ADS502

Analysis and Classification Final Project

New York State Graduation Rates for Pre and During COVID

Jiaqi He, Lane Whitmore, Azucena Faus

Setup

In [1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import glob

from sklearn.model_selection import train_test_split
import random
import statsmodels.tools.tools as stattools
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix
from tabulate import tabulate
from sklearn.ensemble import RandomForestClassifier
from scipy import stats
import seaborn as sns
import plotly.express as px
from sklearn  import metrics
sns.set()
```

In [2]:
```
#import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

In [3]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.tree import plot_tree
from time import time
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from xgboost import XGBClassifier
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow import keras
from sklearn.naive_bayes import MultinomialNB
from itertools import cycle, islice
from pandas.plotting import parallel_coordinates
from kmodes.kmodes import KModes
from apyori import apriori
```

Pre-Process Data

Combine datafiles from original sources and clean:

In [4]:
```
prj_path = 'cleaning data/'
```

In [10]:
```
csv_files = glob.glob(prj_path + "/*.csv")
csv_files = [csv for csv in csv_files if not 'clean_combine.csv' in csv]
# filter out the output csv only keep raw data csv
```

In [11]:
```
df_list = []
for file_path in csv_files:
    df_list.append(f'df_{file_path[-8:-4]}')
    exec(f'df_{file_path[-8:-4]} = pd.read_csv(file_path)')
```

In [12]:
```
clean_combine_df = pd.DataFrame()
for df_name in df_list:
# read df in for loop so don't need to apply same cleaning to 4 different df
    exec(f"df = {df_name}")
    df.columns = df.columns.str.lower()
    df = df.drop(['lea_beds','lea_name', 'nrc_code', 'nrc_desc', 'nyc_ind',
                  'boces_code', 'boces_name'], axis=1)
    # droped those columns because not plan to find relations beween those
    # columns and covid
    df = df.drop(['aggregation_code', 'aggregation_name'], axis=1)
    # drop columns because those are duplicated data for county_name
    # if we focus on county data
    if 'entity_inactive_date' in df:
        df = df.drop(['entity_inactive_date'], axis=1)
    df = df.loc[df.aggregation_index == 2]
    # only pick county data
    df = df.loc[df.subgroup_code.isin(list(range(1,9)))]
    # clean up some subgroups
    print(df.columns)
    print(len(df.columns))
    clean_combine_df = pd.concat([clean_combine_df, df], ignore_index=True)
```

```
Index(['report_school_year', 'aggregation_index', 'aggregation_type',
       'county_code', 'county_name', 'membership_code', 'membership_key',
       'membership_desc', 'subgroup_code', 'subgroup_name', 'enroll_cnt',
       'grad_cnt', 'grad_pct', 'local_cnt', 'local_pct', 'reg_cnt', 'reg_pct',
       'reg_adv_cnt', 'reg_adv_pct', 'non_diploma_credential_cnt',
       'non_diploma_credential_pct', 'still_enr_cnt', 'still_enr_pct',
       'ged_cnt', 'ged_pct', 'dropout_cnt', 'dropout_pct'],
      dtype='object')
27
Index(['report_school_year', 'aggregation_index', 'aggregation_type',
       'county_code', 'county_name', 'membership_code', 'membership_key',
       'membership_desc', 'subgroup_code', 'subgroup_name', 'enroll_cnt',
       'grad_cnt', 'grad_pct', 'local_cnt', 'local_pct', 'reg_cnt', 'reg_pct',
       'reg_adv_cnt', 'reg_adv_pct', 'non_diploma_credential_cnt',
       'non_diploma_credential_pct', 'still_enr_cnt', 'still_enr_pct',
       'ged_cnt', 'ged_pct', 'dropout_cnt', 'dropout_pct'],
      dtype='object')
27
Index(['report_school_year', 'aggregation_index', 'aggregation_type',
       'county_code', 'county_name', 'membership_code', 'membership_key',
       'membership_desc', 'subgroup_code', 'subgroup_name', 'enroll_cnt',
       'grad_cnt', 'grad_pct', 'local_cnt', 'local_pct', 'reg_cnt', 'reg_pct',
       'reg_adv_cnt', 'reg_adv_pct', 'non_diploma_credential_cnt',
```

```
       'non_diploma_credential_pct', 'still_enr_cnt', 'still_enr_pct',
       'ged_cnt', 'ged_pct', 'dropout_cnt', 'dropout_pct'],
      dtype='object')
27
Index(['report_school_year', 'aggregation_index', 'aggregation_type',
       'county_code', 'county_name', 'membership_code', 'membership_key',
       'membership_desc', 'subgroup_code', 'subgroup_name', 'enroll_cnt',
       'grad_cnt', 'grad_pct', 'local_cnt', 'local_pct', 'reg_cnt', 'reg_pct',
       'reg_adv_cnt', 'reg_adv_pct', 'non_diploma_credential_cnt',
       'non_diploma_credential_pct', 'still_enr_cnt', 'still_enr_pct',
       'ged_cnt', 'ged_pct', 'dropout_cnt', 'dropout_pct'],
      dtype='object')
27
```

In [13]:
```
len(clean_combine_df)
```

Out[13]:
```
10587
```

In [14]:
```
clean_combine_df.to_csv(prj_path + '/clean_combine.csv')
```

In [15]:
```
clean_combine_df.isnull().values.any()
```

Out[15]:
```
False
```

Get to know the data:

In [16]:
```
### Import Data
df = pd.read_csv('clean_combine.csv')
```

In [17]:
```
df['report_school_year'].value_counts()
```

Out[17]:
```
2019-20    2892
2018-19    2890
2020-21    2880
2017-18    1925
Name: report_school_year, dtype: int64
```

In [18]:
```
df['subgroup_name'].value_counts()
```

Out[18]:
```
All Students                                  1364
Female                                        1364
Male                                          1364
White                                         1364
Black                                          736
Hispanic                                       734
Asian/Pacific Islander                         722
American Indian/Alaska Native                  614
Hispanic or Latino                             613
Black or African American                      610
American Indian or Alaska Native               498
Asian or Native Hawaiian/Other Pacific Islander    362
Asian or Pacific Islander                      242
Name: subgroup_name, dtype: int64
```

In [19]:
```
df['subgroup_code'].value_counts()
```

Out[19]:
```
1    1364
2    1364
```

```
3    1364
8    1364
6    1347
5    1346
7    1326
4    1112
Name: subgroup_code, dtype: int64
```

In [20]:
```
df['county_name'].value_counts()
```

Out[20]:
```
ALBANY       176
MONROE       176
NEW YORK     176
BRONX        176
KINGS        176
            ...
OTSEGO       158
YATES        157
LEWIS        156
SCHOHARIE    151
HAMILTON     107
Name: county_name, Length: 62, dtype: int64
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10587 entries, 0 to 10586
Data columns (total 28 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Unnamed: 0                   10587 non-null  int64
 1   report_school_year           10587 non-null  object
 2   aggregation_index            10587 non-null  int64
 3   aggregation_type             10587 non-null  object
 4   county_code                  10587 non-null  float64
 5   county_name                  10587 non-null  object
 6   membership_code              10587 non-null  int64
 7   membership_key               10587 non-null  int64
 8   membership_desc              10587 non-null  object
 9   subgroup_code                10587 non-null  int64
 10  subgroup_name                10587 non-null  object
 11  enroll_cnt                   10587 non-null  object
 12  grad_cnt                     10587 non-null  object
 13  grad_pct                     10587 non-null  object
 14  local_cnt                    10587 non-null  object
 15  local_pct                    10587 non-null  object
 16  reg_cnt                      10587 non-null  object
 17  reg_pct                      10587 non-null  object
 18  reg_adv_cnt                  10587 non-null  object
 19  reg_adv_pct                  10587 non-null  object
 20  non_diploma_credential_cnt   10587 non-null  object
 21  non_diploma_credential_pct   10587 non-null  object
 22  still_enr_cnt                10587 non-null  object
 23  still_enr_pct                10587 non-null  object
 24  ged_cnt                      10587 non-null  object
 25  ged_pct                      10587 non-null  object
 26  dropout_cnt                  10587 non-null  object
 27  dropout_pct                  10587 non-null  object
dtypes: float64(1), int64(5), object(22)
memory usage: 2.3+ MB
```

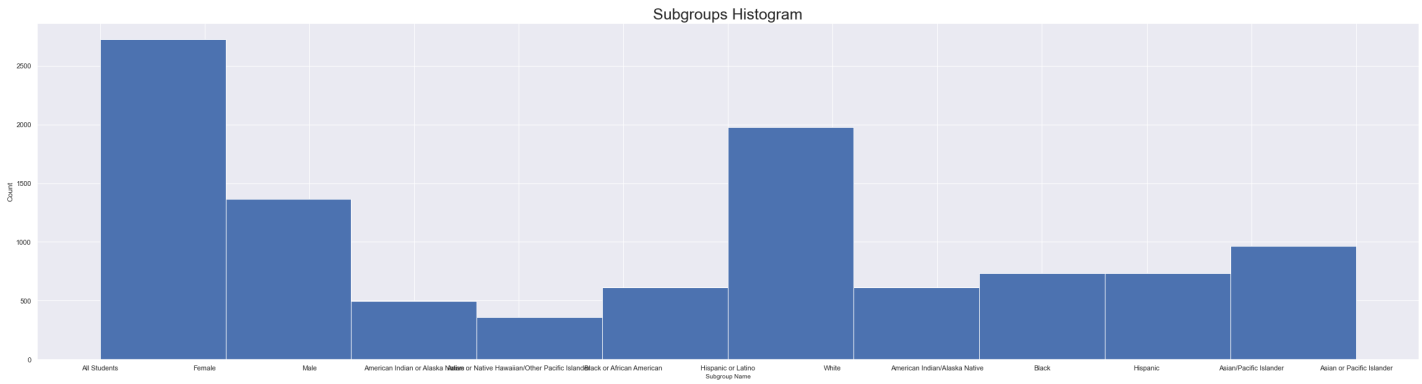
In [22]:
```
df.describe()
```

Out[22]:

	Unnamed: 0	aggregation_index	county_code	membership_code	membership_key	subgroup_code
count	10587.000000	10587.0	10587.000000	10587.000000	10587.000000	10587.000000
mean	5293.000000	2.0	34.473222	10.001511	149.669406	4.499669
std	3056.347984	0.0	20.526319	3.567200	15.485124	2.319321
min	0.000000	2.0	1.000000	6.000000	124.000000	1.000000
25%	2646.500000	2.0	16.000000	8.000000	139.000000	2.000000
50%	5293.000000	2.0	34.000000	9.000000	152.000000	5.000000
75%	7939.500000	2.0	53.000000	11.000000	166.000000	7.000000
max	10586.000000	2.0	68.000000	18.000000	174.000000	8.000000

In [23]:

```
plt.figure(figsize=(40,10))
plt.hist(df['subgroup_name'])
plt.title('Subgroups Histogram', fontsize = 25)
plt.xlabel('Subgroup Name', fontsize = 10)
plt.ylabel('Count')
plt.show()
```



Remove nulls for the target data, which is the bulk of the missing data:

In [24]:

```
df.count().isnull()
df2 = df.replace('-', np.nan)
```

In [27]:

```
print(df2.isnull().sum())

#the empty data needs to be removed because we need the
#numerical data for our classifier predictors. Any records with
#these data point empty are removed

df3 = df2.dropna()
```

```
Unnamed: 0                0
report_school_year        0
aggregation_index         0
aggregation_type          0
county_code               0
county_name               0
membership_code           0
membership_key            0
membership_desc           0
subgroup_code             0
subgroup_name             0
enroll_cnt              592
```

```
grad_cnt                        1358
grad_pct                        1358
local_cnt                       1358
local_pct                       1358
reg_cnt                         1358
reg_pct                         1358
reg_adv_cnt                     1358
reg_adv_pct                     1358
non_diploma_credential_cnt      1358
non_diploma_credential_pct      1358
still_enr_cnt                   1358
still_enr_pct                   1358
ged_cnt                         1358
ged_pct                         1358
dropout_cnt                     1358
dropout_pct                     1358
dtype: int64
```

In [28]:
```
df3['grad_pct']
```

Out[28]:
```
0          90%
1          93%
2          86%
3         100%
4          92%
          ...
10579      84%
10580      82%
10581      83%
10582      80%
10586      83%
Name: grad_pct, Length: 9229, dtype: object
```

In [29]:
```
print(df3.columns.tolist())
```

```
['Unnamed: 0', 'report_school_year', 'aggregation_index', 'aggregation_type', 'county_cod
e', 'county_name', 'membership_code', 'membership_key', 'membership_desc', 'subgroup_cod
e', 'subgroup_name', 'enroll_cnt', 'grad_cnt', 'grad_pct', 'local_cnt', 'local_pct', 'reg_
cnt', 'reg_pct', 'reg_adv_cnt', 'reg_adv_pct', 'non_diploma_credential_cnt', 'non_diploma_
credential_pct', 'still_enr_cnt', 'still_enr_pct', 'ged_cnt', 'ged_pct', 'dropout_cnt', 'd
ropout_pct']
```

Combine class names for subgroups

In [30]:
```
df3['subgroup_name']
```

Out[30]:
```
0                                          All Students
1                                                Female
2                                                  Male
3                       American Indian or Alaska Native
4          Asian or Native Hawaiian/Other Pacific Islander
                               ...
10579                                             White
10580                                      All Students
10581                                            Female
10582                                              Male
10586                                             White
Name: subgroup_name, Length: 9229, dtype: object
```

In [31]:
```
#subgroup names varied by graduation year so we must combine the relevant data
#that share the common subgroup code

df3['subgroup_name'] = df3['subgroup_name'].str.replace('/', ' or ')
```

```
df3['subgroup_name'] = df3['subgroup_name'].str.replace(' or Native Hawaiian', '')
df3['subgroup_name'] = df3['subgroup_name'].str.replace(' Other', '')
df3['subgroup_name'] = df3['subgroup_name'].str.replace(' or Latino', '')
df3['subgroup_name'] = df3['subgroup_name'].str.replace(' or African American', '')
```

Feature Creation

Create numerical version of all percentage data by removing the "%" and converting to floating type

In [32]:

```
#Since numerical percentage data contains a percentage sign at the end,
#this data is considered an object type. Here we recreate the fields but
#as floating types

df3['grad_pct_d'] = df3['grad_pct'].str.replace('%', '').astype('float')

df3['reg_pct_d'] = df3['reg_pct'].str.replace('%', '').astype('float')

df3['local_pct_d'] = df3['local_pct'].str.replace('%', '').astype('float')

df3['reg_adv_pct_d'] = df3['reg_adv_pct'].str.replace('%', '').astype('float')

df3['non_diploma_credential_pct_d'] = df3['non_diploma_credential_pct'].str.replace('%', '

df3['still_enr_pct_d'] = df3['still_enr_pct'].str.replace('%', '').astype('float')

df3['ged_pct_d'] = df3['ged_pct'].str.replace('%', '').astype('float')

df3['dropout_pct_d'] = df3['dropout_pct'].str.replace('%', '').astype('float')
```

In [33]:

```
df3[['grad_pct_d','dropout_pct_d','ged_pct_d','still_enr_pct_d',
     'non_diploma_credential_pct_d',
     'reg_adv_pct_d','reg_pct_d','local_pct_d']].head(10)
```

Out[33]:

	grad_pct_d	dropout_pct_d	ged_pct_d	still_enr_pct_d	non_diploma_credential_pct_d	reg_adv_pct_d	reg_pct_d	loc
0	90.0	7.0	0.0	2.0	1.0	44.0	40.0	
1	93.0	5.0	0.0	1.0	1.0	47.0	41.0	
2	86.0	9.0	0.0	2.0	1.0	41.0	40.0	
3	100.0	0.0	0.0	0.0	0.0	60.0	20.0	
4	92.0	7.0	0.0	1.0	0.0	64.0	26.0	
5	78.0	16.0	1.0	4.0	1.0	13.0	56.0	
6	85.0	12.0	1.0	2.0	1.0	23.0	55.0	
7	94.0	4.0	0.0	1.0	1.0	55.0	35.0	
8	90.0	6.0	0.0	3.0	1.0	48.0	38.0	
9	92.0	5.0	0.0	2.0	1.0	54.0	35.0	

Create Pandemic feature based on report_school_year

In [34]:

```
#We divide the records by years during pre-COVID and COVID
#so we can compare the data and perhaps find patterns/relationships
#between pandemic status and graduation rates

pan = {'2020-21' : 'COVID','2019-20' : 'Pre-COVID',
```

```
        '2018-19': 'Pre-COVID', '2017-18' : 'Pre-COVID'}
df3['pandemic'] = df3['report_school_year'].map(pan)
```

Binary version of the Pandemic feature. Positive is for COVID, zero for Pre-COVID

In [35]:
```
pan_num = {'COVID' : 1, 'Pre-COVID' : 0}
df3['pan_num'] = df3['pandemic'].map(pan_num)
```

In [37]:
```
#Verify data
df3[['pandemic','pan_num']]
```

Out[37]:

	pandemic	pan_num
0	COVID	1
1	COVID	1
2	COVID	1
3	COVID	1
4	COVID	1
...	...	...
10579	Pre-COVID	0
10580	Pre-COVID	0
10581	Pre-COVID	0
10582	Pre-COVID	0
10586	Pre-COVID	0

9229 rows × 2 columns

In [38]:
```
## Take the graduate percentage and bin it out

df3['grad_pct_grade'] = pd.cut(df3['grad_pct_d'],
                               bins=[0,50 , 60, 70 , 80 , 90,100],
                               labels=[0,1,2,3,4,5])
```

In [39]:
```
## Converting object type to integer where necessary.
df3[['ged_cnt','non_diploma_credential_cnt','grad_cnt','enroll_cnt',
     'local_cnt','reg_cnt']] = df3[['ged_cnt','non_diploma_credential_cnt',
                                    'grad_cnt','enroll_cnt', 'local_cnt',
                                    'reg_cnt']].astype('int')
```

In [40]:
```
### total ged, diploma, or other credential % to total enrollment count

ged_dip_cred_cnt = df3['ged_cnt']+df3['non_diploma_credential_cnt']+df3['grad_cnt']
df3['ged_diploma_cred_pct'] = ged_dip_cred_cnt / df3['enroll_cnt'] *100
```

In [41]:
```
df3['ged_diploma_cred_pct'] = df3['ged_diploma_cred_pct'].round(2)
```

In [42]:
```
df3[['enroll_cnt','ged_diploma_cred_pct']]
```

Out[42]:

	enroll_cnt	ged_diploma_cred_pct
0	3069	90.81
1	1536	93.62
2	1533	88.00
3	5	100.00
4	266	92.11
...	...	...
10579	165	85.45
10580	213	83.57
10581	96	86.46
10582	117	81.20
10586	199	84.92

9229 rows × 2 columns

In [43]:
```
df3.to_csv(prj_path + '/final_clean_grad_data.csv')
```

Statistical Exploratory Analysis of Data

In [44]:
```
GRAD_data = df3.copy()
```

In [45]:
```
GRAD_data['report_school_year'].unique()
```

Out[45]:
```
array(['2020-21', '2019-20', '2017-18', '2018-19'], dtype=object)
```

In [46]:
```
GRAD_data['county_code'].unique()
```

Out[46]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 31., 32., 33., 34., 35., 40., 41., 42., 43., 44., 45.,
       46., 47., 48., 49., 50., 51., 52., 53., 54., 55., 56., 57., 58.,
       59., 60., 61., 62., 63., 64., 65., 66., 67., 68.])
```

In [47]:
```
GRAD_data['membership_code'].unique()
```

Out[47]:
```
array([ 6,  8,  9, 10, 11, 18], dtype=int64)
```

In [49]:
```
#We later find that membership_key is unique to the school year
#so this feature cannot be used to predict the Pre-COVID or COVID
#status of data, as it is directly linked to the year of that data

GRAD_data['membership_key'].unique()
```

