Introduction

Research around the human microbiome (the collection of bacteria and microbes that live in and on our bodies) has been growing as a science and field of interest more and more over the last decade (Ursell et al., 2012). Many people experience disruptions to their health that could be attributed to an imbalance of these little organisms that share life with each of us (Ursell et al., 2012). Gastrointestinal (GI/gut) health is of particular interest, and investigations around how to improve this ecosystem within us are expanding. One approach to achieve this is Fecal Microbiota Transplant (FMT).

FMT is a method of transferring a healthy population (high species quantity and diversity) of microbes from one person to another in hopes of restoring the bacterial colonies in the recipient and thus contributing to greater overall health (Orr, et al. 2018). FMT has been used as a therapy for a limited selection of illnesses (and is not approved for many uses in the USA), but it is gaining relevance as a therapeutic application for a wide variety of infections, gastrointestinal issues, and auto-immune disorders (Gupta et al., 2016). FMT could be an effective intervention for a variety of health issues caused by diet, environment, overuse of antibiotics, and other factors.

In order to successfully implement an intervention such as this, we need to understand the structure of human microbiota. Using data sourced from NIH Human Microbiome Project, I will run an EDA to explore the species types and prevalence. For the EDA portion, I will be roughly following this Kaggle notebook. The dataset utilized for that project is older (as the project is in constant motion), so although it is a great framework, my approach will have to be adjusted. Moving beyond this step, I would like to build models to predict the presence of these microbes to potentially I will likely reference this or this dataset.

According to AWS: "The NIH-funded Human Microbiome Project (HMP) is a collaborative effort of over 300 scientists from more than 80 organizations to comprehensively characterize the microbial communities inhabiting the human body and elucidate their role in human health and disease. To accomplish this task, microbial community samples were isolated from a cohort of 300 healthy adult human subjects at 18 specific sites within five regions of the body (oral cavity, airways, urogenital track, skin, and gut). Targeted sequencing of the 16S bacterial marker gene and/or whole metagenome shotgun sequencing was performed for thousands of these samples. In addition, whole genome sequences were generated for isolate strains collected from human body sites to act as reference organisms for analysis. Finally, 16S marker and whole metagenome sequencing was also done on additional samples from people suffering from several disease conditions."

- Gupta, S., Allen-Vercoe, E., & Petrof, E. O. (2016). Fecal microbiota transplantation: in perspective. Therapeutic Advances in Gastroenterology, 9(2), 229-239. https://doi.org/10.1177/1756283X15607414
- Orr, M. R., Kocurek, K. M., & Young, D. L. **(that's me!)** (2018). Gut Microbiota and Human Health: Insights From Ecological Restoration. The Quarterly Review of Biology, 93(2), 73–90. https://doi.org/10.1086/698021
- Ursell, L. K., Metcalf, J. L., Parfrey, L. W., & Knight, R. (2012). Defining the Human Microbiome. Nutrition reviews, 70(Suppl 1), S38. https://doi.org/10.1111/j.1753-4887.2012.00493.

Import and Cleaning

In [1]:
```
#import libraries and dataset
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

#part 2
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

microbes=pd.read_csv('/Users/debane/Documents/MS Data Science/550 Data Minir
#####pd.set_option('display.max_rows', None, 'display.max_columns', None)
```

In [2]:
```
#View sample of data
# microbes.head() #commented out for brevity
```

Out[2]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Projec Statu
0	1	Gi03551	Abiotrophia defectiva ATCC 49176	BACTERIAL	Bacteria	oral	Complet
1	4	Gi03555	Achromobacter piechaudii ATCC 43553	BACTERIAL	Bacteria	airways	Complet
2	5	Gi03554	Achromobacter xylosoxidans C54	BACTERIAL	Bacteria	airways	Complet
3	10	Gi03422	Acinetobacter baumannii ATCC 19606	BACTERIAL	Bacteria	urogenital_tract	Complet
4	12	Gi03421	Acinetobacter calcoaceticus RUH2202	BACTERIAL	Bacteria	skin	Complet

In [3]:
```
#Check shape
microbes.shape
```

Out[3]:
```
(2915, 19)
```

In [4]:
```
#See column names in dataset
microbes.columns
```

Out[4]:
```
Index(['HMP ID', 'GOLD ID', 'Organism Name', 'Domain', 'NCBI Superkingdom',
       'HMP Isolation Body Site', 'Project Status', 'Current Finishing Leve
l',
       'NCBI Submission Status', 'NCBI Project ID', 'Genbank ID', 'Gene Cou
nt',
       'IMG/HMP ID', 'HOMD ID', 'Sequencing Center', 'Funding Source',
       'Strain Repository ID', 'Unnamed: 17', 'Unnamed: 18'],
      dtype='object')
```

In [5]:
```
#View descriptions of data
microbes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2915 entries, 0 to 2914
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   HMP ID                    2915 non-null   int64
 1   GOLD ID                   1783 non-null   object
 2   Organism Name             2915 non-null   object
 3   Domain                    2712 non-null   object
 4   NCBI Superkingdom         2751 non-null   object
 5   HMP Isolation Body Site   2915 non-null   object
 6   Project Status            2915 non-null   object
 7   Current Finishing Level   1579 non-null   object
 8   NCBI Submission Status    2915 non-null   object
 9   NCBI Project ID           2915 non-null   int64
 10  Genbank ID                1579 non-null   object
 11  Gene Count                2915 non-null   int64
 12  IMG/HMP ID                2915 non-null   int64
 13  HOMD ID                   397 non-null    object
 14  Sequencing Center         2911 non-null   object
 15  Funding Source            2915 non-null   object
 16  Strain Repository ID      1377 non-null   object
 17  Unnamed: 17               0 non-null      float64
 18  Unnamed: 18               0 non-null      float64
dtypes: float64(2), int64(4), object(13)
memory usage: 432.8+ KB
```

There is a broad range of information here, some of which may be beneficial to study regardless of project status, but for efficacy of this project, I'd like to check how many of the entires are complete.

In [6]:
```
microbes['Project Status'].value_counts()
```

Out[6]:
```
Project Status
Complete       1579
In Progress    1336
Name: count, dtype: int64
```

I'm going to remove any entries that are "in progress" from the main dataframe and place them in a new dataframe so I have it for running later if I want.

In [7]:
```
# Split Dataframe using groupby() &
# grouping by particular dataframe column
grouped = microbes.groupby(['Project Status'])
microbes_in_progress = grouped.get_group("In Progress")
microbes_in_progress.shape
```

Out[7]:
```
(1336, 19)
```

In [8]:
```
# Split Dataframe using groupby() &
# grouping by particular dataframe column
```

```
grouped = microbes.groupby(['Project Status'])
microbes_complete = grouped.get_group("Complete")
microbes_complete.shape
```

Out[8]:
```
(1579, 19)
```

In [9]:
```
#rename group of "complete" for ease
micro = microbes_complete
micro.shape
```

Out[9]:
```
(1579, 19)
```

In [10]:
```
micro.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1579 entries, 0 to 2914
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   HMP ID                  1579 non-null   int64  
 1   GOLD ID                 1493 non-null   object 
 2   Organism Name           1579 non-null   object 
 3   Domain                  1552 non-null   object 
 4   NCBI Superkingdom       1462 non-null   object 
 5   HMP Isolation Body Site  1579 non-null   object 
 6   Project Status          1579 non-null   object 
 7   Current Finishing Level  1579 non-null   object 
 8   NCBI Submission Status  1579 non-null   object 
 9   NCBI Project ID         1579 non-null   int64  
 10  Genbank ID              1579 non-null   object 
 11  Gene Count              1579 non-null   int64  
 12  IMG/HMP ID              1579 non-null   int64  
 13  HOMD ID                 386 non-null    object 
 14  Sequencing Center       1579 non-null   object 
 15  Funding Source          1579 non-null   object 
 16  Strain Repository ID    1272 non-null   object 
 17  Unnamed: 17             0 non-null      float64
 18  Unnamed: 18             0 non-null      float64
dtypes: float64(2), int64(4), object(13)
memory usage: 246.7+ KB
```

I'm curious about some of the columns that have null values. The ones that are important to organism analysis are "Domain", and "NCBI Superkingdom".

In [11]:
```
micro[['Domain','NCBI Superkingdom']].isnull().sum()
```

Out[11]:
```
Domain                 27
NCBI Superkingdom     117
dtype: int64
```

In [12]:
```
micro.groupby('Domain').count()
```

Out[12]:

Domain	HMP ID	GOLD ID	Organism Name	NCBI Superkingdom	HMP Isolation Body Site	Project Status	Current Finishing Level	Sub
ARCHAEAL	2	2	2	2	2	2	2	
BACTERIAL	1541	1487	1541	1440	1541	1541	1541	
EUKARYAL	4	4	4	4	4	4	4	
VIRUS	5	0	5	5	5	5	5	

In [13]:
```
micro.groupby('NCBI Superkingdom').count()
```

Out[13]:

NCBI Superkingdom	HMP ID	GOLD ID	Organism Name	Domain	HMP Isolation Body Site	Project Status	Current Finishing Level	N Submiss Sta
Archaea	2	2	2	2	2	2	2	
Bacteria	1448	1384	1448	1437	1448	1448	1448	14
Error!!!	3	3	3	3	3	3	3	
Eukaryota	4	4	4	4	4	4	4	
Viruses	5	0	5	5	5	5	5	

There is an "Error!!!" value for Superkingdom, so that's nice to be able to see exactly what I should replace. I'll start by checking those values specfically.

In [14]:
```
micro[micro['NCBI Superkingdom']=='Error!!!']
```

Out[14]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Project Status
2478	9176	Gi05045	Streptococcus downei F0415	BACTERIAL	Error!!!	oral	Complete
2481	9180	Gi05049	Streptococcus peroris ATCC 700780	BACTERIAL	Error!!!	oral	Complete
2487	9192	Gi05061	Streptococcus vestibularis F0396	BACTERIAL	Error!!!	oral	Complete

All three are in the Bacterial domain, so I can replace their values with "Bacteria"

In [15]:
```
micro['NCBI Superkingdom'].replace('Error!!!', 'Bacteria', inplace=True)
```

I can infer the Domain based on the Superkingdom and vice versa, but I can't use any values where both are missing, so I'll check those.

In [16]:
```
len(micro.loc[micro['Domain'].isnull()& micro['NCBI Superkingdom'].isnull()]
```

Out[16]:
```
16
```

There are 16 values that have both missing so I'm going to drop those.

In [17]:
```
micro=micro.drop(micro[(micro['Domain'].isnull()) & (micro['NCBI Superkingdc
micro.shape
```

Out[17]:
```
(1563, 19)
```

In order to replace the other values, I'm going to transform them to NaN first.

In [18]:
```
micro['NCBI Superkingdom'].fillna('NaN', inplace=True)
```

In [19]:
```
(micro['NCBI Superkingdom'] == "NaN").value_counts()
```

```
Out[19]: NCBI Superkingdom
         False    1462
         True      101
         Name: count, dtype: int64
```

```
In [20]: micro['Domain'].fillna('NaN', inplace=True)
```

```
In [21]: #check value counts
         (micro['Domain'] == "NaN").value_counts()
```

```
Out[21]: Domain
         False    1552
         True       11
         Name: count, dtype: int64
```

I'm going to replace all of the Domain values with their relative Superkingdom name where applicable using pandas transform function.

```
In [22]: #make dataframe containing only rows with NaN in Domain or Superkingdom
         micro_null = micro[(micro['Domain'] == "NaN") | (micro['NCBI Superkingdom']
```

```
In [23]: #See which rows have NaN to compare with their Superkingdom value
         micro_null.loc[micro_null['Domain'] == "NaN"]
```

Out[23]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Projec Statu
1314	1978	NaN	Actinomyces graevenitzii F0530	NaN	Bacteria	oral	Complete
1463	2128	NaN	Arthrobacter albus DNF00011	NaN	Bacteria	urogenital_tract	Complete
1464	2129	NaN	Corynebacterium tuscaniense DNF00037	NaN	Bacteria	urogenital_tract	Complete
1465	2130	NaN	Oligella urethralis DNF00040	NaN	Bacteria	urogenital_tract	Complete
1467	2132	NaN	Prevotella histicola JCM 15637 = DNF00424	NaN	Bacteria	urogenital_tract	Complete
1469	2134	NaN	Peptoniphilus lacrimalis DNF00528	NaN	Bacteria	urogenital_tract	Complete
1470	2135	NaN	Staphylococcus haemolyticus DNF00585	NaN	Bacteria	urogenital_tract	Complete
1471	2136	NaN	Prevotella bivia DNF00650	NaN	Bacteria	urogenital_tract	Complete

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Projec Status
1472	2137	NaN	Prevotella buccalis DNF00853	NaN	Bacteria	urogenital_tract	Complete
1474	2139	NaN	Prevotella denticola DNF00960	NaN	Bacteria	urogenital_tract	Complete
1475	2140	NaN	Prevotella buccalis DNF00985	NaN	Bacteria	urogenital_tract	Complete

In [24]:
```
#count nulls to compare
(micro['Domain'] == "NaN").sum()
```

Out[24]:
```
11
```

In [25]:
```
#Replace NaN values with "BACTERIAL"
micro["Domain"] = micro['Domain'].replace(["NaN"], "BACTERIAL")
micro
```

Out[25]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolati Body Si
0	1	Gi03551	Abiotrophia defectiva ATCC 49176	BACTERIAL	Bacteria	o
1	4	Gi03555	Achromobacter piechaudii ATCC 43553	BACTERIAL	Bacteria	airwa
2	5	Gi03554	Achromobacter xylosoxidans C54	BACTERIAL	Bacteria	airwa
3	10	Gi03422	Acinetobacter baumannii ATCC 19606	BACTERIAL	Bacteria	urogenital_tra
4	12	Gi03421	Acinetobacter calcoaceticus RUH2202	BACTERIAL	Bacteria	sk
...	...	...	...	...	...	
2910	9995	Gi08654	Staphylococcus epidermidis NIHLM095	BACTERIAL	Bacteria	unknow
2911	9996	Gi09593	Aggregatibacter actinomycetemcomitans Y4	BACTERIAL	Bacteria	o
2912	9997	Gi09594	Corynebacterium durum F0235	BACTERIAL	Bacteria	o

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolati Body Si
2913	9998	Gi09595	Peptostreptococcus anaerobius VPI 4330	BACTERIAL	Bacteria	o
2914	9999	Gi09596	Prevotella sp. oral taxon 473 str. F0040	BACTERIAL	Bacteria	o

1563 rows × 19 columns

In [26]:
```
#Check that values replaced
(micro['Domain'] == "NaN").sum()
```

Out[26]:
```
0
```

Then I'll replace all of the Superkingdom values with their Domain name where applicable using pandas transform function.

In [27]:
```
#Check values for Domain in regard to Superkingdom NaNs
kingdom = micro_null.loc[micro_null['NCBI Superkingdom'] == "NaN"]
kingdom['Domain'].value_counts()
```

Out[27]:
```
Domain
BACTERIAL    101
Name: count, dtype: int64
```

All of the missing Superkingdom values are in the Bacterial domain, so we can replace them with the relative value of "Bacteria".

In [28]:
```
#count nulls to compare
(micro_null['NCBI Superkingdom'] == "NaN").sum()
```

Out[28]:
```
101
```

In [29]:
```
#Replace NaN values with "BACTERIAL"
micro["NCBI Superkingdom"] = micro['NCBI Superkingdom'].replace(['NaN'], 'Ba
micro
```

Out[29]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolati Body Si
0	1	Gi03551	Abiotrophia defectiva ATCC 49176	BACTERIAL	Bacteria	or
1	4	Gi03555	Achromobacter piechaudii ATCC 43553	BACTERIAL	Bacteria	airwa
2	5	Gi03554	Achromobacter xylosoxidans C54	BACTERIAL	Bacteria	airwa
3	10	Gi03422	Acinetobacter baumannii ATCC 19606	BACTERIAL	Bacteria	urogenital_tra
4	12	Gi03421	Acinetobacter calcoaceticus RUH2202	BACTERIAL	Bacteria	sk
...	...	...	...	...	...	
2910	9995	Gi08654	Staphylococcus epidermidis NIHLM095	BACTERIAL	Bacteria	unknov
2911	9996	Gi09593	Aggregatibacter actinomycetemcomitans Y4	BACTERIAL	Bacteria	or
2912	9997	Gi09594	Corynebacterium durum F0235	BACTERIAL	Bacteria	or

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolati Body Si
2913	9998	Gi09595	Peptostreptococcus anaerobius VPI 4330	BACTERIAL	Bacteria	o
2914	9999	Gi09596	Prevotella sp. oral taxon 473 str. F0040	BACTERIAL	Bacteria	o

1563 rows × 19 columns

In [30]:
```
#Check that values replaced
(micro['NCBI Superkingdom'] == "NaN").sum()
```

Out[30]:
```
0
```

Exploration

Now that those are cleaned up, I'm going to review the full dataset based on Gene Count to start.

In [31]:
```
micro['Gene Count'].describe()
```

Out[31]:
```
count    1563.000000
mean     2729.550864
std      1288.903478
min         0.000000
25%      1956.000000
50%      2411.000000
75%      3176.000000
max      8490.000000
Name: Gene Count, dtype: float64
```

There are no null values for "Gene Count", but some are counted as 0.

In [32]:
```
micro_gene_count=micro[micro['Gene Count']==0]
micro_gene_count['NCBI Superkingdom'].value_counts()
```

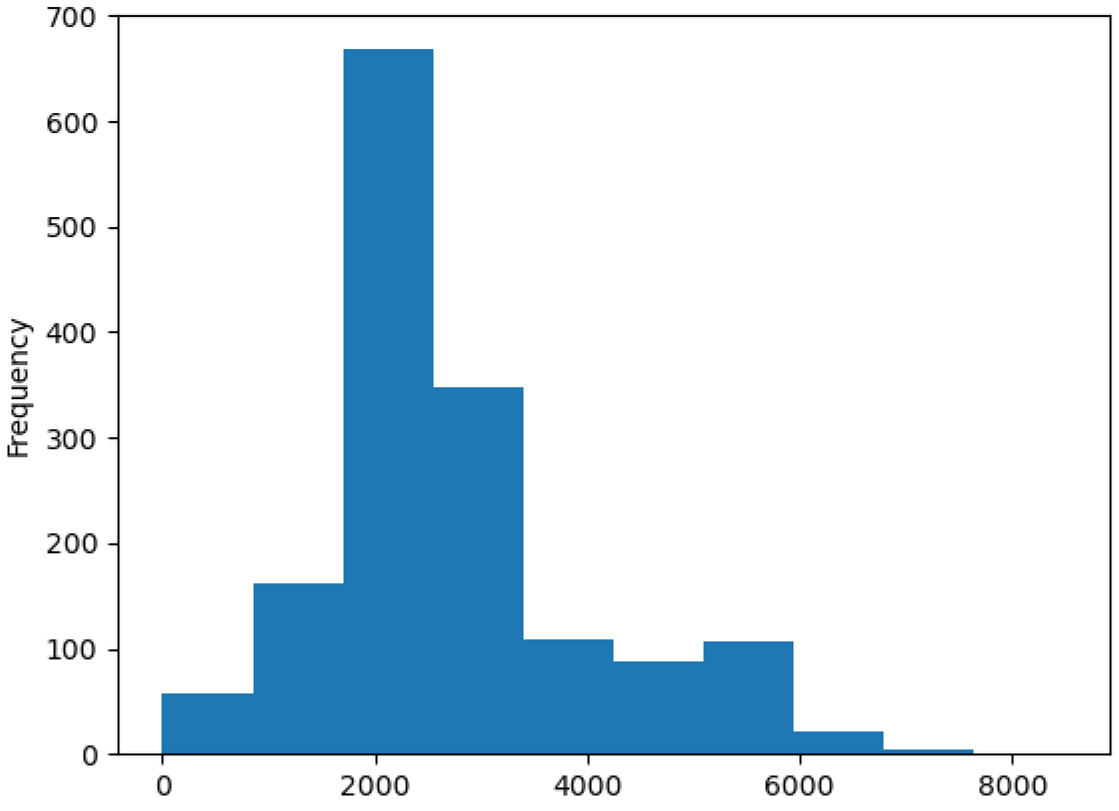
Out[32]:
```
NCBI Superkingdom
Bacteria     47
Viruses       5
Eukaryota     4
Name: count, dtype: int64
```

There are 47 bacteria, 5 viruses, and 4 eukaryota absent from the count. Because these may be based on a reporting error, I may want to drop these later to improve the model, but I'll keep them for now.

There are many species listed in this project, so I want to look at their distribution of gene count frequency.

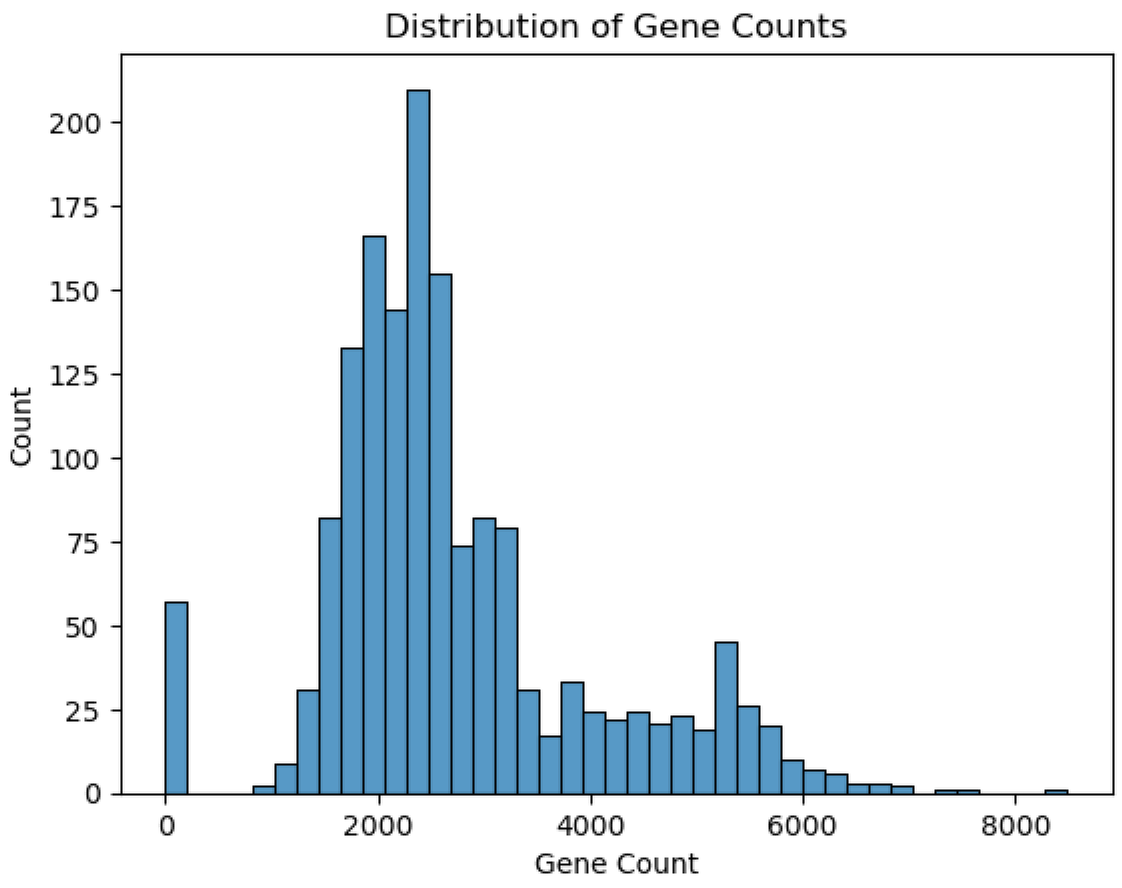
In [33]:
```
micro["Gene Count"].plot(kind='hist')
```

Out[33]:
```
<Axes: ylabel='Frequency'>
```



In [152…
```
sns.histplot(data=micro, x="Gene Count")
plt.title('Distribution of Gene Counts')
```

Out[152…
```
Text(0.5, 1.0, 'Distribution of Gene Counts')
```



Interestingly, there is an almost normal distribution, skewed right, but we can see that the species with gene counts in the middle range have the highest freqency.

I'm curious about the microbe with the highest gene count (max value from the descriptive statistics), with a value of 8490.

```
In [35]: micro[micro['Gene Count']==8490]
```

Out[35]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	P S
679	1211	Gi10716	Streptomyces sp. HGB0020	BACTERIAL	Bacteria	gastrointestinal_tract	Co

I want to check to see if there is another Streptomyces species with high prevalence.

```
In [36]: micro[micro['Organism Name'].str.contains("Streptomyces")]
```

Out[36]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	I
679	1211	Gi10716	Streptomyces sp. HGB0020	BACTERIAL	Bacteria	gastrointestinal_tract	Co
934	1486	Gi16997	Streptomyces sp. HPH0547	BACTERIAL	Bacteria	gastrointestinal_tract	Co

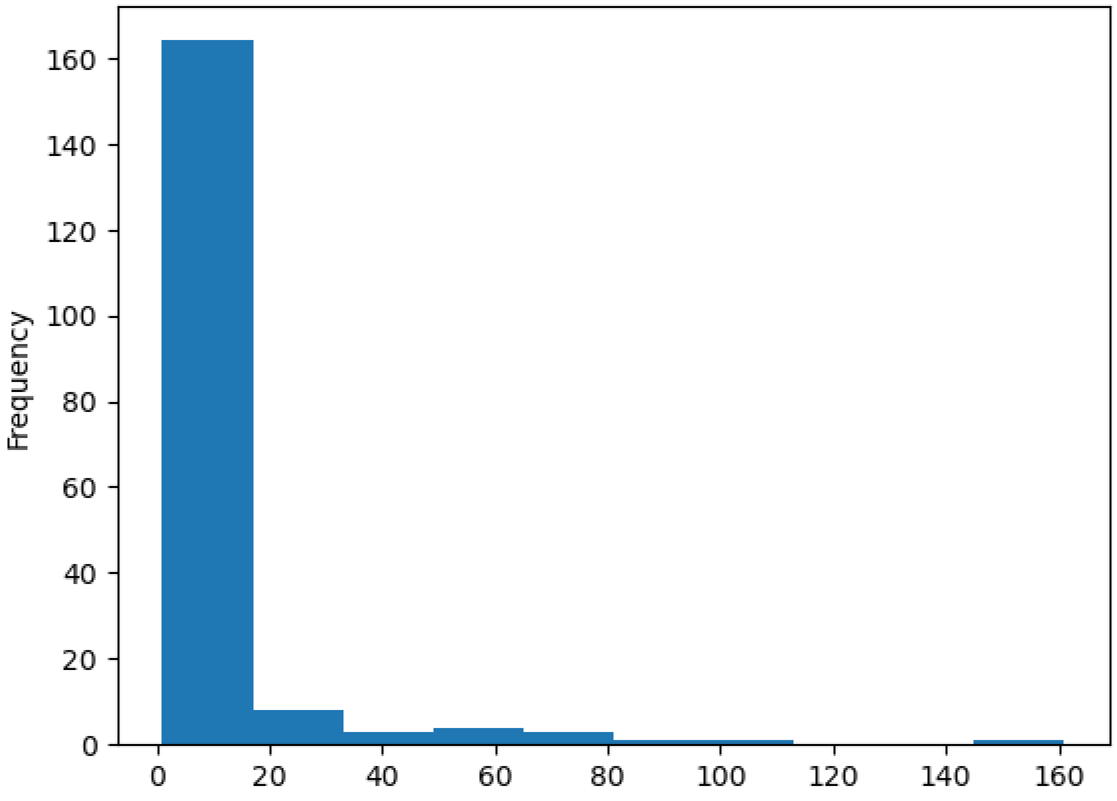
The presences of another Streoptomyces with a high gene count makes me think that it may be beneficial to sort organisms on their genus. I'm going make a new dataframe and attempt to add a column for genus by extracting the first word of the Organism Name.

In [37]:
```
df = micro
df['Genus'] = df['Organism Name'].str.split(' ').str[0]
df['Genus'].nunique()
```

Out[37]:
```
185
```

In [38]:
```
df["Genus"].value_counts().plot(kind='hist')
```

Out[38]:
```
<Axes: ylabel='Frequency'>
```



In [39]:
```
#Seaborn plot - not as beneficial this time so I'm not using it
#plot frequency of genus for all entires
#x = df["Genus"].value_counts()
#sns.histplot(data=df, x=x)
```

Since there are so many distributed around 0-15, I'm going to exclude those and print the value counts of the higher ones.

In [40]:
```
count = df[df.Genus.isin(df["Genus"].value_counts(dropna=False).loc[lambda x
count["Genus"].value_counts()
```

```
Out[40]: Genus
         Streptococcus       161
         Enterococcus        110
         Propionibacterium    92
         Lactobacillus        73
         Helicobacter         70
         Prevotella           65
         Staphylococcus       64
         Bacteroides          63
         Escherichia          61
         Clostridium          58
         Corynebacterium      38
         Fusobacterium        36
         Actinomyces          34
         Bifidobacterium      31
         Treponema            25
         Gardnerella          22
         Klebsiella           21
         Eubacterium          21
         Neisseria            19
         Porphyromonas        17
         Capnocytophaga       17
         Veillonella          16
         Name: count, dtype: int64
```

Since the list is limited, I probably could have just guessed the index number until I got to the value I wanted.

```
In [41]: df['Genus'].value_counts(ascending=False)[:22]
```

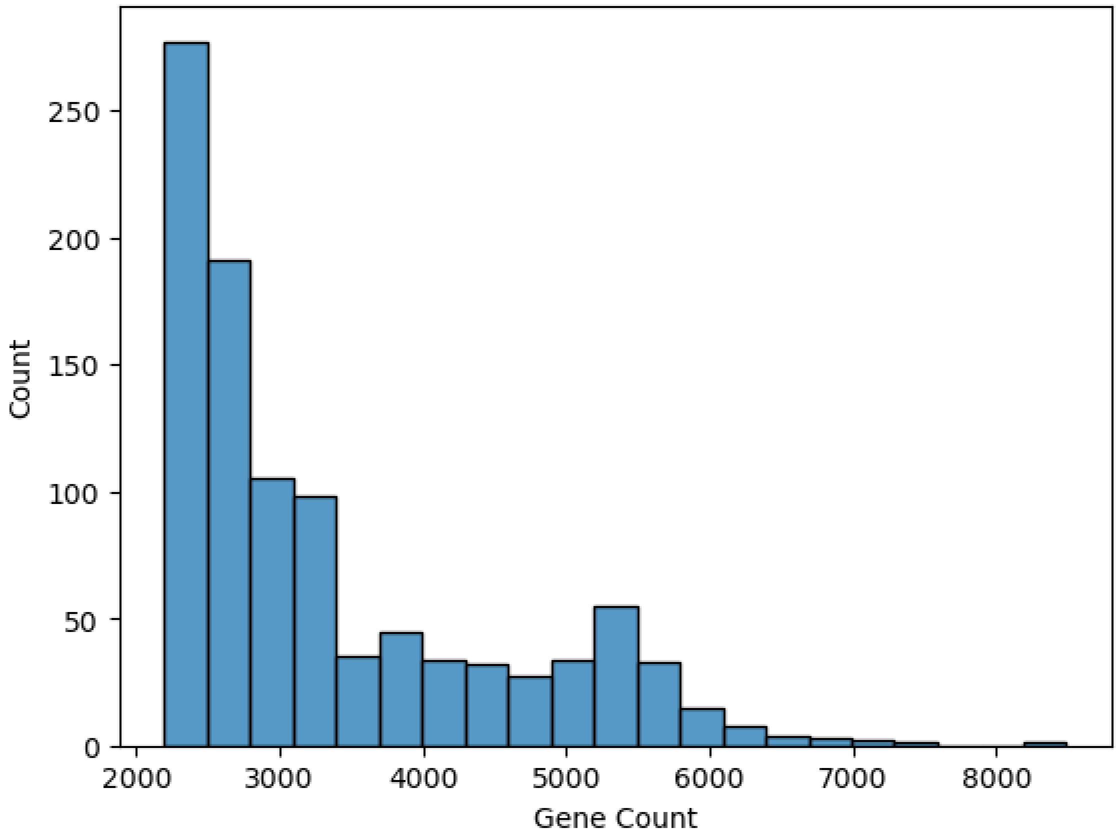
```
Out[41]: Genus
         Streptococcus       161
         Enterococcus        110
         Propionibacterium    92
         Lactobacillus        73
         Helicobacter         70
         Prevotella           65
         Staphylococcus       64
         Bacteroides          63
         Escherichia          61
         Clostridium          58
         Corynebacterium      38
         Fusobacterium        36
         Actinomyces          34
         Bifidobacterium      31
         Treponema            25
         Gardnerella          22
         Klebsiella           21
         Eubacterium          21
         Neisseria            19
         Capnocytophaga       17
         Porphyromonas        17
         Veillonella          16
         Name: count, dtype: int64
```

Let's see if this changes based on a subset of the most prevalent organisms.

```
In [42]: top_organisms=micro.sort_values(by='Gene Count', ascending = False)[:1000]
```

```
In [43]: sns.histplot(data=top_organisms, x="Gene Count")
```

```
Out[43]: <Axes: xlabel='Gene Count', ylabel='Count'>
```



```
In [44]: # add genus column to the top_organisms
         top = top_organisms
         top['Genus'] = top['Organism Name'].str.split(' ').str[0]
```

```
In [45]: top['Genus'].value_counts()[:22]
```

Out[45]:
```
Genus
Enterococcus         109
Propionibacterium     91
Staphylococcus        62
Bacteroides           62
Escherichia           60
Clostridium           57
Streptococcus         53
Prevotella            51
Corynebacterium       32
Treponema             24
Lactobacillus         22
Klebsiella            21
Fusobacterium         20
Actinomyces           17
Neisseria             17
Capnocytophaga        16
Parabacteroides       15
Acinetobacter         14
Bifidobacterium       12
Providencia           11
Eubacterium            9
Selenomonas            8
Name: count, dtype: int64
```

In the original histogram, we saw that the highest frequency of species was between 1800-2400 gene count, so I am going to make a dataframe around that.

In [46]:
```python
mid_microbe = micro[(micro['Gene Count'].values >= 1800) & (micro['Gene Cour
```

In [47]:
```python
mid_microbe['Genus'] = mid_microbe['Organism Name'].str.split(' ').str[0]
```

In [48]:
```python
mid_microbe["Genus"].value_counts().sort_values(ascending=False)[:22]
```

Out[48]:
```
Genus
Streptococcus           142
Lactobacillus            30
Staphylococcus           27
Prevotella               25
Corynebacterium          23
Bifidobacterium          21
Propionibacterium        20
Fusobacterium            19
Actinomyces              14
Veillonella              13
Porphyromonas            10
Selenomonas              10
Haemophilus              10
Mobiluncus                8
Capnocytophaga            5
Anaerococcus              5
Neisseria                 4
Peptostreptococcaceae     4
Helicobacter              4
Oribacterium              4
Leptotrichia              4
Peptoniphilus             4
Name: count, dtype: int64
```

Now that I'm comfortable with having added "Genus" to my dataframes, I'm going to replace the main dataframe with the amended one.

In [49]:
```
df = micro
```

I want to know how many unique sites on the human body were researched.

In [50]:
```
micro['HMP Isolation Body Site'].nunique()
```

Out[50]:
```
12
```

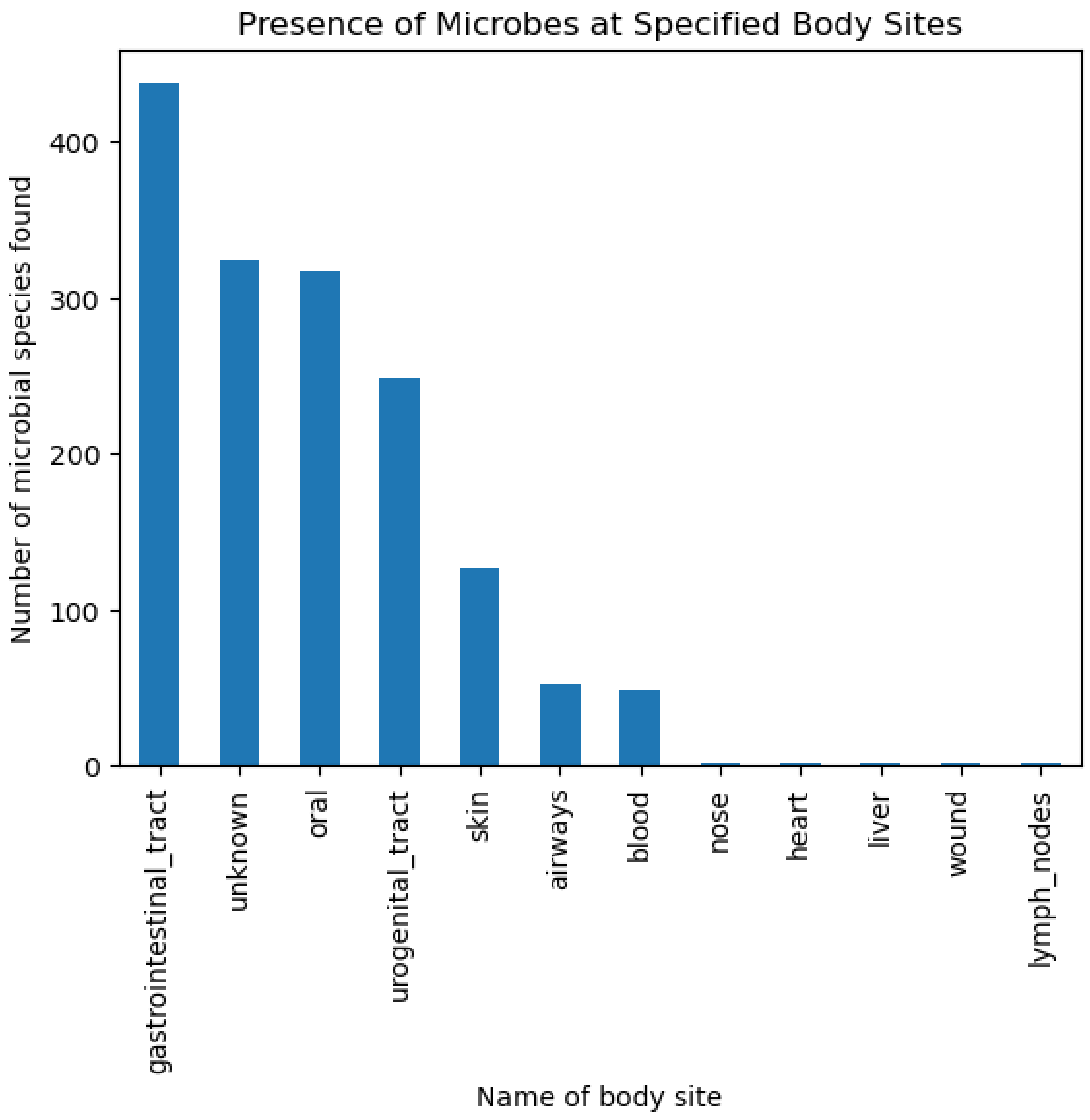
In [51]:
```
micro['HMP Isolation Body Site'].value_counts()
```

Out[51]:
```
HMP Isolation Body Site
gastrointestinal_tract    437
unknown                   324
oral                      317
urogenital_tract          249
skin                      127
airways                    53
blood                      49
nose                        2
heart                       2
liver                       1
wound                       1
lymph_nodes                 1
Name: count, dtype: int64
```

Here is a chart of the species diversity at the different sites.

```
In [52]: micro['HMP Isolation Body Site'].value_counts().plot(kind='bar')
         plt.title('Presence of Microbes at Specified Body Sites')
         plt.ylabel('Number of microbial species found')
         plt.xlabel('Name of body site')
```

```
Out[52]: Text(0.5, 0, 'Name of body site')
```



To find out more about the kingdom variance throughout the body, I'll look into those values.

```
In [53]: micro.groupby('NCBI Superkingdom')['HMP Isolation Body Site'].nunique().sort
```

```
Out[53]: NCBI Superkingdom
         Bacteria     11
         Eukaryota     3
         Archaea       1
         Viruses       1
         Name: HMP Isolation Body Site, dtype: int64
```

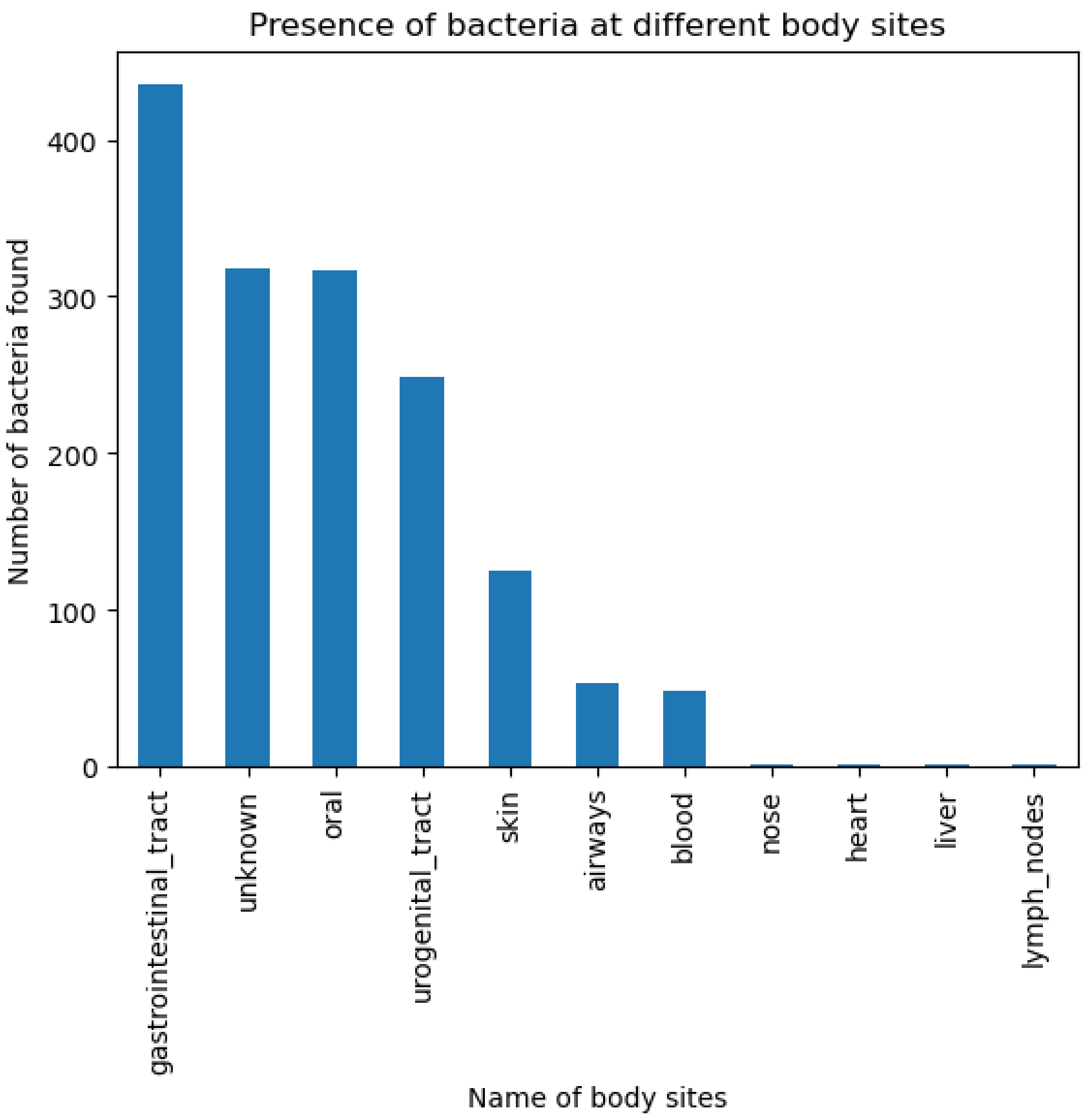
The Bacteria Kingdom is most prevelent throughout the body, so I'm going to look more closely that their locations by making a dataframe with the values from the "bacterial" domain.

```
In [54]: #Select Bacterial domain and check body sites
         bac=micro.loc[micro['Domain']=='BACTERIAL']
         bac['HMP Isolation Body Site'].unique()
```

```
Out[54]: array(['oral', 'airways', 'urogenital_tract', 'skin',
                'gastrointestinal_tract', 'blood', 'unknown', 'liver', 'nose',
                'heart', 'lymph_nodes'], dtype=object)
```

```
In [55]: #Plot
         bac['HMP Isolation Body Site'].value_counts(ascending=False).plot(kind='bar'
         plt.ylabel('Number of bacteria found')
         plt.xlabel('Name of body sites')
         plt.title('Presence of bacteria at different body sites')
```

```
Out[55]: Text(0.5, 1.0, 'Presence of bacteria at different body sites')
```



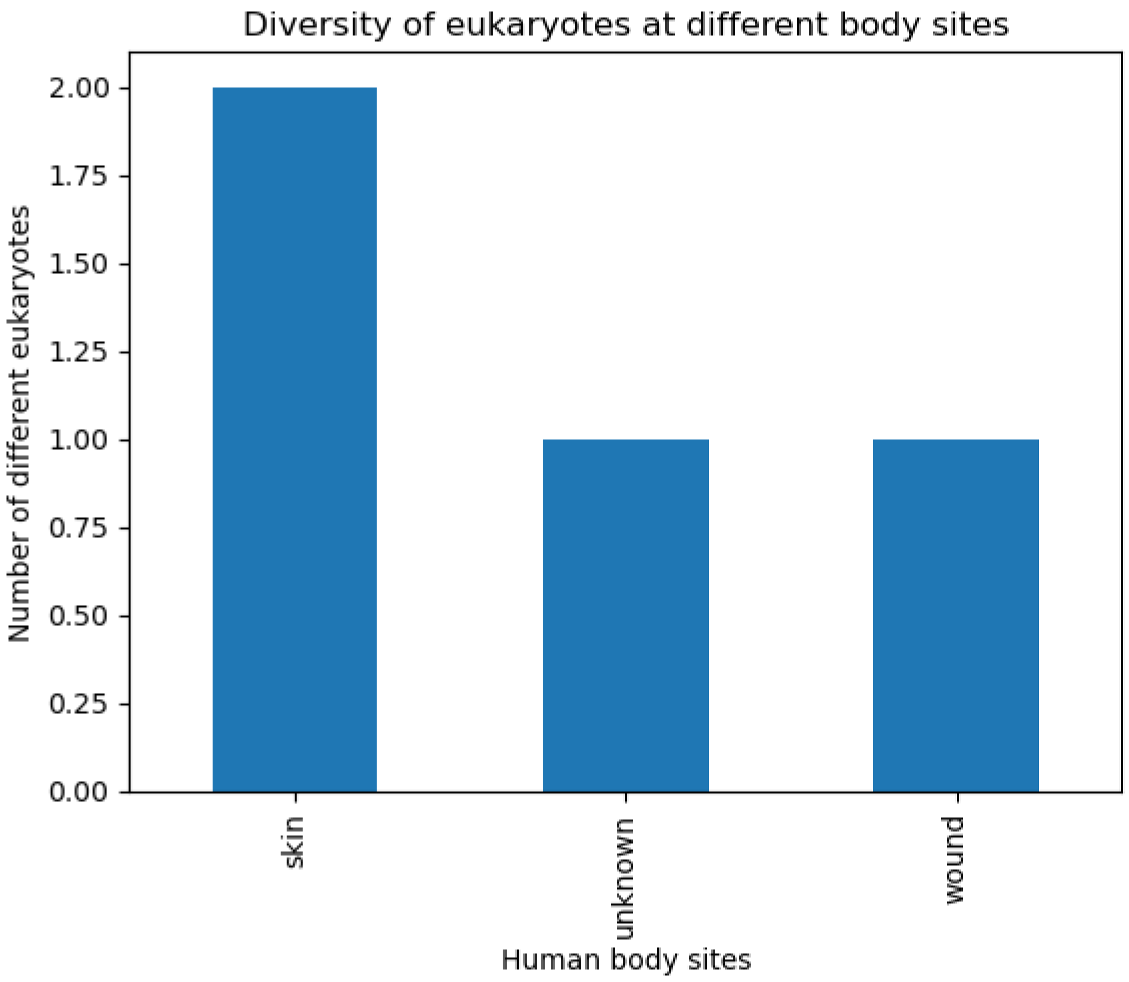
The greatest amount of bacteria are found in the gastrointestinal tract. Let's look at the other kingdoms.

In [56]:
```
#Select Eukaryal domain and check body sites
euk=micro.loc[micro['Domain']=='EUKARYAL']
euk['HMP Isolation Body Site'].unique()
```

Out[56]:
```
array(['unknown', 'skin', 'wound'], dtype=object)
```

In [57]:
```
#plot
euk['HMP Isolation Body Site'].value_counts(ascending=False).plot(kind='bar'
plt.ylabel('Number of different eukaryotes')
plt.xlabel('Human body sites')
plt.title('Diversity of eukaryotes at different body sites')
```

Out[57]:
```
Text(0.5, 1.0, 'Diversity of eukaryotes at different body sites')
```



The greatest amount of eukaryotes are found on the skin.

In [58]:
```
vir=micro.loc[micro['Domain']=='VIRUS']
vir['HMP Isolation Body Site'].unique()
```

Out[58]:
```
array(['unknown'], dtype=object)
```

From the data that we have, we are unable to determine where the greatest number of viruses are located. I'm guessing this is because viruses infect and replicate in cells, sometimes mainly infecting neighboring cells, but often spreading throughout the body.

In [59]:
```
arc=micro.loc[micro['Domain']=='ARCHAEAL']
arc['HMP Isolation Body Site'].unique()
```

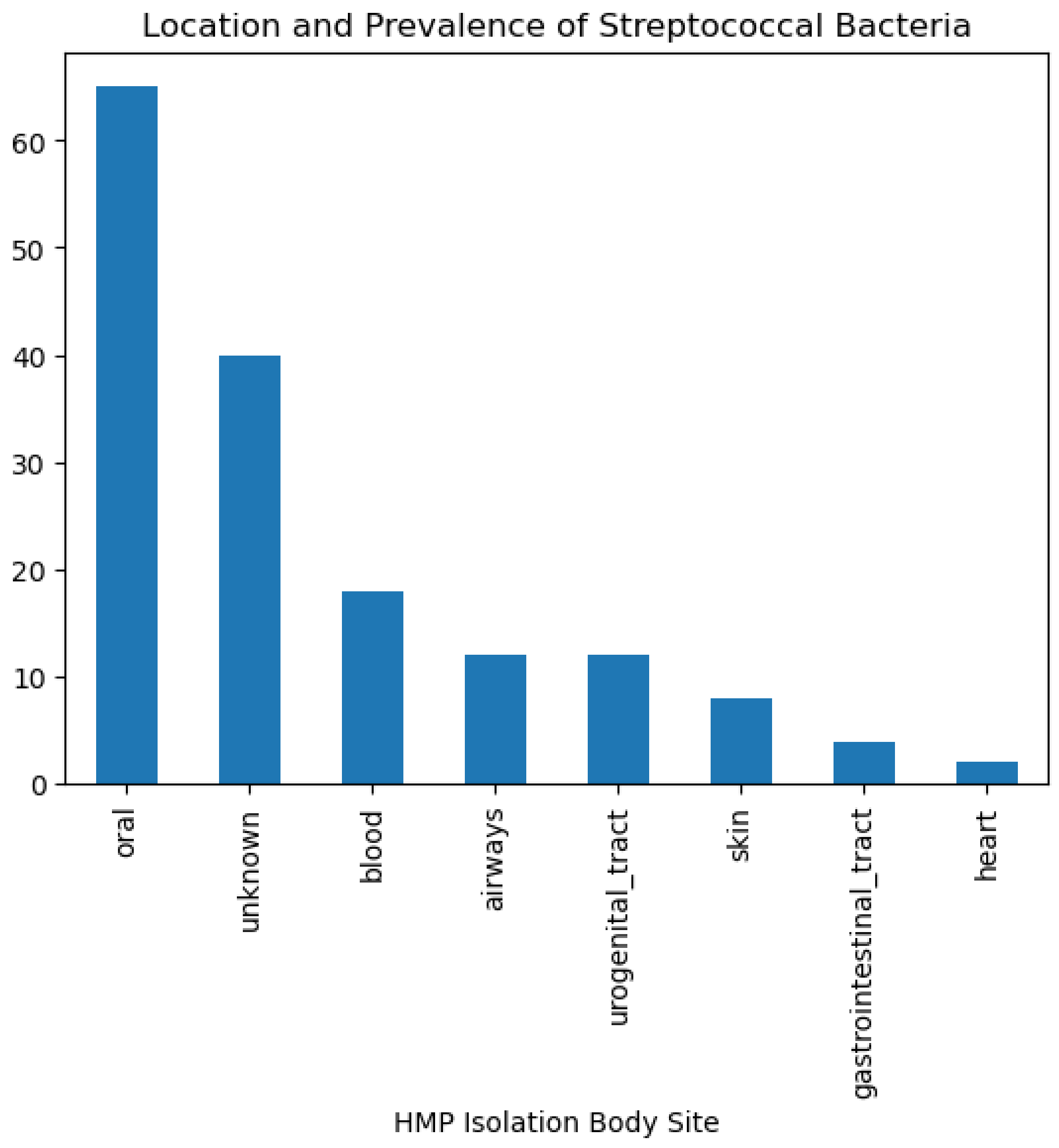
Out[59]:
```
array(['gastrointestinal_tract'], dtype=object)
```

Based on this project, archaea are found solely in the Gastrointestinal Tract.

Streptococcus are the most prevalent throughout the body. Let's look further into this.

In [60]:
```
strep=micro.loc[micro['Genus']=='Streptococcus']
strep['HMP Isolation Body Site'].value_counts().plot(kind='bar')
plt.title("Location and Prevalence of Streptococcal Bacteria")
```

Out[60]:
```
Text(0.5, 1.0, 'Location and Prevalence of Streptococcal Bacteria')
```

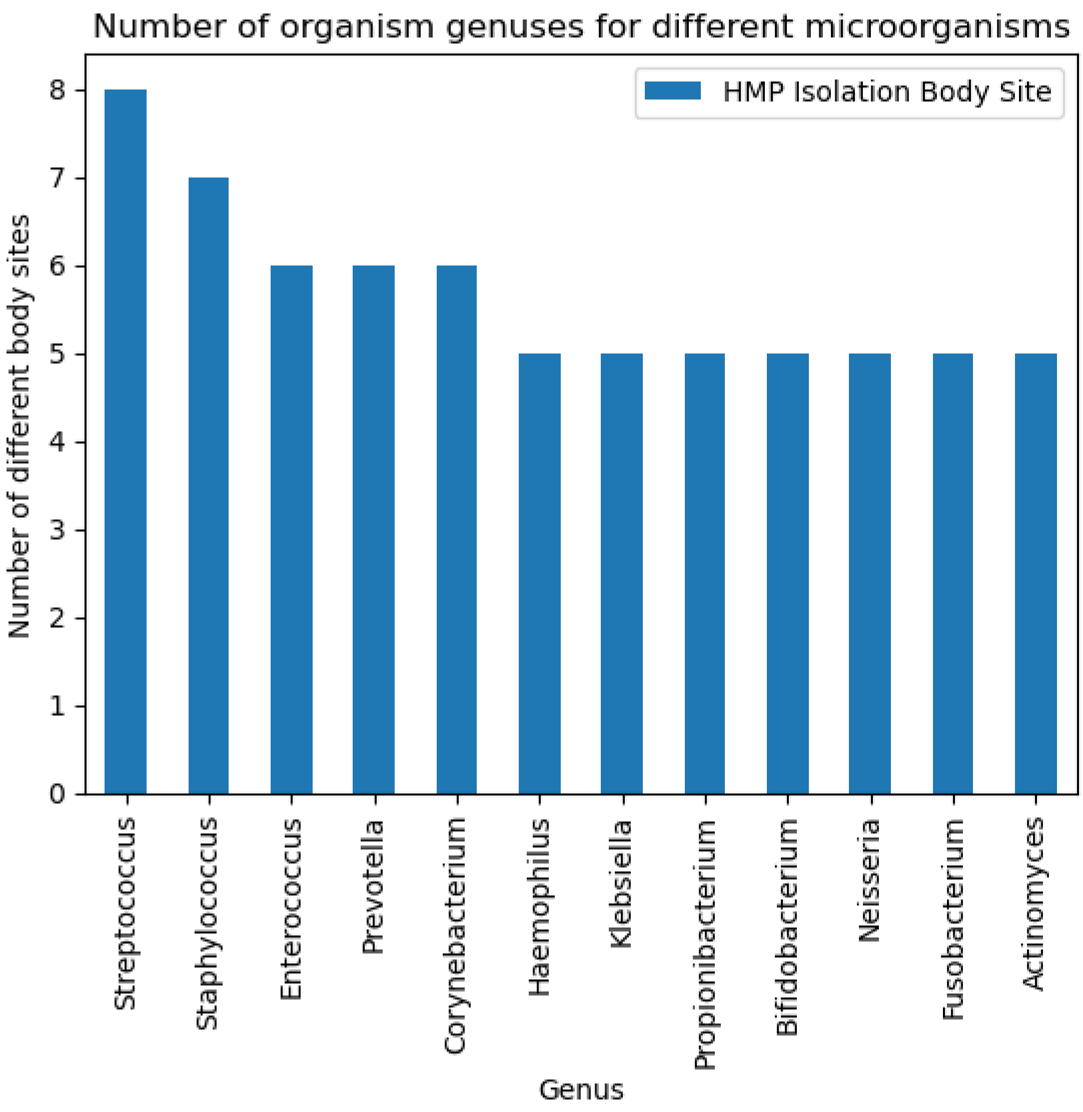


Let's see how other top genuses compare.

In [61]:
```
genus_site=micro.groupby('Genus')['HMP Isolation Body Site'].nunique().sort_
genus_df=pd.DataFrame(genus_site)
top_genus=genus_df[genus_df['HMP Isolation Body Site']>4]
print(top_genus)
top_genus.plot(kind='bar')
plt.ylabel('Number of different body sites')
plt.title('Number of organism genuses for different microorganisms')
```

```
                   HMP Isolation Body Site
Genus
Streptococcus                            8
Staphylococcus                           7
Enterococcus                             6
Prevotella                               6
Corynebacterium                          6
Haemophilus                              5
Klebsiella                               5
Propionibacterium                        5
Bifidobacterium                          5
Neisseria                                5
Fusobacterium                            5
Actinomyces                              5
```

Out[61]:
```
Text(0.5, 1.0, 'Number of organism genuses for different microorganisms')
```



Because viruses are limited, we can print all of their names.

In [62]:
```
viruses= micro[micro['NCBI Superkingdom'] =='Viruses']
viruses['Organism Name']
```

Out[62]:
```
2852    Pseudomonas phage F_HA0480sp/Pa1651
2853                 Pseudomonas phage JBD18
2854                 Pseudomonas phage JBD25
2855                 Pseudomonas phage JBD26
2856                 Pseudomonas phage JBD67
Name: Organism Name, dtype: object
```

Because eukaryotes are limited, we can print all of their names.

In [63]:
```
eukaryotes= micro[micro['NCBI Superkingdom']=='Eukaryota']
eukaryotes['Organism Name']
```

Out[63]:
```
601                  Exophiala dermatitidis NIH/UT8656
983                    Phialophora europaea CBS 101466
985     Mucor circinelloides f. circinelloides 1006PhL
1065                   Sporothrix schenckii ATCC 58251
Name: Organism Name, dtype: object
```

Because archaea are limited, we can print all of their names.

In [64]:
```
archaea= micro[micro['NCBI Superkingdom']=='Archaea']
archaea['Organism Name']
```

Out[64]:
```
302    Methanobrevibacter smithii DSM 2374
303    Methanobrevibacter smithii DSM 2375
Name: Organism Name, dtype: object
```

In [65]:
```
micro['NCBI Superkingdom'].value_counts()
```

Out[65]:
```
NCBI Superkingdom
Bacteria     1552
Viruses         5
Eukaryota       4
Archaea         2
Name: count, dtype: int64
```

Observations from this EDA:

- Gastrointestinal system shows most diversity of microbes
- Streptomyces sp. HGB0020 shows the maximum gene count in human
- Streptococcus is most common genus

In [66]:
```
# change df name for transformations
df = micro
```

In [67]:
```
# df.head() #commented out for brevity
```

Out[67]:

	HMP ID	GOLD ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Projec Statu
0	1	Gi03551	Abiotrophia defectiva ATCC 49176	BACTERIAL	Bacteria	oral	Complet
1	4	Gi03555	Achromobacter piechaudii ATCC 43553	BACTERIAL	Bacteria	airways	Complet
2	5	Gi03554	Achromobacter xylosoxidans C54	BACTERIAL	Bacteria	airways	Complet
3	10	Gi03422	Acinetobacter baumannii ATCC 19606	BACTERIAL	Bacteria	urogenital_tract	Complet
4	12	Gi03421	Acinetobacter calcoaceticus RUH2202	BACTERIAL	Bacteria	skin	Complet

In [68]:
```
#find na values
df.isna().sum()
```

Out[68]:
```
HMP ID                       0
GOLD ID                     70
Organism Name                0
Domain                       0
NCBI Superkingdom            0
HMP Isolation Body Site      0
Project Status               0
Current Finishing Level      0
NCBI Submission Status       0
NCBI Project ID              0
Genbank ID                   0
Gene Count                   0
IMG/HMP ID                   0
HOMD ID                   1177
Sequencing Center            0
Funding Source               0
Strain Repository ID       296
Unnamed: 17               1563
Unnamed: 18               1563
Genus                        0
dtype: int64
```

In [69]:
```
#check percentage missing values
round((df.isnull().sum() * 100/ len(df)),2).sort_values(ascending=False)
```

Out[69]:
```
Unnamed: 18                100.00
Unnamed: 17                100.00
HOMD ID                     75.30
Strain Repository ID        18.94
GOLD ID                      4.48
HMP ID                       0.00
Funding Source               0.00
Sequencing Center            0.00
IMG/HMP ID                   0.00
Gene Count                   0.00
Genbank ID                   0.00
NCBI Project ID              0.00
NCBI Submission Status       0.00
Current Finishing Level      0.00
Project Status               0.00
HMP Isolation Body Site      0.00
NCBI Superkingdom            0.00
Domain                       0.00
Organism Name                0.00
Genus                        0.00
dtype: float64
```

In [70]:
```
# HOMD ID has 75% missing values so I'm checking it
df['HOMD ID']
```

Out[70]:
```
0       HOMD: tax_389
1                 NaN
2       HOMD: tax_343
3       HOMD: tax_554
4                 NaN
            ...
2910              NaN
2911              NaN
2912              NaN
2913              NaN
2914              NaN
Name: HOMD ID, Length: 1563, dtype: object
```

The unnamed columns provide no information so they can be removed. HOMD ID isn't necessary for analysis so it can be removed as well.

In [71]:
```
df2 = df.drop(['Unnamed: 18', 'Unnamed: 17', 'HOMD ID'], axis=1)
```

In [72]:
```
#check percentage missing values
round((df2.isnull().sum() * 100/ len(df2)),2).sort_values(ascending=False)
```

Out[72]:
```
Strain Repository ID       18.94
GOLD ID                     4.48
HMP ID                      0.00
NCBI Project ID             0.00
Funding Source              0.00
Sequencing Center           0.00
IMG/HMP ID                  0.00
Gene Count                  0.00
Genbank ID                  0.00
NCBI Submission Status      0.00
Current Finishing Level     0.00
Project Status              0.00
HMP Isolation Body Site     0.00
NCBI Superkingdom           0.00
Domain                      0.00
Organism Name               0.00
Genus                       0.00
dtype: float64
```

In [73]:
```
df['Strain Repository ID']
```

Out[73]:
```
0                 ATCC 49176, CIP 103242
1       ATCC 43553, CIP 55774, LMG 6100
2                             BEI HM-235
3                   ATCC 19606, DSM 6974
4                              LMG 10517
                     ...
2910                          BEI HM-909
2911                          ATCC 43718
2912                          BEI HM-755
2913                          ATCC 27337
2914                          BEI HM-756
Name: Strain Repository ID, Length: 1563, dtype: object
```

In [74]:
```
df["GOLD ID"]
```

Out[74]:
```
0       Gi03551
1       Gi03555
2       Gi03554
3       Gi03422
4       Gi03421
         ...
2910    Gi08654
2911    Gi09593
2912    Gi09594
2913    Gi09595
2914    Gi09596
Name: GOLD ID, Length: 1563, dtype: object
```

In [75]:
```
#HOMD ID and Strain Repository ID are not necessary for analysis so they car
df2 = df2.drop(['Strain Repository ID', 'GOLD ID'], axis=1)
```

In [76]:
```
df2.columns.tolist()
```

Out[76]:
```
['HMP ID',
 'Organism Name',
 'Domain',
 'NCBI Superkingdom',
 'HMP Isolation Body Site',
 'Project Status',
 'Current Finishing Level',
 'NCBI Submission Status',
 'NCBI Project ID',
 'Genbank ID',
 'Gene Count',
 'IMG/HMP ID',
 'Sequencing Center',
 'Funding Source',
 'Genus']
```

In [77]:
```
#removing the rest of the ID columns
df2 = df2.drop(['NCBI Project ID',
 'Genbank ID', 'IMG/HMP ID'], axis=1)
```

In [78]:
```
#new df name to preserve previous
df = df2
```

In [79]:
```
df.columns.tolist()
```

Out[79]:
```
['HMP ID',
 'Organism Name',
 'Domain',
 'NCBI Superkingdom',
 'HMP Isolation Body Site',
 'Project Status',
 'Current Finishing Level',
 'NCBI Submission Status',
 'Gene Count',
 'Sequencing Center',
 'Funding Source',
 'Genus']
```

In [80]:
```
# subset of df
test_df = df[['HMP ID',
 'Organism Name',
 'Domain',
 'NCBI Superkingdom',
 'HMP Isolation Body Site',
 'Gene Count',
 'Genus']]
```

In [81]:
```
test_df.shape
```

Out[81]:
```
(1563, 7)
```

In [82]:
```
#checking size and unique values of columns
df['Genus'].value_counts()
```

Out[82]:
```
Genus
Streptococcus        161
Enterococcus         110
Propionibacterium     92
Lactobacillus         73
Helicobacter          70
                    ...
Pediococcus            1
Mycobacterium          1
Micrococcus            1
Leuconostoc            1
Acetobacteraceae       1
Name: count, Length: 185, dtype: int64
```

In [83]:
```
# rename for to preserve previous
new_df = test_df
```

In [84]:
```
new_df
```

Out[84]:

	HMP ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Gene Count
0	1	Abiotrophia defectiva ATCC 49176	BACTERIAL	Bacteria	oral	1950
1	4	Achromobacter piechaudii ATCC 43553	BACTERIAL	Bacteria	airways	5755
2	5	Achromobacter xylosoxidans C54	BACTERIAL	Bacteria	airways	6010
3	10	Acinetobacter baumannii ATCC 19606	BACTERIAL	Bacteria	urogenital_tract	3832
4	12	Acinetobacter calcoaceticus RUH2202	BACTERIAL	Bacteria	skin	3632
...	...	...	...	...	...	...
2910	9995	Staphylococcus epidermidis NIHLM095	BACTERIAL	Bacteria	unknown	2300
2911	9996	Aggregatibacter actinomycetemcomitans Y4	BACTERIAL	Bacteria	oral	2343
2912	9997	Corynebacterium durum F0235	BACTERIAL	Bacteria	oral	2823
2913	9998	Peptostreptococcus anaerobius VPI 4330	BACTERIAL	Bacteria	oral	1933
2914	9999	Prevotella sp. oral taxon 473 str. F0040	BACTERIAL	Bacteria	oral	2317

1563 rows × 7 columns

I want to remove Domain or Superkingdom because they seem to have the same information, but first I need to check that. I'm going to convert the strings to lowercase so I may make a clean comparison. Then I'll search for the root words in the other column and see if any are not the same (check if they are all duplicated).

In [85]:

```
# Convert to lowercase for sorting
new_df['Domain'] = new_df['Domain'].str.lower()
new_df['Domain']
```

Out[85]:
```
0       bacterial
1       bacterial
2       bacterial
3       bacterial
4       bacterial
          ...
2910    bacterial
2911    bacterial
2912    bacterial
2913    bacterial
2914    bacterial
Name: Domain, Length: 1563, dtype: object
```

In [86]:
```
# Convert to lowercase for sorting

new_df['NCBI Superkingdom'] = new_df['NCBI Superkingdom'].str.lower()
new_df['NCBI Superkingdom']
```

Out[86]:
```
0       bacteria
1       bacteria
2       bacteria
3       bacteria
4       bacteria
          ...
2910    bacteria
2911    bacteria
2912    bacteria
2913    bacteria
2914    bacteria
Name: NCBI Superkingdom, Length: 1563, dtype: object
```

In [87]:
```
# use apply to find if the "superkingdom" string is in "domain", if it is no
new_df['New'] = new_df.apply(lambda x: x['NCBI Superkingdom'] if x['NCBI Sup
                     x['Domain'] else np.nan, axis=1)
```

In [88]:
```
#check NA value total
new_df['New'].isna().sum()
```

Out[88]:
```
9
```

In [89]:
```
#There are 9 null values so we can see review the entries manually
new_df[new_df['New'].isna()]
```

Out[89]:

	HMP ID	Organism Name	Domain	NCBI Superkingdom	HMP Isolation Body Site	Gene Count	Ger
601	1120	Exophiala dermatitidis NIH/UT8656	eukaryal	eukaryota	unknown	0	Exophi
983	1541	Phialophora europaea CBS 101466	eukaryal	eukaryota	skin	0	Phialoph
985	1544	Mucor circinelloides f. circinelloides 1006PhL	eukaryal	eukaryota	skin	0	Mu
1065	1624	Sporothrix schenckii ATCC 58251	eukaryal	eukaryota	wound	0	Sporoth
2852	9774	Pseudomonas phage F_HA0480sp/Pa1651	virus	viruses	unknown	0	Pseudomor
2853	9843	Pseudomonas phage JBD18	virus	viruses	unknown	0	Pseudomor
2854	9847	Pseudomonas phage JBD25	virus	viruses	unknown	0	Pseudomor
2855	9848	Pseudomonas phage JBD26	virus	viruses	unknown	0	Pseudomor
2856	9886	Pseudomonas phage JBD67	virus	viruses	unknown	0	Pseudomor

I see that these are duplicates as well, so all of the values between "NCBI Superkingdom" and "Domain" are the same. I can remove one of them. I am choosing to drop "NCBI Superkingdom" as well as the "new" column I used for comparison purposes.

In [90]:
```
new_df = new_df.drop(['NCBI Superkingdom', 'New'], axis=1)
```

In [91]:
```
new_df.columns.to_list()
```

Out[91]:
```
['HMP ID',
 'Organism Name',
 'Domain',
 'HMP Isolation Body Site',
 'Gene Count',
 'Genus']
```

In [92]:
```
new_df.shape
```

Out[92]:
```
(1563, 6)
```

In [93]:
```
new_df["Organism Name"].nunique()
```

Out[93]:
```
1557
```

In [94]:
```
new_df["Genus"].nunique()
```

Out[94]:
```
185
```

In [95]:
```
new_df.columns
# new_df.head() #commented out for brevity
```

Out[95]:

	HMP ID	Organism Name	Domain	HMP Isolation Body Site	Gene Count	Genus
0	1	Abiotrophia defectiva ATCC 49176	bacterial	oral	1950	Abiotrophia
1	4	Achromobacter piechaudii ATCC 43553	bacterial	airways	5755	Achromobacter
2	5	Achromobacter xylosoxidans C54	bacterial	airways	6010	Achromobacter
3	10	Acinetobacter baumannii ATCC 19606	bacterial	urogenital_tract	3832	Acinetobacter
4	12	Acinetobacter calcoaceticus RUH2202	bacterial	skin	3632	Acinetobacter

Upcoming process:

1. ColumnTransformer Creation:

- A ColumnTransformer is instantiated, which applies different preprocessing to different subsets of features: StandardScaler for numerical features and OneHotEncoder for categorical features.

2. Data Transformation:

- The fit_transform method is called on the model_df DataFrame, standardizing the numerical features and encoding the categorical features into a format suitable for clustering.

3. Elbow Method:

- The elbow method is used to determine the optimal number of clusters (k) by plotting the within-cluster sum of squares (WCSS) against the number of clusters. The "elbow" point in the graph indicates the optimal k.

4. K-Means Clustering:

- K-Means clustering is applied to the processed data with the chosen number of clusters (in this case, 7).

5. Cluster Analysis:

- The resulting clusters are then added as a new column to the model_df, and the count of data points in each cluster is outputted to give an initial understanding of the cluster distribution.

In [96]:
```
# First, save the identifiers in their own dataframe
identifiers_df = new_df[['HMP ID', 'Organism Name']]
identifiers_df.head()
```

Out[96]:

	HMP ID	Organism Name
0	1	Abiotrophia defectiva ATCC 49176
1	4	Achromobacter piechaudii ATCC 43553
2	5	Achromobacter xylosoxidans C54
3	10	Acinetobacter baumannii ATCC 19606
4	12	Acinetobacter calcoaceticus RUH2202

In [97]:
```
# Removing the identifiers ('HMP ID' and 'Organism Name')
model_df = new_df.drop(['HMP ID', 'Organism Name'], axis=1)
model_df.head()
```

Out[97]:

	Domain	HMP Isolation Body Site	Gene Count	Genus
0	bacterial	oral	1950	Abiotrophia
1	bacterial	airways	5755	Achromobacter
2	bacterial	airways	6010	Achromobacter
3	bacterial	urogenital_tract	3832	Acinetobacter
4	bacterial	skin	3632	Acinetobacter

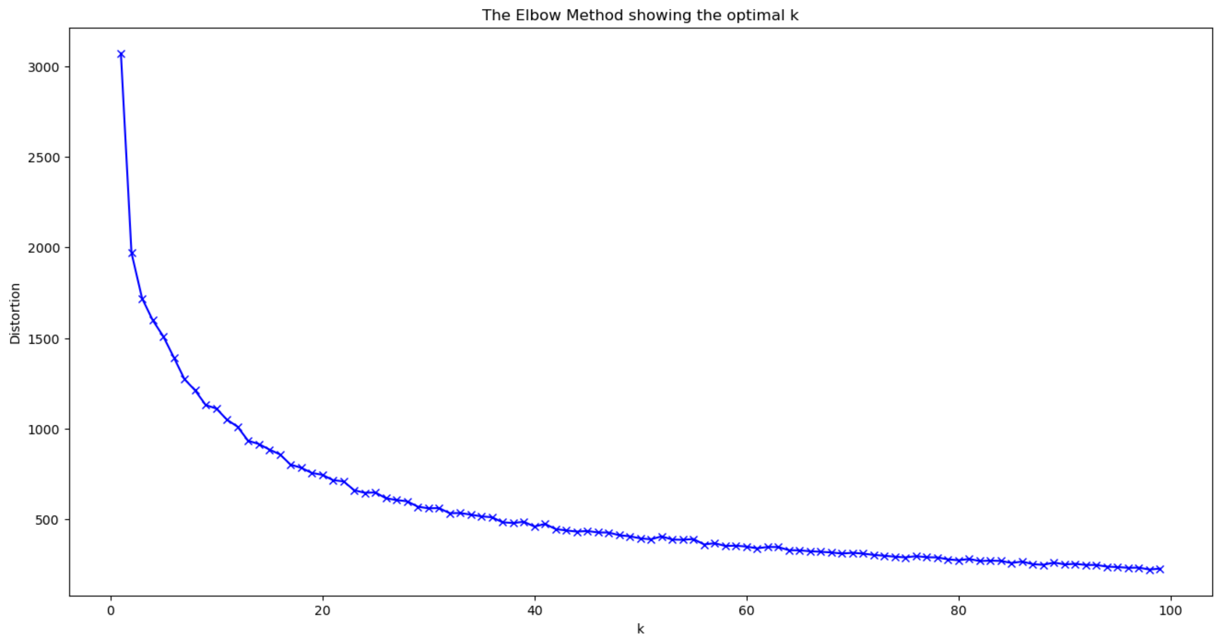
In [153…
```
# Feature Encoding for cluster modeling includes specifying the numerical ar
# Assign features for clustering
features = ['HMP Isolation Body Site', 'Gene Count', 'Genus']

# Separate features for encoding and scaling
categorical_features = ['HMP Isolation Body Site', 'Genus']
numerical_features = ['Gene Count']
```

In [154…
```
# Create the ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)])
```

In [155…
```
# Fit and transform the data
X_processed = preprocessor.fit_transform(model_df)
```

In [156…
```
# Use the elbow method to choose the optimal number of clusters k
distortions = []
K = range(1, 100)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X_processed)
    distortions.append(kmeanModel.inertia_)
```

In [157…
```
# Plot the elbow graph
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



"The elbow method is a graphical representation of finding the optimal ‘K’ in a K-means clustering. It works by finding WCSS (Within-Cluster Sum of Square) i.e. the sum of the square distance between points in a cluster and the cluster centroid." "Now we will use Euclidean distance or Manhattan distance as the metric to calculate the distance of the points from the nearest centroid and assign the points to that nearest cluster centroid, thus creating K clusters." https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/

In [158…
```
# Apply k-means clustering with optimal k (based on elbow method)
kmeans = KMeans(n_clusters=7)
clusters = kmeans.fit_predict(X_processed)
```

In [159…
```
# Add cluster labels to DataFrame
model_df['Cluster'] = clusters
```

In [160…
```
# Analyze the clusters
# For example, you can see how many organisms fall into each cluster
print(model_df['Cluster'].value_counts())
```

```
Cluster
6    551
2    286
3    229
1    219
5    127
0     92
4     59
Name: count, dtype: int64
```

In [161…
```
X_processed
```

Out[161…
```
<1563x186 sparse matrix of type '<class 'numpy.float64'>'
	with 3126 stored elements in Compressed Sparse Row format>
```

PCA does not support sparse input with the solver set to "auto" when dealing with sparse matrices. The PCA implementation in scikit-learn requires dense input or, if using sparse input, the solver must be explicitly set to "arpack". However, PCA is generally not recommended for sparse data due to the densification process, which can be very memory intensive. Instead, using TruncatedSVD is often recommended for dimensionality reduction on sparse datasets because it is designed to handle sparse matrices more efficiently.

Cluster Profiling: For each cluster, calculate the mean or median of the numerical features and the mode of the categorical features. This will give you an insight into what each cluster represents or characterizes.

Statistical Tests: If your dataset has labeled data or you want to understand the statistical significance of the clusters with respect to some numerical attributes, you can perform ANOVA or other relevant tests to see if the mean of the numerical features significantly differs between clusters.

Visualize Clusters: Use dimensionality reduction techniques like PCA or t-SNE to visualize the clusters in two or three dimensions. This can give you a visual understanding of how well-separated the clusters are.

Interpret Clusters: Based on the profiles and visualizations, interpret what each cluster might represent. If you have domain knowledge, use it to label each cluster meaningfully.

Evaluate Cluster Quality: Beyond the elbow method, use metrics like silhouette score, Davies-Bouldin index, or the Calinski-Harabasz index to evaluate the quality of the clusters.

In [162…

```
# Calculating the mean or median for numerical features
numerical_profiles = model_df.groupby('Cluster')[numerical_features].median(

# Calculating the mode for categorical features
categorical_profiles = model_df.groupby('Cluster')[categorical_features].agg
```

In [163…

```
# Cluster Profiling
cluster_profiles = model_df.groupby('Cluster').agg({**{num: 'mean' for num i
                                                    **{cat: lambda x: x.mode
```

In [164…

```
print(cluster_profiles)
```

```
          Gene Count              Genus
Cluster                                
0        2525.239130  Propionibacterium
1        5268.652968        Escherichia
2        3364.475524       Enterococcus
3        1955.414847      Streptococcus
4          34.033898      Lactobacillus
5        2246.196850      Lactobacillus
6        2146.689655     Staphylococcus
```

The cluster summary table reveals distinct microbial community profiles based on gene count and predominant isolation body site. For instance, Cluster 0, with the highest gene count, predominantly comprises Escherichia from the gastrointestinal tract, indicating a robust gene diversity in this environment. Conversely, Cluster 5, associated with the gastrointestinal tract, features Helicobacter with a lower gene count, suggesting variation in gene complexity within the same body site. Clusters also highlight specific microbial presences, like Lactobacillus in the urogenital tract and Propionibacterium on the skin, reflecting their ecological niches and potential roles in health and disease.

In [165…

```
from scipy.stats import f_oneway

# Perform ANOVA across clusters for a numerical attribute
f_oneway(*(model_df[model_df['Cluster'] == cluster]['Gene Count'] for cluste
```

Out[165…

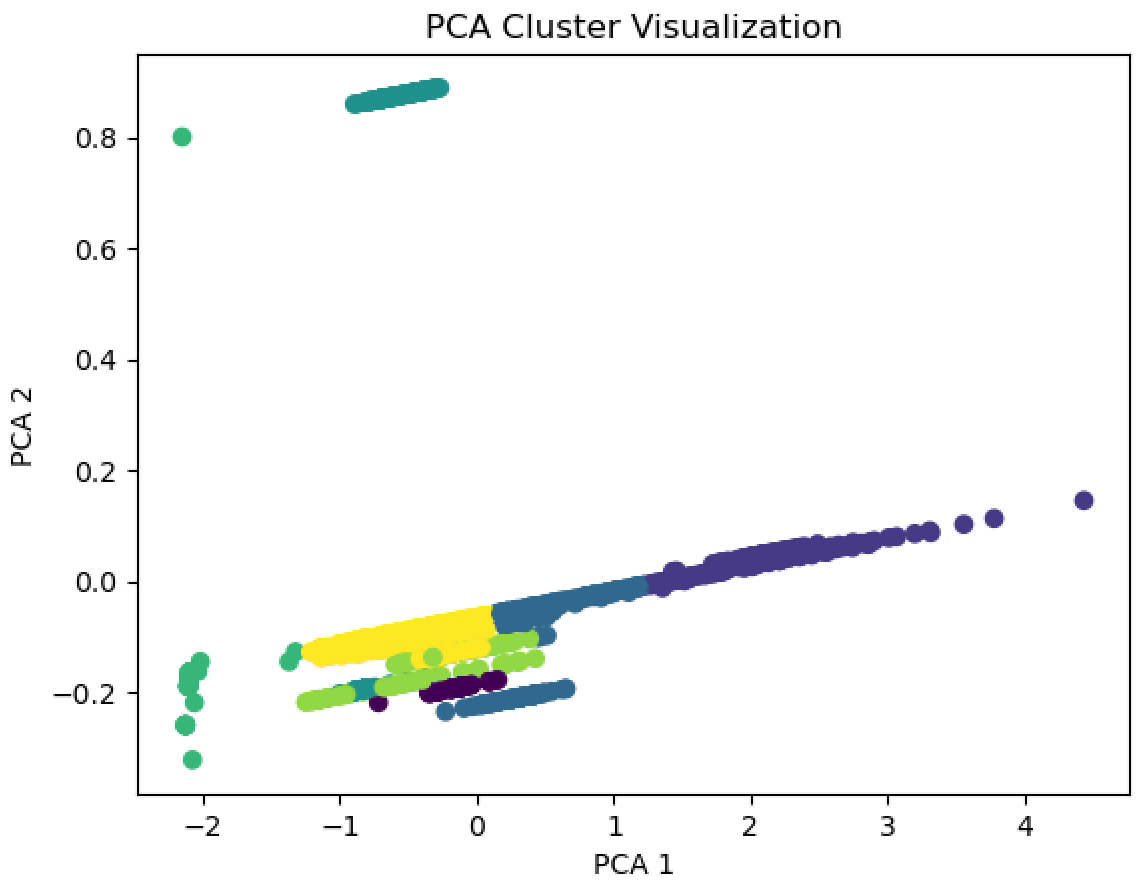
```
F_onewayResult(statistic=2134.7271638526618, pvalue=0.0)
```

The ANOVA test result with a statistic of 1084.1518968807904 and a p-value of 0.0 suggests that there is a statistically significant difference in the mean gene counts across different clusters. The high F-statistic value indicates a strong between-group variance compared to within-group variance, reinforcing the significance of the clusters in terms of gene count variation. The p-value being 0 (or very close to 0) means this result is highly significant, rejecting the null hypothesis that all group means are equal.

This suggests that the clusters formed have distinct microbial characteristics based on their gene counts.

In [166…

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# PCA for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_processed.toarray())  # Convert sparse matrix to

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters)
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('PCA Cluster Visualization')
plt.show()
```

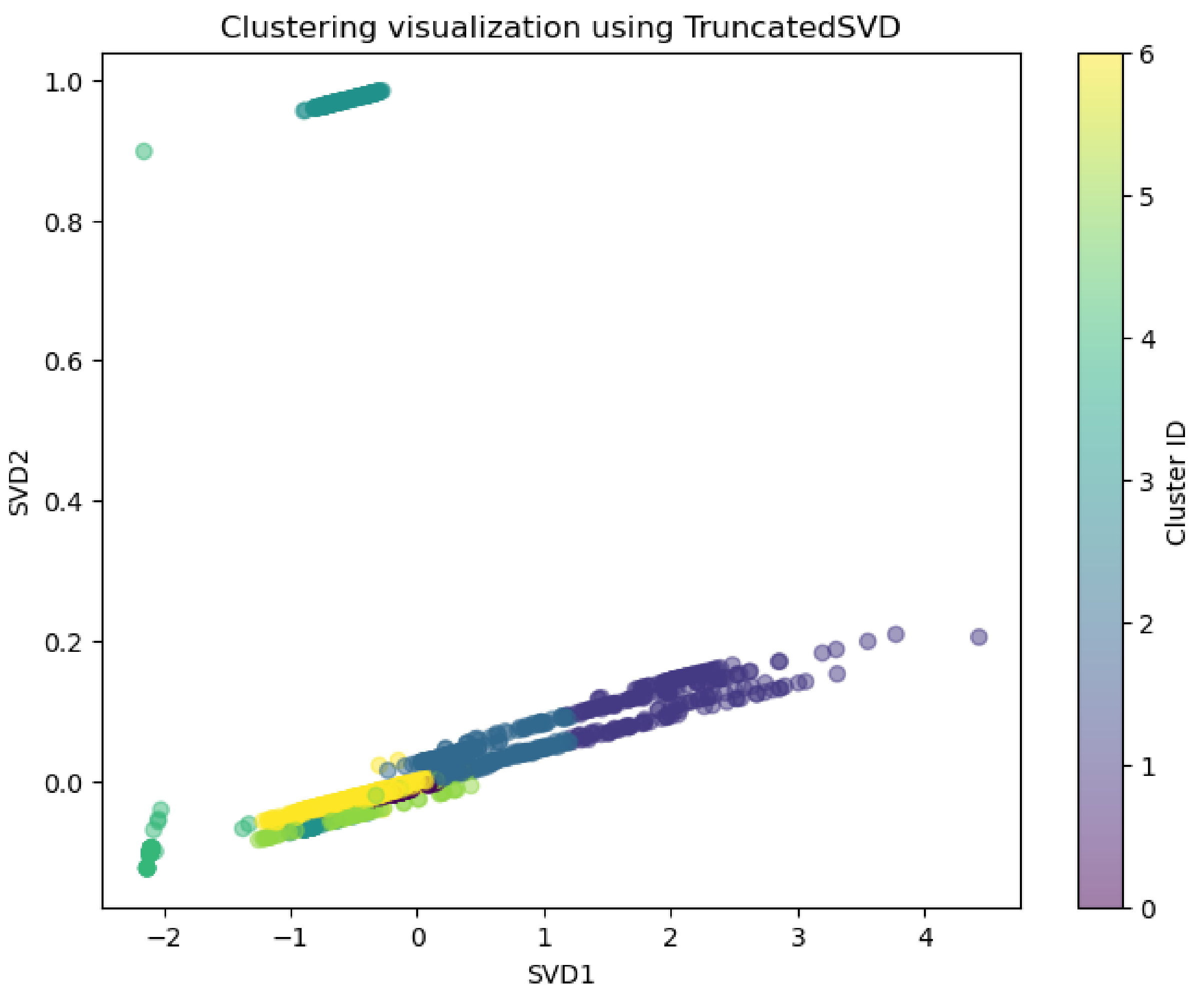


In [167…

```
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt

# Perform TruncatedSVD
svd = TruncatedSVD(n_components=2)
X_reduced = svd.fit_transform(X_processed)
```

In [168…

```
# Plot the transformed data
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=clusters, cmap='viridis', al
plt.xlabel('SVD1')
plt.ylabel('SVD2')
plt.title('Clustering visualization using TruncatedSVD')
plt.colorbar(label='Cluster ID')
plt.show()
```



In [169…

```
from sklearn.metrics import silhouette_score

# Evaluate silhouette score
silhouette_avg = silhouette_score(X_processed, clusters)
print(f'Silhouette Score: {silhouette_avg}')
```

```
Silhouette Score: 0.24429453257794126
```

A silhouette score of 0.25275623208645914 suggests that the cluster separation is fair but not strong. Silhouette scores range from -1 (poor clustering) to +1 (perfect clustering), with scores around 0 indicating overlapping clusters. Your score indicates that while there is some structure to the clusters, there might be room for improvement either by adjusting the number of clusters, reconsidering the features used, or applying a different clustering technique.

This project applies machine learning techniques to analyze the human microbiome, focusing on clustering microbial species based on their genetic characteristics and isolation sites. The model leverages k-means clustering to group organisms, revealing patterns and associations between microbial genuses, gene counts, and their prevalence in different body sites. The statistical analysis, including ANOVA, confirms significant differences among clusters, enhancing our understanding of the microbiome's composition and its potential health implications. The silhouette score indicates moderate cluster separation, suggesting room for model refinement but affirming its utility in microbiome research. This approach offers insights into microbial diversity and its role in human health, paving the way for targeted therapeutic interventions.

This project employs a microbiome-focused approach to examine the human microbiota's composition, aiming to uncover patterns and correlations with health conditions. Through advanced machine learning techniques, including k-means clustering and PCA, the project analyzes microbial diversity and gene count data to identify distinct microbial clusters associated with different body sites. The findings reveal significant microbial community variations, highlighting potential implications for diagnosing and treating health issues. The analysis underscores the microbiome's complexity and its potential as a biomarker for health, paving the way for personalized medicine strategies that consider microbial composition.