```
In [1]:
# importing the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, roc_auc_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
```

```
In [2]:
# defining function for plotting correlation heatmap
def plot_heatmap(correlation, title):
    plt.figure(figsize=(15, 8))
    ax = sns.heatmap(correlation,annot=True,fmt='.3f',linewidths=0.3,annot_kws={"size"
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.title(title, fontsize=20)
    ax.figure.axes[-1].tick_params(labelsize=18) # To increase fontsize of colorbar t
    #lim = len(correlation.columns)
    #ax.set_ylim([0,lim]) # to make the map display correctly without trimming the edg
    plt.show()
    return
```

```
In [3]:
# importing the dataset
dataset = pd.read_csv('covid_train.csv')
```

```
In [4]:
dataset.head(5)
```

Out[4]:

	Age_Group	Client_Gender	Case_AcquisitionInfo	Reporting_PHU_City	Outbreak_Related	Reporting_PHU_Latit
0	50s	MALE	NO KNOWN EPI LINK	Oakville	NaN	43.413
1	20s	FEMALE	CC	Guelph	NaN	43.524
2	90s	FEMALE	OB	Barrie	Yes	44.410
3	20s	FEMALE	MISSING INFORMATION	Toronto	NaN	43.656
4	90s	FEMALE	OB	Ottawa	Yes	45.345

Data Cleaning

```
In [5]:
# checking for missing values in columns
for column in dataset.columns:
    print(column, "-", dataset[column].isna().sum())
```

```
Age_Group - 6
Client_Gender - 0
Case_AcquisitionInfo - 0
Reporting_PHU_City - 0
Outbreak_Related - 9020
Reporting_PHU_Latitude - 0
Reporting_PHU_Longitude - 0
Outcome1 - 0
```

```
In [6]:
# filling missing values in outbreak related with NO
dataset["Outbreak_Related"].fillna("No", inplace=True)

# drop missing columns in Age_Group
dataset.dropna(axis=0, inplace=True)
dataset.reset_index(drop=True, inplace=True)
```

```
In [7]:
# grouping features accourding to their data types
cats = ['Client_Gender','Case_AcquisitionInfo','Reporting_PHU_City','Outbreak_Related
nums = ['Reporting_PHU_Latitude', 'Reporting_PHU_Longitude']
ords = ['Age_Group']
target = ['Outcome1']
```

```
In [8]:
dataset['Age_Group'].replace(["<20", "20s", "30s", "40s", "50s", "60s", "70s", "80s",

# Label encoding the outcomes
dataset['Outcome1'].replace(["Resolved", "Not Resolved", "Fatal"],[0,1,2], inplace = T

dataset.head()
```

Out[8]:

	Age_Group	Client_Gender	Case_AcquisitionInfo	Reporting_PHU_City	Outbreak_Related	Reporting_PHU_Latit
0	4	MALE	NO KNOWN EPI LINK	Oakville	No	43.413
1	1	FEMALE	CC	Guelph	No	43.524
2	8	FEMALE	OB	Barrie	Yes	44.410
3	1	FEMALE	MISSING INFORMATION	Toronto	No	43.656
4	8	FEMALE	OB	Ottawa	Yes	45.345

```
In [9]:
# create a copy of dataset
data = dataset.copy()
```

Data Preparation

One Hot Encoding the categorical features

```
In [10]:
df = pd.get_dummies(dataset.iloc[:,:-1], columns = cats)
df["Outcome1"] = dataset["Outcome1"]
dataset = df

dataset.head()
```

Out[10]:

	Age_Group	Reporting_PHU_Latitude	Reporting_PHU_Longitude	Client_Gender_FEMALE	Client_Gender_GENDE DIVERS
0	4	43.413997	-79.744796	0	
1	1	43.524881	-80.233743	1	
2	8	44.410713	-79.686306	1	
3	1	43.656591	-79.379358	1	
4	8	45.345665	-75.763912	1	

5 rows × 50 columns

```
In [11]:
# Separating the dataset into matrix of features and target'
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:,-1].values
```

```
In [12]:
from sklearn.model_selection import train_test_split
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X, y, test_size=0.2, rand
```

