Q1) Total students = 150

$P(\text{corona}) = \frac{2}{100}$ [for each student].

$P(\text{Omicron} \mid \text{Corona}) = \frac{95}{100}$.

P(entire class is free of corona). = $S_1 \& S_2 \& S_3 \& \ldots \& S_n$ students do not have corona

i.e $\left(1 - \frac{2}{100}\right)^{150} = \left[\left(\frac{98}{100}\right)^{150}\right]$. = no student has corona.

same as. Student 1 does not have corona and ($\cap$)
Student 2 does not have corona and ($\cap$)
:
student 150 does not have corona.

∴ These are independent events, we can multiply them & this results to

$\left[P\left(\text{A student has no corona}\right)\right]^{150}$.

Given: $P(\text{Omicron} \mid \text{Corona}) = 0.95$

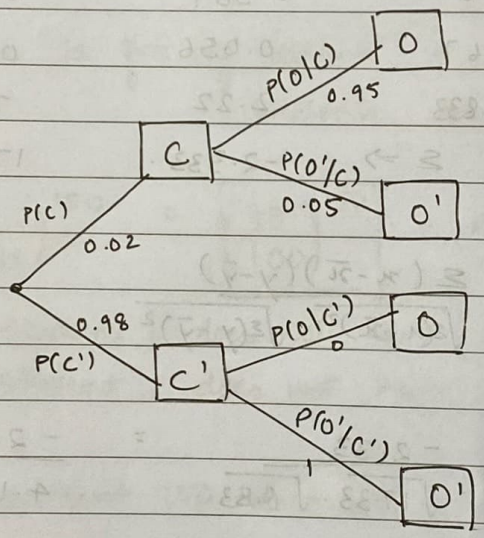
Let C be event that student has Corona. $P(C) = 0.02$

C' = student does not have Corona. $P(C') = 0.98$.

Let O be event that student has omicron = $P(O)$

O' be event that student does not have omicron.

The tree diagram will be



$= P(O \cap C) = (0.95)(0.02)$

$= P(O' \cap C) = (0.95)(0.05)$ (0.02)

$= P(O \cap C') = O \times 0.98$

$= P(O' \cap C') = 1 \times 0.98$

$P(O'|C') = 1 \therefore P(O|C') = 0$.

$\therefore$ This means that if a student does not have corona, then p of that this student has omicron = none = 0

$P(O'|C') = 1 - P(O|C')$

$= 1 - 0$

$= 1$

$P(O|C) = 0.95$ (Given).

$P(O'|C) = 1 - P(O|C)$

$= 1 - 0.95$

$= 0.05$.

P(No student has corona).

let p be the probability that a student has omicron

p' = probability that student does not have omicron

From tree diagram,

p' = (student has corona AND does not have omicron) OR (student does not has corona AND does not have omicron).

$= P(C \cap O') \text{ OR } P(C' \cap O')$

$= P(C \cap O') + P(C' \cap O')$

$= (0.95)(0.02) +$

$= (0.02)(0.05) + (1)(0.98)$

$= 0.001 + 0.98$

$p = \underline{0.981}$

~~Using~~ Binomial concept where p

∴ Prob of No student having omicron becomes

$$= \boxed{(0.981)^{150}}$$

2) - m objects = m data points.
- divided into G groups = some no of different classes.
- where i group is of size m_i

i.e 1 Group size = m_1
2 Group size = m_2
3 Group size = m_3, etc.

- Goal: to obtain sample of size n. , $n < m$.

(a) randomly select $\frac{n \times m_i}{m}$, i.e = $m_i \times \left(\frac{n}{m}\right)$.

here $\frac{n}{m}$ = constant for $m_1 \times \left(\frac{n}{m}\right)$ & $m_2 \times \left(\frac{n}{m}\right)$.

hence we are keeping the % or ratio of data elements taken from each class constant, this is also called stratified sampling where proportions are maintained.

eg: $m = 100$, $m_1 = 60$, $m_2 = 40$ $[m = m_1 + m_2]$.
$n = 20$ = sample size

$\frac{n}{m} = \frac{20}{100} = 20\%$.

sampling: for m_1: $m_1 \times \frac{n}{m} = 60 \times \frac{20}{100} = 12$

for m_2: $m_2 \times \frac{n}{m} = 40 \times \frac{20}{100} = 8$

$\therefore$ $12 + 8 = 20$ = sample size.

i.e $\frac{12}{20} = \frac{6}{10} = \frac{60}{100} = \underline{60\%}$ of m_1 in sample
= same as $\underline{60\%}$ m_1 in data.

and. $\frac{8}{20} = \frac{4}{10} = \frac{40}{100} = \underline{40\%}$ of m_2 in sample
same as $\underline{40\%}$ of m_2 in data

Since proportion is maintained in the sample, this is a better representation of entire population.

eg: When analyzing purchases made by customers, maintaining the proportions of Male & Female customers ensures that the sample represents the actual (Total) data well & not randomly because there can be way more M or F customers in sample [M & F customer's purchased items may differ hugely].

This can be used when the class imbalance is significant, which will ensure that we have a significant amount (equal %) of all classes in the sample.

(b) randomly select n objects, = random sampling
every object has equal probability of selection.

If we can't use stratified sampling i.e if the data is too huge with multiple classes or maintaining the proportions does not make sense or the classes are balanced (then the random sample will be approximately same as stratified one).

eg: 500 data points of class A & B each
so since the probability of selecting one data element randomly is same for both object in A & B, hence no point of using stratified sampling here.

(population)

I think, when the data is always updated i.e sensor data which keeps on generating in regular intervals, since the population size & % of each class keeps on changing every time we can't keep track of the proportions everytime & hence simple random sampling seems the best option here.