Out[49]:
```
array([166, 167, 168, 169, 170, 174, 152, 153, 154, 155, 156, 160, 124,
       125, 126, 128, 138, 139, 140, 141, 142, 146], dtype=int64)
```

In [50]:
```
GRAD_data['grad_pct_d'].unique()
```

Out[50]:
```
array([ 90.,  93.,  86., 100.,  92.,  78.,  85.,  94.,  87.,  75.,  95.,
        81.,  80.,  91.,  84.,  79.,  89.,  73.,  50.,  83.,  70.,  67.,
        88.,  96.,  74.,  71.,  69.,  63.,  76.,  77.,  82.,  54.,  62.,
        61.,  64.,  72.,  66.,  59.,  65.,  60.,  68.,  40.,  55.,  56.,
        97.,  58.,  98.,  99.,  57.,  47.,  38.,  52.,  53.,  43.,  45.,
        46.,  49.,  51.])
```

In [51]:
```
GRAD_data['aggregation_index'].unique()

#since there is only one value for this feature, it will
#not be used as a predictor
```

Out[51]:
```
array([2], dtype=int64)
```

In [162…
```
GRAD_data[['local_pct_d', 'still_enr_pct_d', 'reg_pct_d', 'reg_adv_pct_d',
           'ged_pct_d', 'dropout_pct_d', 'grad_pct_d']].describe()
```

Out[162…

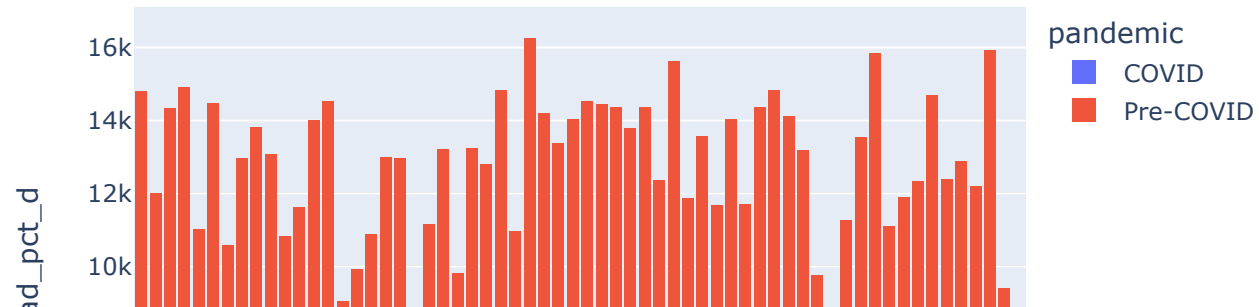
	local_pct_d	still_enr_pct_d	reg_pct_d	reg_adv_pct_d	ged_pct_d	dropout_pct_d	grad_pct_d
count	9229.000000	9229.000000	9229.000000	9229.000000	9229.000000	9229.000000	9229.000000
mean	6.515657	4.459855	44.608300	34.220392	0.653050	8.123849	85.323004
std	5.194849	5.107008	10.695752	15.891768	1.247497	6.053434	8.442802
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	38.000000
25%	3.000000	1.000000	39.000000	23.000000	0.000000	4.000000	81.000000
50%	6.000000	3.000000	45.000000	34.000000	0.000000	7.000000	87.000000
75%	8.000000	6.000000	50.000000	43.000000	1.000000	10.000000	91.000000
max	44.000000	60.000000	100.000000	100.000000	20.000000	60.000000	100.000000

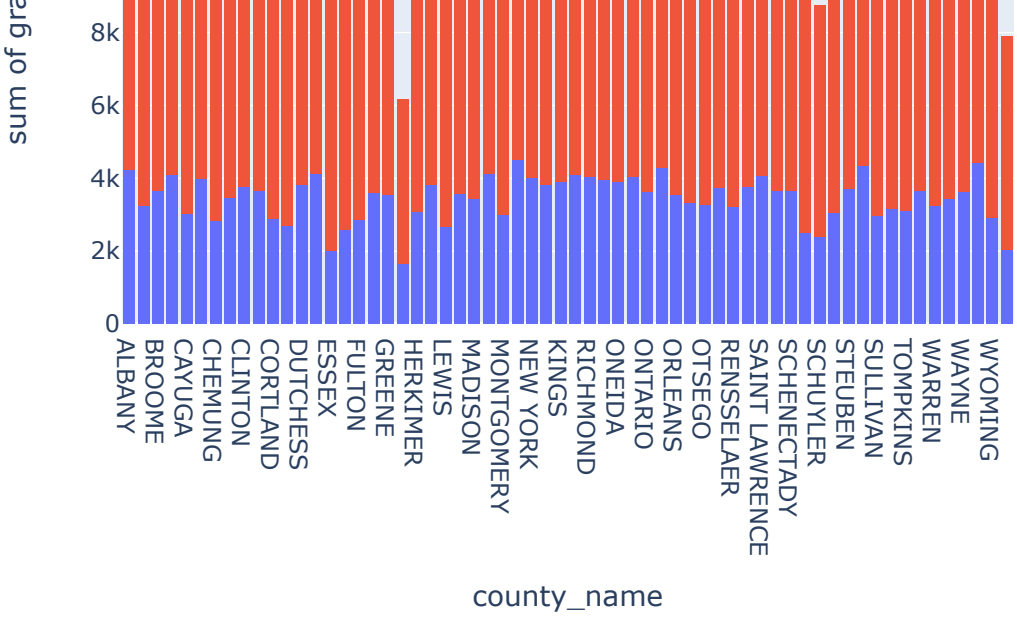
In [163…
```
GRAD_data['subgroup_name'].mode()
```

Out[163…
```
0    All Students
1          Female
2            Male
Name: subgroup_name, dtype: object
```

Stacked bar graphs for subgroups populations for pandemic classes

In [53]:
```
fig = px.histogram(df3, x='county_name', y='grad_pct_d', color='pandemic')
fig.show()
```

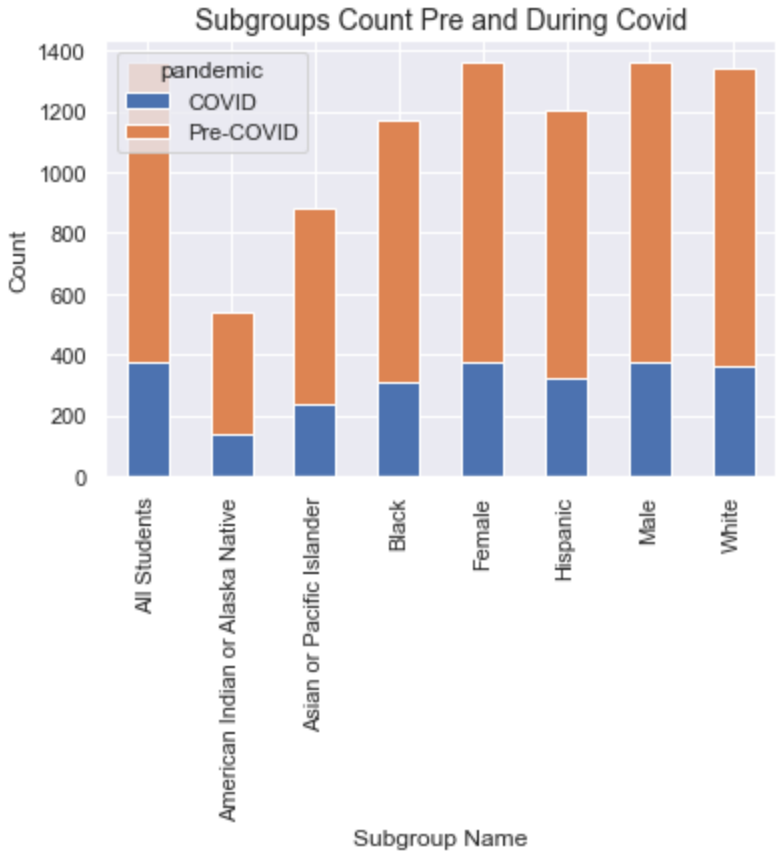




This reveals that 3/4 of our data is Pre-COVID and 1/4 is COVID data, which explains the larger red regions. However, this also reveals a similar pattern for all counties with regard to grad_pct_d

In [54]:
```
crosstab = pd.crosstab(df3['subgroup_name'], df3['pandemic'])
```

In [55]:
```
crosstab.plot(kind = 'bar', stacked  = True)
plt.title('Subgroups Count Pre and During Covid', fontsize = 14)
plt.xlabel('Subgroup Name')
plt.ylabel('Count')
plt.show()
```



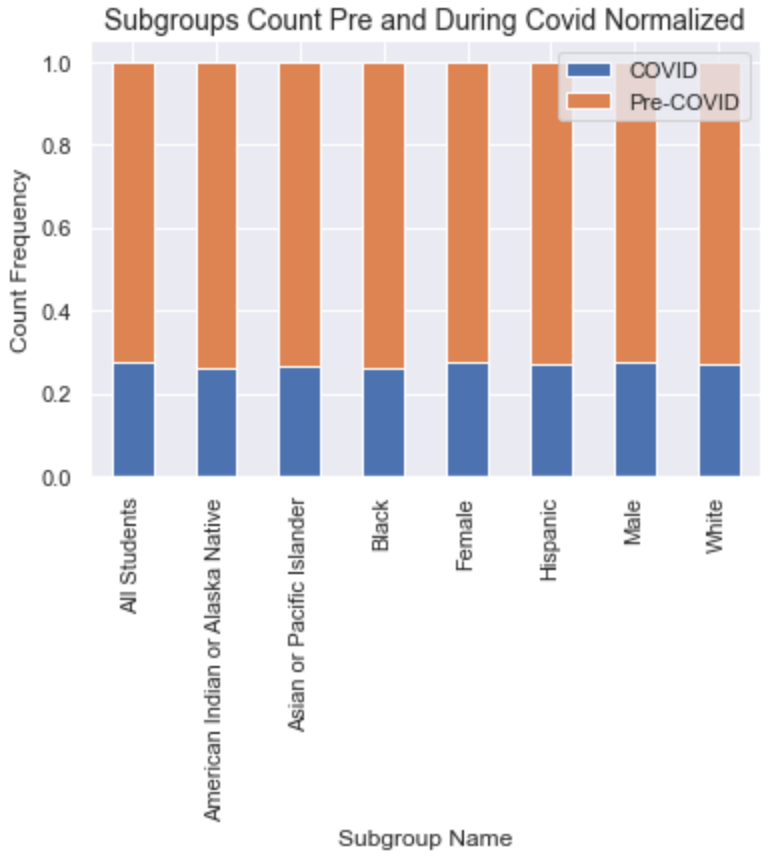
Normalized stacked bar:

In [56]:

```
crosstab_norm = crosstab.div(crosstab.sum(1), axis = 0)
```

In [57]:

```
crosstab_norm.plot(kind = 'bar', stacked  = True)
plt.title('Subgroups Count Pre and During Covid Normalized', fontsize = 14)
plt.xlabel('Subgroup Name')
plt.ylabel('Count Frequency')
plt.legend(loc = 'upper right')
plt.show()
```



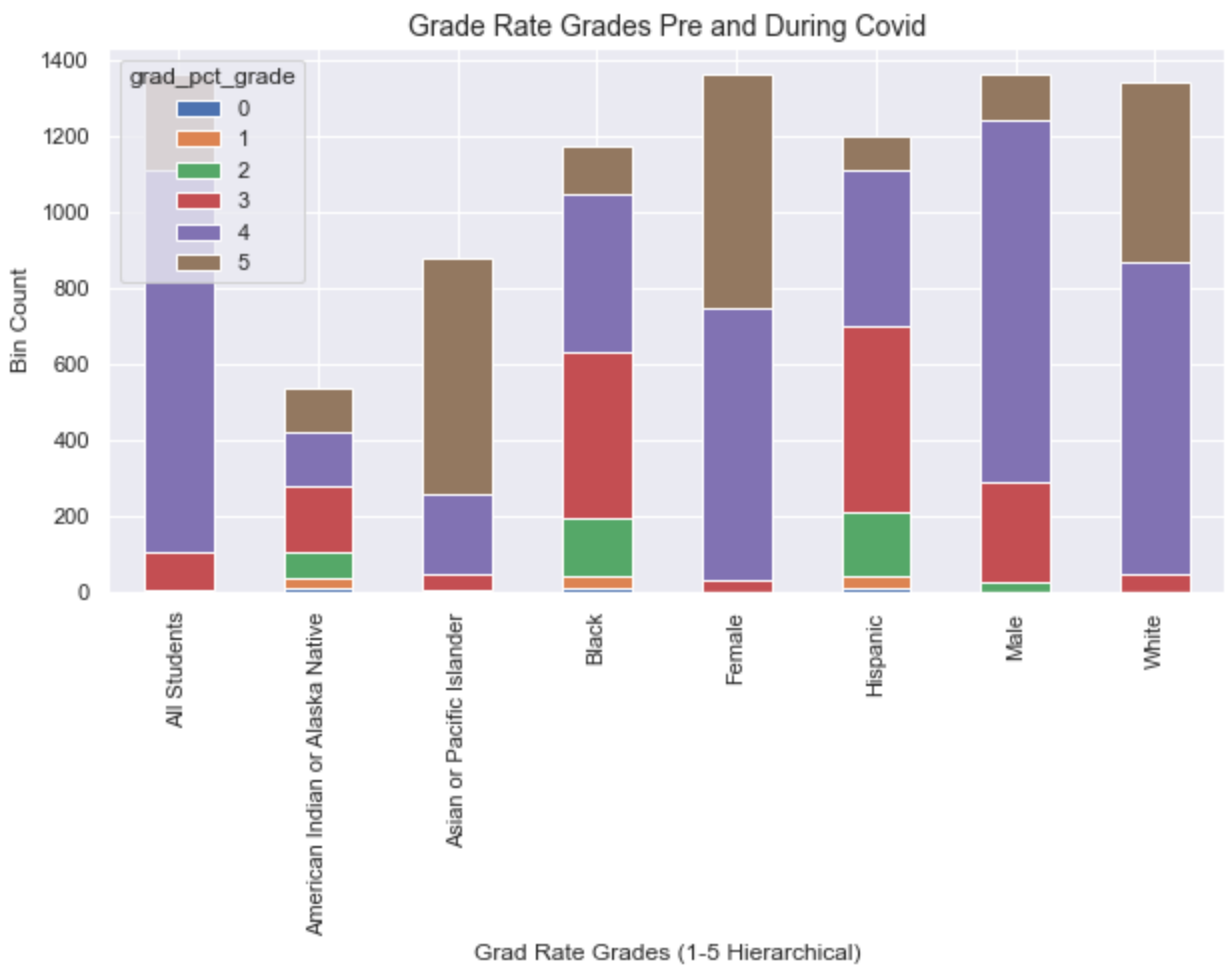
Again we see the 3/4 and 1/4 division of data based on 3 years being Pre-COVID and 1 year as COVID

In [58]:

```
crosstab_grad_sub = pd.crosstab(GRAD_data['subgroup_name'],
                                GRAD_data['grad_pct_grade'])
```

In [59]:

```
sns.set(rc={"figure.figsize":(10, 5)})
crosstab_grad_sub.plot(kind = 'bar', stacked  = True)
plt.title('Grade Rate Grades Pre and During Covid', fontsize = 14)
plt.xlabel('Grad Rate Grades (1-5 Hierarchical)')
plt.ylabel('Bin Count')
plt.show()
```

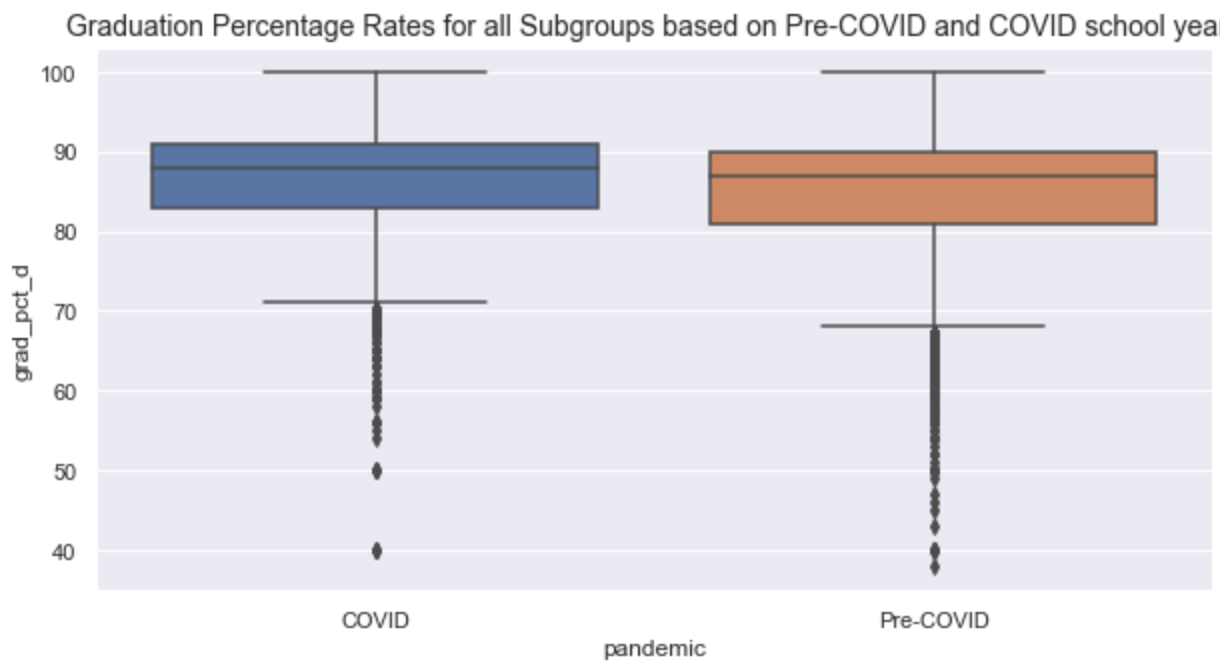


Graduation rate classes (5 indicating the highest percentage of graduates) per subgoups. The subgroups with less presentation from higher graduation percentage class may indicate financial needs/schools with greater public needs in neighborhoods lived in. As is seen when looking at the proportion of grade 3 and 4 graduation percentages for subgroups of Hispanic and Black populations. The Native American subgroup has a somewhat divided graduation percentage grade. Asian or Pacific Islanders, as well as Females have a high presentation of the highest graduation percentage class. While the rest of the groups have a proportionally higher presentation of level 4 or 5.

