Possum Regression

Bhavans Vivekananda College

Group-1

B.sruthi (10722546004)
Amreen (107222546002)
B.saketh(107222546005)

Abstract

-This dataset provides a collection of biometric measurements from 104 possums, with the aim of understanding variations in physical traits across different populations, sexes, and age groups.

Objectives

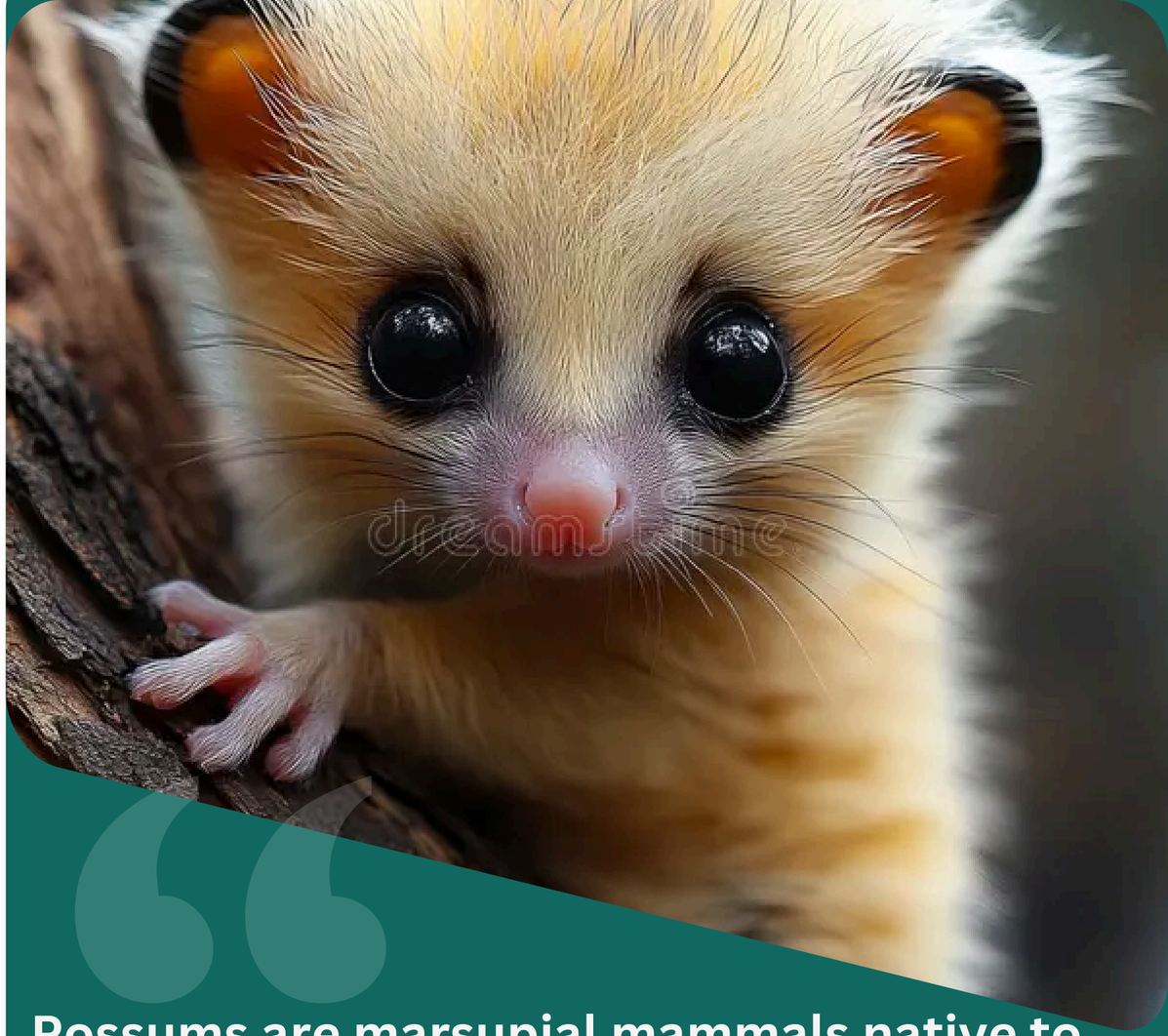
-The key objective is to predict the age of the possum based on the body measurements of the possum. We perform Machine Learning algorithms to predict the best algorithm among other algorithms.

Content

Introduction

This dataset contains measurements and demographic information for 104 possums. It includes 14 columns, with each row representing a unique possum.

This dataset is likely aimed at understanding physical variation among possums from different locations, sexes, or ages, potentially for ecological or conservation studies.



Possums are marsupial mammals native to Australia, New Guinea, and nearby islands. They are known for their adaptability to various environments, ranging from forests to urban areas.

LITERATURE review

Literature review1

Age-Related Morphological Changes

McDonald and Rose (2000) examined age-related changes in skeletal morphology of possums, documenting how certain features, like skull width and body length, increase with age. Their findings are often used to create growth models for wild populations.

Literature review 2

Habitat and Environmental Impact on Morphology

-Tyndale-Biscoe and MacKenzie (1976) examined morphological differences in possums from coastal versus inland habitats, concluding that environmental factors like temperature and food resources significantly impact body size and growth patterns.

Data Preprocessing

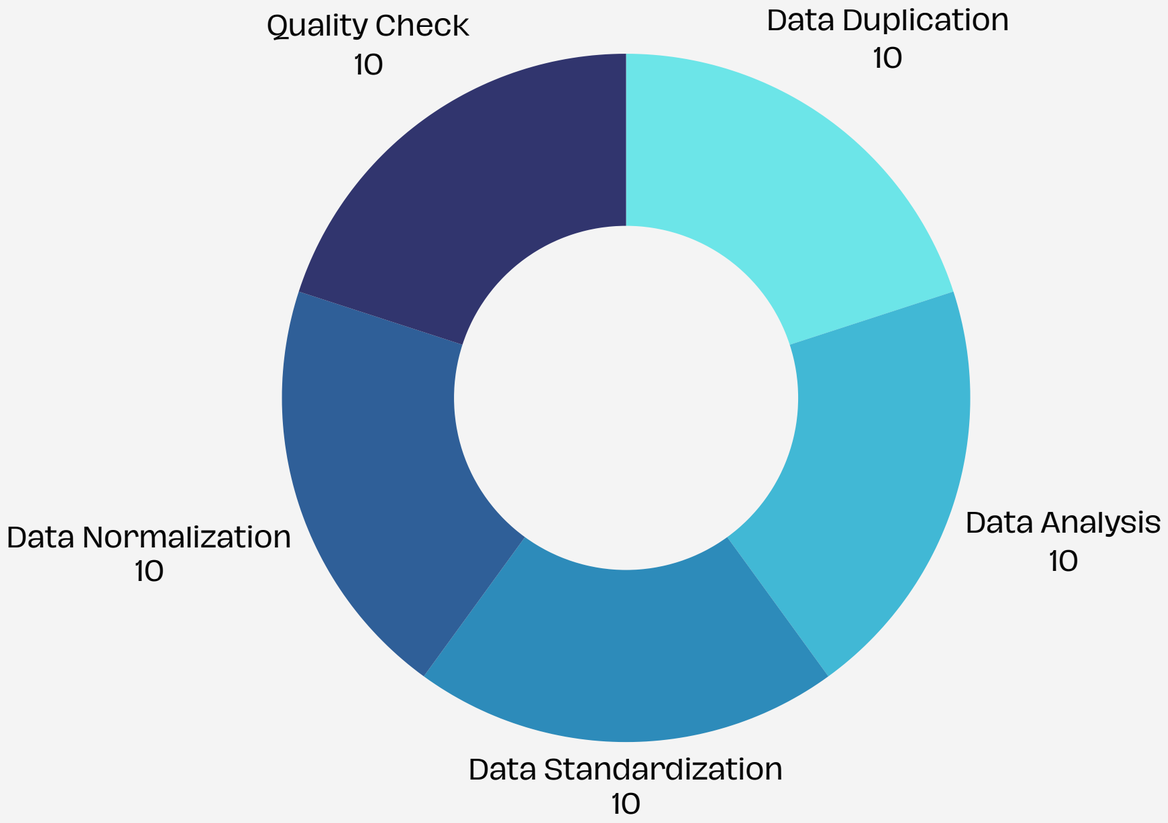
Data

Dataset : Our dataset consists of 14 variables and 104 records.

Source: https://drive.google.com/file/d/1tz7K-gzVZVAwyW2YOW-PI7YXnGUBY_MK/view?usp=drive_link

case	site	Pop	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly
1	1	Vic	m	8	94.1	60.4	89	36	74.5	54.5	15.2	28	36
2	1	Vic	f	6	92.5	57.6	91.5	36.5	72.5	51.2	16	28.5	33
3	1	Vic	f	6	94	60	95.5	39	75.4	51.9	15.5	30	34
4	1	Vic	f	6	93.2	57.1	92	38	76.1	52.2	15.2	28	34
5	1	Vic	f	2	91.5	56.3	85.5	36	71	53.2	15.1	28.5	33
6	1	Vic	f	1	93.1	54.8	90.5	35.5	73.2	53.6	14.2	30	32
7	1	Vic	m	2	95.3	58.2	89.5	36	71.5	52	14.2	30	34.5
8	1	Vic	f	6	94.8	57.6	91	37	72.7	53.9	14.5	29	34
9	1	Vic	f	9	93.4	56.3	91.5	37	72.4	52.9	15.5	28	33
10	1	Vic	f	6	91.8	58	89.5	37.5	70.9	53.4	14.4	27.5	32
11	1	Vic	f	9	93.3	57.2	89.5	39	77.2	51.3	14.9	31	34
12	1	Vic	f	5	94.9	55.6	92	35.5	71.7	51	15.3	28	33
13	1	Vic	m	5	95.1	59.9	89.5	36	71	49.8	15.8	27	32
14	1	Vic	m	3	95.4	57.6	91.5	36	74.3	53.7	15.1	28	31.5
15	1	Vic	m	5	92.9	57.6	85.5	34	69.7	51.8	15.7	28	35
16	1	Vic	m	4	91.6	56	86	34.5	73	51.4	14.4	28	9 32
17	1	Vic	f	1	94.7	67.7	89.5	36.5	73.2	53.2	14.7	29	31
18	1	Vic	m	2	93.5	55.7	90	36	73.7	55.4	15.3	28	32

Data Cleaning

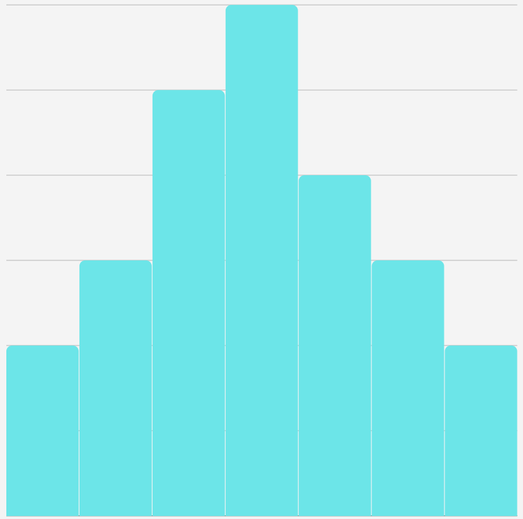


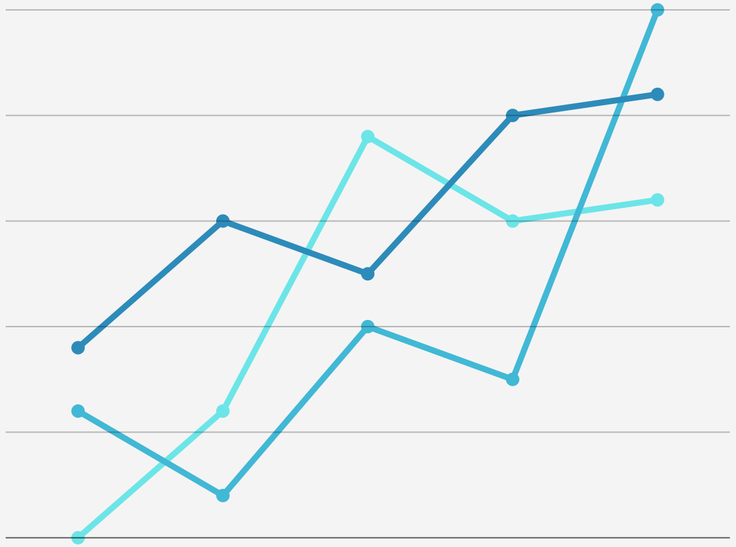
Checked for null /NaN values : We replaced the Nan values with mean and mode of the column based on the attribute type.

