Data Visualization Analysis and Statistical Methods for Populations of Possums

Tejavarma Chekuri

Department of Mathematical Science

MA-541 Statistical Methods

Prof.Hadi Safari Katesari

May 6th, 2023

Abstract:

With the use of statistical analysis, which is a scientific instrument, enormous volumes of data may be gathered, analyzed, and turned into useful information by spotting common patterns and trends. The idea is to employ a dataset on which different statistical techniques are used in order to make precise predictions. We were able to categorize the data with 95.2% accuracy using the projects accuracy score.

Introduction:

The goal of this projects introduction is to put into practice the statistical techniques that were used to analyze actual data. Z-test, ANOVA Analysis, Chi Squared Tests of Independence different distributions, categorical analysis of data, and regression models are some examples of this but not all of them.

We will make use of the Possum dataset to do this. First, we will use a variety of statistical techniques to track dependencies and correlations between the different data columns. To choose the best features from our data to create a logistic regression model, I will next use the Forward, Backward, and Recursive elimination procedures. On the basis of our test data, we will assess the precision of our logistic regression model.

Data Description:

Data originally found in the DAAG R package and used in the book Maindonald, J.H. and Braun, W.J. (2003, 2007, 2010).

This data appears to describe physical characteristics of Possum animal, possibly with different attributes depending on the 'site' and 'Pop' they belong to.

The features include body parts and measurments such as hdlngth, skullw, totlngth, taill, footlgth, earconch, eye, chest and belly.

Other related observations such as age, sex and population.

Loading The Required Libraries:

import numpy as np import pandas as pd

import matplotlib.pyplot as plot import seaborn as sns from scipy import stats

from scipy.stats import norm from scipy.stats import f_oneway

from statsmodels.stats.multicomp import pairwise_tukeyhsd from scipy.stats import shapiro

from scipy.stats import chi2_contingency from sklearn.feature_selection import chi2

from sklearn.model_selection import train_test_split import statsmodels.api as sm

from sklearn.linear_model import LinearRegression from sklearn.metrics import mean_squared_error

from sklearn.linear_model import LogisticRegression from mpl_toolkits.mplot3d import Axes3D

from sklearn.metrics import accuracy_score, f1_score from sklearn.model_selection import StratifiedKFold, KFold

from mlxtend.feature_selection import SequentialFeatureSelector as SFS from sklearn.ensemble import RandomForestClassifier as RFC

from sklearn.linear_model import Lasso, Ridge from sklearn.feature_selection import RFE

from sklearn.decomposition import PCA from sklearn.preprocessing import StandardScaler from sklearn.metrics import r2_score

from sklearn.preprocessing import PolynomialFeatures from scipy.stats import chi2

df = pd.read_csv("C:/Users/Haneesh/Downloads/Possum/possum.csv")

Looking at the head/sample of the dataset, we can see that there are 12 columns that we can use to predict the population of the possum. Since we want to predict the population of the Possum, which is a categorical variable.Consequently, we will fit a logistic regression model.

	case	site	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch
1	1	1	m	8.0	94.1	60.4	89.0	36.0	74.5	54.5
2	2	1	f	6.0	92.5	57.6	91.5	36.5	72.5	51.2
3	3	1	f	6.0	94.0	60.0	95.5	39.0	75.4	51.9
4	4	1	f	6.0	93.2	57.1	92.0	38.0	76.1	52.2
5	5	1	f	2.0	91.5	56.3	85.5	36.0	71.0	53.2

	eye	chest	belly	Pop
1	15.2	28.0	36.0	Vic
2	16.0	28.5	33.0	Vic
3	15.5	30.0	34.0	Vic
4	15.2	28.0	34.0	Vic
5	15.1	28.5	33.0	Vic

Data Preprocessing:

Checking for null value and dropping them from the data set

df.isnull().sum()

case 0 site 0 sex 0 age 2 hdlngth 0 skullw 0 totlngth 0 taill 0 footlgth 1 earconch 0 eye 0 chest 0 belly 0 Pop 0 dtype: int64

df=df.dropna() df.isnull().sum()

