Covid Analysis and Predictions

May 27, 2020

[1]:
```
%config IPCompleter.greedy=True

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1 Final Introduction to AI course : COVID-19 Analysis and Predictions

1.1 Introduction: Dataset

Authors of the dataset: Xu, Bo and Gutierrez, Bernardo and Mekaru, Sumiko and Sewalk, Kara and Goodwin, Lauren and Loskill, Alyssa and Cohn, Emily and Hswen, Yulin and Hill, Sarah C. and Cobo, Maria M and Zarebski, Alexander and Li, Sabrina and Wu, Chieh-Hsi and Hulland, Erin and Morgan, Julia and Wang, Lin and O'Brien, Katelynn and Scarpino, Samuel V. and Brownstein, John S. and Pybus, Oliver G. and Pigott, David M. and Kraemer, Moritz U. G.

Article about the dataset: Epidemiological data from the COVID-19 outbreak, real-time case information}

Github : https://github.com/beoutbreakprepared/nCoV2019

The dataset used is: /latest_data/latestdata.tar.gz (as of May 25th 2020)

[2]:
```
#path = ('https://raw.githubusercontent.com/beoutbreakprepared/nCoV2019/master/
 ↪outside_hubei_20200301.csv')
path = "latestdata.csv"

df = pd.read_csv(path)
df.head()
```

```
/usr/local/anaconda3/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3057: DtypeWarning: Columns
(1,2,9,10,12,13,14,15,16,17,19,21,22,23,24,25,26,27,31,32) have mixed types.
Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[2]:
```
          ID  age   sex         city   province  country   latitude  \
0    000-1-1  NaN  male     Shek Lei  Hong Kong    China  22.365019
1   000-1-10   78  male  Vo Euganeo     Veneto    Italy  45.297748
```

```
2    000-1-100   61  female             NaN        NaN  Singapore   1.353460
3   000-1-1000  NaN     NaN  Zhengzhou City      Henan      China  34.629310
4  000-1-10000  NaN     NaN  Pingxiang City    Jiangxi      China  27.513560

    longitude geo_resolution date_onset_symptoms  ... date_death_or_discharge  \
0  114.133808          point                 NaN  ...                     NaN
1   11.658382          point                 NaN  ...              22.02.2020
2  103.815100         admin0                 NaN  ...              17.02.2020
3  113.468000         admin2                 NaN  ...                     NaN
4  113.902900         admin2                 NaN  ...                     NaN

  notes_for_discussion     location admin3          admin2     admin1  \
0                  NaN     Shek Lei    NaN             NaN  Hong Kong
1                  NaN  Vo' Euganeo    NaN             NaN     Veneto
2                  NaN          NaN    NaN             NaN        NaN
3                  NaN          NaN    NaN  Zhengzhou City      Henan
4                  NaN          NaN    NaN  Pingxiang City    Jiangxi

  country_new admin_id  data_moderator_initials travel_history_binary
0       China   8051.0                      NaN                   NaN
1       Italy   8978.0                      NaN                   NaN
2   Singapore    201.0                      NaN                   NaN
3       China  10115.0                      NaN                   NaN
4       China   7079.0                      NaN                   NaN

[5 rows x 33 columns]
```

[3]:
```
df.columns
```

[3]:
```
Index(['ID', 'age', 'sex', 'city', 'province', 'country', 'latitude',
       'longitude', 'geo_resolution', 'date_onset_symptoms',
       'date_admission_hospital', 'date_confirmation', 'symptoms',
       'lives_in_Wuhan', 'travel_history_dates', 'travel_history_location',
       'reported_market_exposure', 'additional_information',
       'chronic_disease_binary', 'chronic_disease', 'source',
       'sequence_available', 'outcome', 'date_death_or_discharge',
       'notes_for_discussion', 'location', 'admin3', 'admin2', 'admin1',
       'country_new', 'admin_id', 'data_moderator_initials',
       'travel_history_binary'],
      dtype='object')
```

[4]:
```
df.describe(include="all")
```

[4]:
```
                 ID     age     sex    city province country       latitude  \
count        920737  243077  243783  716290   889612  920634  920688.000000
unique       920737     304       2    4614      950     141            NaN
top       003-29520   35-59  female  Moscow  Central  Russia            NaN
freq              1   66683  131809  104060   140612  198301            NaN
mean            NaN     NaN     NaN     NaN      NaN     NaN      44.270574
```

```
std           NaN     NaN     NaN     NaN      NaN     NaN      15.467287
min           NaN     NaN     NaN     NaN      NaN     NaN     -54.000000
25%           NaN     NaN     NaN     NaN      NaN     NaN      41.402211
50%           NaN     NaN     NaN     NaN      NaN     NaN      48.076205
75%           NaN     NaN     NaN     NaN      NaN     NaN      52.580000
max           NaN     NaN     NaN     NaN      NaN     NaN      70.071800

            longitude geo_resolution date_onset_symptoms  ...  \
count   920688.000000         920688              164774  ...
unique            NaN              7                 137  ...
top               NaN         admin2          20.03.2020  ...
freq              NaN         434954                5302  ...
mean         9.667676            NaN                 NaN  ...
std         49.728425            NaN                 NaN  ...
min       -159.727596            NaN                 NaN  ...
25%          4.590656            NaN                 NaN  ...
50%         10.552910            NaN                 NaN  ...
75%         37.617300            NaN                 NaN  ...
max        174.740000            NaN                 NaN  ...

       date_death_or_discharge          notes_for_discussion location  \
count                      522                           642     7614
unique                      78                           204      347
top                 18.02.2020  Could be some cases from 23rd  Chicago
freq                        22                            91      985
mean                       NaN                           NaN      NaN
std                        NaN                           NaN      NaN
min                        NaN                           NaN      NaN
25%                        NaN                           NaN      NaN
50%                        NaN                           NaN      NaN
75%                        NaN                           NaN      NaN
max                        NaN                           NaN      NaN

            admin3  admin2   admin1 country_new       admin_id  \
count         9207  426434   589542      895125  920688.000000
unique         410    1961      469         137            NaN
top     Birmingham  Moscow  Central      Russia            NaN
freq           309  104058   136936      198301            NaN
mean           NaN     NaN      NaN         NaN    6571.943791
std            NaN     NaN      NaN         NaN    4131.611894
min            NaN     NaN      NaN         NaN       1.000000
25%            NaN     NaN      NaN         NaN    1903.500000
50%            NaN     NaN      NaN         NaN    6363.000000
75%            NaN     NaN      NaN         NaN   10857.000000
max            NaN     NaN      NaN         NaN   11910.000000

       data_moderator_initials travel_history_binary
```

```
count                  402183             855158
unique                     11                  2
top                        TR              False
freq                   386327             828363
mean                      NaN                NaN
std                       NaN                NaN
min                       NaN                NaN
25%                       NaN                NaN
50%                       NaN                NaN
75%                       NaN                NaN
max                       NaN                NaN

[11 rows x 33 columns]
```

2 0. Cleaning Dataset

2.1 Age Range to average age conversion

Some of the ages are actually an age range. The goal here if to convert some age ranges to an average age.

ex: 10-20 => 15

[5]:
```
df = df[df["age"] != "7 months"]

tmp = []

for index,value in enumerate(df["age"]):
    try:
        if (type(value) == str):
            # some age values may be a range like : "12-20"
            age_array = value.split("-")
            if len(age_array) == 2:
                tmp.append((float(age_array[0]) + float(age_array[1])) // 2)
            else:
                tmp.append(float(value))
        else:
            tmp.append(float(value))
    except Exception:
        tmp.append(np.NaN)
        continue

ages_transformed = pd.Series(tmp)
```

[6]:
```
df["age"] = ages_transformed
```

2.2 Outcome standarzing

The outcome types are of types : - 'death' - 'discharge' - 'discharged' - 'Discharged' - 'recovered', - 'dead' - 'died' - 'Died'

If the person died the value will be *1* and will be *0* if recovered/dismissed.

[7]:
```
tmp = []

for value in df["outcome"]:
    if type(value) == str:
        lowered_value = value.lower()
        if lowered_value in ["dead", "died","death"]:
            tmp.append(1)
        else:
            tmp.append(0)
    else:
        tmp.append(0)

outcome_standardized = pd.Series(tmp)
```

[8]:
```
df["outcome"] = outcome_standardized
```

2.3 Droping null and NaN values

[9]:
```
df = df.dropna(subset=["age", "sex","city", "outcome", "country", "province"])
```

[10]:
```
df.shape
```

[10]:
```
(166196, 33)
```

2.4 Country, City, Province Standardizing

Countries are string values, for modeling and analysis we will associate the variables to labels

[11]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(df["country"])

df["country_code"] = le.transform(df["country"])
```

[12]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(df["province"])

df["province_code"] = le.transform(df["province"])
```

[13]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
```

```
le.fit(df["city"])

df["city_code"] = le.transform(df["city"])
```

2.5 Standardizing Sex

Male will be considered 1 and female will be considered 0.

[14]:
```
df["sex"].unique()
```

[14]:
```
array(['male', 'female'], dtype=object)
```

[15]:
```
df["sex"].isna().sum()
```

[15]:
```
0
```

[16]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(df["sex"])

df["sex_code"] = le.transform(df["sex"])
```

2.6 Cleaning Age

Ages need to be between 1 and 130 years old and be labelled a category

[17]:
```
df = df[(df["age"] <= 130) & (df["age"] > 0)]
```

[18]:
```
age_groups = pd.cut(df["age"],bins=[0,2,17,65,130],labels=['Toddler/
↪Baby','Child','Adult','Elderly'])
df["age_group"] = age_groups
```

