For each dataset, the variable class indicates whether the data is normal or anomalous. Pick one of the four sets

Requirements

1.) Try to locate some background on the data, ideally a data dictionary.

2.) Some of the data sets are very large, particularly the census set. You may have to reduce the number of variables you used in the detection to just those that seem particularly relevant or interesting. You may also need to work with a limited number of rows, at least for the SVM based encoder.

3.) Determine how many normal data rows you have (class=0) and how many anomalies. Split your data into an anomalous set, and split the anomalous set into two pieces (test and validation) of roughly equal size. Split the regular data (class 0) into three sets, (train, test and validation) with the test and evaluation data sets the same size as the anomalous test and validation sets. Use all the remaining “regular” data as your training set. Remember this is “unsupervised” learning to create an anomaly detector.

4.) Do some exploratory data analysis to understand what you are working with. Do a range of various types of summary or descriptive analysis. Include a heat map of the correlation of the variables.

5.) Create an SVM based anomaly detector, set it with a variable cutoff boundary, set this to 5% to start with, but be sure it is adjustable in code. SVM gives you various options of operation, try several and find the optimum.

6.) Produce a confusion matrix of correct and incorrect anomaly detections, using your validation data. Show this at several different boundary levels.

7.) Now build an anomaly detector (autoencoder) using Tensorflow and Keras. Optimize its operation at least a bit . Determine how to set up the boundary to detect anomalous and non-anomalous results.

8.) Decide which of the two methods seems to work best, using your validation data. Can you figure out how to make simultaneous use of both methods to improve overall performance? Do the two methods make the same mistakes in anomaly detection or are there differences in the two? Once you have figured out what method, or combination of methods, works best, set it up and run it on your test data. Report the rate of correct assignment of both anomalies and regular (normal) data.

Pick one of the four sets

I pick bank-additional-full_normalised

1.) Try to locate some background on the data, ideally a data dictionary.

Okay lets get googleing.

Input variables:

bank client data:

1 - age (numeric) 2 - job : type of job (categorical: "admin.","blue-collar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown") 3 - marital : marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed) 4 - education (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unk 5 - default: has credit in default? (categorical: "no","yes","unknown") 6 - housing: has housing loan? (categorical: "no","yes","unknown") 7 - loan: has personal loan? (categorical: "no","yes","unknown")

related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: "cellular","telephone") 9 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec") 10 - day_of_week: last contact day of the week (categorical: "mon","tue","wed","thu","fri") 11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact) 13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted) 14 - previous: number of contacts performed before this campaign and for this client (numeric) 15 - poutcome: outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")

social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric) 17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric) 20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target): 21 - y - has the client subscribed a term deposit? (binary: "yes","no")

This data set looks like a one hot encoded and normalized version, clean and ready to enter a model.

Seperation

Seperation

Seperation

2.) Some of the data sets are very large, particularly the census set. You may have to reduce the number of variables you used in the detection to just those that seem particularly relevant or interesting. You may also need to work with a limited number of rows, at least for the SVM based encoder.

In [375…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [376…
```
#load the data, change the file address

infile= "/content/bank-additional-full_normalised.csv"
bank=pd.read_csv(infile)
```

In [377…
```
bank.head()
```

Out[377]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	job=r
0	0.209877	0	0	0	0	0	0	
1	0.296296	0	0	1	0	0	0	
2	0.246914	1	0	0	0	0	0	
3	0.160494	0	1	0	0	0	0	
4	0.530864	0	0	0	1	0	0	

In [378…
```
pd.set_option('display.max_columns', None)
bank.head()
```

Out[378]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	job=r
0	0.209877	0	0	0	0	0	0	
1	0.296296	0	0	1	0	0	0	
2	0.246914	1	0	0	0	0	0	
3	0.160494	0	1	0	0	0	0	
4	0.530864	0	0	0	1	0	0	

In [379…
```
bank.shape
```

Out[379]:
```
(41188, 63)
```

In [380…
```
bank.isnull().values.any()
```

Out[380]:
```
False
```

I am going to get rid off day of weeek, and month of year.

In [381…
```
bank2 = bank.drop(columns=["month=may","month=jun","month=jul", "month=aug", "month=oc
```

In [382…
```
bank2.head()
```

Out[382]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	job=r
0	0.209877	0	0	0	0	0	0	
1	0.296296	0	0	1	0	0	0	
2	0.246914	1	0	0	0	0	0	
3	0.160494	0	1	0	0	0	0	
4	0.530864	0	0	0	1	0	0	

As far as column reduction goes I hope removing 15 is enough. These are the largest catagories so I got rid off them, it reduces size and it's of 1 catagory well there is month and day but it's all time.

3.) Determine how many normal data rows you have (class=0) and how many anomalies. Split your data into an anomalous set, and split the anomalous set into two pieces (test and validation) of roughly equal size. Split the regular data (class 0) into three sets, (train, test and validation) with the test and evaluation data sets the same size as the anomalous test and validation sets. Use all the remaining "regular" data as your training set. Remember this is "unsupervised" learning to create an anomaly detector.

In [383…
```
#normal data rows

print("Normal data row size: "+ str(len(bank2[bank2['class']==0])))
print("Anomalies data row size: "+ str(len(bank2[bank2['class']==1])))

normalData = bank2[bank2['class']==0]
anomaliesData = bank2[bank2['class']==1]
```

```
Normal data row size: 36548
Anomalies data row size: 4640
```

In [384…
```
#yNormal = normalData['class']
yAnomalies = anomaliesData['class']
```

In [385…
```
#normalData= normalData.drop(columns=['class'])
anomaliesData= anomaliesData.drop(columns=['class'])
```

In [386…
```
normalData.shape
```

Out[386]:
```
(36548, 48)
```

In [387…
```
anomaliesData.shape
```

Out[387]:
```
(4640, 47)
```

In [388…
```
#yNormal.shape
```

In [389…
```
yAnomalies.shape
```

Out[389]:
```
(4640,)
```

In [390…
```
anomaliesData
```

Out[390]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
4	0.530864	0	0	0	1	0	0	
23	0.197531	0	0	0	1	0	0	
33	0.160494	0	0	1	0	0	0	
44	0.209877	0	0	0	0	1	0	
50	0.111111	0	0	1	0	0	0	
…	…	…	…	…	…	…	…	…
41149	0.061728	0	0	0	0	0	0	
41163	0.246914	0	0	0	1	0	0	
41166	0.567901	0	0	0	0	0	1	
41177	0.123457	0	0	1	0	0	0	
41184	0.333333	0	0	0	0	0	0	

4640 rows × 47 columns

In [391…
```
yAnomalies
```

Out[391]:
```
4        1
23       1
33       1
44       1
50       1
        ..
41149    1
41163    1
41166    1
41177    1
41184    1
Name: class, Length: 4640, dtype: int64
```

In [392…
```
# Split the anomalous set into two pieces (test and validation) of roughly equal size.

from sklearn.model_selection import train_test_split

X_test_anomalies, X_validation_anomalies, y_test_anomalies, y_validation_anomalies = t
```

In [393…
```
X_test_anomalies.head()
```

Out[393]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
22279	0.098765	0	0	1	0	0	0	
3877	0.456790	1	0	0	0	0	0	
32463	0.234568	0	1	0	0	0	0	
13250	0.444444	0	1	0	0	0	0	
9707	0.320988	0	0	0	1	0	0	

Split the regular data (class 0) into three sets, (train, test and validation) with the test and evaluation data sets the same size as the anomalous test and validation sets.

In [394…
```
normalData.shape
```

Out[394]:
```
(36548, 48)
```

In [395…
```
X_test_normal = normalData.sample(len(X_test_anomalies))
```

In [396…
```
y_test_normal = X_test_normal['class']
X_test_normal = X_test_normal.drop(columns=['class'])
```

In [397…
```
normalData=normalData.drop(X_test_normal.index)
```

In [398…
```
X_validation_normal = normalData.sample(len(X_validation_anomalies))
```

In [399…
```
y_validation_normal = X_validation_normal['class']
X_validation_normal = X_validation_normal.drop(columns=['class'])
```

In [400…
```
normalData=normalData.drop(X_validation_normal.index)
```

In [401…
```
normalData.shape
```

Out[401]:
```
(31908, 48)
```

In [402…
```
X_validation_normal
```

Out[402]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
35979	0.493827	1	0	0	0	0	0	
16088	0.283951	0	0	0	0	0	0	
31279	0.160494	0	0	1	0	0	0	
1801	0.419753	0	0	0	0	1	0	
35082	0.296296	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	
39237	0.222222	0	0	0	1	0	0	
6804	0.271605	0	0	0	1	0	0	
22717	0.172840	0	0	0	1	0	0	
36427	0.469136	0	0	0	0	0	1	
22447	0.271605	0	0	1	0	0	0	

2320 rows × 47 columns

In [403…

```
X_test_normal
```

Out[403]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
26488	0.234568	0	0	0	1	0	0	
3792	0.259259	0	0	0	0	1	0	
19839	0.432099	0	0	0	1	0	0	
33220	0.098765	0	0	0	1	0	0	
28937	0.160494	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
23812	0.407407	0	0	0	0	1	0	
30177	0.259259	0	0	0	0	0	0	
15758	0.308642	0	0	0	1	0	0	
8647	0.197531	0	0	0	1	0	0	
373	0.382716	0	0	0	1	0	0	

2320 rows × 47 columns

In [404…

```
y_train_normal = normalData['class']
X_train_normal = normalData.drop(columns=['class'])
```

In [405…
```
y_train_normal
```

Out[405]:
```
0        0
1        0
2        0
3        0
5        0
        ..
41182    0
41183    0
41185    0
41186    0
41187    0
Name: class, Length: 31908, dtype: int64
```