3) $x = (1, 0, 0, 1, 1, 1)$

$y = (1, 1, 1, 0, 0, 1)$

A] (a) $\cos(x, y) = \frac{x \cdot y}{|x| \cdot |y|}$ = vector dot product / lengths multiplied

$$= \frac{(1\times1) + (0\times1) + (0\times1) + (1\times0) + (1\times0) + (1\times1)}{\sqrt{1^2+0^2+0^2+1^2+1^2+1^2} \times \sqrt{1^2+1^2+1^2+0^2+0^2+1^2}}$$

$$= \frac{1+0+0+0+0+1}{\sqrt{4} \times \sqrt{4}} = \frac{2}{4} = \frac{1}{2}$$

(b) $d(x, y) = \sqrt{(x-y)^2}$ = Euclidean

$$= \sqrt{(1-1)^2 + (0-1)^2 + (0-1)^2 + (1-0)^2 + (1-0)^2 + (1-1)^2}$$

$$= \sqrt{0+1+1+1+1+0}$$

$$= 2$$

(c) $\text{Corr}(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \cdot \sqrt{\sum (y_i - \bar{y})^2}}$

$\bar{x} = \frac{1+0+0+1+1+1}{6} = 1$, $\bar{y} = \frac{4}{4} = 1$

$$\text{Corr} = \frac{(1-1)(1-1) + (0-1)(1-1) + (0-1)(1-1) + (1-1)(0-1) + (1-1)(0-1) + (1-1)(1-1)}{\sqrt{(1-1)^2+(0-1)^2+(0-1)^2+(1-1)^2+(1-1)^2+(1-1)^2} \times \sqrt{(1-1)^2+(1-1)^2+(1-1)^2+(0-1)^2+(0-1)^2+(1-1)^2}}$$

$$= \frac{0+0+0+0+0+0}{\sqrt{2} \cdot \sqrt{2}} = 0$$

(c) $\bar{x} = \frac{1+0+0+1+1+1}{6} = \frac{4}{6} = \frac{2}{3} = 0.67$

$\bar{y} = \frac{1+1+1+0+0+1}{6} = \frac{4}{6} = \frac{2}{3} = 0.67$

$$\text{corr}(x,y) = \frac{\Sigma (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma (x_i - \bar{x})^2}\sqrt{\Sigma (y_i - \bar{y})^2}}$$

$$= \frac{\left(1-\frac{2}{3}\right)\left(1-\frac{2}{3}\right) + \left(0-\frac{2}{3}\right)\left(1-\frac{2}{3}\right) + \left(0-\frac{2}{3}\right)\left(1-\frac{2}{3}\right) + \left(1-\frac{2}{3}\right)\left(0-\frac{2}{3}\right) + \left(1-\frac{2}{3}\right)\left(0-\frac{2}{3}\right) + \left(1-\frac{2}{3}\right)\left(1-\frac{2}{3}\right)}{\sqrt{\left(1-\frac{2}{3}\right)^2 + \left(0-\frac{2}{3}\right)^2 + \left(0-\frac{2}{3}\right)^2 + \left(1-\frac{2}{3}\right)^2 + \left(1-\frac{2}{3}\right)^2 + \left(1-\frac{2}{3}\right)^2} \times \sqrt{\left(1-\frac{2}{3}\right)^2 + \left(1-\frac{2}{3}\right)^2 + \left(1-\frac{2}{3}\right)^2 + \left(0-\frac{2}{3}\right)^2 + \left(0-\frac{2}{3}\right)^2 + \left(1-\frac{2}{3}\right)^2}}$$

$$= \frac{\frac{1}{3}\left[\frac{1}{3} \; \frac{-2}{3} \; \frac{-2}{3} \; \frac{-2}{3} \; \frac{-2}{3} \; \frac{1}{3}\right]}{\sqrt{\left(\frac{1}{3}\right)^2 + \left(\frac{-2}{3}\right)^2 + \left(\frac{-2}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2} \sqrt{\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{-2}{3}\right)^2 + \left(\frac{-2}{3}\right)^2 + \left(\frac{1}{3}\right)^2}}$$

$$= \frac{\frac{1}{3}\left[\frac{-6}{3}\right]}{\frac{1}{3}\left[\sqrt{1+4+4+1+1+1} \cdot \sqrt{1+1+1+4+4+1}\right]}$$

$$= \frac{\frac{-6}{3}}{\sqrt{12} \cdot \sqrt{12}} = \frac{-6}{3 \times 12} = \boxed{\frac{-1}{6}}.$$

$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})(y-\bar{y})$	$(x-\bar{x})^2$	$(y-\bar{y})^2$
0.33	0.33	0.11	0.11	0.11
-0.67	0.33	-0.22	0.44	0.11
-0.67	0.33	-0.22	0.44	0.11
0.33	-0.67	-0.22	0.11	0.44
0.33	-0.67	-0.22	0.11	0.44
0.33	0.33	0.11	0.11	0.11
$\Sigma \Rightarrow$ -	-	-0.66	1.32	1.32.

$$\text{Corr}(x,y) = \frac{\Sigma (x-\bar{x})(y-\bar{y})}{\sqrt{\Sigma(x-\bar{x})^2} \cdot \sqrt{\Sigma(y-\bar{y})^2}}$$

$$= \frac{-0.66}{\sqrt{1.32} \cdot \sqrt{1.32}}$$

$$= -\frac{0.66}{1.32.} = \boxed{-0.5}$$

(d) Jaccard.

