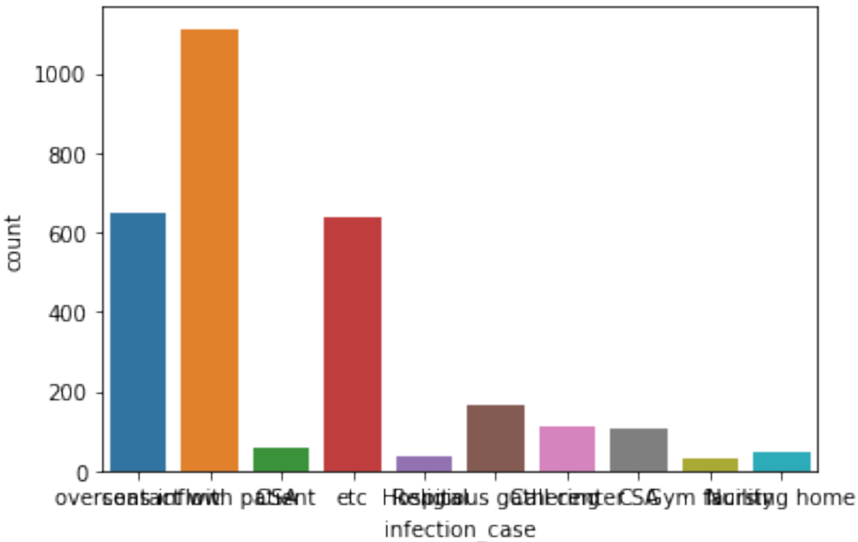
In [36]:
```
sns.countplot(x = "sex", data=data)
plt.show()
sns.countplot(x = "age", data=data)
plt.show()
```

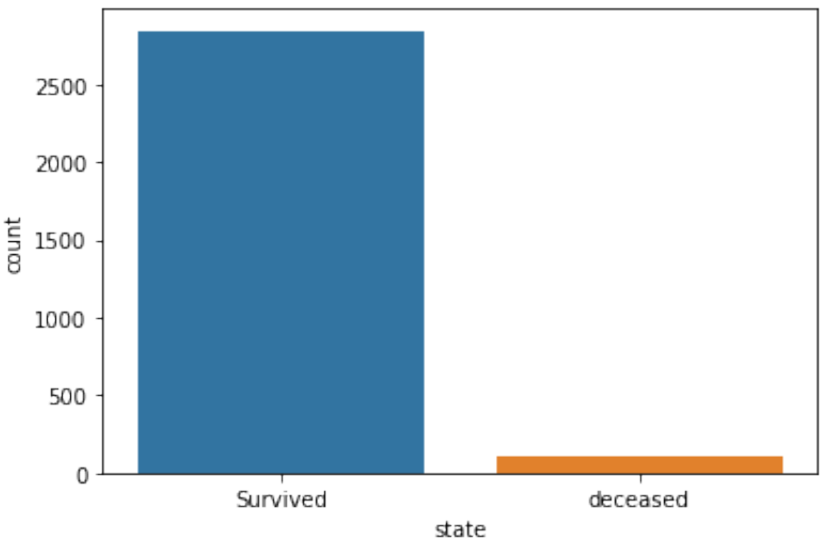




In [35]:
```
sns.countplot(x = "province", data=data)
plt.show()
sns.countplot(x = "infection_case", data=data)
plt.show()
sns.countplot(x = "state", data=data)
plt.show()
```







5. Encode categorical data

Our data contains categorical data, so we must encode it to numbers before we can fit and evaluate the model.we will use one hot encoding approach to encode categorical features.

In [30]:
```
data['sex'].replace(['female','male'],[0,1],inplace=True)
data['state'].replace(['deceased','Survived'],[0,1],inplace=True)
data['age'].replace(['0s','10s','20s','30s','40s','50s','60s','70s'
,'80s','90s','100s'],[1,2,3,4,5,6,7,8,9,10,11],inplace=True)

# using one hot encoding approach to encode categorical features
data = pd.get_dummies(data, columns=["province"])
data = pd.get_dummies(data, columns=["infection_case"])

# create intercept
data['intercept'] = [1] * data.iloc[:,:].shape[0]

# move depedendent variable to last column
# move intercept to first column
data = data[[ col for col in data.columns if col != 'state' ] + ['s
tate']]
data = data[['intercept'] + [ col for col in data.columns if col !=
'intercept' ]]
data
```

Out[30]:

	intercept	sex	age	province_Busan	province_Chungcheongbuk-do	province_Chungche
0	1	1	6	0	0	
1	1	1	4	0	0	
2	1	1	6	0	0	
3	1	1	3	0	0	
4	1	0	3	0	0	
...	...	...	...	...	...	
2950	1	1	4	0	0	
2951	1	0	3	0	0	
2952	1	0	2	0	0	
2953	1	0	4	0	0	
2954	1	0	4	0	0	

2955 rows × 31 columns

In [60]:
```
type(data.iloc[0,30])
```

Out[60]: numpy.int64

6. Logistic regression model

6.1 Logistic regression method 1

In method 1, we define the model by ourselves. We use sigmoid function as activation function, and set threshold to 0.5.