A little about our predictors:

In [60]:

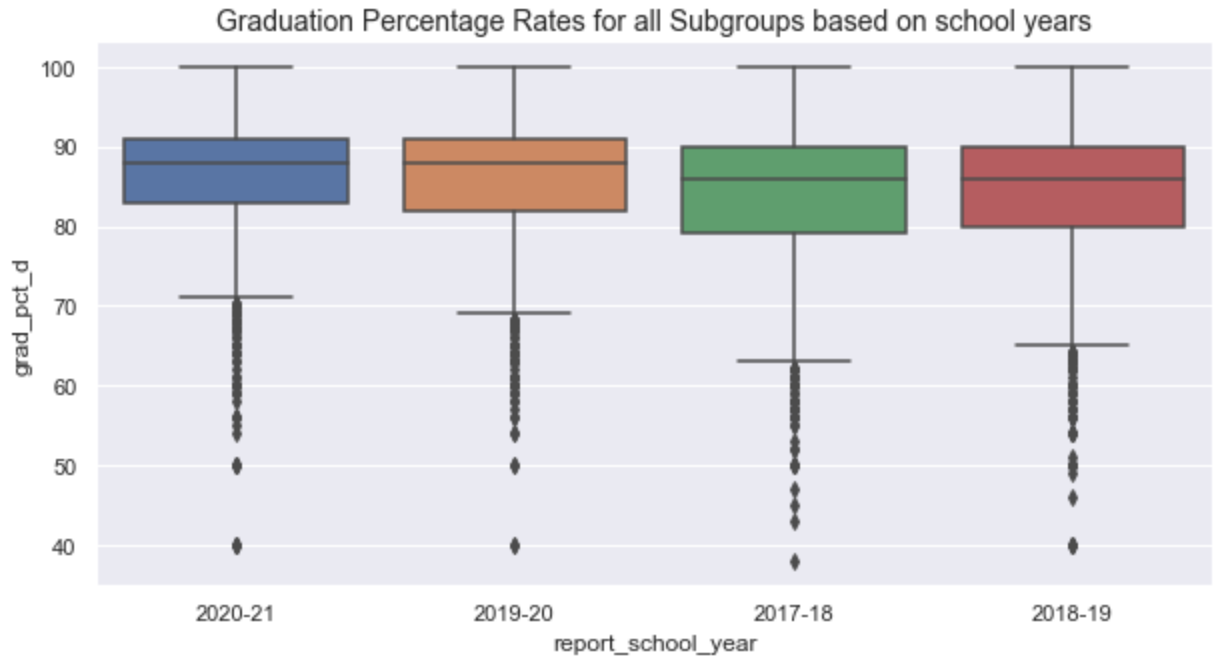
```
sns.boxplot( y=GRAD_data["grad_pct_d"], x=GRAD_data["pandemic"] );
plt.title('Graduation Percentage Rates for all Subgroups based on Pre-COVID and COVID scho
          fontsize = 14)

plt.show()
```



The boxplots reveal a smaller variance and higher general median and mean for graduation percentage in the COVID period than in the Pre-COVID. This is impressive given that the COVID data represents only 1 year versus the Pre-COVID, and can still give a marked difference. Since these are boxplots, we see a true median that is unaffected by the outliers.

In [61]:

```
sns.boxplot( y=GRAD_data["grad_pct_d"], x=GRAD_data["report_school_year"] );
plt.title('Graduation Percentage Rates for all Subgroups based on school years',
          fontsize = 14)

plt.show()
```

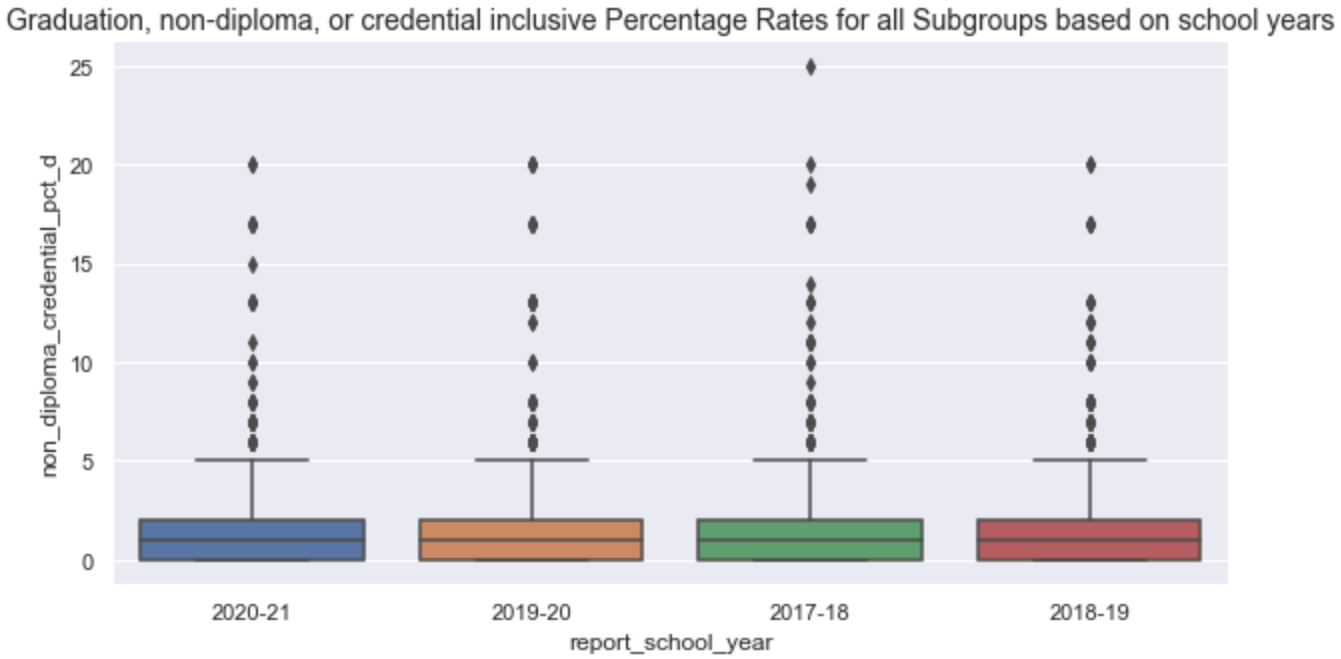


Here, we see that the year 2019-20 seems to behave similarly to 2020-21 with respect to graduation percentage, but since most of the school year was during the Pre-COVID timeframe, with only the last 3 months shutdown, it is possible the ones who were going to graduate anyway, simply went ahead to graduate and the pandemic might have had very little effect on this.

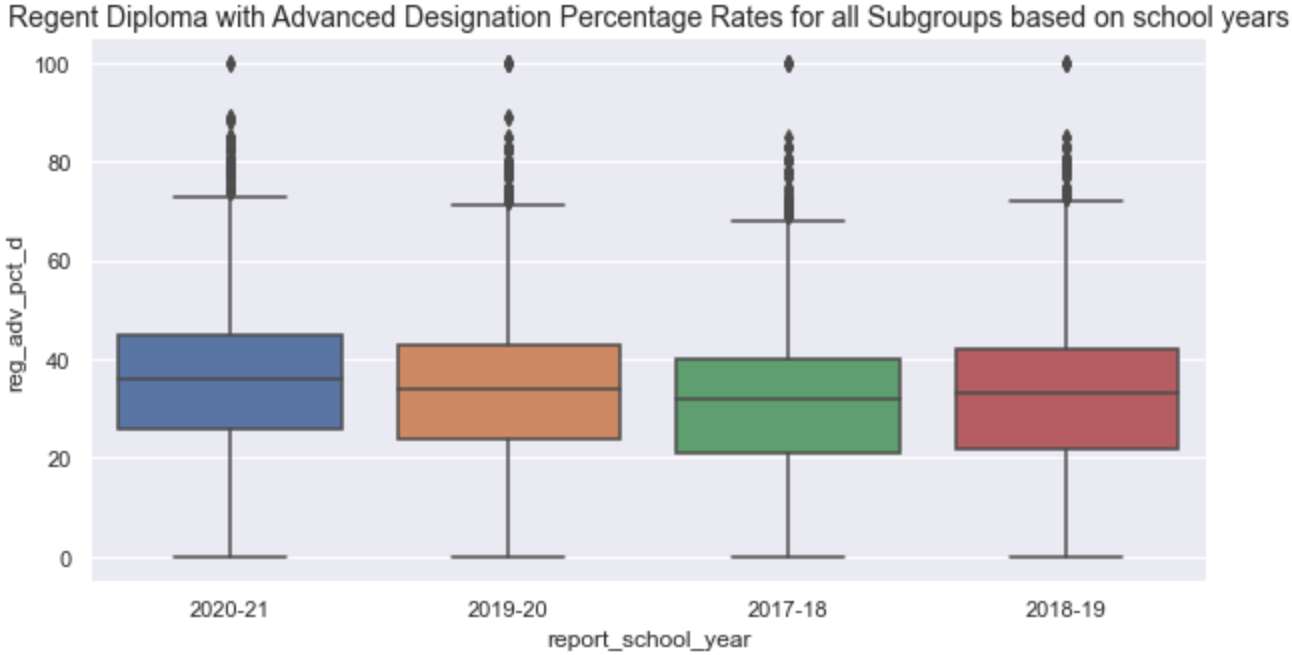
Therefore, we included 2019-20 school year as Pre-COVID.

By doing so, it significantly improved our classification problem, which is further proof that the data for 2019-20 fits more with the Pre-COVID timeframe data.

In [62]:
```
sns.boxplot( y=GRAD_data["non_diploma_credential_pct_d"], x=GRAD_data["report_school_year"
plt.title('Graduation, non-diploma, or credential inclusive Percentage Rates for all Subgr
          fontsize=14)

plt.show()
```

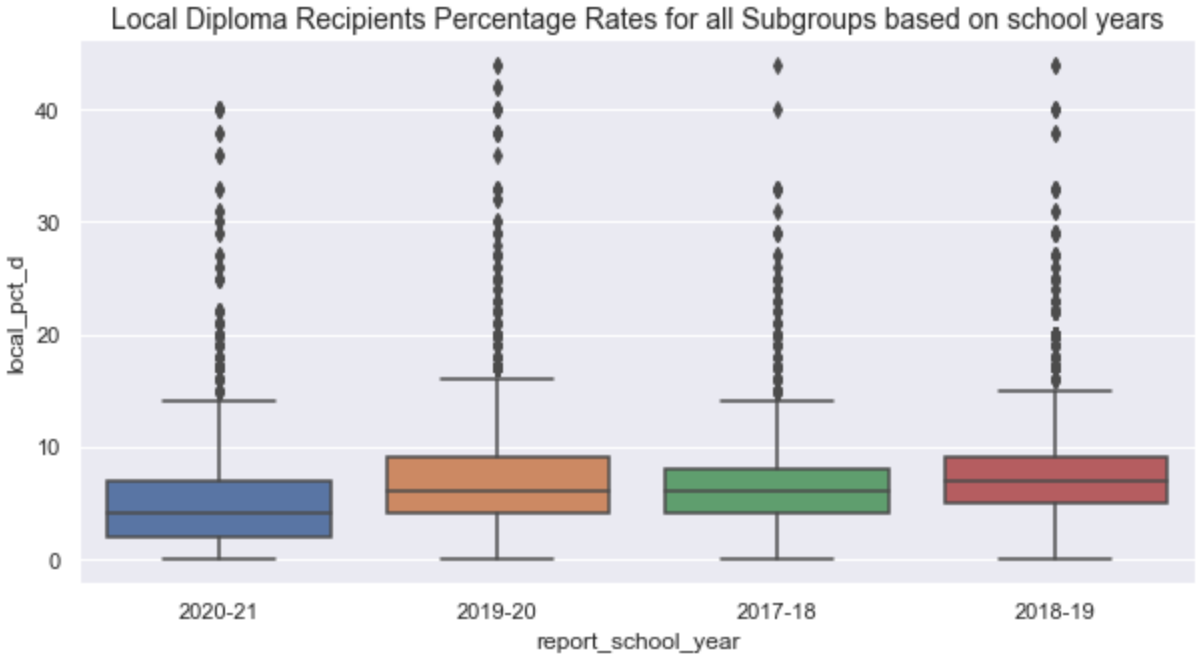


In [63]:
```
sns.boxplot( y=GRAD_data["reg_adv_pct_d"], x=GRAD_data["report_school_year"] );
plt.title('Regent Diploma with Advanced Designation Percentage Rates for all Subgroups bas
          fontsize=14)

plt.show()
```



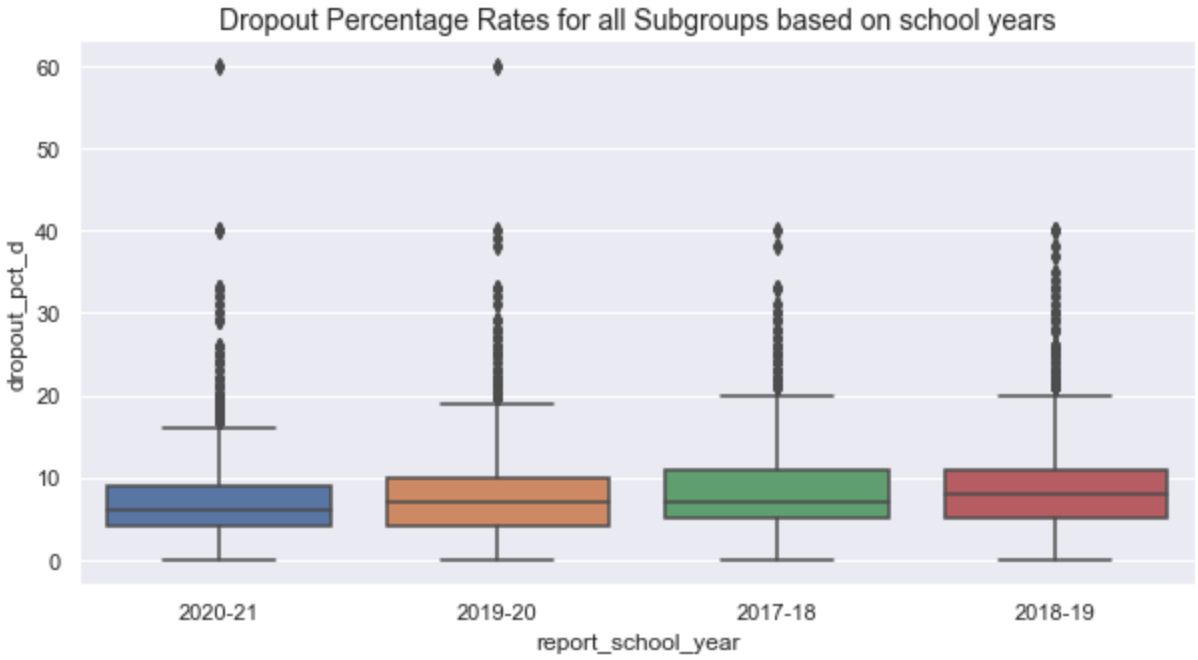
In [64]:
```
sns.boxplot( y=GRAD_data["local_pct_d"], x=GRAD_data["report_school_year"] );
plt.title('Local Diploma Recipients Percentage Rates for all Subgroups based on school yea
          fontsize=14)
```

```
plt.show()
```



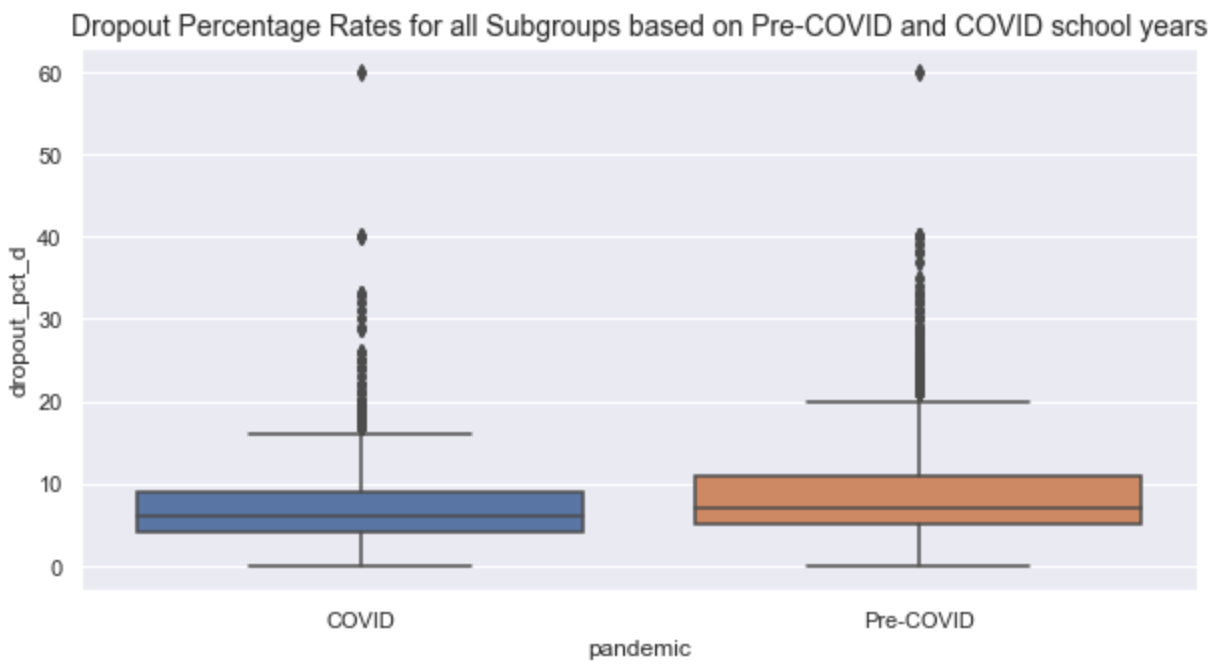
In [65]:

```
sns.boxplot( y=GRAD_data["dropout_pct_d"], x=GRAD_data["report_school_year"] );
plt.title('Dropout Percentage Rates for all Subgroups based on school years',
          fontsize=14)

plt.show()
```



In [66]:

```
sns.boxplot( y=GRAD_data["dropout_pct_d"], x=GRAD_data["pandemic"] );
plt.title('Dropout Percentage Rates for all Subgroups based on Pre-COVID and COVID school
          fontsize=14)

plt.show()
```

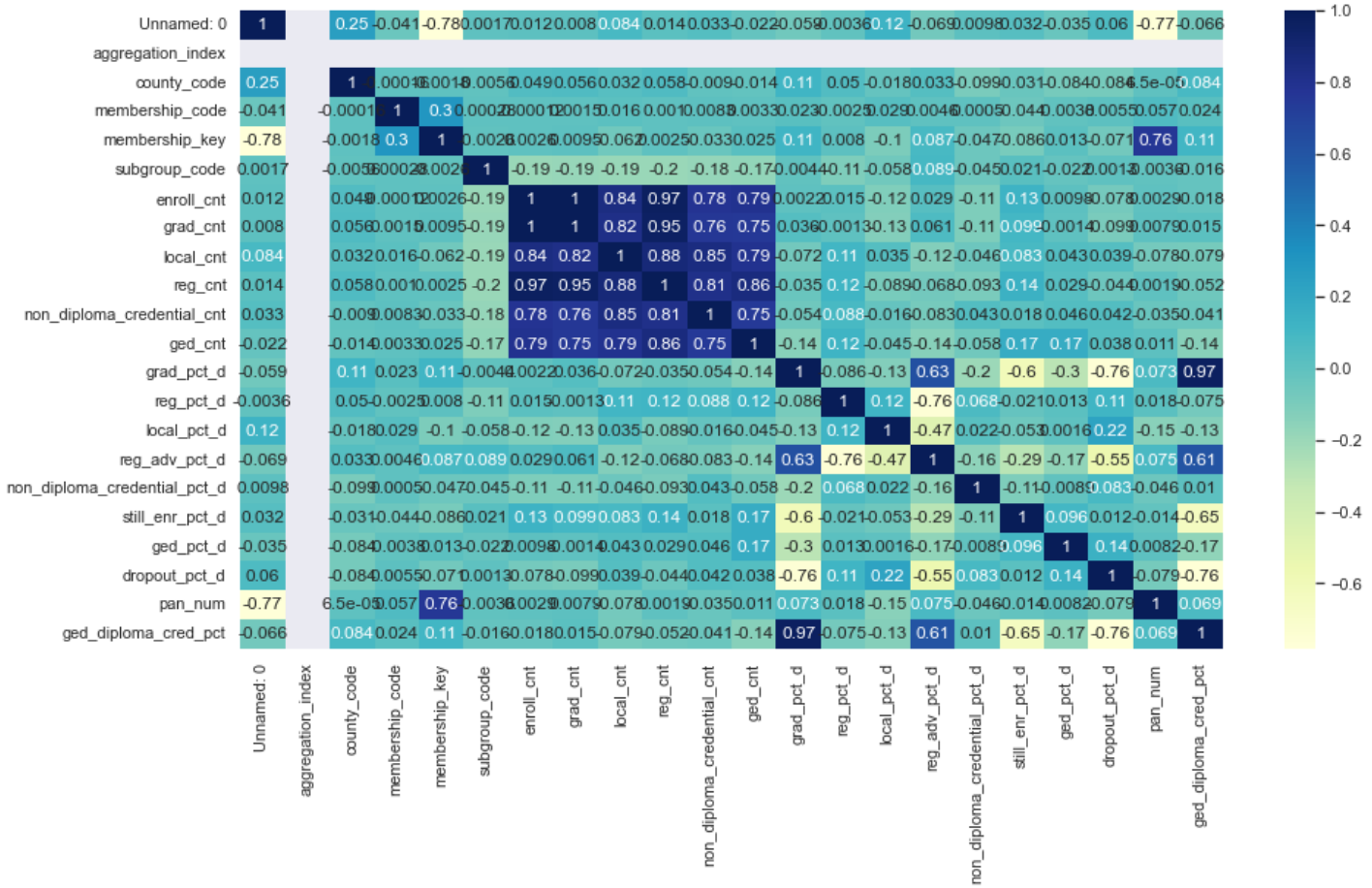


As can be perhaps intuited, dropout rates decreased during COVID, along with the variation in dropout rates.

