In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.model_selection import validation_curve
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC

from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

In [ ]:
```
# Load Data
df = pd.read_csv("5241dataset.csv", encoding = 'unicode_escape')
```

1. Dataset Description

In [ ]:
```
df.info()
```

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

2. Data Cleaning

2.1 Missing Values

In [ ]:
```
col_na = df.columns[df.isna().any()]
col_na_ratio = df.isna().sum()/df.shape[0]

print("The columns contain missing values are:\n", col_na)

fig,ax = plt.subplots(figsize=(15,4))
sns.barplot(x = df.columns, y = col_na_ratio, ax = ax)

ax.tick_params(axis = 'x', rotation = 90)
ax.set_xlabel('Features')
ax.set_ylabel('Missing Rate')
ax.set_title('Missing Rates of Different Features')

plt.tick_params(labelsize=6)
plt.show()
```

In [ ]:
```
df.drop(['Individual ID','Predator TL/FL/SL conversion reference','Pre
```

In [ ]:
```
df.columns
```

2.2 Dataset Split

In [ ]:
```
df_predator = df[['Record number', 'Predator', 'Predator common name',
                  'Predator lifestage', 'Type of feeding interaction', '
                  'Predator length unit', 'Predator standard length','Pr
                  'Standardised predator length','Predator mass', 'Preda
                  'Predator mass check diff', 'Predator ratio mass/mass'
                  'Geographic location', 'Latitude', 'Latitude Degree',
                  'Longitude', 'Longitude Degree', 'Longitude Minute','L
                  'Mean annual temp', 'SD annual temp','Mean PP', 'SD PP
df_prey = df[['Prey', 'Prey common name', 'Prey taxon', 'Prey length',
                  'Prey mass', 'Prey mass unit', 'Prey mass check', 'Pre
                  'SI prey mass', 'Geographic location', 'Latitude', 'La
                  'Latitude label', 'Longitude', 'Longitude Degree', 'Lo
                  'Depth', 'Mean annual temp', 'SD annual temp','Mean PP
```

2.2.1 Predator Dataframe

In [ ]:
```
df_predator['Predator length unit'].value_counts()
```

```
In [ ]: for i in range(df_predator.shape[0]):
            if df_predator['Predator length unit'][i] == 'mm':
                df_predator['Predator length'][i] = df_predator['Predator leng
            if df_predator['Predator length unit'][i] == 'µm':
                df_predator['Predator length'][i] = df_predator['Predator leng
            else:
                continue
```

```
In [ ]: df_predator['Predator mass unit'].value_counts()
```

```
In [ ]: df_predator.drop(['Predator length unit'], axis = 1, inplace = True)
```

```
In [ ]: df_predator['Predator common name'].value_counts()
```

```
In [ ]: major_predator = df_predator['Predator common name'].value_counts()/df
        major_predator_names = major_predator[major_predator == True].index
        major_predator_names
```

```
In [ ]: df_names = []
        for name in major_predator_names:
            df_names.append(df_predator[df_predator['Predator common name'] ==
        df_predator_new = pd.concat(df_names, ignore_index= True)
```

```
In [ ]: predator_y = df_predator_new['Predator common name']
        predator_x = df_predator_new[['Predator taxon','Predator lifestage', '
                                      'Diet coverage','Geographic location','Dep
```

2.2.2 Prey Dataframe

```
In [ ]: df_prey['Prey length unit'].value_counts()
```

```
In [ ]: for i in range(df_prey.shape[0]):
            if df_prey['Prey length unit'][i] == 'mm':
                df_prey['Prey length'][i] = df_prey['Prey length'][i]*0.1
            if df_prey['Prey length unit'][i] == 'µm':
                df_prey['Prey length'][i] = df_prey['Prey length'][i]*0.0001
            else:
                continue
```

```
In [ ]: df_prey['Prey mass unit'].value_counts()
```

```
In [ ]: for i in range(df_prey.shape[0]):
            if df_prey['Prey mass unit'][i] == 'mg':
                df_prey['Prey mass'][i] = df_prey['Prey mass'][i]*0.001
            else:
                continue
```

```
In [ ]: df_prey.drop(['Prey length unit','Prey mass unit'], axis = 1, inplace
```

```
In [ ]: prey_y = df_prey['Prey taxon']
        prey_x = df_prey[['Prey length','Prey mass', 'Geographic location','De
```