In [31]:
```
# retrieve numpy array
dataset = data.values
# split into input (X) and output (y) variables
X = dataset[:, :-1]
y = dataset[:,-1]
# reshape target to be a 2d array
y = y.reshape((len(y), 1))
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.2, random_state=1)
# summarize
print('Train', X_train.shape, y_train.shape)
print('Test', X_test.shape, y_test.shape)

theta = np.zeros((X_train.shape[1], 1))
```

```
Train (2364, 30) (2364, 1)
Test (591, 30) (591, 1)
```

In [32]:
```
def sigmoid(x):
    # Activation function used to map any real value between 0 and
1
    return 1 / (1 + np.exp(-x))

def net_input(theta, x):
    # Computes the weighted sum of inputs
    return np.dot(x, theta)

def probability(theta, x):
    # Returns the probability after passing through sigmoid
    return sigmoid(net_input(theta, x))

def cost_function(theta, x, y):
    # Computes the cost function for all the training samples
    m = x.shape[0]
    total_cost = -(1 / m) * np.sum(
        y * np.log(probability(theta, x)) + (1 - y) * np.log(
            1 - probability(theta, x)))
    return total_cost
```

```
def gradient(theta, x, y):
    # Computes the gradient of the cost function at the point theta
    m = x.shape[0]
    return (1 / m) * np.dot(x.T, sigmoid(net_input(theta,   x)) - y
)

def fit(x, y, theta):
    opt_weights = fmin_tnc(func=cost_function, x0=theta,
                  fprime=gradient,args=(x, y.flatten()))
    return opt_weights[0]

def predict(x):
    theta = parameters[:, np.newaxis]
    return probability(theta, x)

def accuracy(x, actual_classes, probab_threshold=0.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    accuracy_score = np.mean(predicted_classes == actual_classes)
    return accuracy_score

def auc(x, actual_classes, probab_threshold=0.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    auc_score = metrics.roc_auc_score(actual_classes.flatten(),pred
icted_classes)
    return auc_score

def precision(x, actual_classes, probab_threshold=0.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    precision_score = metrics.precision_score(actual_classes.flatte
n(),predicted_classes)
    return precision_score

def recall(x, actual_classes, probab_threshold=0.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    recall_score = metrics.recall_score(actual_classes.flatten(),pr
edicted_classes)
    return recall_score

def class_report(x, actual_classes, probab_threshold=0.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    print(classification_report(actual_classes.flatten(), predicted
_classes))
    return
```

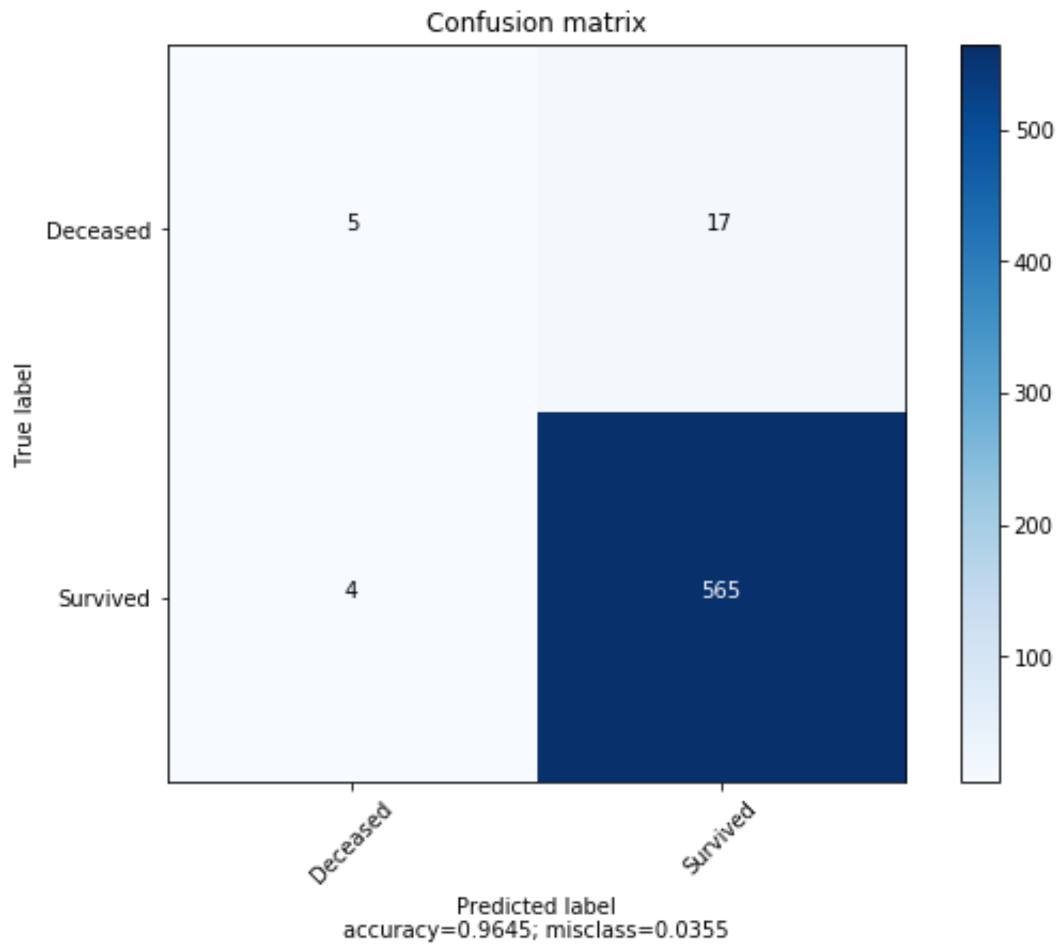
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=False):

    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shap
e[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black
")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black
")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}
'.format(accuracy, misclass))
    plt.show()

def confusion_matrix_method_1(x, actual_classes, probab_threshold=0
.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    plot_confusion_matrix(confusion_matrix(actual_classes.flatten()
```

```
, predicted_classes), ['Deceased', 'Survived'])
```

In [33]:

```
# fit the training data
parameters = fit(X_train, y_train, theta)
# calculate accurancy result
accuracy_6_1 = accuracy(X_test, y_test)
auc_6_1 = auc(X_test, y_test)
precision_6_1 = precision(X_test, y_test)
recall_6_1 = recall(X_test, y_test)

print('Accuracy:', accuracy_6_1)
print('Auc:', auc_6_1)
print('Precision:', precision_6_1)
print('Recall:', recall_6_1)
#class_report(X_test, y_test)
confusion_matrix_method_1(X_test, y_test)
```

```
Accuracy: 0.9486802889364151
Auc: 0.6101214251477872
Precision: 0.9707903780068728
Recall: 0.9929701230228472
```



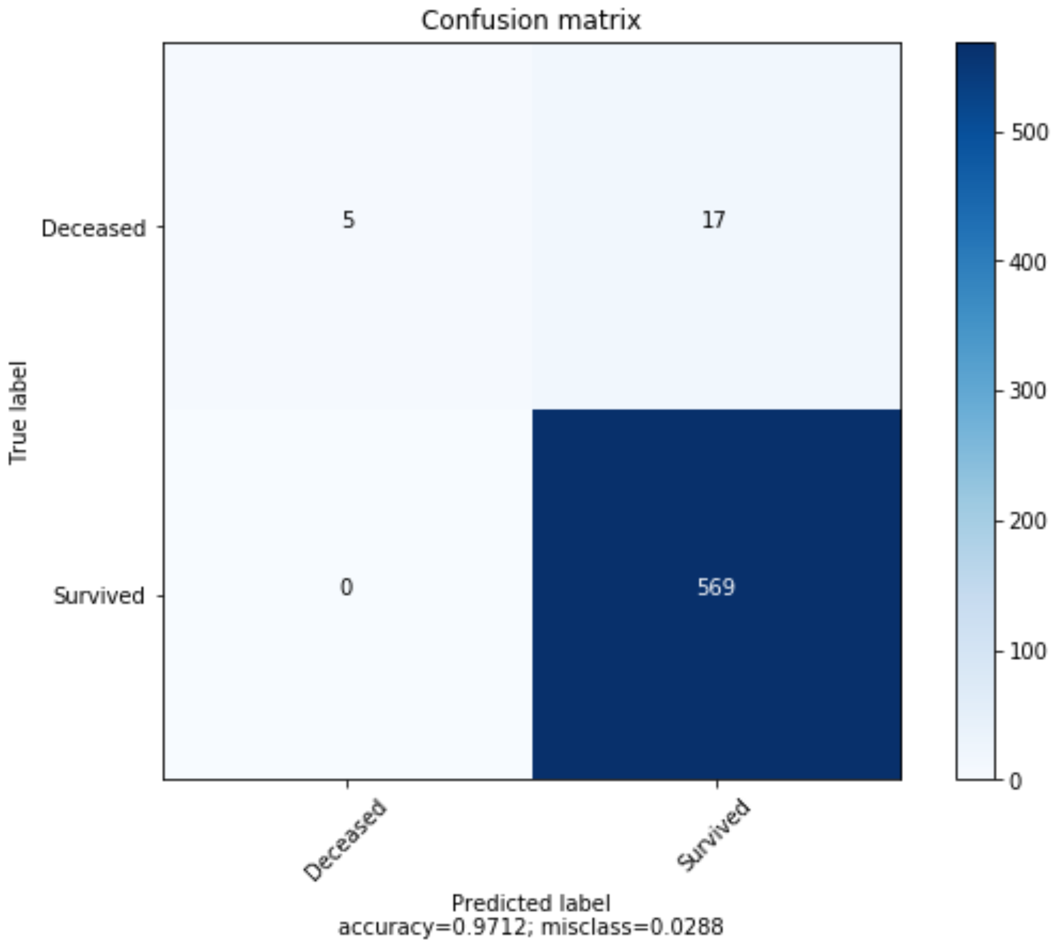
We get a revelative low AUC acore compared with other matrics.

From the confusion matrix we can see that we have 21 worng cases. There are 17 decreased cases but we predicte as survived and 4 survived cases but we predict as decreased.

6.2 Logistic regression method 2 : implement classifier using scikit-learn

In [34]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
predicted_classes = model.predict(X_test)
#parameters = model.coef_
accuracy_6_2 = metrics.accuracy_score(y_test,predicted_classes)
auc_6_2 = metrics.roc_auc_score(y_test,predicted_classes)
precision_6_2 = metrics.precision_score(y_test.flatten(),predicted_
classes)
recall_6_2 = metrics.recall_score(y_test.flatten(),predicted_classe
s)

print('Accuracy:', accuracy_6_2)
print('Auc:', auc_6_2)
print('Precision:', precision_6_2)
print('Recall:', recall_6_2)
#print(classification_report(y_test, predicted_classes))
plot_confusion_matrix(confusion_matrix(y_test, predicted_classes),
['Deceased', 'Survived'])
```

```
Accuracy: 0.9712351945854484
Auc: 0.6136363636363636
Precision: 0.9709897610921502
Recall: 1.0
```



The AUC score is also low. This might be caused by imbalanced data. So we will deal with imbalanced data in the following dection.