Correlations:

In [67]:

```
# plot the heatmap and annotation on it
#Correlation matrix
corr_matrix_test = GRAD_data.corr()
sns.set(rc = {'figure.figsize':(15,8)})
sns.heatmap(corr_matrix_test, cmap="YlGnBu", annot=True)
plt.show()
```

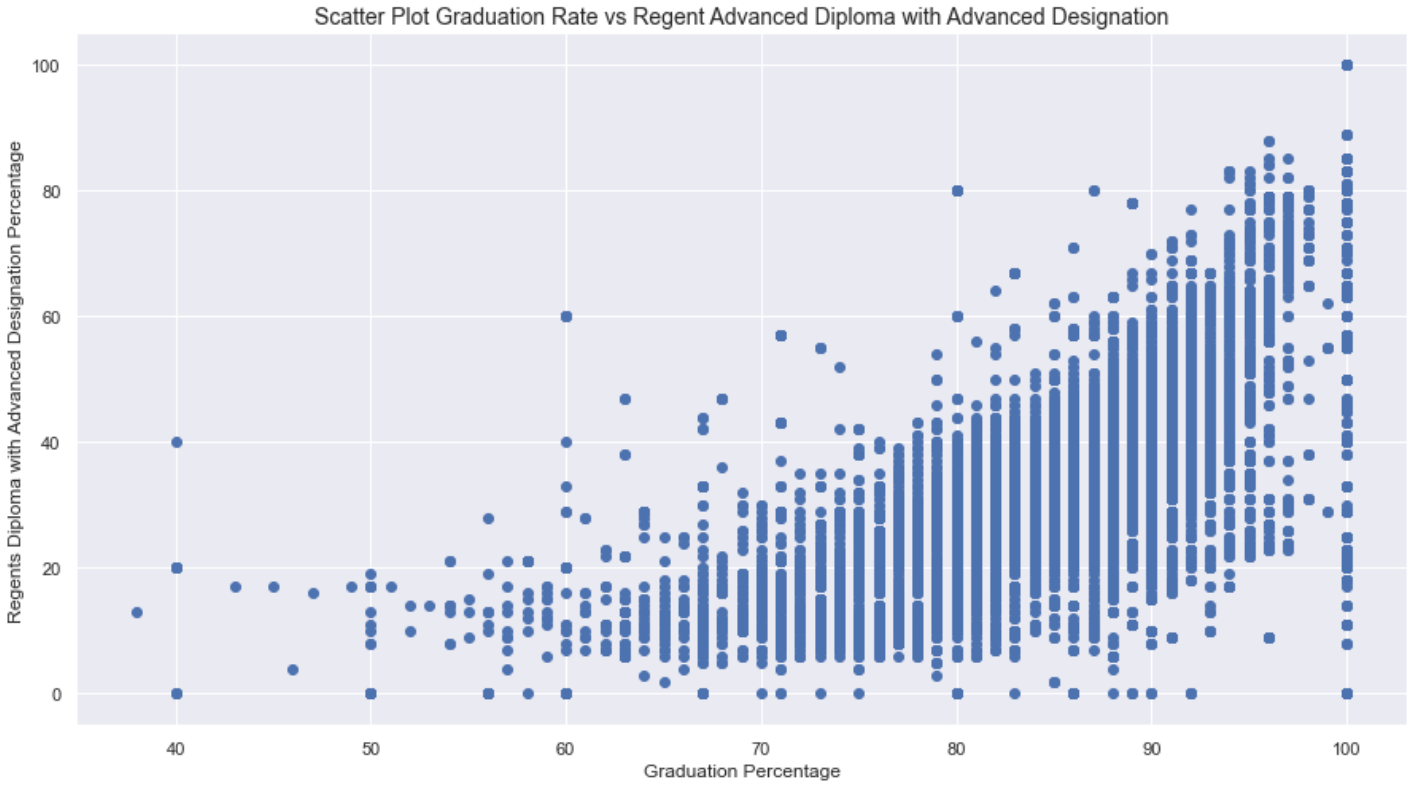


From the correlation heat map, we see:

- the counts are highly correlated with eachother.
- reg_adv_pct_d is moderately (0.63) correlated with grad_pct_d. Reg_Adv_pct_d is the percentage of graduates that recieve regents diploma with advanced designation, so as graduation rates increase, so too does this percentage increase.
- grad_pct_d is also related to variables that were derived using graduation rates/counts like ged_diploma_cred_pct.
- Finally, grad_pct_d is also negatively correlated with dropout_pct_d

In [68]:
```
#Verifying the relationship between reg_adv_pct and grad_pct_d

plt.scatter(GRAD_data.grad_pct_d, GRAD_data.reg_adv_pct_d)
plt.title('Scatter Plot Graduation Rate vs Regent Advanced Diploma with Advanced Designati
          fontsize=14)
plt.xlabel('Graduation Percentage')
plt.ylabel('Regents Diploma with Advanced Designation Percentage')
```

Out[68]:
```
Text(0, 0.5, 'Regents Diploma with Advanced Designation Percentage')
```



There is a positive linear correlation between the Reg_ADV_pct and graduation rate.

The strong correlation is partly due to the fact that grad_pct_d includes both local_pct_d and reg_pct_d, which is somewhat related to reg_adv_pct_d, and these all use enroll_cnt in their derivation.

But the stronger correlation of grad_pct_d with reg_adv_pct_d might also be be because that in the cases where graduation rates were generally higher, you find more students getting the regents degree with an advanced designation.

Pre-COVID and COVID Graduation Rate Stacked Histograms

In [70]:
```
#create a normalized version of the grad_pct_d:
GRAD_data['grad_pct_norm'] = np.log1p(GRAD_data['grad_pct_d'])
```

In [71]:
```
data_PRE1=GRAD_data[(GRAD_data.pandemic == 'Pre-COVID')]['grad_pct_d']
```

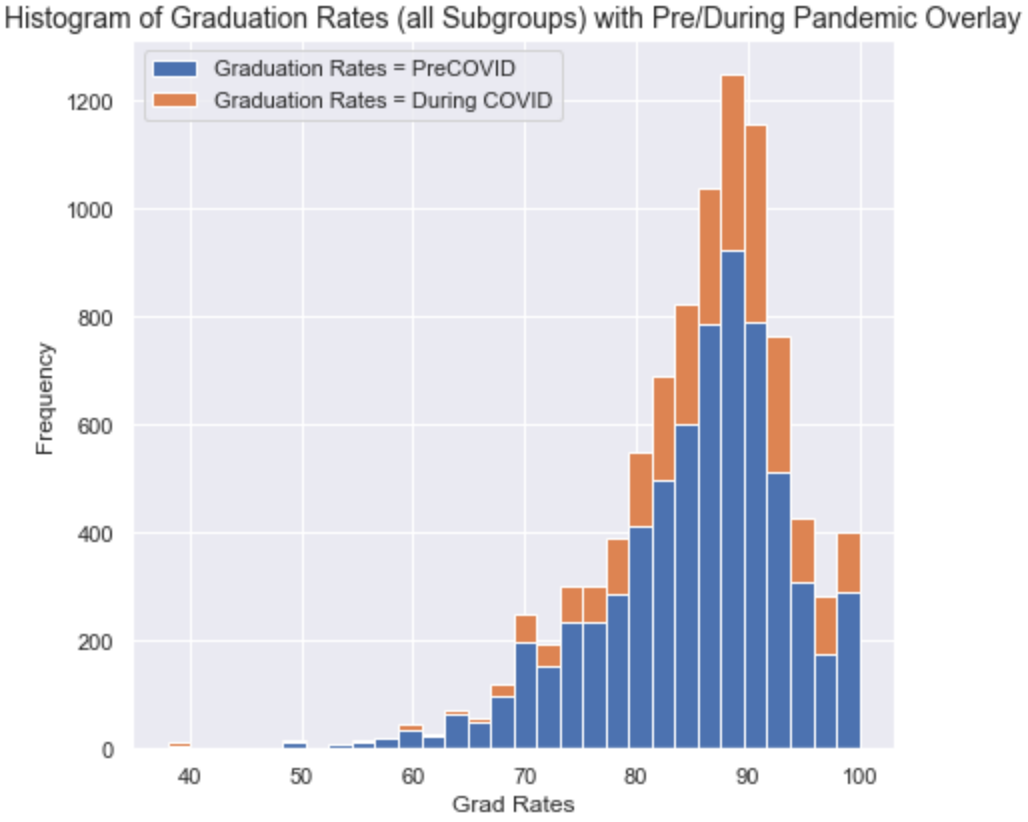
```
data_COVID1=GRAD_data[(GRAD_data.pandemic == 'COVID')]['grad_pct_d']

data_PRE1norm=GRAD_data[(GRAD_data.pandemic == 'Pre-COVID')]['grad_pct_norm']
data_COVID1norm=GRAD_data[(GRAD_data.pandemic == 'COVID')]['grad_pct_norm']
```

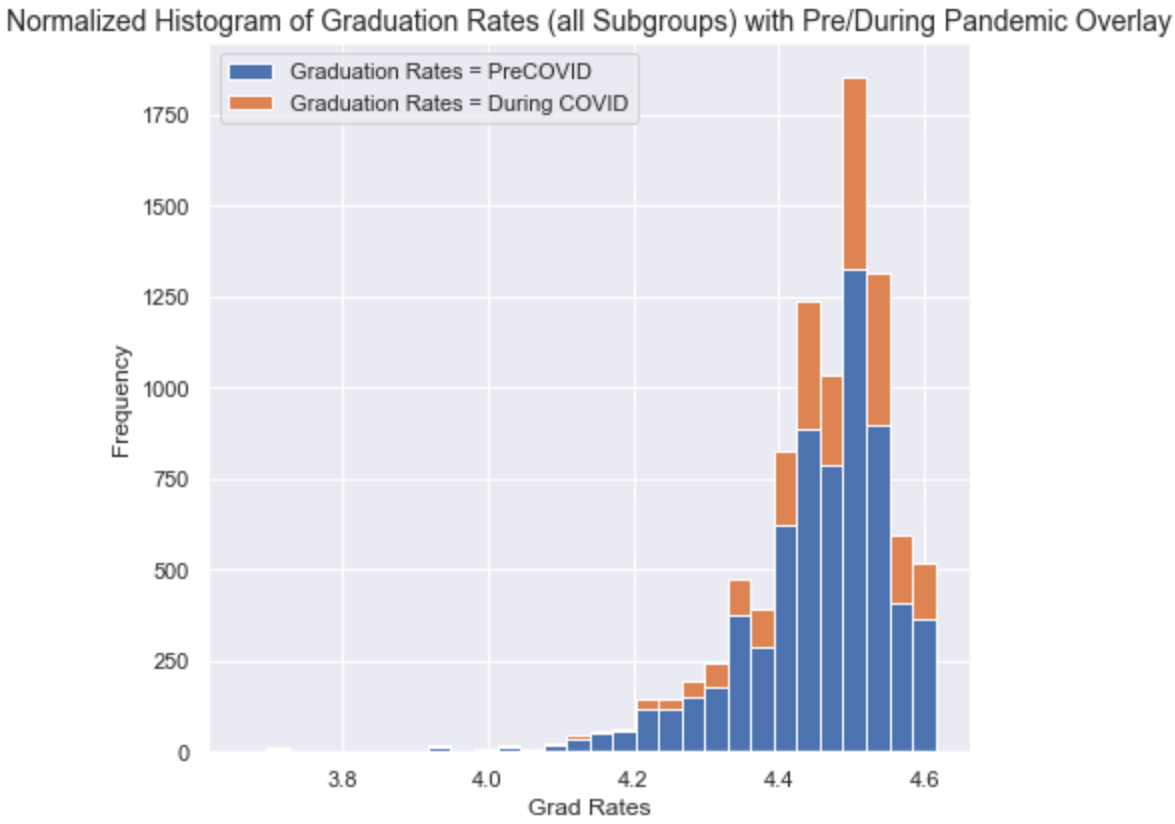
In [72]:

```
#Compare shape of histograms with raw graduation rates and normalized version
plt.figure(figsize = (15,15))
plt.subplots_adjust(hspace=0.3)
ax1 = plt.subplot(2,2,1)
(n, bins, patches) = plt.hist([data_PRE1, data_COVID1], bins=30, stacked = True)
plt.legend(['Graduation Rates = PreCOVID', 'Graduation Rates = During COVID'])
plt.title('Histogram of Graduation Rates (all Subgroups) with Pre/During Pandemic Overlay'
          fontsize = 14)
plt.xlabel('Grad Rates')
plt.ylabel('Frequency')
ax2 = plt.subplot(2,2,3)
(n, bins, patches) = plt.hist([data_PRE1norm, data_COVID1norm], bins=30, stacked = True)
plt.legend(['Graduation Rates = PreCOVID', 'Graduation Rates = During COVID'])
plt.title('Normalized Histogram of Graduation Rates (all Subgroups) with Pre/During Pander
          fontsize = 14)
plt.xlabel('Grad Rates')
plt.ylabel('Frequency')
plt.show
```

Out[72]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

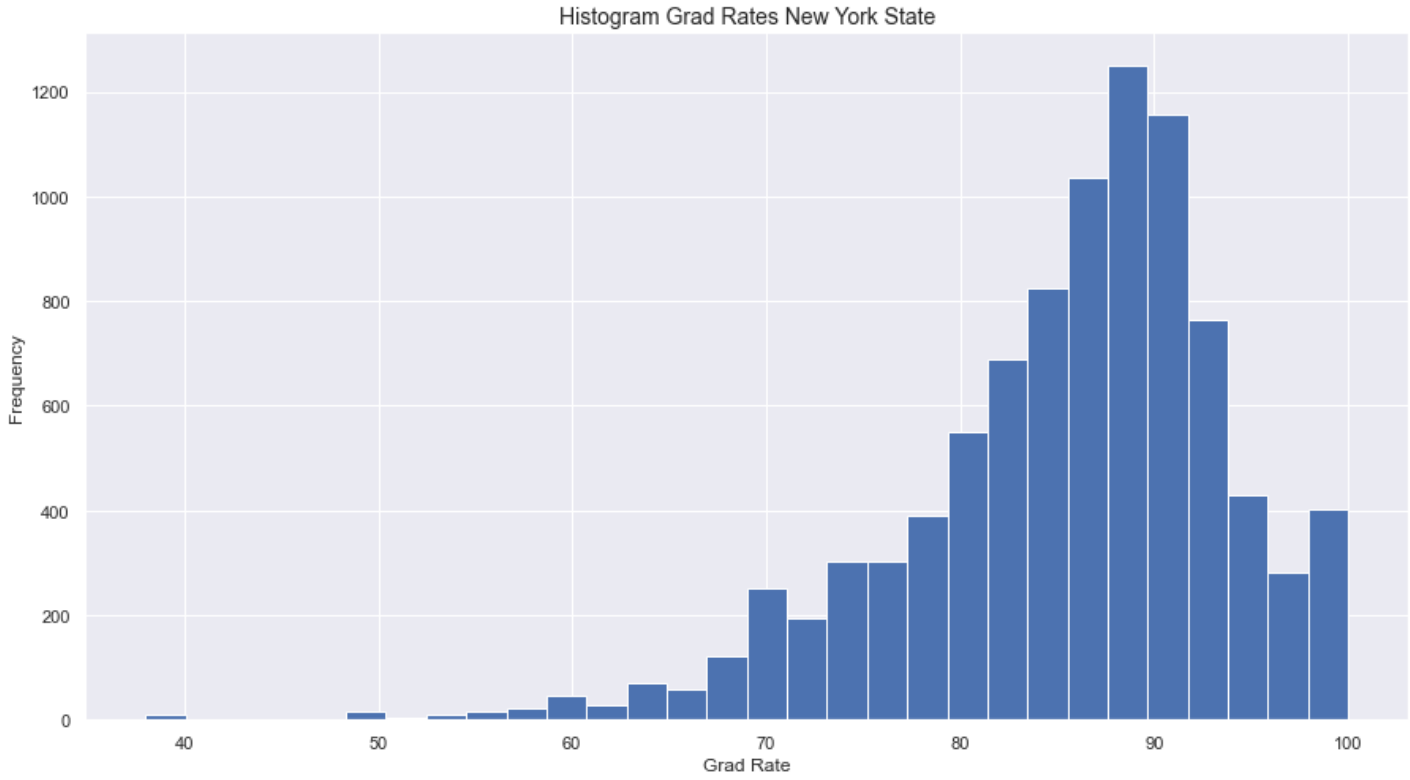




All graduation percentage data Histogram:

In [73]:
```
plt.hist([GRAD_data.grad_pct_d], bins=30)
plt.title('Histogram Grad Rates New York State', fontsize = 14)
plt.xlabel('Grad Rate')
plt.ylabel('Frequency')
```

Out[73]:
```
Text(0, 0.5, 'Frequency')
```

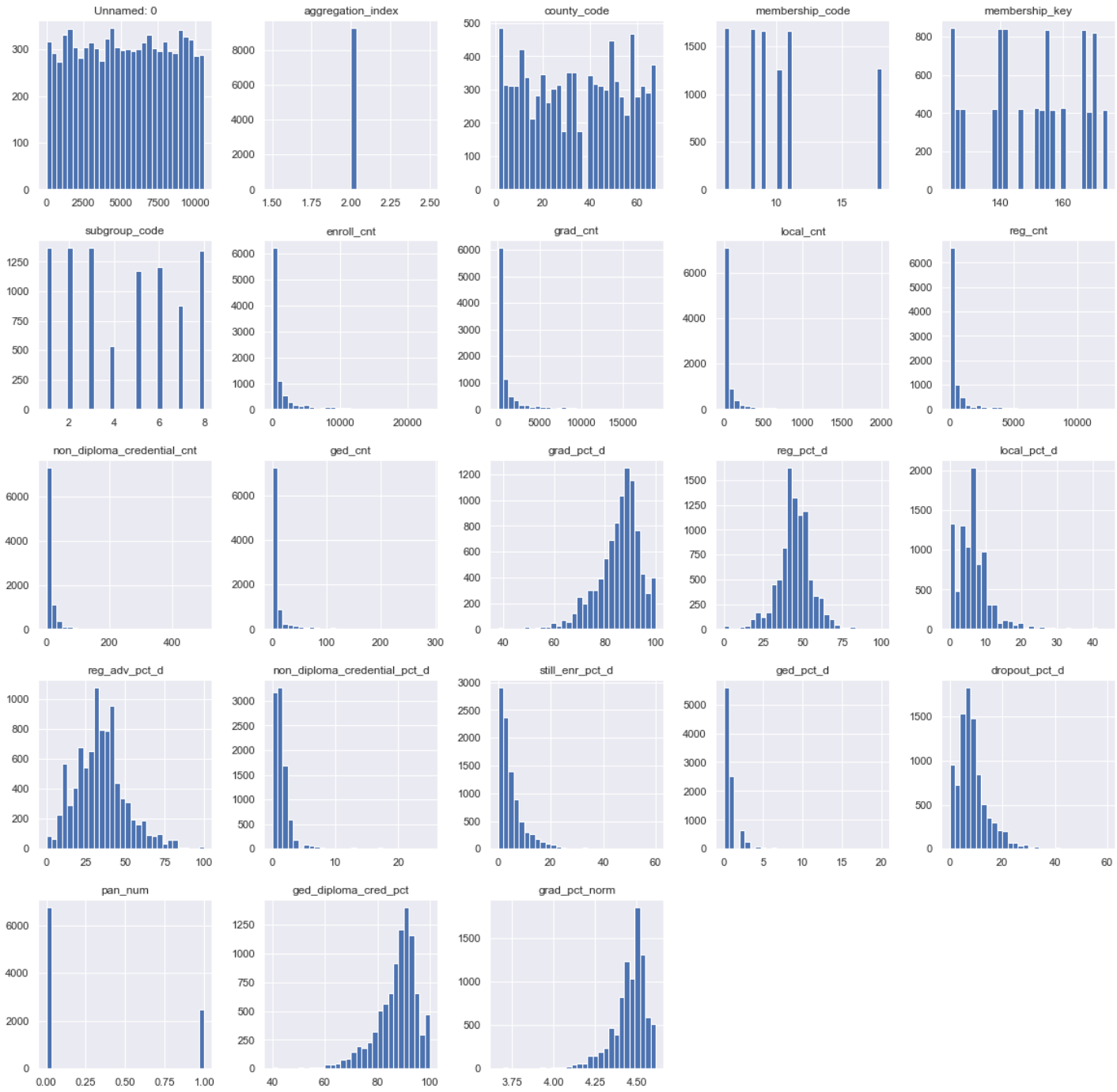


Left skew in the grad rates reveal that there are higher grad rates overall when we use complete dataset, which is good news as this means most students graduate in general.