$f_{00} = \#[x=0 \ \& \ y=0] = 0$

$f_{01} = \#[x=0 \ \& \ y=1] = 2$

$f_{10} = \#[x=1 \ \& \ y=0] = 2$

$f_{11} = \#[x=1 \ \& \ y=1] = 2$

$$J = \frac{f_{11}}{f_{01}+f_{10}+f_{11}} = \frac{\text{matching pairs}}{\text{not involved in "00" matching}}$$

$$= \frac{2}{2+2+2} = \frac{2}{6} = \frac{1}{3}.$$

B]

$x = (1, -2, 0, 2, 0, -3)$

$y = (-1, 2, -1, 0, 0, -1)$.

-0.178.

(a) $\cos(x,y) = \frac{x \cdot y}{|x| \cdot |y|} = \frac{(-1)+(-4)+(0)+(0)+(0)+(3)}{\sqrt{1+4+0+4+0+9} \cdot \sqrt{1+4+1+0+0+1}}$

$= \frac{-2}{\sqrt{18} \cdot \sqrt{7}} = \frac{-2}{\sqrt{18}\sqrt{7}\sqrt{4} \cdot \sqrt{7}}$ ~~$\frac{-2}{7 \times 2}$~~ $= \frac{-1}{7}$

(b) $d(x,y) = \sqrt{(1+1)^2 + (-4)^2 + (1)^2 + (2)^2 + (0)^2 + (-2)^2}$

$= \sqrt{4+16+1+4+4} = \sqrt{29}$.

(c) ~~$corr(x,y) = \&$~~

$\bar{x} = \frac{1-2+2-3}{6} = \frac{-2}{6}$ ~~$= -0.5$~~

$\bar{y} = \frac{-1+2-1-1}{6} = \frac{-1}{6}$

~~$\Sigma(x_i - \bar{x}) = \left(1+\frac{2}{6}\right) + \left(-2+\frac{2}{6}\right) + \left(0-\frac{2}{6}\right)$~~

~~$+ \left(2-\frac{2}{6}\right) + ($~~

(c) Corr (x, y). $X = (1, -2, 0, 2, 0, -3)$ $\bar{x} = -1/3$

$Y = (-1, 2, -1, 0, 0, -1)$ $\bar{y} = -1/6$.

$x - \bar{x}$	$y - \bar{y}$	$(x-\bar{x})(y-\bar{y})$	$(x-\bar{x})^2$	$(y-\bar{y})^2$
1.33	-0.833	-1.111	1.778	0.694
-1.67	2.167	-3.611	2.778	4.694
0.33	-0.833	-0.278	0.111	0.694 ~~0.028~~
2.33	0.167	0.389	5.444	0.028
0.33	0.167	0.056	0.111	0.028
-2.67	-0.833	2.22	7.111	0.694
	$\Sigma \Rightarrow$	-2.333.	17.33	6.83.

$$\text{Corr}(x, y) = \frac{\Sigma (x-\bar{x})(y-\bar{y})}{\sqrt{\Sigma (x-\bar{x})^2}\ \sqrt{\Sigma (y-\bar{y})^2}}$$

$$= \frac{-2.33}{\sqrt{17.33} \cdot \sqrt{6.83}} = \frac{-2.33}{4.163 \times 2.613}$$

$$= \boxed{-0.214}.$$

Question 4

Loading the dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
```

In [2]:
```
data_og = pd.read_csv('T2D_abundance.csv', sep='\t')
df = data_og.copy()
print(df.shape)
```

```
(344, 574)
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	k__Archaea|p__Euryarchaeota|c__Methanobacteria|o__Methanobacteriales|f__Methanc
0	con-001	
1	con-002	
2	con-003	
3	con-004	
4	con-005	

5 rows × 574 columns

In [4]:
```
df.describe()
```

Out[4]:

	k__Archaea|p__Euryarchaeota|c__Methanobacteria|o__Methanobacteriales|f__Methanobacteri
count	
mean	
std	
min	
25%	
50%	
75%	
max	

8 rows × 572 columns

PCA

- *Checking out the min, max and mean, we can see that the scales of attributes aren't close enough, so I guess Normalization will be needed.*

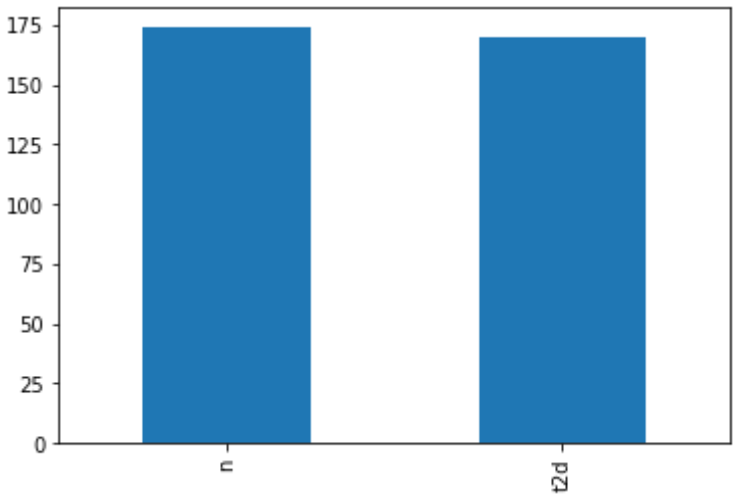
In [5]:
```
# Check is there are any missing values
isna_out = df.isna().sum()

count = 0
for i in range(len(isna_out)):
    if isna_out[i] != 0:
        count+=1
print(f'Columns with missing values: {count}')
```

```
Columns with missing values: 0
```