Building the Base Model

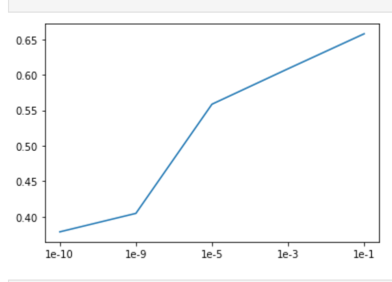
```
In [13]:
%%time
from sklearn.model_selection import GridSearchCV

# Fitting the Classification Model
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb_params = [{'var_smoothing':[1e-10, 1e-9, 1e-5, 1e-3, 1e-1]}]
nb_grid = GridSearchCV(nb, nb_params, cv=10)
nb_grid.fit(X_train_1, y_train_1)
nb_average_score = nb_grid.cv_results_['mean_test_score'].astype(float)
result = nb_grid.cv_results_
nb_grid.best_estimator_
```

```
Wall time: 678 ms
Out[13]: GaussianNB(var_smoothing=0.1)
```

```
In [14]:
nb_average_score
```

```
Out[14]: array([0.37875625, 0.4047738 , 0.55860423, 0.6085387 , 0.6577997 ])
```

```
In [15]:
plt.figure()
sns.lineplot(x=["1e-10", "1e-9", "1e-5", "1e-3", "1e-1"], y=nb_average_score)
plt.show()
```




```
In [16]:
%%time
# Building model with optimal parameters
nb_1 = GaussianNB(var_smoothing=0.1)
nb_1.fit(X_train_1, y_train_1)

print("Training Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_train_1, nb_1.predict(X_train_1)), 2)
print("F1_score: ", round(f1_score(y_train_1, nb_1.predict(X_train_1), average = 'weig
print("AUC: ", round(roc_auc_score(y_train_1, nb_1.predict_proba(X_train_1), average =
```

```
Training Set Evaluation
Accuracy:  65.82
F1_score:  0.64
AUC:  0.82
Wall time: 85 ms
```

```
In [17]:
# Evaluating the model on the test set
nb_pred = nb_1.predict(X_test_1)

print("Test Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_test_1, nb_pred), 2))
print("F1_score: ", round(f1_score(y_test_1, nb_pred, average = 'weighted'), 2))
print("AUC: ", round(roc_auc_score(y_test_1, nb_1.predict_proba(X_test_1), average = '
```

```
Test Set Evaluation
Accuracy:  65.44
F1_score:  0.64
AUC:  0.82
```

Notes About Smoothing Parameter

1e-1 is the best smoothing parameter with an accuracy of 65.76%. Accuracy increases almost linearly with smoothing variable.

The smoothing variable represents the portion of the largest variance of all features that is added to variances for calculation stability.

A Gaussian curve can serve as a "low pass" filter, allowing only the samples close to its mean to "pass." In the context of Naive Bayes, assuming a Gaussian distribution is essentially giving more weights to the samples closer to the distribution mean. This might filter out some values that we want to "pass".

The variable, var_smoothing, artificially adds a user-defined value to the distribution's variance (whose default value is derived from the training data set). This essentially widens (or "smooths") the curve and accounts for more samples that are further away from the distribution mean.

Tuning this parameter, will also modify the variance in a way that will give the best accuracy.

Improving Model Computation Time

Method 1: Dropping the Reporting_PHU_City column

The Reporting_PHU_City column is a categorical column with 34 distinct values this means that after one-hot encoding, it adds 34 columns to the data thus, making it computationally expensive.

Considering its low correlation with the outcome, we will deal with this by removing the feature from the matrix of features. This reduces the columns in the matrix of features from 49 to 15

```
In [18]:
# removing the Reporting_PHU_City column
dataset2 =  data[['Age_Group', 'Client_Gender', 'Case_AcquisitionInfo',
                  'Outbreak_Related', 'Reporting_PHU_Latitude',
                 'Reporting_PHU_Longitude', 'Outcome1']]

df = pd.get_dummies(dataset2.iloc[:,:-1], columns = ['Client_Gender','Case_Acquisition
df["Outcome1"] = dataset2["Outcome1"]
dataset2 = df

dataset2.head()
```

Out[18]:

	Age_Group	Reporting_PHU_Latitude	Reporting_PHU_Longitude	Client_Gender_FEMALE	Client_Gender_GENDE DIVERS
0	4	43.413997	-79.744796	0	
1	1	43.524881	-80.233743	1	
2	8	44.410713	-79.686306	1	
3	1	43.656591	-79.379358	1	
4	8	45.345665	-75.763912	1	

```
In [19]:
# Separating the dataset into matrix of features and target'
X = dataset2.iloc[:,:-1].values
y = dataset2.iloc[:,-1].values
```

```
In [20]:
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X, y, test_size=0.2, rando
```