From the confusion matrix we can see that we have 17 worng cases. There are 17 decreased cases but we predicte as survived

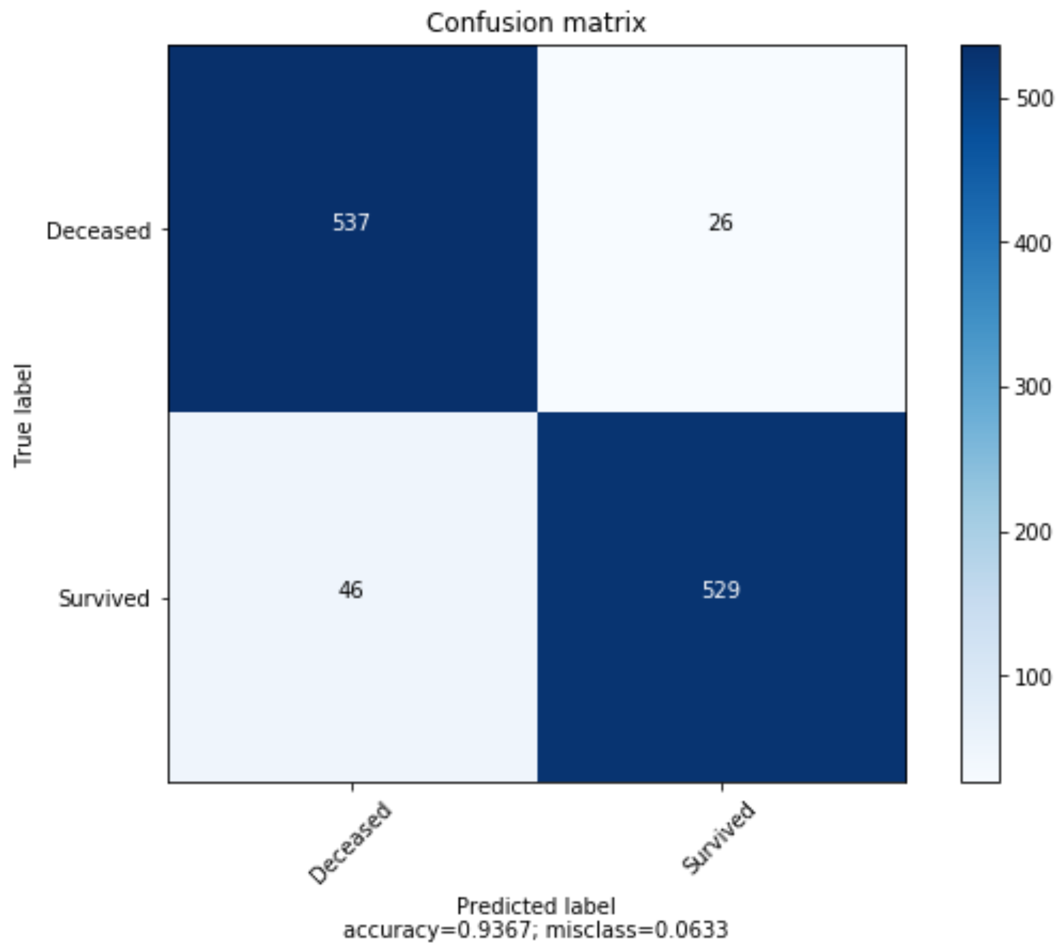
6.3 Logistic regression: using SMOTE to deal with the imbalanced data

As our data was imbalanced, we applied one oversampling technique called synthetic minority oversampling technique (SMOTE) to enhance the learning on the training data (Chawla et al.,2002; Nnamoko & Korkontzelos, 2020). SMOTE creates synthetic samples from the minority class (cases with deaths in our data) according to feature space similarities between nearest neighbors.

In [47]:
```
# Using SMOTE to deal with the imbalanced data

x_resampled,y_resampled = SMOTE().fit_sample(X,y)
X_train_re, X_test_re, y_train_re, y_test_re = train_test_split(x_r
esampled, y_resampled, test_size=0.2, random_state=1)
```

