PREPROCESSING_REPORT

Importing Required Packages

In [ ]:
```
# NumPy is a powerful numerical computing library in Python
import numpy as np
# Pandas is a powerful data manipulation and analysis library for Python
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading Codon_Usage Dataset About the Dataset: The Codon_usage dataset is a collection of information regarding the frequency of occurrence of different codons within the genetic code of various organisms.

In [ ]:
```
# Read the Codon_usage dataset into a pandas DataFrame object
# Setting low_memory=False because it disables the memory optimization and makes pandas read the entire da
df1 = pd.read_csv('Group_17_Raw_Data.csv', low_memory=False)
```

Part: 1 - Exploratory Data Analysis

In [ ]:
```
# Displaying the first few rows of a DataFrame
df1.head()
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG	CUU	...	CGG	
0	vrl	0	100217	1995	Epizootic haematopoietic necrosis virus	0.01654	0.01203	0.00050	0.00351	0.01203	...	0.00451	
1	vrl	0	100220	1474	Bohle iridovirus	0.02714	0.01357	0.00068	0.00678	0.00407	...	0.00136	0
2	vrl	0	100755	4862	Sweet potato leaf curl virus	0.01974	0.0218	0.01357	0.01543	0.00782	...	0.00596	
3	vrl	0	100880	1915	Northern cereal mosaic virus	0.01775	0.02245	0.01619	0.00992	0.01567	...	0.00366	
4	vrl	0	100887	22831	Soil-borne cereal mosaic virus	0.02816	0.01371	0.00767	0.03679	0.01380	...	0.00604	0

5 rows × 69 columns

In [ ]:
```
# Displaying the last few rows of a DataFrame
df1.tail()
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG	CUU	...	C
13023	pri	0	9601	1097	Pongo pygmaeus abelii	0.02552	0.03555	0.00547	0.01367	0.01276	...	0.008
13024	pri	1	9601	2067	mitochondrion Pongo abelii mitochondrion	0.01258	0.03193	0.01984	0.00629	0.01451	...	0.001
13025	pri	1	9602	1686	Pongo pygmaeus pygmaeus	0.01423	0.03321	0.01661	0.00356	0.01127	...	0.000
13026	pri	0	9606	40662582	Homo sapiens	0.01757	0.02028	0.00767	0.01293	0.01319	...	0.011
13027	pri	1	9606	8998998	mitochondrion Homo sapiens	0.01778	0.03724	0.01732	0.00600	0.01689	...	0.000

5 rows × 69 columns

In [ ]:
```
# Dimensions of a DataFrame
df1.shape
```

Out[ ]: (13028, 69)

In [ ]:
```
# Displaying list of column names of df1
print(df1.columns.to_list())
```

```
['Kingdom', 'DNAtype', 'SpeciesID', 'Ncodons', 'SpeciesName', 'UUU', 'UUC', 'UUA', 'UUG', 'CUU', 'CUC', 'CU
A', 'CUG', 'AUU', 'AUC', 'AUA', 'AUG', 'GUU', 'GUC', 'GUA', 'GUG', 'GCU', 'GCC', 'GCA', 'GCG', 'CCU', 'CC
C', 'CCA', 'CCG', 'UGG', 'GGU', 'GGC', 'GGA', 'GGG', 'UCU', 'UCC', 'UCA', 'UCG', 'AGU', 'AGC', 'ACU', 'AC
C', 'ACA', 'ACG', 'UAU', 'UAC', 'CAA', 'CAG', 'AAU', 'AAC', 'UGU', 'UGC', 'CAU', 'CAC', 'AAA', 'AAG', 'CG
U', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG', 'GAU', 'GAC', 'GAA', 'GAG', 'UAA', 'UAG', 'UGA']
```

In [ ]:
```
# Displaying summary of a DataFrame
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13028 entries, 0 to 13027
Data columns (total 69 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Kingdom      13028 non-null  object
 1   DNAtype      13028 non-null  int64
 2   SpeciesID    13028 non-null  int64
 3   Ncodons      13028 non-null  int64
 4   SpeciesName  13028 non-null  object
 5   UUU          13028 non-null  object
 6   UUC          13028 non-null  object
 7   UUA          13028 non-null  float64
 8   UUG          13028 non-null  float64
 9   CUU          13028 non-null  float64
 10  CUC          13028 non-null  float64
 11  CUA          13028 non-null  float64
 12  CUG          13028 non-null  float64
 13  AUU          13028 non-null  float64
 14  AUC          13028 non-null  float64
 15  AUA          13028 non-null  float64
 16  AUG          13028 non-null  float64
 17  GUU          13028 non-null  float64
 18  GUC          13028 non-null  float64
 19  GUA          13028 non-null  float64
 20  GUG          13028 non-null  float64
 21  GCU          13028 non-null  float64
 22  GCC          13028 non-null  float64
 23  GCA          13028 non-null  float64
 24  GCG          13028 non-null  float64
 25  CCU          13028 non-null  float64
 26  CCC          13028 non-null  float64
 27  CCA          13028 non-null  float64
 28  CCG          13028 non-null  float64
 29  UGG          13028 non-null  float64
 30  GGU          13028 non-null  float64
 31  GGC          13028 non-null  float64
 32  GGA          13028 non-null  float64
 33  GGG          13028 non-null  float64
 34  UCU          13028 non-null  float64
 35  UCC          13028 non-null  float64
 36  UCA          13028 non-null  float64
```

```
 37  UCG             13028  non-null  float64
 38  AGU             13028  non-null  float64
 39  AGC             13028  non-null  float64
 40  ACU             13028  non-null  float64
 41  ACC             13028  non-null  float64
 42  ACA             13028  non-null  float64
 43  ACG             13028  non-null  float64
 44  UAU             13028  non-null  float64
 45  UAC             13028  non-null  float64
 46  CAA             13028  non-null  float64
 47  CAG             13028  non-null  float64
 48  AAU             13028  non-null  float64
 49  AAC             13028  non-null  float64
 50  UGU             13028  non-null  float64
 51  UGC             13028  non-null  float64
 52  CAU             13028  non-null  float64
 53  CAC             13028  non-null  float64
 54  AAA             13028  non-null  float64
 55  AAG             13028  non-null  float64
 56  CGU             13028  non-null  float64
 57  CGC             13028  non-null  float64
 58  CGA             13028  non-null  float64
 59  CGG             13028  non-null  float64
 60  AGA             13028  non-null  float64
 61  AGG             13028  non-null  float64
 62  GAU             13028  non-null  float64
 63  GAC             13028  non-null  float64
 64  GAA             13028  non-null  float64
 65  GAG             13028  non-null  float64
 66  UAA             13028  non-null  float64
 67  UAG             13028  non-null  float64
 68  UGA             13028  non-null  float64
dtypes: float64(62), int64(3), object(4)
memory usage: 6.9+ MB
```

In [ ]:
```
# Displaying descriptive statistics summary
df1.describe()
```

Out[ ]:

	DNAtype	SpeciesID	Ncodons	UUA	UUG	CUU	CUC	
count	13028.000000	13028.000000	1.302800e+04	13028.000000	13028.000000	13028.000000	13028.000000	13028.0
mean	0.367209	130451.105926	7.960576e+04	0.020637	0.014104	0.017820	0.018288	0.0
std	0.688726	124787.086107	7.197010e+05	0.020709	0.009280	0.010586	0.014572	0.0
min	0.000000	7.000000	1.000000e+03	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	28850.750000	1.602000e+03	0.005610	0.007108	0.010890	0.007830	0.0
50%	0.000000	81971.500000	2.927500e+03	0.015260	0.013360	0.016130	0.014560	0.0
75%	1.000000	222891.250000	9.120000e+03	0.029485	0.019810	0.022730	0.025112	0.0
max	12.000000	465364.000000	4.066258e+07	0.151330	0.101190	0.089780	0.100350	0.1

8 rows × 65 columns

In [ ]:
```
# Identify non-numeric columns
non_numeric_cols = df1.select_dtypes(exclude=['number']).columns

# Handle non-numeric columns (example for categorical data)
for col in non_numeric_cols:
    df1[col] = df1[col].astype('category').cat.codes
```

In [ ]:
```
# computing pairwise correlation of columns
Cor = df1.corr();
Cor
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG
Kingdom	1.000000	0.169479	0.058942	-0.075663	0.193193	0.081105	0.155345	-0.029766	-0.071014
DNAtype	0.169479	1.000000	-0.020671	-0.053869	0.624801	0.427581	0.180234	0.460140	-0.039368
SpeciesID	0.058942	-0.020671	1.000000	0.051598	0.031807	0.036512	-0.026193	0.025723	-0.047561
Ncodons	-0.075663	-0.053869	0.051598	1.000000	-0.051957	-0.035572	-0.025966	-0.033718	0.009483
SpeciesName	0.193193	0.624801	0.031807	-0.051957	1.000000	0.291329	0.286301	0.408656	-0.225587
...	...	...	...	...	...	...	...	...	...
GAA	-0.263315	-0.100670	-0.014164	0.028578	-0.265031	0.264775	-0.483575	0.188387	0.315423
GAG	-0.194438	-0.497575	-0.079152	0.053022	-0.516202	-0.510980	-0.035401	-0.577686	0.115329
UAA	0.049586	0.122864	0.048676	-0.026197	0.191300	0.171500	-0.027586	0.256173	-0.126004
UAG	0.016156	-0.029795	-0.008848	0.003334	0.010939	0.001805	-0.042075	0.009338	0.054169
UGA	0.261019	0.458572	0.050192	-0.047956	0.662834	0.142985	0.414407	0.331463	-0.468188

69 rows × 69 columns

Converting categorical features to categorical variable

In [ ]:
```
# Converts the 'Kingdom' column in df1 into a categorical variable.
df1["Kingdom"] = df1["Kingdom"].astype('category')
# Creates a new column 'Kingdom_cat' in df1 containing the category codes corresponding to the 'Kingdom' c
df1["Kingdom_cat"] = df1["Kingdom"].cat.codes
df1.head()
df2 = df1.loc[:,df1.columns[6:]]
```

In [ ]:
```
# Displaying the first few rows of a DataFrame
df2.head()
```

Out[ ]:

	UUC	UUA	UUG	CUU	CUC	CUA	CUG	AUU	AUC	AUA	...	AGA	AGG	GA
0	789	0.00050	0.00351	0.01203	0.03208	0.00100	0.04010	0.00551	0.02005	0.00752	...	0.01303	0.03559	0.010
1	938	0.00068	0.00678	0.00407	0.02849	0.00204	0.04410	0.01153	0.02510	0.00882	...	0.01696	0.03596	0.012
2	1750	0.01357	0.01543	0.00782	0.01111	0.01028	0.01193	0.02283	0.01604	0.01316	...	0.01974	0.02489	0.031
3	1815	0.01619	0.00992	0.01567	0.01358	0.00940	0.01723	0.02402	0.02245	0.02507	...	0.01410	0.01671	0.037
4	952	0.00767	0.03679	0.01380	0.00548	0.00473	0.02076	0.02716	0.00867	0.01310	...	0.01494	0.01734	0.041

5 rows × 64 columns

Handling Outliers

In [ ]:
```
# Define a function to handle outliers using IQR
def handle_outliers(df):
    # Compute the first quartile (Q1) and third quartile (Q3)
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    # Compute the interquartile range (IQR)
    IQR = Q3 - Q1
    # Define the outlier threshold
    outlier_threshold = 1.5
    # Identify outliers
    outliers = (df < (Q1 - outlier_threshold * IQR)) | (df > (Q3 + outlier_threshold * IQR))
    # Remove outliers
    df_no_outliers = df[~outliers.any(axis=1)]
    return df_no_outliers
```

In [ ]:
```
# Select only numerical columns for outlier handling
numerical_columns = df1.select_dtypes(include='number')
# Handle outliers for numerical columns
codon_df_no_outliers = handle_outliers(numerical_columns)
# Display the shape of the original and outlier-handled DataFrames
print("Original DataFrame shape:", df1.shape)
print("DataFrame shape after removing outliers:", codon_df_no_outliers.shape)
```

```
Original DataFrame shape: (13028, 70)
DataFrame shape after removing outliers: (3744, 69)
```

Observation: The original DataFrame shape is (13028, 69). After Handling Outliers using IQR, the DataFrame shape is (3744, 65).

DataPreprocessingandDatacleaning

In [ ]:
```
# Printing null values in our dataset
df1.isna()
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG	CUU	...	AGA	AGG	GAU	
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
13023	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
13024	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
13025	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
13026	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F
13027	False	False	False	False	False	False	False	False	False	False	...	False	False	False	F

13028 rows × 70 columns

In [ ]:
```
# Check for missing values
null_counts = df1.isnull().sum()
# Print the null value counts for each column
print(null_counts)
```

```
Kingdom          0
DNAtype          0
SpeciesID        0
Ncodons          0
SpeciesName      0
                --
GAG              0
UAA              0
UAG              0
UGA              0
Kingdom_cat      0
Length: 70, dtype: int64
```

In [ ]:
```
# Displaying notnull columns count
notnull_count = df1.notnull().any().sum()
print('Number of notnull columns:',notnull_count)
```

```
Number of notnull columns: 70
```

Observation: We have used isnull(), isna() and notnull() function to find null or missing values, the results shows that there are no null or missing values in our dataset. notnull() function helps us to find the number of notnull rows, we got 69 as output and we have 69 columns. so there are no missing values in our dataset.

In [ ]:
```
# Check for duplicates and drop them if necessary
print("\nNumber of duplicate rows:", df1.duplicated().sum())
df1.drop_duplicates(inplace=True)
```

```
Number of duplicate rows: 0
```

Part: 2 - Exploratory Data Analysis

In [ ]:
```
# Modifing and adding new columns to df1a
df1a = df1
K_names = {'arc': 'archaea',
'bct': ' bacteria',
'phg' : 'bacteriophage',
'plm': 'plasmid',
'pln': 'plant',
'inv': 'invertebrate',
'vrt': 'vertebrate',
'mam': 'mammal',
```

```
'rod': 'rodent',
'pri':  'primate',
'vrl':  'virus'}
K_names2 = {0: 'archaea',
1:  ' bacteria',
2  : 'bacteriophage',
3: 'plasmid',
4: 'plant',
5: 'invertebrate',
6: 'vertebrate',
7: 'mammal',
8: 'rodent',
9: 'primate',
10: 'virus'}
D_names = {0:'genomic',
1:'mitochondrial',
2:'chloroplast',
3:'cyanelle',
4:'plastid',
5:'nucleomorph',
6:'secondary_endosymbiont',
7:'chromoplast', '8':'leucoplast',
9:'NA',
10:'proplastid',
11:'apicoplast',
12:'kinetoplast'}
# Replacing Kingdom_Names to Kingdom
df1b = df1a.replace({"Kingdom": K_names})
df1a['Kingdom_Names'] = df1b['Kingdom']
# Replacing DNAtype_Names to DNAtype
df1b = df1a.replace({"DNAtype": D_names})
df1a['DNAtype_Names'] = df1b['DNAtype']
```

In [ ]:

```
# Displaying the first few rows of a DataFrame
df1a.head()
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG	CUU	...	GAU	GAC
0	9	0	100217	1995	2827	1437	789	0.00050	0.00351	0.01203	...	0.01003	0.04612
1	9	0	100220	1474	1087	2476	938	0.00068	0.00678	0.00407	...	0.01221	0.04545
2	9	0	100755	4862	8252	1752	1750	0.01357	0.01543	0.00782	...	0.03126	0.02036
3	9	0	100880	1915	5480	1556	1815	0.01619	0.00992	0.01567	...	0.03760	0.01932
4	9	0	100887	22831	7577	2572	952	0.00767	0.03679	0.01380	...	0.04148	0.02483

5 rows × 72 columns

In [ ]:
```
# Displaying the first few rows of a DataFrame
df1b.head()
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG	CUU	...	AGG	GAU
0	9	genomic	100217	1995	2827	1437	789	0.00050	0.00351	0.01203	...	0.03559	0.01003
1	9	genomic	100220	1474	1087	2476	938	0.00068	0.00678	0.00407	...	0.03596	0.01221
2	9	genomic	100755	4862	8252	1752	1750	0.01357	0.01543	0.00782	...	0.02489	0.03126
3	9	genomic	100880	1915	5480	1556	1815	0.01619	0.00992	0.01567	...	0.01671	0.03760
4	9	genomic	100887	22831	7577	2572	952	0.00767	0.03679	0.01380	...	0.01734	0.04148

5 rows × 71 columns

Barplot(Toshowthekingdomfrequency)

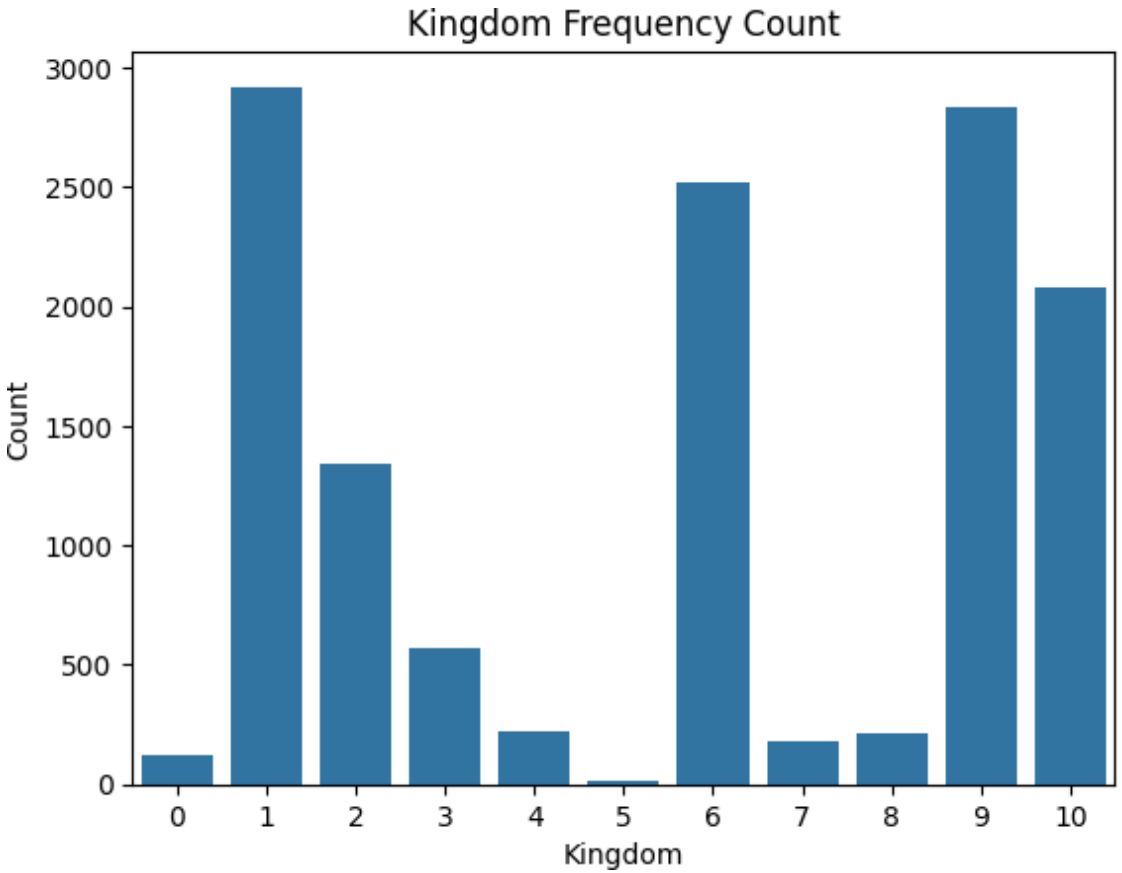
In [ ]:
```
# Get the frequency counts of 'Kingdom_Names'
Kingdom_counts = df1a['Kingdom_Names'].value_counts()

# Create the barplot
ax = sns.barplot(x=Kingdom_counts.index, y=Kingdom_counts.values)

# Set labels and title
ax.set(xlabel='Kingdom', ylabel='Count', title='Kingdom Frequency Count')
```

```
plt.show()

# displays the frequency count of each unique value in the 'Kingdom_Names' column of the original DataFram
df1['Kingdom_Names'].value_counts()
```



Out[ ]:

Kingdom_Names	count
1	2920
9	2832
6	2523
10	2077
2	1345
3	572
4	220
8	215
7	180
0	126
5	18

dtype: int64

In [ ]:

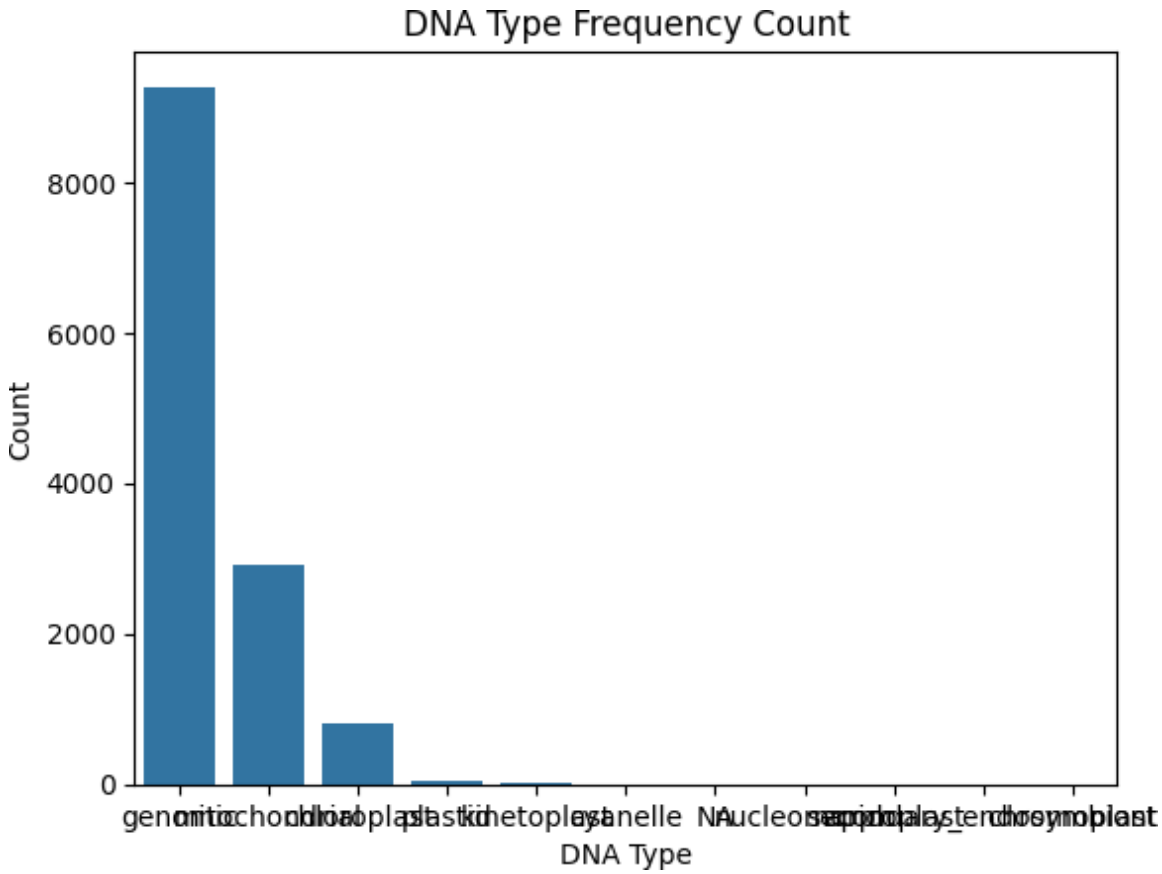
```
# Get the frequency counts of 'DNAtype_Names'
dna_type_counts = df1a['DNAtype_Names'].value_counts()