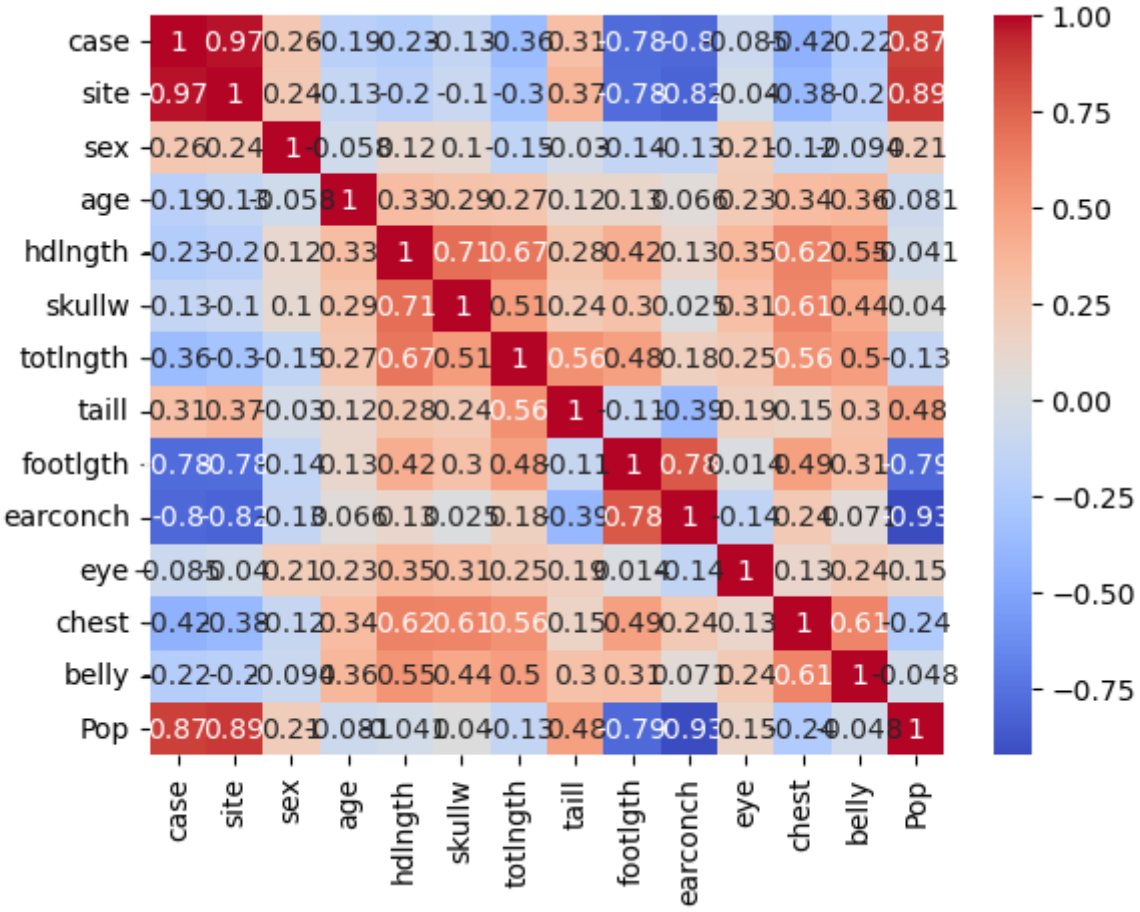
case 0 site 0 sex 0 age 0 hdlngth 0 skullw 0 totlngth 0 taill 0 footlgth 0 earconch 0 eye 0 chest 0 belly 0 Pop 0 dtype: int64

df.replace('Vic',0, inplace=True) df.replace('other',1, inplace=True) df.replace('f',10,inplace=True) df.replace('m',20,inplace=True)

Checking the correlation between all the variables

corr_matrix = df_cat.corr() corr_matrix

sns.heatmap(corr_matrix, cmap="coolwarm", annot=True) plot.show()