3. Data Visualization

3.1 Correlations of Independent Variables

```
In [ ]: # Predators
        corr_matrix = predator_x.corr()
        fig = plt.figure(figsize = (4,4))
        plt.title('Correlations of Predators')
        sns.heatmap(corr_matrix, annot=True, annot_kws={"size":6}, center=0, c
        plt.show()
```

```
In [ ]: # Preys
        corr_matrix = prey_x.corr()
        fig = plt.figure(figsize = (4,4))
        plt.title('Correlations of Preys')
        sns.heatmap(corr_matrix, annot=True, annot_kws={"size":6}, center=0, c
        plt.show()
```

```
In [ ]: df['Prey taxon'].unique()
```

3.2 Distribution of Dependent Variable

```
In [ ]: fig, ax=plt.subplots(figsize=(5,5))
        sns.histplot(predator_y, ax=ax)
        plt.xticks(rotation = 90)
        plt.show()
```

```
In [ ]: fig, ax=plt.subplots(figsize=(5,5))
        sns.histplot(prey_y, ax=ax)
        plt.xticks(rotation = 90)
        plt.show()
```

3.3 Numerial Features

In [ ]:
```
#predator and prey
df_v = pd.concat([predator_x,prey_x],axis=1)

numeric_cols = ['Predator length', 'Predator mass','Prey length', 'Pre
print(len(numeric_cols))

fig, ax=plt.subplots(nrows=2, ncols=2, figsize=(10,10))

for var, subplot in zip(numeric_cols, ax.flatten()):
    b=sns.histplot(x=var, data=df_v, ax=subplot)
    b.set_xlabel(str(var), fontsize = 5)
    b.set_title(f'Distribution of {var}')

plt.show()
```

In [ ]:
```
# geographical
df_v = pd.concat([predator_x,prey_x],axis=1)

numeric_cols = ['Depth', 'Mean annual temp','Mean PP']
print(len(numeric_cols))

fig, ax=plt.subplots(nrows=1, ncols=3, figsize=(17,6))

for var, subplot in zip(numeric_cols, ax.flatten()):
    b=sns.histplot(x=var, data=df, ax=subplot)
    b.set_xlabel(str(var), fontsize = 10)
    b.set_title(f'Distribution of {var}')

plt.show()
```

3.4 Categorical Features

In [ ]:
```
# Prey Common Names
def myformat(value):
    return f'{value:.1f}%'

threshold = 1000
y = df['Prey common name'].value_counts()
mylabels =y.index
print(mylabels[y >= threshold])
small_values = y[y < threshold]
other_value = sum(small_values)
y = y[y >= threshold]
mylabels = y.index
y = np.append(y, other_value)
mylabels = np.append(mylabels, "Other")

fig1, ax1 = plt.subplots(figsize = (5,5))
ax1.pie(y, labels=mylabels, textprops={'rotation': 0}, autopct=myforma
ax1.set_title("Prey Common Name distribution")
plt.show()
```

In [ ]:
```
# Prey taxon
def myformat(value):
    return f'{value:.1f}%'

threshold = 1000
y = df['Prey taxon'].value_counts()
mylabels =y.index

fig1, ax1 = plt.subplots(figsize = (5,5))
ax1.pie(y, labels=mylabels, textprops={'rotation': 0}, autopct=myforma
ax1.set_title("Prey taxon distribution")
plt.show()
```

In [ ]:
```
# predator common names
def myformat(value):
    return f'{value:.1f}%'

threshold = 1000
y = df['Predator common name'].value_counts()
mylabels =y.index
print(mylabels[y >= threshold])
small_values = y[y < threshold]
other_value = sum(small_values)
y = y[y >= threshold]
mylabels = y.index
y = np.append(y, other_value)
mylabels = np.append(mylabels, "Other")

fig1, ax1 = plt.subplots(figsize = (5,5))
ax1.pie(y, labels=mylabels, textprops={'rotation': 0}, autopct=myforma
ax1.set_title("Predator Common Name distribution")
plt.show()
```

In [ ]:
```
# Predator taxon
def myformat(value):
    return f'{value:.1f}%'

threshold = 1000
y = df['Predator taxon'].value_counts()
mylabels =y.index

fig1, ax1 = plt.subplots(figsize = (5,5))
ax1.pie(y, labels=mylabels, textprops={'rotation': 0}, autopct=myforma
ax1.set_title("Predator taxon distribution")
plt.show()
```