# Create the barplot
ax = sns.barplot(x=dna_type_counts.index, y=dna_type_counts.values)

# Set labels and title
ax.set(xlabel='DNA Type', ylabel='Count', title='DNA Type Frequency Count')

plt.show()

# displays the frequency count of each unique value in the 'DNAtype_Names' column of the original DataFram
df1['DNAtype_Names'].value_counts()
```


DNA Type Frequency Count
Count
8000
6000
4000
2000
0
DNA Type

Out[ ]:

DNAtype_Names	count
genomic	9267
mitochondrial	2899
chloroplast	816
plastid	31
kinetoplast	5
cyanelle	2
NA	2
nucleomorph	2
apicoplast	2
secondary_endosymbiont	1
chromoplast	1

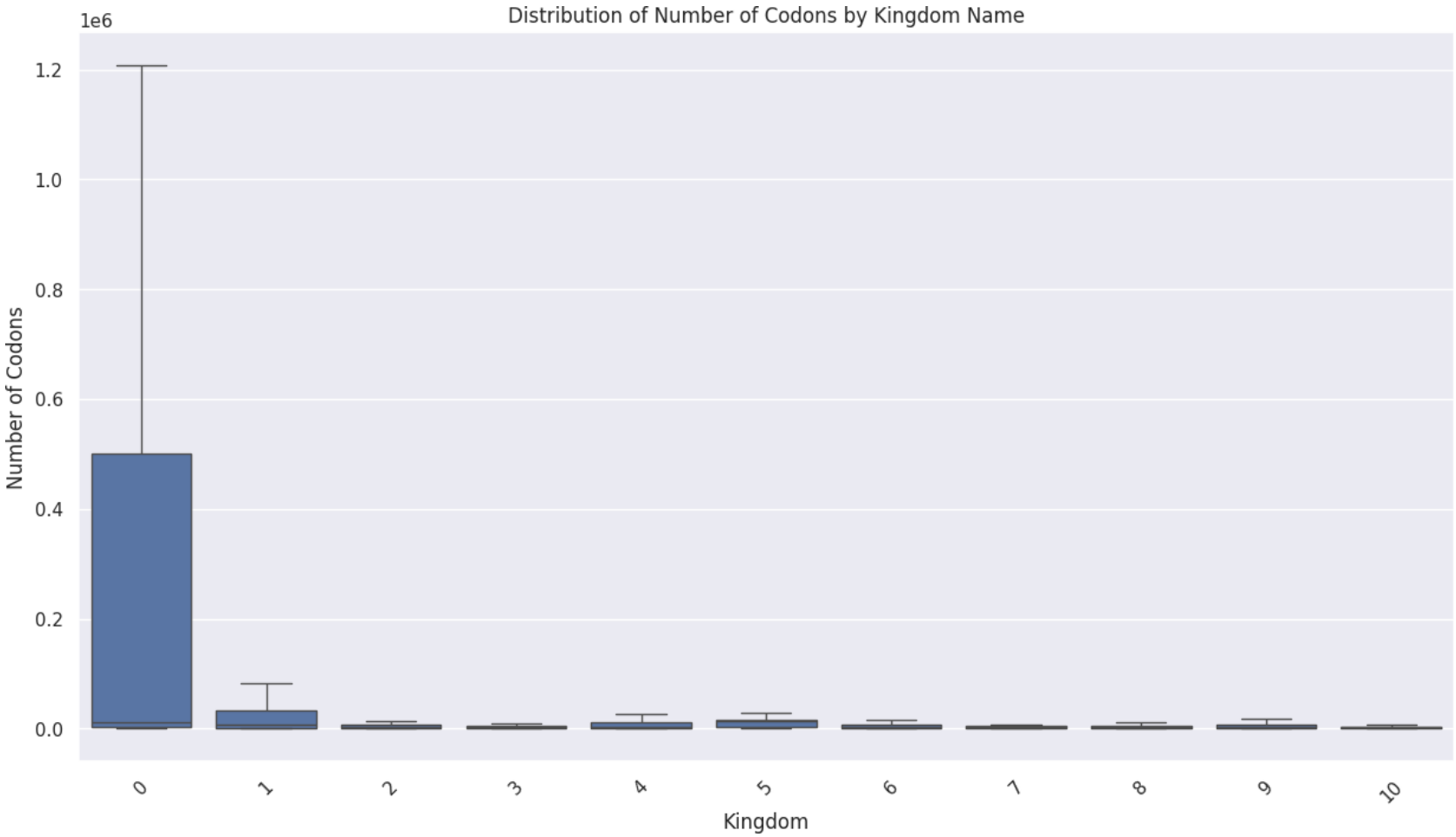
dtype: int64

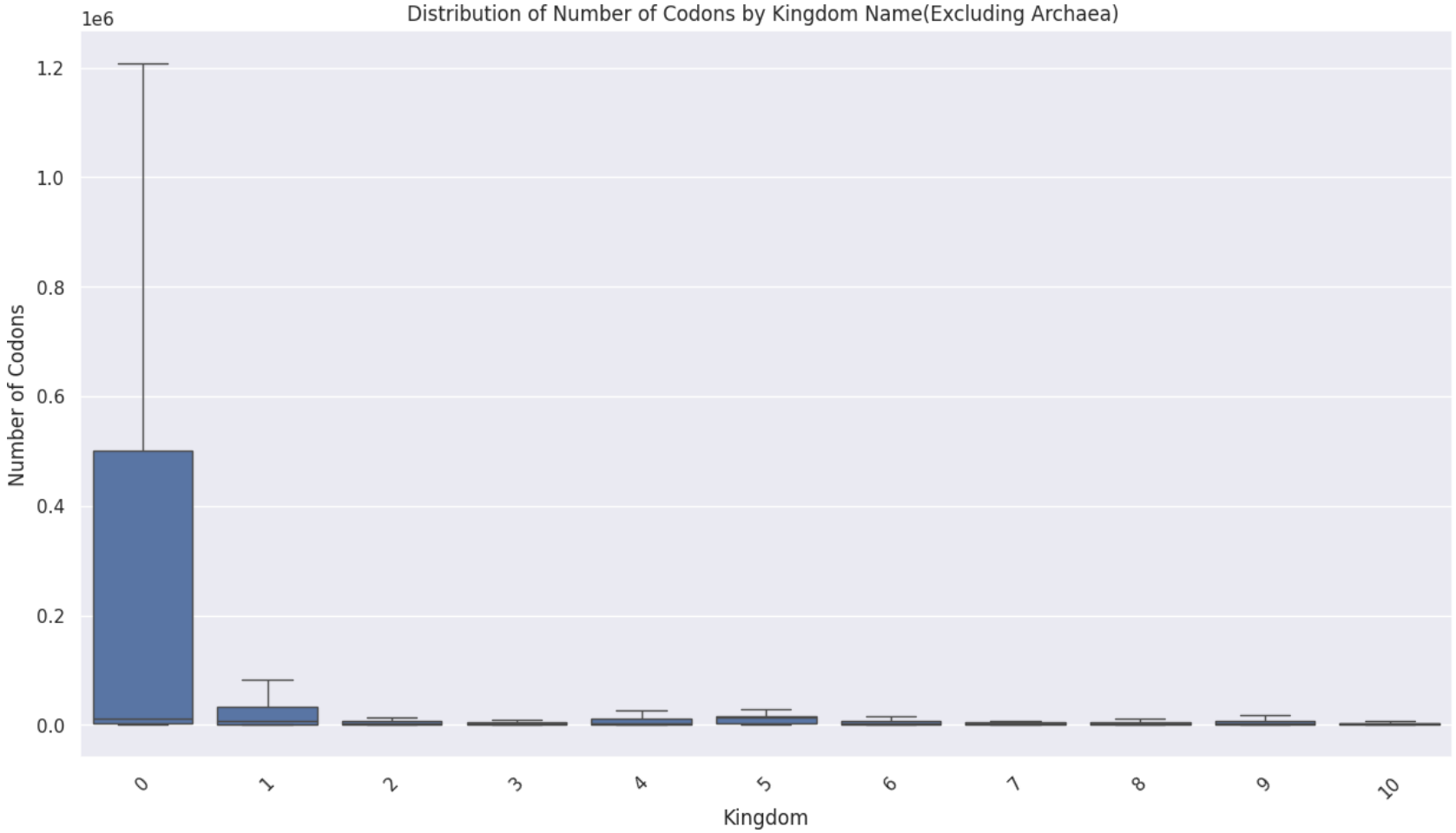
Boxplot (Distribution of No.of Codons by Kingdom Name)

In [ ]:

```
# setting the size of the figures generated by Seaborn to be 15 inches in width and 8 inches in height.
sns.set(rc  =  {'figure.figsize':(15,8)})
# first boxplot displays the distribution of the number of codons for all kingdoms
ax = sns.boxplot(x="Kingdom_Names", y="Ncodons",data=df1a,
                 showfliers = False)
ax.set(xlabel='Kingdom', ylabel='Number of Codons',
title = 'Distribution of Number of Codons by Kingdom Name')
plt.xticks(rotation=45)
plt.show()
# second boxplot excludes the kingdom "archaea" from the plot
ax = sns.boxplot(x="Kingdom_Names", y="Ncodons",
data=df1a[df1a['Kingdom_Names']  !=  'archaea']
,showfliers = False)
```

```
ax.set(xlabel='Kingdom', ylabel='Number of Codons',
title = 'Distribution of Number of Codons by Kingdom Name(Excluding Archaea) ')
plt.xticks(rotation=45)
plt.show()
```

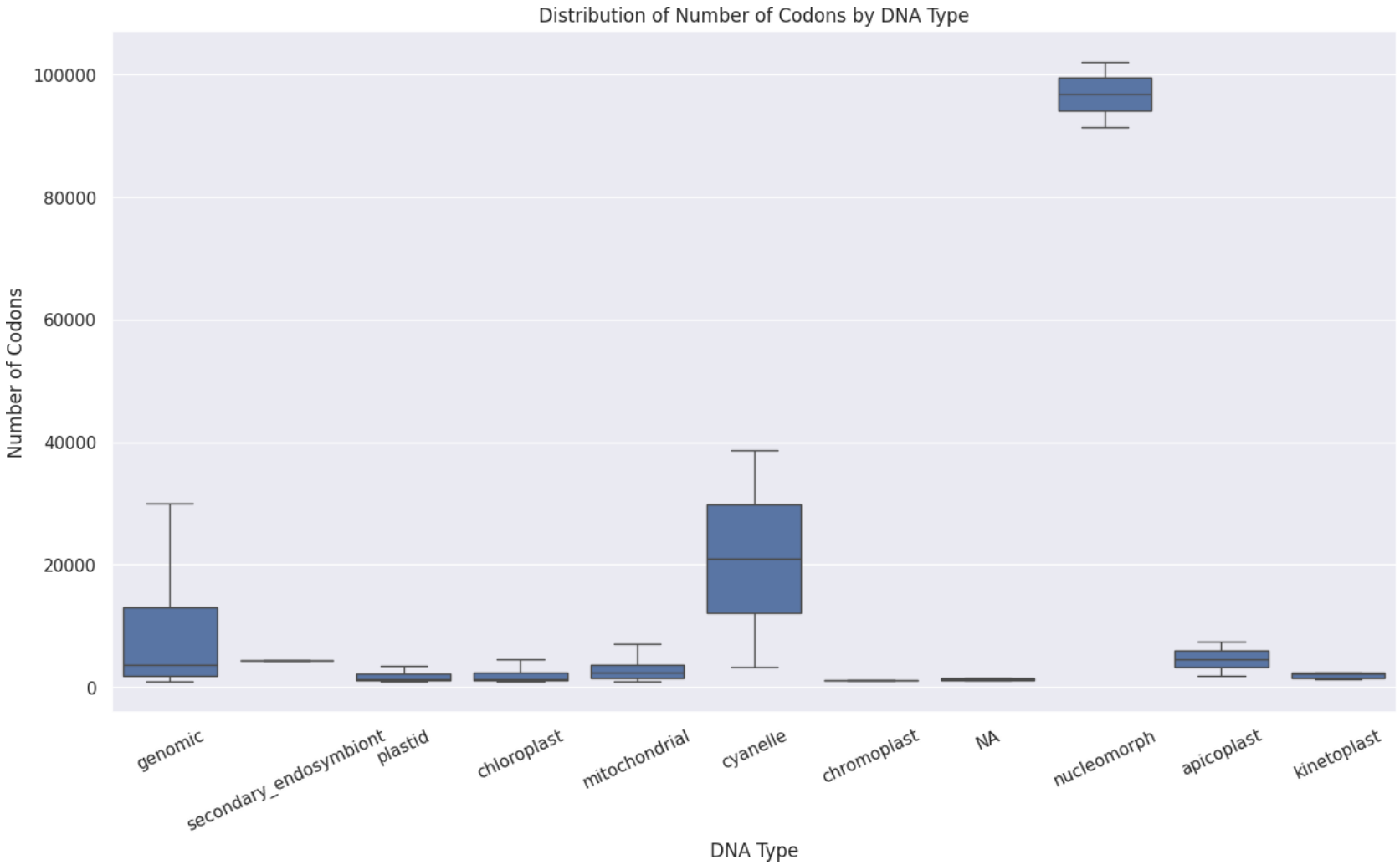


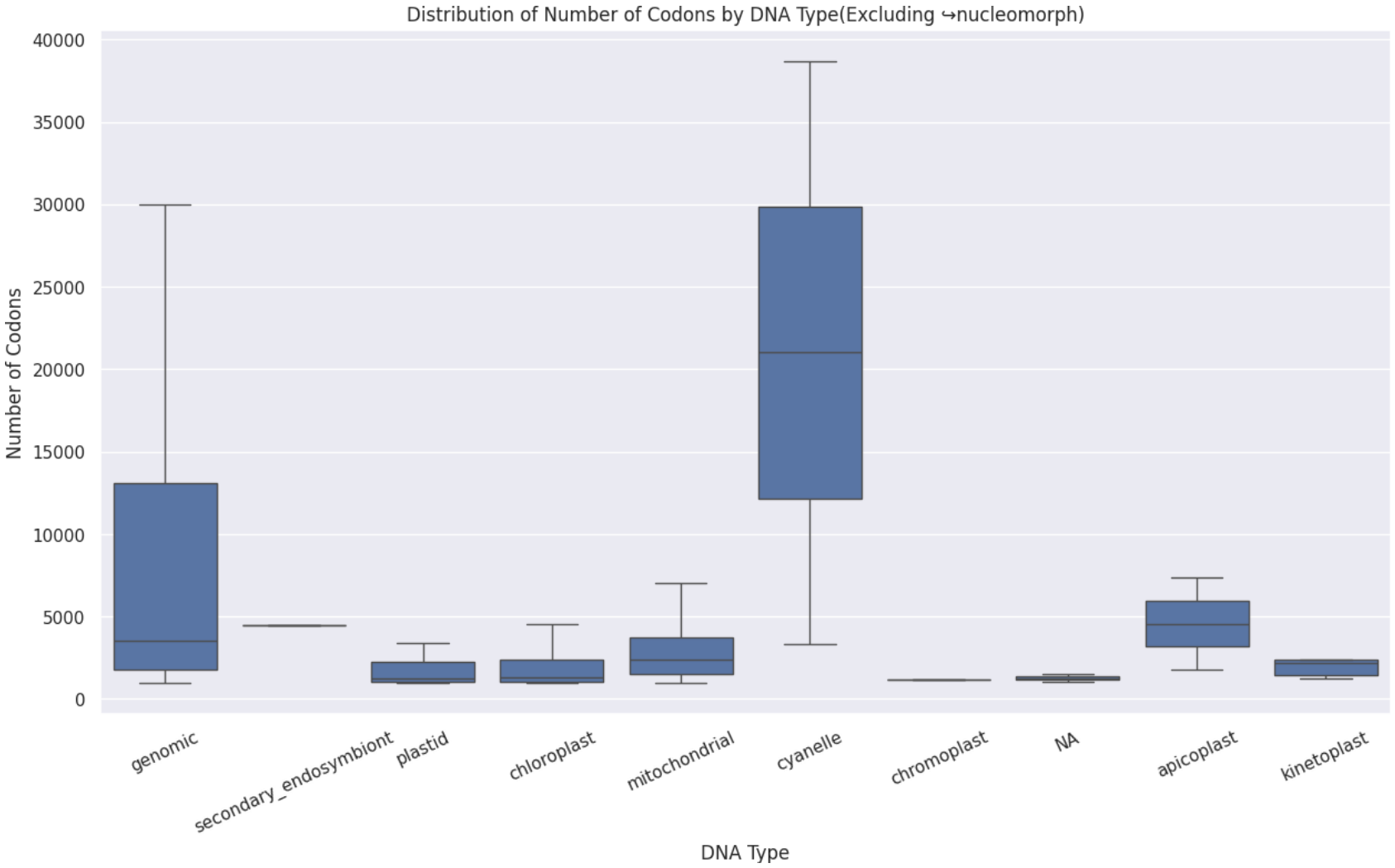


In [ ]:

```
sns.set(rc = {'figure.figsize':(15,8)})
# The first boxplot displays the distribution of the number of codons for all DNA types
ax = sns.boxplot(x="DNAtype_Names", y="Ncodons",
data=df1a,showfliers = False)
ax.set(xlabel='DNA Type', ylabel='Number of Codons',
title = 'Distribution of Number of Codons by DNA Type')
plt.xticks(rotation=25)
plt.show()
# The second boxplot excludes the DNA type "nucleomorph" from the plot
ax = sns.boxplot(x="DNAtype_Names", y="Ncodons",
data=df1a[df1a['DNAtype_Names'] != 'nucleomorph']
,showfliers = False)
```

```
ax.set(xlabel='DNA Type', ylabel='Number of Codons',
title = 'Distribution of Number of Codons by DNA Type(Excluding ↪nucleomorph)')
plt.xticks(rotation=25)
plt.show()
```





In [ ]:
```
# Displaying the first few rows of a DataFrame
df1.head()
```

Out[ ]:

	Kingdom	DNAtype	SpeciesID	Ncodons	SpeciesName	UUU	UUC	UUA	UUG	CUU	...	GAU	GAC
0	9	0	100217	1995	2827	1437	789	0.00050	0.00351	0.01203	...	0.01003	0.04612
1	9	0	100220	1474	1087	2476	938	0.00068	0.00678	0.00407	...	0.01221	0.04545
2	9	0	100755	4862	8252	1752	1750	0.01357	0.01543	0.00782	...	0.03126	0.02036
3	9	0	100880	1915	5480	1556	1815	0.01619	0.00992	0.01567	...	0.03760	0.01932
4	9	0	100887	22831	7577	2572	952	0.00767	0.03679	0.01380	...	0.04148	0.02483

5 rows × 72 columns

In [ ]:

```
# droping row 5063 from DataFrame df2.
df2 = df2.drop([5063])
# Displaying the first few rows of the modified DataFrame df2
print(df2.head())
# converting the columns to numeric dtype. Ensure that all columns contain numeric values for further nume
df2 = df2.apply(pd.to_numeric)
# droping row 5063 from DataFrame df1.
df1 = df1.drop([5063])
```

	UUC	UUA	UUG	CUU	CUC	CUA	CUG	AUU	\
0	789	0.00050	0.00351	0.01203	0.03208	0.00100	0.04010	0.00551	
1	938	0.00068	0.00678	0.00407	0.02849	0.00204	0.04410	0.01153	
2	1750	0.01357	0.01543	0.00782	0.01111	0.01028	0.01193	0.02283	
3	1815	0.01619	0.00992	0.01567	0.01358	0.00940	0.01723	0.02402	
4	952	0.00767	0.03679	0.01380	0.00548	0.00473	0.02076	0.02716	

	AUC	AUA	...	AGA	AGG	GAU	GAC	GAA	\
0	0.02005	0.00752	...	0.01303	0.03559	0.01003	0.04612	0.01203	
1	0.02510	0.00882	...	0.01696	0.03596	0.01221	0.04545	0.01560	
2	0.01604	0.01316	...	0.01974	0.02489	0.03126	0.02036	0.02242	
3	0.02245	0.02507	...	0.01410	0.01671	0.03760	0.01932	0.03029	
4	0.00867	0.01310	...	0.01494	0.01734	0.04148	0.02483	0.03359	

	GAG	UAA	UAG	UGA	Kingdom_cat
0	0.04361	0.00251	0.00050	0.00000	9
1	0.04410	0.00271	0.00068	0.00000	9
2	0.02468	0.00391	0.00000	0.00144	9
3	0.03446	0.00261	0.00157	0.00000	9
4	0.03679	0.00000	0.00044	0.00131	9

[5 rows x 64 columns]

NPZ file creation

In [ ]:
```
# converting the DataFrame df2 to a NumPy array and then saves it to a .npz file by using np.savez functio
data_array = df2.to_numpy()
np.savez('Group_17_Clean_Data.npz',data_array)
```

Exporting Clean_Dataset

In [ ]:
```
# exporting the contents of df2 to a CSV file
df2.to_csv('Group_17_Clean_Dataset.csv')
```

Model Selection:

As the given problem Pretains to Classification, we would like to use KNN(K-Nearest Neighbours) and LogisticRegression(Multiclass Logistic Regression-one versus rest technique) Algorithms to acheive desired results.