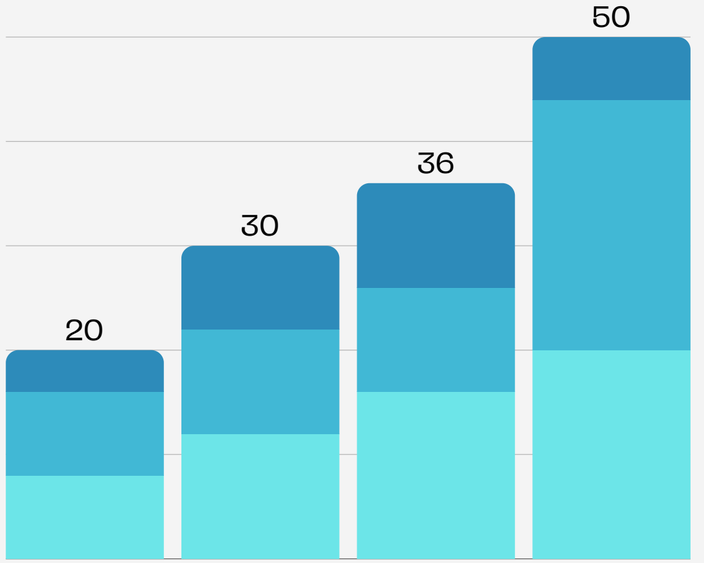
Garbage values : We replaced the garbage values of the column based on the domain knowledge of the dataset.

Dummification: enabled and dummified the categorical variables.

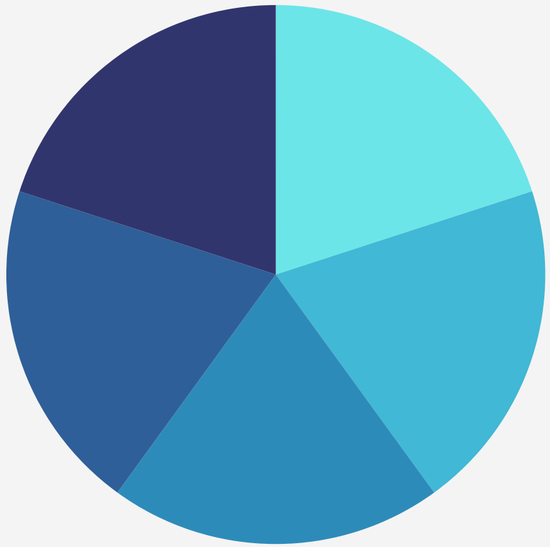
variables: categorical variables: pop ,sex
continuous variables: case, site, age, hdlngth,skullw ,totlngth, taill ,footlgth,ear-conch,eye,chest,belly