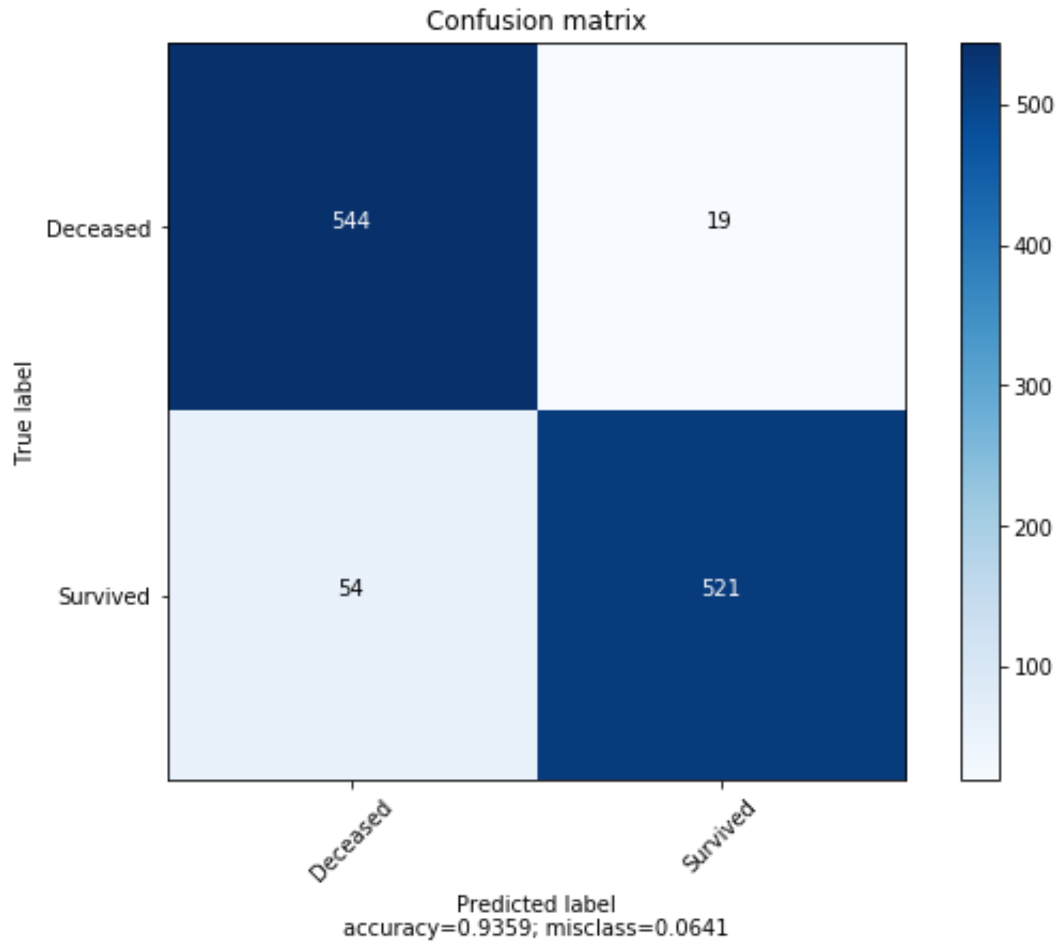
In [35]:
```
# fit Logistic regression method 1 with balanced data
parameters = fit(X_train_re, y_train_re, theta)
accuracy_6_3_1 = accuracy(X_test_re, y_test_re)
auc_6_3_1 = auc(X_test_re, y_test_re)
precision_6_3_1 = precision(X_test_re, y_test_re)
recall_6_3_1 = recall(X_test_re, y_test_re)

print('#######Logistic regression method 1 with balanced dataset:##
#####')
print('Accuracy:', accuracy_6_3_1)
print('Auc:', auc_6_3_1)
print('Precision:', precision_6_3_1)
print('Recall:', recall_6_3_1)
#class_report(X_test_re, y_test_re)
confusion_matrix_method_1(X_test_re, y_test_re)
print('')
print('')

# fit Logistic regression method 2 with balanced data
model.fit(X_train_re, y_train_re)
predicted_classes_re = model.predict(X_test_re)
accuracy_6_3_2 = metrics.accuracy_score(y_test_re,predicted_classes
_re)
auc_6_3_2 = metrics.roc_auc_score(y_test_re,predicted_classes_re)
precision_6_3_2 = metrics.precision_score(y_test_re.flatten(),predi
cted_classes_re)
recall_6_3_2 = metrics.recall_score(y_test_re.flatten(),predicted_c
lasses_re)

print('#######Logistic regression method 2 with balanced dataset:##
######')
print('Accuracy:', accuracy_6_3_2)
print('Auc:', auc_6_3_2)
print('Precision:', precision_6_3_2)
print('Recall:', recall_6_3_2)
#print(classification_report(y_test_re, predicted_classes_re))
plot_confusion_matrix(confusion_matrix(y_test_re, predicted_classes
_re), ['Deceased', 'Survived'])
```

```
#######Logistic regression method 1 with balanced dataset:#######
Accuracy: 0.9367311072056239
Auc: 0.9369094138543517
Precision: 0.9531531531531532
Recall: 0.92
```




```
#######Logistic regression method 2 with balanced dataset:########
Accuracy: 0.9358523725834798
Auc: 0.9361695883852035
Precision: 0.9648148148148148
Recall: 0.9060869565217391
```

```
Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```



All four metrics performed well on our models.

Method 1: From the confusion matrix we can see that we have 72 worng cases. There are 26 decreased cases but we predicte as survived and 46 survived cases but we predict as decreased.

Method 2: From the confusion matrix we can see that we have 73 worng cases. There are 19 decreased cases but we predicte as survived and 54 survived cases but we predict as decreased.

7. SVM method

7.1 SVM Classifier using linear kernal

In [12]:
```
#Create a svm Classifier
svm_linear = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
svm_linear.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_linear = svm_linear.predict(X_test)

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_linear))
print("AUC:",metrics.roc_auc_score(y_test, y_pred_linear))
```

```
Accuracy: 0.9644670050761421
AUC: 0.56642434893753
```

7.2 SVM Classifier using poly kernal

In [38]:
```
#Create a svm Classifier using poly kernal
svm_poly = svm.SVC(kernel='poly')

svm_poly.fit(X_train, y_train)

y_pred_poly = svm_poly.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred_poly))
print("AUC:",metrics.roc_auc_score(y_test, y_pred_poly))
```

```
Accuracy: 0.961082910321489
AUC: 0.5209698034829845
```

7.3 SVM Classifier using RBF kernal

In [39]:
```
#Create a svm Classifier using RBF kernal
svm_rbf = svm.SVC(kernel='rbf')