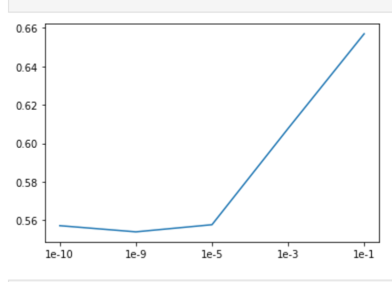
```
In [21]:
%%time

# Fitting the Classification Model
nb_grid.fit(X_train_2, y_train_2)
nb_average_score = nb_grid.cv_results_['mean_test_score'].astype(float)
result = nb_grid.cv_results_
nb_grid.best_estimator_
```

```
Wall time: 237 ms
Out[21]: GaussianNB(var_smoothing=0.1)
```

```
In [22]:
nb_average_score
```

```
Out[22]: array([0.55717183, 0.55397254, 0.55767915, 0.60769759, 0.6569571 ])
```

```
In [23]:
plt.figure()
sns.lineplot(x=["1e-10", "1e-9", "1e-5", "1e-3", "1e-1"], y=nb_average_score)
plt.show()
```




```
In [24]:
%%time
# Building model with optimal parameters
nb_2 = GaussianNB(var_smoothing=0.1)
nb_2.fit(X_train_2, y_train_2)

print("Training Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_train_2, nb_2.predict(X_train_2)), 2)
print("F1_score: ", round(f1_score(y_train_2, nb_2.predict(X_train_2), average = 'weig
print("AUC: ", round(roc_auc_score(y_train_2, nb_2.predict_proba(X_train_2), average =
```

```
Training Set Evaluation
Accuracy:  65.68
F1_score:  0.64
AUC:  0.82
Wall time: 43 ms
```

```
In [25]:
# Evaluating the model on the test set
nb_pred = nb_2.predict(X_test_2)

print("Test Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_test_2, nb_pred), 2))
print("F1_score: ", round(f1_score(y_test_2, nb_pred, average = 'weighted'), 2))
print("AUC: ", round(roc_auc_score(y_test_2, nb_2.predict_proba(X_test_2), average = '
```

```
Test Set Evaluation
Accuracy:  65.34
F1_score:  0.63
AUC:  0.82
```

Method 2: Aggregating categories in the columns

In the Client_Gender feature, UNSPECIFIED and GENDER DIVERSE have very low counts compared to the rest and they can both be grouped into one category OTHERS.

In the Case_AcquisitionInfo feature, TRAVEL and UNSPECIFIED EPI LINK have very low counts compared to the rest and they can both be grouped into one category OTHERS.

In the Reporting_PHU_City feature, there are 34 distince categories. Rather than dropping the entire feature, the categories can be grouped by their counts. City_Class1 = > 500 City_Class2 = 75 to 500 City_Class3 = < 75

```
In [26]:
# inspecting the unique values in categorical columns and their frequencies
dataset3 = data.copy()
```

```
In [27]:
dataset3["Client_Gender"].replace(['UNSPECIFIED', 'GENDER DIVERSE'], 'OTHER', inplace
dataset3["Case_AcquisitionInfo"].replace(['TRAVEL', 'UNSPECIFIED EPI LINK'], 'OTHER',
```

```
In [28]:
City_Class1 = list((dataset3['Reporting_PHU_City'].value_counts()[dataset3['Reporting_
```

```
In [29]:
City_Class2 = list((dataset3['Reporting_PHU_City'].value_counts()[(dataset3['Reporting
```

```
In [30]:
City_Class3 = list((dataset3['Reporting_PHU_City'].value_counts()[dataset3['Reporting_
```

```
In [31]:
dataset3["Reporting_PHU_City"].replace(City_Class1, 1, inplace = True)
dataset3["Reporting_PHU_City"].replace(City_Class2, 2, inplace = True)
dataset3["Reporting_PHU_City"].replace(City_Class3, 3, inplace = True)
```

```
In [32]:
df = pd.get_dummies(dataset3.iloc[:,:-1], drop_first=True, columns = cats)
df["Outcome1"] = dataset3["Outcome1"]
dataset3 = df

dataset3.head()
```

Out[32]:

	Age_Group	Reporting_PHU_Latitude	Reporting_PHU_Longitude	Client_Gender_MALE	Client_Gender_OTHER
0	4	43.413997	-79.744796	1	0
1	1	43.524881	-80.233743	0	0
2	8	44.410713	-79.686306	0	0
3	1	43.656591	-79.379358	0	0
4	8	45.345665	-75.763912	0	0

```
In [33]:
# Separating the dataset into matrix of features and target'
X = dataset3.iloc[:,:-1].values
y = dataset3.iloc[:,-1].values
```

```
In [34]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

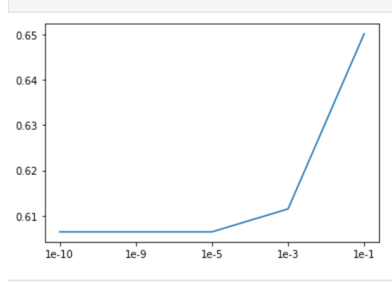
```
In [35]:
%%time