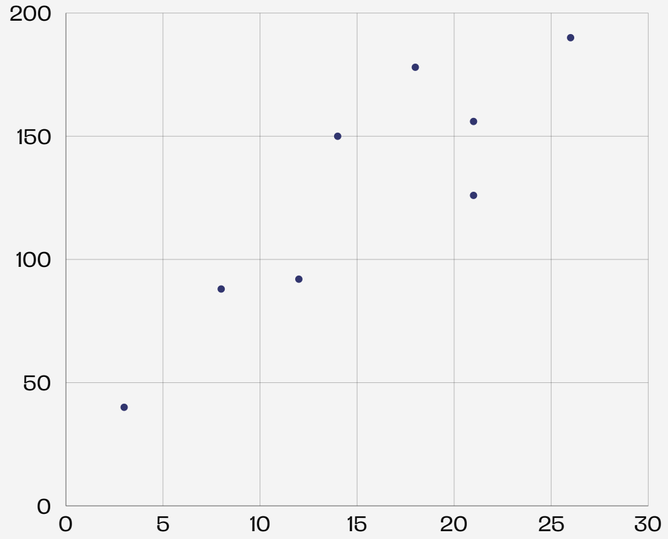


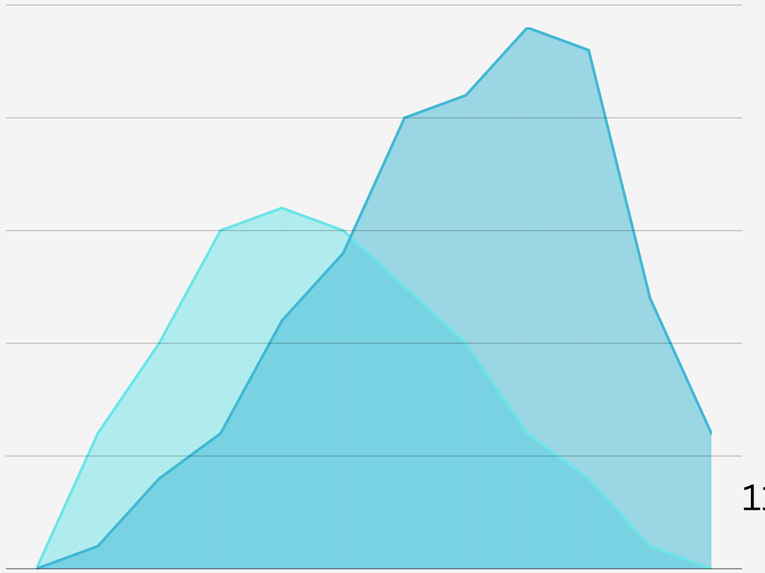




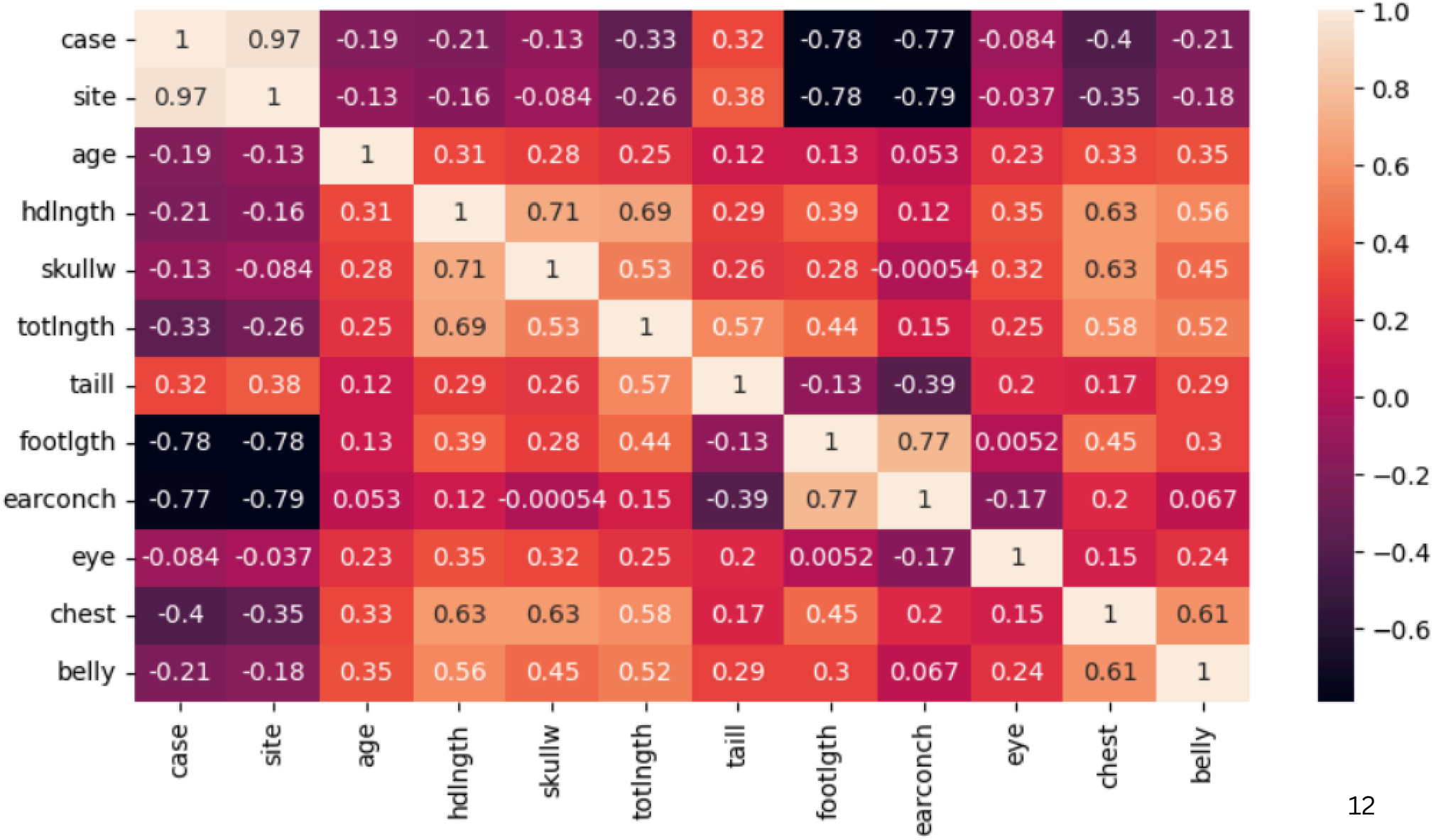
Exploratory Data Analysis



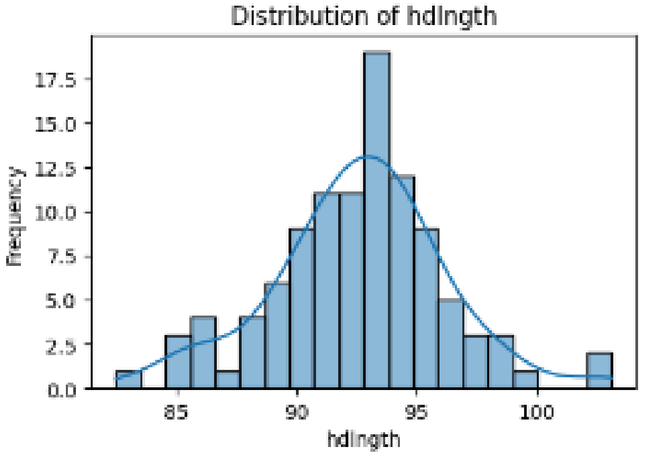


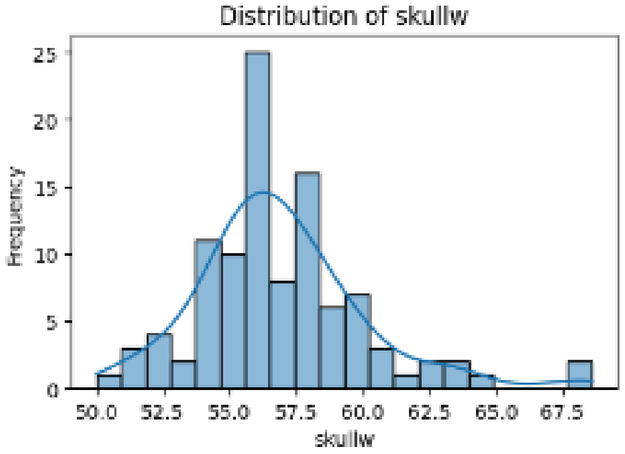


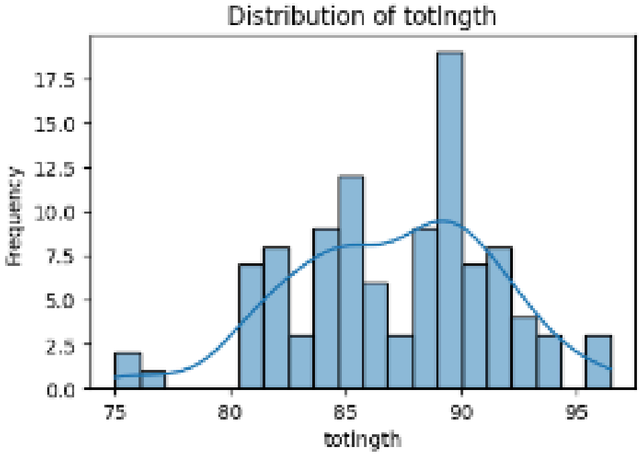
Correlation Matrix

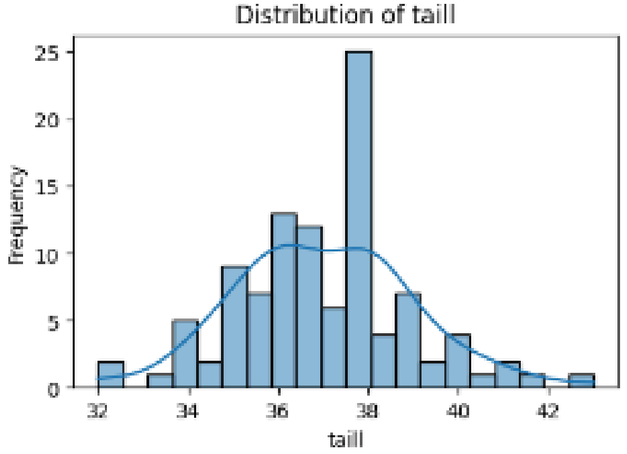


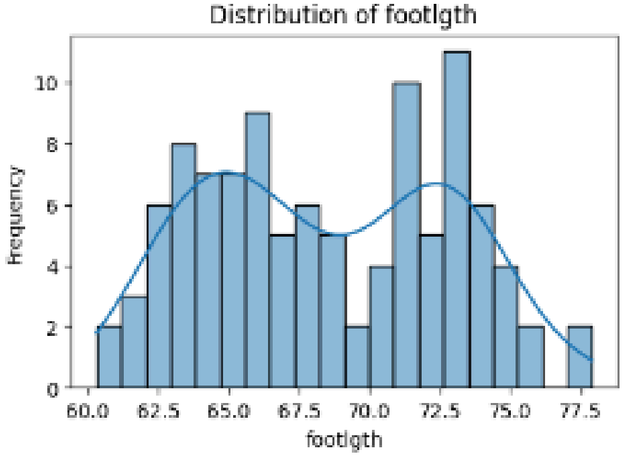
Histogram

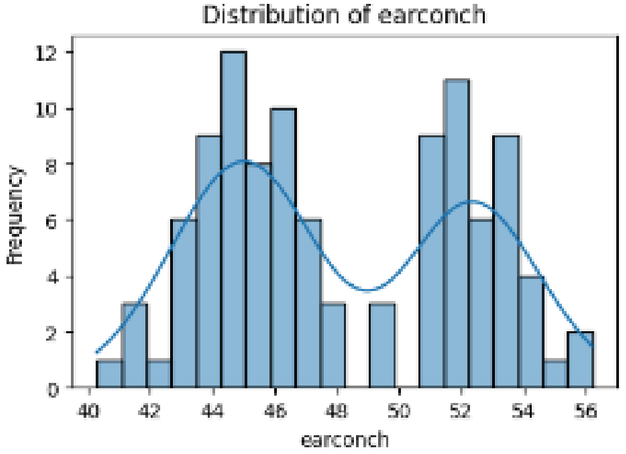

Distribution of hdlngth
Frequency
hdlngth

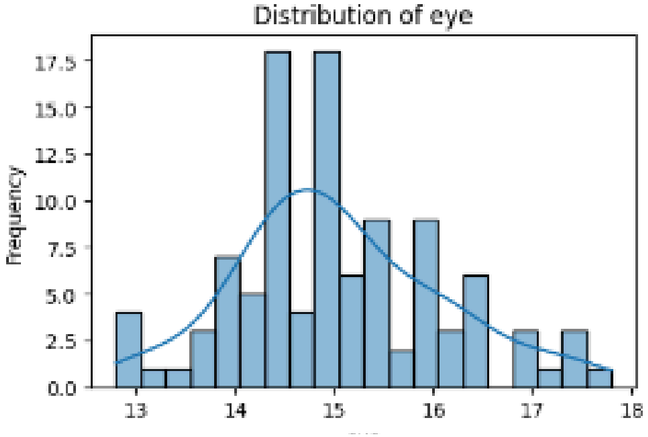

Distribution of skullw
Frequency
skullw

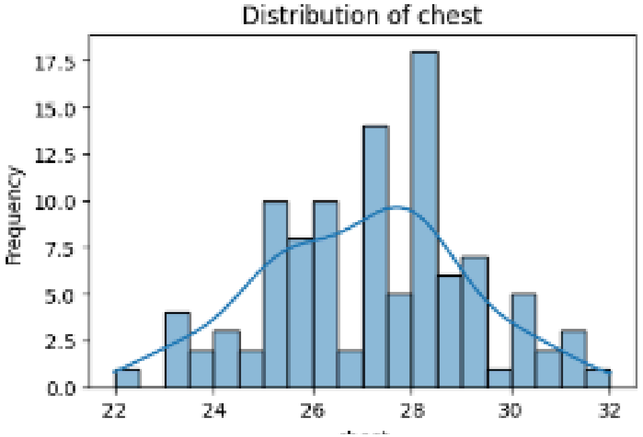

Distribution of totlngth
Frequency
totlngth

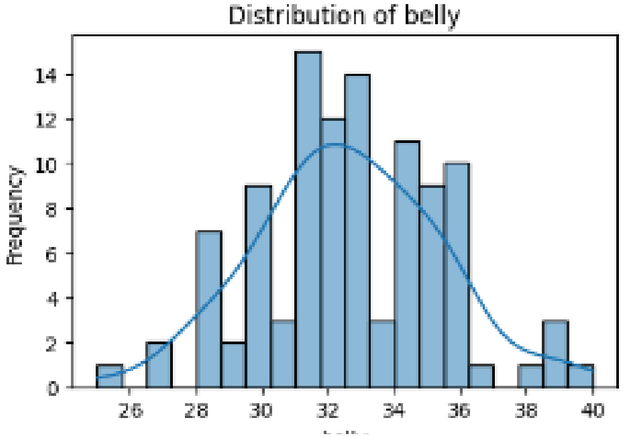