NonParametric Tests

Z_Test

A z test is conducted on a population that follows a normal distribution with independent data points and has a sample size that is greater than or equal to 30. It is used to check whether the means of two populations are equal to each other when the population variance is known. The null hypothesis of a z test can be rejected if the z test statistic is statistically significant when compared with the critical value

```
sample1=df.Pop
sample2=df.sex

mean1=np.mean(sample1)
mean2=np.mean(sample2)

std1=np.std(sample1)
std2=np.std(sample2)

n1=len(sample1)
n2=len(sample2)

Z_score=(mean1 - mean2) / np.sqrt((std1**2 / n1)+(std2**2 / n2))
p_value= 2 * norm.cdf(- abs(Z_score))

alpha = 0.05
if p_value < alpha:
    print('reject H0')
else:
    print('fail to reject H0')
```

```
Z_score: -30.97560810963735
p_value: 1.148767599466977e-210
reject H0
```

From the above result we can say that the means are not equal.

F-Test(One Way Anova):

```
stat,p_value=f_oneway(df.sex,df.site,df.age,df.hdlngth,df.skullw,df.totlngth,d
f.footlgth,df.taill,df.earconch,df.eye,
df.chest,df.belly)
print(stat,p_value)
print('%.3f'%p_value)
alpha = 0.05
if p_value > alpha:
    print('fail to reject H0')
else:
    print('reject H0')
```

```
Statistics: 8954.468220777773
p-value:  0.000
reject H0
```

We can see that the overall p-value from the ANOVA table is not significant ie. less than .05, so we have sufficient evidence to say that the mean values across each group are not equal.However, this doesn't tell us which groups are different from each other. It simply tells us that not all of the group means are equal. In order to find out exactly which groups are different from each other, we must conduct a post hoc test.

Thus, we can perform Tukey's Test to determine exactly which group means are different.

Tukey's Honest Significant Test:

Tukey's test compares the means of every treatment to the means of every other treatment; that is, it applies simultaneously to the set of all pairwise comparisons and identifies any difference between two means that is greater than the expected standard error.

The purpose of Tukey's test is to figure out which groups in your sample differ. It uses the "Honest Significant Difference," a number that represents the distance between groups, to compare every mean with every other mean.

```
tukey=pairwise_tukeyhsd(endog=df['site'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj lower  upper reject
-------------------------------------------------
     0      1   4.2502   0.0 3.8214 4.679   True
-------------------------------------------------
```

```
tukey=pairwise_tukeyhsd(endog=df['age'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj   lower  upper  reject
-------------------------------------------------
     0      1  -0.3103 0.4234 -1.0764 0.4557  False
-------------------------------------------------
```

```
tukey=pairwise_tukeyhsd(endog=df['hdlngth'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj   lower  upper  reject
-------------------------------------------------
   Vic  other   0.0091 0.9898 -1.3971 1.4153  False
-------------------------------------------------
```

```
tukey=pairwise_tukeyhsd(endog=df['skullw'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj   lower  upper  reject
-------------------------------------------------
     0      1   0.2469 0.6946 -0.9973 1.4911  False
-------------------------------------------------



tukey=pairwise_tukeyhsd(endog=df['totlngth'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj  lower  upper  reject
-------------------------------------------------
     0      1  -1.1307 0.182 -2.7997 0.5384  False
-------------------------------------------------



tukey=pairwise_tukeyhsd(endog=df['taill'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj lower  upper  reject
-------------------------------------------------
     0      1   1.9086   0.0 1.2149 2.6023   True
-------------------------------------------------



tukey=pairwise_tukeyhsd(endog=df['footlgth'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj  lower   upper  reject
-------------------------------------------------
     0      1  -7.0338   0.0 -8.1153 -5.9523   True
-------------------------------------------------



tukey=pairwise_tukeyhsd(endog=df['earconch'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj  lower   upper  reject
-------------------------------------------------
     0      1   -7.563   0.0 -8.1798 -6.9461   True
-------------------------------------------------
```

```
tukey=pairwise_tukeyhsd(endog=df['eye'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
==================================================
group1 group2 meandiff p-adj   lower  upper  reject
--------------------------------------------------
     0      1   0.3147 0.1404 -0.1055 0.7349  False
--------------------------------------------------
```

```
tukey=pairwise_tukeyhsd(endog=df['chest'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
==================================================
group1 group2 meandiff p-adj   lower   upper  reject
--------------------------------------------------
     0      1  -0.9814 0.0151 -1.7683 -0.1944   True
```

```
tukey=pairwise_tukeyhsd(endog=df['belly'],groups=df['Pop'],alpha=0.05)
print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
==================================================
group1 group2 meandiff p-adj   lower  upper  reject
--------------------------------------------------
     0      1  -0.2648 0.6319 -1.3582 0.8286  False
--------------------------------------------------
```