[19]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(df["age_group"])

df["age_group_code"] = le.transform(df["age_group"])
```

3 1. Analysis of the Dataset

Number of people in "cleaned" dataset

[20]:
```
df.shape[0]
```

[20]:
```
166184
```

Total number of deaths in dataset

[21]:
```
df["outcome"].sum()
```

[21]:
```
183.0
```

Number of deaths per age group and sex

[22]:
```
df.pivot_table("outcome", index="sex", columns="age_group", aggfunc="sum")
```

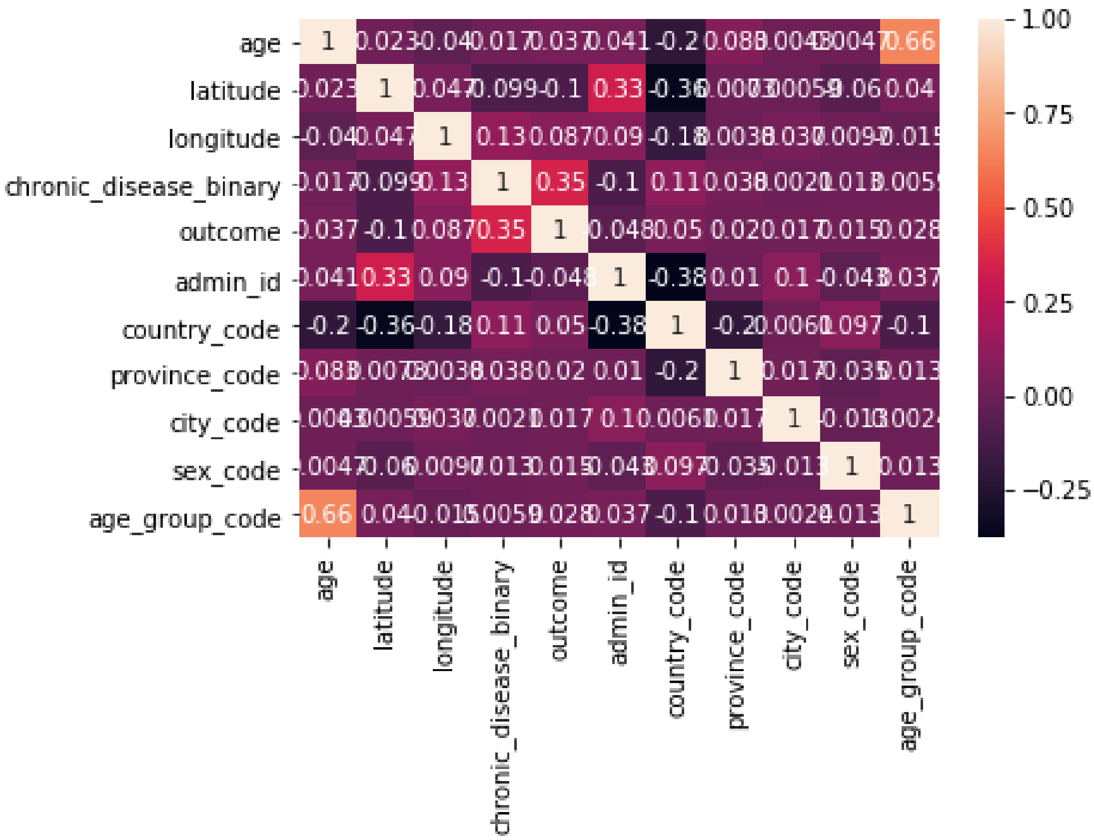
[22]:
```
age_group  Toddler/Baby  Child  Adult  Elderly
sex
female              0.0    0.0   10.0     47.0
male                0.0    0.0   55.0     71.0
```

Correlation matrix

3.1 A. Variable correlation

[23]:
```
import seaborn as sn

corrMatrix = df.corr()
sn.heatmap(corrMatrix, annot=True)
plt.show()
```



The most correlated variables to the outcome is the chronic_disease feature.

This correlation matrix doesn't show any other notable correlation between the outcome and another feature.

3.2 B. Plotting Cleaned Dataset using PCA

[24]:
```
df_trimmed = df.drop(columns=["age_group", "country",␣
 ↪"date_onset_symptoms","city", "province",
                     ␣
 ↪"date_death_or_discharge","notes_for_discussion","location","notes_for_discussion",
                      "ID", "geo_resolution", "date_onset_symptoms",␣
 ↪"data_moderator_initials","travel_history_dates","date_confirmation",␣
 ↪"travel_history_binary",
                     "country_new", "city","chronic_disease",␣
 ↪"additional_information","source",␣
 ↪"sequence_available","reported_market_exposure","date_admission_hospital",␣
 ↪"symptoms", "travel_history_location",
                     "lives_in_Wuhan", "admin3", "admin2",␣
 ↪"sex","admin1","admin_id", "latitude","province_code",␣
 ↪"age_group_code","longitude",
                              "country_code","city_code"])

df_trimmed.head()
```

[24]:

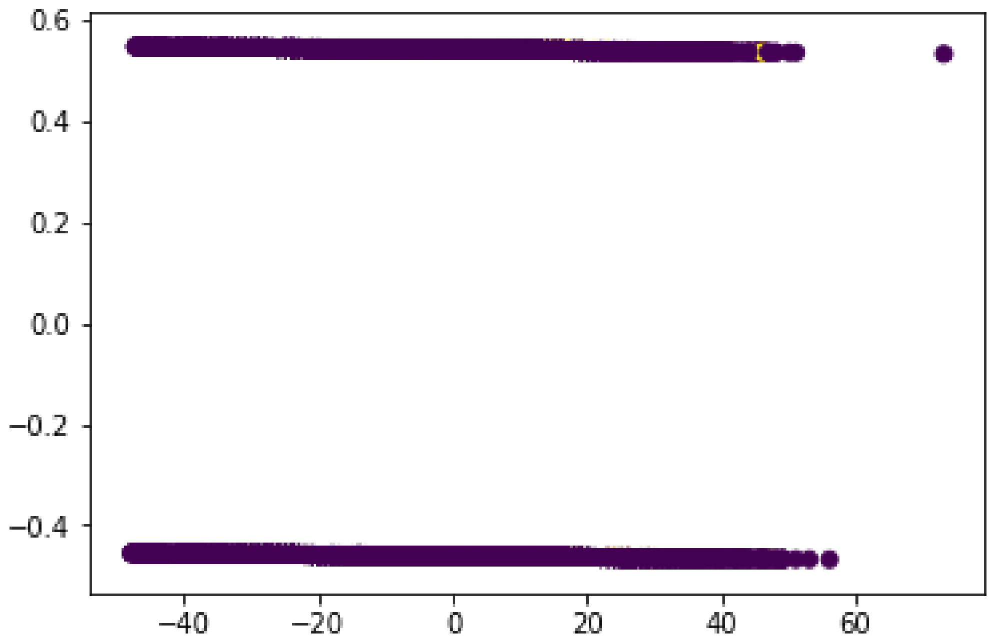
	age	chronic_disease_binary	outcome	sex_code
1	78.0	False	1.0	1
27	66.0	False	0.0	1
28	27.0	False	0.0	0
29	17.0	False	0.0	1
30	51.0	False	0.0	0

[25]:
```
from sklearn import decomposition

Y = df_trimmed["outcome"].values
X = df_trimmed.drop(columns=["outcome"]).values # Droping useless columns (for␣
 ↪this PCA)

pca = decomposition.PCA(n_components=2)
```

[26]:
```
pca.fit(X)
X_pca = pca.transform(X)
plt.figure()
plt.scatter(X_pca[:,0], X_pca[:,1], c=Y)
plt.show()
```



4 2. Bayes Nets

Bayes Theorem

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

4.1 A. $P(HasSymptoms|VisitedWuhan)$

Question: What is the probability for a person to have symptoms of COVID-19 (symptom_onset=date) if this person visited Wuhan (visiting Wuhan = 1) ? Consider that (symptom_onset=N/A) means that the patient is asymptomatic.

We are trying to solve:

$$P(HasSymptoms|VisitedWuhan) = \frac{P(HasSymptoms \cap VisitedWuhan)}{P(VisitedWuhan)}$$

$P(VisitedWuhan)$: the person visited Wuhan if the *travel_history_location* is *Wuhan*.

[27]:
```
total_visited_wuhan = df[df["travel_history_location"] ==␣
 ↪"Wuhan"]["travel_history_location"].count()
total_history_location = df["travel_history_location"].count() # counting only␣
 ↪when the data is about the location is present