Distribution of taill
frequency
taill


Distribution of footlgth
Frequency
footlgth

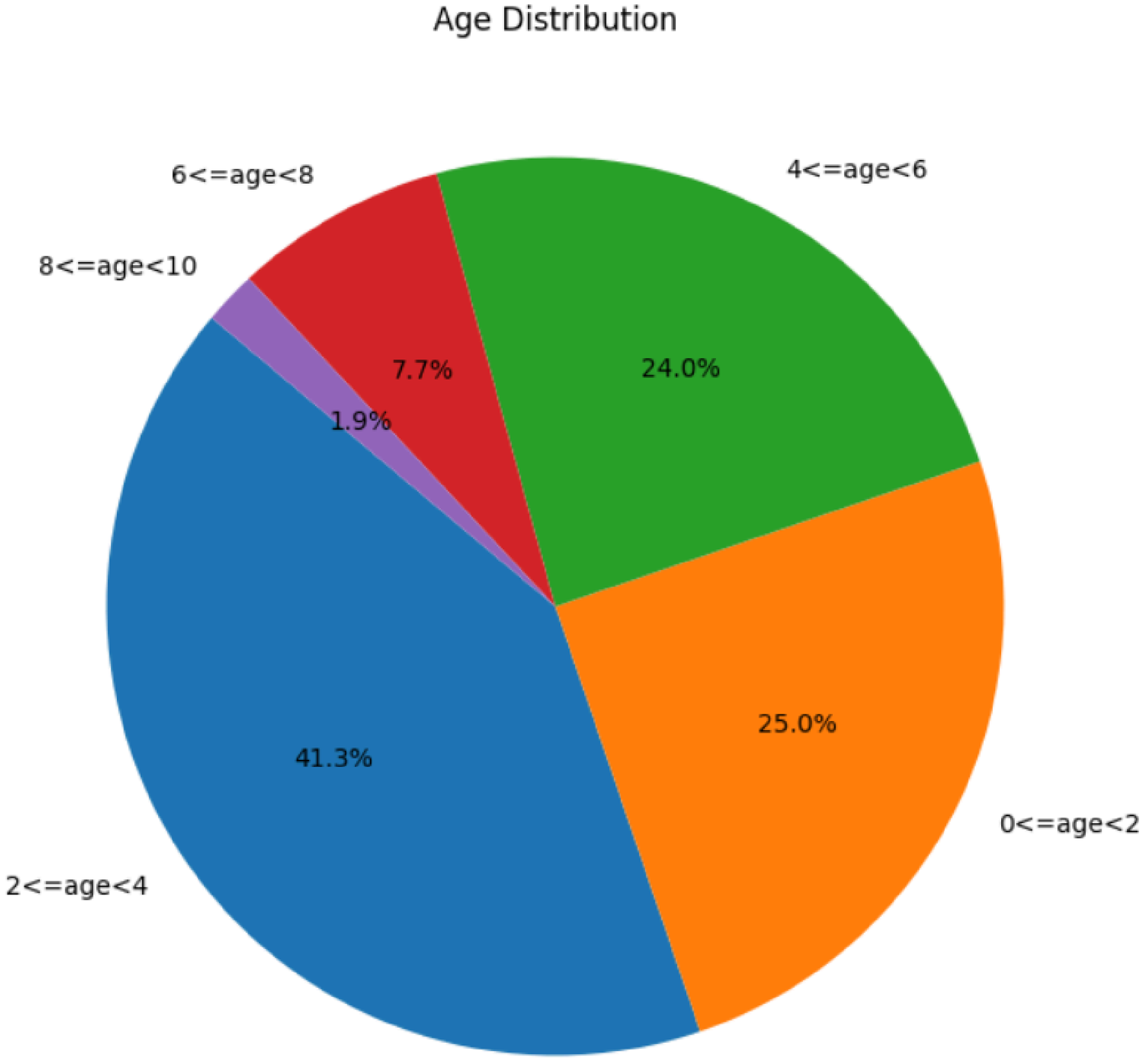

Distribution of earconch
Frequency
earconch


Distribution of eye
frequency

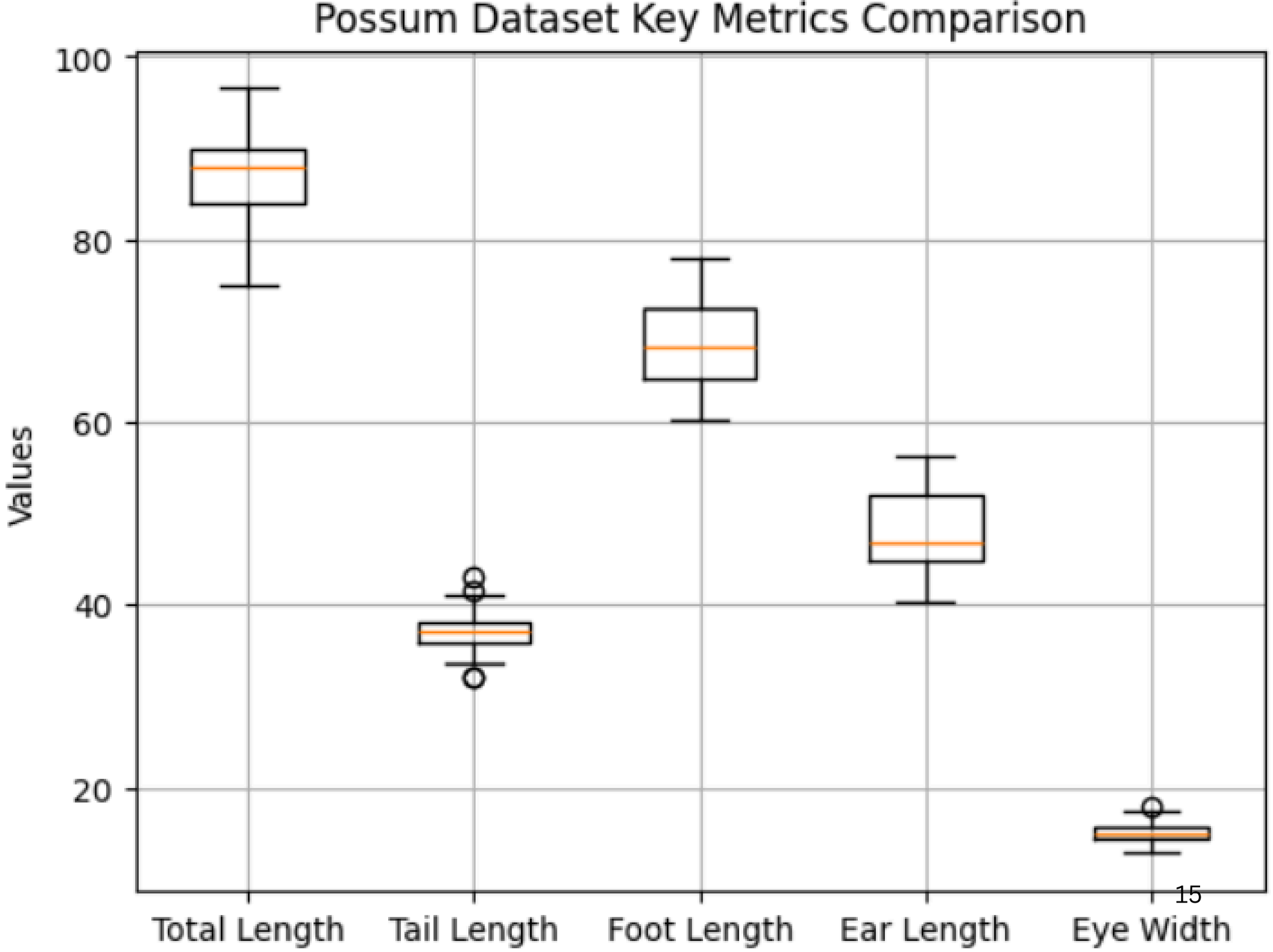

Distribution of chest
Frequency


Distribution of belly
Frequency

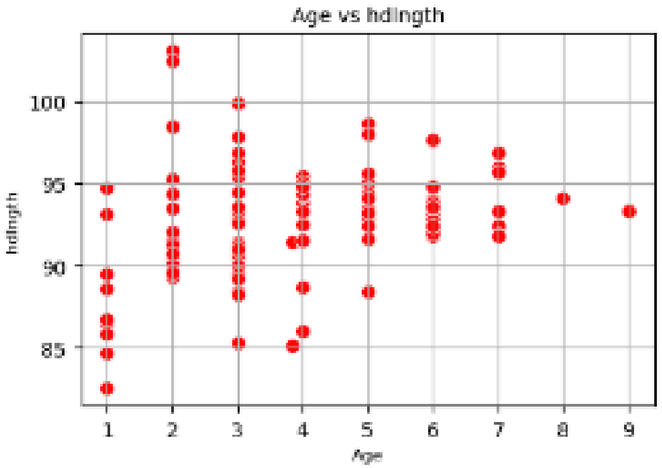
Pie chart

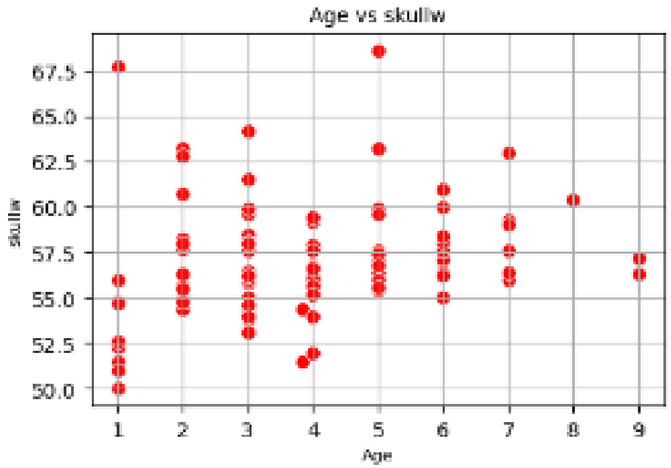

Age Distribution
6<=age<8
4<=age<6
8<=age<10
7.7%
24.0%
1.9%
25.0%
41.3%
0<=age<2
2<=age<4

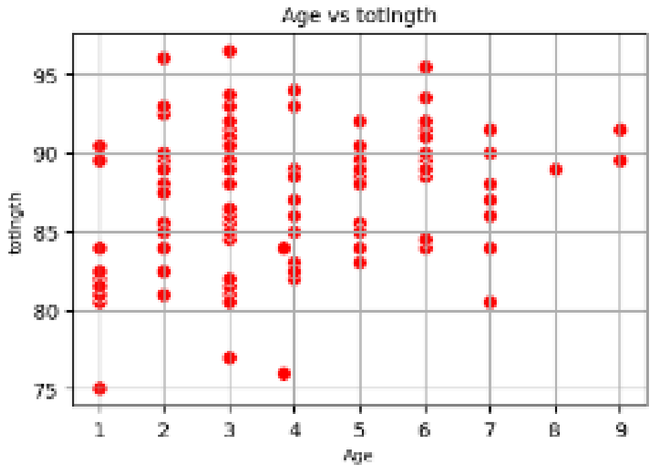
Box plot

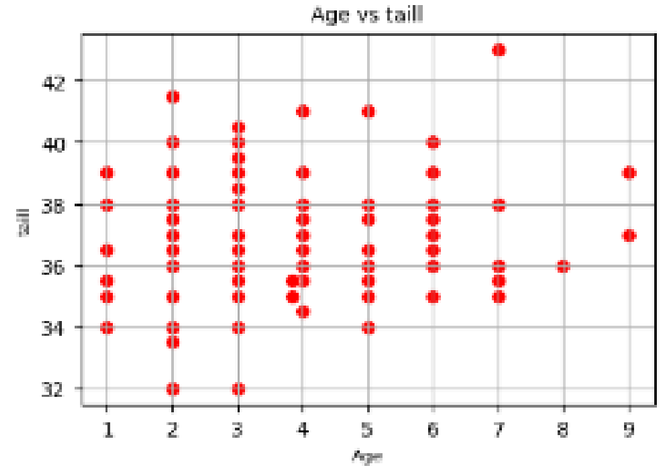

Possum Dataset Key Metrics Comparison
Values
100
80
60
40
20
Total Length
Tail Length
Foot Length
Ear Length
Eye Width