The Analysis of Categorical Data

CHI SQUARED TEST OF INDEPENDENCE:

The chi-square test of independence is a statistical test used to determine whether two categorical variables are related in the population. It evaluates a null and alternative hypothesis and determines whether the values of one categorical variable depend on the value of other categorical variables. The test is valid when the test statistic is chi-squared distributed under the null hypothesis, and it is used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more categories of a contingency table.

```
columns= ['sex','age']
contingency_table = pd.crosstab(df[columns[0]], df[columns[1]])
chi_2,p_value,dof,expected= chi2_contingency(contingency_table)
q=0.05
if chi_2 > critical:
    print('Dependent (reject H0)')
else:
    print('Independent (fail to reject H0)')
```

```
Chi-squared value: 7.6552261221500535
P-value: 0.46785207442448074
Independent (fail to reject H0)
```

```
H0: (null hypothesis) The two variables are independent.
H1: (alternative hypothesis) The two variables are not independent. Since the p
-value (0.4678) of the test is not
less than 0.05, we fail to reject the null hypothesis. This means we do not hav
e sufficient evidence to say that
there is an association between sex and age. In other words, they are independe
nt.
```

Resampling Method:

Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform.

In stratified k-fold cross-validation, the data is split into k-folds, where each fold has an equal proportion of samples of each target class as the complete data set. This ensures that the distribution of the target variable is preserved across all folds, which can lead to better and more reliable model performance estimation.

```
X=df[['site','taill']]
Y=df['Pop']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25,random_state=
42)
SKF = StratifiedKFold(n_splits = 7)
KF = KFold(n_splits = 7)
scores_logistic_KF = []
scores_logistic_SKF = []
for train_index, test_index in KF.split(X,Y):
    X_train,X_test,Y_train,Y_test=X.iloc[train_index], X.iloc[test_index],Y.ilo
c[train_index], Y.iloc[test_index]
    lr.fit(X_train, Y_train)
    score = lr.score(X_test, Y_test)
    scores_logistic_KF.append(score)

for train_index, test_index in SKF.split(X,Y):
    X_train,X_test,Y_train,Y_test=X.iloc[train_index], X.iloc[test_index],Y.ilo
c[train_index], Y.iloc[test_index]
    lr.fit(X_train, Y_train)
    score = lr.score(X_test, Y_test)
    scores_logistic_SKF.append(score)



scores_logistic_KF = [1.0, 1.0, 1.0, 0.9285714285714286, 1.0, 1.0, 1.0]
scores_logistic_SKF = [0.7333333333333333, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
average scores_logistic_KF = 0.9897959183673469
average scores_logistic_SKF = 0.9619047619047619
```

The K-fold cross-validation approach resulted in an average score of 0.9897959183673469, which is slightly better than the Stratified K-fold cross-validation approach that resulted in an average score of 0.9619047619047619. Therefore, it can be suggested that using K-fold cross-validation can provide better performance for the logistic regression model. However, further evaluation is needed to determine if these scores are statistically significant and to determine if the model is suitable for deployment in a real-world setting.

Linear Model Selection and Regularization:

Linear model selection is the process of choosing which variables or features to include in a linear regression model. The main aim of model selection is to identify the most important and relevant variables to predict the response variable.