4. Classification Model

4.1 Classify Predator common names

In [ ]:
```
# Categorical Feature Encoding
ordinalencoder = OrdinalEncoder()
predator_x['Predator taxon'] = ordinalencoder.fit_transform(predator_x
predator_x['Predator lifestage'] = ordinalencoder.fit_transform(predat
predator_x['Type of feeding interaction'] = ordinalencoder.fit_transfo
predator_x['Diet coverage'] = ordinalencoder.fit_transform(predator_x[
predator_x['Geographic location'] = ordinalencoder.fit_transform(preda
predator_x['Specific habitat'] = ordinalencoder.fit_transform(predator
```

In [ ]:
```
# Train Test Split
X_train_predator, X_test_predator, y_train_predator, y_test_predator =




# Scaling
scaler = StandardScaler()
X_train_predator = scaler.fit_transform(X_train_predator)
X_test_predator = scaler.transform(X_test_predator)
```

4.1.1 SVC

In [ ]:
```
svc = SVC(C=1,gamma='scale')
svc.fit(X_train_predator,y_train_predator)
print(f"The train score is:",svc.score(X_train_predator,y_train_predat
print(f"The test score is:",svc.score(X_test_predator,y_test_predator)
```

In [ ]:
```
# Grid Search
params = {'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01]}
svc_gscv = GridSearchCV(estimator = SVC(random_state=123), param_grid=
svc_gscv.fit(X_train_predator, y_train_predator)
print(f'svc best hyperparams      : {svc_gscv.best_params_}')
print(f'svc best mean cv accuracy : {svc_gscv.best_score_:.2f}')
```

In [ ]:
```
svc_best = SVC(C=10,gamma=1,kernel='linear')
svc_best.fit(X_train_predator,y_train_predator)
```

In [ ]:
```
importances = svc_best.coef_[0]
features_dict = dict(zip(predator_x.columns, abs(importances)))
important_features = sorted(features_dict.items(), key=lambda x: -x[1]
important_features
```

In [ ]:
```
sns.barplot(x=predator_x.columns, y=abs(importances))
plt.xticks(rotation=90)
plt.title("Feature Importance of SVC")
plt.xlabel("Feature names")
plt.ylabel("Feature importance")
plt.show()
```

4.1.2 Decision Tree

In [ ]:
```
dt = DecisionTreeClassifier(criterion='gini', max_depth = 5)
dt.fit(X_train_predator,y_train_predator)
print(f"The train score is:",dt.score(X_train_predator,y_train_predato
print(f"The test score is:",dt.score(X_test_predator,y_test_predator))
```

In [ ]:
```
# Grid Search
params = {'criterion': ['gini', 'entropy'], 'max_depth' : [2,3,4,5,6]}
dt_gscv = GridSearchCV(estimator = DecisionTreeClassifier(random_state
dt_gscv.fit(X_train_predator, y_train_predator)
print(f'dt best hyperparams      : {dt_gscv.best_params_}')
print(f'dt best mean cv accuracy : {dt_gscv.best_score_:.2f}')
```

In [ ]:
```
dt_best = DecisionTreeClassifier(criterion='entropy',max_depth=6)
dt_best.fit(X_train_predator, y_train_predator)
```

In [ ]:
```
importances = dt_best.feature_importances_
features_dict = dict(zip(predator_x.columns, abs(importances)))
important_features = sorted(features_dict.items(), key=lambda x: -x[1]
important_features
```

In [ ]:
```
sns.barplot(x=predator_x.columns, y=abs(importances))
plt.xticks(rotation=90)
plt.title("Feature Importance of Decision Tree")
plt.xlabel("Feature names")
plt.ylabel("Feature importance")
plt.show()
```

4.1.3 Random Forest

In [ ]:
```
# normal random forest
rfc = RandomForestClassifier(n_estimators=50, max_depth=5)
rfc.fit(X_train_predator,y_train_predator)
print(f"The train score is:",rfc.score(X_train_predator,y_train_predat
print(f"The test score is:",rfc.score(X_test_predator,y_test_predator)
```

In [ ]:
```
# cross validation
rfc_cv_scores = cross_val_score(rfc, X_train_predator, y_train_predato
rfc_cv_scores
```