# Fitting the Classification Model
nb_grid.fit(X_train, y_train)
nb_average_score = nb_grid.cv_results_['mean_test_score'].astype(float)
result = nb_grid.cv_results_
nb_grid.best_estimator_
```

```
Wall time: 311 ms
Out[35]: GaussianNB(var_smoothing=0.1)
```

```
In [36]:
nb_average_score
```

```
Out[36]: array([0.60643595, 0.60643595, 0.60643588, 0.61148809, 0.65013757])
```

```
In [37]:
plt.figure()
sns.lineplot(x=["1e-10", "1e-9", "1e-5", "1e-3", "1e-1"], y=nb_average_score)
plt.show()
```




```
In [38]:
%%time
# Building model with optimal parameters
nb = GaussianNB(var_smoothing=0.1)
nb.fit(X_train, y_train)

print("Training Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_train, nb.predict(X_train)), 2))
print("F1_score: ", round(f1_score(y_train, nb.predict(X_train), average = 'weighted')
print("AUC: ", round(roc_auc_score(y_train, nb.predict_proba(X_train), average = "macr
```

```
Training Set Evaluation
Accuracy:  64.9
F1_score:  0.62
AUC:  0.82
Wall time: 43 ms
```

```
In [39]:
# Evaluating the model on the test set
nb_pred = nb.predict(X_test)

print("Test Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_test, nb_pred), 2))
print("F1_score: ", round(f1_score(y_test, nb_pred, average = 'weighted'), 2))
print("AUC: ", round(roc_auc_score(y_test, nb.predict_proba(X_test), average = "macro'
```

```
Test Set Evaluation
Accuracy:  64.8
F1_score:  0.62
AUC:  0.82
```

Impact of Feature Selection and Engineering

Using all features

	Accuracy	F1 Score	AUC
Training Set	65.82	0.64	0.82
Test Set	65.44	0.64	0.82

Dropping the City column

	Accuracy	F1 Score	AUC
Training Set	65.68	0.64	0.82
Test Set	65.34	0.63	0.82

Aggregating the categories of the features

	Accuracy	F1 Score	AUC
Training Set	64.9	0.62	0.82
Test Set	64.8	0.62	0.82

Observations

There in no much difference in performance between adding the city feature and dropping the city feature. So we can drop the city feature to improve computation performance.

Improving the model by feature scaling

Standard Scaler

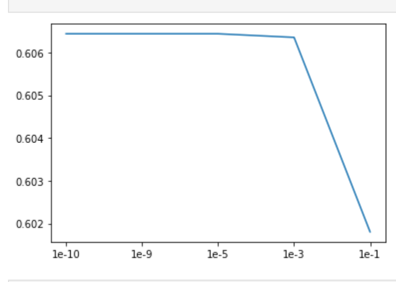
```
In [40]:
X_scaler = StandardScaler()
X_SS = X_scaler.fit_transform(X_train)
X_SS_test = X_scaler.transform(X_test)
nb_grid.fit(X_SS, y_train)
nb_average_score = nb_grid.cv_results_['mean_test_score'].astype(float)
result = nb_grid.cv_results_
nb_grid.best_estimator_
```

```
Out[40]: GaussianNB(var_smoothing=1e-10)
```

```
In [41]:
nb_average_score
```

```
Out[41]: array([0.60643595, 0.60643595, 0.60643595, 0.60635178, 0.60180512])
```

```
In [42]:
plt.figure()
sns.lineplot(x=["1e-10", "1e-9", "1e-5", "1e-3", "1e-1"], y=nb_average_score)
plt.show()
```




```
In [43]:
%%time
# Building model with optimal parameters
nb = GaussianNB(var_smoothing=1e-10)
nb.fit(X_SS, y_train)

print("Training Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_train, nb.predict(X_SS)), 2))
```

```
Training Set Evaluation
Accuracy:  60.99
Wall time: 12 ms
```

```
In [44]:
# Evaluating the model on the test set
nb_pred = nb.predict(X_SS_test)
accuracy_1 = round(100 * accuracy_score(y_test, nb_pred), 2)
print("Test Set Evaluation")
print("Accuracy: ", accuracy_1)
```