Lasso and Ridge regularization methods work by adding a penalty term to the objective function of our linear model, which can help prevent overfitting by shrinking the coefficients towards zero. Lasso has the additional property of performing variable selection by driving some coefficients to exactly zero, effectively removing those variables from the model. This can be useful in identifying the most important predictor variables for our response variable.

```
X=df[['site','sex','age','hdlngth','skullw','totlngth','taill','footlgth','earc
onch','eye','chest','belly']]
Y=df['Pop']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25,random_state=
42)
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, Y_train)
ridge_reg = Ridge(alpha=0.1)
ridge_reg.fit(X_train, Y_train)
y_pred_lasso = lasso_reg.predict(X_test)
y_pred_ridge = ridge_reg.predict(X_test)
mse_lasso = mean_squared_error(Y_test, y_pred_lasso)
rmse_lasso = np.sqrt(mse_lasso)
mse_ridge = mean_squared_error(Y_test, y_pred_ridge)
rmse_ridge = np.sqrt(mse_ridge)
```

```
Lasso RMSE: 0.13298649075759097
Ridge RMSE: 0.1302069092435807
```

The Lasso RMSE score of 0.13298649075759097 indicates that the model has an average error of approximately 0.13, while the Ridge RMSE score of 0.1302069092435807 indicates that the model has an average error of approximately 0.13 as well.

feature selection is an important step,it can help improve model performance, reduce overfitting, and provide insights into the underlying problem.

Forward Selection:

Forward selection is a type of stepwise feature selection technique which begins with an empty model and adds in variables one by one. In each forward step, you add the one variable that gives the single best improvement to your model.

```
X=df[['site','sex','age','hdlngth','skullw','totlngth','taill','footlgth','earc
onch','eye','chest','belly']]
Y=df['Pop']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random
_state=42)
ffs = SFS(RFC(n_jobs=-1), k_features=(1,12), forward=True, floating=False, verb
ose=3, scoring='accuracy', cv=3).fit(X_train,Y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    4.5s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    4.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    9.0s finished

[2023-05-05 16:25:03] Features: 1/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    5.0s finished

[2023-05-05 16:25:08] Features: 2/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    4.5s finished

[2023-05-05 16:25:12] Features: 3/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    4.1s finished

[2023-05-05 16:25:17] Features: 4/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    3.5s finished

[2023-05-05 16:25:20] Features: 5/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    3.1s finished

[2023-05-05 16:25:23] Features: 6/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    2.6s finished

[2023-05-05 16:25:26] Features: 7/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.2s finished

[2023-05-05 16:25:29] Features: 8/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.8s finished

[2023-05-05 16:25:30] Features: 9/12 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.3s finished

[2023-05-05 16:25:32] Features: 10/12 -- score: 1.0[Parallel(n_jobs=1)]: Using
backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s finished

[2023-05-05 16:25:33] Features: 11/12 -- score: 1.0[Parallel(n_jobs=1)]: Using
backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s finished

[2023-05-05 16:25:33] Features: 12/12 -- score: 1.0
```

The resulting score was 1.0 indicates that the selected features were very useful in accurately predicting the target variable. it is important to note that forward feature selection can lead to overfitting. While selecting all features may lead to an accurate model, it may not be the most optimal approach. It is crucial to carefully evaluate the performance of the model using other techniques to ensure that the model has good.

Thus, we can perform next methods to get the best features.

Backward Selection:

Backward elimination is totally opposite to forward. In that, you start with a model that includes every possible variable and eliminate the extraneous variables one by one

```
X=df[['site','sex','age','hdlngth','skullw','totlngth','taill','footlgth','earc
onch','eye','chest','belly']]
Y=df['Pop']
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_st
ate=42)
bfs = SFS(RFC(n_jobs=-1), k_features=(1,12), forward=False, floating=True, verb
ose=2, scoring='accuracy', cv=3)
bfs.fit(X_train, Y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    5.5s finished

[2023-05-05 16:25:39] Features: 11/1 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    4.9s finished

[2023-05-05 16:25:45] Features: 10/1 -- score: 1.0[Parallel(n_jobs=1)]: Using b
ackend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    4.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s finished

[2023-05-05 16:25:50] Features: 9/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    3.9s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.3s finished

[2023-05-05 16:25:55] Features: 8/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    3.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.8s finished

[2023-05-05 16:26:01] Features: 7/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    3.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.1s finished

[2023-05-05 16:26:06] Features: 6/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    2.6s finished
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    2.5s finished

[2023-05-05 16:26:12] Features: 5/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    3.1s finished

[2023-05-05 16:26:17] Features: 4/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    3.5s finished

[2023-05-05 16:26:23] Features: 3/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    4.1s finished

[2023-05-05 16:26:28] Features: 2/1 -- score: 1.0[Parallel(n_jobs=1)]: Using ba
ckend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    4.4s finished

[2023-05-05 16:26:34] Features: 1/1 -- score: 1.0
```

The result from backward feature selection suggests that the selected feature is significant and useful in predicting the target variable. Nonetheless, further evaluation of the models performance and the effect of other features may be needed to make a more conclusive judgment about the significance of the selected feature.

Thus, we can perform next methods to get the best features.

L1 Regularization:

L1 and L2 regularization are techniques to prevent overfitting by adding a penalty term to the cost function. L1 regularization uses the absolute value of the weights.

```
X=df[['site','sex','age','hdlngth','skullw','totlngth','taill','footlgth','earc
onch','eye','chest','belly']]
Y=df['Pop']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random
_state=42)
clf = LogisticRegression(penalty='l1', solver='saga', C=0.1, max_iter=1000)
clf.fit(X_train, Y_train)
```

```
Training accuracy: 1.0
Testing accuracy: 1.0
```

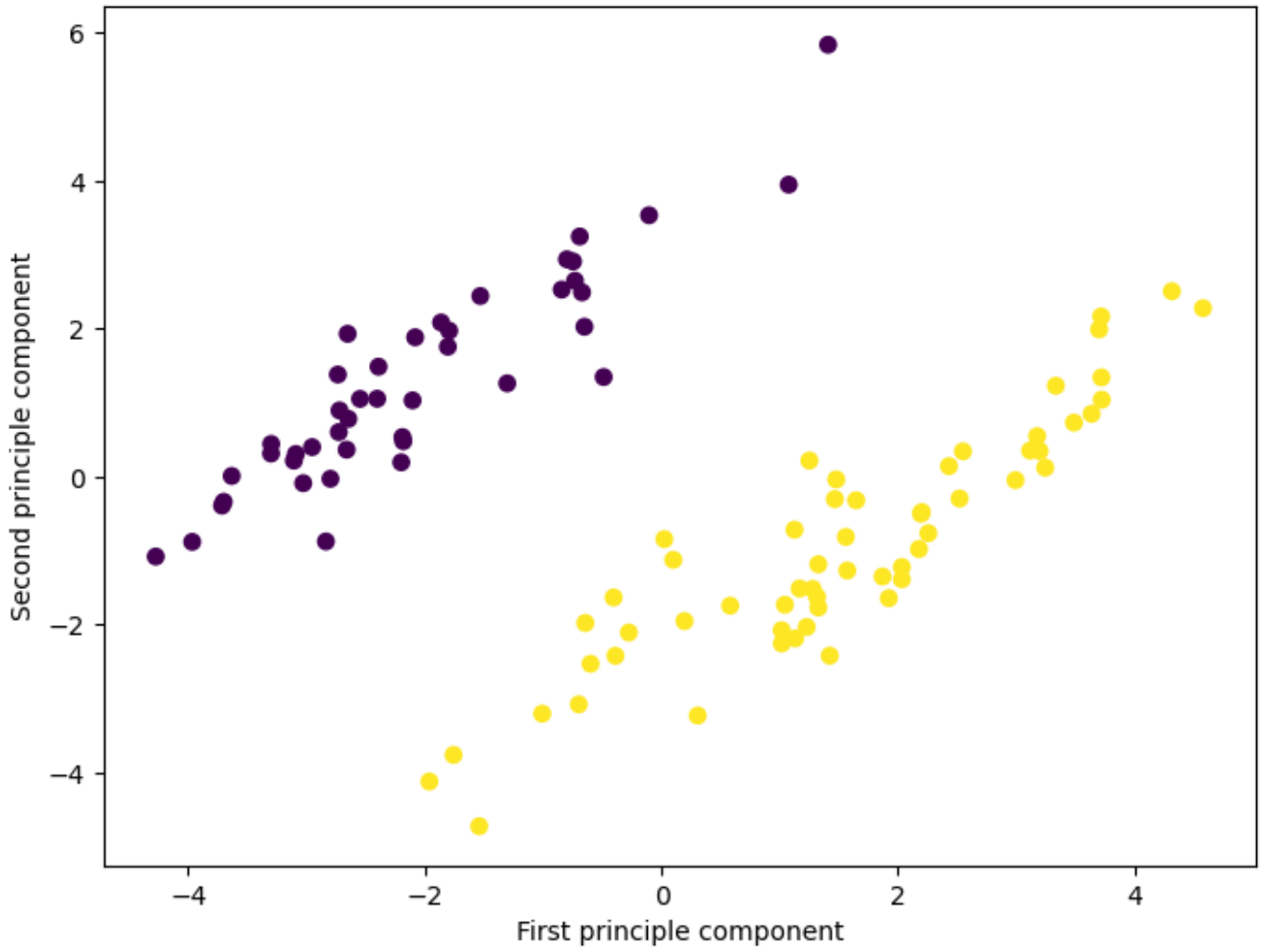
The model is able to predict the target variable with 100% accuracy using the features in the dataset. It is possible that the model may be overfitting on the training data. Further evaluation is required to determine if the models are suitable for deployment in real-world applications.