In [406…
```
X_train_normal
```

Out[406]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
0	0.209877	0	0	0	0	0	0	
1	0.296296	0	0	1	0	0	0	
2	0.246914	1	0	0	0	0	0	
3	0.160494	0	1	0	0	0	0	
5	0.456790	0	0	1	0	0	0	
...	...	...	...	...	...	...	...	
41182	0.432099	0	1	0	0	0	0	
41183	0.271605	0	0	0	1	0	0	
41185	0.172840	0	0	0	0	1	0	
41186	0.148148	0	0	1	0	0	0	
41187	0.382716	0	0	0	0	1	0	

31908 rows × 47 columns

In [407…
```
X_test_normal
```

Out[407]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
26488	0.234568	0	0	0	1	0	0	
3792	0.259259	0	0	0	0	1	0	
19839	0.432099	0	0	0	1	0	0	
33220	0.098765	0	0	0	1	0	0	
28937	0.160494	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...
23812	0.407407	0	0	0	0	1	0	
30177	0.259259	0	0	0	0	0	0	
15758	0.308642	0	0	0	1	0	0	
8647	0.197531	0	0	0	1	0	0	
373	0.382716	0	0	0	1	0	0	

2320 rows × 47 columns

In [408…
```
y_test_normal
```

Out[408]:
```
26488    0
3792     0
19839    0
33220    0
28937    0
        ..
23812    0
30177    0
15758    0
8647     0
373      0
Name: class, Length: 2320, dtype: int64
```

In [409…
```
y_validation_normal
```

Out[409]:
```
35979    0
16088    0
31279    0
1801     0
35082    0
        ..
39237    0
6804     0
22717    0
36427    0
22447    0
Name: class, Length: 2320, dtype: int64
```

Seesh too many things to keep track of.

4.) Do some exploratory data analysis to understand what you are working with. Do a range of various types of summary or descriptive analysis. Include a heat map of the correlation of the variables.

In [410…

```
normalData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 31908 entries, 0 to 41187
Data columns (total 48 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   age                             31908 non-null  float64
 1   job=housemaid                   31908 non-null  int64
 2   job=services                    31908 non-null  int64
 3   job=admin.                      31908 non-null  int64
 4   job=blue-collar                 31908 non-null  int64
 5   job=technician                  31908 non-null  int64
 6   job=retired                     31908 non-null  int64
 7   job=management                  31908 non-null  int64
 8   job=unemployed                  31908 non-null  int64
 9   job=self-employed               31908 non-null  int64
 10  job=unknown                     31908 non-null  int64
 11  job=entrepreneur                31908 non-null  int64
 12  job=student                     31908 non-null  int64
 13  marital=married                 31908 non-null  int64
 14  marital=single                  31908 non-null  int64
 15  marital=divorced                31908 non-null  int64
 16  marital=unknown                 31908 non-null  int64
 17  education=basic.4y              31908 non-null  int64
 18  education=high.school           31908 non-null  int64
 19  education=basic.6y              31908 non-null  int64
 20  education=basic.9y              31908 non-null  int64
 21  education=professional.course   31908 non-null  int64
 22  education=unknown               31908 non-null  int64
 23  education=university.degree     31908 non-null  int64
 24  education=illiterate            31908 non-null  int64
 25  default=0                       31908 non-null  int64
 26  default=unknown                 31908 non-null  int64
 27  default=1                       31908 non-null  int64
 28  housing=0                       31908 non-null  int64
 29  housing=1                       31908 non-null  int64
 30  housing=unknown                 31908 non-null  int64
 31  loan=0                          31908 non-null  int64
 32  loan=1                          31908 non-null  int64
 33  loan=unknown                    31908 non-null  int64
 34  contact=cellular                31908 non-null  int64
 35  duration                        31908 non-null  float64
 36  campaign                        31908 non-null  float64
 37  pdays                           31908 non-null  float64
 38  previous                        31908 non-null  float64
 39  poutcome=nonexistent            31908 non-null  int64
 40  poutcome=failure                31908 non-null  int64
 41  poutcome=success                31908 non-null  int64
 42  emp.var.rate                    31908 non-null  float64
 43  cons.price.idx                  31908 non-null  float64
 44  cons.conf.idx                   31908 non-null  float64
 45  euribor3m                       31908 non-null  float64
 46  nr.employed                     31908 non-null  float64
 47  class                           31908 non-null  int64
dtypes: float64(10), int64(38)
memory usage: 11.9 MB
```

In [411…

```
anomaliesData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4640 entries, 4 to 41184
Data columns (total 47 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   age                             4640 non-null   float64
 1   job=housemaid                   4640 non-null   int64
 2   job=services                    4640 non-null   int64
 3   job=admin.                      4640 non-null   int64
 4   job=blue-collar                 4640 non-null   int64
 5   job=technician                  4640 non-null   int64
 6   job=retired                     4640 non-null   int64
 7   job=management                  4640 non-null   int64
 8   job=unemployed                  4640 non-null   int64
 9   job=self-employed               4640 non-null   int64
 10  job=unknown                     4640 non-null   int64
 11  job=entrepreneur                4640 non-null   int64
 12  job=student                     4640 non-null   int64
 13  marital=married                 4640 non-null   int64
 14  marital=single                  4640 non-null   int64
 15  marital=divorced                4640 non-null   int64
 16  marital=unknown                 4640 non-null   int64
 17  education=basic.4y              4640 non-null   int64
 18  education=high.school           4640 non-null   int64
 19  education=basic.6y              4640 non-null   int64
 20  education=basic.9y              4640 non-null   int64
 21  education=professional.course   4640 non-null   int64
 22  education=unknown               4640 non-null   int64
 23  education=university.degree     4640 non-null   int64
 24  education=illiterate            4640 non-null   int64
 25  default=0                       4640 non-null   int64
 26  default=unknown                 4640 non-null   int64
 27  default=1                       4640 non-null   int64
 28  housing=0                       4640 non-null   int64
 29  housing=1                       4640 non-null   int64
 30  housing=unknown                 4640 non-null   int64
 31  loan=0                          4640 non-null   int64
 32  loan=1                          4640 non-null   int64
 33  loan=unknown                    4640 non-null   int64
 34  contact=cellular                4640 non-null   int64
 35  duration                        4640 non-null   float64
 36  campaign                        4640 non-null   float64
 37  pdays                           4640 non-null   float64
 38  previous                        4640 non-null   float64
 39  poutcome=nonexistent            4640 non-null   int64
 40  poutcome=failure                4640 non-null   int64
 41  poutcome=success                4640 non-null   int64
 42  emp.var.rate                    4640 non-null   float64
 43  cons.price.idx                  4640 non-null   float64
 44  cons.conf.idx                   4640 non-null   float64
 45  euribor3m                       4640 non-null   float64
 46  nr.employed                     4640 non-null   float64
dtypes: float64(10), int64(37)
memory usage: 1.7 MB
```

In [412…

```
X_train_normal.describe()
```

Out[412]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job
count	31908.000000	31908.000000	31908.000000	31908.000000	31908.000000	31908.000000	31908
mean	0.282734	0.026294	0.099379	0.249185	0.234581	0.165350	
std	0.122412	0.160012	0.299176	0.432548	0.423743	0.371502	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.185185	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.259259	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	0.370370	0.000000	0.000000	0.000000	0.000000	0.000000	
max	0.962963	1.000000	1.000000	1.000000	1.000000	1.000000	

In [413…
```
X_test_normal.describe()
```

Out[413]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=reti
count	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000
mean	0.284759	0.024569	0.105172	0.249138	0.240517	0.153448	0.031
std	0.119561	0.154841	0.306842	0.432607	0.427490	0.360497	0.174
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.185185	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	0.259259	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	0.370370	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
max	0.827160	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

In [414…
```
X_validation_normal.describe()
```

Out[414]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=reti
count	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000(
mean	0.282604	0.025000	0.099569	0.233190	0.246983	0.164224	0.034
std	0.121912	0.156159	0.299489	0.422953	0.431350	0.370559	0.182
min	0.012346	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.197531	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
50%	0.259259	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
75%	0.358025	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
max	0.851852	1.000000	1.000000	1.000000	1.000000	1.000000	1.000(

Lets do the anomaliesData sets.

In [415…
```
X_test_anomalies.describe()
```

Out[415]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=reti
count	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000(
mean	0.290336	0.021552	0.074138	0.285776	0.132759	0.160776	0.096
std	0.169098	0.145246	0.262052	0.451881	0.339387	0.367403	0.295
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.160494	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
50%	0.246914	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
75%	0.407407	0.000000	0.000000	1.000000	0.000000	0.000000	0.000(
max	0.925926	1.000000	1.000000	1.000000	1.000000	1.000000	1.000(

In [416…
```
X_validation_anomalies.describe()
```

Out[416]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=reti
count	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000000	2320.000(
mean	0.300112	0.024138	0.065086	0.296983	0.142241	0.153879	0.090(
std	0.172449	0.153510	0.246731	0.457027	0.349373	0.360911	0.286:
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.172840	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
50%	0.259259	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
75%	0.407407	0.000000	0.000000	1.000000	0.000000	0.000000	0.000(
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000(

In [417…
```
X_test_anomalies.head()
```

Out[417]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
22279	0.098765	0	0	1	0	0	0	
3877	0.456790	1	0	0	0	0	0	
32463	0.234568	0	1	0	0	0	0	
13250	0.444444	0	1	0	0	0	0	
9707	0.320988	0	0	0	1	0	0	

In [418…
```
X_validation_anomalies.head()
```

Out[418]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
11341	0.234568	0	0	0	1	0	0	
15351	0.481481	0	0	0	1	0	0	
24020	0.074074	0	0	0	0	0	0	
31584	0.197531	0	1	0	0	0	0	
434	0.024691	0	0	0	0	0	0	

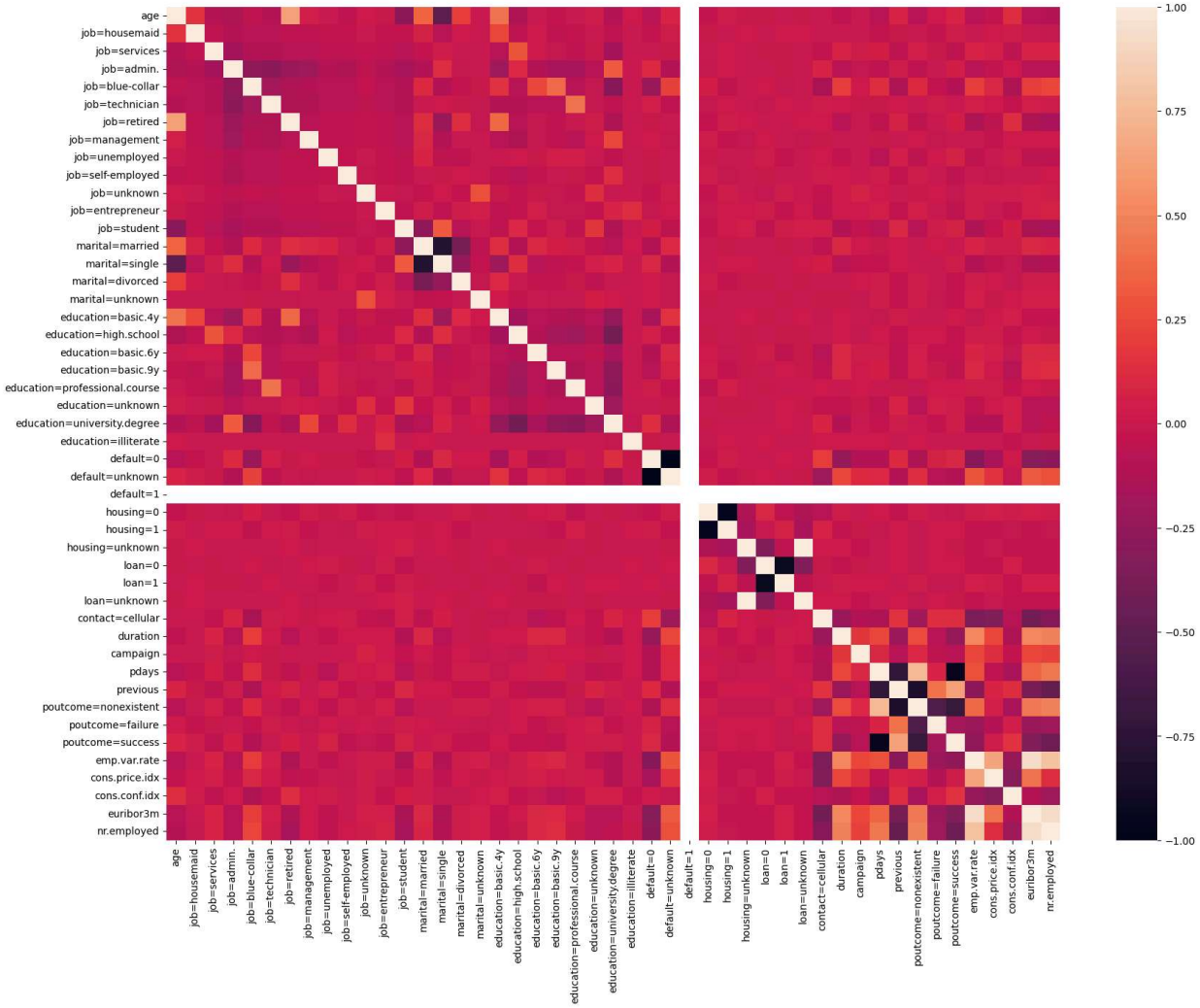
they are diffrent but freakishly close statistic

Include a heat map of the correlation of the variables.

In [419…
```
import seaborn as sns
```

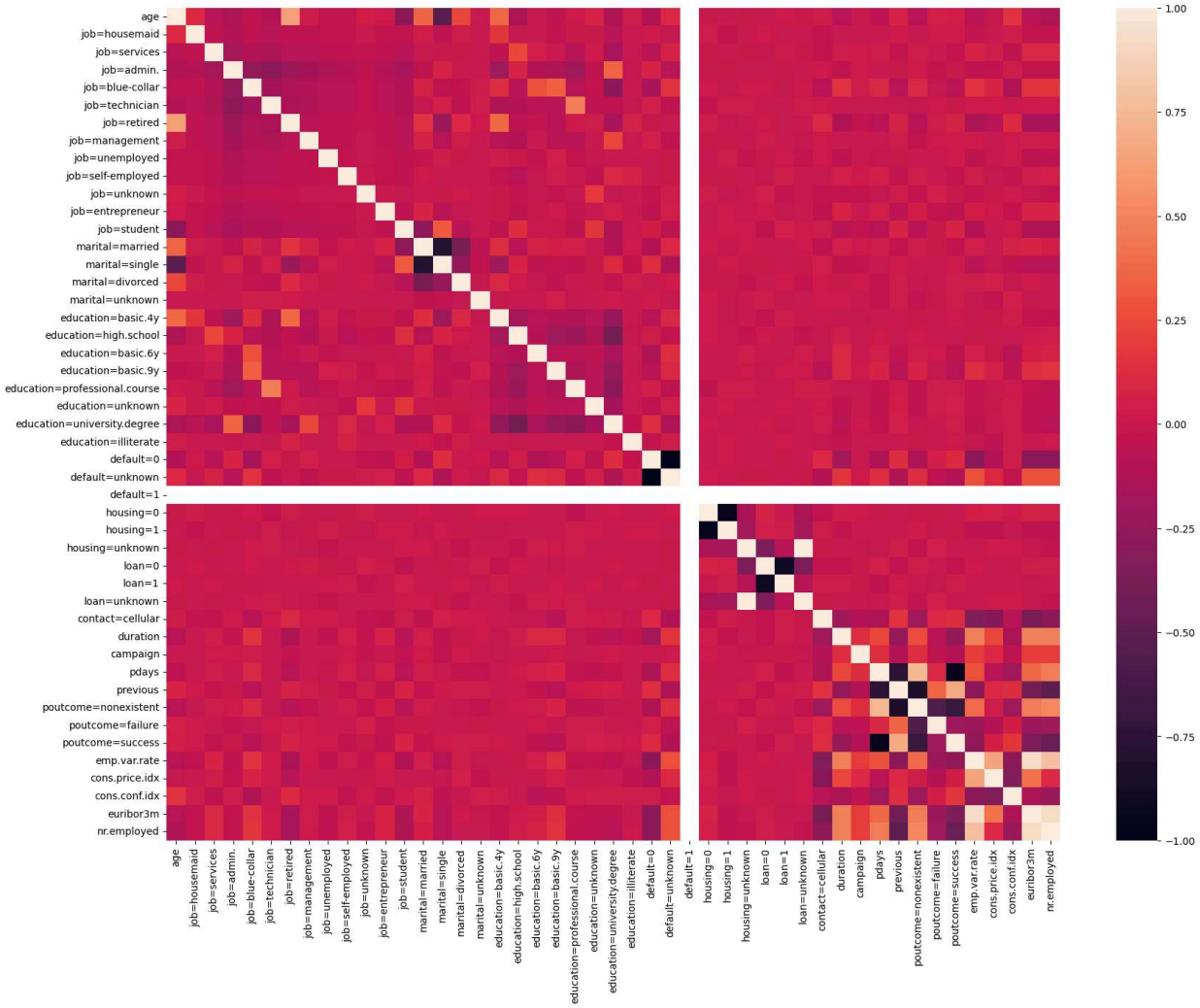
In [420…
```
corr1 = X_validation_anomalies.corr()
#sn.heatmap(corr1, annot = True)
plt.subplots(figsize=(20,15))
sns.heatmap(corr1)
```

Out[420]: <Axes: >



In [421…
```
corr2 = X_test_anomalies.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr2)
```

Out[421]: <Axes: >



Lets do the normal ones

In [422…

```
X_validation_normal
```

Out[422]:

	age	job=housemaid	job=services	job=admin.	job=blue-collar	job=technician	job=retired	j
35979	0.493827	1	0	0	0	0	0	
16088	0.283951	0	0	0	0	0	0	
31279	0.160494	0	0	1	0	0	0	
1801	0.419753	0	0	0	0	1	0	
35082	0.296296	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	
39237	0.222222	0	0	0	1	0	0	
6804	0.271605	0	0	0	1	0	0	
22717	0.172840	0	0	0	1	0	0	
36427	0.469136	0	0	0	0	0	1	
22447	0.271605	0	0	1	0	0	0	

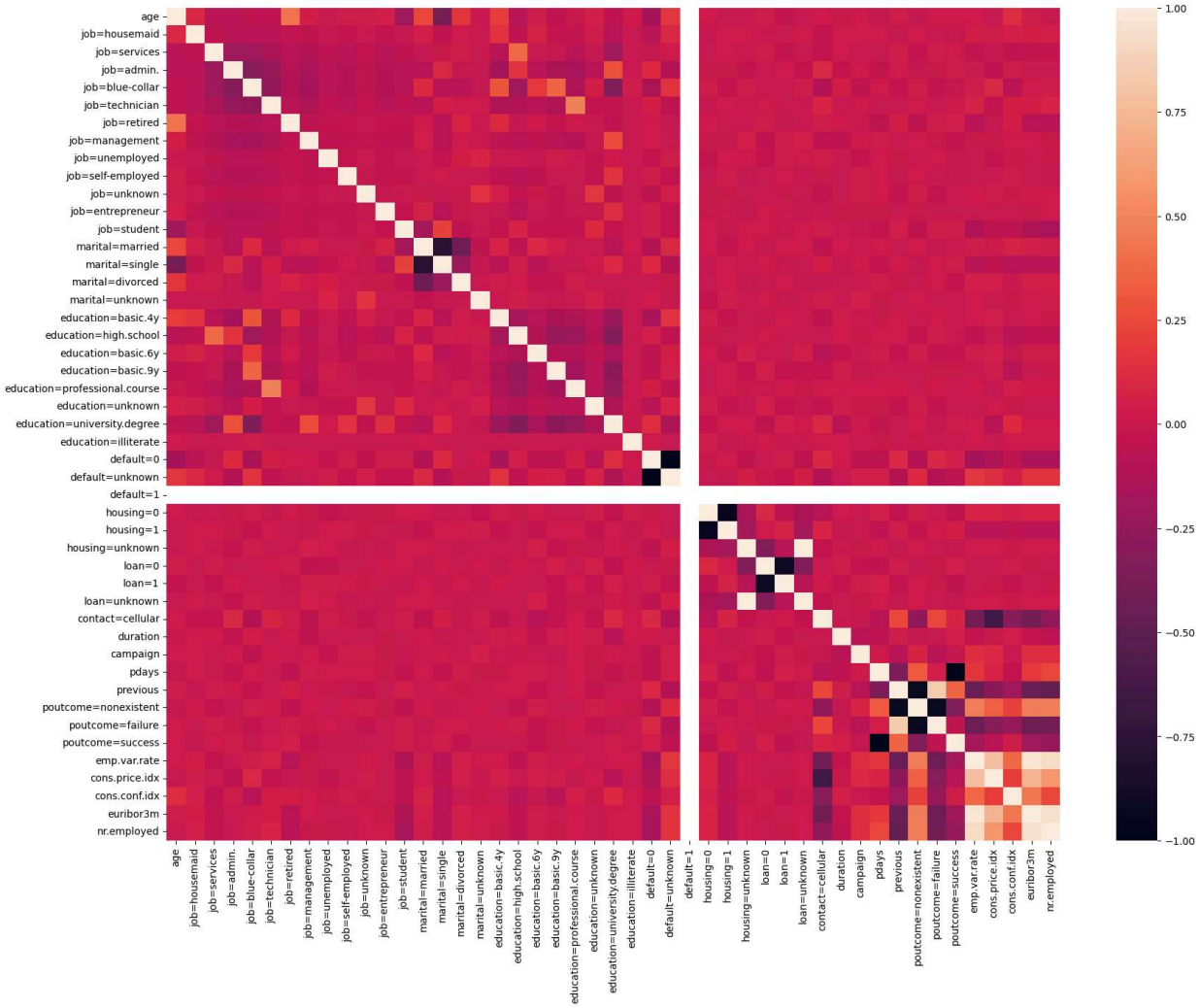
2320 rows × 47 columns

In [423…

```
corr3 = X_validation_normal.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr3)
```

Out[423]:

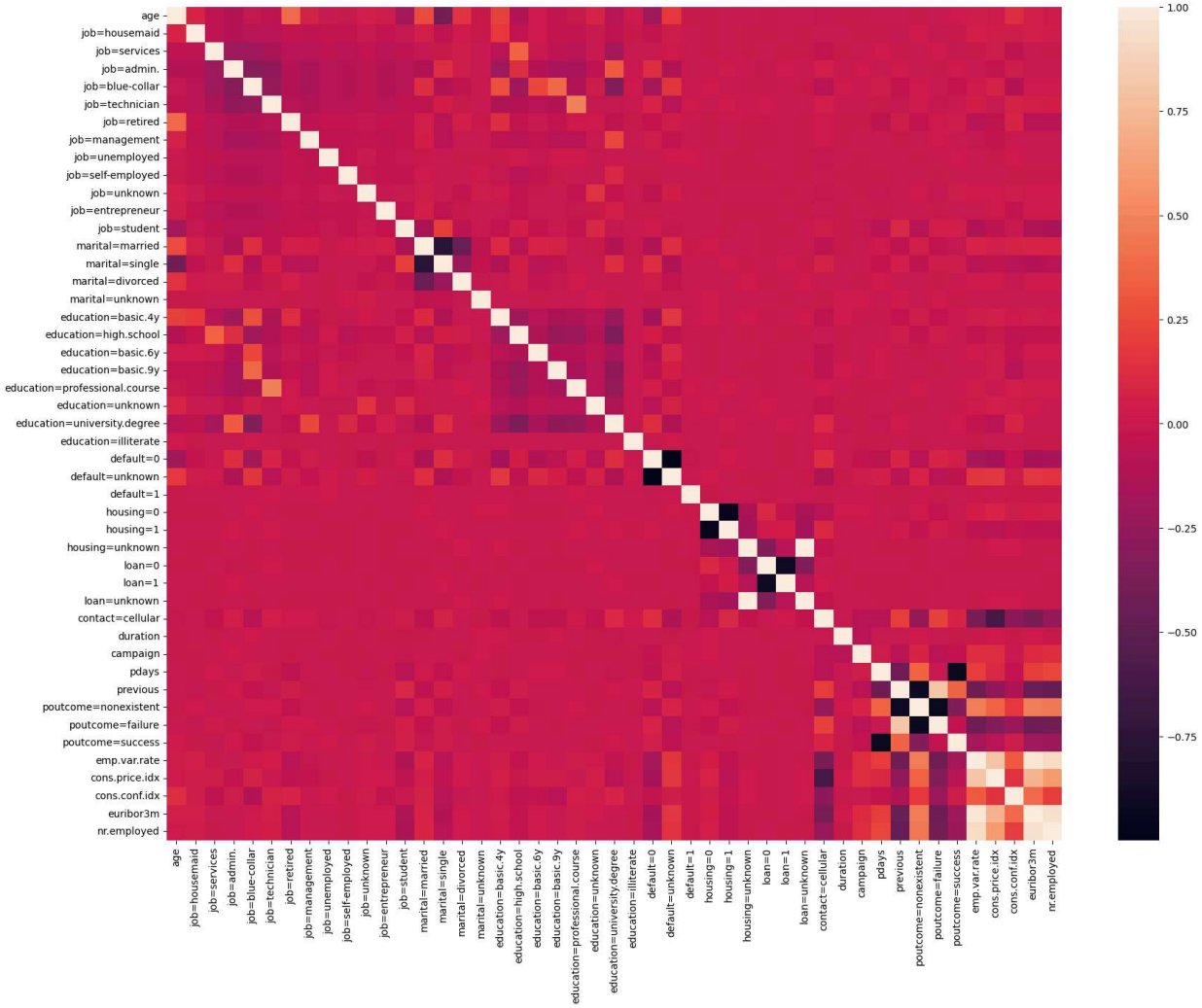
```
<Axes: >
```



In [424…

```
corr4 = X_train_normal.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr4)
```

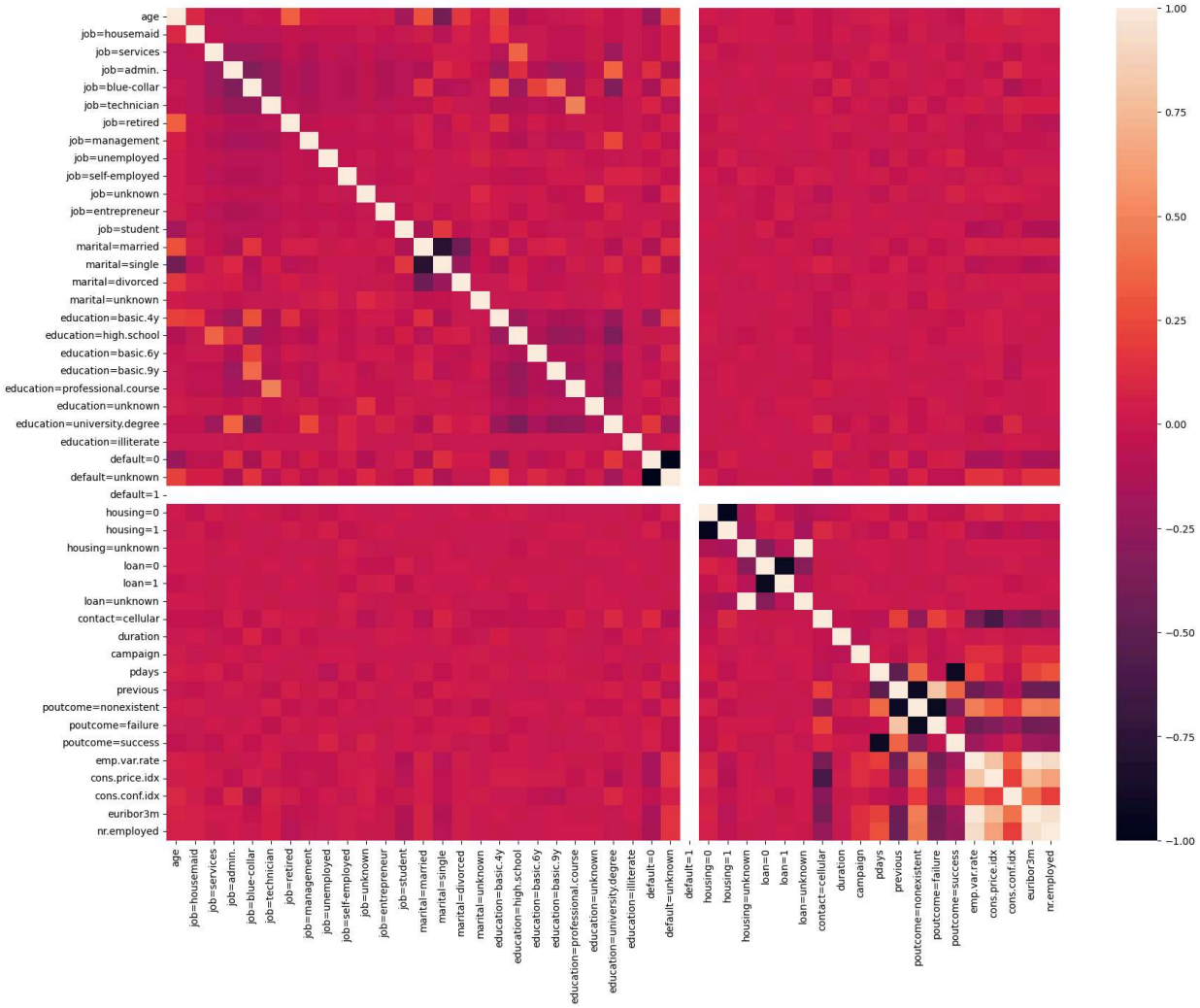
Out[424]: <Axes: >



In [425…

```
corr5 = X_test_normal.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr5)
```

Out[425]: <Axes: >



In [426…
```
sum(normalData['education=illiterate'])
```

Out[426]:
```
11
```

I was worrdied why this wasn't showing up in the test and validation but there is only 14 illiterates, harsh lable.