p_VisitedWuhan = total_visited_wuhan / total_history_location
```

```
p_VisitedWuhan
```

[27]: 0.18578898800369117

$P(HasSymptoms \cap VisitedWuhan)$: the person visited Wuhan if the travel_history_location and is symptomatic

[28]:
```
total = df["date_onset_symptoms"].size
total
```

[28]: 166184

[29]:
```
df["date_onset_symptoms"].isnull().sum()
```

[29]: 56956

[30]:
```
total_has_symptoms_and_visited_wuhan = df[(df["date_onset_symptoms"].isnull()␣
 ↪== False)
                                          & (df["travel_history_location"] ==␣
 ↪"Wuhan")]["travel_history_location"].count()
total_has_symptoms_and_visited_wuhan
```

[30]: 337

[31]:
```
p_HasSymptomsAndVisitedWuhan = total_has_symptoms_and_visited_wuhan/total
p_HasSymptomsAndVisitedWuhan
```

[31]: 0.002027872719395369

$P(HasSymptoms|VisitedWuhan)$ that we can compute from the previous probabilities

[32]:
```
p_SymptomsKnowingThatVisitedWuhan = p_HasSymptomsAndVisitedWuhan  /␣
 ↪p_VisitedWuhan
p_SymptomsKnowingThatVisitedWuhan
```

[32]: 0.010914924189990636

To conclude there is roughly 1.1% chance that a person who visited Wuhan has the symptoms.

4.2 B. $P(HasSymptoms \cap VisitedWuhan)$

What is the probability for a person to be a true patient if this person have symptoms of COVID-19 (symptom_onset=date) and this person visited Wuhan ?

[33]:
```
p_HasSymptomsAndVisitedWuhan
```

[33]: 0.002027872719395369

There is roughly a 0.2% chance that the person has the symptoms and visited Wuhan

4.3 C. $P(Died|VisitedWuhan)$

What is the probability for a person to death if this person visited Wuhan ?

We are trying to solve:

$$P(Died|VisitedWuhan) = \frac{P(Died \cap VisitedWuhan)}{P(VisitedWuhan)}$$

$P(Died \cap VisitedWuhan)$: People who died (outcome = 1.0) and who visited Wuhan.

[34]:
```python
total_died_and_visited_wuhan = df[(df["travel_history_location"] == "Wuhan") &␣
↪(df["outcome"] == 1.0)]["travel_history_location"].count()
p_DiedAndVisitedWuhan = total_died_and_visited_wuhan/total_history_location
p_DiedAndVisitedWuhan
```

[34]:
```
0.0009227929867733005
```

$P(Died|VisitedWuhan)$: finally we can compute the probability of a person for death if visiting Wuhan.

[35]:
```python
p_DiedKnowingThatVisitedWuhan = p_DiedAndVisitedWuhan / p_VisitedWuhan
p_DiedKnowingThatVisitedWuhan*100
```

[35]:
```
0.49668874172185434
```

To conclude there is a 0.1% chance that a person dies after visitign Wuhan.

4.4 D. Average Recovery Interval for a Person visiting Wuhan

We only keep the people who visited wuhan and recovered from the COVID (outcome = 0.0 and who actually have dates relating to the times of their sickness

[36]:
```python
df_visiting_wuhan_not_dead = df.
↪dropna(subset=["date_onset_symptoms","date_death_or_discharge"])
df_visiting_wuhan_not_dead =␣
↪df_visiting_wuhan_not_dead[(df_visiting_wuhan_not_dead["travel_history_location"]␣
↪== "Wuhan") & (df_visiting_wuhan_not_dead["outcome"] == 0.0)]
```

We'll assume $[dateOnsetSymptoms, dateDeathOrDischarge]$ as the interval during which the patient is sick.

[37]:
```python
df_visiting_wuhan_not_dead
```

[37]:
```
                ID   age     sex                      city  \
3223     000-1-129  44.0  female             Wushan County
3335     000-1-130  39.0  female  Xishuangbanna Prefecture
4334     000-1-139  54.0    male                   Toronto
5223     000-1-147  38.0  female                    Manila
8112     000-1-173  30.0  female                     Paris
8223     000-1-174  31.0    male                     Paris
8667     000-1-178  24.0  female                    London
122905    002-3379  38.0  female                    Manila
124235     002-368  39.0  female  Xishuangbanna Prefecture
127257      002-64  44.0  female             Wushan County
305301   005-47764  71.0  female                    London

                          province      country   latitude   longitude  \
3223                     Chongqing        China  31.117740  109.896200
3335                        Yunnan        China  21.931390  100.947700
```

```
4334                          Ontario       Canada  43.725290  -79.387000
5223    National Capital Region (NCR)  Philippines  14.595800  120.977200
8112                    Ile-de-France       France  48.856660    2.342325
8223                    Ile-de-France       France  48.856660    2.342325
8667                          Ontario       Canada  42.983611  -81.249722
122905  National Capital Region (NCR)  Philippines  14.595800  120.977200
124235                         Yunnan        China  21.931390  100.947700
127257                      Chongqing        China  31.117740  109.896200
305301                        Ontario       Canada  42.983611  -81.249722

       geo_resolution date_onset_symptoms  ...  country_new admin_id  \
3223           admin3          15.01.2020  ...        China   9426.0
3335           admin2          18.01.2020  ...        China   9649.0
4334           admin2          22.01.2020  ...       Canada   8739.0
5223            point          25.01.2020  ...  Philippines   5933.0
8112           admin2          23.01.2020  ...       France   6904.0
8223           admin2          19.01.2020  ...       France   6904.0
8667            point          24.01.2020  ...       Canada   5660.0
122905          point          25.01.2020  ...  Philippines   5933.0
124235         admin2          18.01.2020  ...        China   9649.0
127257         admin3          15.01.2020  ...        China   9426.0
305301          point          24.01.2020  ...       Canada   5660.0

       data_moderator_initials travel_history_binary country_code  \
3223                       NaN                   NaN           10
3335                       NaN                   NaN           10
4334                       NaN                   NaN            8
5223                       NaN                   NaN           32
8112                       NaN                   NaN           16
8223                       NaN                   NaN           16
8667                       NaN                   NaN            8
122905                     NaN                  True           32
124235                     NaN                  True           10
127257                     NaN                  True           10
305301                     NaN                 False            8

       province_code city_code sex_code  age_group age_group_code
3223              77      1838        0      Adult              0
3335             390      1851        0      Adult              0
4334             272      1703        1      Adult              0
5223             242      1052        0      Adult              0
8112             157      1293        0      Adult              0
8223             157      1293        1      Adult              0
8667             272       981        0      Adult              0
122905           242      1052        0      Adult              0
124235           390      1851        0      Adult              0
127257            77      1838        0      Adult              0
```

```
305301             272        981         0     Elderly              2

[11 rows x 39 columns]
```

To facilitate the computation of average recovery time, each date will be computed into seconds.

[38]:
```python
from datetime import datetime

computed_row = []

for row in df_visiting_wuhan_not_dead.T.to_dict().values():
    start_date = datetime.strptime(row["date_onset_symptoms"],"%d.%m.%Y").
 ↪strftime("%s")
    end_date = datetime.strptime(row["date_death_or_discharge"],"%d.%m.%Y").
 ↪strftime("%s")
    recovery_time = int(end_date) - int(start_date)
    computed_row.append(recovery_time)

df_visiting_wuhan_not_dead["recovery_time"] = computed_row
```

[39]:
```python
average_recovery_seconds = int(df_visiting_wuhan_not_dead["recovery_time"].
 ↪mean())
```

[40]:
```python
from datetime import timedelta

average_datetime = timedelta(seconds=average_recovery_seconds)
average_datetime
```

[40]:
```
datetime.timedelta(days=17, seconds=47127)
```

The Average recovery interval a patient who visited Wuhan is 17 days.

5 3. Machine Learning

[41]:
```python
to_drop = ['ID','city', 'province', 'country', 'latitude',
           'longitude', 'geo_resolution','reported_market_exposure',
           'additional_information','notes_for_discussion',␣
 ↪'data_moderator_initials',
          ␣
 ↪'date_onset_symptoms','date_admission_hospital','symptoms','lives_in_Wuhan',
          ␣
 ↪'travel_history_dates','travel_history_location','chronic_disease','source',
           'sequence_available','date_death_or_discharge',
           'location','admin3','admin2','admin1','travel_history_binary',␣
 ↪'age_group',
          "admin_id", "admin_id", "sex", "date_confirmation", "country_new",␣
 ↪"age_group_code", "province_code","city_code" ]
```

```
df_ml = df.drop(to_drop,axis=1)
df_ml.columns
```

[41]:
```
Index(['age', 'chronic_disease_binary', 'outcome', 'country_code', 'sex_code'],
dtype='object')
```

[42]:
```
df_ml.isna().sum()
```

[42]:
```
age                       0
chronic_disease_binary    0
outcome                   0
country_code              0
sex_code                  0
dtype: int64
```

Spliting Data set into train and test

[43]:
```
Y = df_ml["outcome"].values
X = df_ml.drop(columns=["outcome"]).values
```

Spliting Data set into train and test

[44]:
```
from sklearn import model_selection

x_train, x_test, y_train, y_test = model_selection.train_test_split(X,␣
↪Y,test_size=0.3)
```

Normalizing dataset

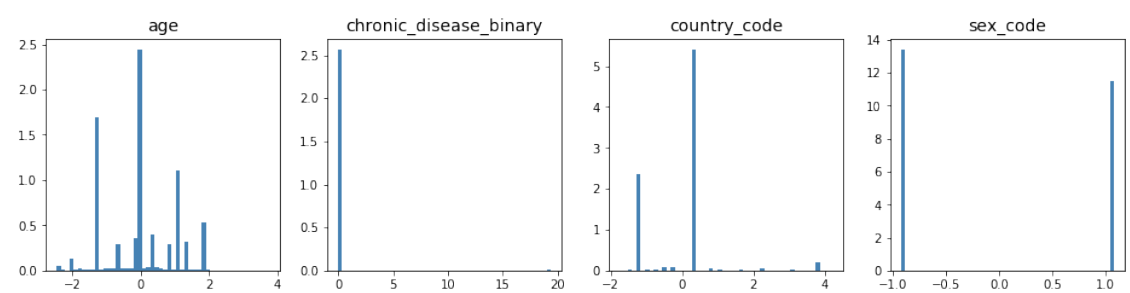
[45]:
```
from sklearn import preprocessing

std_scale = preprocessing.StandardScaler().fit(X)
x_train_std = std_scale.transform(x_train)
x_test_std = std_scale.transform(x_test)
```

vizualizing un normalized training set

[46]:
```
fig = plt.figure(figsize=(16, 12))
column_names = [col for col in list(df_ml.columns) if col != "outcome"]

for feat_idx in range(x_train_std.shape[1]):
    ax = fig.add_subplot(3,4, (feat_idx+1))
    h = ax.hist(x_train_std[:, feat_idx], bins=50, color = 'steelblue',␣
↪density=True, edgecolor='none')
    ax.set_title(column_names[feat_idx], fontsize=14)
```



5.1 A.Selecting a model

5.1.1 Computing a baseline

The baseline is a classiefier model with poor resultats that we will use to give us an idea of what are relatively to our problem poor results.

Here we chose a Dummy Classifier that will always predicts the most frequent class of the dataset (which in our case will be outcome = 0).

[47]:
```
from sklearn.dummy import DummyClassifier
dum = DummyClassifier(strategy='most_frequent')


dum.fit(x_train_std, y_train)
y_pred_dum = dum.predict(x_test_std)
```

5.1.2 Evaluating the baseline

RMSE: Root Mean Square Value. Differences between values predicted by a model and the values observed.

$$rmse = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$$

Acuracy Score: ratio of number of correct predictions to the total number of input samples.

[48]:
```
from sklearn import metrics

print("RMSE : {:.2f}".format(np.sqrt(metrics.mean_squared_error(y_test,␣
↪y_pred_dum))))
print("Accuracy Score: {:.2f}".format(metrics.
↪accuracy_score(y_test,y_pred_dum)))
```

