Progress Report

Introduction

This project aims to analyze the detailed dataset of the vaginal microbiome from pregnant women to explore potential indicators of preterm birth, which is medically defined as birth before the completion of 37 weeks of gestation. Given the significant prevalence of preterm births globally and its associated health risks for infants, our academic pursuit is to derive meaningful insights from the data that could reveal underlying biological patterns or correlations.

Our project's scope has narrowed, focusing more closely on age-related analysis and the role of microbial diversity in preterm births. Some aspects initially considered, such as certain demographic factors, have been set aside to make the study more focused on biologically relevant features.

Goal: Develop a ML or deep NN model for preterm prediction. Determine the risk of preterm birth (delivery <37 weeks vs. >=37 weeks)

Null Hypothesis: Age does not have an impact on pre-term birth.

Alternate Hypothesis: Age has an impact on pre-term birth.

```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.stats import shapiro, mannwhitneyu, ttest_ind, chi2_contingency
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# colab-specific file access
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/My Drive/Colab Notebooks/Project
```

```
Mounted at /content/drive
/content/drive/My Drive/Colab Notebooks/Project
```

```
# getting the data from alpha_diveristy table
alpha_diversity = pd.read_csv('alpha_diversity.csv')
#cst_valencia_w_taxons = pd.read_csv('cst_valencia_w_taxons.csv')
#cst_valencia = pd.read_csv('cst_valencia.csv')
#krd_distance_long = pd.read_csv('krd_distance_long.csv')
#krd_distance_wide = pd.read_csv('krd_distance_wide.csv')
metadata_normalized = pd.read_csv('metadata_normalized.csv')
metadata = pd.read_csv('metadata.csv')
#phylotype_nreads_1e_1 = pd.read_csv('phylotype_nreads.1e_1.csv')
#phylotype_nreads_1e_0 = pd.read_csv('phylotype_nreads.1e0.csv')
#phylotype_nreads_5e_1 = pd.read_csv('phylotype_nreads.5e_1.csv')
#phylotype_relabd_1e_1 = pd.read_csv('phylotype_relabd.1e_1.csv')
#phylotype_relabd_1e_0 = pd.read_csv('phylotype_relabd.1e0.csv')
#phylotype_relabd_5e_1 = pd.read_csv('phylotype_relabd.5e_1.csv')
```

```
alpha_diversity.head()
```

	specimen	shannon	inv_simpson	bwpd	phylo_entropy	quadratic	unrooted_pd	rooted_pd
0	A00001-05	1.00000	1.00000	0.00000	-0.00000	0.000000	0.00000	2.53935
1	A00002-01	1.96362	1.81277	2.62894	1.31887	0.876314	3.94341	4.14816
2	A00003-02	1.00000	1.00000	0.00000	-0.00000	0.000000	0.00000	2.62632
3	A00004-08	1.00000	1.00000	0.00000	-0.00000	0.000000	0.00000	1.83870
4	A00004-12	6.94884	4.07385	2.78896	3.13422	1.219900	15.51850	15.58460

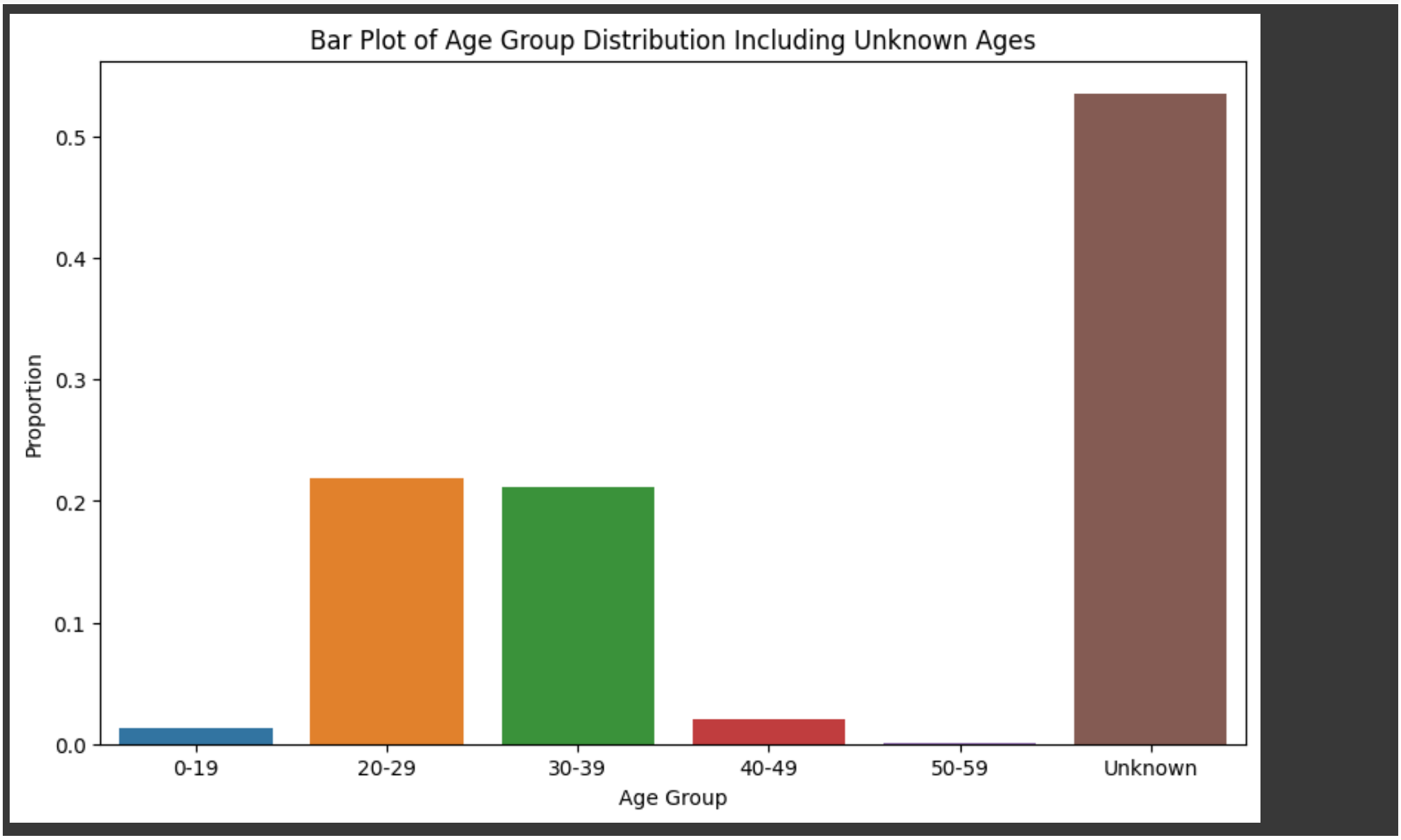
```
metadata
```

	project	specimen	participant_id	was_term	delivery_wk	collect_wk	race	age	NIH Racial Category	NIH Ethnicity Category
0	A	A00001-05	A00001	False	38.0	33.0	American Indian	Unknown	American Indian or Alaska Native	Unknown
1	A	A00002-01	A00002	False	40.0	38.0	White	Unknown	White	Unknown
2	A	A00003-02	A00003	False	40.0	30.0	Asian-Japanese	Unknown	Asian	Unknown
3	A	A00004-08	A00004	False	40.0	27.0	White	Unknown	White	Unknown
4	A	A00004-12	A00004	False	40.0	29.0	White	Unknown	White	Unknown
...	...	...	...	...	...	...	...	...	...	...
3573	J	J00111-01	J00111	False	40.0	17.0	Caucasian	27	White	Unknown

The metadata contains all the details of the conducted experiment, with various columns including 'project', 'specimen', 'participant_id', 'was_term', 'delivery_wk', 'collect_wk', 'race', 'age', 'NIH Racial Category', 'NIH Ethnicity Category', 'was_preterm', and 'was_early_preterm'. In light of our hypothesis that "Age does not have an impact on pre-term birth," we have streamlined the dataset by removing columns unnecessary for our analysis. This included the deletion of demographic columns like 'NIH Racial Category' and 'NIH Ethnicity Category'. Additionally, we refined our approach to focus on the influence of maternal age and the incidence of preterm births.

During the data cleaning process, ages that were not specified and labeled as 'Unknown' were converted to NaN values and then removed to ensure the integrity of our age-related analysis. We then narrowed down the dataset to include only the most recent samples collected before the 37th week of gestation, aligning the microbiome data with the critical timeframe relevant to our study of preterm birth risks.

```
df1 = metadata
# Handle missing values for 'age' by replacing them with 'Unknown'
df1['age'].replace('Unknown', value=np.nan, inplace=True)  # Replace 'Unknown' with NaN for conversion
df1['age'] = pd.to_numeric(df1['age'], errors='coerce')  # Convert age to numeric, coercing errors to NaN
df1['age_group'] = pd.cut(df1['age'], bins=[0, 20, 30, 40, 50, 60],
                          right=False, labels=['0-19', '20-29', '30-39', '40-49', '50-59'])  # Bin ages into groups
df1['age_group'] = df1['age_group'].cat.add_categories(['Unknown'])
df1['age_group'].fillna('Unknown', inplace=True)  # Assign 'Unknown' to NaN values

# Now let's plot the bar plot including the 'Unknown' age values
plt.figure(figsize=(10, 6))
age_group_counts = df1['age_group'].value_counts(normalize=True).sort_index()
sns.barplot(x=age_group_counts.index, y=age_group_counts.values)
plt.title('Bar Plot of Age Group Distribution Including Unknown Ages')
plt.xlabel('Age Group')
plt.ylabel('Proportion')
plt.show()
```



The bar plot displays the proportions of the dataset's participants across different age groups, with a specific category for those whose ages are not known ('Unknown'). The x-axis represents the age groups, typically binned into ranges like '0-19', '20-29', '30-39', '40-49', '50-59', and 'Unknown'. The y-axis shows the proportion of participants that fall into each of these age groups.

Each bar in the plot corresponds to an age group, with the bar's height indicating the proportion of the dataset's participants in that group. For instance, if the '20-29' age group has a bar that is twice as tall as the '30-39' group, it means that the proportion of participants aged 20-29 is double that of those aged 30-39 in the dataset.

The 'Unknown' category has its own bar, which is particularly informative. The height of this bar indicates the proportion of the dataset for which age information is missing. This is an important aspect of the data to consider, as a significant proportion of 'Unknown' values could affect the reliability of any age-related analysis or conclusions drawn from the dataset.

By providing a clear visual comparison of the different age groups, this bar plot helps identify which age ranges are most prevalent within the study and the extent of missing age information, which might impact subsequent analyses that are dependent on age data.

Cleaning Data and Preprocessing (Done by Victor & Ricardo):

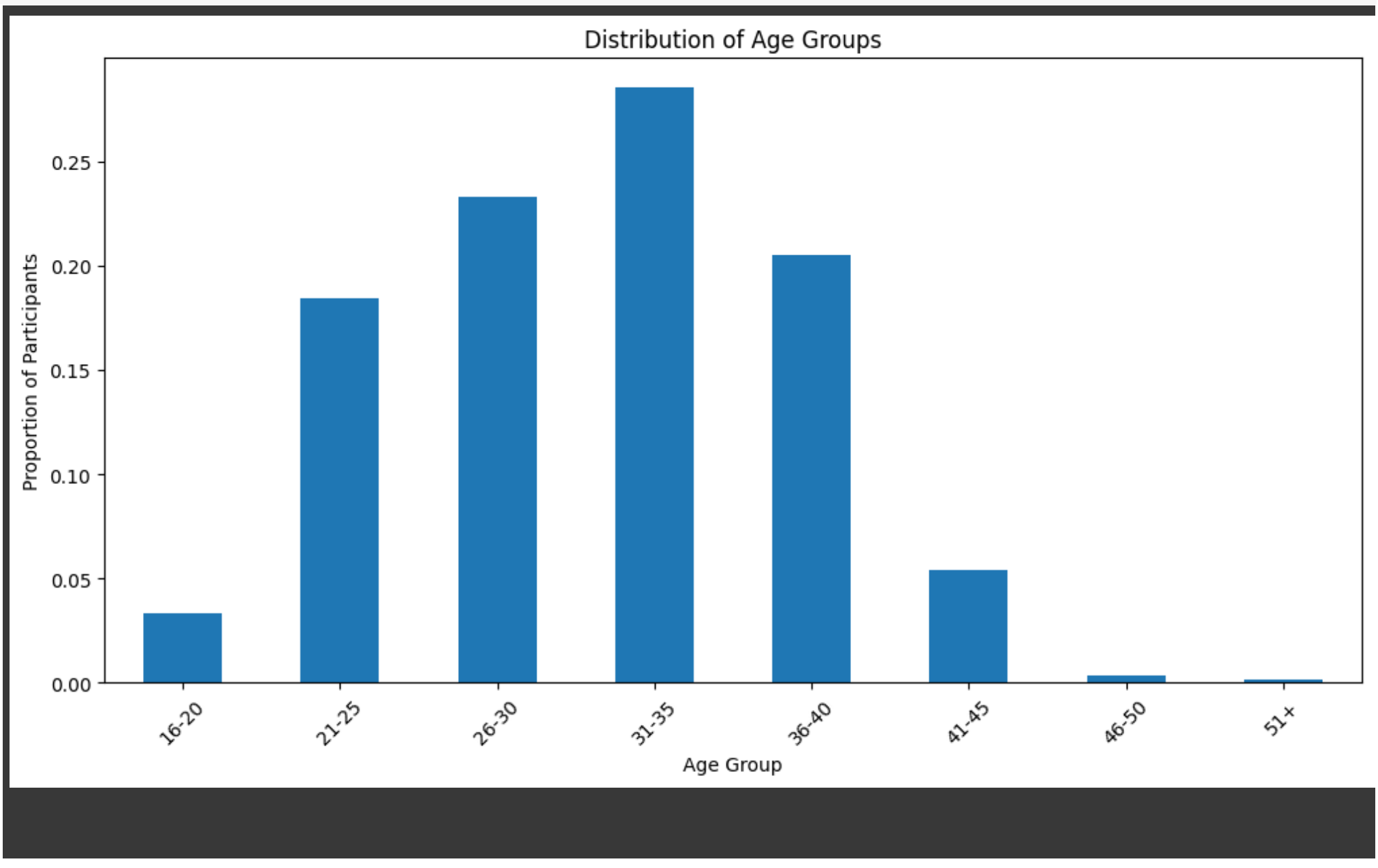
```
cleanData = metadata[metadata['age'] != 'Unknown']
columnsDelete = ['project', 'NIH Racial Category', 'NIH Ethnicity Category']

updatedData = cleanData.drop(columns=columnsDelete)

df = updatedData
# Data cleaning: Convert 'age' to numeric and create 'age_group' categories
df['age'] = pd.to_numeric(df['age'], errors='coerce')
bins = [15, 20, 25, 30, 35, 40, 45, 50, 55]
labels = ['16-20', '21-25', '26-30', '31-35', '36-40', '41-45', '46-50', '51+']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)
```

```
# Filter the dataset for samples collected before week 37 and get the latest sample for each participant
pre_week_37_df = df[df['collect_wk'] < 37]
sorted_df = pre_week_37_df.sort_values(by=['participant_id', 'collect_wk'], ascending=[True, False])
latest_samples_df = sorted_df.drop_duplicates(subset='participant_id')
```

We limited our analysis to the latest sample collected before week 37 of pregnancy for each participant. The rationale for using 'collect_wk' instead of 'delivery_wk' is rooted in the focus of the study. By focusing on 'collect_wk', we ensure that our analysis is based on the condition of the participant and the sample status before the onset of full-term labor (which is typically around week 37). This can be particularly important if the purpose is to predict or understand factors that contribute to pre-term labor or other conditions before full-term delivery. Additionally, 'delivery_wk' might not be available or accurately known for all records, especially if the delivery occurred unexpectedly or outside of a clinical setting.

EDA and Visualization: (Done by Kalpkumar & Naivik)

```
# Exploratory Data Analysis (EDA) with visualization
# Distribution of Age Groups
plt.figure(figsize=(12, 6))
age_group_dist = latest_samples_df['age_group'].value_counts(normalize=True).sort_index()
age_group_dist.plot(kind='bar')
plt.title('Distribution of Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Proportion of Participants')
plt.xticks(rotation=45)
plt.show()
```



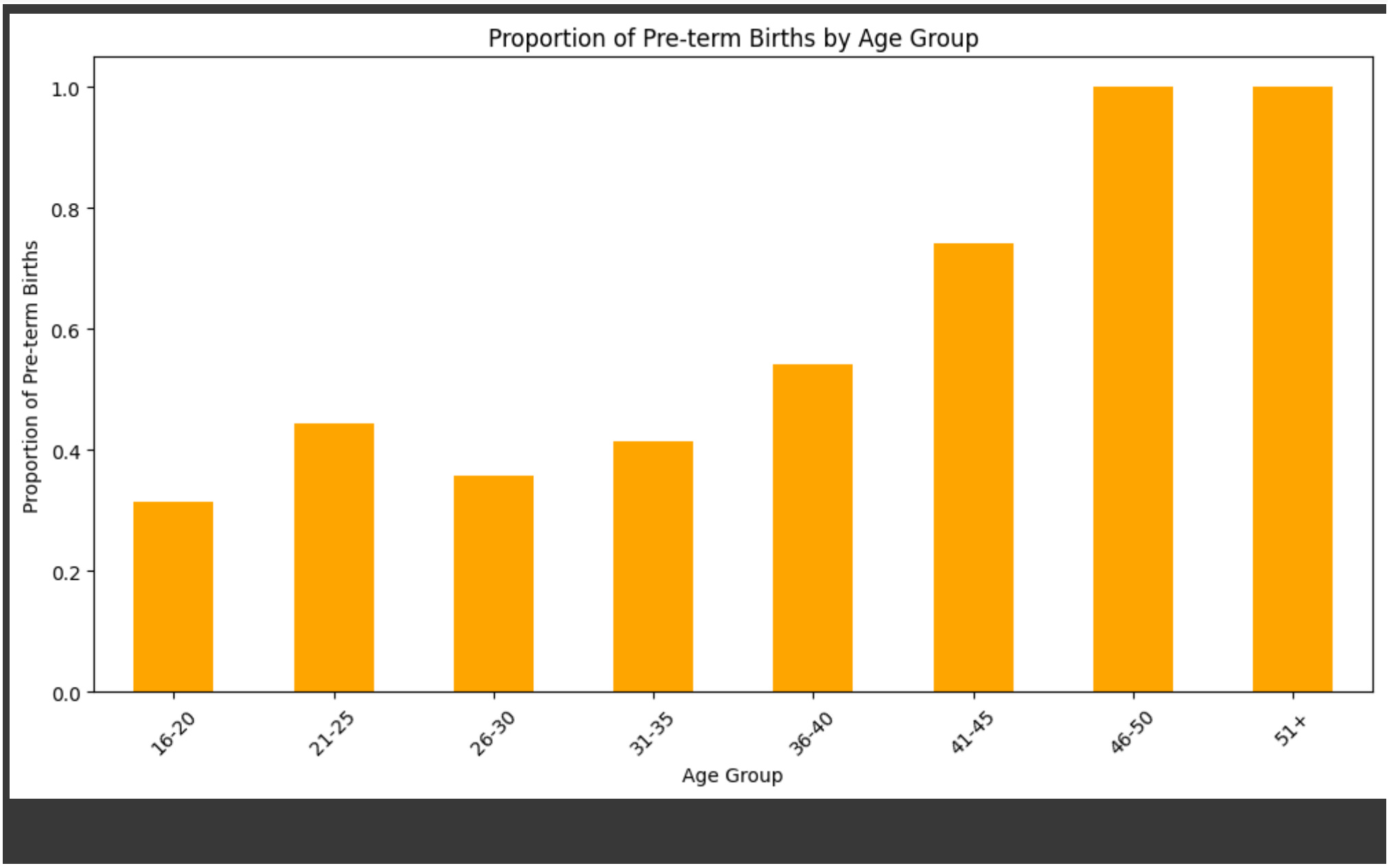
This bar chart illustrates the distribution of age groups among participants in a birth data study. The age groups are divided into seven categories: '16-20', '21-25', '26-30', '31-35', '36-40', '41-45', '46-50', and '51+'. The y-axis represents the proportion of participants in each age group relative to the total number of participants in the study.

From the chart, we can observe that:

The '31-35' age group has the highest proportion of participants, suggesting that individuals in this age range are the most represented in the study population. The second and third most represented age groups are '26-30' and '36-40', respectively. The '46-50' and '51+' age groups have the smallest proportions, indicating fewer participants in these age ranges.

This bar chart helps to quickly assess which age groups are most represented in the dataset and may also provide initial insights into age-related patterns regarding pre-term birth.

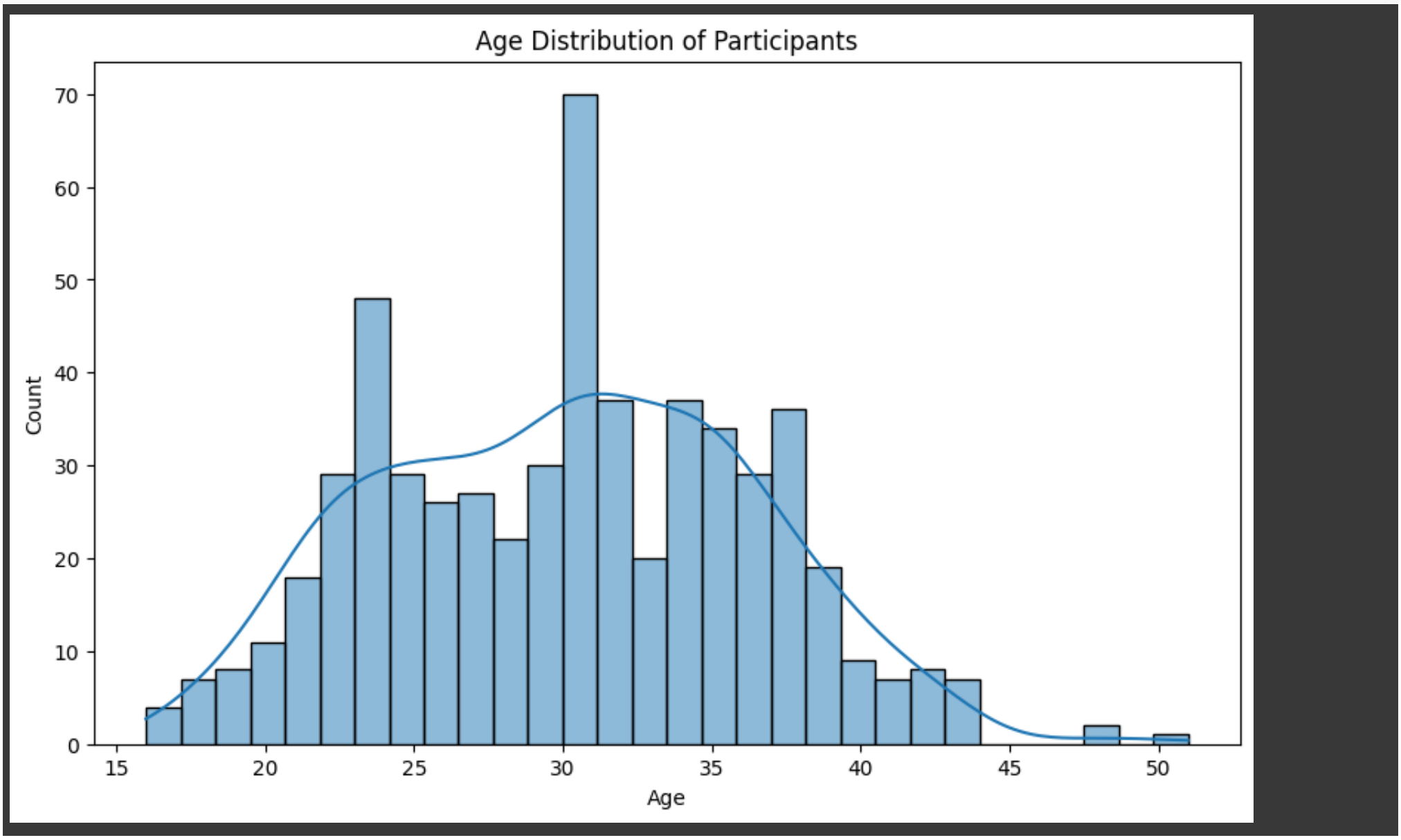
```
# Proportion of Pre-term Births by Age Group
plt.figure(figsize=(12, 6))
preterm_by_age = latest_samples_df.groupby('age_group')['was_preterm'].mean().sort_index()
preterm_by_age.plot(kind='bar', color='orange')
plt.title('Proportion of Pre-term Births by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Proportion of Pre-term Births')
plt.xticks(rotation=45)
plt.show()
```



The Proportion of Pre-term Births by Age Group visualization aims to explore the relationship between age and the incidence of pre-term births. It can suggest whether age is a factor in pre-term birth rates and, if so, which age groups are most at risk. This is particularly relevant for testing hypotheses about age as a risk factor and can inform healthcare providers about which age groups may require more attention or intervention.

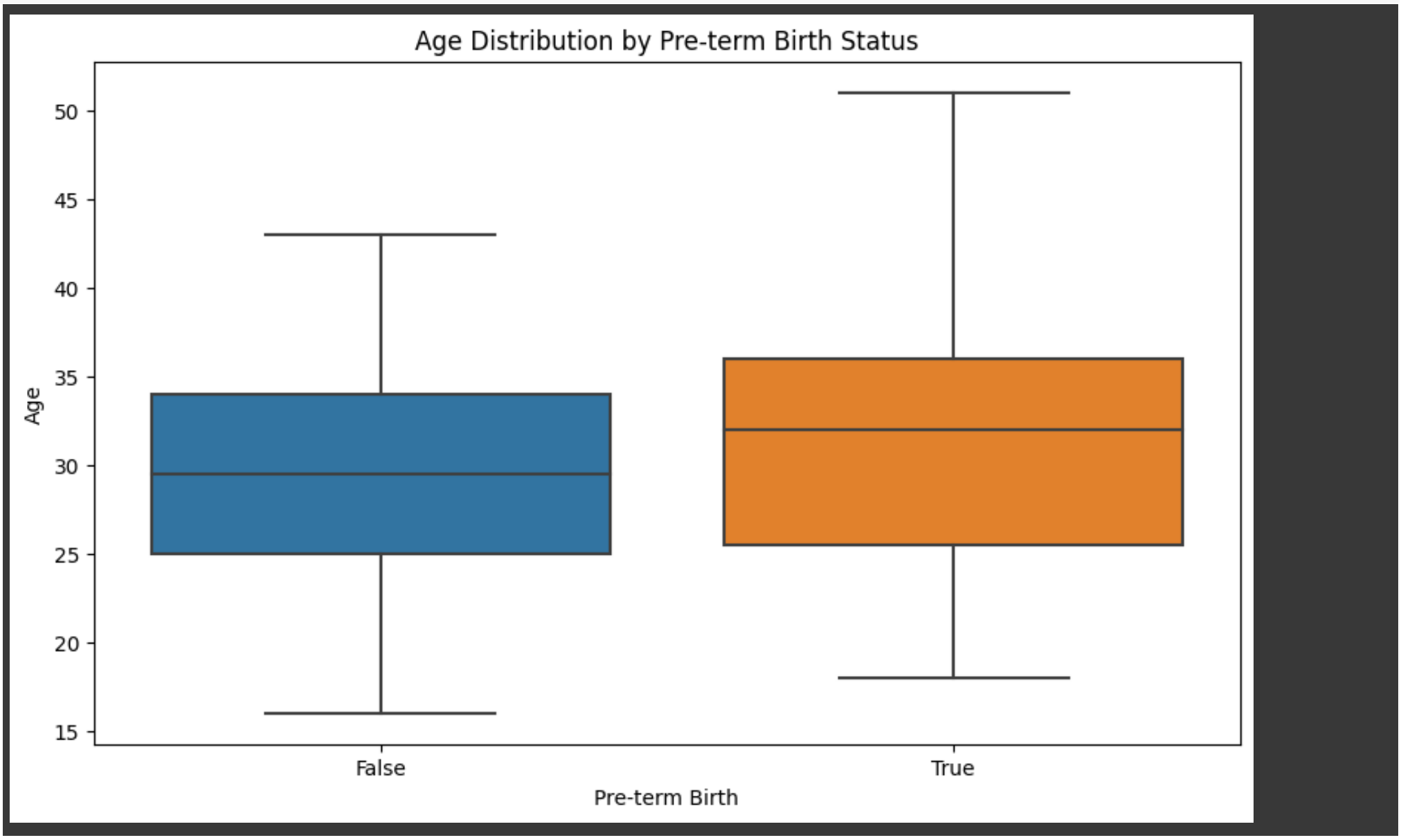
Both visualizations provide a foundation for the hypothesis testing that age may have an impact on pre-term birth rates. By visualizing the data before conducting any statistical tests or machine learning analyses, we can better understand the data's structure and identify any patterns or anomalies that may exist.

```
# Histogram of ages
plt.figure(figsize=(10, 6))
sns.histplot(data=latest_samples_df, x='age', bins=30, kde=True)
plt.title('Age Distribution of Participants')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



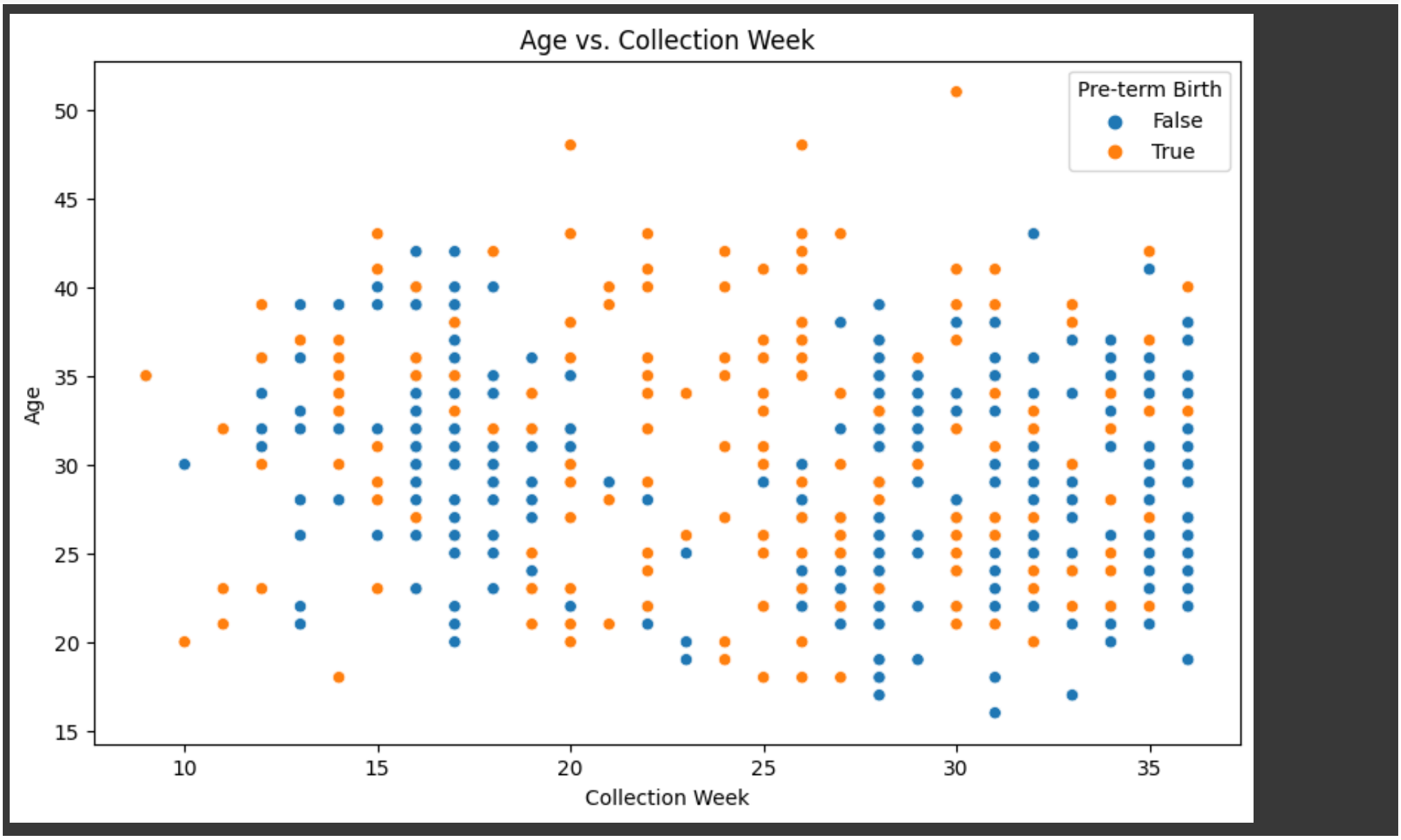
This histogram provides a visual representation of the distribution of ages among participants. It helps to understand the central tendency, dispersion, and the shape of the age distribution. By using kernel density estimation (KDE), it also gives an idea of the probability density of ages, highlighting the most common ages within the dataset.

```
# Boxplot comparing age distribution between pre-term and full-term births
plt.figure(figsize=(10, 6))
sns.boxplot(x='was_preterm', y='age', data=latest_samples_df)
plt.title('Age Distribution by Pre-term Birth Status')
plt.xlabel('Pre-term Birth')
plt.ylabel('Age')
plt.show()
```



A boxplot will compare the age distribution between two groups: those who had pre-term births and those who did not. This visualization can reveal differences in the median age, the spread of the ages (interquartile range), and any potential outliers. It's particularly useful for spotting differences in age distribution related to pre-term birth status.

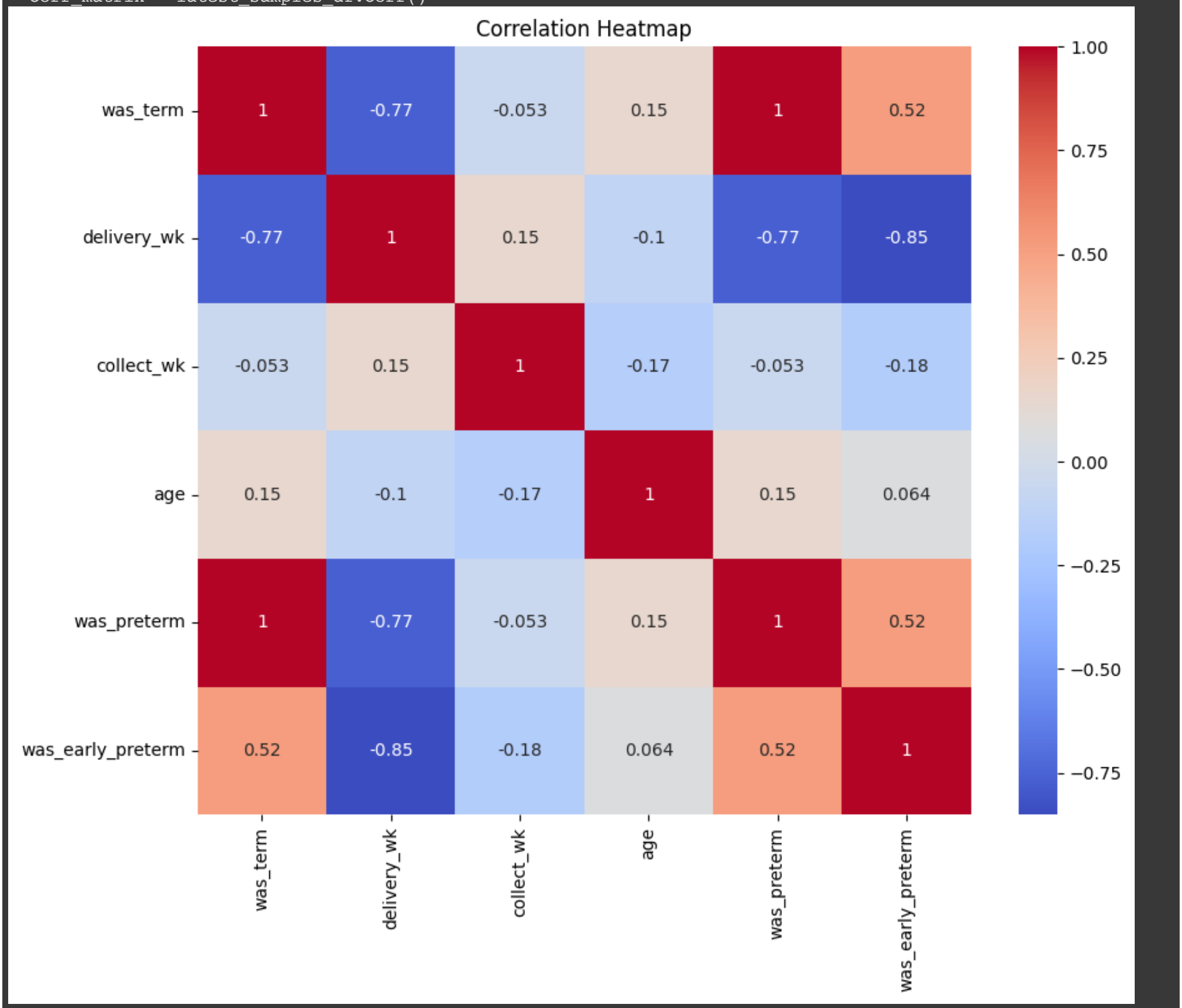
```
# Scatter plot of age versus collection week
plt.figure(figsize=(10, 6))
sns.scatterplot(x='collect_wk', y='age', data=latest_samples_df, hue='was_preterm')
plt.title('Age vs. Collection Week')
plt.xlabel('Collection Week')
plt.ylabel('Age')
plt.legend(title='Pre-term Birth')
plt.show()
```



The scatter plot visualizes the relationship between participant age and the gestational week of sample collection, with a clear distinction between pre-term and full-term births. The data points, representing individual samples, are spread across a range of ages and collection weeks without apparent clustering, suggesting no strong correlation between age and the timing of sample collection. Pre-term births, highlighted in orange, are interspersed throughout the plot, occurring across the full spectrum of ages and collection weeks. This dispersion indicates that within the dataset, pre-term birth outcomes are not predominantly associated with a specific age group or collection period prior to week 37. The absence of outliers and even distribution of points across the plot suggest that pre-term birth is a complex outcome likely influenced by multiple factors beyond just maternal age or the gestational age at the time of sample collection.

```
# Heatmap of correlations between numerical variables
plt.figure(figsize=(10, 8))
corr_matrix = latest_samples_df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

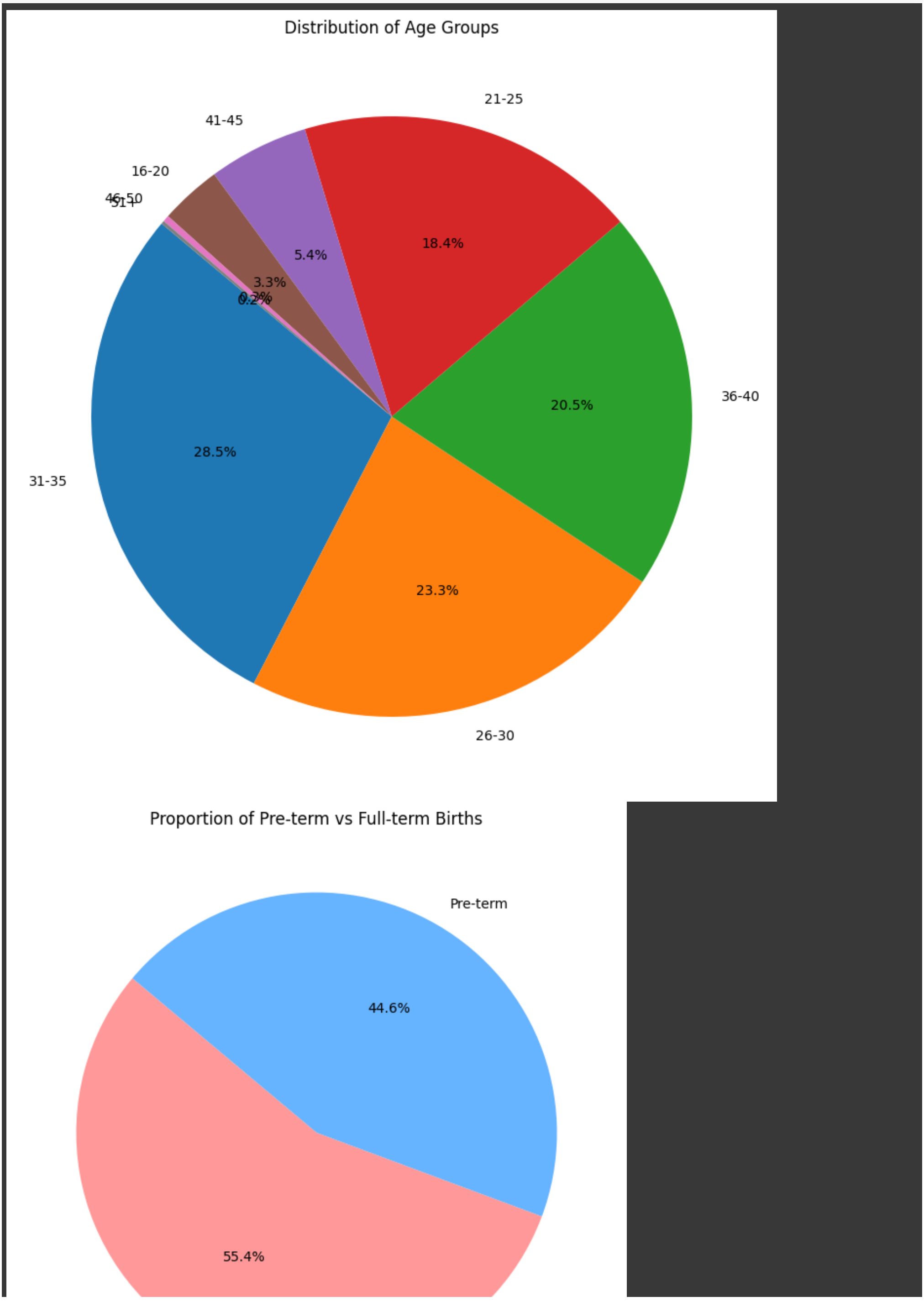
```
<ipython-input-58-94992cd772e3>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is depreca
  corr_matrix = latest_samples_df.corr()
```



The heatmap visualizes the correlation coefficients between several variables related to childbirth. Notably, 'was_term' and 'delivery_wk' exhibit a strong negative correlation, implying that full-term births typically occur later in the gestational timeline. In contrast, 'was_preterm' has a moderate positive correlation with 'age', suggesting a slight increase in pre-term births with maternal age. There is also a robust negative correlation between 'was_preterm' and 'delivery_wk', confirming that pre-term births happen earlier in pregnancy. Similarly, 'was_early_preterm' is strongly negatively correlated with 'delivery_wk', indicating that early pre-term births are associated with significantly earlier delivery. The positive correlation between 'was_preterm' and 'was_early_preterm' suggests common factors may influence both pre-term and early pre-term births. Other variables show weaker correlations, suggesting less direct relationships.

```
# Pie Chart of Age Group Distribution
age_group_counts = latest_samples_df['age_group'].value_counts()
plt.figure(figsize=(10, 10))
plt.pie(age_group_counts, labels=age_group_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Age Groups')
plt.show()

# Pie Chart of Pre-term vs Full-term Births
preterm_counts = latest_samples_df['was_preterm'].value_counts()
```

```
plt.figure(figsize=(8, 8))
plt.pie(preterm_counts, labels=['Full-term', 'Pre-term'], autopct='%1.1f%%', startangle=140, colors=['#ff9999','#66b3
plt.title('Proportion of Pre-term vs Full-term Births')
plt.show()
```



The 1st pie chart provides a clear and immediate sense of the relative sizes of the age groups. Each 'slice' represents a different age group, and the size of the slice is proportional to the number of participants in that group. This visualization is useful for identifying the most and least common age groups at a glance.

By illustrating the overall proportion of pre-term births, this chart can quickly convey the prevalence of pre-term births within the study population. It can help to set the stage for more in-depth analyses by providing a fundamental understanding of the dataset's composition regarding the outcome variable.

p-value Testing

```
from scipy.stats import pearsonr

df2 = metadata

# Remove any infinite or NaN values from the data
df2 = df2.replace([np.inf, -np.inf], np.nan).dropna(subset=['age', 'was_preterm'])


# List of variables to test
variables = ['was_term', 'delivery_wk', 'collect_wk', 'age', 'was_preterm', 'was_early_preterm']

# Perform the point-biserial correlation test
corr, p_value = pearsonr(df2['age'], df2['was_preterm'])

# Output the results
print(f"Correlation coefficient: {corr}")
print(f"P-value: {p_value}")

# Interpret the p-value
alpha = 0.05
if p_value < alpha:
    print("We reject the null hypothesis. There is a statistically significant correlation between age and pre-term b
else:
    print("We fail to reject the null hypothesis. There is no statistically significant correlation between age and p
```

```
Correlation coefficient: 0.12611568909493454
P-value: 2.514432169887928e-07
We reject the null hypothesis. There is a statistically significant correlation between age and pre-term birth.
```

The output from the correlation test indicates a correlation coefficient of approximately 0.126, which suggests a weak positive correlation between age and the incidence of pre-term birth. In other words, as age increases, there is a slight tendency for the risk of pre-term birth to increase as well. However, the strength of this relationship is not strong based on the correlation coefficient value.

The p-value is approximately (2.51 * 10^{-7}), which is much smaller than the commonly used significance level of 0.05. This low p-value indicates that the observed correlation is statistically significant. In statistical terms, the probability of observing such a correlation by chance, if there were actually no correlation in the population, is extremely low.

Given this result, we would reject the null hypothesis, which stated that there is no correlation between age and pre-term birth. By rejecting the null hypothesis, we are supporting the alternative hypothesis that there is indeed a correlation between age and the likelihood of pre-term birth.

It is important to note that while the correlation is statistically significant, the actual correlation is weak. This implies that while age can be considered a factor in the incidence of pre-term birth, it is not a strong predictor on its own. There are likely other factors at play that, when combined with age, would provide a more comprehensive understanding of the risks for pre-term birth.

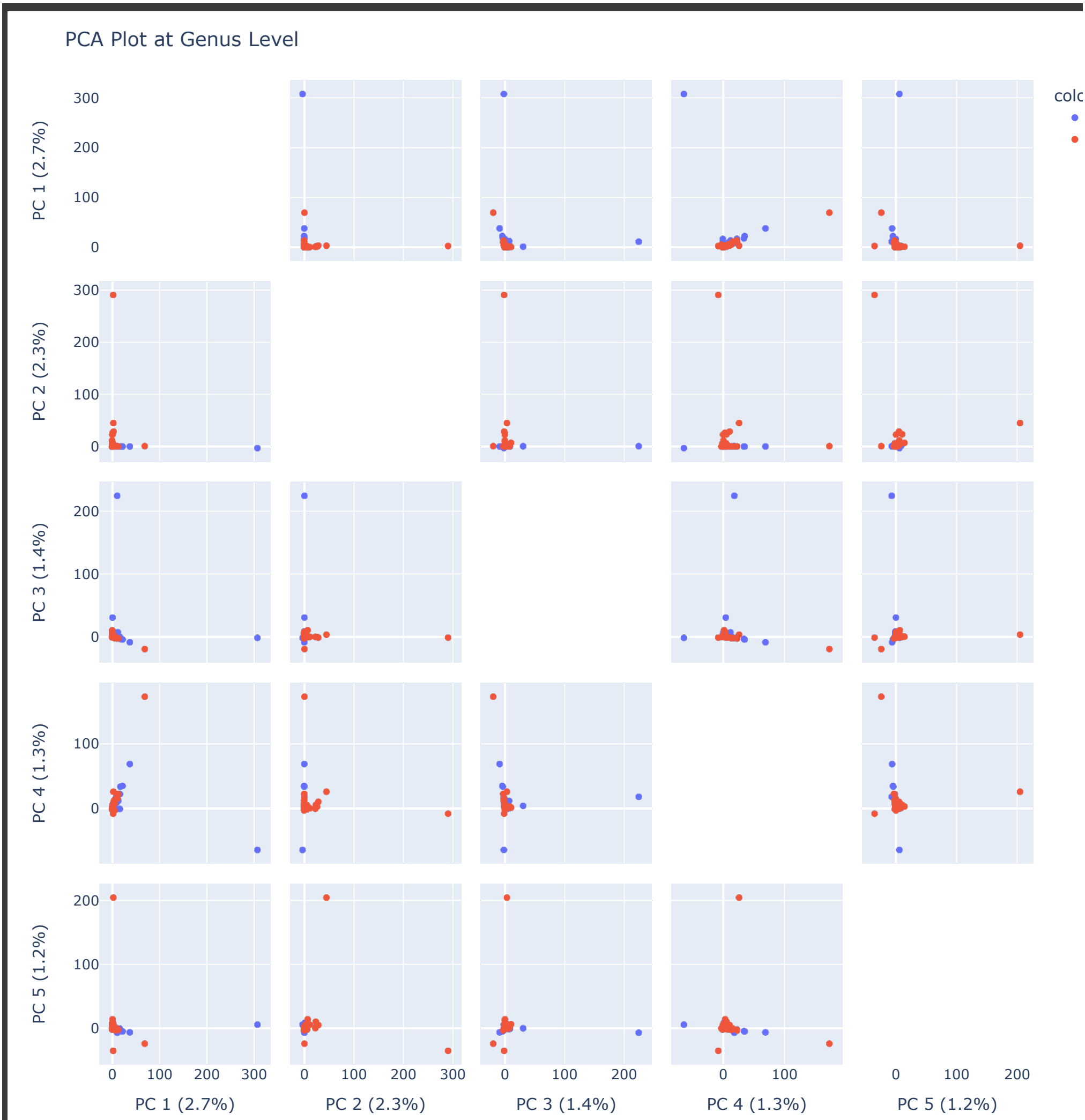
PCA Plots of taxanomy features: (Done by Ricardo and Tina)

Microbiome data is a comprehensive representation of the microbial communities residing in a particular environment, and it can be analyzed and characterized at various taxonomic levels, including species, genus, and family. At the genus level, microbiome data provides insights into groups of closely related microorganisms that share common traits, allowing for a more detailed understanding of the composition and diversity within a community. Moving to the species level refines the analysis, providing information about specific microbial entities capable of interbreeding. Furthermore, exploring microbiome data at the family level offers a broader perspective, grouping together related genera and highlighting shared evolutionary characteristics. The relationship between these taxonomic levels reflects the hierarchical nature of biological classification, where species make up genera, and genera constitute families. In this project, we possess microbiome data categorized at three distinct taxonomic levels: species, genus, and family. Employing Principal Component Analysis (PCA) plots, we aim to delve into each datase to choose one for the final model.

```
#PCA plot at species level:
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
scaler = StandardScaler()
metadata = pd.read_csv('metadata.csv')
microbiome = pd.read_csv('taxonomy_relabd.species.csv')
mergedDf = pd.merge(microbiome, metadata, on='specimen', how='left').reset_index(drop=True)

label = mergedDf[['was_term']]

specimenCol = microbiome[['specimen']]
microbiome = microbiome.iloc[:,1:]

scaler.fit(microbiome)
# Step 2: Transform the training set
scaled_microbiome = scaler.transform(microbiome)

pca = PCA()
components = pca.fit_transform(scaled_microbiome)
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_[0:5] * 100)
}

fig = px.scatter_matrix(
    components,
    labels=labels,
    dimensions=range(5),
    color=np.array(label['was_term'])
)

fig.update_traces(diagonal_visible=False)

# Set the size of the plot
fig.update_layout(
    title="PCA Plot at Species Level",
    width=1000,  # Set the width of the plot
    height=1000  # Set the height of the plot
)

fig.show()
```

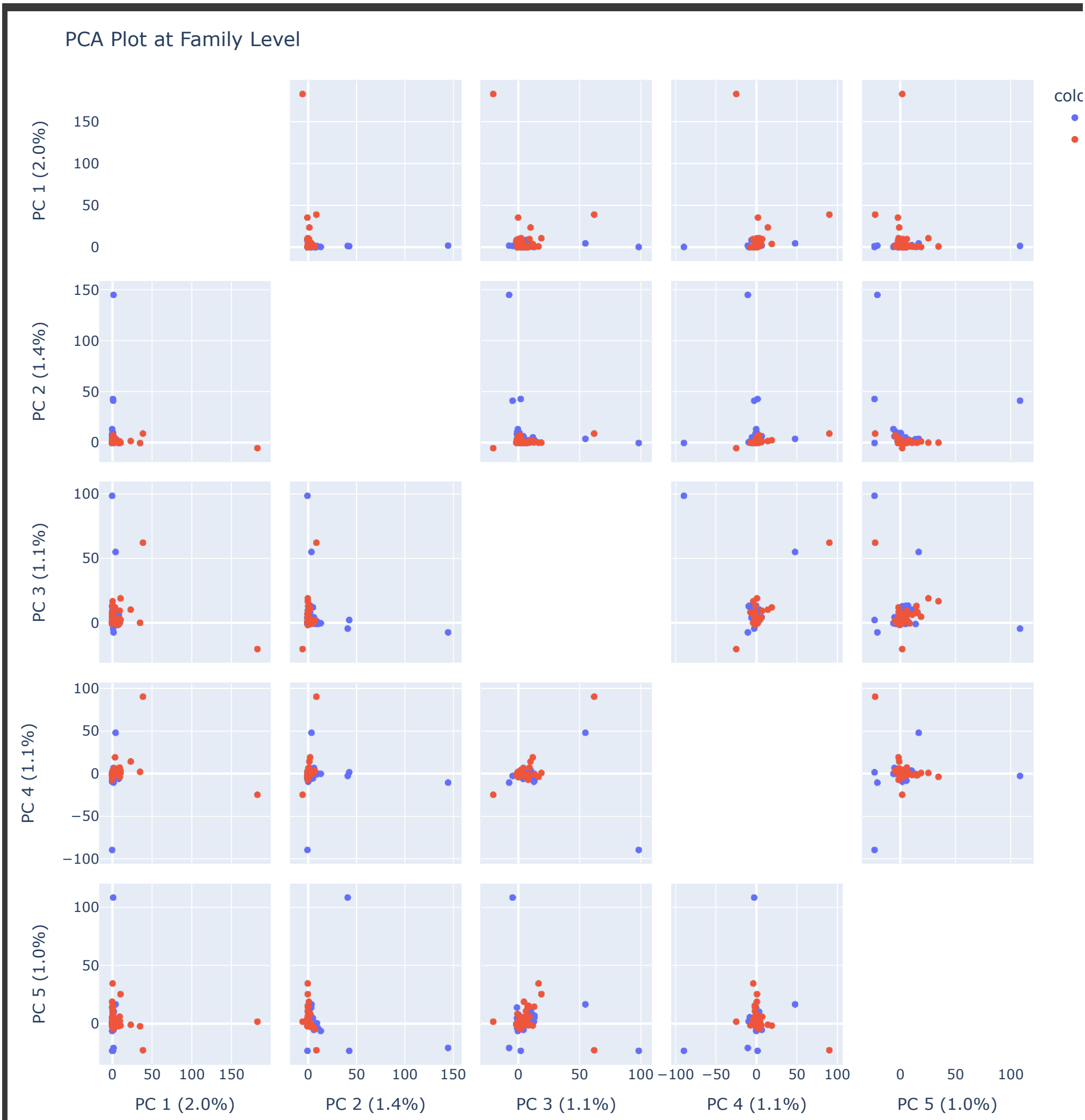

```
#PCA plot at genus level:
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
scaler = StandardScaler()
metadata = pd.read_csv('metadata.csv')
microbiome = pd.read_csv('taxonomy_relabd.genus.csv')
mergedDf = pd.merge(microbiome, metadata, on='specimen', how='left').reset_index(drop=True)

label = mergedDf[['was_term']]

specimenCol = microbiome[['specimen']]
microbiome = microbiome.iloc[:,1:]

scaler.fit(microbiome)
# Step 2: Transform the training set
scaled_microbiome = scaler.transform(microbiome)
```

```
pca = PCA()
components = pca.fit_transform(scaled_microbiome)
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_[0:5] * 100)
}

fig = px.scatter_matrix(
    components,
    labels=labels,
    dimensions=range(5),
    color=np.array(label['was_term'])
)

fig.update_traces(diagonal_visible=False)

# Set the size of the plot
fig.update_layout(
    title="PCA Plot at Genus Level",
    width=1000,  # Set the width of the plot
    height=1000  # Set the height of the plot
)

fig.show()
```




```
#PCA plot at family level:
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
scaler = StandardScaler()
metadata = pd.read_csv('metadata.csv')
microbiome = pd.read_csv('taxonomy_relabd.family.csv')
mergedDf = pd.merge(microbiome, metadata, on='specimen', how='left').reset_index(drop=True)

label = mergedDf[['was_term']]

specimenCol = microbiome[['specimen']]
microbiome = microbiome.iloc[:,1:]

scaler.fit(microbiome)
# Step 2: Transform the training set
scaled_microbiome = scaler.transform(microbiome)
```

```
pca = PCA()
components = pca.fit_transform(scaled_microbiome)
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_[0:5] * 100)
}

fig = px.scatter_matrix(
    components,
    labels=labels,
    dimensions=range(5),
    color=np.array(label['was_term'])
)

fig.update_traces(diagonal_visible=False)

# Set the size of the plot
fig.update_layout(
    title="PCA Plot at Family Level",
    width=1000,  # Set the width of the plot
    height=1000  # Set the height of the plot
)

fig.show()
```



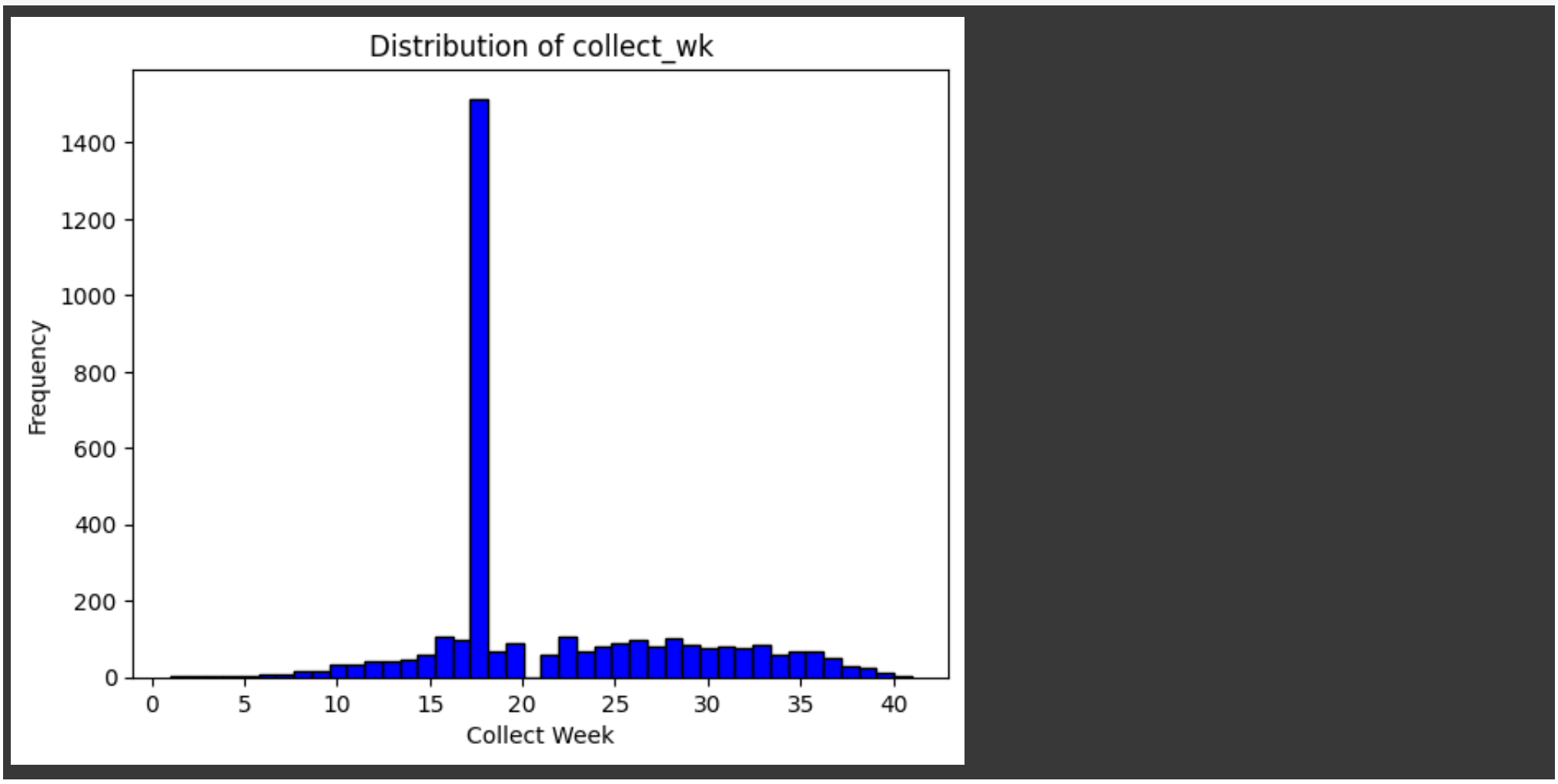
The presence of outliers in the data is impeding our ability to achieve a clear visualization and assess the distinctiveness of classes. In response, our next step involves refining our dataset by focusing specifically on the genus level. This targeted approach aims to mitigate the influence of outliers and enhance the clarity of our analysis. Subsequently, we will generate a PCA plot

Machine Learning Model: (Done by Tina)

```
metadata.columns
```

```
Index(['project', 'specimen', 'participant_id', 'was_term', 'delivery_wk',
       'collect_wk', 'race', 'age', 'NIH Racial Category',
       'NIH Ethnicity Category', 'was_preterm', 'was_early_preterm',
       'age_group'],
      dtype='object')
```

```
import matplotlib.pyplot as plt
plt.hist(metadata['collect_wk'], bins=42, color='blue', edgecolor='black')

# Set the title for the histogram
plt.title('Distribution of collect_wk')

# Add labels and other customizations if needed
plt.xlabel('Collect Week')
plt.ylabel('Frequency')

plt.show()
# add explanation
```

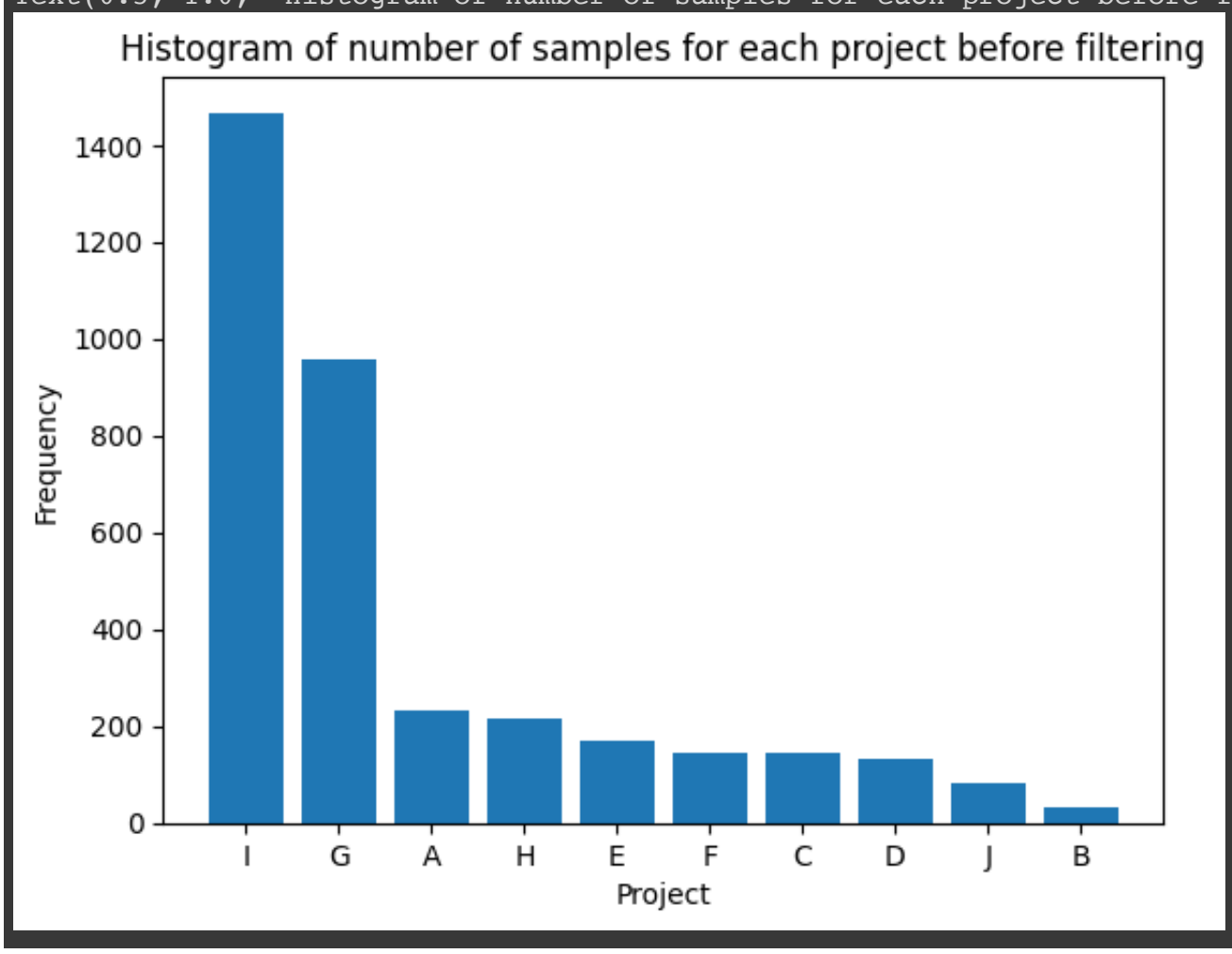


This histogram depicts the distribution of collected weeks for the provided samples. It is noteworthy that preterm births, defined as occurrences before week 37. Consequently, we will refine our dataset by retaining only those samples where the collection week is less than 37.

```
microbiome = pd.read_csv('taxonomy_relabd.family.csv')
filteredMetadata= metadata[metadata['collect_wk'] < 37].groupby('participant_id').apply(lambda group: group[group['co
```

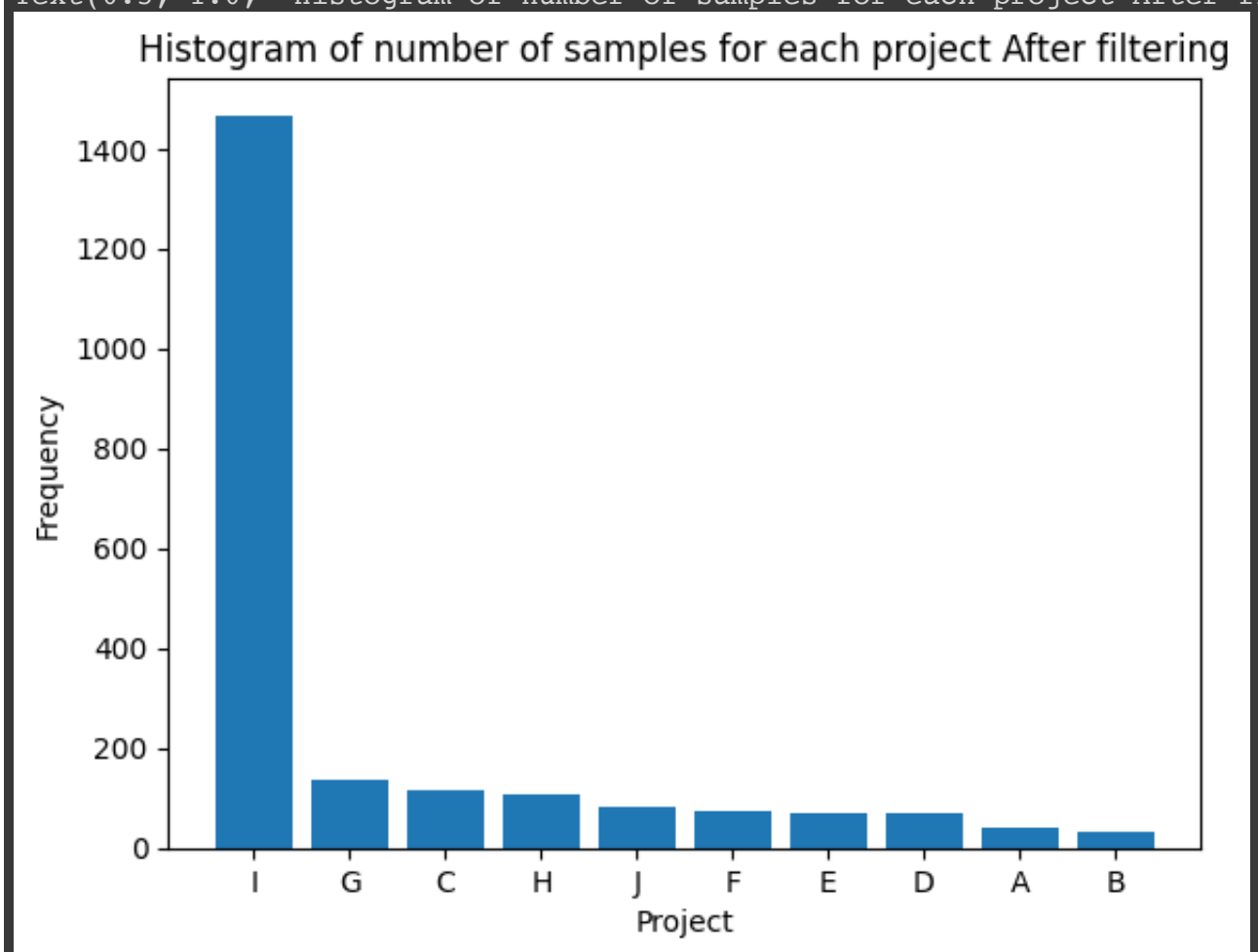
```
# visualization of projects: frequency of samples in each project after filtering:
plt.bar(metadata['project'].value_counts().index, metadata['project'].value_counts())
plt.xlabel('Project')
plt.ylabel('Frequency')
plt.title('Histogram of number of samples for each project before filtering')
```

```
Text(0.5, 1.0, 'Histogram of number of samples for each project before filtering')
```




```
plt.bar(filteredMetadata['project'].value_counts().index, filteredMetadata['project'].value_counts())
plt.xlabel('Project')
plt.ylabel('Frequency')
plt.title('Histogram of number of samples for each project After filtering')
```

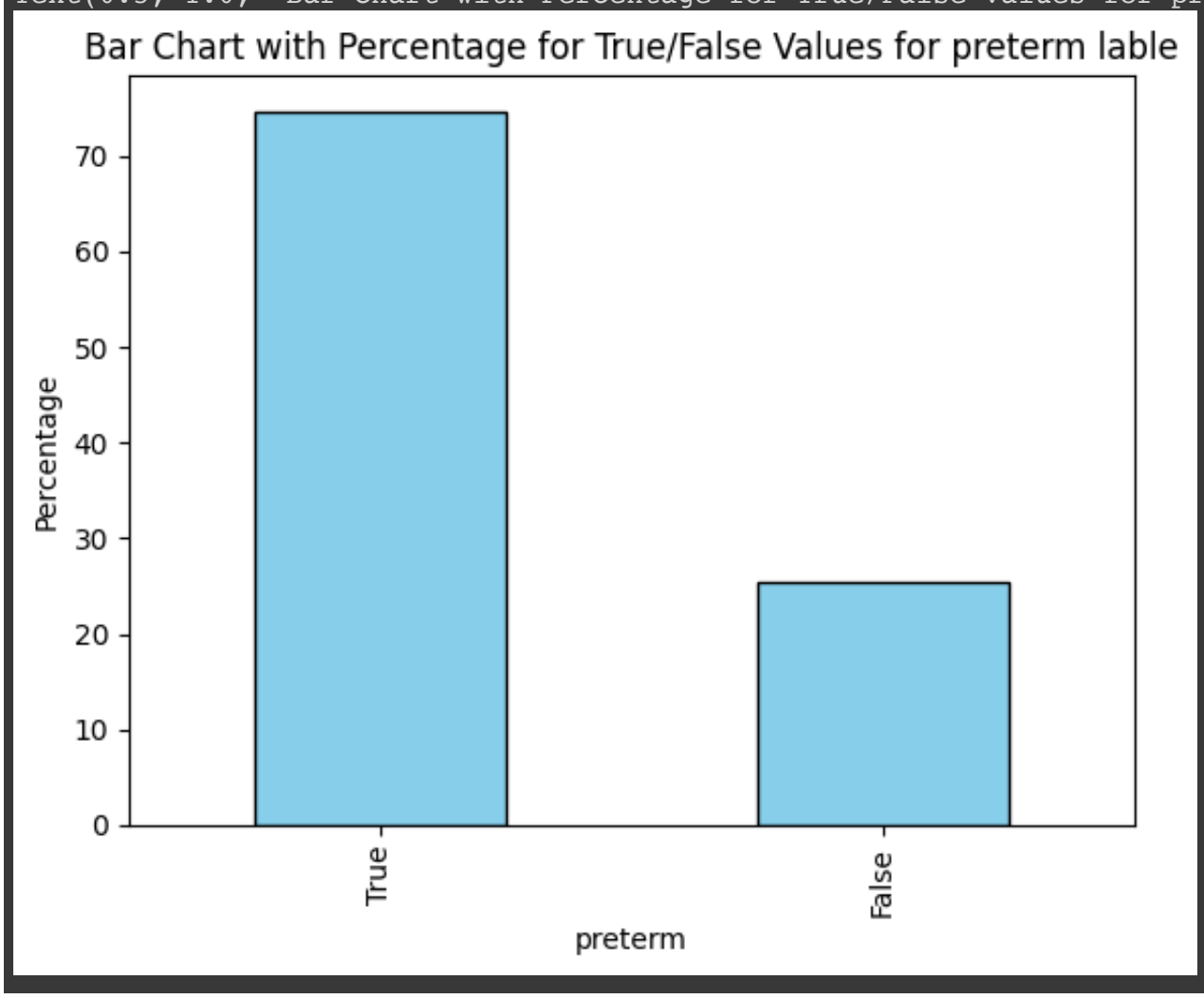
```
Text(0.5, 1.0, 'Histogram of number of samples for each project After filtering')
```



Out of the initial pool of 3,578 samples, 1,382 samples were filtered, resulting in a final dataset comprising 2,196 samples.

```
percentage_values = filteredMetadata['was_term'].value_counts(normalize=True) * 100

percentage_values.plot(kind='bar', color='skyblue', edgecolor='black')

# Set labels and title
plt.xlabel('preterm')
plt.ylabel('Percentage')
plt.title('Bar Chart with Percentage for True/False Values for preterm lable')
```

```
Text(0.5, 1.0, 'Bar Chart with Percentage for True/False Values for preterm lable')
```



As we can see the data is imbalance so we need to choose appropriate metrics like f1 score.

To train the machine learning model, it is essential to partition our dataset into distinct training and testing sets. In this phase, we distinguish the training and testing sets by utilizing genus-level as:

Train set: Projects C, E, G, I, J, and H

Test set: Projects A, B, D, and F.

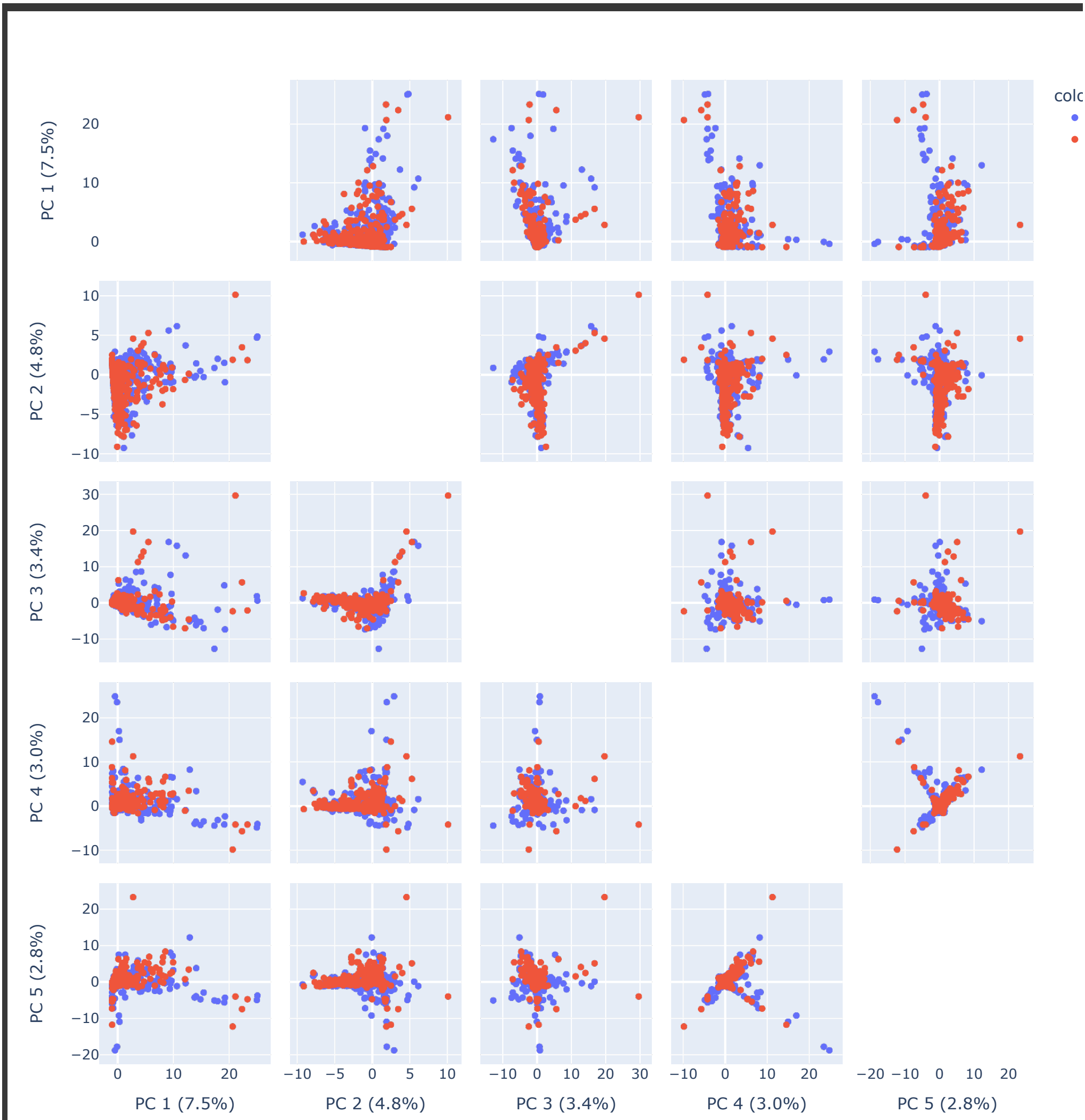
```
#seperate train and test specimen:
trainingSet= filteredMetadata[filteredMetadata['project'].isin(['C', 'E', 'G', 'I', 'J', 'H'])]
testingSet = filteredMetadata[filteredMetadata['project'].isin(['A', 'B', 'D', 'F'])]
```

```
from pandas.core.frame import DataFrame
#explain the data file the hierarchy the nread files and relabd file and why we used genus level
#load the genus level data and normalize the train and test data: explain the normalization procedure
microbiome = pd.read_csv('taxonomy_relabd.genus.csv')
specimenCol = microbiome[['specimen']]
microbiome = microbiome.iloc[:,1:]
#preprocessing: Remove microbes that are not present in any sample and the lowest 10 percent abundant microbes:
microbiome = microbiome.loc[:, (microbiome != 0).sum(axis=0) > 0]
# Remove features that are present in less than 10% of samples
threshold_percentage = 10
threshold_count = len(microbiome) * (threshold_percentage / 100)
microbiome = microbiome.loc[:, (microbiome != 0).sum(axis=0) >= threshold_count]
print(microbiome.shape)
microbiome = pd.concat([specimenCol, microbiome], axis=1)
print(microbiome.shape)
```

```
(3578, 69)
(3578, 70)
```

PCA visualization(done by Ricardo and Tina)

```
#PCA plot at genus level:
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.decomposition import PCA
scaler = StandardScaler()

filtered_preprocessed_microbiome = microbiome[microbiome['specimen'].isin(filteredMetadata['specimen'])]
mergedDf = pd.merge(filtered_preprocessed_microbiome, filteredMetadata, on='specimen', how='left').reset_index(drop=T
filteredLabel = mergedDf[['was_term']]

filtered_preprocessed_microbiome = filtered_preprocessed_microbiome.iloc[:,1:]

scaler.fit(filtered_preprocessed_microbiome)
# Step 2: Transform the training set
scaled_filtered_preprocessed_microbiome = scaler.transform(filtered_preprocessed_microbiome)

pca = PCA()
components = pca.fit_transform(scaled_filtered_preprocessed_microbiome)
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_[0:5] * 100)
}

fig = px.scatter_matrix(
    components,
    labels=labels,
    dimensions=range(5),
    color=np.array(filteredLabel['was_term'])
)

fig.update_traces(diagonal_visible=False)

# Set the size of the plot
fig.update_layout(
    width=1000,  # Set the width of the plot
    height=1000  # Set the height of the plot
)

fig.show()
```



The data doesn't exhibit two distinct clusters. Since PCA is a linear dimension reduction method, we can explore t-SNE and UMAP for the final presentation.

Rest of Machine Learning Model: (Done by Tina)

After filtering the data we kept 70 features

```
microbiomeTrain = microbiome[microbiome['specimen'].isin(trainingSet['specimen'])]
microbiomeTest = microbiome[microbiome['specimen'].isin(testingSet['specimen'])]
# restore the labels:
#train:
mergedDf = pd.merge(microbiomeTrain, trainingSet, on='specimen', how='left').reset_index(drop=True)
microbiomeTrain = microbiomeTrain.reset_index(drop=True)
lableTrain = mergedDf[['was_term']]
#test:
mergedDf = pd.merge(microbiomeTest, testingSet, on='specimen', how='left').reset_index(drop=True)
microbiomeTest = microbiomeTest.reset_index(drop=True)
lableTest = mergedDf[['was_term']]

print(microbiome.shape)
print(microbiomeTrain.shape)
print(microbiomeTest.shape)
print(lableTrain.shape)
print(lableTest.shape)
```

```
(3578, 70)
(1980, 70)
(216, 70)
(1980, 1)
(216, 1)
```

```
microbiomeTrain
```

	specimen	Achromobacter	Acinetobacter	Actinomyces	Actinomyces/Gleimia	Aerococcus	Anaerococcus	Arcanoba
0	C00001-03	0.000000	0.000000	0.0	0.0	0.000000	0.000000	
1	C00002-01	0.000000	0.000000	0.0	0.0	0.001799	0.000000	
2	C00003-02	0.000000	0.000427	0.0	0.0	0.000000	0.001706	
3	C00004-02	0.000000	0.000000	0.0	0.0	0.000000	0.000000	
4	C00005-01	0.002226	0.000000	0.0	0.0	0.010172	0.005000	
...	...	...	...	...	...	...	...	
1975	J00111-01	0.000000	0.000000	0.0	0.0	0.000000	0.000000	
1976	J00112-01	0.000000	0.000000	0.0	0.0	0.000000	0.000000	
1977	J00113-01	0.000000	0.000000	0.0	0.0	0.000000	0.000000	
1978	J00115-01	0.000000	0.000000	0.0	0.0	0.000000	0.000000	
1979	J00116-01	0.000000	0.000000	0.0	0.0	0.000000	0.000000	

1980 rows × 70 columns

```
microbiomeTest
```

	specimen	Achromobacter	Acinetobacter	Actinomyces	Actinomyces/Gleimia	Aerococcus	Anaerococcus	Arcanobac
0	A00001-05	0.0	0.0	0.000525	0.0	0.0	0.000525	
1	A00003-02	0.0	0.0	0.000000	0.0	0.0	0.000000	
2	A00004-18	0.0	0.0	0.000144	0.0	0.0	0.000144	
3	A00005-11	0.0	0.0	0.000000	0.0	0.0	0.000178	
4	A00006-03	0.0	0.0	0.000000	0.0	0.0	0.000000	
...	...	...	...	...	...	...	...	
211	F00070-01	0.0	0.0	0.000000	0.0	0.0	0.000000	
212	F00071-01	0.0	0.0	0.000000	0.0	0.0	0.000000	
213	F00072-01	0.0	0.0	0.000000	0.0	0.0	0.000000	
214	F00073-01	0.0	0.0	0.000000	0.0	0.0	0.000000	
215	F00074-01	0.0	0.0	0.000000	0.0	0.0	0.000000	

216 rows × 70 columns

training set size: (1980, 70) testing set size: (216, 70)

```
#standerd scaling the datasets: first apply to the train set then testing set
from sklearn.preprocessing import StandardScaler
#remove specimen:
microbiomeTrain = microbiomeTrain.iloc[:,1:]
microbiomeTest = microbiomeTest.iloc[:,1:]
scaler = StandardScaler()
scaler.fit(microbiomeTrain)
train_scaled = scaler.transform(microbiomeTrain)
test_scaled = scaler.transform(microbiomeTest)
```

```
np.array(lableTrain['was_term'])
```

```
array([False, False, False, ...,  True,  True,  True])
```

```
#train the model: SVM
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

import sklearn
def learn_SVM_classifier(X_train, y_train, kernel):
    clf = sklearn.svm.SVC(kernel=kernel)
    clf.fit(X_train, y_train)
    return clf


def evaluate_classifier(classifier, X_validation, y_validation):
    pred = classifier.predict(X_validation)
    accuracy = accuracy_score(y_validation, pred)
    precision = precision_score(y_validation, pred)
    recall = recall_score(y_validation, pred)
    f1 = f1_score(y_validation, pred)
    return accuracy, precision, recall, f1



def best_model_selection(kf, X, y):
    kernels =  ['linear', 'rbf', 'poly', 'sigmoid']
    performance = []
    for kernel in ['linear', 'rbf', 'poly', 'sigmoid']:
      accuracy = 0

      for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        classifier = learn_SVM_classifier(X_train, y_train, kernel)
        accuracy = evaluate_classifier(classifier, X, y)[0]+accuracy
      performance.append(accuracy/4)
    max_index = performance.index(max(performance))
    print(performance)
    return kernels[max_index]
```

```
#train_scaled, lableTrain, test_scaled, lableTest
kf = sklearn.model_selection.KFold(n_splits=4, random_state=1, shuffle=True)
best_kernel = best_model_selection(kf, train_scaled, np.array(lableTrain['was_term']))
best_kernel
```

```
[0.8017676767676767, 0.8122474747474747, 0.8094696969696971, 0.766540404040404]
'rbf'
```

The best kernel is rb. so we will set the kernel hyper parameter = rb

```
#Report train test accuracy:
classifier = learn_SVM_classifier(train_scaled, lableTrain, 'rbf')
trainAccuracy, trainPrecision, trainRecall, trainF1 = evaluate_classifier(classifier, train_scaled, np.array(lableTra
testAccuracy, testPrecision, testRecall, testF1= evaluate_classifier(classifier, test_scaled, np.array(lableTest))
print("Training result:")
print("accuracy:", trainAccuracy, "precision:", trainPrecision, "recall:", trainRecall, "F1 score:", trainF1)
print("Testing result:")
print("accuracy:", testAccuracy, "precision:", testPrecision, "recall:", testRecall, "F1 score:", testF1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex

Training result:
accuracy: 0.8222222222222222 precision: 0.8131041890440387 recall: 0.997364953886693 F1 score: 0.895857988165680
Testing result:
accuracy: 0.5787037037037037 precision: 0.5721153846153846 recall: 0.9834710743801653 F1 score: 0.7234042553191
```

The reported accuracy for both the training and test sets is provided. Given the data imbalance, especially in the medical context, it is crucial to consider additional metrics. The inclusion of F1 score, precision, and recall offers a more comprehensive evaluation. Notably, the SVM model with an RBF kernel achieved an F1 score of 0.72 on the test set, demonstrating a satisfactory performance level.

```
# RF model:
import sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def learn_RF_classifier(X_train, y_train, n_estimators):
    clf = RandomForestClassifier(n_estimators=n_estimators)
    clf.fit(X_train, y_train)
    return clf

def evaluate_RF_classifier(classifier, X_validation, y_validation):
    pred = classifier.predict(X_validation)
    accuracy = accuracy_score(y_validation, pred)
    precision = precision_score(y_validation, pred)
    recall = recall_score(y_validation, pred)
    f1 = f1_score(y_validation, pred)
    return accuracy, precision, recall, f1

def best_RF_model_selection(kf, X, y):
    n_estimators_values = [10, 50, 100, 200, 500]
    performance = []

    for n_estimators in n_estimators_values:
        accuracy = 0
        for train_index, test_index in kf.split(X):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]
            classifier = learn_RF_classifier(X_train, y_train, n_estimators)
            accuracy += evaluate_classifier(classifier, X, y)[0]

        performance.append(accuracy / 4)

    max_index = performance.index(max(performance))
    print(performance)
    return n_estimators_values[max_index]
```

```
kf = sklearn.model_selection.KFold(n_splits=4, random_state=1, shuffle=True)
n_estimator = best_RF_model_selection(kf, train_scaled, np.array(lableTrain['was_term']))
n_estimator
```

```
[0.9246212121212121, 0.9376262626262626, 0.9387626262626263, 0.9388888888888889, 0.9398989898989899]
500
```

The hyperparameter n_estimator denotes the number of trees in the forest. Through our exploration, we considered various values of 10, 50, 100, 200, and 500 for this parameter. Notably, the value of 500 yielded the highest accuracy, establishing it as the optimal choice for the number of trees in the forest.

```
classifier = learn_RF_classifier(train_scaled, lableTrain, 50)
trainAccuracy, trainPrecision, trainRecall, trainF1 = evaluate_RF_classifier(classifier, train_scaled, np.array(lable
testAccuracy, testPrecision, testRecall, testF1= evaluate_RF_classifier(classifier, test_scaled, np.array(lableTest))
print("Training result:")
print("accuracy:", trainAccuracy, "precision:", trainPrecision, "recall:", trainRecall, "F1 score:", trainF1)
print("Testing result:")
print("accuracy:", testAccuracy, "precision:", testPrecision, "recall:", testRecall, "F1 score:", testF1)
```

```
<ipython-input-40-32c43da5d18e>:9: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for exa

Training result:
accuracy: 0.9863636363636363 precision: 0.9825242718446602 recall: 1.0 F1 score: 0.9911851126346718
Testing result:
accuracy: 0.4537037037037037 precision: 0.5130434782608696 recall: 0.48760330578512395 F1 score: 0.5
```

Clearly, SVM outperformed RF in terms of performance.

Reflection

Reflecting on our work up to this point, the most challenging aspect has been working through the complex microbiome data to identify features that could predict preterm birth. Turning this large volume of data into useful insights has been a considerable challenge. Our early analysis has shown that the link between the vaginal microbiome and preterm birth is subtle. We've seen that factors like maternal age do have some connection with preterm births, but these relationships are not strong enough to act as standalone indicators.

Currently, we can report a statistically significant, connection between maternal age and preterm birth. The usefulness of this result for prediction is limited, highlighting the need for an approach that considers multiple aspects to understand the risks of preterm birth. Our main obstacles now are improving the accuracy of our predictive models and dealing with any imbalance in our data which could skew our findings.

Regarding the direction of our project, we believe we are generally moving forward correctly but also understand that more effort is needed in processing our data and refining our models. With the data's complexity, it's crucial that the models we create are both accurate in their predictions and clear in their explanations. This might mean we need to spend more time understanding the biological importance of the features we're studying.

Continuing with our project seems justified; the preliminary findings and the chance to discover patterns that could lead to interventions for preterm birth are compelling reasons to keep going. Looking ahead, we plan to use more advanced predictive models like neural networks and re-examine our approach to selecting features. We aim to build a model that more effectively captures the detailed nature of our high dimensional data, with the ultimate aim of offering predictions that are backed by a deeper understanding of the biological aspects of preterm birth.