In [ ]:
```
depths = [2,4,6,8,10]
train_scores,test_scores = validation_curve(RandomForestClassifier(n_e
                                            X_train_predator, y_train_
                                            param_name='max_depth', pa
mean_train_scores = np.average(train_scores, axis=1)
mean_test_scores = np.average(test_scores, axis=1)
pd.DataFrame([mean_train_scores.round(2),mean_test_scores.round(2)],
             columns=pd.Series(depths,name='max_depth'),
             index=['mean_train_scores','mean_test_scores'])
```

In [ ]:
```
# Grid Search
params = {'n_estimators':[10,50,100,150],'max_depth':[2,3,4,5,6]}
rfc_gscv = GridSearchCV(estimator=RandomForestClassifier(random_state=
rfc_gscv.fit(X_train_predator, y_train_predator)
print(f'rfc best hyperparams      : {rfc_gscv.best_params_}')
print(f'rfc best mean cv accuracy : {rfc_gscv.best_score_:.2f}')
```

In [ ]:
```
rfc_best = RandomForestClassifier(max_depth=6,n_estimators=10)
rfc_best.fit(X_train_predator, y_train_predator)
```

In [ ]:
```
importances = rfc_best.feature_importances_
features_dict = dict(zip(predator_x.columns, abs(importances)))
important_features = sorted(features_dict.items(), key=lambda x: -x[1]
important_features
```

In [ ]:
```
sns.barplot(x=predator_x.columns, y=abs(importances))
plt.xticks(rotation=90)
plt.title("Feature Importance of Random Forest")
plt.xlabel("Feature names")
plt.ylabel("Feature importance")
plt.show()
```

4.1.4 Adaboost

In [ ]:
```
ada = AdaBoostClassifier(n_estimators=50, learning_rate=1)
ada.fit(X_train_predator,y_train_predator)
print(f"The train score is:",ada.score(X_train_predator,y_train_predat
print(f"The test score is:",ada.score(X_test_predator,y_test_predator)
```

In [ ]:
```
# Grid Search
params = {'n_estimators':[10,50,100,150],'learning_rate':[0.01,0.1,0.5
ada_gscv = GridSearchCV(estimator=AdaBoostClassifier(random_state=123)
ada_gscv.fit(X_train_predator, y_train_predator)
print(f'ada best hyperparams      : {ada_gscv.best_params_}')
print(f'ada best mean cv accuracy : {ada_gscv.best_score_:.2f}')
```

In [ ]:
```
ada_best = AdaBoostClassifier(learning_rate=0.5,n_estimators=50)
ada_best.fit(X_train_predator, y_train_predator)
```

In [ ]:
```
importances = ada_best.feature_importances_
features_dict = dict(zip(predator_x.columns, abs(importances)))
important_features = sorted(features_dict.items(), key=lambda x: -x[1]
important_features
```

```
In [ ]: sns.barplot(x=predator_x.columns, y=abs(importances))
        plt.xticks(rotation=90)
        plt.title("Feature Importance of AdaBoost")
        plt.xlabel("Feature names")
        plt.ylabel("Feature importance")
        plt.show()
```

4.2 Classify Prey taxon

```
In [ ]: ordinalencoder = OrdinalEncoder()
        prey_x['Geographic location'] = ordinalencoder.fit_transform(prey_x[['
        prey_x['Specific habitat'] = ordinalencoder.fit_transform(prey_x[['Spe
```

```
In [ ]: from sklearn.model_selection import train_test_split

        # Train Test Split
        X_train_prey, X_test_prey, y_train_prey, y_test_prey = train_test_spli




        # Scaling
        scaler = StandardScaler()
        X_train_prey = scaler.fit_transform(X_train_prey)
        X_test_prey = scaler.transform(X_test_prey)
```