```
RMSE : 0.03
Accuracy Score: 1.00
```

AUC - ROC curve: performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.

AUC: Area Under The Curve
ROC: Receiver Operating Characteristics
TP: True Positive
FP: False Positive
FN: False Negative
TN: True Negative

True Positive Rate / Recall / Sensitivity:

$$Recall = \frac{TP}{TP + FN}$$

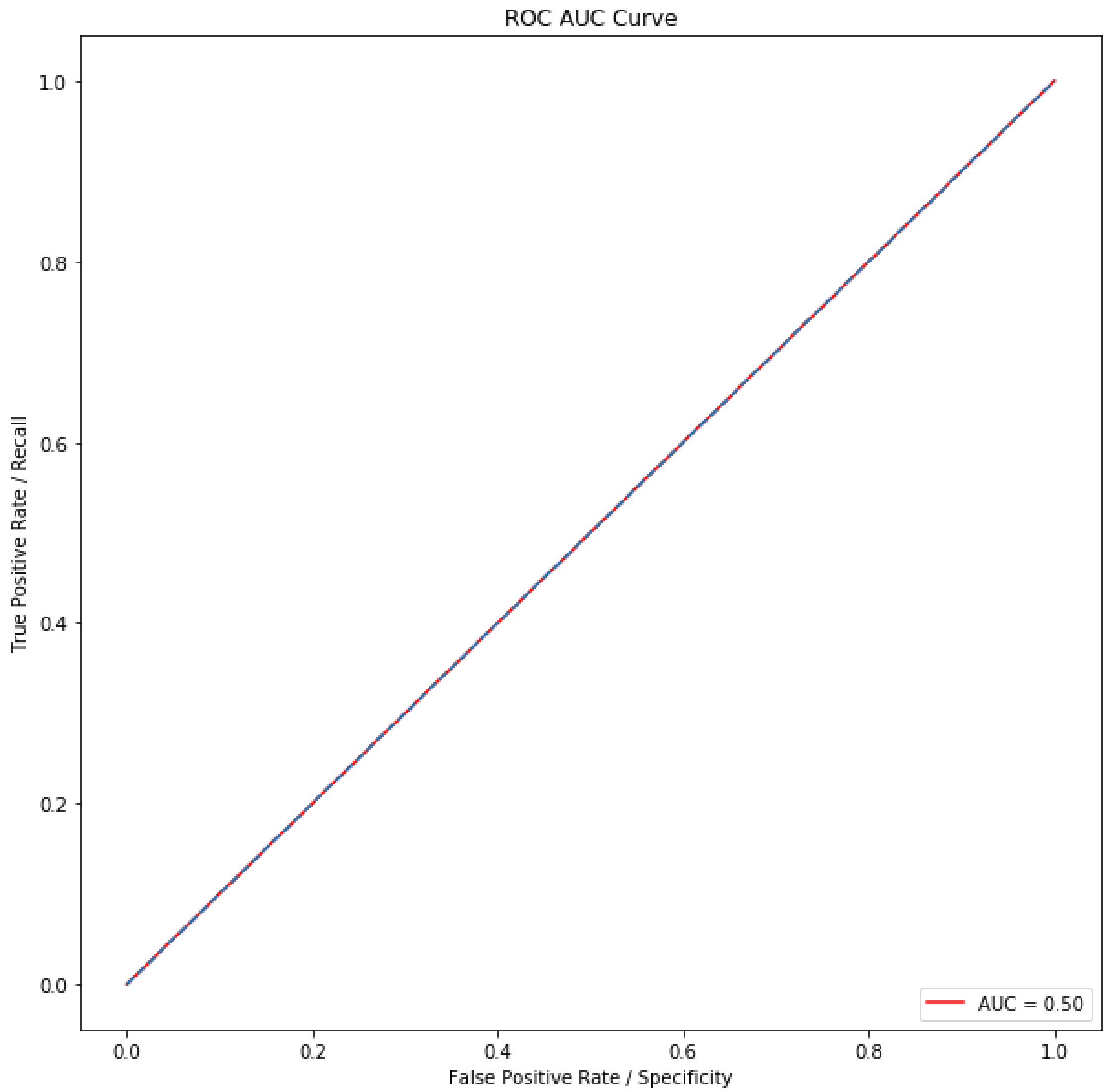
False Positive Rate:

$$1 - Specificity = 1 - \frac{FP}{TN + FP}$$

Higher the Area Under the curve, the better the model is ! (AUC closer to 1.0)
source : https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[49]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_pred_dum)
roc_auc = auc(false_positive_rate, true_positive_rate)

plt.figure(figsize=(10,10))
plt.title('ROC AUC Curve')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.
 ↪2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate / Recall')
plt.xlabel('False Positive Rate / Specificity')
plt.show()
```



The ROC curve here indicates that the Dummy Classifier doesn't make any differences between the classes.

5.1.3 KNN Modeling

[50]:
```
from sklearn import neighbors, metrics

clf = neighbors.KNeighborsClassifier(4)

clf.fit(x_train_std, y_train)
```

[50]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

[51]:
```
y_predict = clf.predict(x_test_std)
```

5.1.4 Evaluating the 3 neighbor KNN Unbalanced Dataset

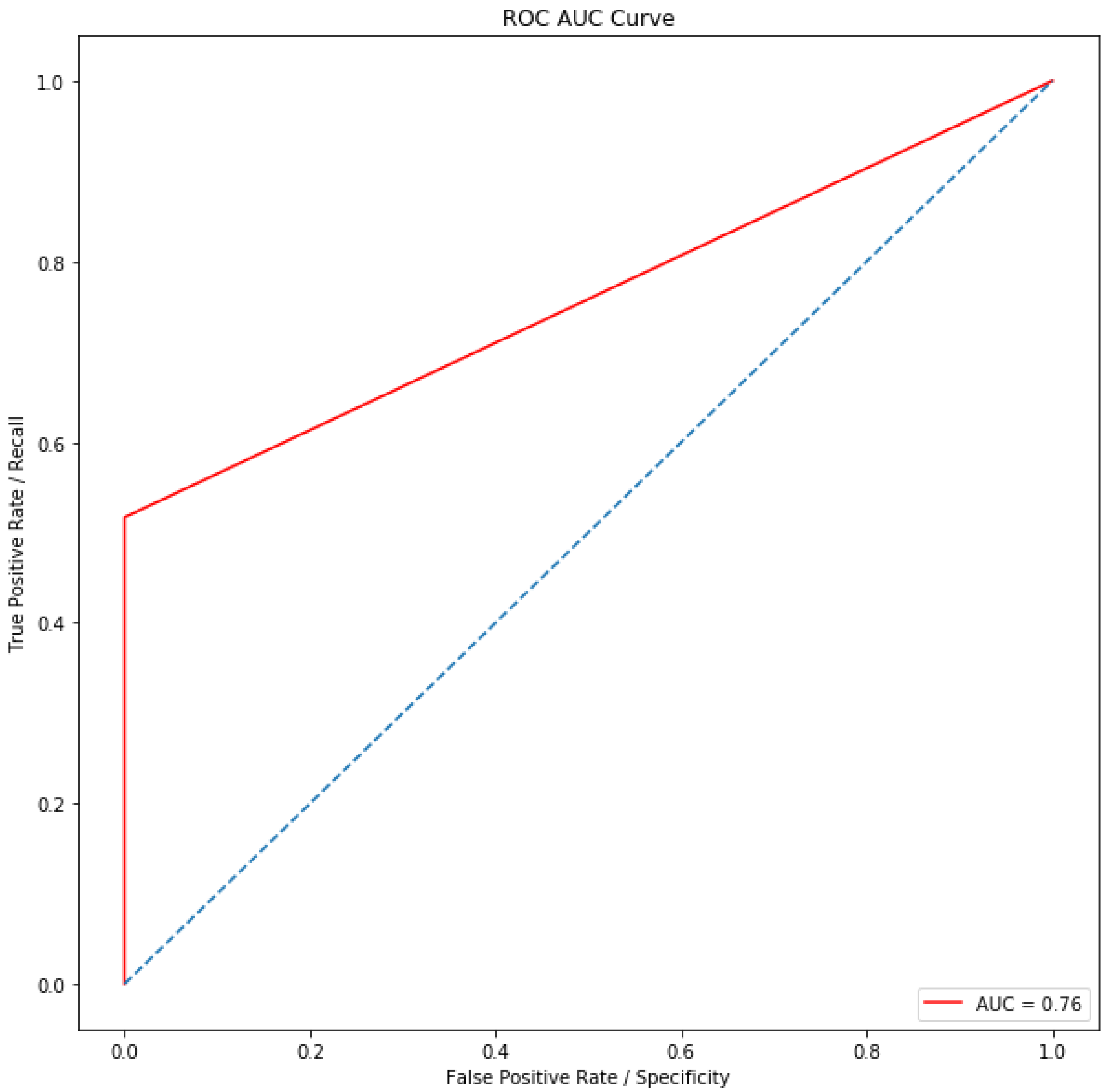
[52]:
```
print("RMSE : {:.2f}".format(np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_predict))))
print("Accuracy Score: {:.2f}".format(metrics.accuracy_score(y_test,y_predict)))
```

```
RMSE : 0.03
Accuracy Score: 1.00
```

The results of the RMSE and the Accuracy are the same as the baseline which isn't a good indicator.