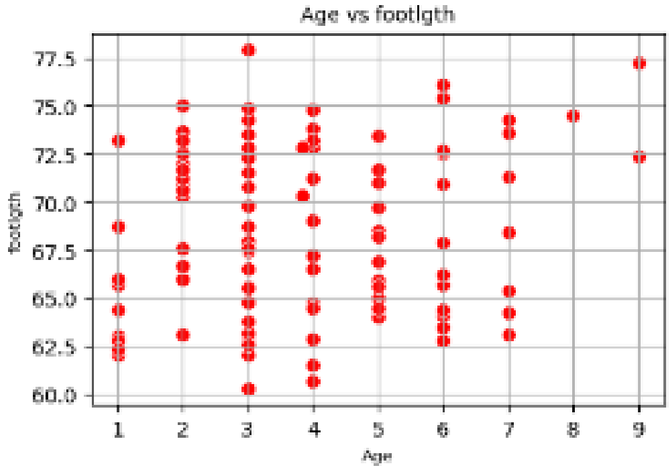
Scatter plot

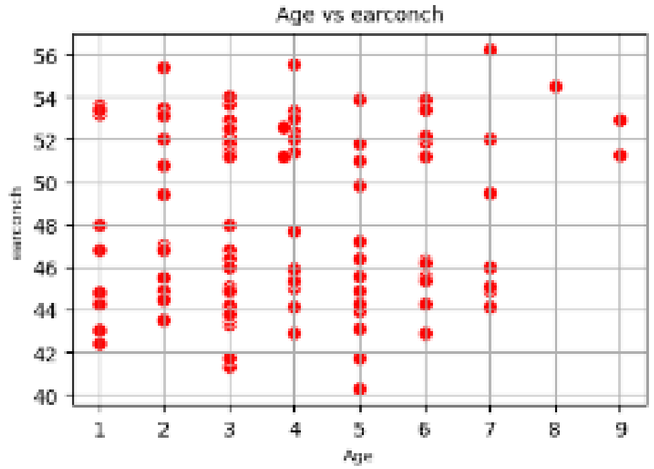


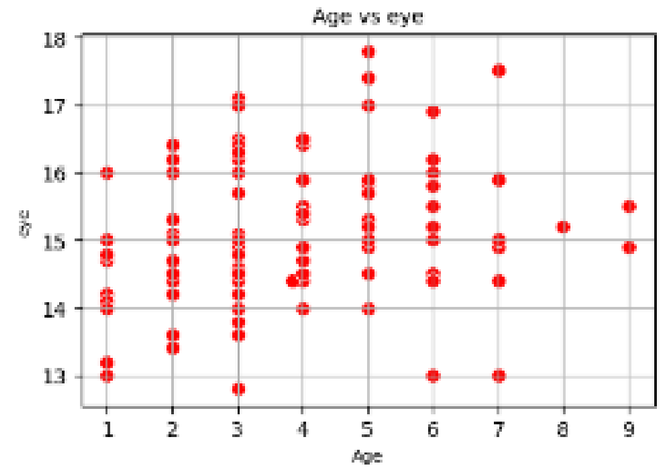


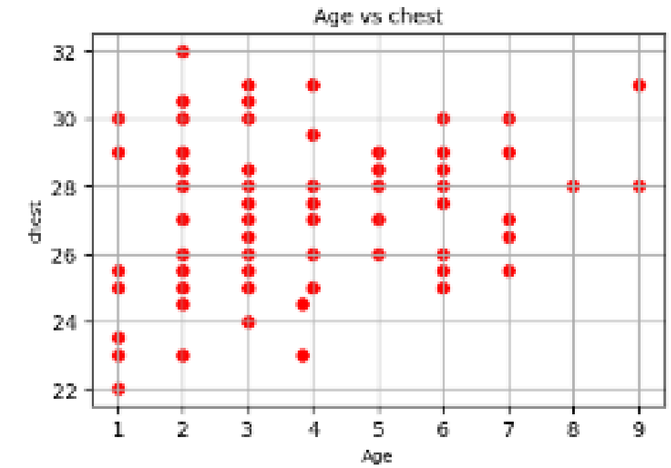




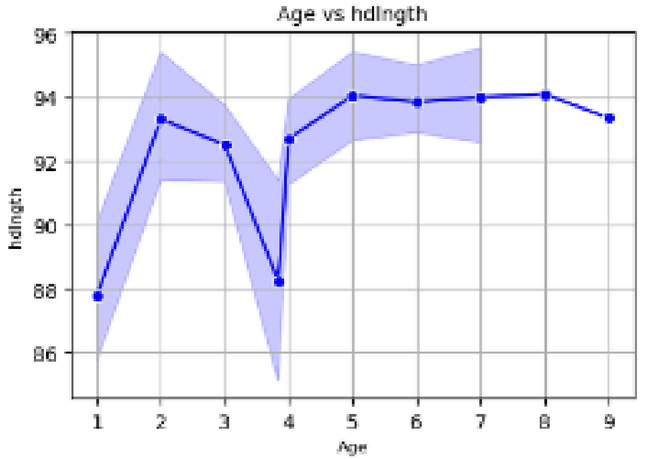


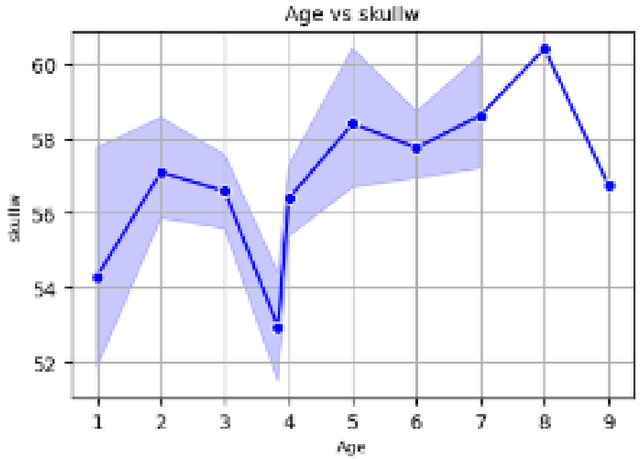


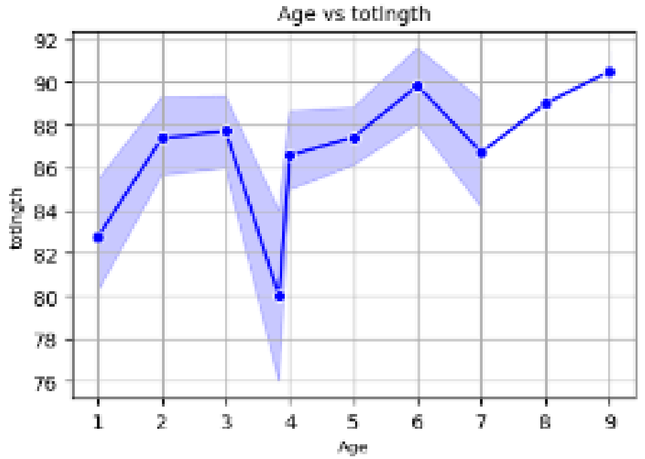


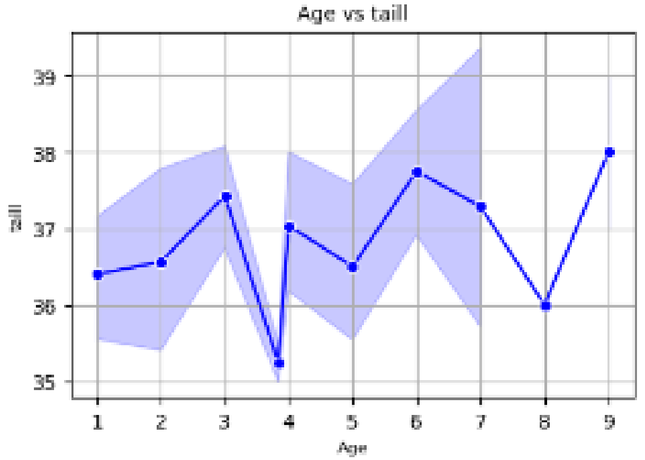


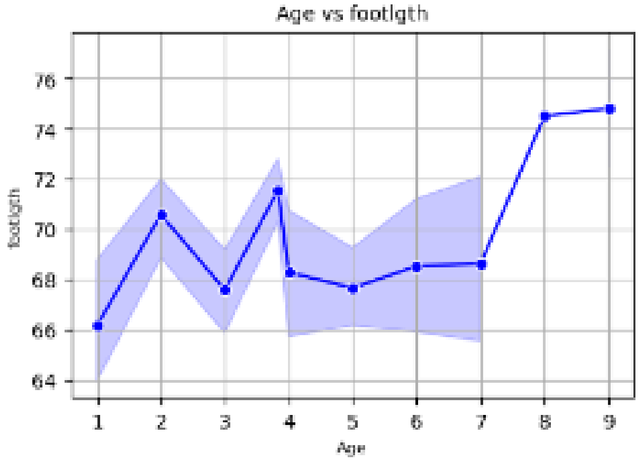
Line plot

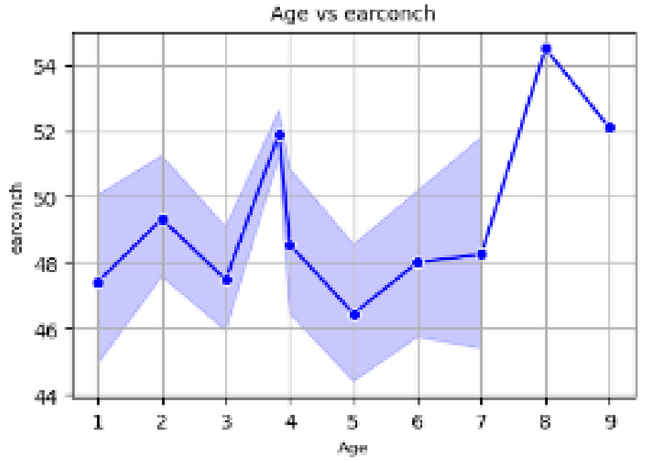

Age vs hdlngth
hdlngth
Age

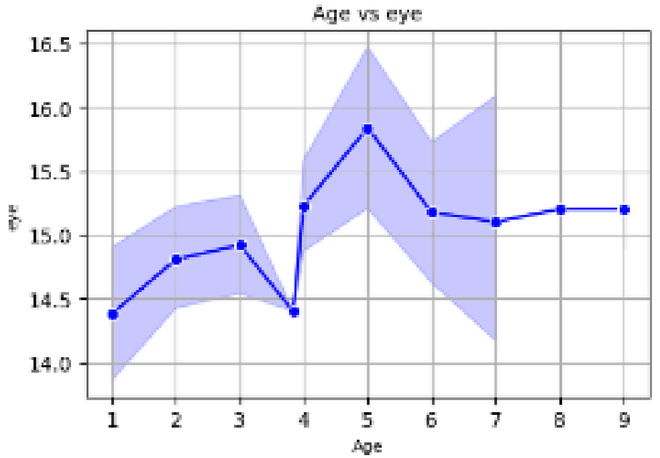

Age vs skullw
skullw
Age

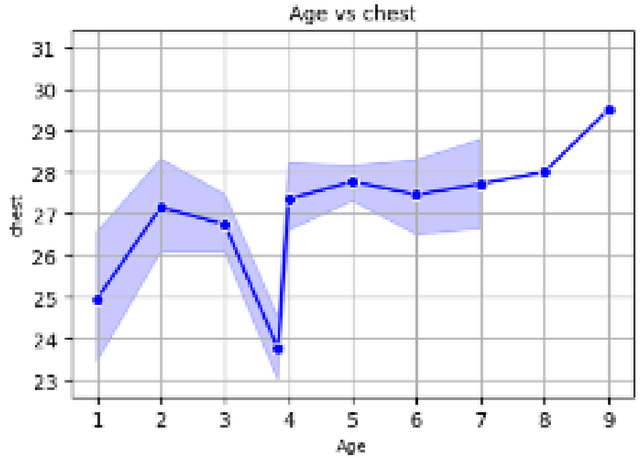

Age vs totlngth
totlngth
Age

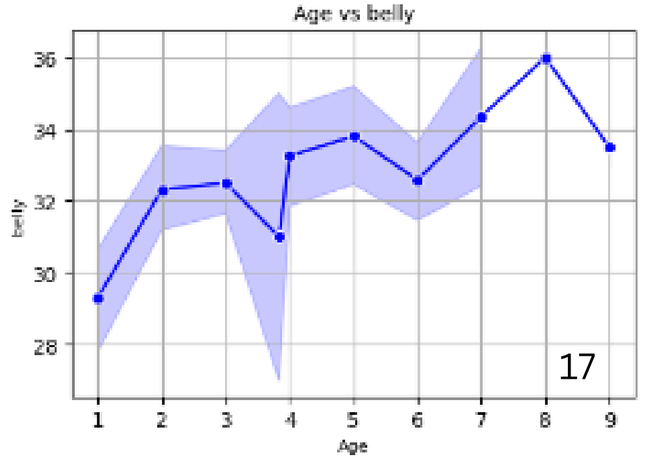

Age vs taill
taill
Age


Age vs footlgth
footlgth
Age


Age vs earconch
earconch
Age


Age vs eye
eye
Age


Age vs chest
chest
Age


Age vs belly
belly
Age

Multicollinearity check

	variables	VIF
0	case	85.9
1	site	80.3
2	hdlngth	2111.3
3	skullw	937.0
4	totlngth	1967.4
5	taill	975.7
6	footlgth	1036.3
7	earconch	384.4
8	eye	230.7
9	chest	486.9
10	belly	274.8

→

	variables	VIF
0	case	84.7
1	site	79.9
2	skullw	815.9
3	totlngth	1577.7
4	taill	925.4
5	footlgth	1018.2
6	earconch	356.7
7	eye	206.9
8	chest	469.7
9	belly	269.6

→

	variables	VIF
0	case	82.0
1	site	79.4
2	skullw	742.5
3	taill	513.4
4	footlgth	946.1
5	earconch	356.6
6	eye	206.5
7	chest	468.8
8	belly	265.7