In [6]:
```
print(df['Class'].value_counts())
df['Class'].value_counts().plot.bar()
```

```
n      174
t2d    170
Name: Class, dtype: int64
```

Out[6]:
```
<AxesSubplot:>
```



In [7]:
```
from sklearn.decomposition import PCA

# create a copy of dataframe so that the original data is not modified in the p
df = data_og.copy()
# Label(strings) can't be processed in PCA, so remove it from the DF and store
label = df.pop('Class')
# Remove the first column, since it has string values
df.drop('Unnamed: 0', axis=1, inplace=True)

# PCA with 5 components
pca = PCA(n_components=5)
pca.fit(df)
print(f'Before Normalizing: {pca.explained_variance_ratio_}')
```

```
print(f'Shape before PCA: {df.shape}')
df_transformed = pca.transform(df)
print(f'Shape after PCA: {df_transformed.shape}')
```

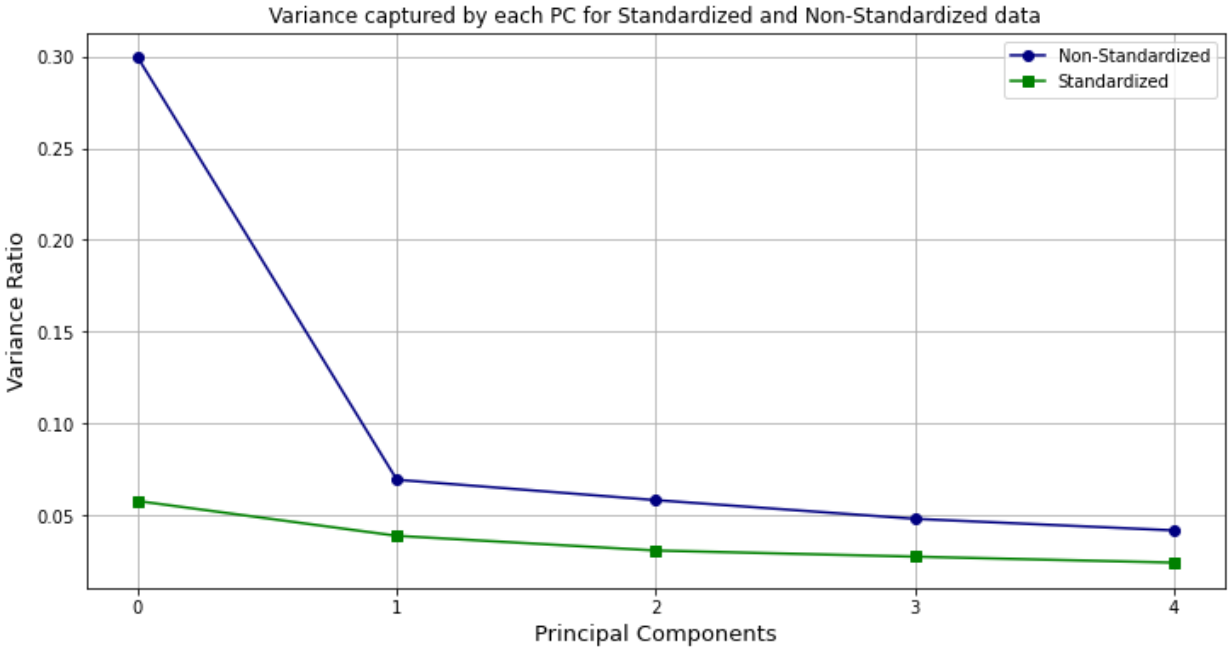
```
Before Normalizing: [0.29922075 0.06936047 0.05825253 0.04807923 0.04177536]
Shape before PCA: (344, 572)
Shape after PCA: (344, 5)
```

In [8]:
```
# Standard scaling the data and then performing PCA on it using the previous st

from sklearn.preprocessing import StandardScaler
df = data_og.copy()
label = df.pop('Class')
df.drop('Unnamed: 0', axis=1, inplace=True)

dfs = StandardScaler().fit_transform(df)

pca2 = PCA(n_components=5)
pca2.fit(dfs)
print(f'After Normalizing: {pca2.explained_variance_ratio_}')

print(f'Shape before PCA: {dfs.shape}')
dfs_transformed = pca2.transform(dfs)
print(f'Shape after PCA: {dfs_transformed.shape}')
```

```
After Normalizing: [0.05774328 0.0388078  0.03080242 0.02745622 0.02420982]
Shape before PCA: (344, 572)
Shape after PCA: (344, 5)
```

Plot Variance ratios from the PCA

In [11]:
```
# Plotting the variance ratios for both PCAs where we can visualize %variance c

fig, ax = plt.subplots(figsize=(12,6))
plt.title('Variance captured by each PC for Standardized and Non-Standardized c

ax.plot(pca.explained_variance_ratio_, color='navy', marker='o', label='Non-Sta
ax.plot(pca2.explained_variance_ratio_, color='green', marker='s', label='Stanc
ax.set_xlabel('Principal Components', fontsize='13')
ax.set_ylabel('Variance Ratio', fontsize='13')

ax.set_xticks(np.arange(0,5))
ax.legend()
ax.grid()
plt.show()
```



For Non-Standardized Data

- *Since the values for each Attribute has different ranges, most of the variance is captured by the first component and then the variance takes a steep drop for all other components.*
- *The scale on which these attributes are not mentioned(I guess if the attributes are in same scale then we don't need to standardize) so in this case, checking out the mean, min, max we can assume that standardization will help improve PCA results.*
- *Anyways, the First PC captures 30% of the variance, and the second around 7%, so they both capture around 37% of the variance. Ideally we require 75-95% of the variance to better represent the original data.*