[53]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_predict)
roc_auc = auc(false_positive_rate, true_positive_rate)

plt.figure(figsize=(10,10))
plt.title('ROC AUC Curve')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.
 ↪2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate / Recall')
plt.xlabel('False Positive Rate / Specificity')
plt.show()
```



The ROC shows that the model performs better than the dummy classifier. The model has a a fair AUC of 0.67 but therefore isn't precise at all

[54]:
```
# positive percentage in test set
(y_test.sum()/y_test.shape[0]) * 100
```

[54]: 0.12034659820282415

[55]:
```
# positive percentage in test set
(y_train.sum()/y_train.shape[0]) * 100
```

[55]: 0.10573550649886528

The proportion of positive classes (outcome = 1.0 which means patient dies) is very small and is around 0.1% of the dataset in train and in test.

This heavily impacts the predictions of our KNN. One solution would be increase the proportion of "positive" outcomes for the training or changing to another type of model.

5.2 B. Age Prediction using Regression

Use the Regression to predict the age of persons based on other variables. You have the choice on these explanatory variables? How you choose these variables? Compute the quality of the prediction using MSE error (Mean Squared Error)

[56]:
```
df_ml.columns
```

[56]:
```
Index(['age', 'chronic_disease_binary', 'outcome', 'country_code', 'sex_code'],
dtype='object')
```

As a label of the regression we will use an the age and use the following variables : - *chronic_disease_binary*: boolean whether the person has a chronic disease, chronic disease can be increasted at a late age. - *outcome*: number 1.0 died, 0.0 recovered - *country_code*: number associated to the country - *sex_code*: 0.0 female, 1.0 male

5.2.1 Preparing dataset

[57]:
```
Y = df_ml["age"].values
X = df_ml.drop(columns=["age"]).values
```

[58]:
```
x_train, x_test, y_train, y_test = model_selection.train_test_split(X,␣
↪Y,test_size=0.3)
```

[59]:
```
from sklearn import preprocessing

std_scale = preprocessing.StandardScaler().fit(X)
x_train_std = std_scale.transform(x_train)
x_test_std = std_scale.transform(x_test)
```

5.2.2 Training Regression Model

[60]:
```
from sklearn import neighbors, model_selection

knn = neighbors.KNeighborsRegressor(4)

knn.fit(x_train_std, y_train)
```

[60]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                    weights='uniform')
```

5.2.3 Evaluating Model Performances

[61]:
```
y_pred = knn.predict(x_test_std)
```

Rounding the predicted ages

[62]:
```
y_pred = np.around(y_pred, decimals=0)
```

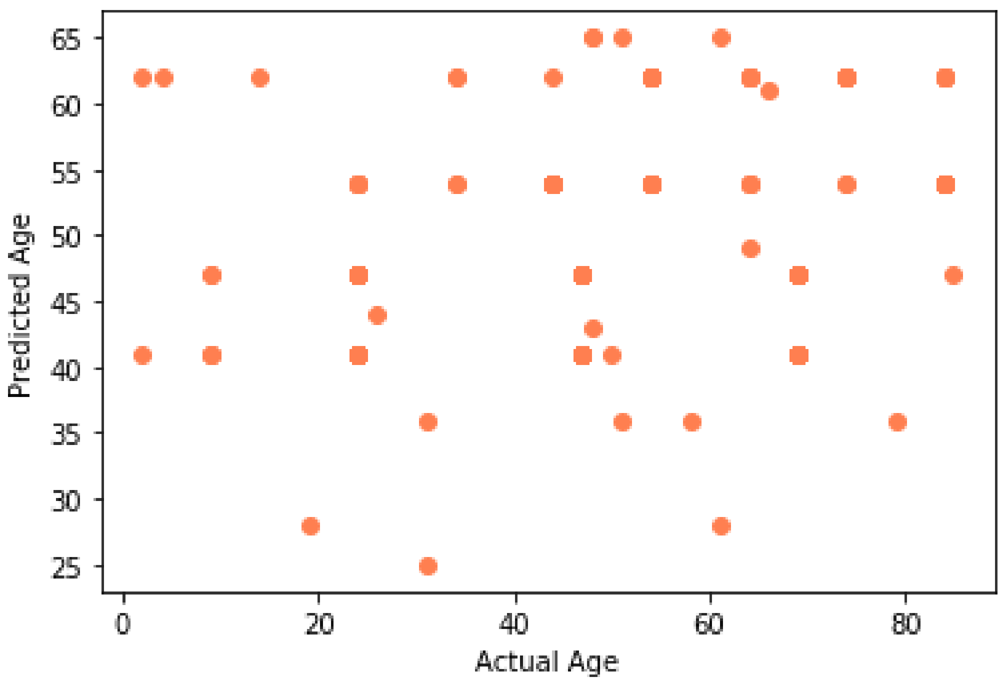
[63]:
```
print("MSE : {:.2f}".format(np.sqrt(metrics.mean_squared_error(y_test, y_pred)␣
↪)))
print("MAD : {:.2f}".format(np.sqrt(metrics.mean_absolute_error(y_test, y_pred)␣
↪)))
```

```
MSE : 18.81
MAD : 3.87
```

Plotting Results

[64]:
```
plt.scatter(y_test[::200],y_pred[::200], color='coral')
plt.ylabel('Predicted Age')
plt.xlabel('Actual Age')
```

[64]:
```
Text(0.5, 0, 'Actual Age')
```



Our Model seems to perform poorly

5.3 C. Clustering Method

Apply a clustering method (K-means) on the dataset to segment the persons in different clusters. Use the Silhouette index to find out the best number of clusters. Plot the results using scatter to visually analyse the clustering structure.

[65]:
```
X = df_ml.values[::50]
```

5.3.1 Finding the most appropriate number of clusters

Finding best number of clusters

[66]:
```
from sklearn import cluster, metrics


scores = []
for i in range(2,15):
    km = cluster.KMeans(i)
    km.fit(X)
    scores.append(metrics.silhouette_score(X, km.labels_))
```

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

[67]:
```
max_score = max(scores)

nb_clusters = [i for i,j in enumerate(scores) if j == max_score][0]

km = cluster.KMeans(nb_clusters)
km.fit(X)

print(f"The best silhouette score here is : {max_score} and  represents␣
 ↪{nb_clusters}.")
```

```
The best silhouette score here is : 0.8870294673826611 and  represents 12.
```

5.3.2 Reducing dimensions

[68]:
```
from sklearn import preprocessing

X_norm = preprocessing.scale(X)
```

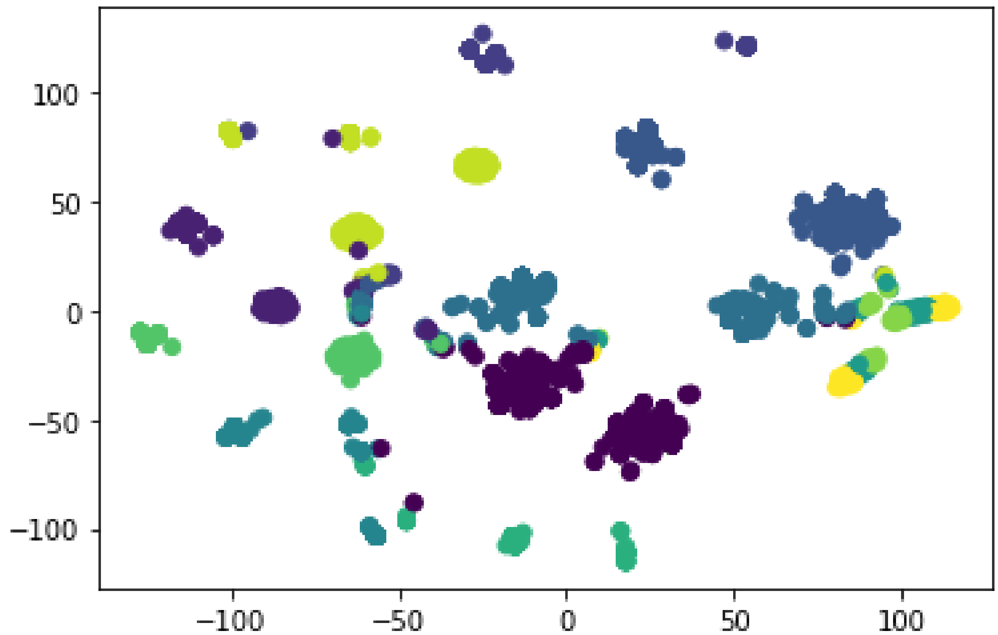
[69]:
```
from sklearn import manifold, preprocessing

tsne = manifold.TSNE(n_components=2, init='pca')
X_tsne = tsne.fit_transform(X_norm)
```