→

	variables	VIF
0	case	81.6
1	site	75.6
2	skullw	712.4
3	taill	424.7
4	earconch	189.8
5	eye	206.3
6	chest	451.9
7	belly	265.6

	variables	VIF
0	case	79.2
1	site	75.4
2	taill	421.1
3	earconch	179.6
4	eye	174.7
5	chest	296.8
6	belly	258.9

	variables	VIF
0	case	79.2
1	site	71.7
2	earconch	150.8
3	eye	148.7
4	chest	280.1
5	belly	240.0

	variables	VIF
0	case	78.8
1	site	71.0
2	earconch	122.1
3	eye	143.8
4	belly	157.5

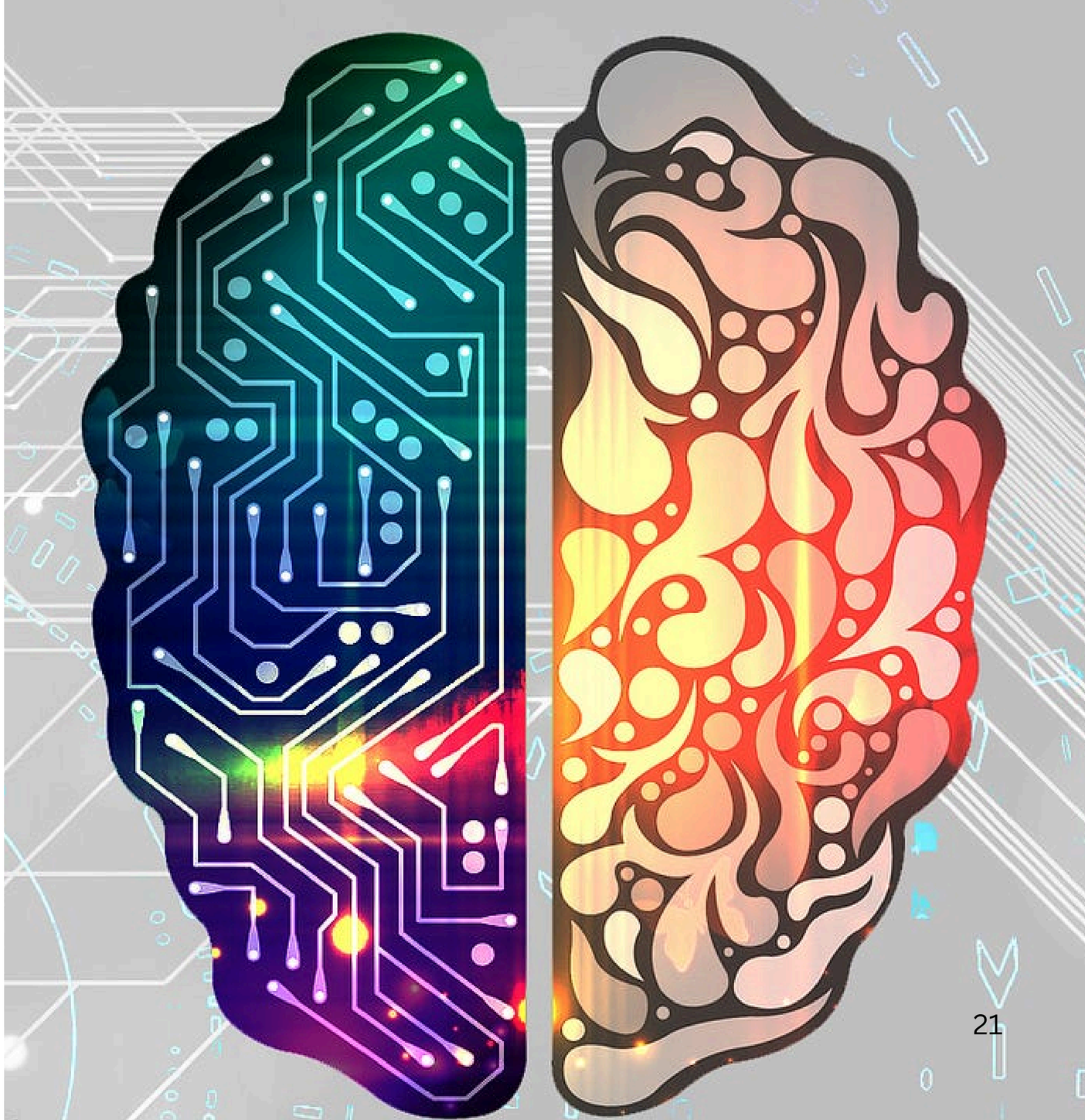
	variables	VIF
0	case	78.4
1	site	70.1
2	earconch	88.1
3	eye	103.0

	variables	VIF
0	case	78.1
1	site	67.2
2	earconch	3.8

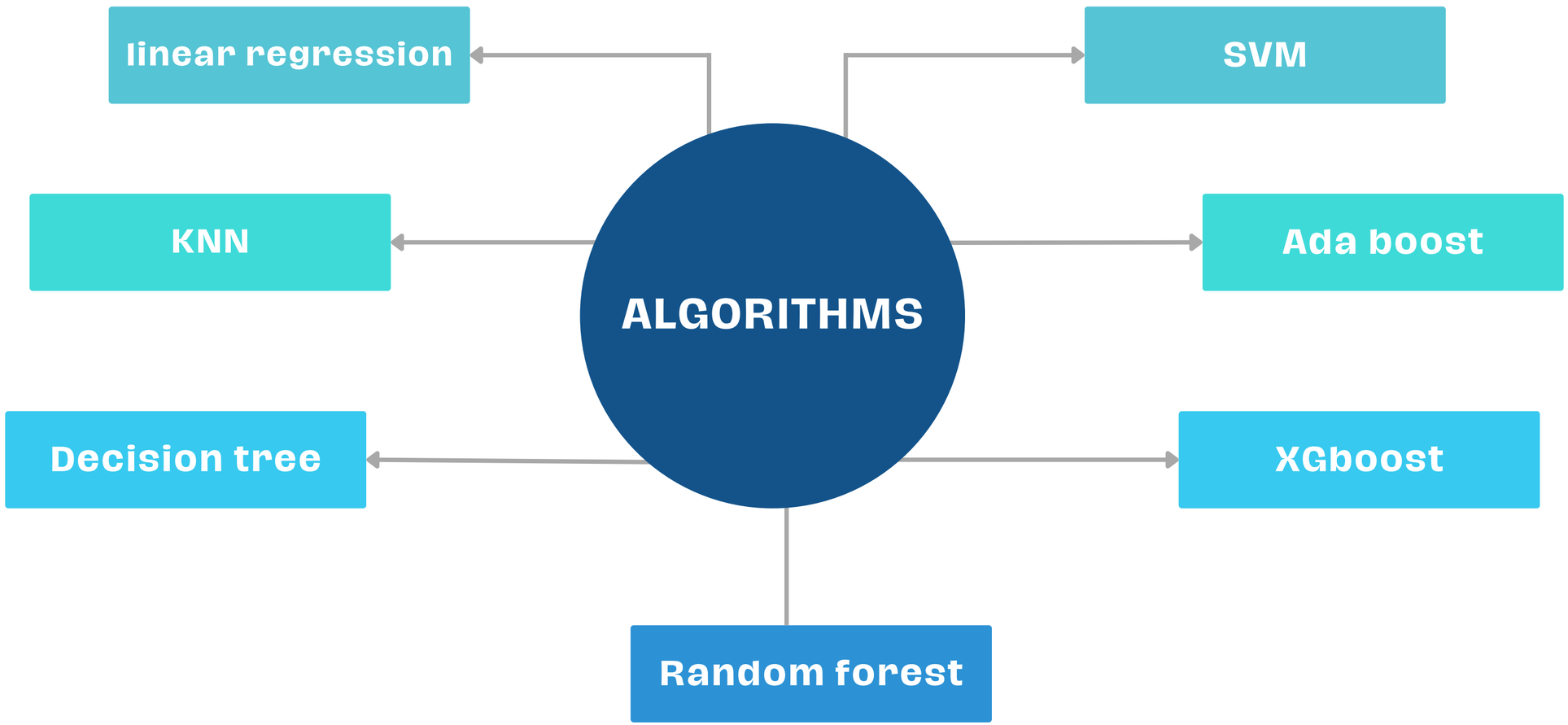
	variables	VIF
0	site	3.0
1	earconch	3.0

The test for multicollinearity is not suitable for this dataset since all the variables are highly correlated to each other.

Machine learning algorithms



MACHINE LEARNING ALGORITHMS USED



80-20 Train-test Split

algorithms	r2 score	MAE(Mean absolute error)
linear regression	0.104	1.525
KNN	0.016	1.665
Decision tree	-0.543	1.897
Random forest	0.143	1.459
Ada boost	0.109	1.413
SVM	0.016	1.664
XGboost	0.003	1.615

75-25 Train-test Split

algorithms	r2 score	MAE(Mean absolute error)
linear regression	0.148	1.450
KNN	0.034	1.618
Decision tree	0.172	1.455
Random forest	0.234	1.343
Ada boost	0.044	1.589
SVM	0.033	1.618
XGboost	0.139	1.455

70-30 Train-test Split

algorithms	r2 score	MAE(Mean absolute error)
linear regression	0.150	1.345
KNN	0.021	1.526
Decision tree	0.205	1.276
Random forest	0.309	1.222
Ada boost	0.430	0.975
SVM	0.022	1.526
XGboost	0.283	1.300

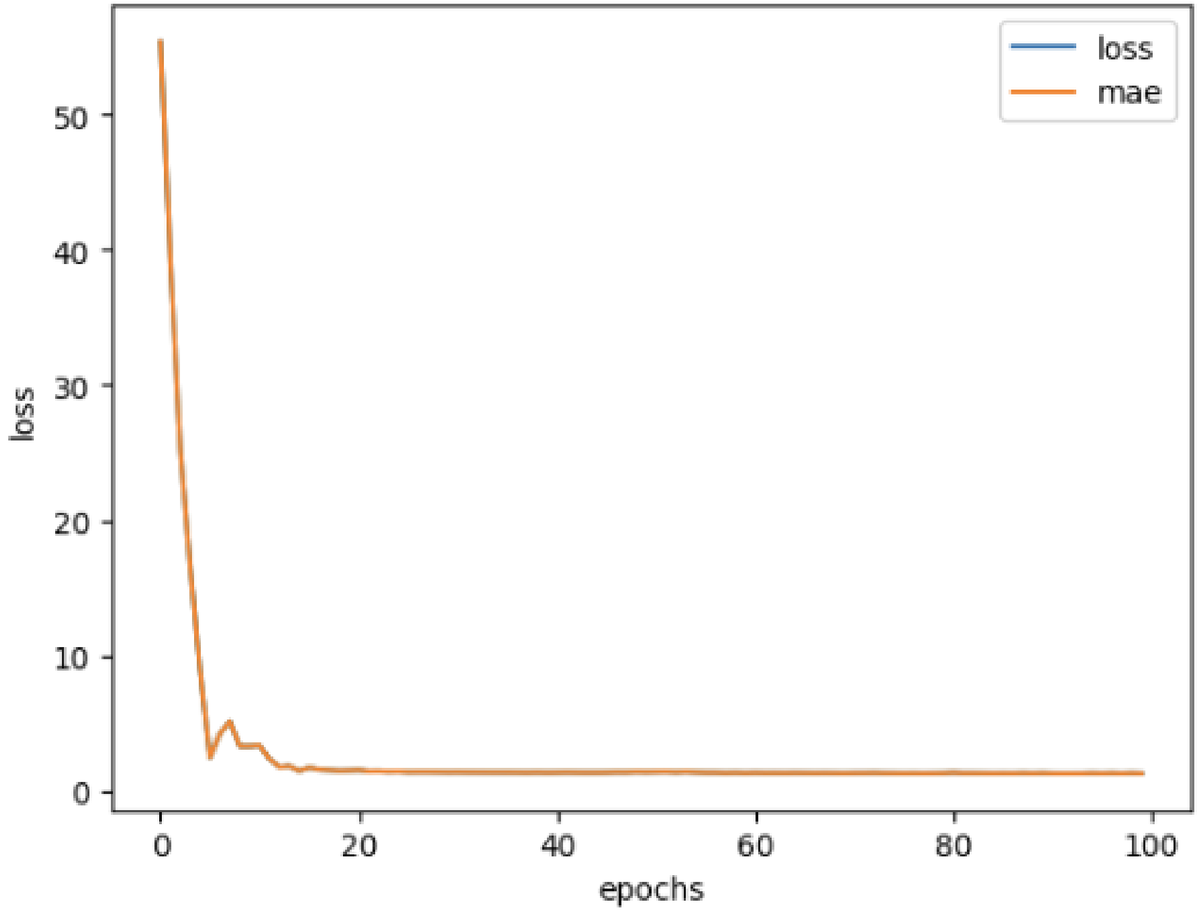
60-40 Train-test Split

algorithms	r2 score	MAE(Mean absolute error)
linear regression	0.072	1.531
KNN	0.037	1.623
Decision tree	-0.477	1.948
Random forest	0.198	3.304
Ada boost	0.098	1.462
SVM	0.027	1.620
XGboost	0.101	1.550