4.2.1 SVC

```
In [ ]: svc = SVC(C=1,gamma='scale')
        svc.fit(X_train_prey,y_train_prey)
        print(f"The train score is:",svc.score(X_train_prey,y_train_prey))
        print(f"The test score is:",svc.score(X_test_prey,y_test_prey))
```

```
In [ ]: # Grid Search
        params = {'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01]}
        svc_gscv = GridSearchCV(estimator = SVC(random_state=123), param_grid=
        svc_gscv.fit(X_train_prey, y_train_prey)
        print(f'svc best hyperparams      : {svc_gscv.best_params_}')
        print(f'svc best mean cv accuracy : {svc_gscv.best_score_:.2f}')
```

```
In [ ]: svc_best = SVC(C=10,gamma=1,kernel='linear')
        svc_best.fit(X_train_prey,y_train_prey)
```

```
In [ ]: importances = svc_best.coef_[0]
        features_dict = dict(zip(prey_x.columns, abs(importances)))
        important_features = sorted(features_dict.items(), key=lambda x: -x[1]
        important_features
```

```
In [ ]: sns.barplot(x=prey_x.columns, y=abs(importances))
        plt.xticks(rotation=90)
        plt.title("Feature Importance of SVC")
        plt.xlabel("Feature names")
        plt.ylabel("Feature importance")
        plt.show()
```

4.2.2 Decision Tree

```
In [ ]: dt = DecisionTreeClassifier(criterion='gini', max_depth = 5)
        dt.fit(X_train_prey,y_train_prey)
        print(f"The train score is:",dt.score(X_train_prey,y_train_prey))
        print(f"The test score is:",dt.score(X_test_prey,y_test_prey))
```

```
In [ ]: # Grid Search
        params = {'criterion': ['entropy'], 'max_depth' : [2,3,4,5,6]}
        dt_gscv = GridSearchCV(estimator = DecisionTreeClassifier(random_state
        dt_gscv.fit(X_train_prey, y_train_prey)
        print(f'decision tree best hyperparams      : {dt_gscv.best_params_}')
        print(f'decision tree best mean cv accuracy : {dt_gscv.best_score_:.2f
```

```
In [ ]: dt_best = DecisionTreeClassifier(criterion='entropy',max_depth=6)
        dt_best.fit(X_train_prey, y_train_prey)
```

```
In [ ]: importances = dt_best.feature_importances_
        features_dict = dict(zip(prey_x.columns, abs(importances)))
        important_features = sorted(features_dict.items(), key=lambda x: -x[1]
        important_features
```

```
In [ ]: sns.barplot(x=prey_x.columns, y=abs(importances))
        plt.xticks(rotation=90)
        plt.title("Feature Importance of Decision Tree")
        plt.xlabel("Feature names")
        plt.ylabel("Feature importance")
        plt.show()
```

4.2.3 Random Forest

In [ ]:
```
# normal random forest
rfc = RandomForestClassifier(n_estimators=50, max_depth=5)
rfc.fit(X_train_prey,y_train_prey)
print(f"The train score is:",rfc.score(X_train_prey,y_train_prey))
print(f"The test score is:",rfc.score(X_test_prey,y_test_prey))
```

In [ ]:
```
# cross validation
rfc_cv_scores = cross_val_score(rfc, X_train_prey, y_train_prey, cv=5,
rfc_cv_scores
```

In [ ]:
```
depths = [2,4,6,8,10]
train_scores,test_scores = validation_curve(RandomForestClassifier(n_e
                                            X_train_prey, y_train_prey
                                            param_name='max_depth', pa
mean_train_scores = np.average(train_scores, axis=1)
mean_test_scores = np.average(test_scores, axis=1)
pd.DataFrame([mean_train_scores.round(2),mean_test_scores.round(2)],
             columns=pd.Series(depths,name='max_depth'),
             index=['mean_train_scores','mean_test_scores'])
```

In [ ]:
```
# Grid Search
params = {'n_estimators':[10,50,100,150],'max_depth':[2,3,4,5,6]}
rfc_gscv = GridSearchCV(estimator=RandomForestClassifier(random_state=
rfc_gscv.fit(X_train_prey, y_train_prey)
print(f'random forest best hyperparams      : {rfc_gscv.best_params_}'
print(f'random forest best mean cv accuracy : {rfc_gscv.best_score_:.2
```

In [ ]:
```
rfc_best = RandomForestClassifier(max_depth=6,n_estimators=10)
rfc_best.fit(X_train_prey, y_train_prey)
```

In [ ]:
```
importances = rfc_best.feature_importances_
features_dict = dict(zip(prey_x.columns, abs(importances)))
important_features = sorted(features_dict.items(), key=lambda x: -x[1]
important_features
```

In [ ]:
```
sns.barplot(x=prey_x.columns, y=abs(importances))
plt.xticks(rotation=90)
plt.title("Feature Importance of Random Forest")
plt.xlabel("Feature names")
plt.ylabel("Feature importance")
plt.show()
```