5.3.3 Plotting with outlined clusters

[70]:
```
plt.scatter(X_tsne[:,0],X_tsne[:,1], c=km.labels_)
```

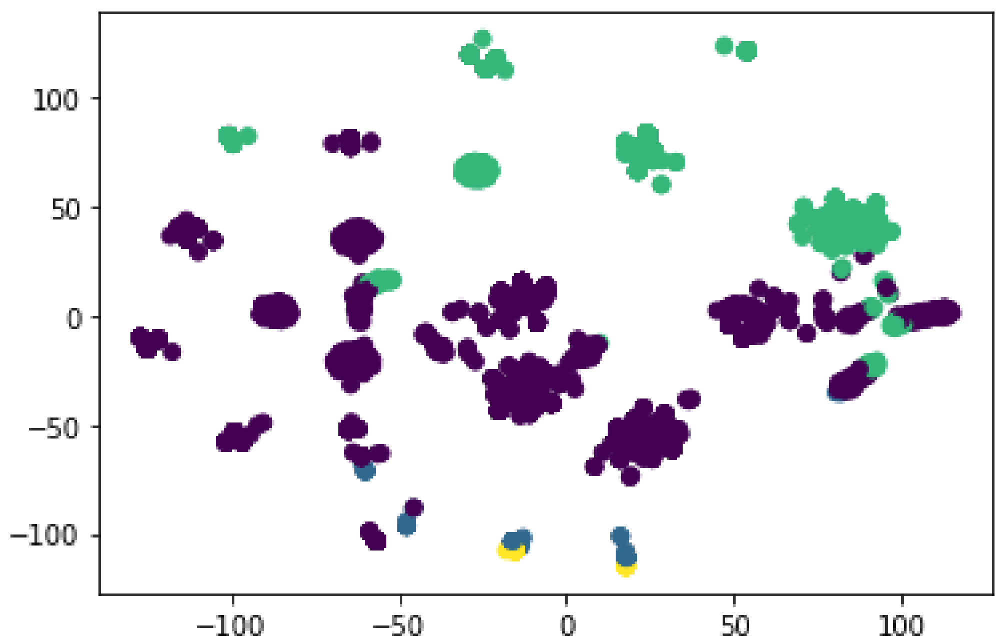
[70]:
```
<matplotlib.collections.PathCollection at 0x13a1c7ba8>
```



5.3.4 Coloring the clusters according to different classes

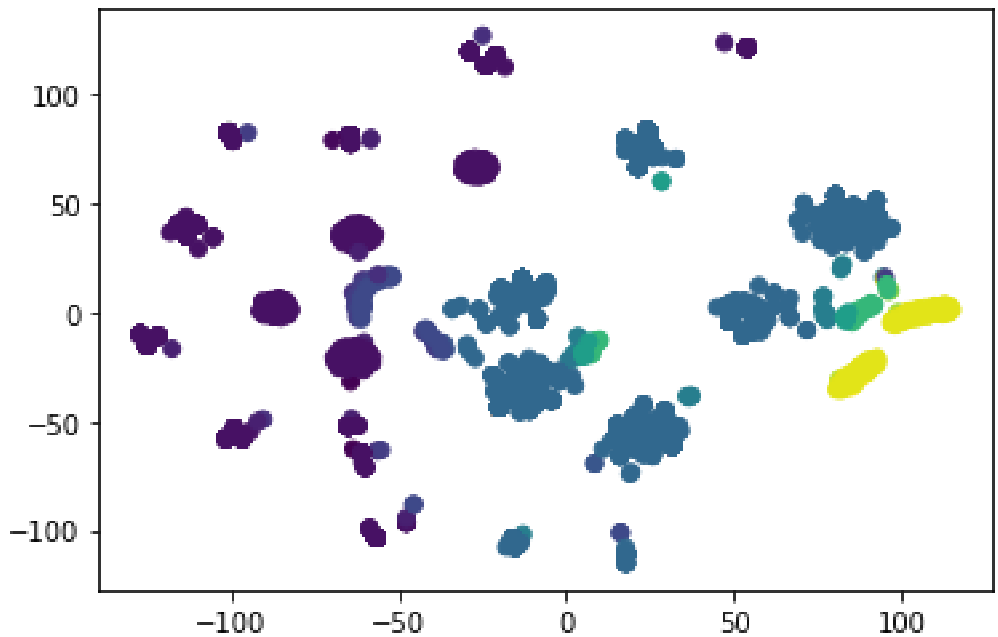
[71]:
```
plt.scatter(X_tsne[:,0],X_tsne[:,1], c=df["age_group_code"].values[::50])
```

[71]: `<matplotlib.collections.PathCollection at 0x13b409748>`



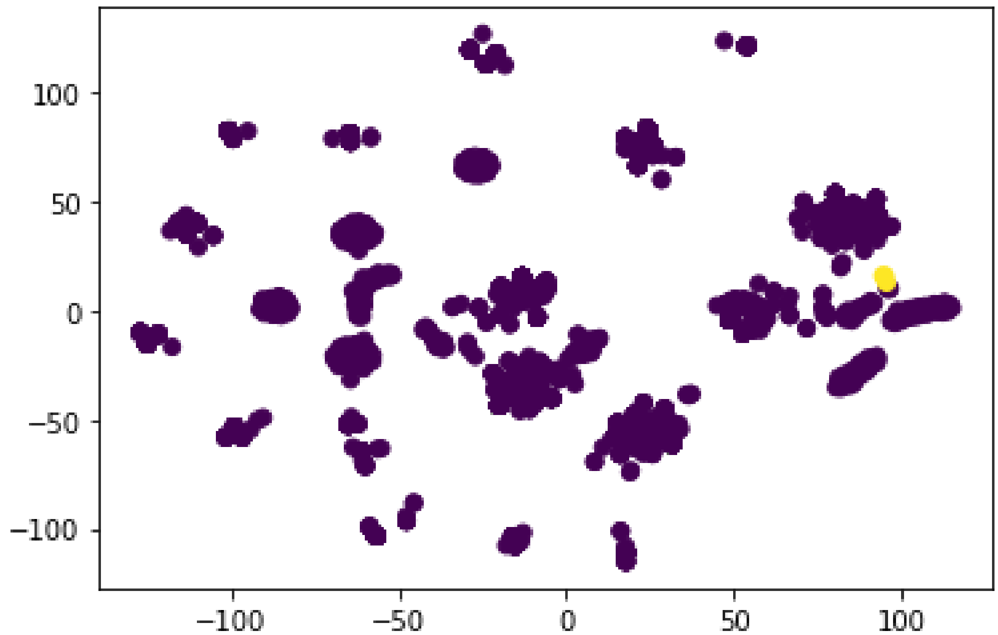
[72]:
```
plt.scatter(X_tsne[:,0],X_tsne[:,1], c=df_ml["country_code"].values[::50])
```

[72]: <matplotlib.collections.PathCollection at 0x13a23b2b0>



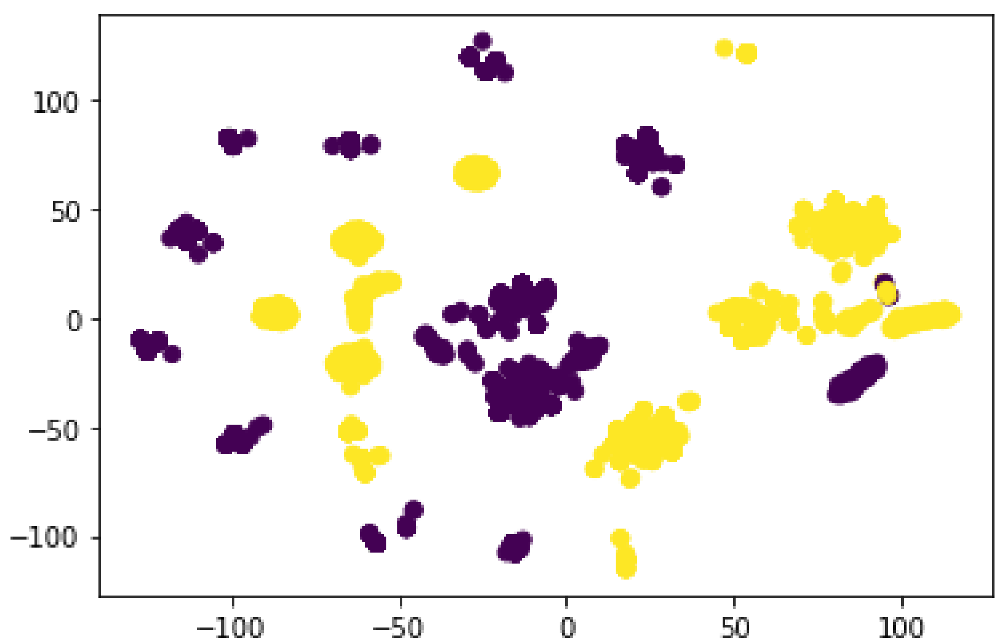
[73]:
```
plt.scatter(X_tsne[:,0],X_tsne[:,1], c=df_ml["outcome"].values[::50])
```

[73]: <matplotlib.collections.PathCollection at 0x13a2c8da0>



[74]:
```
plt.scatter(X_tsne[:,0],X_tsne[:,1], c=df_ml["sex_code"].values[::50])
```

[74]: <matplotlib.collections.PathCollection at 0x13105d4e0>



6 4. Improving the results and Theoretical formalism

6.1 A. Balancing out the majority dataset

The data is unbalanced. You can balance it by reducing randomly the majority class. Assume that you extract randomly samples that are balanced. How the prediction results will change?