```
Test Set Evaluation
Accuracy:  59.99
```

MinMax Scaler

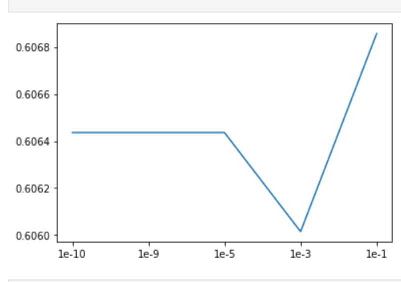
```
In [45]:
X_scaler = MinMaxScaler()
X_MMS = X_scaler.fit_transform(X_train)
X_MMS_test = X_scaler.transform(X_test)
nb_grid.fit(X_MMS, y_train)
nb_average_score = nb_grid.cv_results_['mean_test_score'].astype(float)
result = nb_grid.cv_results_
nb_grid.best_estimator_
```

```
Out[45]: GaussianNB(var_smoothing=0.1)
```

```
In [46]:
nb_average_score
```

```
Out[46]: array([0.60643595, 0.60643595, 0.60643595, 0.60601494, 0.60685704])
```

```
In [47]:
plt.figure()
sns.lineplot(x=["1e-10", "1e-9", "1e-5", "1e-3", "1e-1"], y=nb_average_score)
plt.show()
```




```
In [48]:
%%time
# Building model with optimal parameters
nb = GaussianNB(var_smoothing=0.1)
nb.fit(X_MMS, y_train)

print("Training Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_train, nb.predict(X_MMS)), 2))
```

```
Training Set Evaluation
Accuracy:  61.04
Wall time: 12 ms
```

```
In [49]:
# Evaluating the model on the test set
nb_pred = nb.predict(X_MMS_test)
accuracy_1 = round(100 * accuracy_score(y_test, nb_pred), 2)
print("Test Set Evaluation")
print("Accuracy: ", accuracy_1)
```

```
Test Set Evaluation
Accuracy:  60.19
```

Robust Scaler

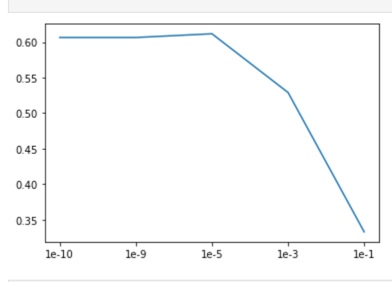
```
In [50]:
X_scaler = RobustScaler()
X_RS = X_scaler.fit_transform(X_train)
X_RS_test = X_scaler.transform(X_test)
nb_grid.fit(X_RS, y_train)
nb_average_score = nb_grid.cv_results_['mean_test_score'].astype(float)
result = nb_grid.cv_results_
nb_grid.best_estimator_
```

```
Out[50]: GaussianNB(var_smoothing=1e-05)
```

```
In [51]:
nb_average_score
```

```
Out[51]: array([0.60643595, 0.60643595, 0.61165644, 0.52905352, 0.33310882])
```

```
In [52]:
plt.figure()
sns.lineplot(x=["1e-10", "1e-9", "1e-5", "1e-3", "1e-1"], y=nb_average_score)
plt.show()
```




```
In [53]:
%%time
# Building model with optimal parameters
nb = GaussianNB(var_smoothing=1e-05)
nb.fit(X_RS, y_train)

print("Training Set Evaluation")
print("Accuracy: ", round(100 * accuracy_score(y_train, nb.predict(X_RS)), 2))
```

```
Training Set Evaluation
Accuracy:  61.27
Wall time: 18 ms
```

```
In [54]:
# Evaluating the model on the test set
nb_pred = nb.predict(X_RS_test)
accuracy_1 = round(100 * accuracy_score(y_test, nb_pred), 2)
print("Test Set Evaluation")
print("Accuracy: ", accuracy_1)
```

```
Test Set Evaluation
Accuracy:  60.59
```

The Naive Bayes classifier performs better without feature scalling

Notes About Feature Scaling on Naive Bayes

Effects of Scaling on Training Set and Test set accuracy

Scaling Method	Train Accuracy	Test Accuracy
Standard Scaler	60.99	59.99
MinMax Scaler	61.04	60.19
Robust Scaler	61.27	60.59
No Scaler	64.9	64.8

The Naive Bayes classifier performs better without feature scaling. Naive Bayes is not very sensitive to distance between values like KNN, so we did not expect much improvement from Feature Scaling.

```
In [ ]:
```