*
*

For Standardized Data

- *The first and second components count for around (5.7+3.8) ~9% of the variance, this is not good either.*

Scatter plots for PCA with 2 PC's

In [12]:
```
from sklearn.decomposition import PCA

# create a copy of dataframe so that the original data is not modified in the p
df = data_og.copy()
# Label(strings) can't be processed in PCA, so remove it from the DF and store
label = df.pop('Class')
# Remove the first column, since it has string values
df.drop('Unnamed: 0', axis=1, inplace=True)

# PCA with 2 components
```

```
pca = PCA(n_components=2)
pca.fit(df)
print(f'Before Normalizing: {pca.explained_variance_ratio_}')

print(f'Shape before PCA: {df.shape}')
df_transformed = pca.transform(df)
print(f'Shape after PCA: {df_transformed.shape}')
print()
# Standard scaling the data and then performing PCA on it using the previous st

from sklearn.preprocessing import StandardScaler
df = data_og.copy()
label = df.pop('Class')
df.drop('Unnamed: 0', axis=1, inplace=True)

dfs = StandardScaler().fit_transform(df)

pca2 = PCA(n_components=2)
pca2.fit(dfs)
print(f'After Normalizing: {pca2.explained_variance_ratio_}')

print(f'Shape before PCA: {dfs.shape}')
dfs_transformed = pca2.transform(dfs)
print(f'Shape after PCA: {dfs_transformed.shape}')
```

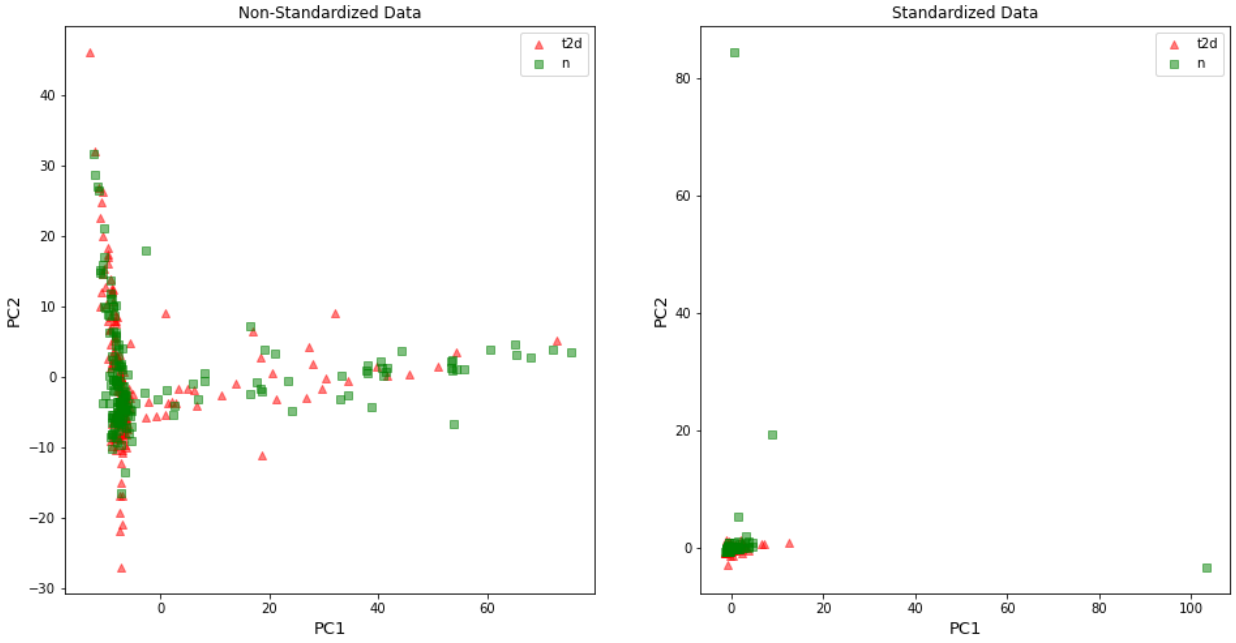
```
Before Normalizing: [0.29922075 0.06936047]
Shape before PCA: (344, 572)
Shape after PCA: (344, 2)

After Normalizing: [0.05774328 0.0388078 ]
Shape before PCA: (344, 572)
Shape after PCA: (344, 2)
```

In [13]:
```
# Attatch labels to the processed data
df_transformed = pd.DataFrame(df_transformed)
df_transformed['Class']=label
df_transformed.rename(columns={0:'PC1',1:'PC2'},inplace=True)
print(df_transformed.head())
print()

dfs_transformed = pd.DataFrame(dfs_transformed)
dfs_transformed['Class']=label
dfs_transformed.rename(columns={0:'PC1',1:'PC2'},inplace=True)
print(dfs_transformed.head())
```

```
         PC1       PC2 Class
0  -2.943049 -2.248392     n
1  -7.215752  1.835211     n
2  24.051681 -4.851263     n
3  -8.684207  6.432474     n
4   2.565706 -3.969767     n

        PC1       PC2 Class
0 -1.141024 -0.438906     n
1 -0.968028 -0.449684     n
2  3.600220 -0.127532     n
3 -0.564942 -0.295775     n
4 -0.814759 -0.414534     n
```