4.2.4 Adaboost

In [ ]:
```
ada = AdaBoostClassifier(n_estimators=50, learning_rate=1)
ada.fit(X_train_prey,y_train_prey)
print(f"The train score is:",ada.score(X_train_prey,y_train_prey))
print(f"The test score is:",ada.score(X_test_prey,y_test_prey))
```

In [ ]:
```
# Grid Search
params = {'n_estimators':[10,50,100,150],'learning_rate':[0.01,0.1,0.5
ada_gscv = GridSearchCV(estimator=AdaBoostClassifier(random_state=123)
ada_gscv.fit(X_train_prey, y_train_prey)
print(f'ada best hyperparams      : {ada_gscv.best_params_}')
print(f'ada best mean cv accuracy : {ada_gscv.best_score_:.2f}')
```

In [ ]:
```
ada_best = AdaBoostClassifier(learning_rate=1,n_estimators=10)
ada_best.fit(X_train_prey, y_train_prey)
```

In [ ]:
```
importances = ada_best.feature_importances_
features_dict = dict(zip(prey_x.columns, abs(importances)))
important_features = sorted(features_dict.items(), key=lambda x: -x[1]
important_features
```

In [ ]:
```
sns.barplot(x=prey_x.columns, y=abs(importances))
plt.xticks(rotation=90)
plt.title("Feature Importance of AdaBoost")
plt.xlabel("Feature names")
plt.ylabel("Feature importance")
plt.show()
```

5 PCA

5.1 PCA for predator

In [ ]:
```
pca = PCA(n_components=3, random_state=123)

X_train_pca_predator = pca.fit_transform(X_train_predator)
X_test_pca_predator = pca.transform(X_test_predator)

pca.explained_variance_ratio_
```

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X_train_pca_predator[:,0],X_train_pca_predator[:,1],X_train
plt.title('PCA for Predator')
```

In [ ]:
```
fig, ax=plt.subplots(nrows=1, ncols=3, figsize=(15,5))

sns.scatterplot(x = X_train_pca_predator[:,0], y = X_train_pca_predato
ax[0].set_title("Dimensions 0&1 of the PCA Transformation")

sns.scatterplot(x = X_train_pca_predator[:,0], y = X_train_pca_predato
ax[1].set_title("Dimensions 0&2 of the PCA Transformation")

sns.scatterplot(x = X_train_pca_predator[:,1], y = X_train_pca_predato
ax[2].set_title("Dimensions 1&2 of the PCA Transformation")
```

5.2 PCA for prey

In [ ]:
```
pca = PCA(n_components=3, random_state=123)

X_train_pca_prey = pca.fit_transform(X_train_prey)
X_test_pca_prey = pca.transform(X_test_prey)

pca.explained_variance_ratio_
```

In [ ]:
```
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X_train_pca_prey[:,0],X_train_pca_prey[:,1],X_train_pca_pre
plt.title('PCA for Predator')
```

In [ ]:
```
fig, ax=plt.subplots(nrows=1, ncols=3, figsize=(15,5))

sns.scatterplot(x = X_train_pca_prey[:,0], y = X_train_pca_prey[:,1],
ax[0].set_title("Dimensions 0&1 of the PCA Transformation")

sns.scatterplot(x = X_train_pca_prey[:,0], y = X_train_pca_prey[:,2],
ax[1].set_title("Dimensions 0&2 of the PCA Transformation")

sns.scatterplot(x = X_train_pca_prey[:,1], y = X_train_pca_prey[:,2],
ax[2].set_title("Dimensions 1&2 of the PCA Transformation")
```

6 Clustering Model: Kmeans

6.1 Kmeans to cluster predator

In [ ]:
```
inertia = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(predator_x)
    inertia.append(km.inertia_)

plt.plot(range(1, 11), inertia, marker='o')
```

In [ ]:
```
kmeans_predator = KMeans(n_clusters=2, random_state=123)
kmeans_predator.fit(predator_x)
labels_predator = kmeans_predator.labels_
labels_predator = pd.DataFrame(labels_predator, columns=["kmeans_label
labels_predator.value_counts()
```

In [ ]:
```
X_predator_cluster = pd.concat([predator_x, predator_y, labels_predato
X_predator_cluster.groupby('Predator common name').mean()
```

In [ ]:
```
y_train_predator.value_counts()
```

6.2 Kmeans to cluster prey

In [ ]:
```
inertia = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(prey_x)
    inertia.append(km.inertia_)