Exhaustive Histograms for all Features:

In [74]:
```
GRAD_data.hist(bins=30, figsize=(20, 20))
```

Out[74]:
```
array([[<AxesSubplot:title={'center':'Unnamed: 0'}>,
        <AxesSubplot:title={'center':'aggregation_index'}>,
        <AxesSubplot:title={'center':'county_code'}>,
        <AxesSubplot:title={'center':'membership_code'}>,
        <AxesSubplot:title={'center':'membership_key'}>],
       [<AxesSubplot:title={'center':'subgroup_code'}>,
        <AxesSubplot:title={'center':'enroll_cnt'}>,
        <AxesSubplot:title={'center':'grad_cnt'}>,
        <AxesSubplot:title={'center':'local_cnt'}>,
        <AxesSubplot:title={'center':'reg_cnt'}>],
       [<AxesSubplot:title={'center':'non_diploma_credential_cnt'}>,
        <AxesSubplot:title={'center':'ged_cnt'}>,
        <AxesSubplot:title={'center':'grad_pct_d'}>,
        <AxesSubplot:title={'center':'reg_pct_d'}>,
        <AxesSubplot:title={'center':'local_pct_d'}>],
       [<AxesSubplot:title={'center':'reg_adv_pct_d'}>,
        <AxesSubplot:title={'center':'non_diploma_credential_pct_d'}>,
        <AxesSubplot:title={'center':'still_enr_pct_d'}>,
        <AxesSubplot:title={'center':'ged_pct_d'}>,
        <AxesSubplot:title={'center':'dropout_pct_d'}>],
       [<AxesSubplot:title={'center':'pan_num'}>,
        <AxesSubplot:title={'center':'ged_diploma_cred_pct'}>,
        <AxesSubplot:title={'center':'grad_pct_norm'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



Here we see the variables that are discrete versus continuous, with general left skew for graduation/ged percentages, and right skew for dropout percentage. The regents percentage is pretty centered and the regents with advanced designation is slightly right skewed meaning it is not that common to receive such designation.

Check if we are working with balanced data:

In [75]:
```
GRAD_data['pandemic'].value_counts()
```

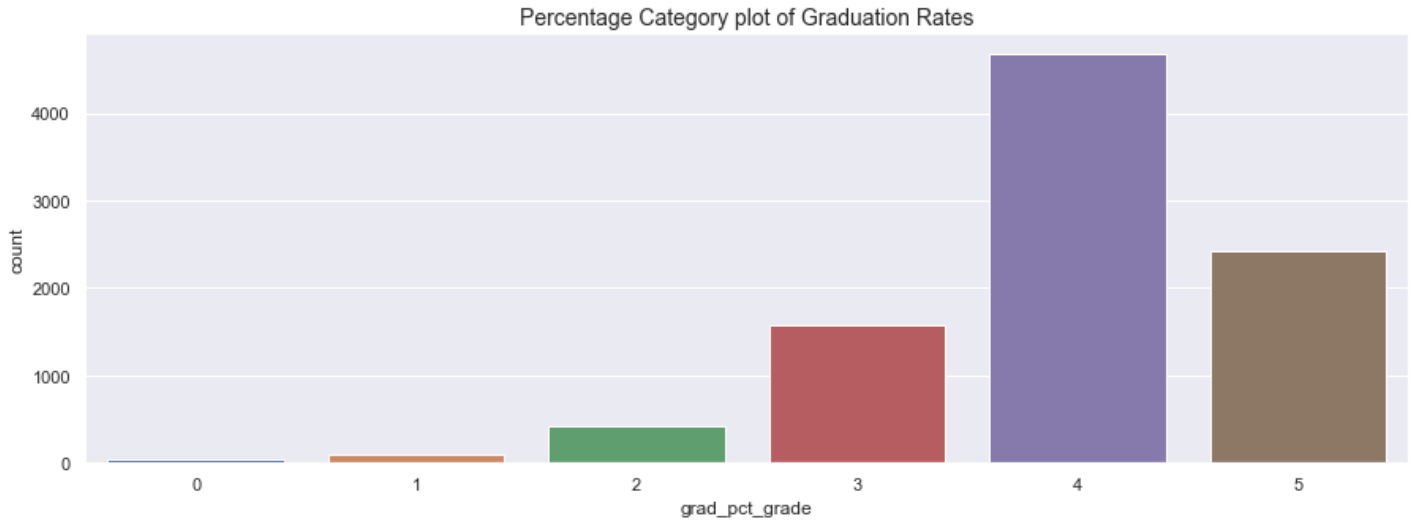
Out[75]:
```
Pre-COVID    6745
COVID        2484
Name: pandemic, dtype: int64
```

As we know from the designation done during data cleaning and feature creation, Pre-COVID constitutes about 3/4 data and COVID about 1/4 the data.

Bar chart for the grad percentage category

In [76]:
```
plt.figure(figsize=(15,5))
sns.countplot(GRAD_data['grad_pct_grade'],label="Count")
```

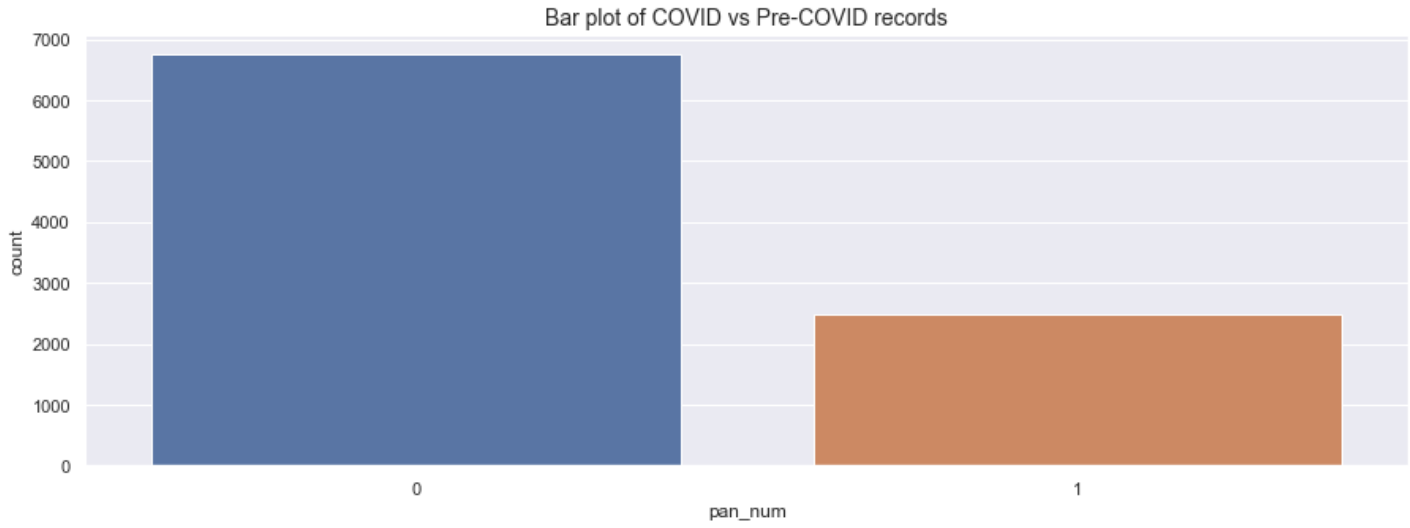
```
plt.title('Percentage Category plot of Graduation Rates', fontsize = 14)
plt.show()
```



If we were to elect predicting or classifying based on Graduation rates, we would have a larger number of class 4 records, but still within 50% of the total dataset

Checking on the balance of our target feature *pan_num*

In [77]:
```
plt.figure(figsize=(15,5))
sns.countplot(GRAD_data['pan_num'],label="Count")
plt.title('Bar plot of COVID vs Pre-COVID records', fontsize = 14)
plt.show()
```



Baseline Model with Pan_Num as Target:

Because 75% of the records are considered Pre-COVID, and our classifier target is pan_num, the baseline model would correctly predict with an accuracy of 75% of the '0' records that are Pre-COVID, even if it predicted all '0's. Therefore, a valid classifier must predict above this baseline accuracy based simply on the number of records found to be Pre-COVID

Summary of Observations Gathered from EDA:

- Many of the correlations found are based on the numerical features of the data having been derived using the same count variables such as enroll_cnt.
- We do see a difference in median and spread of the graduation rates data based on pandemic status.

- However, the imbalance in the number of records that correspond to Pre-COVID (75%) vs COVID (25%) does make some histograms appear as if there were more graduates/etc during Pre-COVID, but that is simply due to the count and number of records.
- Data indicates that in fact, the average graduation percentage was slightly higher, and more compact (less variation) for the COVID graduation data, than Pre-COVID

Classification Techniques to be Evaluated

- CART Decision Tree
- Neural Network
- Naive Bayes
- Bagging
- Boosting
- Random Forest

Setup Data

Split data into training and testing datasets:

In [78]:
```
GRAD_train, GRAD_test = train_test_split(GRAD_data, test_size = 0.30,
                                         random_state = 7)
```

In [79]:
```
train = GRAD_train.reset_index(drop=True)
test = GRAD_test.reset_index(drop=True)
```

Data to be used for Decision Tree, Naive Bayes, Bagging, Boosting, and Random Forest:

In [80]:
```
sub_dum = pd.get_dummies(train['subgroup_name'])
sub_dum_test = pd.get_dummies(test['subgroup_name'])
```

In [81]:
```
X_nb = pd.concat([sub_dum, train[['local_pct_d','reg_pct_d','reg_adv_pct_d',
                                  'non_diploma_credential_pct_d',
                                 'still_enr_pct_d','ged_pct_d',
                                  'dropout_pct_d']]], axis = 1)

y_nb = train['pan_num']

X_nb_test = pd.concat([sub_dum_test, test[['local_pct_d','reg_pct_d','reg_adv_pct_d',
                                           'non_diploma_credential_pct_d',
                                           'still_enr_pct_d','ged_pct_d',
                                           'dropout_pct_d']]], axis = 1)

y_nb_test = test['pan_num']
X_nb_test.head(10)
```

Out[81]:

	All Students	American Indian or Alaska Native	Asian or Pacific Islander	Black	Female	Hispanic	Male	White	local_pct_d	reg_pct_d	reg_adv_pct_d	non_
0	0	0	0	0	0	0	1	0	3.0	42.0	40.0	
1	0	0	0	1	0	0	0	0	38.0	38.0	13.0	

	All Students	American Indian or Alaska Native	Asian or Pacific Islander	Black	Female	Hispanic	Male	White	local_pct_d	reg_pct_d	reg_adv_pct_d	non_
2	0	1	0	0	0	0	0	0	12.0	53.0	10.0	
3	0	0	1	0	0	0	0	0	1.0	30.0	61.0	
4	1	0	0	0	0	0	0	0	9.0	48.0	34.0	
5	0	0	0	0	0	0	1	0	19.0	44.0	25.0	
6	0	0	1	0	0	0	0	0	1.0	23.0	73.0	
7	0	0	0	0	0	0	1	0	8.0	47.0	30.0	
8	0	0	1	0	0	0	0	0	1.0	22.0	73.0	
9	0	0	0	0	0	0	1	0	7.0	52.0	27.0	

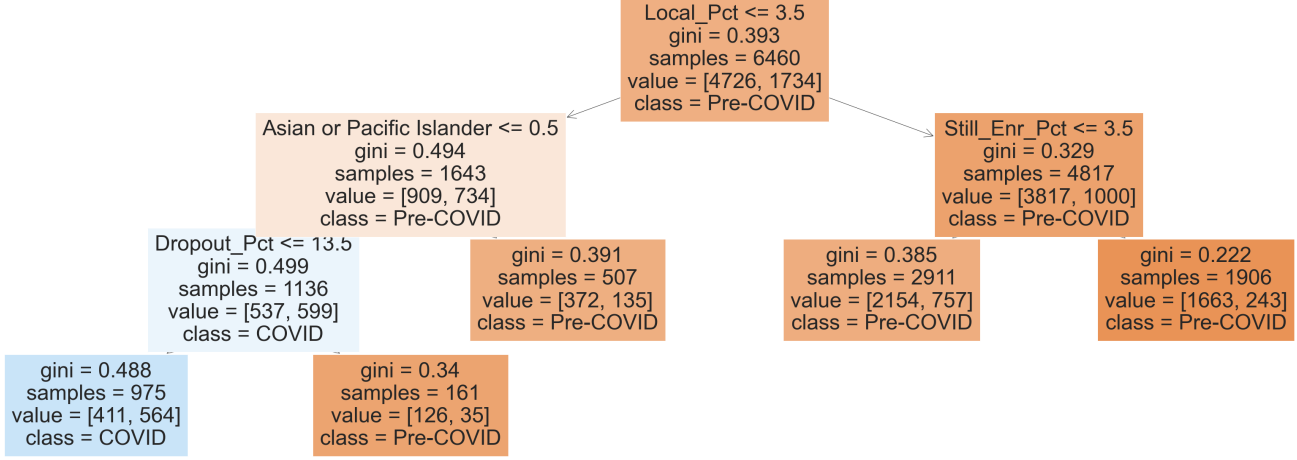
CART Decision Tree

In [82]:

```
X_names = ["All Students", "American Indian or Alaska Native",
           "Asian or Pacific Islander",
           "Black", "Female", "Hispanic", "Male", "White","Local_Pct",
           "Reg_Pct","Reg_Adv_Pct", 'Non_Diploma_Cred_Pct',
           'Still_Enr_Pct', 'GED_pct', 'Dropout_Pct']
y_names = ['Pre-COVID', 'COVID']
cart01 = DecisionTreeClassifier(criterion = "gini",
                                max_leaf_nodes=5).fit(X_nb,y_nb)

#dt = DecisionTreeClassifier().fit(X_CART, y_CART)
plt.figure(figsize=(60,20))
plot_tree(cart01,feature_names = X_names,
          class_names=y_names, filled = True)
```

Out[82]:

```
[Text(1860.0, 951.3000000000001, 'Local_Pct <= 3.5\ngini = 0.393\nsamples = 6460\nvalue =
[4726, 1734]\nclass = Pre-COVID'),
 Text(1116.0, 679.5, 'Asian or Pacific Islander <= 0.5\ngini = 0.494\nsamples = 1643\nvalu
e = [909, 734]\nclass = Pre-COVID'),
 Text(744.0, 407.70000000000005, 'Dropout_Pct <= 13.5\ngini = 0.499\nsamples = 1136\nvalue
= [537, 599]\nclass = COVID'),
 Text(372.0, 135.89999999999998, 'gini = 0.488\nsamples = 975\nvalue = [411, 564]\nclass =
COVID'),
 Text(1116.0, 135.89999999999998, 'gini = 0.34\nsamples = 161\nvalue = [126, 35]\nclass =
Pre-COVID'),
 Text(1488.0, 407.70000000000005, 'gini = 0.391\nsamples = 507\nvalue = [372, 135]\nclass
= Pre-COVID'),
 Text(2604.0, 679.5, 'Still_Enr_Pct <= 3.5\ngini = 0.329\nsamples = 4817\nvalue = [3817, 1
000]\nclass = Pre-COVID'),
 Text(2232.0, 407.70000000000005, 'gini = 0.385\nsamples = 2911\nvalue = [2154, 757]\nclas
s = Pre-COVID'),
 Text(2976.0, 407.70000000000005, 'gini = 0.222\nsamples = 1906\nvalue = [1663, 243]\nclas
s = Pre-COVID')]
```



Evaluate CART model on Test data:

In [83]:
```
predIncomeCART = cart01.predict(X_nb_test)
```

In [84]:
```
modelCART_cont=confusion_matrix(GRAD_test[['pan_num']],
                                predIncomeCART)
data = {'actual':GRAD_test['pan_num'],
        'prediction':predIncomeCART}


df = pd.DataFrame(data)
df.head()

contingency_matrix_modelCART = pd.crosstab(index=df['actual'],
                                           columns=df['prediction'])

print(contingency_matrix_modelCART)
```

```
prediction     0    1
actual
0           1832  187
1            514  236
```

In [85]:
```
TNmodel1=modelCART_cont[0][0]
FPmodel1=modelCART_cont[0][1]
FNmodel1=modelCART_cont[1][0]
TPmodel1=modelCART_cont[1][1]

TANmodel1=TNmodel1+FPmodel1
TAPmodel1=TPmodel1+FNmodel1
TPPmodel1=FPmodel1+TPmodel1
TPNmodel1=TNmodel1+FNmodel1

GTmodel1=TANmodel1+TAPmodel1

AccuracyM1=(TNmodel1+TPmodel1)/GTmodel1
ErrorRateM1=1-AccuracyM1
SensitivityM1=TPmodel1/(TAPmodel1)
RecallM1=SensitivityM1
SpecificityM1=TNmodel1/TANmodel1
PrecisionM1=TPmodel1/TPPmodel1
F1M1=2*PrecisionM1*RecallM1/(PrecisionM1 + RecallM1)
F2M1=5*(PrecisionM1*RecallM1)/((4*PrecisionM1)+RecallM1)
Fp5M1=(1.25)*(PrecisionM1*RecallM1)/((0.25*PrecisionM1)+RecallM1)
```

In [153…
```
header = ["Accuracy", "Error Rate", "Sensitivity", "Recall", "Specificity",
```