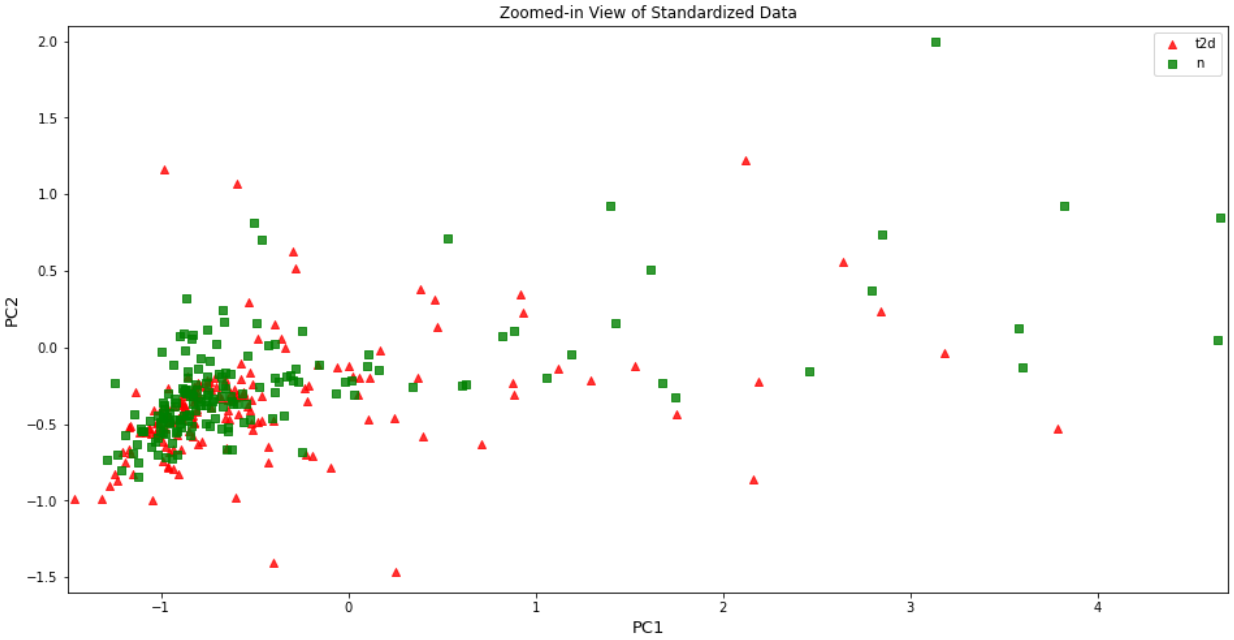
In [14]:
```
# Scatter plots where each point represent a person
```

```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,8))
ax1.title.set_text('Non-Standardized Data')

temp = df_transformed[df_transformed['Class']=='t2d']
ax1.scatter(temp['PC1'],temp['PC2'], label='t2d', color='red', marker='^', alph
temp = df_transformed[df_transformed['Class']=='n']
ax1.scatter(temp['PC1'],temp['PC2'], label='n', color='green', marker='s', alph
ax1.set_xlabel('PC1', fontsize='13')
ax1.set_ylabel('PC2', fontsize='13')
ax1.legend()

ax2.title.set_text('Standardized Data')
temp = dfs_transformed[dfs_transformed['Class']=='t2d']
ax2.scatter(temp['PC1'],temp['PC2'], label='t2d', color='red', marker='^', alph
temp = dfs_transformed[df_transformed['Class']=='n']
ax2.scatter(temp['PC1'],temp['PC2'], label='n', color='green', marker='s', alph
ax2.set_xlabel('PC1', fontsize='13')
ax2.set_ylabel('PC2', fontsize='13')
ax2.legend()
plt.show()
```



In [191…

```
# Zooming the 2nd plot a bit, towards the main cluster

fig, ax2 = plt.subplots(figsize=(16,8))

ax2.title.set_text('Zoomed-in View of Standardized Data')

temp = dfs_transformed[dfs_transformed['Class']=='t2d']
ax2.scatter(temp['PC1'],temp['PC2'], label='t2d', color='red', marker='^', alph
temp = dfs_transformed[df_transformed['Class']=='n']
ax2.scatter(temp['PC1'],temp['PC2'], label='n', color='green', marker='s', alph
ax2.set_xlabel('PC1', fontsize='13')
ax2.set_ylabel('PC2', fontsize='13')
ax2.legend()

plt.axis([-1.5, 4.7, -1.6, 2.1])

plt.show()
```



Summary for Question 4(PCA)

- *Using PCA(2 components) we are able to capture around 9%(Normalized data) of the variability.*
- *Here PCA with 2 components does not seem a viable approach of dimensonality reduction for this dataset, bacause only 9% of the total variance is captured, generally speaking we need more than 75% for good representation of the actual data*
- *The clusters of people with T2D and No-Diabetes are overlapping, so this means that PCA with 2 componenets is not useful in this case, there may be a non-linear relationship thats why, since PCA is only capable of establishing linear relationship with the actual data, this can be a reason that results from PCA aren't that good.*

In [184…

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, learning_rate= 'auto', random_state=1)
out_tsne = tsne.fit_transform(df)

df_tsne = pd.DataFrame({'Class':label})
df_tsne['C1'] = out_tsne[:,0]
df_tsne['C2'] = out_tsne[:,1]

print(out_tsne[0])
print(df_tsne)
```

```
/Users/rushank/opt/anaconda3/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:780: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
```

```
[-5.876167 -2.363706]
    Class         C1         C2
0       n  -5.876167  -2.363706
1       n  -0.812157   0.647112
2       n -15.971931  -2.868236
3       n   6.349443   2.529033
4       n  -8.368555  -2.368832
..    ...        ...        ...
339     n   0.652143  -2.947324
340     n   8.353833   6.606295
341     n  13.548102  -7.695888
342     n   1.997106  -2.456973
343     n  10.238225 -11.014678