plt.plot(range(1, 11), inertia, marker='o')
```

In [ ]:
```
kmeans_prey = KMeans(n_clusters=2, random_state=123)
kmeans_prey.fit(prey_x)
labels_prey = kmeans_prey.labels_
labels_prey = pd.DataFrame(labels_prey, columns=["kmeans_label"])
labels_prey.value_counts()
```

In [ ]:
```
X_prey_cluster = pd.concat([prey_x, prey_y, labels_prey], axis=1)
X_prey_cluster.groupby('Prey taxon').mean()
```

7. Regression Model

```
In [ ]: merged_df = pd.merge(df_predator, df_prey, left_index=True, right_inde

        # print(merged_df.columns)

        merged_df['Mass difference'] = merged_df['Predator mass'] - merged_df[
        # merged_df['Mass difference'] = merged_df['Predator length'] - mergec

        reg_x = merged_df[['Type of feeding interaction', 'Geographic location
                           'Latitude Minute_x', 'Latitude label_x', 'Longitude
                           'Longitude label_x', 'Depth_x', 'Mean annual temp_x
                           'Mean PP_x', 'SD PP_x', 'Specific habitat_x']]

        reg_y = merged_df['Mass difference']
```

```
In [ ]: reg_y
```

```
In [ ]: non_numeric_columns = reg_x.select_dtypes(exclude=['int64', 'float64']

        ordinalencoder = OrdinalEncoder()

        for col_name in list(non_numeric_columns):
        #     print(col_name)
            reg_x[col_name] = reg_x[col_name].astype(str)
            reg_x[col_name] = ordinalencoder.fit_transform(reg_x[[col_name]]).
```

```
In [ ]: # Train Test Split
        X_train_len, X_test_len, y_train_len, y_test_len = train_test_split(re
                                                                             re
                                                                             te
                                                                             ra

        # Scaling
        scaler = StandardScaler()
        X_train_len = scaler.fit_transform(X_train_len)
        X_test_len = scaler.transform(X_test_len)
```

7.1 Linear Regression

```
In [ ]: reg = LinearRegression()
        reg.fit(X_train_len,y_train_len)
        print(f"The train score:", reg.score(X_train_len,y_train_len))
        print(f"The test score:", reg.score(X_test_len,y_test_len))
```

7.2 Ridge Regression

```
In [ ]: ridge = Ridge(alpha=1)
        ridge.fit(X_train_len,y_train_len)
        print(f"The train score:", ridge.score(X_train_len,y_train_len))
        print(f"The test score:", ridge.score(X_test_len,y_test_len))
```

```
In [ ]: params = {'alpha':[0.01,0.1,0.5,1]}
        ridge_gscv = GridSearchCV(estimator=Ridge(), param_grid=params, cv=3,
        ridge_gscv.fit(X_train_len, y_train_len)
        print(f'ada best hyperparams      : {ridge_gscv.best_params_}')
        print(f'ada best mean cv accuracy : {ridge_gscv.best_score_:.2f}')
```

7.3 Lasso Regression

```
In [ ]: lasso = Lasso(alpha=1)
        lasso.fit(X_train_len,y_train_len)
        print(f"The train score:", lasso.score(X_train_len,y_train_len))
        print(f"The test score:", lasso.score(X_test_len,y_test_len))
```

```
In [ ]: params = {'alpha':[0.01,0.1,0.5,1]}
        lasso_gscv = GridSearchCV(estimator=Lasso(), param_grid=params, cv=3,
        lasso_gscv.fit(X_train_len, y_train_len)
        print(f'ada best hyperparams      : {lasso_gscv.best_params_}')
        print(f'ada best mean cv accuracy : {lasso_gscv.best_score_:.2f}')
```

7.5 Decision Tree Regression

```
In [ ]: dtr = DecisionTreeRegressor(max_depth=5)
        dtr.fit(X_train_len,y_train_len)
        print(f"The train score:", dtr.score(X_train_len,y_train_len))
        print(f"The test score:", dtr.score(X_test_len,y_test_len))
```