```
          "Precision", "F1", "F2", "F0.5"]
data1 = [["Accuracy", AccuracyM1], ["Error Rate", ErrorRateM1],
         ["Sensitivity", SensitivityM1],
         ["Recall", RecallM1], ["Specificity", SpecificityM1],
         ["Precision", PrecisionM1],
         ["F1", F1M1], ["F2", F2M1], ["F0.5", Fp5M1]]
col_names=["Measurement", "CART Model"]

ModelEvaluationTable = tabulate(data1, headers=col_names,
                                tablefmt="fancy_grid")
```

In [154…

```
print(ModelEvaluationTable)
```

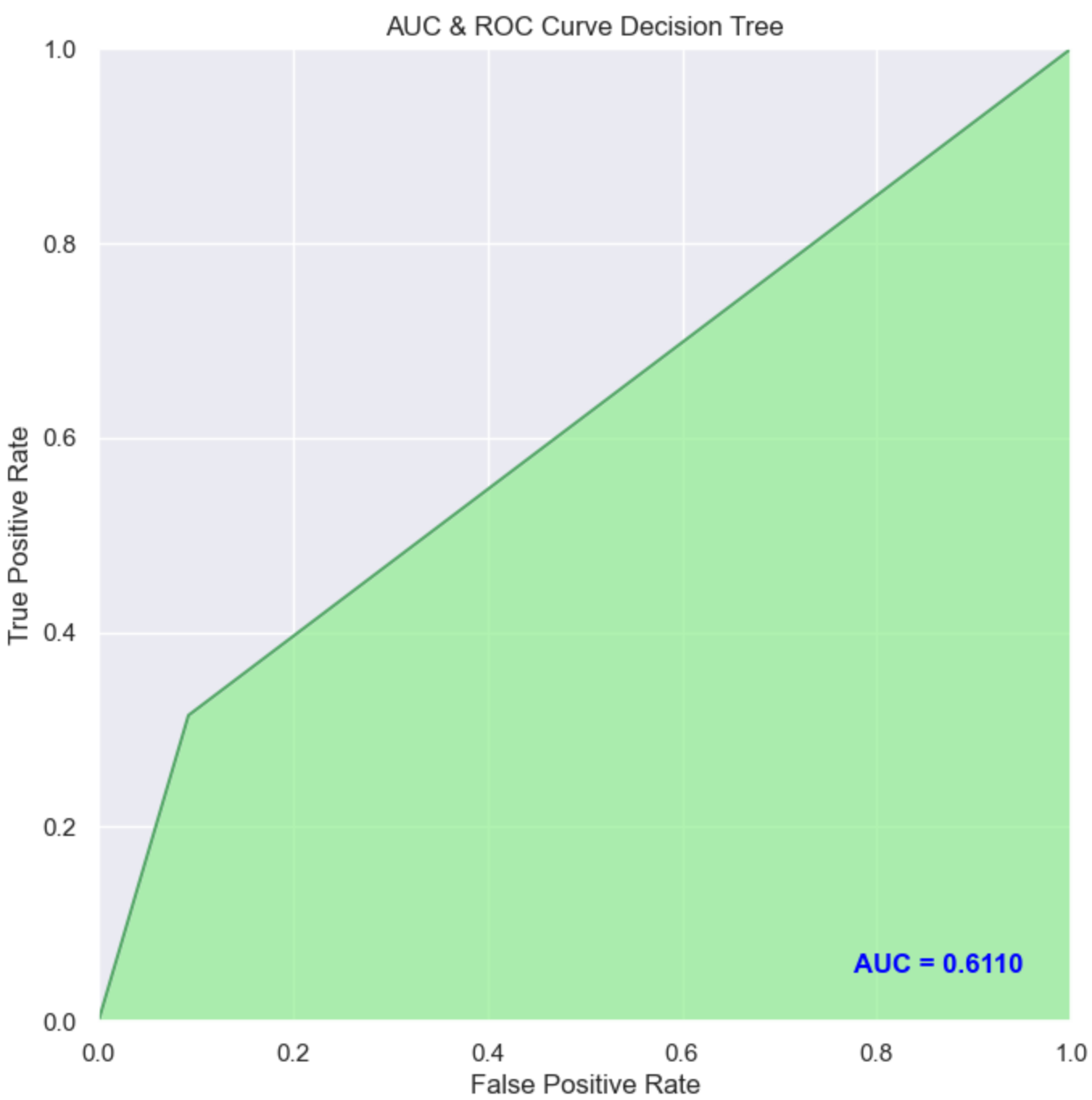
```
╒═══════════════╤═══════════╕
│ Measurement   │ CART Model│
╞═══════════════╪═══════════╡
│ Accuracy      │   0.74684 │
├───────────────┼───────────┤
│ Error Rate    │   0.25316 │
├───────────────┼───────────┤
│ Sensitivity   │   0.314667│
├───────────────┼───────────┤
│ Recall        │   0.314667│
├───────────────┼───────────┤
│ Specificity   │   0.90738 │
├───────────────┼───────────┤
│ Precision     │   0.55792 │
├───────────────┼───────────┤
│ F1            │   0.402387│
├───────────────┼───────────┤
│ F2            │   0.344727│
├───────────────┼───────────┤
│ F0.5          │   0.48321 │
╘═══════════════╧═══════════╛
```

Since we know that about 75% of the data is Pre-COVID and only 25 % of the data is COVID, this accuracy does not do well enough to predict the smaller class of COVID. In fact, it practically matches the baseline model with an accuracy of 75%.

Nothing changed when we oversample the COVID data.

In [88]:

```
aucdt = metrics.roc_auc_score(GRAD_test[['pan_num']], predIncomeCART)
false_positive_ratedt, true_positive_ratedt, thresoldsdt = metrics.roc_curve(GRAD_test[['p
plt.figure(figsize=(10, 8), dpi=100)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve Decision Tree")
plt.plot(false_positive_ratedt, true_positive_ratedt, 'g')
plt.fill_between(false_positive_ratedt, true_positive_ratedt, facecolor='lightgreen', alph
plt.text(0.95, 0.05, 'AUC = %0.4f' % aucdt, ha='right', fontsize=12, weight='bold', color=
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



The Area Under the Curve for the ROC graph should be ideally bigger than 0.5 and close to 1. An AUC of 0.5 would indicate a classifier that basically randomly generates predictions so has a 50-50 chance of a correct prediction.

The AUC for the CART decision tree is a bit higher than the 0.5 limit, but it is still very close and too low at 0.6 to be considered a candidate as a valid classifier for our problem

Neural Network

Predictor Data Preparation

In [89]:
```
df3['subgroup_name'].value_counts()
```

Out[89]:
```
All Students                    1364
Female                          1364
Male                            1364
White                           1342
Hispanic                        1203
Black                           1175
```

```
Asian or Pacific Islander            879
American Indian or Alaska Native     538
Name: subgroup_name, dtype: int64
```

In [90]:

```
scaler = MinMaxScaler()
scaled = scaler.fit_transform(train[['local_pct_d','reg_pct_d','reg_adv_pct_d',
                                     'non_diploma_credential_pct_d',
                                     'still_enr_pct_d','ged_pct_d',
                                     'dropout_pct_d']])
scaled_test = scaler.fit_transform(test[['local_pct_d','reg_pct_d','reg_adv_pct_d',
                                         'non_diploma_credential_pct_d',
                                         'still_enr_pct_d','ged_pct_d',
                                         'dropout_pct_d']])


sub_dum = pd.get_dummies(train['subgroup_name'])
sub_dum_test = pd.get_dummies(test['subgroup_name'])

sub_dum.columns = ['All_Students', 'American_Indian_Alaska_Native',
                   'Asian_Pacific_Islander', 'Black','Female','Hispanic','Male',
                  'White']
sub_dum_test.columns = ['All_Students', 'American_Indian_Alaska_Native',
                        'Asian_Pacific_Islander', 'Black','Female','Hispanic','Male',
                  'White']
```

In [91]:

```
scaled = pd.DataFrame(scaled)
scaled_test = pd.DataFrame(scaled_test)

scaled.columns = ['local_pct_mm','reg_pct_mm','reg_adv_pct_mm',
                  'non_diploma_credential_pct_mm','still_enr_pct_mm',
                                 'ged_pct_mm','dropout_pct_mm']
scaled_test.columns = ['local_pct_mm','reg_pct_mm','reg_adv_pct_mm',
                       'non_diploma_credential_pct_mm','still_enr_pct_mm',
                                 'ged_pct_mm','dropout_pct_mm']

X = pd.concat([sub_dum,scaled], axis=1)
y = train[['pan_num']]

X_test = pd.concat([sub_dum_test, scaled_test], axis=1)
y_test = test[['pan_num']]

X_test.head()
```

Out[91]:

	All_Students	American_Indian_Alaska_Native	Asian_Pacific_Islander	Black	Female	Hispanic	Male	White	local_p
0	0	0	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0

Create model

In [92]:

```
NN = keras.Sequential([
    keras.layers.Dense(32, input_shape=(15,), activation='relu'),
    keras.layers.Dense(2, activation='sigmoid')])

NN.compile(optimizer='adam',
```

```
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [93]:

```
X_arr = np.column_stack((X.All_Students.values, X.American_Indian_Alaska_Native.values,
                         X.Asian_Pacific_Islander.values,
                         X.Black.values, X.Female.values, X.Hispanic.values,
                         X.Male.values, X.White.values,
                         X.local_pct_mm.values,
                         X.reg_pct_mm.values, X.reg_adv_pct_mm.values,
                         X.non_diploma_credential_pct_mm.values,
                         X.still_enr_pct_mm.values, X.ged_pct_mm.values,
                         X.dropout_pct_mm.values))

X_test_arr = np.column_stack((X_test.All_Students.values,
                              X_test.American_Indian_Alaska_Native.values,
                              X_test.Asian_Pacific_Islander.values,
                         X_test.Black.values, X_test.Female.values,
                              X_test.Hispanic.values, X_test.Male.values,
                              X_test.White.values,
                         X_test.local_pct_mm.values,
                         X_test.reg_pct_mm.values, X_test.reg_adv_pct_mm.values,
                              X_test.non_diploma_credential_pct_mm.values,
                         X_test.still_enr_pct_mm.values, X_test.ged_pct_mm.values,
                              X_test.dropout_pct_mm.values))
```

In [94]:

```
NN.fit(X_arr, y, batch_size=64, epochs=25)
```

```
Epoch 1/25
101/101 [==============================] - 1s 1ms/step - loss: 0.6229 - accuracy: 0.6810
Epoch 2/25
101/101 [==============================] - 0s 941us/step - loss: 0.5827 - accuracy: 0.7316
Epoch 3/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5775 - accuracy: 0.7316
Epoch 4/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5739 - accuracy: 0.7316
Epoch 5/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5703 - accuracy: 0.7316
Epoch 6/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5668 - accuracy: 0.7316
Epoch 7/25
101/101 [==============================] - 0s 983us/step - loss: 0.5645 - accuracy: 0.7316
Epoch 8/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5620 - accuracy: 0.7314
Epoch 9/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5605 - accuracy: 0.7316
Epoch 10/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5589 - accuracy: 0.7319
Epoch 11/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5578 - accuracy: 0.7325
Epoch 12/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5565 - accuracy: 0.7314
Epoch 13/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5556 - accuracy: 0.7317
Epoch 14/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5546 - accuracy: 0.7324
Epoch 15/25
101/101 [==============================] - 0s 993us/step - loss: 0.5534 - accuracy: 0.7331
Epoch 16/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5527 - accuracy: 0.7320
Epoch 17/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5518 - accuracy: 0.7350
Epoch 18/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5508 - accuracy: 0.7336
```

```
Epoch 19/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5500 - accuracy: 0.7347
Epoch 20/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5491 - accuracy: 0.7370
Epoch 21/25
101/101 [==============================] - 0s 905us/step - loss: 0.5476 - accuracy: 0.7364
Epoch 22/25
101/101 [==============================] - 0s 997us/step - loss: 0.5469 - accuracy: 0.7398
Epoch 23/25
101/101 [==============================] - 0s 998us/step - loss: 0.5454 - accuracy: 0.7381
Epoch 24/25
101/101 [==============================] - 0s 967us/step - loss: 0.5451 - accuracy: 0.7418
Epoch 25/25
101/101 [==============================] - 0s 1ms/step - loss: 0.5438 - accuracy: 0.7424
```

Out[94]:
```
<keras.callbacks.History at 0x245b2109970>
```

In [95]:
```
NN.evaluate(X_test_arr, y_test['pan_num'])
```

Out[95]:
```
87/87 [==============================] - 0s 1ms/step - loss: 0.5629 - accuracy: 0.7299
[0.5628826022148132, 0.7298663854598999]
```

Evaluate Model on Test data

In [96]:
```
y_pred = NN.predict(X_test)
classes_x = np.argmax(y_pred,axis=1)
y_pred = np.round(y_pred).astype(int)
```

Results:

In [97]:
```
NN_pred_df = pd.DataFrame(y_pred)
pred = pd.DataFrame(classes_x)
```

In [98]:
```
nnpred = pd.crosstab(y_test.pan_num.values, classes_x,
                     rownames=['Actual'],
                     colnames=['Predicted'])
nnpred['Total'] = nnpred.sum(axis=1); nnpred.loc['Total'] = nnpred.sum()
nnpred
```

Out[98]:

Predicted	0	1	Total
Actual			
0	1995	24	2019
1	724	26	750
Total	2719	50	2769

In [99]:
```
TNnn = nnpred[0][0]
FPnn = nnpred[0][1]
FNnn = nnpred[1][0]
TPnn = nnpred[1][1]
TANnn = TNnn + FPnn
TAPnn = FNnn + TPnn
TPNnn = TNnn + FNnn
TPPnn = FPnn + TPnn
GTnn = TNnn + FPnn + FNnn + TPnn
```

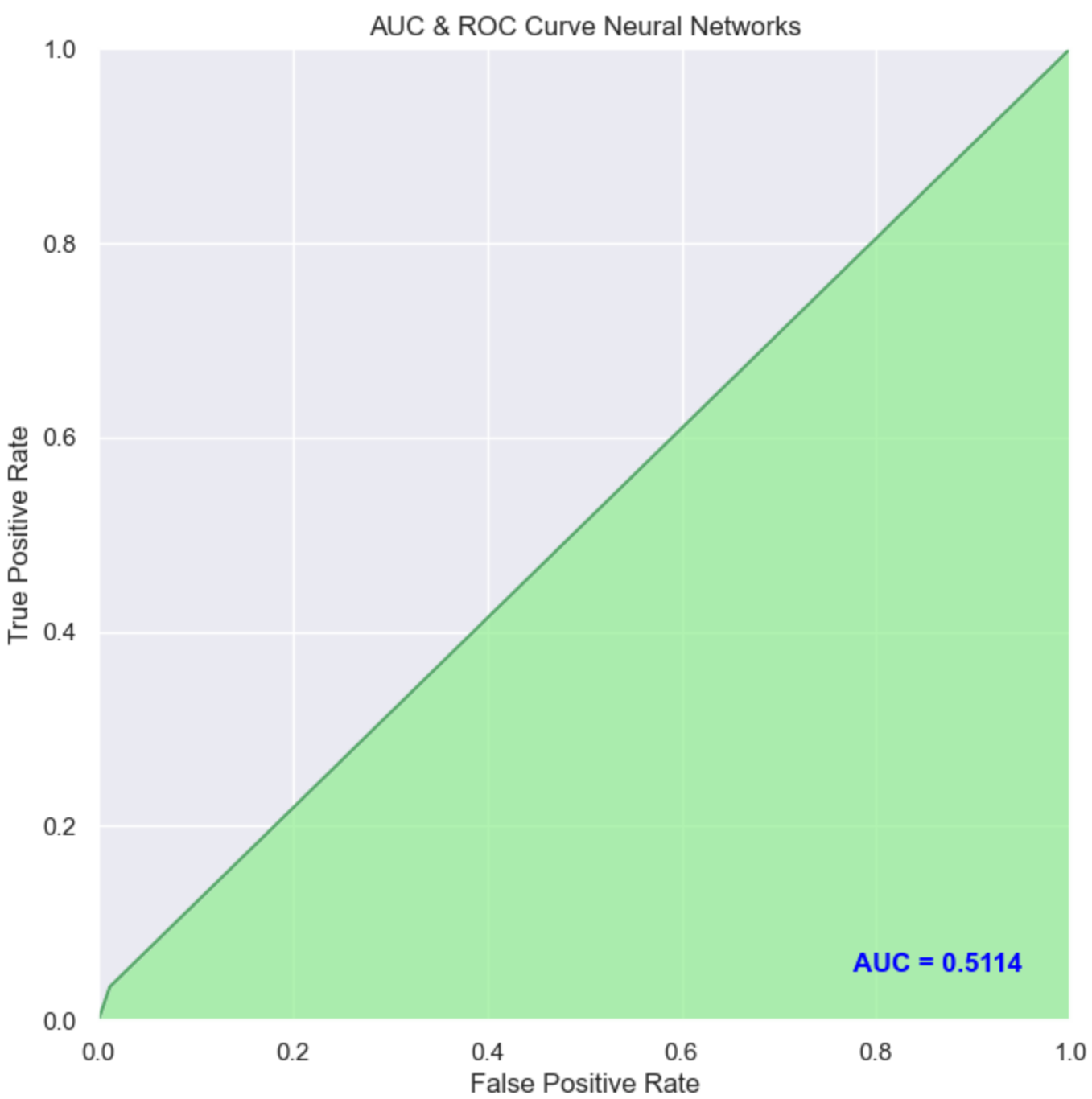
In [100…
```
acc_nn = (TNnn+TPnn)/GTnn
err_nn = 1 - acc_nn
sens_nn = TPnn/TAPnn
spec_nn = TNnn/TANnn
prec_nn = TPnn/TPPnn
f1nn = 2*((prec_nn*sens_nn)/(prec_nn+sens_nn))
f2nn = 5*((prec_nn*sens_nn)/((4*prec_nn)+sens_nn))
f3nn = 1.25*((prec_nn*sens_nn)/((.25*prec_nn)+sens_nn))

OVM_nn = (FNnn*3)+(FPnn*1)

PPC_nn = (-OVM_nn)/GTnn
```

In [101…
```
Datann = [['Accuracy', acc_nn],
        ['Error Rate', err_nn],
        ['Sensitivity', sens_nn],
        ['Specificity', spec_nn],
        ['Precision', prec_nn],
        ['f1', f1nn],
        ['f2',f2nn],
        ['f0.5', f3nn]]
col_namesnn = ['Measurement', 'Neural Network Classifier']
nn_evaluation = print(tabulate(Datann, headers = col_namesnn))
```

```
Measurement      Neural Network Classifier
-------------  --------------------------
Accuracy                         0.729866
Error Rate                       0.270134
Sensitivity                      0.52
Specificity                      0.733726
Precision                        0.0346667
f1                               0.065
f2                               0.136842
f0.5                             0.042623
```

Accuracy is about the value for the majority class for pandemic although Sensitivity is much better than what it was even for the balanced data in the Decision Tree model

In [149…
```
aucnn = metrics.roc_auc_score(y_test.pan_num.values, classes_x)
false_positive_ratenn, true_positive_ratenn, thresoldsnn = metrics.roc_curve(y_test.pan_nu
plt.figure(figsize=(10, 8), dpi=100)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve Neural Networks")
plt.plot(false_positive_ratenn, true_positive_ratenn, 'g')
plt.fill_between(false_positive_ratenn, true_positive_ratenn, facecolor='lightgreen', alph
plt.text(0.95, 0.05, 'AUC = %0.4f' % aucnn, ha='right', fontsize=12, weight='bold', color=
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



AUC is very low, too close to the 0.5 limit for Area Under Curve, therefore the Neural Network is not a contender for a valid classifier in this case.