svm_rbf.fit(X_train,

y_pred_rbf = svm_rbf

print("Accuracy:",me                  ore(y_test, y_pred_rbf))
print("AUC:",metrics                  test, y_pred_rbf))
```

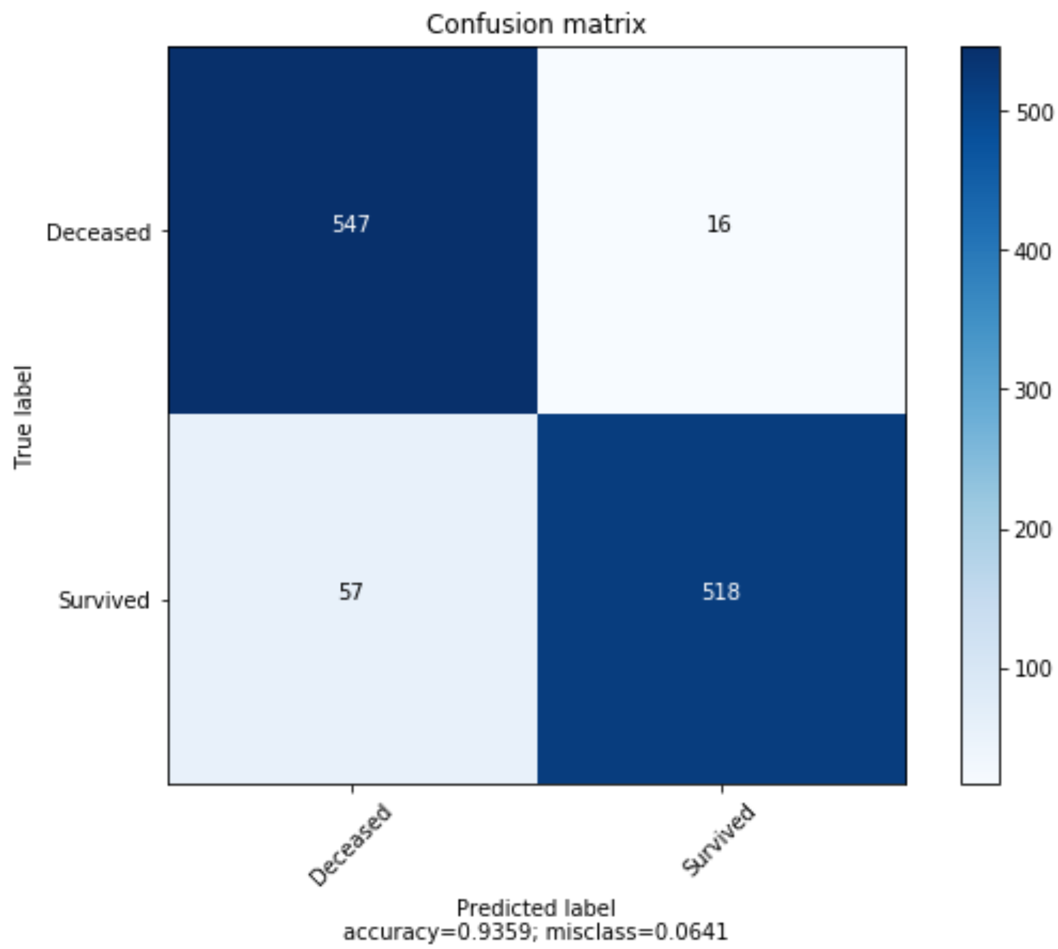
```
Accuracy: 0.9627749576988156
AUC: 0.5
```

7.4 SVM: using SMOTE to deal with the imbalanced data

As we can see, if we don't deal with the imbalanced data, the AUC is close to 0.5, which is very low. So we also used SMOTE to try to enhance the learning on the training data.

In [48]:
```
svm_linear = svm.SVC(kernel='linear') # Linear Kernel

svm_linear.fit(X_train_re, y_train_re)

y_pred_linear_re = svm_linear.predict(X_test_re)

accuracy_7_4 = metrics.accuracy_score(y_test_re,y_pred_linear_re)
auc_7_4 = metrics.roc_auc_score(y_test_re,y_pred_linear_re)
precision_7_4 = metrics.precision_score(y_test_re,y_pred_linear_re)
recall_7_4 = metrics.recall_score(y_test_re,y_pred_linear_re)

print('Accuracy:', accuracy_7_4)
print('Auc:', auc_7_4)
print('Precision:', precision_7_4)
print('Recall:', recall_7_4)
#print(classification_report(y_test_re, y_pred_linear_re))

plot_confusion_matrix(confusion_matrix(y_test_re, y_pred_linear_re)
, ['Deceased', 'Survived'])
```

```
Accuracy: 0.9358523725834798
Auc: 0.9362251911344505
Precision: 0.9700374531835206
Recall: 0.9008695652173913
```



From the confusion matrix we can see that we have 73 worng cases. There are 16 decreased cases but we predicte as survived and 57 survived cases but we predict as decreased.

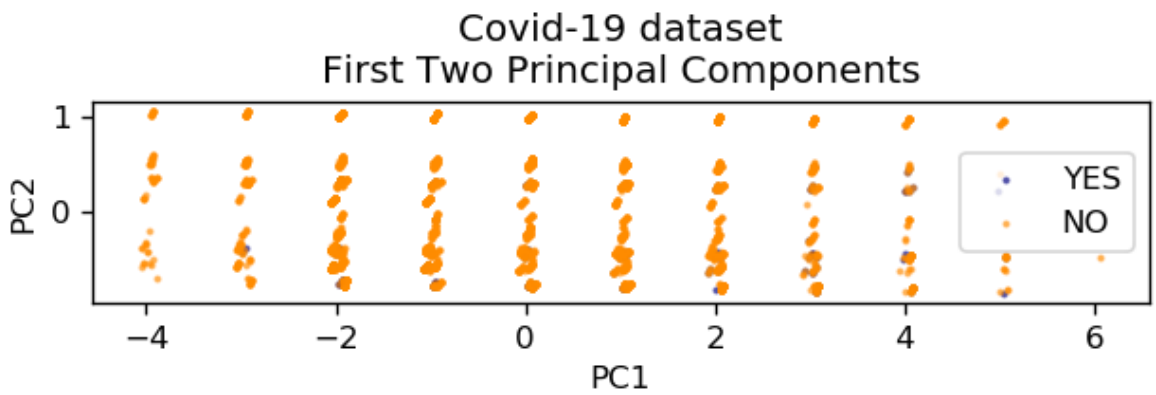
8. Visualize LR classifier

The purpose of this section is to visualize the decision boundary of a logistic regression ruler. In order to better visualize the decision boundary, we will perform principal component analysis (PCA) on the data to reduce the dimensionality to 2 dimensions.

8.1 Visualize on imbalanced data

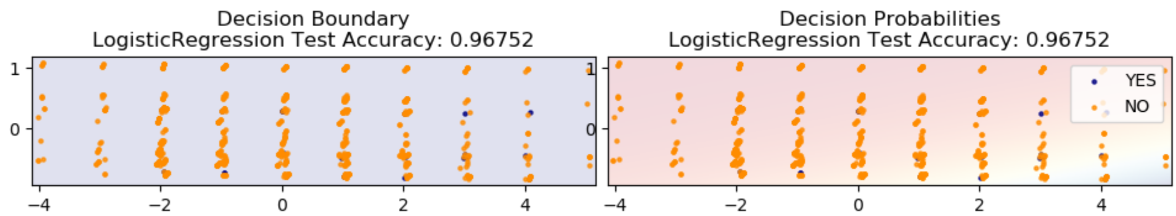
In [55]:

```
X = data.iloc[:,:-1]
y = data.iloc[:,-1]
pca = PCA(n_components=2).fit_transform(X)
X_train1, X_test1, y_train1, y_test1 = train_test_split(pca, y, ran
dom_state=0)

plt.figure(dpi=120)
plt.scatter(pca[y.values==0,0], pca[y.values==0,1], alpha=0.5, labe
l='YES', s=2, color='navy')
plt.scatter(pca[y.values==1,0], pca[y.values==1,1], alpha=0.5, labe
l='NO', s=2, color='darkorange')
plt.legend()
plt.title('Covid-19 dataset\nFirst Two Principal Components')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.gca().set_aspect('equal')
plt.show()
```



In [59]:
```
def plot_bank(X, y, fitted_model):
    plt.figure(figsize=(9.8,5), dpi=100)
    for i, plot_type in enumerate(['Decision Boundary', 'Decision P
robabilities']):
        plt.subplot(1,2,i+1)
        mesh_step_size = 0.01  # step size in the mesh
        x_min, x_max = X[:, 0].min() - .1, X[:, 0].max() + .1
        y_min, y_max = X[:, 1].min() - .1, X[:, 1].max() + .1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, mesh_step_size
), np.arange(y_min, y_max, mesh_step_size))
        if i == 0:
            Z = fitted_model.predict(np.c_[xx.ravel(), yy.ravel()])
        else:
            try:
                Z = fitted_model.predict_proba(np.c_[xx.ravel(), yy
.ravel()])[:,1]
            except:
                plt.text(0.4, 0.5, 'Probabilities Unavailable', hor
izontalalignment='center',
                         verticalalignment='center', transform = pl
t.gca().transAxes, fontsize=12)
                plt.axis('off')
                break
        Z = Z.reshape(xx.shape)
        plt.scatter(X[y.values==0,0], X[y.values==0,1], alpha=0.8,
label='YES', s=5, color='navy')
        plt.scatter(X[y.values==1,0], X[y.values==1,1], alpha=0.8,
label='NO', s=5, color='darkorange')
        plt.imshow(Z, interpolation='nearest', cmap='RdYlBu_r', alp
ha=0.15,
                   extent=(x_min, x_max, y_min, y_max), origin='low
er')
        plt.title(plot_type + '\n' +
                  str(fitted_model).split('(')[0]+ ' Test Accuracy:
' + str(np.round(fitted_model.score(X, y), 5)))
    plt.gca().set_aspect('equal');
    plt.tight_layout()
    plt.legend()
    plt.subplots_adjust(top=0.9, bottom=0.08, wspace=0.02)
```

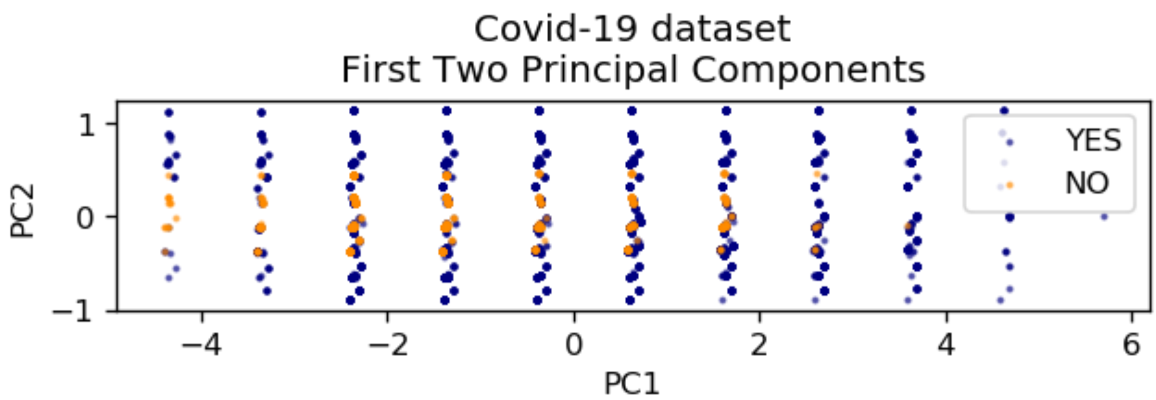
In [60]:
```
model = LogisticRegression()
model.fit(X_train1,y_train1)
plot_bank(X_test1, y_test1, model)
plt.show()
```



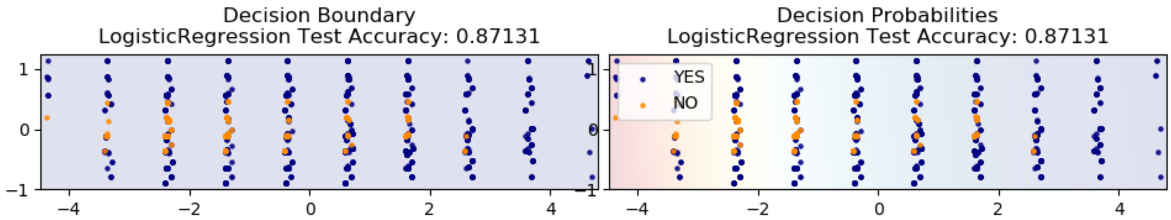
We can see that our data shows the problem caused by imbalanced data. we can barely see the blue points. So we will implement the method on balanced data set.

8.2 Visualize on balanced data

In [51]:
```
data1 = pd.DataFrame(data=x_resampled)
X = data1.iloc[:,:-1]
y = data1.iloc[:,-1]
pca = PCA(n_components=2).fit_transform(X)
X_train1, X_test1, y_train1, y_test1 = train_test_split(pca, y, ran
dom_state=0)