[344 rows x 3 columns]
```

In [189…

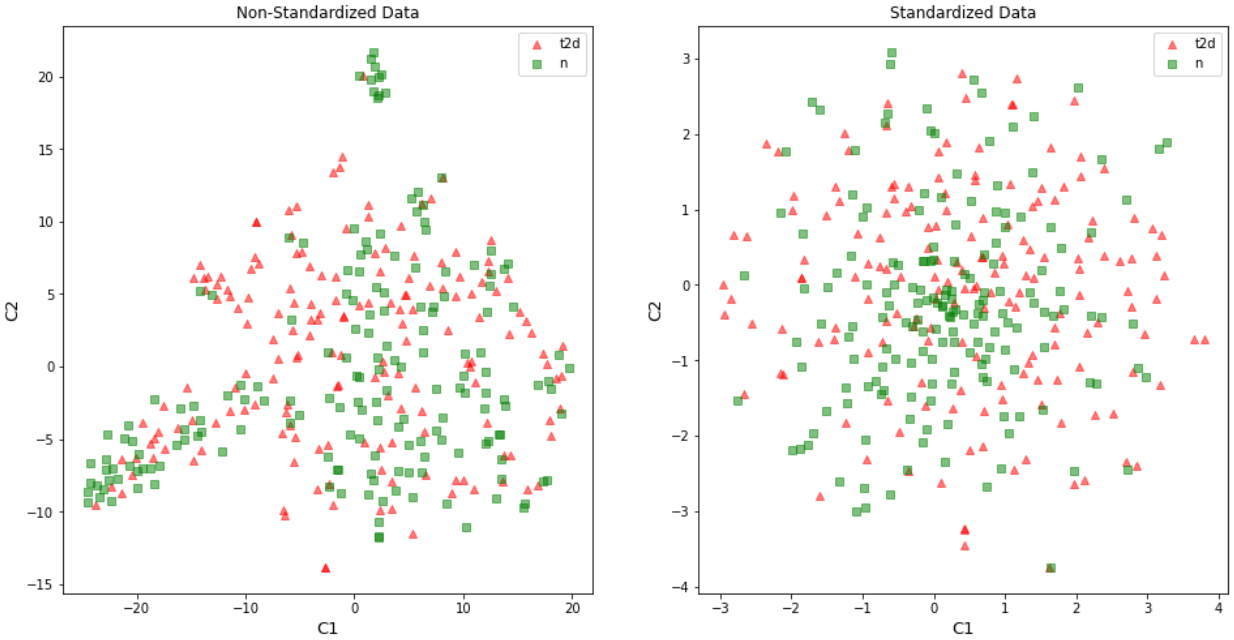
```
tsne = TSNE(n_components=2, learning_rate='auto', random_state=1, verbose=1, pe
out_tsne = tsne.fit_transform(dfs)

dfs_tsne = pd.DataFrame({'Class':label})
dfs_tsne['C1'] = out_tsne[:,0]
dfs_tsne['C2'] = out_tsne[:,1]

print(out_tsne[0])
print(dfs_tsne)
```

```
/Users/rushank/opt/anaconda3/lib/python3.9/site-packages/sklearn/manifold/_t_s
ne.py:780: FutureWarning: The default initialization in TSNE will change from
'random' to 'pca' in 1.2.
  warnings.warn(
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 344 samples in 0.001s...
[t-SNE] Computed neighbors for 344 samples in 0.018s...
[t-SNE] Computed conditional probabilities for sample 344 / 344
[t-SNE] Mean sigma: 5.197160
[t-SNE] KL divergence after 250 iterations with early exaggeration: 70.903633
[t-SNE] KL divergence after 1000 iterations: 1.067005
[ 2.7272556 -2.4521618]
    Class        C1        C2
0       n  2.727256 -2.452162
1       n  0.722584 -0.970531
2       n -0.733959 -0.422877
3       n -0.130944  0.310664
4       n  2.968985 -1.227048
..    ...       ...       ...
339     n -0.957849 -2.042692
340     n -1.941234 -0.744129
341     n -0.697166  2.146732
342     n -2.768622 -1.523944
343     n -1.121923 -0.978937

[344 rows x 3 columns]
```

In [190…

```
# Scatter plots where each point represent a person

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,8))
ax1.title.set_text('Non-Standardized Data')

temp = df_tsne[df_tsne['Class']=='t2d']
ax1.scatter(temp['C1'],temp['C2'], label='t2d', color='red', marker='^', alpha=
```

```
temp = df_tsne[df_tsne['Class']=='n']
ax1.scatter(temp['C1'],temp['C2'], label='n', color='green', marker='s', alpha=
ax1.set_xlabel('C1', fontsize='13')
ax1.set_ylabel('C2', fontsize='13')
ax1.legend()