Naive Bayes Classifier

In [103…
```
NB = MultinomialNB().fit(X_nb, y_nb)
NB.score(X_nb_test, y_nb_test)
```

Out[103…
```
0.6417479234380643
```

In [104…
```
nb_predictions = NB.predict(X_nb_test)
```

In [105…
```
nbpred = pd.crosstab(y_nb_test, nb_predictions, rownames = ['Actual'],
                     colnames = ['Predicted'])
nbpred['Total'] = nbpred.sum(axis=1); nbpred.loc['Total'] = nbpred.sum()

nbpred
```

Out[105…

Predicted	0	1	Total
Actual			
0	1457	562	2019
1	430	320	750
Total	1887	882	2769

In [106…

```
TNnb = nbpred[0][0]
FPnb = nbpred[0][1]
FNnb = nbpred[1][0]
TPnb = nbpred[1][1]
TANnb = TNnb + FPnb
TAPnb = FNnb + TPnb
TPNnb = TNnb + FNnb
TPPnb = FPnb + TPnb
GTnb = TNnb + FPnb + FNnb + TPnb
```

In [107…

```
acc_nb = (TNnb+TPnb)/GTnb
err_nb = 1 - acc_nb
sens_nb = TPnb/TAPnb
spec_nb = TNnb/TANnb
prec_nb = TPnb/TPPnb
f1nb = 2*((prec_nb*sens_nb)/(prec_nb+sens_nb))
f2nb = 5*((prec_nb*sens_nb)/((4*prec_nb)+sens_nb))
f3nb = 1.25*((prec_nb*sens_nb)/((.25*prec_nb)+sens_nb))

OVM_nb = (FNnb*3)+(FPnb*1)

PPC_nb = (-OVM_nb)/GTnb
```

In [108…

```
Datanb = [['Accuracy', acc_nb],
        ['Error Rate', err_nb],
        ['Sensitivity', sens_nb],
        ['Specificity', spec_nb],
        ['Precision', prec_nb],
        ['f1', f1nb],
        ['f2',f2nb],
        ['f0.5', f3nb]]
col_namesnb = ['Measurement', 'Naive Bayes Classifier']
nb_evaluation = print(tabulate(Datanb, headers = col_namesnb))
nb_evaluation
```

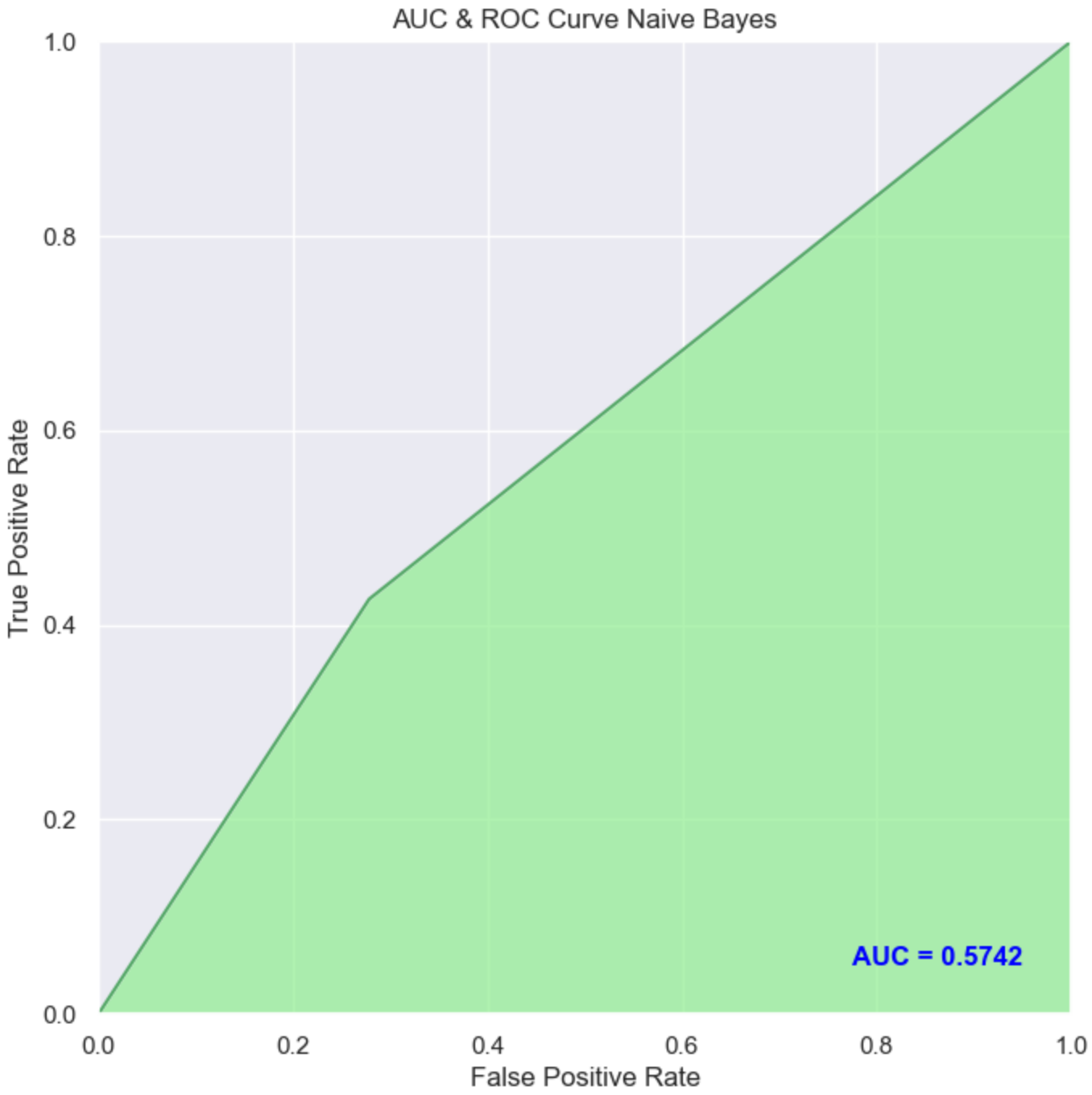
```
Measurement      Naive Bayes Classifier
-------------  ------------------------
Accuracy                       0.641748
Error Rate                     0.358252
Sensitivity                    0.362812
Specificity                    0.772125
Precision                      0.426667
f1                             0.392157
f2                             0.374007
f0.5                           0.412159
```

Accuracy for the Naive Bayes method is too low at 64% and Sensitivity of 36.5%

In [109…

```
aucnb = metrics.roc_auc_score(y_nb_test, nb_predictions)
false_positive_ratenb, true_positive_ratenb, thresoldsnb = metrics.roc_curve(y_nb_test, nb
```

```
plt.figure(figsize=(10, 8), dpi=100)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve Naive Bayes")
plt.plot(false_positive_ratenb, true_positive_ratenb, 'g')
plt.fill_between(false_positive_ratenb, true_positive_ratenb, facecolor='lightgreen', alph
plt.text(0.95, 0.05, 'AUC = %0.4f' % aucnb, ha='right', fontsize=12, weight='bold', color=
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



The AUC is pretty low and close to 0.5 which means it is not much better than a random classifier. This shows the Naive Bayes is not a contender as a valid classifier.

Bagging Method

In [110…

```
bag = BaggingClassifier()
start = time()
```

```
cv = RepeatedStratifiedKFold(n_splits = 5, n_repeats = 2, random_state=0)
score = cross_val_score(bag, X_nb, y_nb, scoring='accuracy',cv=cv, n_jobs=1)
```

In [111…

```
score
```

Out[111…

```
array([0.81346749, 0.80727554, 0.81656347, 0.81501548, 0.79798762,
       0.80340557, 0.79798762, 0.81888545, 0.82275542, 0.79566563])
```

In [112…

```
%%time

res = {}
for i in [10,25,50,100,200,400,800]:
    bags = BaggingClassifier(n_estimators=i)

    cv = RepeatedStratifiedKFold(n_splits = 3, n_repeats = 2, random_state=0)
    res[f'{i}'] = cross_val_score(bags, X_nb, y_nb, scoring='accuracy',
                                  cv=cv, n_jobs=-1)
```

```
Wall time: 26.4 s
```

In [113…

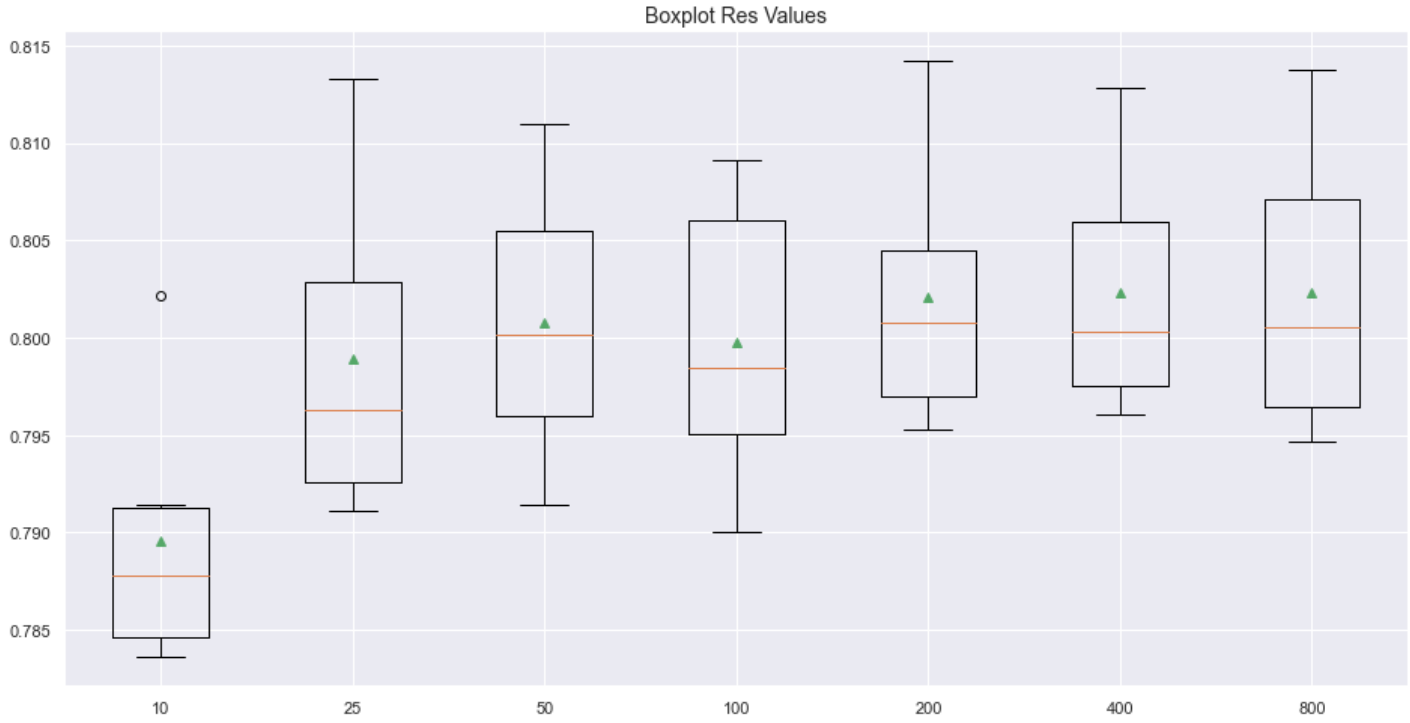
```
res
```

Out[113…

```
{'10': array([0.78365831, 0.78495123, 0.78448676, 0.7906221 , 0.80213655,
        0.79145379]),
 '25': array([0.79108635, 0.79795634, 0.79470506, 0.80454968, 0.81328379,
        0.79191825]),
 '50': array([0.79712163, 0.80631677, 0.795634  , 0.80315692, 0.81096145,
        0.79145379]),
 '100': array([0.7994429 , 0.79749187, 0.79424059, 0.8082637 , 0.80910358,
        0.79006038]),
 '200': array([0.79526462, 0.80445889, 0.7970274 , 0.80454968, 0.81421273,
        0.7970274 ]),
 '400': array([0.79665738, 0.80027868, 0.80027868, 0.80779944, 0.81281932,
        0.79609847]),
 '800': array([0.79758589, 0.80352996, 0.79609847, 0.8082637 , 0.81374826,
        0.79470506])}
```

In [117…

```
plt.figure(figsize=(16,8))
plt.title('Boxplot Res Values', fontsize=14)
plt.boxplot(res.values(), labels = res.keys(), showmeans = True)
plt.show()
```



In [118…
```
bag__ = BaggingClassifier(n_estimators=200)
bag_mod = bag__.fit(X_nb, y_nb)
```

In [119…
```
bag_pred = bag_mod.predict(X_nb_test)
```

In [120…
```
bagpred = pd.crosstab(y_nb_test, bag_pred, rownames = ['Actual'],
                      colnames = ['Predicted'])
bagpred['Total'] = bagpred.sum(axis=1); bagpred.loc['Total'] = bagpred.sum()

bagpred
```

Out[120…

Predicted	0	1	Total
Actual			
0	1869	150	2019
1	247	503	750
Total	2116	653	2769

In [121…
```
TNbag = bagpred[0][0]
FPbag = bagpred[0][1]
FNbag = bagpred[1][0]
TPbag = bagpred[1][1]
TANbag = TNbag + FPbag
TAPbag = FNbag + TPbag
TPNbag = TNbag + FNbag
TPPbag = FPbag + TPbag
GTbag = TNbag + FPbag + FNbag + TPbag
```

In [122…
```
acc_bag = (TNbag+TPbag)/GTbag
err_bag = 1 - acc_bag
sens_bag = TPbag/TAPbag
spec_bag = TNbag/TANbag
prec_bag = TPbag/TPPbag
```

```
f1bag = 2*((prec_bag*sens_bag)/(prec_bag+sens_bag))
f2bag = 5*((prec_bag*sens_bag)/((4*prec_bag)+sens_bag))
f3bag = 1.25*((prec_bag*sens_bag)/((.25*prec_bag)+sens_bag))

OVM_bag = (FNbag*3)+(FPbag*1)

PPC_bag = (-OVM_bag)/GTbag
```

In [123…

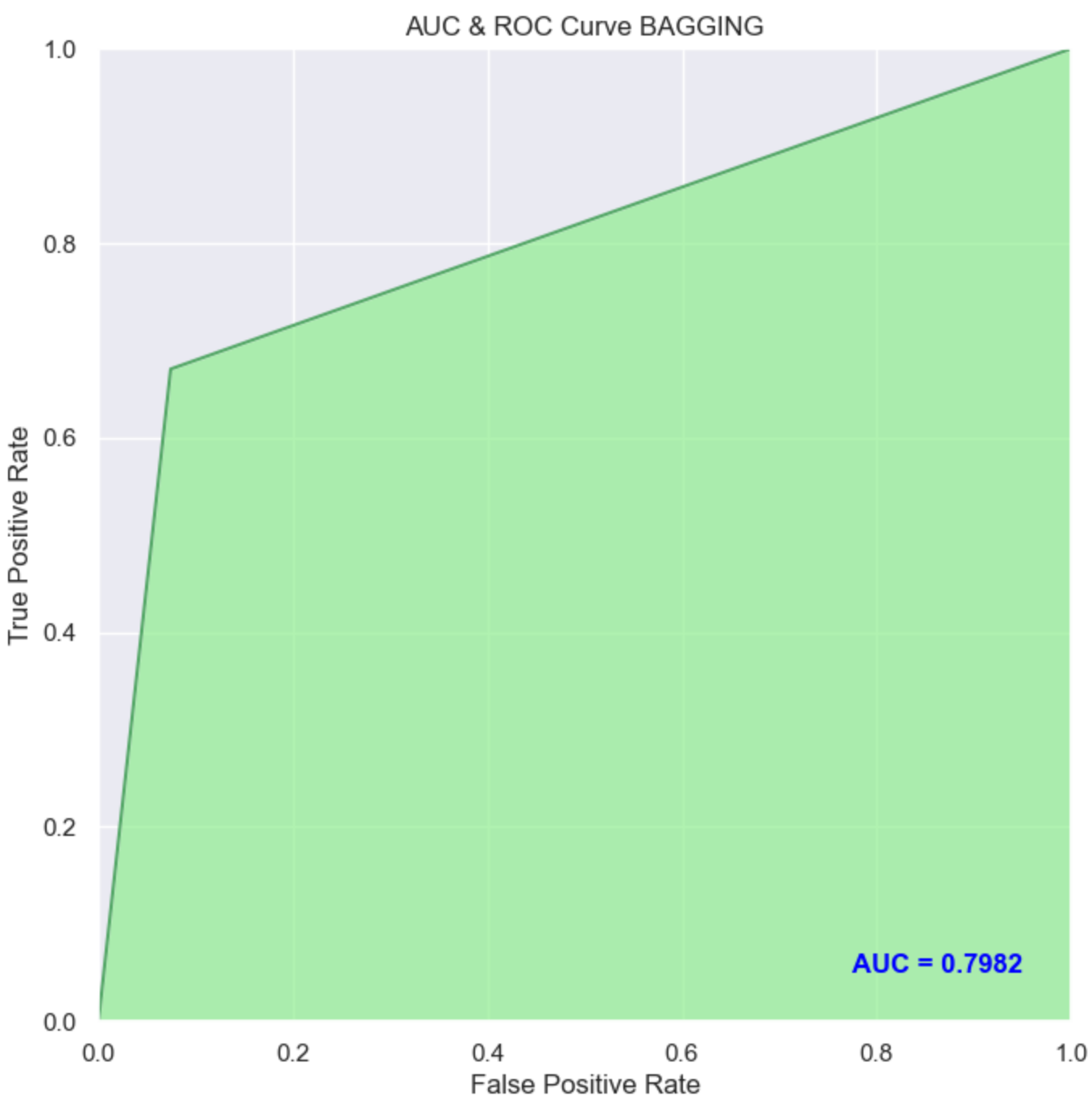
```
Databag = [['Accuracy', acc_bag],
        ['Error Rate', err_bag],
        ['Sensitivity', sens_bag],
        ['Specificity', spec_bag],
        ['Precision', prec_bag],
        ['f1', f1bag],
        ['f2',f2bag],
        ['f0.5', f3bag]]
col_names2 = ['Measurement', 'Bagging Ensemble']
bag_evaluation = print(tabulate(Databag, headers = col_names2))
bag_evaluation
```

```
Measurement      Bagging Ensemble
-------------  ------------------
Accuracy                0.856627
Error Rate              0.143373
Sensitivity             0.770291
Specificity             0.88327
Precision               0.670667
f1                      0.717035
f2                      0.748067
f0.5                    0.688475
```

Accuracy for the Bagging method is great at 86% and Sensitivity at 77%

In [148…

```
aucbag = metrics.roc_auc_score(y_nb_test, bag_pred)
false_positive_ratebag, true_positive_ratebag, thresoldsbag = metrics.roc_curve(y_nb_test,
plt.figure(figsize=(10, 8), dpi=100)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve BAGGING")
plt.plot(false_positive_ratebag, true_positive_ratebag, 'g')
plt.fill_between(false_positive_ratebag, true_positive_ratebag, facecolor='lightgreen', al
plt.text(0.95, 0.05, 'AUC = %0.4f' % aucbag, ha='right', fontsize=12, weight='bold', colo
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



The area under the ROC curve reveals this model is a valid model as it comes close to 1 at 0.8

Boosting Method

In [125…

```
boost = XGBClassifier(n_estimators = 200)
bo_mod = boost.fit(X_nb, y_nb)
```

```
[19:58:39] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [126…

```
bo_pred = bo_mod.predict(X_nb_test)
```

In [127…

```
bopred = pd.crosstab(y_nb_test, bo_pred, rownames = ['Actual'],
                     colnames = ['Predicted'])
bopred['Total'] = bopred.sum(axis=1); bopred.loc['Total'] = bopred.sum()

bopred
```

Out[127…

Predicted	0	1	Total
Actual			
0	1854	165	2019
1	322	428	750
Total	2176	593	2769

In [128…