Thus, we can perform the next methods to get the best features.

PCA(Principal Component Analysis):

PCA (Principal Component Analysis) is a statistical technique used for data reduction without losing its properties. It is a popular technique for analyzing large datasets containing a high number of dimensions/features per observation. PCA enables us to better generalize machine learning models by reducing the dimensionality of the data.

```
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)
pca=PCA(n_components=2)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
plot.figure(figsize=(8,6))
plot.scatter(x_pca[:,0],x_pca[:,1],c=df['Pop'])
plot.xlabel('First principle component')
plot.ylabel('Second principle component')
```



I compared these PCA using x_pca with logistic regression but it has given 100% accuracy. It is possible that the model may be overfitting on the training data. Furthur evalution is required to get the best results on model.

Thus, we can perform the next methods to get the best model.

Recursive Feature Selection:

Recursive Feature Elimination, or RFE for short, is a popular feature selection algorithm. RFE is popular because it is easy to configure and use and because it is effective at selecting those features (columns) in a training dataset that are more or most relevant in predicting the target variable.

```
X=df[['site','sex','age','hdlngth','skullw','totlngth','taill','footlgth','earc
onch','eye','chest','belly']]
Y=df['Pop']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random
_state=42)
selector = RFE(lr, n_features_to_select=3, step=1)
selector = selector.fit(X_train, Y_train)
selected_features = X.columns[selector.support_]
lr.fit(X_train[selected_features], Y_train)
y_pred = lr.predict(X_test[selected_features])
R2 = r2_score(Y_test, y_pred)
```

```
Output:
Index(['site', 'taill', 'earconch'], dtype='object')
R2 = 0.8964557631416399
```

The high R2 value of 0.8964557631416399 suggests that these features are important predictors of the target variable and can be used to build a reliable model.

I got the top best 3 features using recursive feature selection and we can continue with logistic regression using these features.

I want to try another method may be we can get better performance on the model compare to recursive thus, we can perform Non-linear regression method and see if we able to get the better accuracy or not.