As said in 2.A:

"The proportion of positive classes (outcome = 1.0 which means patient dies) is very small and is around 0.1% of the dataset in train and in test."

The KNN keeps in memory every datapoint and minimizes finds the minimum distance between an input X and the K number of neighbors.

In our case we have very few neighbors to relate to for the "positive" class, which probably highly impacts prediction.

We will try balancing out the majority class

6.1.1 Before class resampling

[75]:
```
print(f"Number of samples with outcome negative outcome (patient recovers): {␣
↪df_ml[df_ml['outcome'] == 0.0].count()[0]}" )
print(f"Number of samples with outcome positive outcome (patient dies): {␣
↪df_ml[df_ml['outcome'] == 1.0].count()[0]}")
```

```
Number of samples with outcome negative outcome (patient recovers): 166001
Number of samples with outcome positive outcome (patient dies): 183
```

Here is a link to the evaluation of the KNN training with 9 neighbors with the highly imbalanced dataset

Section **??**

6.1.2 Resampling to balance out the proportion of the positive and negative classes

[76]:
```
df_majority = df_ml[df_ml['outcome'] == 0.0] # negative class
df_minority = df_ml[df_ml['outcome'] == 1.0] # positive class
```

[77]:
```
from sklearn.utils import resample

df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=df_minority.shape[0],
                                   random_state=123)

df_downsampled = pd.concat([df_majority_downsampled, df_minority])
df_downsampled["outcome"] = np.concatenate([np.full(df_minority.shape[0], 0.0),␣
↪np.full(df_minority.shape[0], 1.0)])
```

[78]:
```
neg_count = df_downsampled[df_downsampled['outcome'] == 0.0].count()[0]
pos_count = df_downsampled[df_downsampled['outcome'] == 1.0].count()[0]
total_count = df_downsampled.count()[0]
```

```
print(f"Number of samples with outcome negative outcome (patient recovers):␣
 ↪{neg_count}")
print(f"Number of samples with outcome positive outcome (patient dies):␣
 ↪{pos_count}")
print(f"Total new Training set size: {total_count}")
```

```
Number of samples with outcome negative outcome (patient recovers): 183
Number of samples with outcome positive outcome (patient dies): 183
Total new Training set size: 366
```

[79]:
```
Y = df_downsampled["outcome"].values
X = df_downsampled.drop(columns=["outcome"]).values
```

Spliting Data set into train and test

[80]:
```
from sklearn import model_selection

x_train, x_test, y_train, y_test = model_selection.train_test_split(X,␣
 ↪Y,test_size=0.3)
```

6.1.3 Training the same KNN with balanced training set dataset

Normalizing dataset to avoid euclidian distance instability due to the magnitude of other features

[81]:
```
from sklearn import preprocessing

std_scale = preprocessing.StandardScaler().fit(X)
x_train_std = std_scale.transform(x_train)
x_test_std = std_scale.transform(x_test)
```

Training the KNN Classifier with 4 neighbors

[82]:
```
from sklearn import neighbors, metrics

clf = neighbors.KNeighborsClassifier(4)

clf.fit(x_train_std, y_train)
```

[82]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

6.1.4 Evaluating the KNN with balanced dataset

[83]:
```
y_predict = clf.predict(x_test_std)
```

[84]:
```
print("MSE : {:.2f}".format(np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_predict))))
print("Accuracy Score: {:.2f}".format(metrics.accuracy_score(y_test,y_predict)))
```

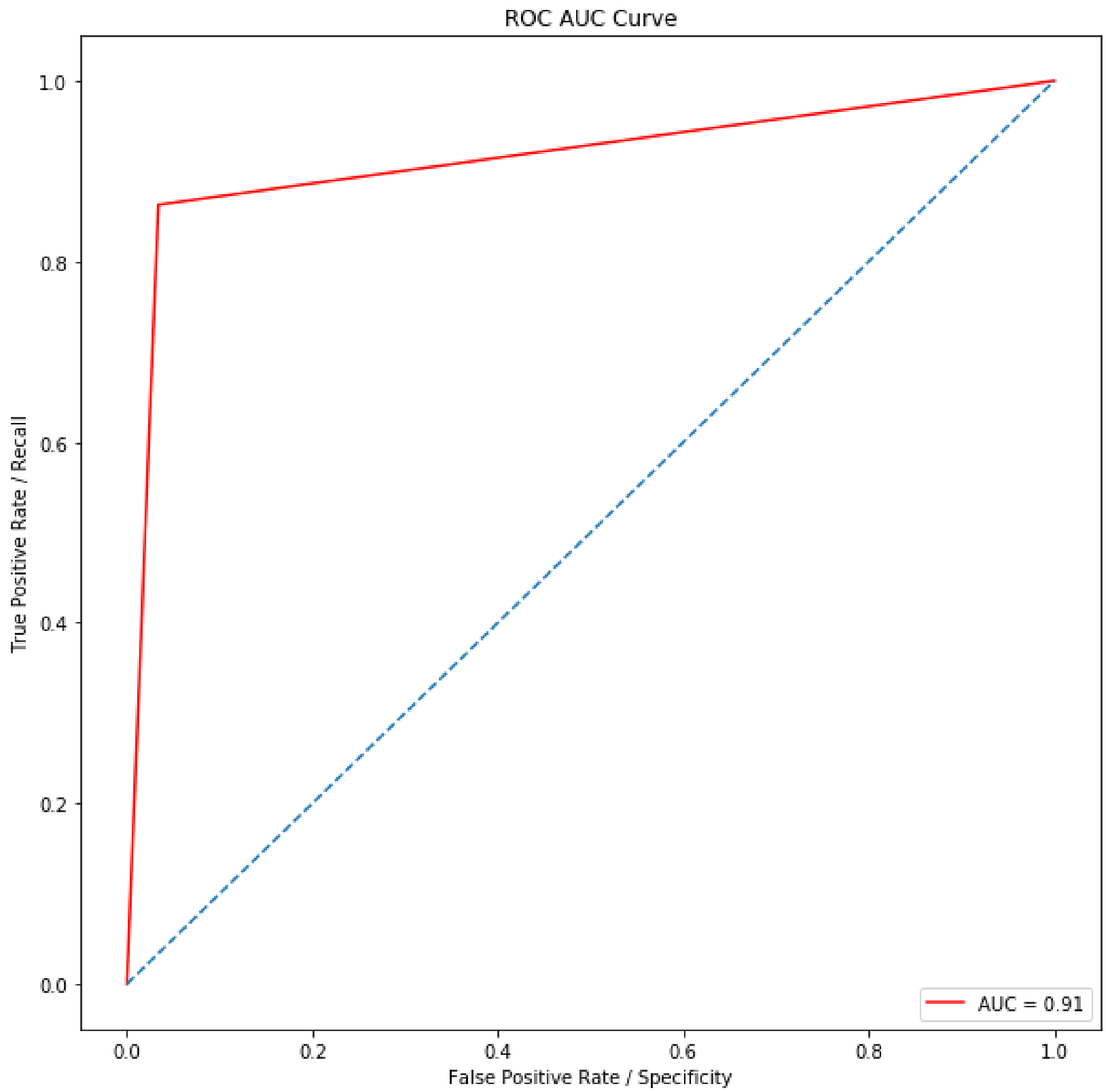
```
MSE : 0.29
Accuracy Score: 0.92
```

[85]:
```python
print("Predicted Died: {:.2f}".format(y_predict.sum()))
print("Actually Died: {:.2f}".format(y_test.sum()))
```

```
Predicted Died: 46.00
Actually Died: 51.00
```

[86]:
```python
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_predict)
roc_auc = auc(false_positive_rate, true_positive_rate)

plt.figure(figsize=(10,10))
plt.title('ROC AUC Curve')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.
 ↪2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate / Recall')
plt.xlabel('False Positive Rate / Specificity')
plt.show()
```



6.1.5 Conclusion

The downsampling of the negative class improved the AUC Curve lot bettter. The AUC improved from .67 to .90 which is a big improvement and the MSE decreased.

The accuracy of the model decreased however because the proprtion of people who recovered really increased.

Here for example:

6.2 B. Better managing missing values

The KNN model accepts the missing values to make predictions. However it is important to see the proportion of missing values in the dataset.

This dataset contains lots of missing value and the cleanup job was quite tedious. The missing values weren't replaced in this dataset to avoid unexpected behavoir, especially if the "positive" class had a very low proportion.

The KNN in this notebook was trained with values that weren't missing.

6.2.1 First approach

Use an algorithm that replaces each missing value in every column by either : - the mean (positive + negative targets) - the mode (positive + negative targets) - the median (positive + negative targets)

The algorithm *SimpleImputer* in sklean can do so and fill up the dataset.

6.2.2 Second Approach

Use an algorithm that replaces each missing value in every column by either : - the mean of the target class - the mode of the target class - the median of the target class

6.2.3 Third Approach

Depending on the proportion of missing values in a column. Use a model to predict the missing values. This approach can be tedious and inappropriate if there is a big proportion of the dataset

However it is important to make hypothese and use features that are linked to what our model aims at. In this dataset very few features are actually relevant for our ,

6.3 C. Finding best parameters using Grid Search

To find the best parameters for the models, the Grid Search algorithm can be used which is available in scikit-learn library. Explain the algorithm and use it for the learning models to find the best parameters

6.3.1 Explaination

The Grid Search uses Cross Validation to find the best parameter for a given score.