plt.figure(dpi=120)
plt.scatter(pca[y.values==0,0], pca[y.values==0,1], alpha=0.5, labe
l='YES', s=2, color='navy')
plt.scatter(pca[y.values==1,0], pca[y.values==1,1], alpha=0.5, labe
l='NO', s=2, color='darkorange')
plt.legend()
plt.title('Covid-19 dataset\nFirst Two Principal Components')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.gca().set_aspect('equal')
plt.show()
```



In [52]:
```
model = LogisticRegression()
model.fit(X_train1,y_train1)
plot_bank(X_test1, y_test1, model)
plt.show()
```

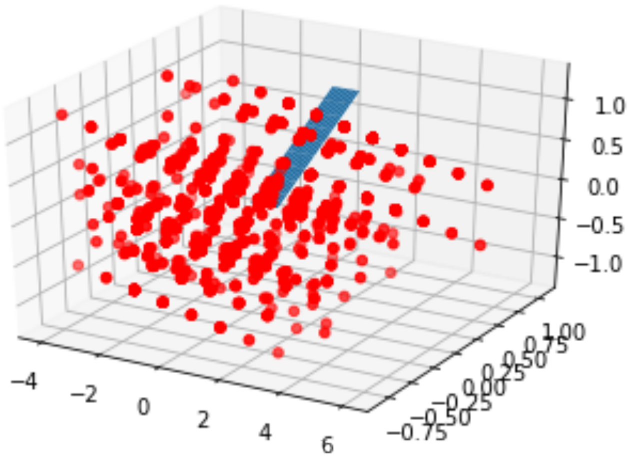


The plot on balanced data is much more better. While the PCA has reduced the accuracy of our Logistic Regression model.This is because we use PCA to reduce the amount of the dimension, which means we removed information from our data and the accuracy become lower.

9. Visualize SVM classifier

In [74]:
```
import numpy as np
import csv
from sklearn import svm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
n_Support_vector = svm_linear.n_support_
sv_idx = svm_linear.support_
w = svm_linear.coef_
b = svm_linear.intercept_

X = data.iloc[:,:-1]
y = data.iloc[:,-1]
pca2 = PCA(n_components=3).fit_transform(X)
X_train2, X_test2, y_train2, y_test2 = train_test_split(pca2, y, ra
ndom_state=0)
    # plot

ax = plt.subplot(111, projection='3d')
x = np.arange(0,1,0.1)
y = np.arange(0,1,0.1)
x, y = np.meshgrid(x, y)
z = (w[0,0]*x + w[0,1]*y + b) / (-w[0,2])
surf = ax.plot_surface(x, y, z, rstride=1, cstride=1)


x_array = np.array(X_train2, dtype=float)
y_array = np.array(y_train2, dtype=int)
pos = x_array[np.where(y_array==1)]
neg = x_array[np.where(y_array==-1)]
ax.scatter(pos[:,0], pos[:,1], pos[:,2], c='r', label='pos')
ax.scatter(neg[:,0], neg[:,1], neg[:,2], c='b', label='neg')
```

Out[74]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1a27539ac8>



10. Performance of the machine learning algorithms

In [53]:

```
df = pd.DataFrame({"Algorithms": ['Logistic Regression Method 1', 'Logistic Regression Method 1', 'SVM Method'],
                   "Oversampling method": ['SMOTE', 'SMOTE', 'SMOTE'],
                   "Area under ROC curve": [auc_6_3_1, auc_6_3_2, auc_7_4],
                   "Accuracy": [accuracy_6_3_1, accuracy_6_3_2, accuracy_7_4],
                   "Precision": [precision_6_3_1, precision_6_3_2, precision_7_4],
                   "Recall": [recall_6_3_1, recall_6_3_2, recall_7_4]})
df
```

Out[53]:

	Algorithms	Oversampling method	Area under ROC curve	Accuracy	Precision	Recall
0	Logistic Regression Method 1	SMOTE	0.936909	0.936731	0.953153	0.920000
1	Logistic Regression Method 1	SMOTE	0.936170	0.935852	0.964815	0.906087
2	SVM Method	SMOTE	0.936225	0.935852	0.970037	0.900870