Moving Beyond Linearity:

Nonlinear regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables.

```
X=df[['site','sex','age','hdlngth','skullw','totlngth','taill','footlgth','earc
onch','eye','chest','belly']]
Y=df['Pop']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random
_state=42)
degree = [2,3,4]
for i in degree:
    poly = PolynomialFeatures(degree=int(i))
    X_poly_train = poly.fit_transform(X_train)
    X_poly_test = poly.transform(X_test)
    lin_reg = LinearRegression()
    lin_reg.fit(X_poly_train,Y_train)
    y_pred = lin_reg.predict(X_poly_test)
    r_squared = r2_score(Y_test,y_pred)
    print('R-squared for degree'+str(i)+':'+' '+str(r_squared))
```

```
R-squared for degree2: 0.024992102850834086
R-squared for degree3: 0.5644619092409253
R-squared for degree4: 0.5625941013275446
```

If you check the R-squared values they are low and it is might difficult to get the best model using this method.

We will use the top best 3 features which we got from the recursive regression and will use those features and do a logistic regression to get the good performance on the model and we can use this model and deploy it for real world applications.

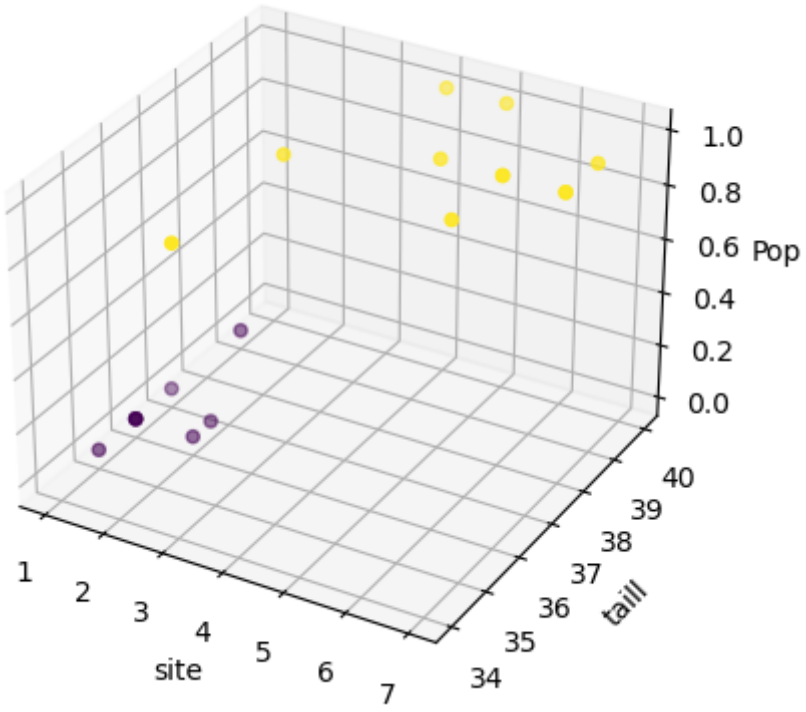
Fitting the Model:

Logistic Regression:

Logistic regression is a statistical technique used for modeling the relationship between a binary response variable and one or more predictor variables. The aim of logistic regression is to predict the binary outcome (e.g., 0 or 1, yes or no) based on the values of the predictor variables.

```
X=df[['site', 'taill']]
Y=df['Pop']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random
_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
```

The logistic regression model has achieved an accuracy of 95.2% on the test data, indicating that it is a highly accurate model in predicting the target variable based on the given features.



Conclusion:

In order to ascertain the relationship between the numeric input columns and output categorical columns, we successfully applied statistical techniques like non-parametric testing and categorical data analysis. We also looked at the relationship between the input and output categorical variables using the chi-squared test of independence.

Then, in order to select the most accurate predictive features and fit the model to predict the possum population, we used feature selection techniques and the logistic regression model.

Finally, when the model fit was evaluated using the test set, we have achieved an accuracy of 95.2%. This indicates that the model is capable of accurately predicting the possum population using the selected features.

References:

1. https://www.kaggle.com/datasets/abrambeyer/openintro-possum (https://www.kaggle.com/datasets/abrambeyer/openintro-possum).
2. Mathematical Statistics and Data Analysis, by J. A. Rice.
3. An Introduction to Statistical Learning with Applications in R, by G. James, D. Witten, T. Hastie, & R. Tibshirani.