The Cross Validation algorithm is the following:

```
Input: X data (dimension nxp), Y labels (dimension n), number of folds k
Cut [0, 1, ..., n-1] into k parts of size (n/k). (The last part will be a little smaller if n i

for i=0 to (k-1):
    Form the test dataset (X_test, y_test) by restricting X and y to the indices contained in t
    Form the training game (X_train, y_train) by restricting X and y to the other clues.
    Training the algorithm on the training game
    Use the resulting model to predict on the test set
        Calculate the model error by comparing the predicted labels to the real labels containe

Output: the mean value of the errors calculated on the k folds.
```

The Grid Search will take as an input the scoring method used and the hyper parameters to use the results of the Cross Validation and return the most optimized hyperparameter for the given scoring method (minimum or maximum depending on score method).

For example: the "accuracy" scoring startegy will be maximized.

6.3.2 Using Grid Search on the KNN and balanced Dataset

[87]:
```
from sklearn import neighbors, metrics

# hyperparameters to set
param_grid = {"n_neighbors": [1,2,3,4,5,6,7,9,10,11,13,15]}

score = 'accuracy'

clf = model_selection.GridSearchCV(
    neighbors.KNeighborsClassifier(),
    param_grid, # hyperparameters to test
    cv=5, # folds for cross validation (5 or 10 generally)
    scoring=score # score to optimize
)

# optimize the classifier on the training set
clf.fit(x_train_std, y_train)

print("Best Hyperparameters on training test")
print(clf.best_params_)

print("Cross validation results")
for mean, std, params in zip(
        clf.cv_results_['mean_test_score'],
        clf.cv_results_['std_test_score'],
        clf.cv_results_['params']
    ):

    print("{} = {:.3f} (+/-{:.03f}) for {}".format(
        score,
        mean,
        std*2,
        params
    ) )
```

```
Best Hyperparameters on training test
{'n_neighbors': 3}
Cross validation results
accuracy = 0.906 (+/-0.058) for {'n_neighbors': 1}
accuracy = 0.879 (+/-0.095) for {'n_neighbors': 2}
accuracy = 0.910 (+/-0.073) for {'n_neighbors': 3}
```

```
accuracy = 0.910 (+/-0.052) for {'n_neighbors': 4}
accuracy = 0.906 (+/-0.046) for {'n_neighbors': 5}
accuracy = 0.879 (+/-0.069) for {'n_neighbors': 6}
accuracy = 0.883 (+/-0.068) for {'n_neighbors': 7}
accuracy = 0.871 (+/-0.055) for {'n_neighbors': 9}
accuracy = 0.855 (+/-0.054) for {'n_neighbors': 10}
accuracy = 0.867 (+/-0.087) for {'n_neighbors': 11}
accuracy = 0.832 (+/-0.054) for {'n_neighbors': 13}
accuracy = 0.840 (+/-0.050) for {'n_neighbors': 15}

/usr/local/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:813: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[88]:
```
y_predict = clf.predict(x_test_std)
```

6.3.3 Evaluating model optimized with Cross Validation

[89]:
```
print("MSE : {:.2f}".format(np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_predict))))
print("Accuracy Score: {:.2f}".format(metrics.accuracy_score(y_test,y_predict)))
```

```
MSE : 0.33
Accuracy Score: 0.89
```

[90]:
```
print("Predicted Died: {:.2f}".format(y_predict.sum()))
print("Actually Died: {:.2f}".format(y_test.sum()))
np.logical_and(y_predict, y_test).sum()
```

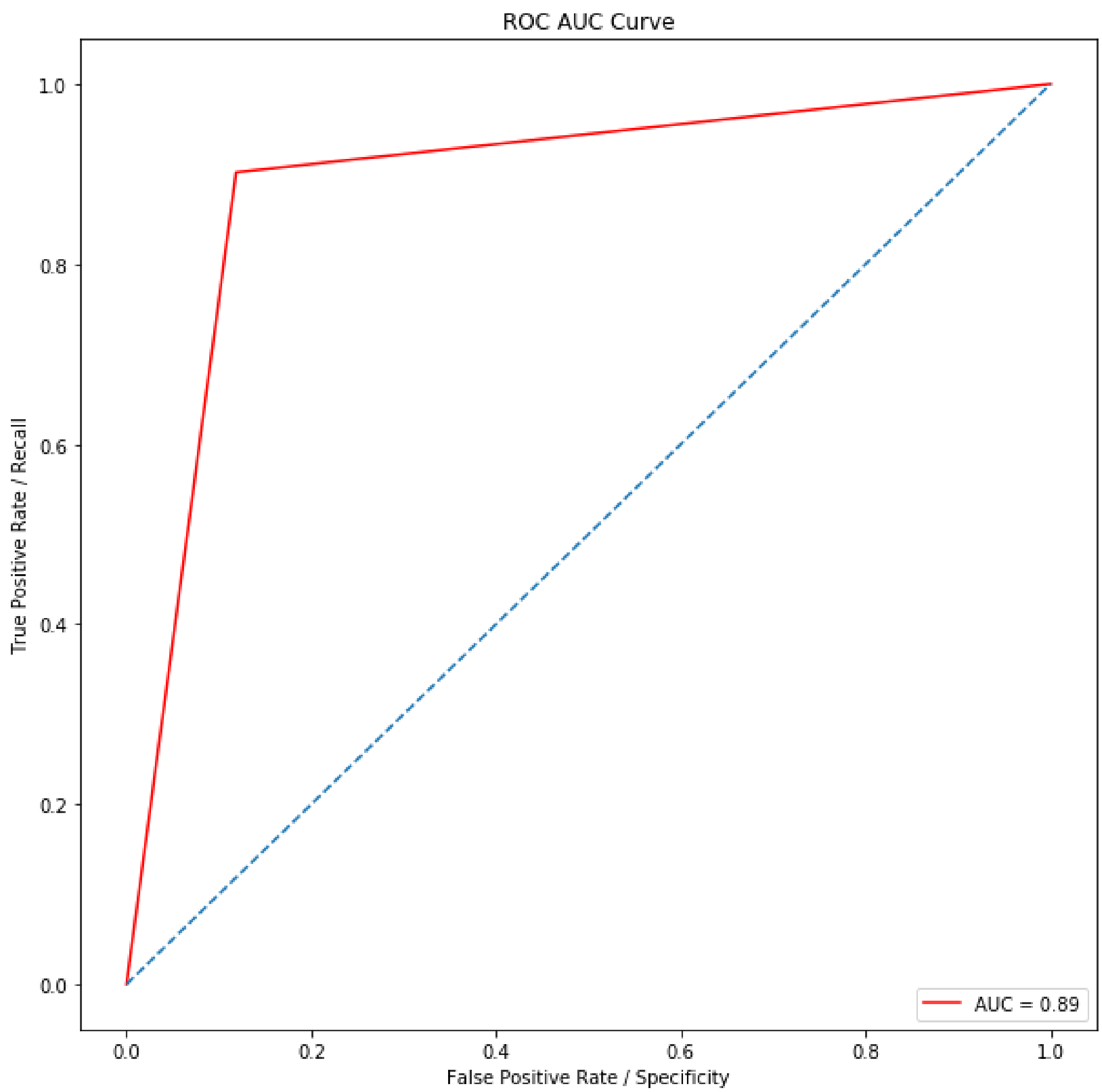
```
Predicted Died: 53.00
Actually Died: 51.00
```

[90]:
```
46
```

[91]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_predict)
roc_auc = auc(false_positive_rate, true_positive_rate)

plt.figure(figsize=(10,10))
plt.title('ROC AUC Curve')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.
 ↪2f' % roc_auc)
plt.legend(loc = 'lower right')
```

```
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate / Recall')
plt.xlabel('False Positive Rate / Specificity')
plt.show()
```



As expected by choosing to optimize the accuracy score using Grid Search the accuracy went from 0.93 to 0.94 and the MSE went from 0.27 to 0.25 (1 neighbor hyper parameter). Adding to this the AUC went from 0.93 to 0.97 which means means that overall the given metrics would consider this model as very good.

However the dataset here is balanced and is quite small (366 rows), this model probably need further evaluation with a larger dataset and predictions with unbalanced data.

6.4 D. Mathematical Formalism and Conclusion

Give the algorithmically (mathematical) formalism of the method which give the best results. Explain all the parameters of the used method and their impact on the results. Some comparison with public results should me made to conclude the project.

6.4.1 Mathematical formalism

In our cas the "best" classifier was the KNN with 1 neighbor using the following features to predict the outcome of a patient (recovers : 0.0 or dies : 1.0): - age: rounded Int - chronic_disease_binary: boolean - sex_code (0: female, 1: male) - country_code : standeradized int assigned to a country

The KNN calculates a distance bewteen the test data and the input to give prediction using the Eucleadean distance Formula.

$$EuclideanDistance = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

In our case in a 1-NN the model will predict the class of the closest neighbor.

However the KNN is makes no prior assumption of the data but tends to become instable if the data isn't standardized (the euclidean outpiuted distance could be higlhly biased on the the the magnitude of features).
The KNN has to keep in memory every datapoint that it uses for training and the increasing the number of features really increases computation time.

6.4.2 Public Results and Conclusion

https://www.nature.com/articles/s42256-020-0180-7

predict the mortality of patients 90% accuracy model based on three bio markers to help prioritise patients using the following features : - lactic dehydrogenase (LDH), - lymphocyte - high-sensitivity C-reactive protein (hs-CRP). The model uses a very different dataset because it uses extra bio markers. The model seems to be using a combination of decision trees, and the features werer selected using a XGBoost algortihm.

In our study we probably should try and use extra features to make our predictions and using more complexe models and approaches that work better on dataset with a small proportion of positive classes.

[ ]:

```

```