Neural Networks

Train-Test	Architecture	Optimizer	Epochs	MAE
60-40	64-32-16-1	Adam	100	1.535
70-30	64-32-16-1	Adam	100	1.2451
75-25	64-32-16-1	Adam	100	2.4261
80-20	64-32-16-1	Adam	100	1.5826

Neural Network plot for observation



Train-Test-Split	75-25
Architecture	64-32-16-1
Optimizer	Adam
Epochs	100

Best split
70-30 Train-test Split

algorithms	r2 score
linear regression	0.150
KNN	0.021
Decision tree	0.205
Random forest	0.309
Ada boost	0.430
SVM	0.022
XGboost	0.283

algorithms	MAE(Mean absolute error)
linear regression	1.345
KNN	1.526
Decision tree	1.276
Random forest	1.222
Ada boost	0.975
SVM	1.526
XGboost	1.300

summary

-The main aim of this research is to predict the age of the possum based on the body measurements of a possum and gender.
-After analysis we conclude that ,Adaboost is the best algorithm among others.The best split is 70-30 train-test split .

Future research directions

-Future research may focus on integrating molecular data with morphometric data to improve age estimation in mature animals.

-Given the rapid environmental changes, future research could explore how climate change affects possum morphology over time.

Insights

-By studying features, researchers can see how things like climate, predators, and food availability shape the way possums look and behave.

-Sexual dimorphism means that males and females look different in terms of size and body structure. These differences often relate to how they reproduce. For example, males may be larger or have stronger features to compete for mates, while females may have traits that help them care for their young.

work distribution

B.sruthi	**collecting information about possum and literature Review**
Amreen	**Data preprocessing and Exploratory Data Analysis**
B.saketh	**implementing machine learning algorithms**

colob notebook

Thank you

B.sruthi

Amreen

B.saketh

Appendix

Loading the dataset

```
[7] data=pd.read_csv('/content/possum.csv')
    data.head()
```

	case	site	Pop	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly
0	1	1	Vic	m	8.0	94.1	60.4	89.0	36.0	74.5	54.5	15.2	28.0	36.0
1	2	1	Vic	f	6.0	92.5	57.6	91.5	36.5	72.5	51.2	16.0	28.5	33.0
2	3	1	Vic	f	6.0	94.0	60.0	95.5	39.0	75.4	51.9	15.5	30.0	34.0
3	4	1	Vic	f	6.0	93.2	57.1	92.0	38.0	76.1	52.2	15.2	28.0	34.0
4	5	1	Vic	f	2.0	91.5	56.3	85.5	36.0	71.0	53.2	15.1	28.5	33.0

Null values

Checking for the datatype

Checking Null values

```
data.isna().sum()
```

	0
case	0
site	0
Pop	0
sex	0
age	2
hdlngth	0
skullw	0
totlngth	0
taill	0
footlgth	1
earconch	0
eye	0
chest	0
belly	0

```
[ ] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104 entries, 0 to 103
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   case      104 non-null    int64
 1   site      104 non-null    int64
 2   Pop       104 non-null    object
 3   sex       104 non-null    object
 4   age       102 non-null    float64
 5   hdlngth   104 non-null    float64
 6   skullw    104 non-null    float64
 7   totlngth  104 non-null    float64
 8   taill     104 non-null    float64
 9   footlgth  103 non-null    float64
 10  earconch  104 non-null    float64
 11  eye       104 non-null    float64
 12  chest     104 non-null    float64
 13  belly     104 non-null    float64
dtypes: float64(10), int64(2), object(2)
memory usage: 11.5+ KB
```

```
for i in range(data.shape[1]):
  print(data.iloc[:,i].unique())
  print(data.iloc[:,i].value_counts())
```

```
[  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104]
case
1      1
2      1
77     1
76     1
75     1
      ..
32     1
31     1
30     1
29     1
104    1
Name: count, Length: 104, dtype: int64
[1 2 3 4 5 6 7]
```

Replacing missing values

```
[11] data.isna().sum()
```

	0
case	0
site	0
Pop	0
sex	0
age	0
hdlngth	0
skullw	0
totlngth	0
taill	0
footlgth	0
earconch	0
eye	0
chest	0
belly	0

Replacing missing values

```
[10] continuous_columns = data.select_dtypes(include=['float64', 'int64']).columns
     # Replace null values in continuous columns with the mean
     for column in continuous_columns:
         mean_value = data[column].mean()
         data[column].fillna(mean_value, inplace=True)
```

```
[12] df=data
     df.iloc[:,4].value_counts()
```

age	count
3.000000	27
2.000000	16
4.000000	14
5.000000	13
6.000000	12
1.000000	10
7.000000	7
9.000000	2
3.833333	2
8.000000	1

```
[14] df.describe()
```

	case	site	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly
count	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000	104.000000
mean	52.500000	3.625000	3.833333	92.602885	56.883654	87.088462	37.009615	68.459223	48.130769	15.046154	27.000000	32.586538
std	30.166206	2.349086	1.890617	3.573349	3.113426	4.310549	1.959518	4.373917	4.109380	1.050374	2.045597	2.761949
min	1.000000	1.000000	1.000000	82.500000	50.000000	75.000000	32.000000	60.300000	40.300000	12.800000	22.000000	25.000000
25%	26.750000	1.000000	2.750000	90.675000	54.975000	84.000000	35.875000	64.650000	44.800000	14.400000	25.500000	31.000000
50%	52.500000	3.000000	3.000000	92.800000	56.350000	88.000000	37.000000	68.100000	46.800000	14.900000	27.000000	32.500000
75%	78.250000	6.000000	5.000000	94.725000	58.100000	90.000000	38.000000	72.500000	52.000000	15.725000	28.000000	34.125000
max	104.000000	7.000000	9.000000	103.100000	68.600000	96.500000	43.000000	77.900000	56.200000	17.800000	32.000000	40.000000

Dropping target variable

Dropping the column 'age' and assigning it as target variable

```
X=df.drop(['age'],axis=1)
print(X)
y=df['age']
print(y)
```

```
     case  site    Pop sex  hdlngth  skullw  totlngth  taill  footlgth  \
0       1     1    Vic   m     94.1    60.4      89.0   36.0      74.5
1       2     1    Vic   f     92.5    57.6      91.5   36.5      72.5
2       3     1    Vic   f     94.0    60.0      95.5   39.0      75.4
3       4     1    Vic   f     93.2    57.1      92.0   38.0      76.1
4       5     1    Vic   f     91.5    56.3      85.5   36.0      71.0
..    ...   ...    ...  ..      ...     ...       ...    ...       ...
99    100     7  other   m     89.5    56.0      81.5   36.5      66.0
100   101     7  other   m     88.6    54.7      82.5   39.0      64.4
101   102     7  other   f     92.4    55.0      89.0   38.0      63.5
102   103     7  other   m     91.5    55.2      82.5   36.5      62.9
103   104     7  other   f     93.6    59.9      89.0   40.0      67.6

     earconch   eye  chest  belly
0        54.5  15.2   28.0   36.0
1        51.2  16.0   28.5   33.0
2        51.9  15.5   30.0   34.0
3        52.2  15.2   28.0   34.0
4        53.2  15.1   28.5   33.0
..        ...   ...    ...    ...
99       46.8  14.8   23.0   27.0
100      48.0  14.0   25.0   33.0
101      45.4  13.0   25.0   30.0
102      45.9  15.4   25.0   29.0
103      46.0  14.8   28.5   33.5
```

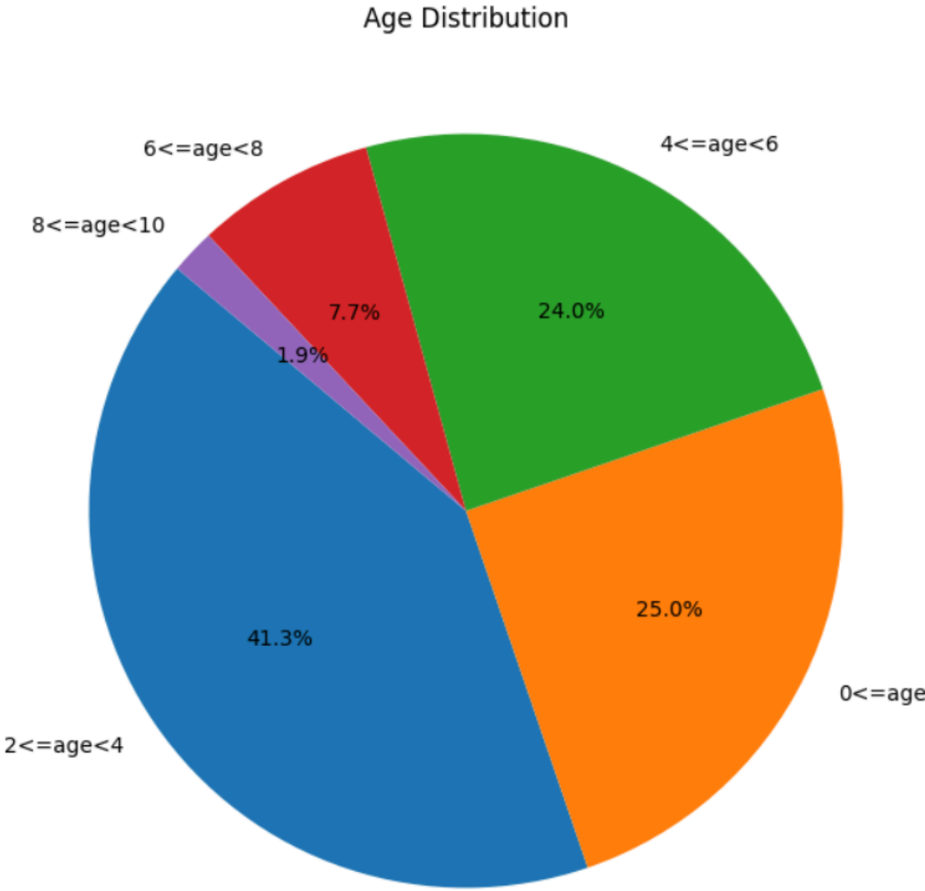
Dummification

```
X=pd.get_dummies(X,dtype='int',drop_first=True)
print(X)
```

```
     case  site  hdlngth  skullw  totlngth  taill  footlgth  earconch   eye  \
0       1     1     94.1    60.4      89.0   36.0      74.5      54.5  15.2
1       2     1     92.5    57.6      91.5   36.5      72.5      51.2  16.0
2       3     1     94.0    60.0      95.5   39.0      75.4      51.9  15.5
3       4     1     93.2    57.1      92.0   38.0      76.1      52.2  15.2
4       5     1     91.5    56.3      85.5   36.0      71.0      53.2  15.1
..    ...   ...      ...     ...       ...    ...       ...       ...   ...
99    100     7     89.5    56.0      81.5   36.5      66.0      46.8  14.8
100   101     7     88.6    54.7      82.5   39.0      64.4      48.0  14.0
101   102     7     92.4    55.0      89.0   38.0      63.5      45.4  13.0
102   103     7     91.5    55.2      82.5   36.5      62.9      45.9  15.4
103   104     7     93.6    59.9      89.0   40.0      67.6      46.0  14.8

     chest  belly  Pop_other  sex_m
0     28.0   36.0          0      1
1     28.5   33.0          0      0
2     30.0   34.0          0      0
3     28.0   34.0          0      0
4     28.5   33.0          0      0
..     ...    ...        ...    ...
99    23.0   27.0          1      1
100   25.0   33.0          1      1
101   25.0   30.0          1      0
102   25.0   29.0          1      1
103   28.5   33.5          1      0

[104 rows x 13 columns]
```