```
In [ ]: params = {'criterion': ['mse', 'mae'], 'max_depth' : [2,3,4,5,6]}
        dt_gscv = GridSearchCV(estimator = DecisionTreeRegressor(random_state=
        dt_gscv.fit(X_train_len, y_train_len)
        print(f'decision tree best hyperparams      : {dt_gscv.best_params_}')
        print(f'decision tree best mean cv accuracy : {dt_gscv.best_score_:.2f
```

```
In [ ]: df = pd.DataFrame(dtr.feature_importances_, index=reg_x.columns).sort_
        df['index'] = df['index'].str.split('_').str[0]
        df
```

```
In [ ]: sns.set_style("whitegrid")
        sns.set(rc={'figure.figsize':(11.7,5)})

        # Create the bar chart using seaborn's barplot function
        ax = sns.barplot(y=0, x='index', data=df)

        # Set the title and axis labels
        ax.set_title("Feature Importance for Decision Tree")
        ax.set_xlabel("Importance")
        ax.set_ylabel("Feature")

        # Show the plot
        plt.show()
```

5.6 Adaboost Regression

```
In [ ]: adarg = AdaBoostRegressor(n_estimators=100, random_state=123)
        adarg.fit(X_train_len,y_train_len)
        print(f"The train score:", adarg.score(X_train_len,y_train_len))
        print(f"The test score:", adarg.score(X_test_len,y_test_len))
```

```
In [ ]: params = {'n_estimators':[10,50,100,150],'learning_rate':[0.01,0.1,0.5
        ada_gscv = GridSearchCV(estimator=AdaBoostRegressor(random_state=123),
        ada_gscv.fit(X_train_len, y_train_len)
        print(f'ada best hyperparams      : {ada_gscv.best_params_}')
        print(f'ada best mean cv accuracy : {ada_gscv.best_score_:.2f}')
```

```
In [ ]: df = pd.DataFrame(adarg.feature_importances_, index=reg_x.columns).sor
        df['index'] = df['index'].str.split('_').str[0]
        df
```

```
In [ ]: sns.set_style("whitegrid")
        sns.set(rc={'figure.figsize':(11.7,5)})

        # Create the bar chart using seaborn's barplot function
        ax = sns.barplot(y=0, x='index', data=df)

        # Set the title and axis labels
        ax.set_title("Feature Importance for AdaBoost")
        ax.set_xlabel("Importance")
        ax.set_ylabel("Feature")

        # Show the plot
        plt.show()
```

5.6 Random Forest Regression

```
In [ ]: rf = RandomForestRegressor(max_depth=5)
        rf.fit(X_train_len,y_train_len)
        print(f"The train score:", dtr.score(X_train_len,y_train_len))
        print(f"The test score:", dtr.score(X_test_len,y_test_len))


        depths = [2,4,6,8,10]
        train_scores,test_scores = validation_curve(RandomForestRegressor(n_es
                                                    X_train_len, y_train_len,
                                                    param_name='max_depth', pa
        mean_train_scores = np.average(train_scores, axis=1)
        mean_test_scores = np.average(test_scores, axis=1)
        pd.DataFrame([mean_train_scores.round(2),mean_test_scores.round(2)],
                     columns=pd.Series(depths,name='max_depth'),
                     index=['mean_train_scores','mean_test_scores'])
```

```
In [ ]: params = {'n_estimators':[10,50,100,150],'max_depth':[2,3,4,5,6]}
        rfc_gscv = GridSearchCV(estimator=RandomForestRegressor(random_state=1
        rfc_gscv.fit(X_train_len, y_train_len)
        print(f'random forest best hyperparams      : {rfc_gscv.best_params_}'
        print(f'random forest best mean cv accuracy : {rfc_gscv.best_score_:.2
```

```
In [ ]: df = pd.DataFrame(rf.feature_importances_, index=reg_x.columns).sort_v
        df['index'] = df['index'].str.split('_').str[0]
        df
```

```
In [ ]: sns.set_style("whitegrid")
        sns.set(rc={'figure.figsize':(11.7,5)})

        # Create the bar chart using seaborn's barplot function
        ax = sns.barplot(y=0, x='index', data=df)

        # Set the title and axis labels
        ax.set_title("Feature Importance for Random Forest")
        ax.set_xlabel("Importance")
        ax.set_ylabel("Feature")

        # Show the plot
        plt.show()
```