```
TNbo = bopred[0][0]
FPbo = bopred[0][1]
FNbo = bopred[1][0]
TPbo = bopred[1][1]
TANbo = TNbo + FPbo
TAPbo = FNbo + TPbo
TPNbo = TNbo + FNbo
TPPbo = FPbo + TPbo
GTbo = TNbo + FPbo + FNbo + TPbo
```

In [129…

```
acc_bo = (TNbo+TPbo)/GTbo
err_bo = 1 - acc_bo
sens_bo = TPbo/TAPbo
spec_bo = TNbo/TANbo
prec_bo = TPbo/TPPbo
f1bo = 2*((prec_bo*sens_bo)/(prec_bo+sens_bo))
f2bo = 5*((prec_bo*sens_bo)/((4*prec_bo)+sens_bo))
f3bo = 1.25*((prec_bo*sens_bo)/((.25*prec_bo)+sens_bo))

OVM_bo = (FNbo*3)+(FPbo*1)

PPC_bo = (-OVM_bo)/GTbo
```

In [130…

```
Databo = [['Accuracy', acc_bo],
        ['Error Rate', err_bo],
        ['Sensitivity', sens_bo],
        ['Specificity', spec_bo],
        ['Precision', prec_bo],
        ['f1', f1bo],
        ['f2',f2bo],
        ['f0.5', f3bo]]
col_namesbo = ['Measurement', 'Boosting Ensemble']
bo_evaluation = print(tabulate(Databo, headers = col_namesbo))
bo_evaluation
```

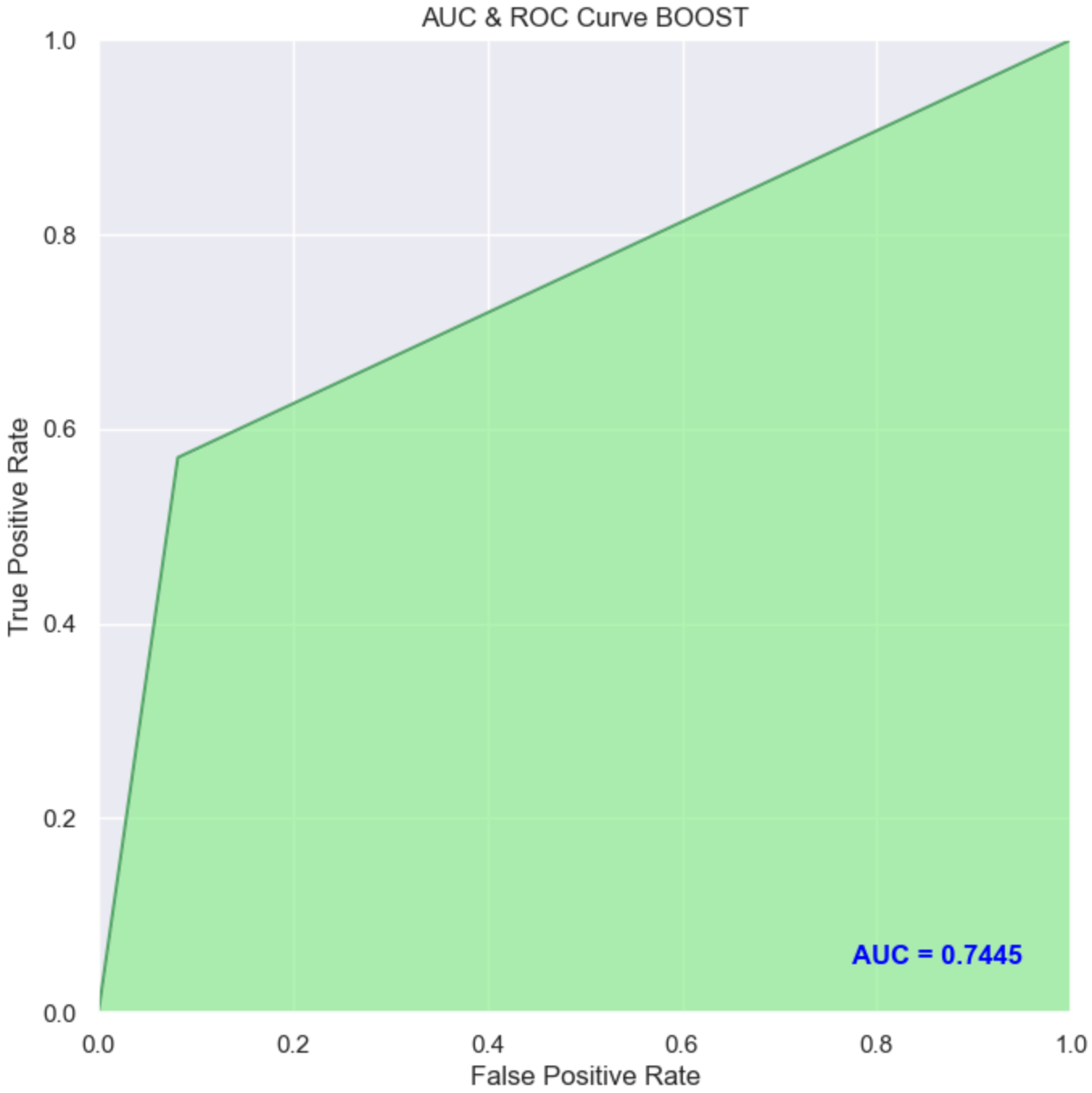
```
Measurement      Boosting Ensemble
-------------  -------------------
Accuracy                  0.824124
Error Rate                0.175876
Sensitivity               0.721754
Specificity               0.852022
Precision                 0.570667
f1                        0.637379
f2                        0.685458
f0.5                      0.595603
```

Boosting method gave good results with 82.4% Accuracy and 72% Sensitivity.

In [147…

```
aucbo = metrics.roc_auc_score(y_nb_test, bo_pred)
false_positive_ratebo, true_positive_ratebo, thresoldsbo = metrics.roc_curve(y_nb_test, bo
plt.figure(figsize=(10, 8), dpi=100)
```

```
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve BOOST")
plt.plot(false_positive_ratebo, true_positive_ratebo, 'g')
plt.fill_between(false_positive_ratebo, true_positive_ratebo, facecolor='lightgreen', alph
plt.text(0.95, 0.05, 'AUC = %0.4f' % aucbo, ha='right', fontsize=12, weight='bold', color=
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



The area under the curve for BOOST is good at 0.745 greater than random classifier and pretty close to 1, the ideal

Random Forest Classifier

In [132…

```
rfy_pure2=np.ravel(y_nb)
```

In [133…

```
rf_model_pure2=RandomForestClassifier(n_estimators = 128,
                                      criterion = "gini"). fit(X_nb,y_nb)
```

In [134…

```
rf_prediction_pure2=rf_model_pure2.predict(X_nb)
```

Create contingency tables and evaluate the performance of the model on the training data before we set forth and try it on the testing data:

In [135…

```
RF_CT_pure2 = confusion_matrix(GRAD_train[['pan_num']],
                               rf_prediction_pure2)
datatest_pure2 = {'actual':GRAD_train['pan_num'],
            'prediction':rf_prediction_pure2}


dftest_pure2 = pd.DataFrame(datatest_pure2)
dftest_pure2.head()

contingency_matrix_RF_train_pure2 = pd.crosstab(index=dftest_pure2['actual'],
                                                columns=dftest_pure2['prediction'])

print(contingency_matrix_RF_train_pure2)
```

```
prediction     0     1
actual
0           4655    71
1            149  1585
```

In [136…

```
TP_RF_pure2train=contingency_matrix_RF_train_pure2[0][0]+contingency_matrix_RF_train_pure2
GT_RF_pure2train = 0
for i in [0, 1]:
    for j in [0, 1]:
        GT_RF_pure2train = GT_RF_pure2train + contingency_matrix_RF_train_pure2[i][j]

Accuracy_RF_pure2train = TP_RF_pure2train/GT_RF_pure2train
Error_RF_pure2train = 1 - Accuracy_RF_pure2train
```

In [137…

```
print('Accuracy for the Random Forest Classifier Pandemic Num: ',
      Accuracy_RF_pure2train*100)
print('Error for the Random Forest Classifier Pandemic Num: ',
      Error_RF_pure2train*100)
```

```
Accuracy for the Random Forest Classifier Pandemic Num:  96.59442724458205
Error for the Random Forest Classifier Pandemic Num:  3.4055727554179516
```

Evaluate Pure Random Forest on unseen test data:

In [138…

```
rftest_prediction_pure2=rf_model_pure2.predict(X_nb_test)
```

In [139…

```
RFtest_CT_pure2 = confusion_matrix(GRAD_test[['pan_num']],
                               rftest_prediction_pure2)
datatest2_pure2 = {'actual':GRAD_test['pan_num'],
            'prediction':rftest_prediction_pure2}


dftest2_pure2 = pd.DataFrame(datatest2_pure2)
dftest2_pure2.head()

contingency_matrix_RF_test_pure2 = pd.crosstab(index=dftest2_pure2['actual'],
```

```
                                          columns=dftest2_pure2['prediction'])

print(contingency_matrix_RF_test_pure2)
```

```
prediction     0    1
actual
0           1883  136
1            252  498
```

In [140…
```
TP_RF_pure2=contingency_matrix_RF_test_pure2[0][0]+contingency_matrix_RF_test_pure2[1][1]
GT_RF_pure2 = 0
for i in [0, 1]:
    for j in [0, 1]:
        GT_RF_pure2 = GT_RF_pure2 + contingency_matrix_RF_test_pure2[i][j]

Accuracy_RF_pure2 = TP_RF_pure2/GT_RF_pure2
Error_RF_pure2 = 1 - Accuracy_RF_pure2
TP_RF_pure2
```

Out[140…
```
2381
```

In [141…
```
print('Accuracy for the Random Forest Classifier: ', Accuracy_RF_pure2*100)
print('Error for the Random Forest Classifier: ', Error_RF_pure2*100)
```

```
Accuracy for the Random Forest Classifier:  85.98772119898881
Error for the Random Forest Classifier:  14.012278801011192
```

In [142…
```
TNmodel1RF=RFtest_CT_pure2[0][0]
FPmodel1RF=RFtest_CT_pure2[0][1]
FNmodel1RF=RFtest_CT_pure2[1][0]
TPmodel1RF=RFtest_CT_pure2[1][1]

TANmodel1RF=TNmodel1RF+FPmodel1RF
TAPmodel1RF=TPmodel1RF+FNmodel1RF
TPPmodel1RF=FPmodel1RF+TPmodel1RF
TPNmodel1RF=TNmodel1RF+FNmodel1RF

GTmodel1RF=TANmodel1RF+TAPmodel1RF

AccuracyM1RF=(TNmodel1RF+TPmodel1RF)/GTmodel1RF
ErrorRateM1RF=1-AccuracyM1RF
SensitivityM1RF=TPmodel1RF/(TAPmodel1RF)
RecallM1RF=SensitivityM1RF
SpecificityM1RF=TNmodel1RF/TANmodel1RF
PrecisionM1RF=TPmodel1RF/TPPmodel1RF
F1M1RF=2*PrecisionM1RF*RecallM1RF/(PrecisionM1RF + RecallM1RF)
F2M1RF=5*(PrecisionM1RF*RecallM1RF)/((4*PrecisionM1RF)+RecallM1RF)
Fp5M1RF=(1.25)*(PrecisionM1RF*RecallM1RF)/((0.25*PrecisionM1RF)+RecallM1RF)
```

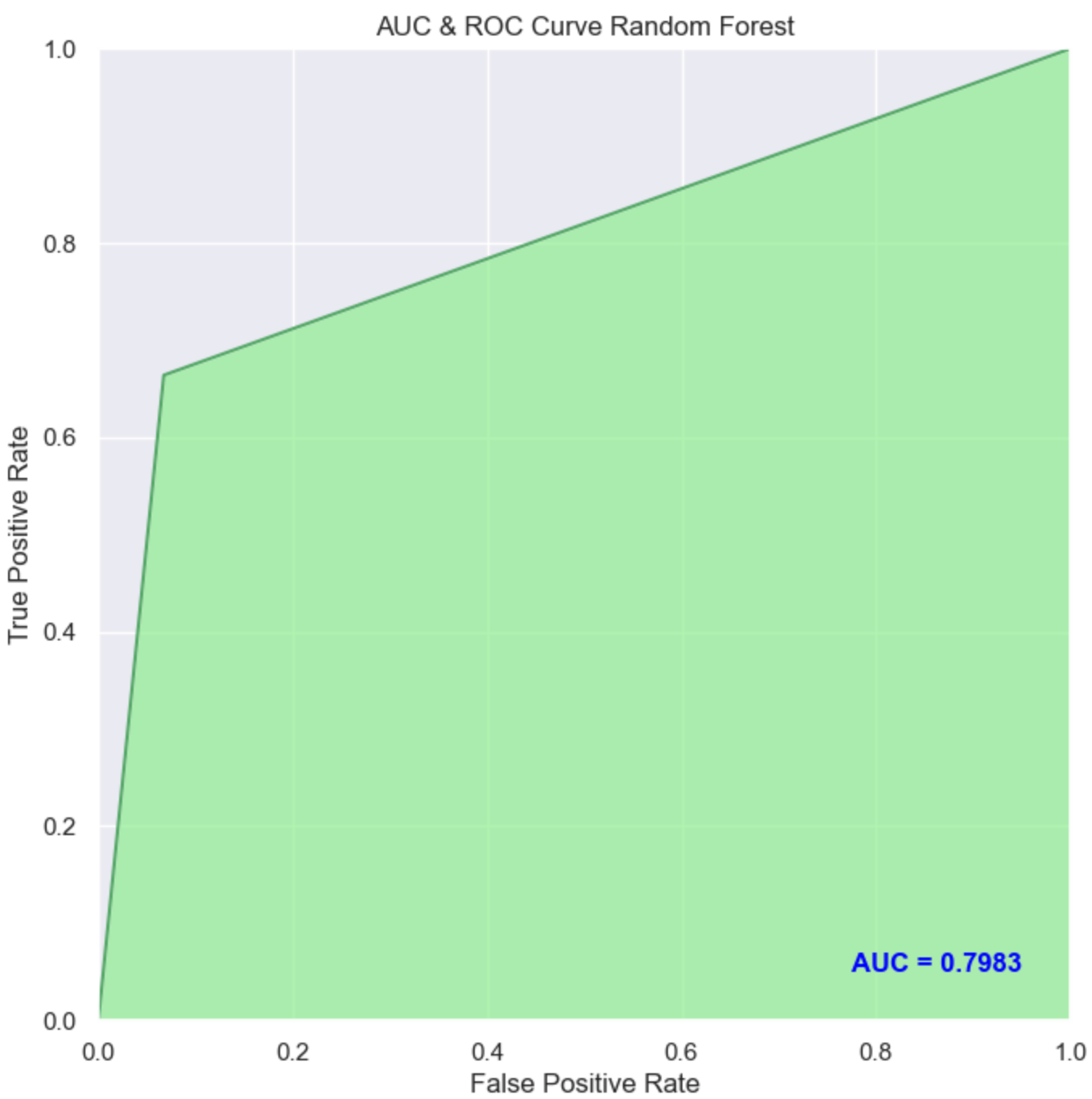
In [143…
```
header = ["Accuracy", "Error Rate", "Sensitivity", "Recall", "Specificity",
          "Precision", "F1", "F2", "F0.5"]
data1RF = [["Accuracy", AccuracyM1RF], ["Error Rate", ErrorRateM1RF],
          ["Sensitivity", SensitivityM1RF],
          ["Recall", RecallM1RF], ["Specificity", SpecificityM1RF],
          ["Precision", PrecisionM1RF],
          ["F1", F1M1RF], ["F2", F2M1RF], ["F0.5", Fp5M1RF]]
col_names=["Measurement", "Random Forest"]

ModelEvaluationTableRF = tabulate(data1RF, headers=col_names,
                                  tablefmt="fancy_grid")
```

In [144…
```
print(ModelEvaluationTableRF)
```

Measurement	Random Forest
Accuracy	0.859877
Error Rate	0.140123
Sensitivity	0.664
Recall	0.664
Specificity	0.93264
Precision	0.785489
F1	0.719653
F2	0.685195
F0.5	0.75776

Accuracy for the Random Forest is considerably better at 86% with a moderately good Sensitivity/Recall meaning it catches the smaller class with about 66.4% rate, even without oversampling the data.

In [145…
```
aucrf = metrics.roc_auc_score(y_nb_test, rftest_prediction_pure2)
false_positive_raterf, true_positive_raterf, thresoldsrf = metrics.roc_curve(y_nb_test, rf
plt.figure(figsize=(10, 8), dpi=100)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve Random Forest")
plt.plot(false_positive_raterf, true_positive_raterf, 'g')
plt.fill_between(false_positive_raterf, true_positive_raterf, facecolor='lightgreen', alph
plt.text(0.95, 0.05, 'AUC = %0.4f' % aucrf, ha='right', fontsize=12, weight='bold', color=
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



The AUC (area under the curve) of the ROC curve is quite close to 1, at 0.798 similar to the bagging method, so it is also a contender as a valid classifier

---

Summary Model Evaluation Results:

In [156…

```
data_complete = [['Accuracy',AccuracyM1, AccuracyM1RF, acc_nn],
        ['Error Rate', ErrorRateM1, ErrorRateM1RF, err_nn],
        ['Sensitivity', SensitivityM1, SensitivityM1RF, sens_nn],
        ['Specificity', SpecificityM1, SpecificityM1RF, spec_nn],
        ['Precision', PrecisionM1, PrecisionM1RF, prec_nn],
        ['f1', F1M1, F1M1RF, f1nn],
        ['f2',F2M1, F2M1RF, f2nn],
        ['f0.5', Fp5M1, Fp5M1RF, f3nn]]

col_name_comp = ['Measurement', 'Decision Tree', 'Random Forest Classifier',
            'Neural Network Classifier']

evaluation = print(tabulate(data_complete, headers = col_name_comp))
```

```
evaluation

data_complete2 = [['Accuracy', acc_nb, acc_bo,acc_bag],
        ['Error Rate', err_nb, err_bo,err_bag],
        ['Sensitivity', sens_nb, sens_bo, sens_bag],
        ['Specificity', spec_nb, spec_bo, spec_bag],
        ['Precision', prec_nb, prec_bo, prec_bag],
        ['f1', f1nb, f1bo, f1bag],
        ['f2', f2nb, f2bo, f2bag],
        ['f0.5', f3nb, f3bo, f3bag]]

col_name_comp2 = ['Measurement', 'Naive Bayes',
            'Boosting', 'Bagging']

print("\n")
evaluation2 = print(tabulate(data_complete2, headers = col_name_comp2))
evaluation2
```

Measurement	Decision Tree	Random Forest Classifier	Neural Network Classifier
Accuracy	0.74684	0.859877	0.729866
Error Rate	0.25316	0.140123	0.270134
Sensitivity	0.314667	0.664	0.52
Specificity	0.90738	0.93264	0.733726
Precision	0.55792	0.785489	0.0346667
f1	0.402387	0.719653	0.065
f2	0.344727	0.685195	0.136842
f0.5	0.48321	0.75776	0.042623

Measurement	Naive Bayes	Boosting	Bagging
Accuracy	0.641748	0.824124	0.856627
Error Rate	0.358252	0.175876	0.143373
Sensitivity	0.362812	0.721754	0.770291
Specificity	0.772125	0.852022	0.88327
Precision	0.426667	0.570667	0.670667
f1	0.392157	0.637379	0.717035
f2	0.374007	0.685458	0.748067
f0.5	0.412159	0.595603	0.688475

Given that Random Forest implements bagging into the algorithm, it makes sense that these two classifiers worked best in this classification scenario.