5.) Create an SVM based anomaly detector, set it with a variable cutoff boundary, set this to 5% to start with, but be sure it is adjustable in code. SVM gives you various options of operation, try several and find the optimum.

In [427…
```
from sklearn import svm
from sklearn.metrics import accuracy_score
```

https://www.stratascratch.com/blog/machine-learning-algorithms-explained-support-vector-machine/

Best explanation I have ever seen I finally get it.

Self notes about how this works:

Since the algorithm has to classify data points that consist of only one class, then one possibility is that the resulting Support Vector Classifier would be in a circular shape. Thus, the goal is to fit the radius of the Support Vector Classifier with some optimization algorithms.

So I can train on only one class as it only creates a radius and anything that falls outside it is an anomoly.

In [428…
```
clf_svm = svm.OneClassSVM(kernel="rbf", gamma=0.10, nu=0.15)
#clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf_svm.fit(X_train_normal)
```

Out[428]:
```
OneClassSVM(gamma=0.1, nu=0.15)
```

In [429…
```
y_predict_test_normal =  clf_svm.predict(X_test_normal)
y_predict_test_anomalies =  clf_svm.predict(X_test_anomalies)
y_predict_validation_anomalies =  clf_svm.predict(X_validation_anomalies)
y_predict_validation_normal =  clf_svm.predict(X_validation_normal)
```

In [430…
```
n_error_train = y_predict_test_normal[y_predict_test_normal == -1].size
n_error_test = y_predict_validation_normal[y_predict_validation_normal == -1].size
n_error_outliers = y_predict_test_anomalies[y_predict_test_anomalies == 1].size
```

In [431…
```
n_error_train
```

Out[431]:
```
324
```

In [432…
```
n_error_test
```

Out[432]:
```
344
```

In [433…
```
# Number incorrectly classified
n_error_outliers/len(y_test_anomalies)
```

Out[433]:
```
0.6133620689655173
```

In [434…
```
svm_predict1 = pd.Series(y_predict_test_normal).replace([-1,1],[1,0])
# Calculate accuracy
accuracy = accuracy_score(y_test_normal, svm_predict1)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8603448275862069
```

In [435…
```
svm_predict2 = pd.Series(y_predict_test_anomalies).replace([-1,1],[1,0])
# Calculate accuracy
```

```
accuracy2 = accuracy_score(y_test_anomalies, svm_predict2)
print("Accuracy:", accuracy2)
```

```
Accuracy: 0.38663793103448274
```

In [436…
```
svm_predict3 = pd.Series(y_predict_validation_anomalies).replace([-1,1],[1,0])
# Calculate accuracy
accuracy3 = accuracy_score(y_validation_anomalies, svm_predict3)
print("Accuracy:", accuracy3)
```

```
Accuracy: 0.37198275862068964
```

In [437…
```
svm_predict4 = pd.Series(y_predict_validation_normal).replace([-1,1],[1,0])
# Calculate accuracy
accuracy4 = accuracy_score(y_validation_normal, svm_predict4)
print("Accuracy: ", accuracy4)
```

```
Accuracy:  0.8517241379310345
```

I am going to leave it like this

As you decrease nu value it overfits but as you increase it gets better at recognizing outliers but wourse at recognizing normal values

I had it at nu = .1 accuracy got worse. But ofcouse that just means the circle is small and can classyfy this specific one.

6.) Produce a confusion matrix of correct and incorrect anomaly detections, using your validation data. Show this at several different boundary levels.

In [438…
```
clf_svm1 = svm.OneClassSVM(kernel="rbf", gamma=0.10, nu=0.15)
#clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf_svm1.fit(X_train_normal)
```

Out[438]:
```
OneClassSVM(gamma=0.1, nu=0.15)
```

In [439…
```
y_predict_validation_anomalies1 =  clf_svm1.predict(X_validation_anomalies)
y_predict_validation_normal1 =  clf_svm1.predict(X_validation_normal)
```

In [440…
```
y_predict_validation_anomalies1 = pd.Series(y_predict_validation_anomalies1).replace([
y_predict_validation_normal1 = pd.Series(y_predict_validation_normal1).replace([-1,1],
```

In [441…
```
clf_svm2 = svm.OneClassSVM(kernel="rbf", gamma=0.10, nu=0.10)
#clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf_svm2.fit(X_train_normal)
```

Out[441]:
```
OneClassSVM
OneClassSVM(gamma=0.1, nu=0.1)
```

In [442…
```
y_predict_validation_anomalies2 =  clf_svm2.predict(X_validation_anomalies)
y_predict_validation_normal2 =  clf_svm2.predict(X_validation_normal)
```

In [443…
```
y_predict_validation_anomalies2 = pd.Series(y_predict_validation_anomalies2).replace([
y_predict_validation_normal2 = pd.Series(y_predict_validation_normal2).replace([-1,1],
```

In [444…
```
clf_svm3 = svm.OneClassSVM(kernel="rbf", gamma=0.10, nu=0.05)
#clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf_svm3.fit(X_train_normal)
```

Out[444]:
```
OneClassSVM
OneClassSVM(gamma=0.1, nu=0.05)
```

In [445…
```
y_predict_validation_anomalies3 =  clf_svm3.predict(X_validation_anomalies)
y_predict_validation_normal3 =  clf_svm3.predict(X_validation_normal)
```

In [446…
```
y_predict_validation_anomalies3 = pd.Series(y_predict_validation_anomalies3).replace([
y_predict_validation_normal3 = pd.Series(y_predict_validation_normal3).replace([-1,1],
```

In [447…
```
clf_svm4 = svm.OneClassSVM(kernel="rbf", gamma=0.10, nu=0.01)
#clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf_svm4.fit(X_train_normal)
```

Out[447]:
```
OneClassSVM
OneClassSVM(gamma=0.1, nu=0.01)
```

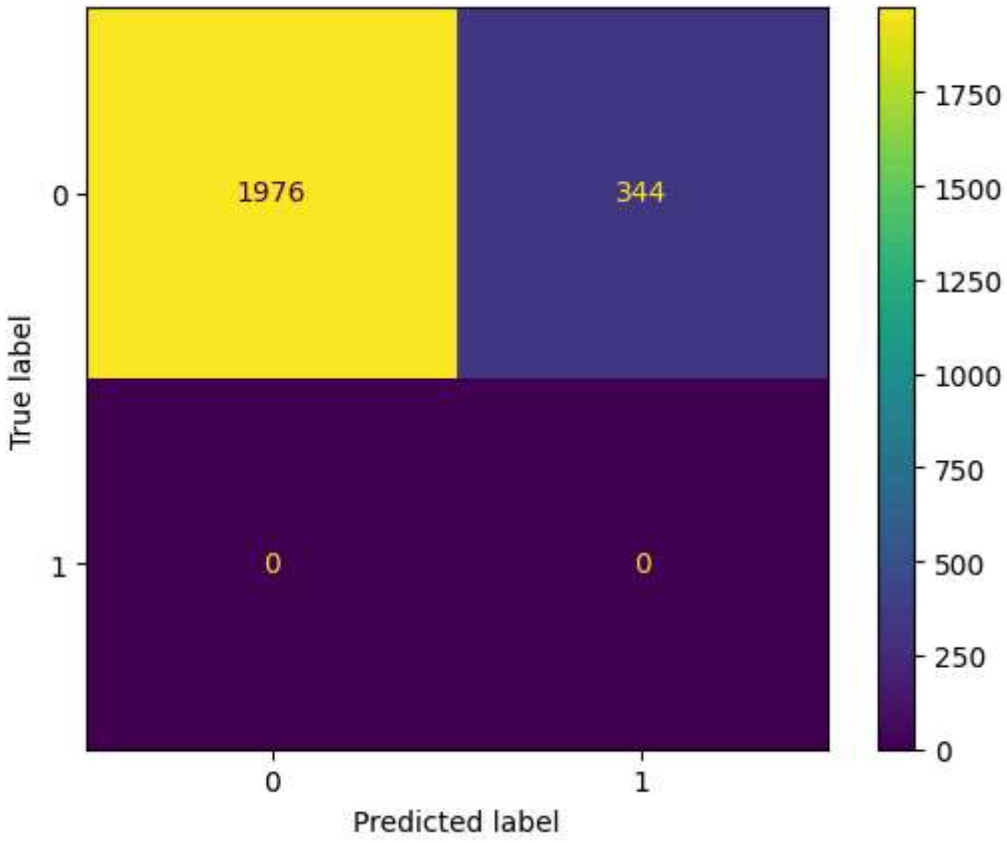
In [448…
```
y_predict_validation_anomalies4 =  clf_svm4.predict(X_validation_anomalies)
y_predict_validation_normal4 =  clf_svm4.predict(X_validation_normal)
```

In [449…
```
y_predict_validation_anomalies4 = pd.Series(y_predict_validation_anomalies4).replace([
y_predict_validation_normal4 = pd.Series(y_predict_validation_normal4).replace([-1,1],
```

In [450…
```
from sklearn.metrics import ConfusionMatrixDisplay
```

In [451…
```
ConfusionMatrixDisplay.from_predictions(y_validation_normal, y_predict_validation_norm
```

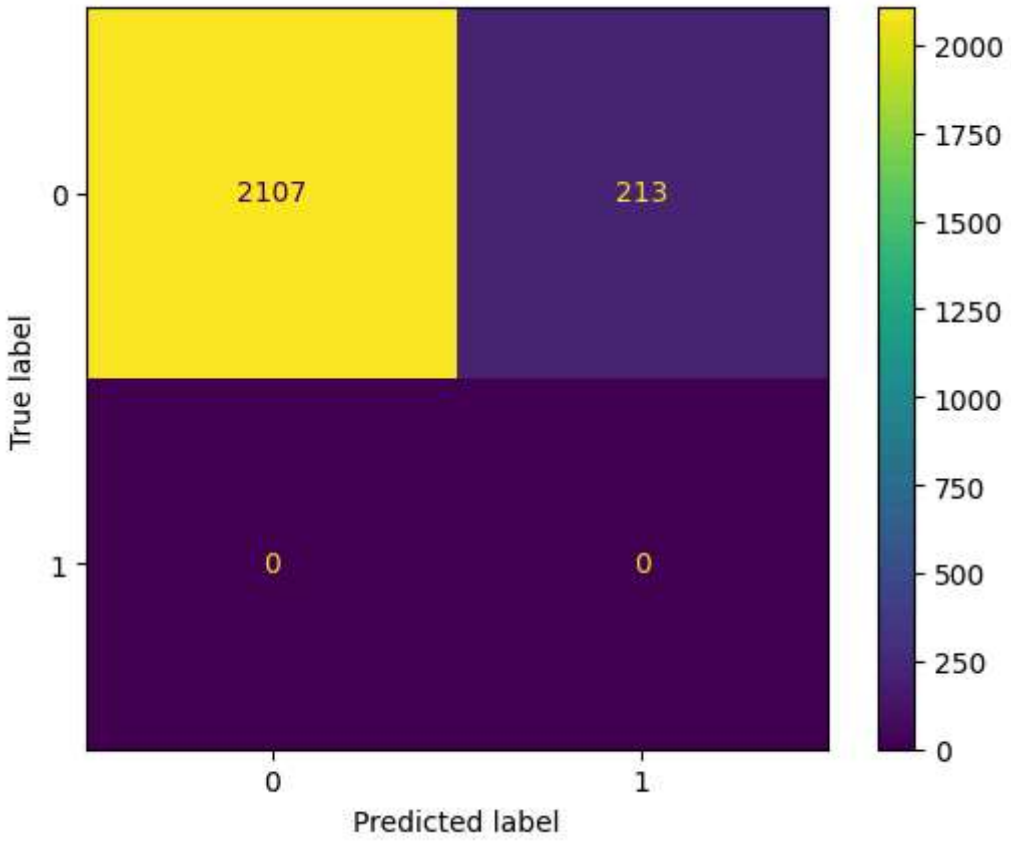
Out[451]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba20db2f070>
```



I was confused for a second about why the bottom 2 are zero but all the lables are seperate for this project so actual lables are going to all be the same 0 or 1s

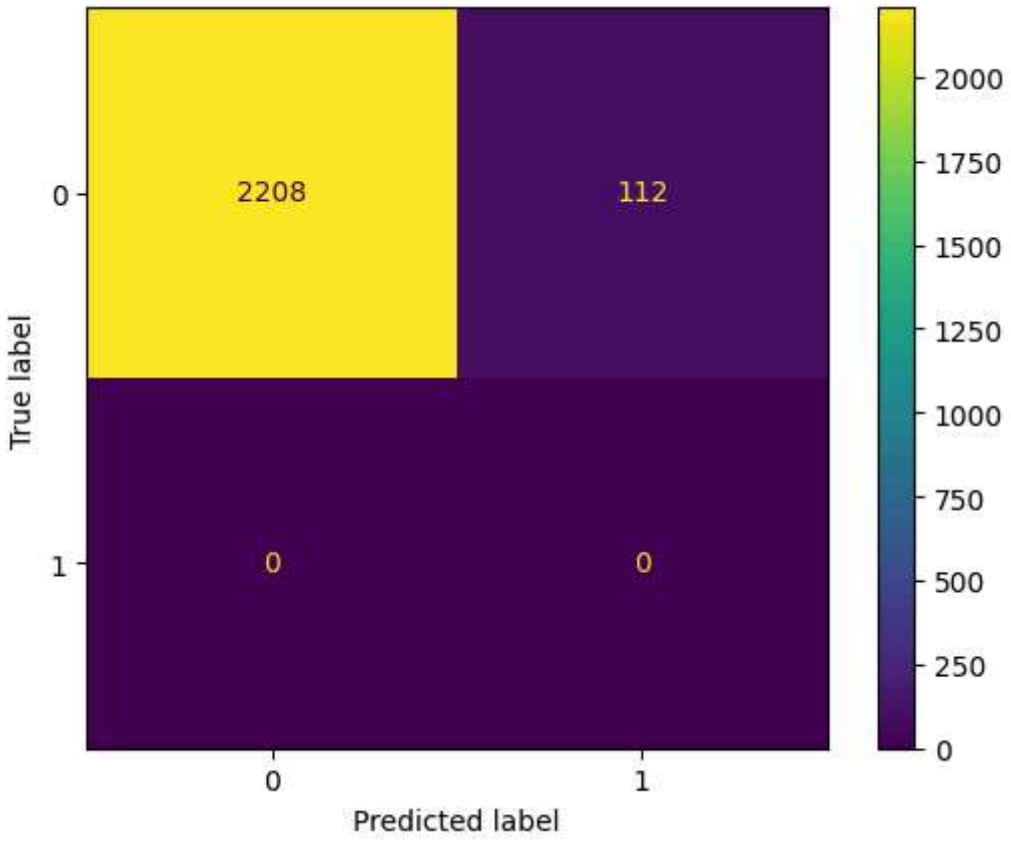
In [452…
```
ConfusionMatrixDisplay.from_predictions(y_validation_normal, y_predict_validation_norm
```

Out[452]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba20db45d20>
```



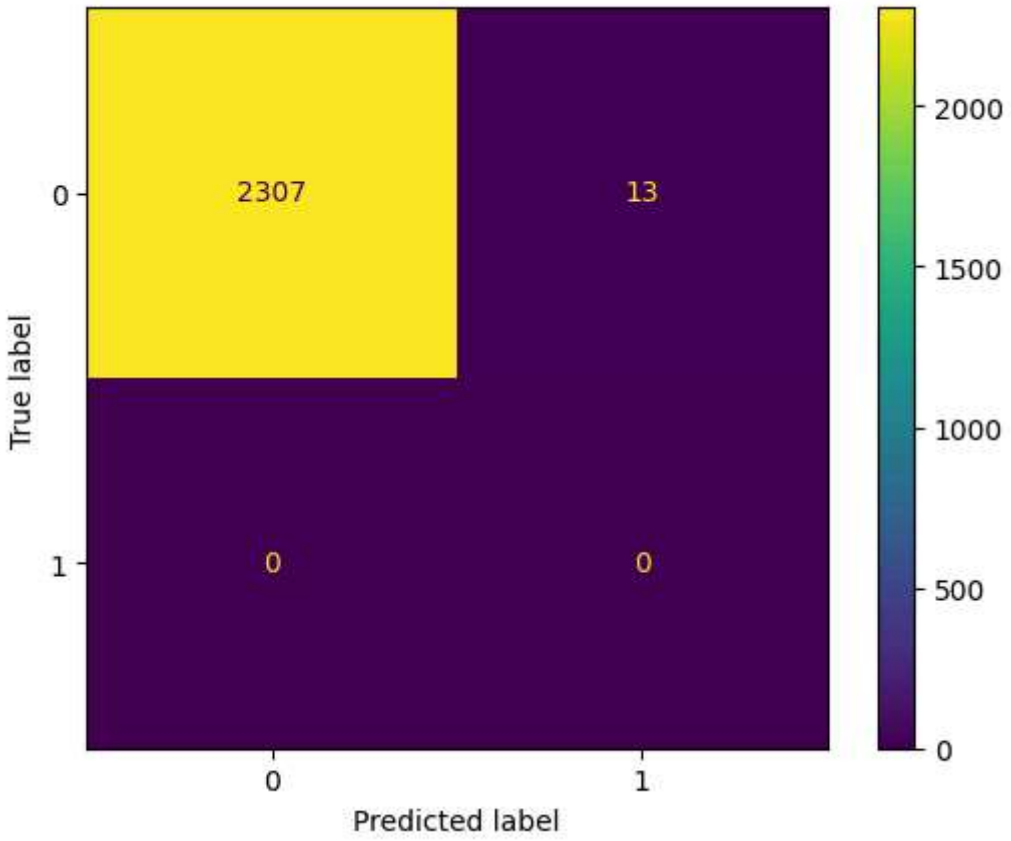
In [453…
```
ConfusionMatrixDisplay.from_predictions(y_validation_normal, y_predict_validation_norm
```

Out[453]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba20db2f1f0>
```



In [454…
```
ConfusionMatrixDisplay.from_predictions(y_validation_normal, y_predict_validation_norm
```

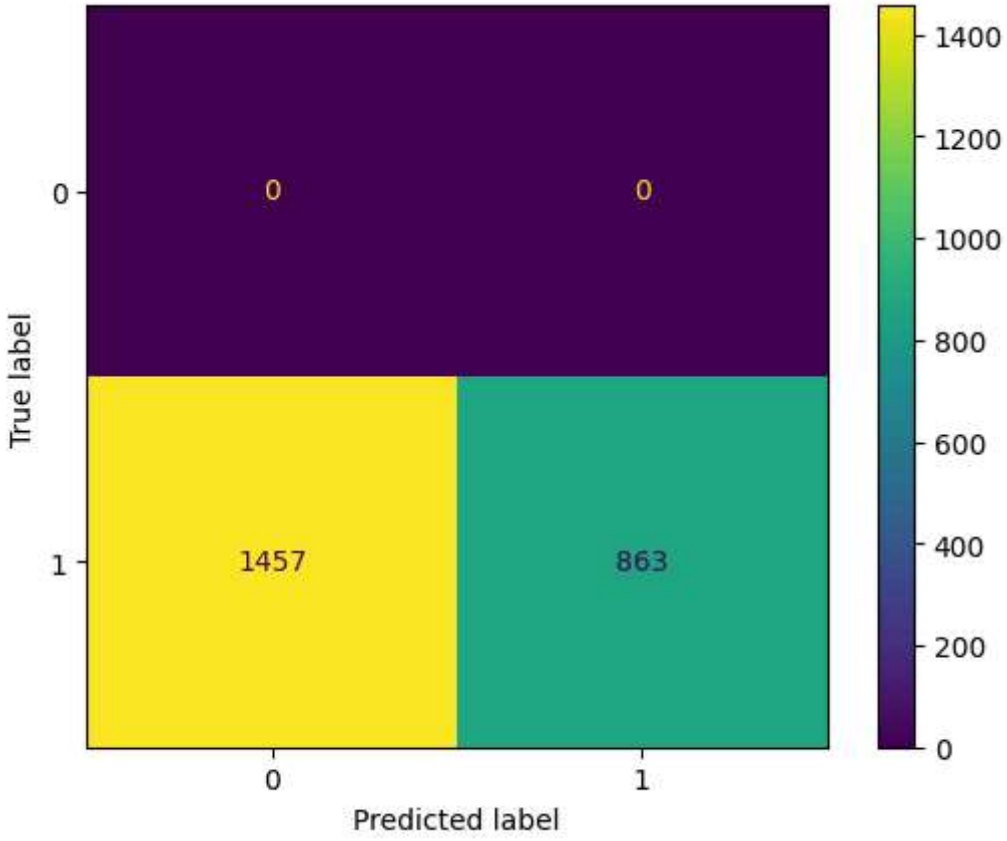
Out[454]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba20db2ffd0>
```



anomaliesData

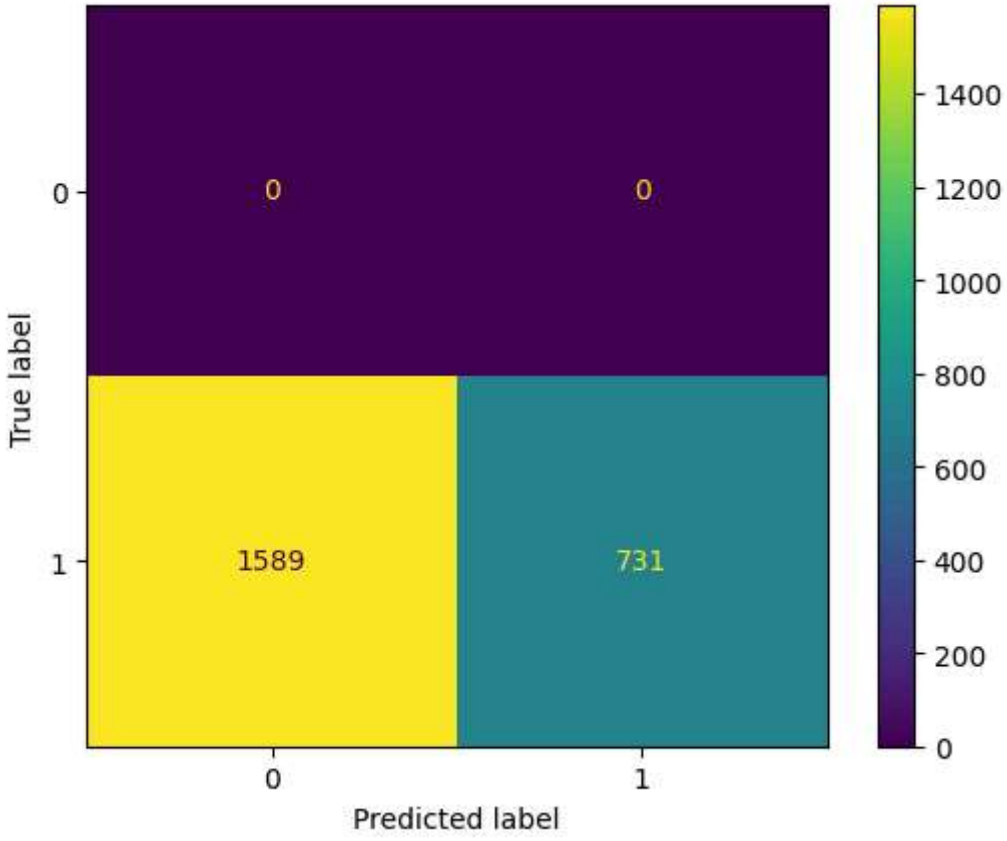
In [455…
```
ConfusionMatrixDisplay.from_predictions(y_validation_anomalies, y_predict_validation_a
```

Out[455]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba20db2e3b0>
```



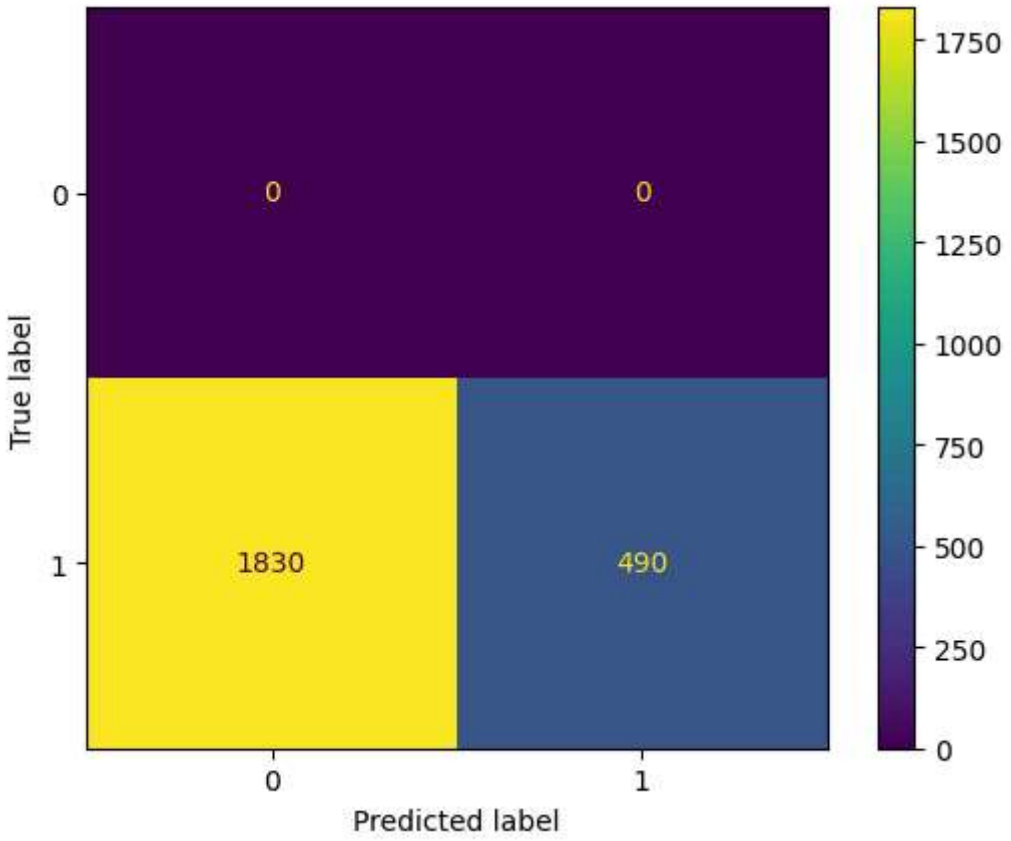
In [456…
```
ConfusionMatrixDisplay.from_predictions(y_validation_anomalies, y_predict_validation_a
```

Out[456]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba266c4aad0>
```



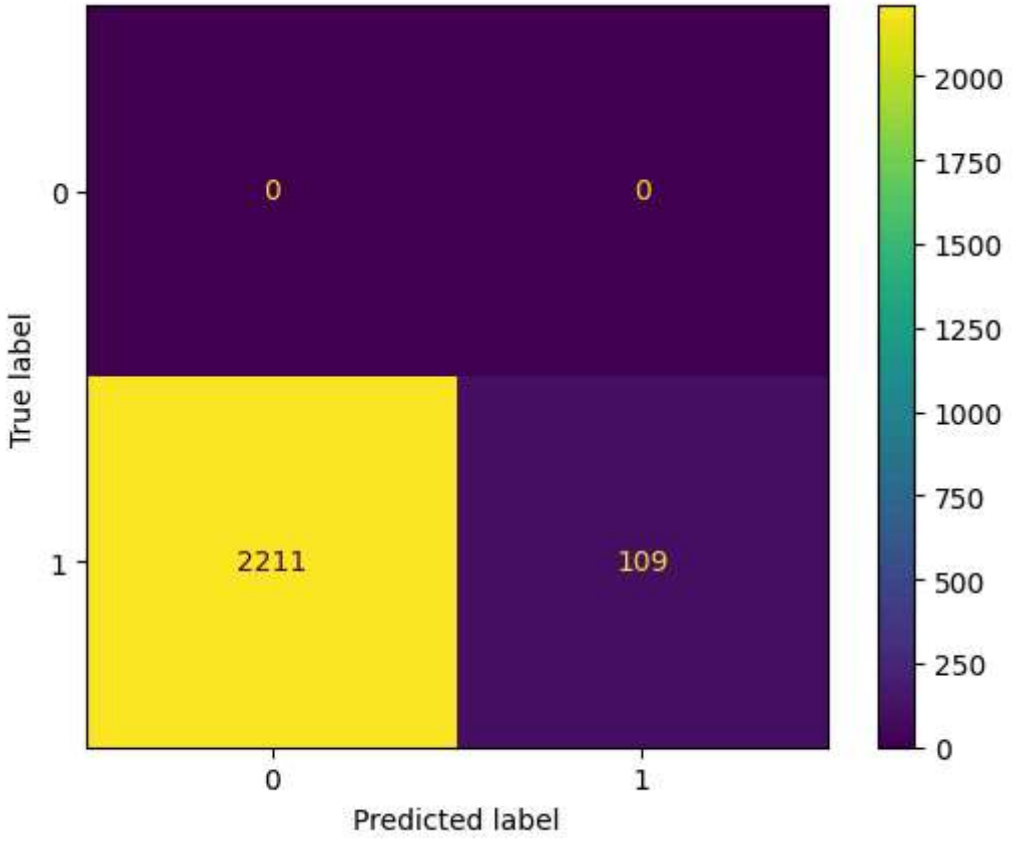
In [457…
```
ConfusionMatrixDisplay.from_predictions(y_validation_anomalies, y_predict_validation_a
```

Out[457]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba20d91a6e0>
```



In [458…
```
ConfusionMatrixDisplay.from_predictions(y_validation_anomalies, y_predict_validation_a
```

Out[458]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ba266c78880>
```



7.) Now build an anomaly detector (autoencoder) using Tensorflow and Keras. Optimize its operation at least a bit . Determine how to set up the boundary to detect anomalous and non-anomalous results.

Anomaly Detector: After all that training, it can spot anomalies. It does this by turning data into input data and comparing them. There's this thing called the "reconstruction error" – basically, how much the reconstructed data differs from the original. If the difference is too big, it's a red alert – an anomaly! Step-by-step

/www.geeksforgeeks.org

In [504…
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model
```

In [505…
```
X_train_normal.shape
```

Out[505]:
```
(31908, 47)
```

In [519…
```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Dense(32, activation="relu"),
      layers.Dense(32, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(8, activation="relu")])

    self.decoder = tf.keras.Sequential([
      layers.Dense(16, activation="relu"),
      layers.Dense(32, activation="relu"),
      layers.Dense(47, activation="sigmoid")])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = AnomalyDetector()
```

In [520…
```
autoencoder.compile(optimizer='adam', loss='mae')
```

In [521…
```
history = autoencoder.fit(X_train_normal, X_train_normal,
          epochs=20,
          batch_size=512,
          validation_data=(X_test_normal, X_test_normal),
          shuffle=True)
```

```
Epoch 1/20
63/63 [==============================] - 2s 7ms/step - loss: 0.3390 - val_loss: 0.153
5
Epoch 2/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1375 - val_loss: 0.136
3
Epoch 3/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1355 - val_loss: 0.136
1
Epoch 4/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1350 - val_loss: 0.135
2
Epoch 5/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1329 - val_loss: 0.131
2
Epoch 6/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1270 - val_loss: 0.125
7
Epoch 7/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1246 - val_loss: 0.125
3
Epoch 8/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1240 - val_loss: 0.124
3
Epoch 9/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1231 - val_loss: 0.123
7
Epoch 10/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1228 - val_loss: 0.123
6
Epoch 11/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1227 - val_loss: 0.123
6
Epoch 12/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1226 - val_loss: 0.123
6
Epoch 13/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1223 - val_loss: 0.122
9
Epoch 14/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1220 - val_loss: 0.122
8
Epoch 15/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1219 - val_loss: 0.122
9
Epoch 16/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1219 - val_loss: 0.122
8
Epoch 17/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1219 - val_loss: 0.122
7
Epoch 18/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1219 - val_loss: 0.122
6
Epoch 19/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1218 - val_loss: 0.122
7
Epoch 20/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1217 - val_loss: 0.122
6
```

In [509…
```
# I want to kill myseflf I was stuck on a bug for 2 hours! end me.
```

In [543…
```
reconstructions = autoencoder.predict(X_train_normal)
train_loss = tf.keras.losses.mae(reconstructions, X_train_normal)
```

```
998/998 [==============================] - 2s 2ms/step
```

In [544…
```
threshold = np.mean(train_loss) + np.std(train_loss)
print("Threshold: ", threshold)
```

```
Threshold:  0.1638163668246907
```

In [545…
```
reconstructions = autoencoder.predict(X_test_anomalies)
test_loss = tf.keras.losses.mae(reconstructions, X_test_anomalies)
```

```
73/73 [==============================] - 0s 2ms/step
```

In [542…
```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_test_anomalies)


mse = np.mean(np.power(X_test_anomalies - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold2 = 0.1  # You may need to adjust this threshold based on experimentation

# Classify anomalies based on the threshold
anomalies = mse > threshold2

# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_test_anomalies, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 0s 3ms/step
Test Accuracy: 0.6957
```

In [538…
```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_validation_anomalies)


mse = np.mean(np.power(X_validation_anomalies - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold = 0.1  # You may need to adjust this threshold based on experimentation

# Classify anomalies based on the threshold
anomalies = mse > threshold

# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_validation_anomalies, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 0s 3ms/step
Test Accuracy: 0.6935
```

In [539…
```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_validation_normal)


mse = np.mean(np.power(X_validation_normal - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold = 0.1  # You may need to adjust this threshold based on experimentation

# Classify anomalies based on the threshold
anomalies = mse > threshold

# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_validation_normal, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 0s 3ms/step
Test Accuracy: 0.3823
```

8.) Decide which of the two methods seems to work best, using your validation data. Can you figure out how to make simultaneous use of both methods to improve overall performance? Do the two methods make the same mistakes in anomaly detection or are there differences in the two? Once you have figured out what method, or combination of methods, works best, set it up and run it on your test data. Report the rate of correct assignment of both anomalies and regular (normal) data

In [548…
```
from sklearn.ensemble import VotingClassifier
```

In [551…
```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_validation_anomalies)


mse = np.mean(np.power(X_validation_anomalies - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold = 0.1  # You may need to adjust this threshold based on experimentation

# Classify anomalies based on the threshold
anomalies = mse > threshold
```

```
# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_validation_anomalies, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 0s 3ms/step
Test Accuracy: 0.6935
```

In [552…

```
svm_predict3 = pd.Series(y_predict_validation_anomalies).replace([-1,1],[1,0])
# Calculate accuracy
accuracy3 = accuracy_score(y_validation_anomalies, svm_predict3)
print("Accuracy:", accuracy3)
```

```
Accuracy: 0.37198275862068964
```

In [554…

```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_validation_normal)


mse = np.mean(np.power(X_validation_normal - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold = 0.1  # You may need to adjust this threshold based on experimentation

# Classify anomalies based on the threshold
anomalies = mse < threshold

# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_validation_normal, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 0s 3ms/step
Test Accuracy: 0.6177
```

In [555…

```
svm_predict4 = pd.Series(y_predict_validation_normal).replace([-1,1],[1,0])
# Calculate accuracy
accuracy4 = accuracy_score(y_validation_normal, svm_predict4)
print("Accuracy: ", accuracy4)
```

```
Accuracy:  0.8517241379310345
```

This question heavyly depeends on your threshold for your nural net and your nu value and what ratio you want for regular vs abnorbal detections.

In [557…

```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_test_anomalies)


mse = np.mean(np.power(X_test_anomalies - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold = 0.1  # You may need to adjust this threshold based on experimentation
```

```
# Classify anomalies based on the threshold
anomalies = mse > threshold

# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_test_anomalies, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 1s 7ms/step
Test Accuracy: 0.6957
```

In [558…

```
svm_predict2 = pd.Series(y_predict_test_anomalies).replace([-1,1],[1,0])
# Calculate accuracy
accuracy2 = accuracy_score(y_test_anomalies, svm_predict2)
print("Accuracy:", accuracy2)
```

```
Accuracy: 0.38663793103448274
```

Nural net wins

In [559…

```
# Evaluate the Autoencoder
predictions = autoencoder.predict(X_test_normal)


mse = np.mean(np.power(X_test_anomalies - predictions, 2), axis=1)

# Set a threshold for anomaly detection
threshold = 0.1  # You may need to adjust this threshold based on experimentation

# Classify anomalies based on the threshold
anomalies = mse > threshold

# Evaluate the Anomaly Detection Model

y_pred = anomalies.astype(int)

accuracy = accuracy_score(y_test_normal, y_pred)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
73/73 [==============================] - 0s 2ms/step
Test Accuracy: 0.1961
```

In [560…

```
svm_predict1 = pd.Series(y_predict_test_normal).replace([-1,1],[1,0])
# Calculate accuracy
accuracy = accuracy_score(y_test_normal, svm_predict1)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8603448275862069
```

Now SVM wins.