Pie chart

```
#pie chart
# Define bins and labels for age categories
df=pd.DataFrame(data)
bins = [0, 2, 4, 6, 8, 10]  # Define age ranges
labels = ['0<=age<2', '2<=age<4', '4<=age<6', '6<=age<8', '8<=age<10']  # Define corresponding labels

# Bin the age data
df['age_category'] = pd.cut(df['age'], bins=bins, labels=labels, right=True)

# Count the number of occurrences in each age category
age_counts = df['age_category'].value_counts()
# Plot the pie chart
plt.figure(figsize=(8, 8))
plt.pie(age_counts, labels=age_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Age Distribution')
plt.show()
```



Multicollinearity check

```
import pandas as pd
import seaborn as sns
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[ ]  # Import library for VIF
     from statsmodels.stats.outliers_influence import variance_inflation_factor

     def calc_vif(X):

         # Calculating VIF
         vif = pd.DataFrame()
         vif["variables"] = X.columns
         vif["VIF"] = [variance_inflation_factor(X.values, i).round(1) for i in range(X.shape[1])]

         return(vif)

     calc_vif(X)
```

OT:

	variables	VIF
0	site	2.8
1	earconch	2.8

Linear Regression

LINEAR REGRESSION

```
### SCIKIT-LEARN ###
#multiple regression
# create X and y
feature_cols = ['case','site','hdlngth', 'skullw', 'totlngth', 'taill', 'footlgth', 'earconch', 'eye', 'chest', 'belly']
X = data[feature_cols]
y = data.age

# instantiate and fit
lm2 = LinearRegression()
lm2.fit(X, y)

# print the coefficients
print(lm2.intercept_)
print(lm2.coef_)
```

```
-5.20235066804844
[-0.05649674  0.5077668   0.0796396   0.04261819 -0.06877445  0.08789572
 -0.0889329   0.01835949  0.13979907  0.05951186  0.14094195]
```

```
from sklearn.metrics import r2_score
r2_score(y_test4,y_pred4)
```

```
0.01632530404080046
```

```
from sklearn import metrics
metrics.mean_absolute_error(y_test4,y_pred4)
```

```
1.6644444444444444
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test4,y_pred4)
```

```
4.422198941798943
```

```
mse = mean_squared_error(y_test4, y_pred4)
rmse = np.sqrt(mse)
rmse
```

```
2.1029025041116247
```

```
from sklearn.metrics import mean_absolute_percentage_error
mape = mean_absolute_percentage_error(y_test4, y_pred4)
print(mape)
mape = mape * 100
mape
```

```
0.4891007920076242
48.91007920076242
```

KNN

KNN

60-40

```
[ ] from sklearn.neighbors import KNeighborsRegressor
```

```
[ ] model=KNeighborsRegressor(n_neighbors=25)
```

```
[ ] model.fit(X_train1, y_train1)
```

```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=25)
```

```
[ ] y_pred1 = model.predict(X_test1)
```

```
[ ] y_pred1
```

```
array([3.87333333, 4.04      , 4.        , 4.2       , 3.63333333,
       3.72      , 3.72      , 3.87333333, 3.95333333, 3.95333333,
       3.95333333, 3.96      , 3.72      , 3.72      , 3.64      ,
       3.95333333, 3.8       , 3.83333333, 3.95333333, 3.83333333,
       3.72      , 4.2       , 3.95333333, 3.76      , 3.63333333,
       3.8       , 3.95333333, 3.63333333, 3.59333333, 3.95333333,
       3.72      , 3.95333333, 3.72      , 3.64      , 3.83333333,
       3.44      , 3.95333333, 4.        , 4.12      , 3.59333333,
       3.55333333, 3.96      ])
```

Random Forest

RANDOM FOREST

60-40

```
[ ] from sklearn.ensemble import RandomForestRegressor
    from sklearn.tree import plot_tree
```

```
[ ] rf=RandomForestRegressor()
```

```
[ ] rf.fit(X_train1,y_train1)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
[ ] y_pred1=rf.predict(X_test1)
    y_pred1
```

```
array([4.22666667, 3.48      , 4.85      , 3.23      , 2.49333333,
       4.5       , 2.64      , 4.48      , 5.05      , 5.24      ,
       3.25      , 3.92      , 4.32      , 3.73      , 2.71333333,
       5.54833333, 3.38      , 2.62666667, 4.19      , 2.22      ,
       5.        , 3.92      , 4.6       , 4.34      , 3.035     ,
       2.79666667, 3.74      , 2.92833333, 3.18833333, 5.58833333,
       3.27      , 4.28      , 4.56      , 2.75666667, 3.62666667,
       2.69666667, 4.77833333, 4.3       , 3.36      , 2.755     ,
       3.69833333, 4.02      ])
```

Boosting

BOOSTING

XGboost

```
[ ] import xgboost as xgb
```

60-40

```
[ ] model1 = xgb.XGBRegressor()
    model2 = xgb.XGBRegressor(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)

    train_model1 = model1.fit(X_train1, y_train1)
    train_model2 = model2.fit(X_train1, y_train1)
```

```
[ ] pred1 = train_model1.predict(X_test1)
    pred2 = train_model2.predict(X_test1)
```

EVALUATION METRICS

```
[ ] print("RMSE1:",np.sqrt(metrics.mean_squared_error(y_test1, pred1)))
    print("RMSE2:",np.sqrt(metrics.mean_squared_error(y_test1, pred2)))
    print("R2 score1:",metrics.r2_score(y_test1,pred1))
    print("R2 score2:",metrics.r2_score(y_test1,pred2))
```

```
RMSE1: 1.7318548517013523
RMSE2: 1.925143300542351
R2 score1: 0.2723557167707735
R2 score2: 0.10087046232640773
```

```
[ ] mae1=metrics.mean_absolute_error(y_test1,pred1)
    mae2=metrics.mean_absolute_error(y_test1,pred2)
    print("MAE1:",mae1)
    print("MAE2:",mae2)
```

```
MAE1: 1.398267458355616
MAE2: 1.5501069530608165
```

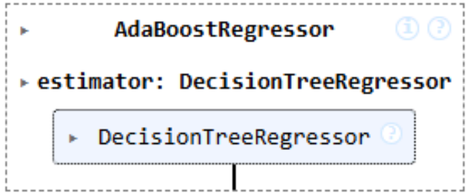
AdaBoost

```
[ ] from sklearn.ensemble import AdaBoostRegressor
```

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

# Replace 'base_estimator' with 'estimator'
base_estimator = DecisionTreeRegressor(max_depth=3, random_state=0)
adaboost = AdaBoostRegressor(estimator=base_estimator, # Changed argument name here
                             n_estimators=3,random_state=0)
```

60-40

```
[ ] adaboost.fit(X_train1, y_train1)
```



Decision Tree

Decision Tree

60-40

```
[ ] from sklearn.tree import DecisionTreeRegressor
```

```
[ ] reg = DecisionTreeRegressor()
    reg = reg.fit(X_train1,y_train1)
```

```
[ ] y_pred1 = reg.predict(X_test1)
```

Evaluation Metric

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
-0.4634800447736678
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
1.8531746031746035
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test1,y_pred1)
```

```
6.032407407407407
```

```
[ ] mse = mean_squared_error(y_test1, y_pred1)
    rmse = np.sqrt(mse)
    rmse
```

```
2.4560959686883996
```

SVM

SVM

60-40

```
[ ] from sklearn.svm import SVR
```

```
[ ] model = SVR(kernel='linear')
```

```
[ ] model.fit(X_train1, y_train1)
```

```
SVR(kernel='linear')
```

```
[ ] y_pred1 = model.predict(X_test1)
    y_pred1
```

```
array([3.78349348, 4.41898078, 3.66060923, 5.09121879, 0.86563047,
       3.10057874, 1.95960846, 4.06685823, 4.37329096, 5.52784653,
       3.42793154, 2.29967115, 3.92479408, 2.55413257, 3.09316111,
       6.03890767, 3.35093645, 2.57073635, 4.88493594, 2.38128164,
       2.84346003, 4.08931164, 3.38687573, 4.16790598, 1.75766679,
       1.81515822, 3.53821012, 1.1654959 , 2.66874514, 5.56223427,
       3.991962  , 4.67050832, 1.96700702, 2.83582736, 1.03266035,
       3.6287386 , 5.15237627, 4.71475792, 5.29207748, 1.97820386,
       1.76123744, 4.38749148])
```

```
[ ] svm = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
    svm
```

	Predicted	Actual
30	3.783493	3.000000
65	4.418981	3.000000
64	3.660609	5.000000
53	5.091219	7.000000
45	0.865630	3.833333
93	3.100579	7.000000
91	1.959608	2.000000

Neural Networks

60-40

```python
tf.random.set_seed(42)

# STEP1: Creating the model
model1 = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation="relu", input_shape=(11,)),  # Input layer with appropriate shape
    tf.keras.layers.Dense(32, activation="relu"),  # Additional hidden layer
    tf.keras.layers.Dense(16, activation="relu"),  # Deeper learning for hierarchical patterns
    tf.keras.layers.Dense(1)  # Output layer for regression
])

# STEP2: Compiling the model
model1.compile(
    loss=tf.keras.losses.mae,  # MAE remains effective for regression tasks
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),  # Improved learning rate specification
    metrics=["mae"]
)

# STEP3: Fit the model
history1 = model1.fit(X_train1, y_train1, epochs=100, verbose=1)
```

```
Epoch 1/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 3.0834 - mae: 3.0834
Epoch 2/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.3179 - mae: 2.3179
Epoch 3/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.8902 - mae: 1.8902
Epoch 4/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 2.0042 - mae: 2.0042
Epoch 5/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 1.5594 - mae: 1.5594
Epoch 6/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1.7213 - mae: 1.7213
Epoch 7/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 1.3848 - mae: 1.3848
Epoch 8/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.5795 - mae: 1.5795
Epoch 9/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.4635 - mae: 1.4635
Epoch 10/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.4710 - mae: 1.4710
```