ax2.title.set_text('Standardized Data')
temp = dfs_tsne[dfs_tsne['Class']=='t2d']
ax2.scatter(temp['C1'],temp['C2'], label='t2d', color='red', marker='^', alpha=
temp = dfs_tsne[dfs_tsne['Class']=='n']
ax2.scatter(temp['C1'],temp['C2'], label='n', color='green', marker='s', alpha=
ax2.set_xlabel('C1', fontsize='13')
ax2.set_ylabel('C2', fontsize='13')
ax2.legend()
plt.show()
```



- *t-SNE's performance seems better(not much) than PCA because the overlapping between classes has reduced a lot, cuters are kind of visually noticable than were in PCA*

- *Anyways, t-SNE does not seem good(better than PCA) for dimensionality reduction for this dataset, But tuning the hyperparameters can result in improved results.*

Question 5

A funtion that returns list of euclidean distances between any two points in the given dataframe

In [4]:
```
def get_dsts(v):
    dsts = []
    row1=0
    while row1<v.shape[0]:
```

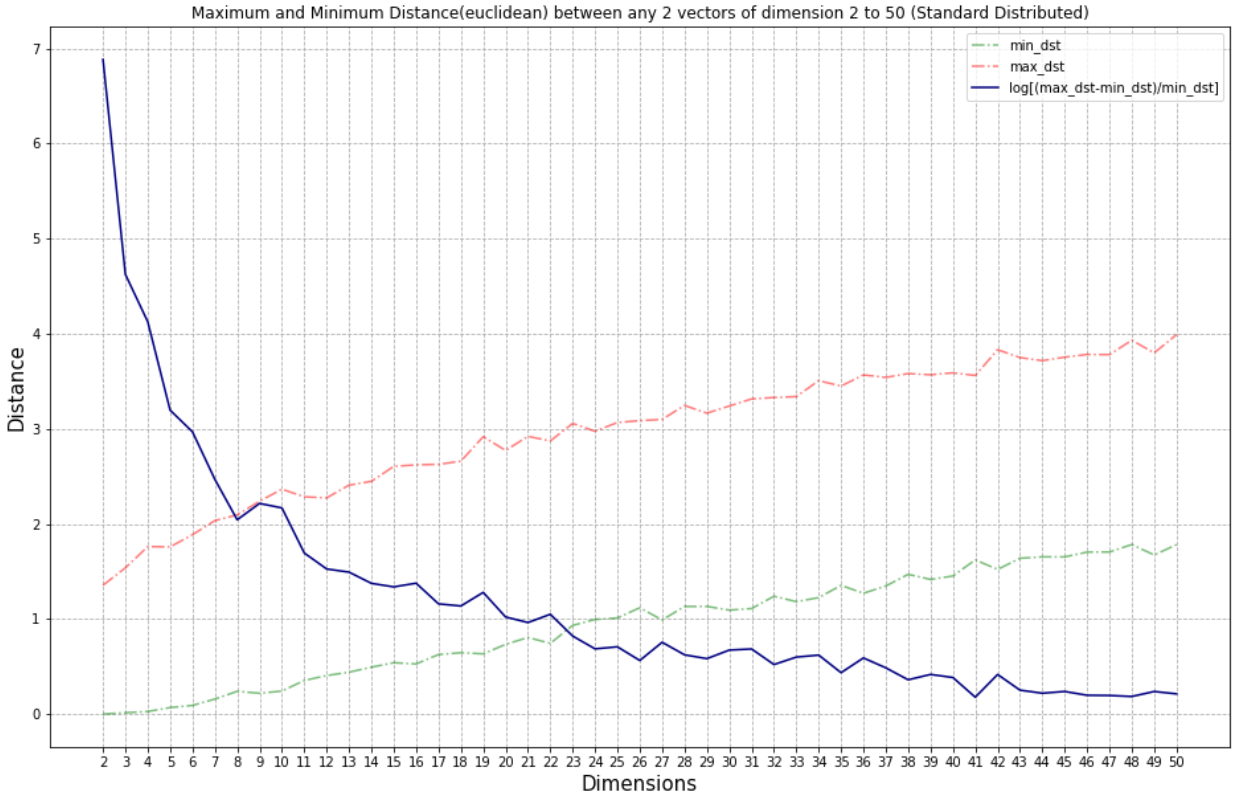
```
        for row2 in range(row1+1,v.shape[0]):
    #         print(v.iloc[row1][0])
    #         print(v.iloc[row2][0])
            dsts.append(np.linalg.norm(v.iloc[row1]-v.iloc[row2]))
    #         print()
        row1+=1
    return dsts
```

Generate dataframe(with 2 to 50 dimensions), compute distances between each 500 points, and find the maximum and minimum distance for one dataframe with dimension 'n'

In [15]:
```
arr_max = []
arr_min = []
for dim in tqdm(range(2,51)):
    v = pd.DataFrame(np.random.rand(500,dim))
    dsts = get_dsts(v)
    arr_max.append(max(dsts))
    arr_min.append(min(dsts))
```

```
  0%|          | 0/49 [00:00<?, ?it/s]
```

In [24]:
```
final = []
for i in range(len(arr_max)):
    final.append(np.log((arr_max[i]-arr_min[i])/arr_min[i]))


fig, ax = plt.subplots(figsize=(16,10))
ax.plot(list(range(2,51)),arr_min, label='min_dst', color='green', ls='-.', alp
ax.plot(list(range(2,51)),arr_max, label='max_dst', color='red', ls='-.', alpha
ax.plot(list(range(2,51)), final, label='log[(max_dst-min_dst)/min_dst]', colo
# diff = []
# for i in range(len(arr_min)):
#     temp = arr_max[i]-arr_min[i]
#     diff.append(temp)

# ax.plot(list(range(2,51)), diff, label='max_dst - min_dst', color='navy', lin

ax.set_xticks(np.arange(2,51, step=1))
ax.set_yticks(np.arange(0,8, step=1))
plt.xlabel('Dimensions', fontsize='15')
plt.ylabel('Distance', fontsize='15')
ax.legend()
plt.grid(ls='--')
plt.title('Maximum and Minimum Distance(euclidean) between any 2 vectors of dim
plt.show()
```


Maximum and Minimum Distance(euclidean) between any 2 vectors of dimension 2 to 50 (Standard Distributed)
min_dst
max_dst
log[(max_dst-min_dst)/min_dst]
Distance
Dimensions
0
1
2
3
4
5
6
7
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 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

In [ ]: