Data Analysis Using Statistical Methods: Categorizing the Species of Iris Flowers

Vamsi Kasukurthi

Department of Mathematical Science, Stevens Institute of Technology, Hoboken, NJ

MA-541 Statistical Methods

Prof.Hadi Safari Katesari

May 6th, 2023

ABSTRACT

Statistical analysis is an analytical tool that assists in the collection and analysis of huge amounts of data in order to uncover common patterns and trends and convert them into useful knowledge. The idea is to use a dataset on which various statistical approaches can be applied to provide reliable predictions. While categorizing the data, the accuracy metric used in the project gives us a 96.667% accuracy.

INTRODUCTION

The project developed here is to implement various statistical methods across analysis of data. This includes non-parametric tests(Z-test), ANOVA, Chi squared test of independence, Categorical analysis of data and Logistic regression.

First we will implement different statistical methods for observing the correlation between the columns of the dataset. With the help of these columns, we will be implementing a logistic regression model. We then check the accuracy of the model with the help of testing data.

DATA DESCRIPTION

Dataset: 'Iris.csv'

The 'Iris Species' dataset was used in R.A. Fisher's classic 1936 paper, "The Use of Multiple Measurements in Taxonomic Problems", and can also be found on the UCI Machine Learning Repository.
It includes three iris species with 50 samples each as well as some properties about each flower. The columns in this dataset are:
•Id
•SepalLengthCm
•SepalWidthCm
•PetalLengthCm
•PetalWidthCm
•Species - 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'
The datatype of the columns SepalLengthCm, PetalLengthCm, SepalWidthCm, PetalWidthCm is float and the datatype of the column Species is object.

LIBRARIES REQUIRED

import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns

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

from scipy.stats import shapiro from scipy.stats import chi2_contingency from sklearn.metrics import f1_score

from scipy.stats import f_oneway from scipy.stats import chi2 from sklearn.linear_model import LinearRegression as LR

from sklearn.model_selection import train_test_split from sklearn.metrics import r2_score from sklearn.metrics import mean_squared_error

from sklearn.model_selection import StratifiedKFold, KFold from sklearn.linear_model import LinearRegression from sklearn.linear_model import Lasso, Ridge

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

Looking at the head of the dataset, we can notice that there are four columns which will be used to predict the species. As the species is a categorical variable and contains more than two values, we will fit multinomial logistic regression model.

df.head()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

df.describe()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

DATA PREPROCESSING

checking for all the null values and dropping them if any. Changing the categories to 0,1 and 2.

df.usnull().sum()

```
Id               0
SepalLengthCm    0
```

df.replace('Iris-setosa', 0, inplace=True) df.replace('Iris-versicolor', 1, inplace=True) df.replace('Iris-virginica', 2, inplace=True)

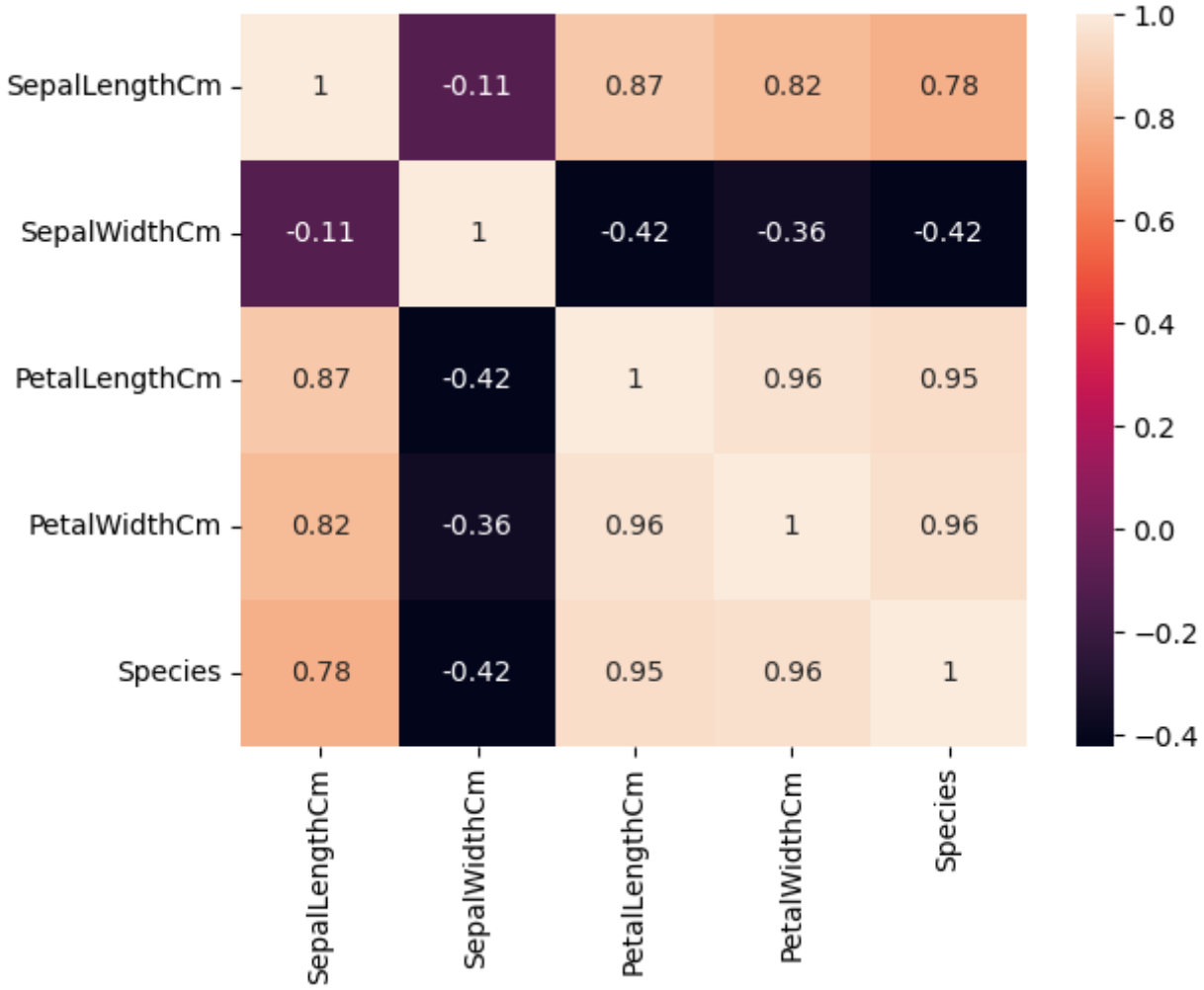
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

checking the correlation between all the variables

corr_matrix = df.corr() print(corr_matrix)

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954	0.782561
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544	-0.419446
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757	0.949043
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000	0.956464
Species	0.782561	-0.419446	0.949043	0.956464	1.000000

correlation heatmap of the iris dataset



NONPARAMETRIC TESTS

Z-TEST

The z-test assumes that the sample is normally distributed or that the sample size is large enough to apply the central limit theorem. It also assumes that the sample is drawn randomly from the population.

The hypothesis being tested in a z-test is typically formulated as a null hypothesis, which states that there is no significant difference between the sample mean and the population mean. The alternative hypothesis, on the other hand, states that there is a significant difference between the two means.

If the calculated test statistic exceeds the critical value from a z-table or is less than the negative of the critical value, then the null hypothesis is rejected in favor of the alternative hypothesis.

m_1 = df['SepalLengthCm'].mean() m_2 = df['PetalLengthCm'].mean()

s_1 = df['SepalLengthCm'].std() s_2 = df['PetalLengthCm'].std()

n_1, n_2 = len(df['SepalLengthCm']), len(df['PetalLengthCm'])

Z_value = (m_1 - m_2)/np.sqrt((s_1**2/n_1)+(s_2**2/n_2)) p_value = 2 * (1 - stats.norm.cdf(abs(Z_value)))

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

Z_value=13.099504, p_value=0.000

(reject H0) which means the means are not equal

ONE WAY ANOVA (F-TEST)

If we have more than two groups to analyze, we will use F-test.
Null hypothesis(H0): Median values accross each group are equal.
Alternate hypothesis(H1): Not H0.

```
stat, p = f_oneway(df.PetalLengthCm, df.PetalWidthCm, df.SepalWidthCm, df.SepalLengthCm)
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p<alpha:
print('reject H0')
else:
print('reject H1')
```

Statistics=483.571, p=0.000
reject H0
We can see that the value of 'p' is less than 0.05 which tells us that the means are not equal. But we don't know about which groups are not equal.

TUKEY HONEST SIGNIFICANT TEST

Tukey test is a single-step multiple comparison procedure and statistical test. It can be used to find means that are significantly different from each other.
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.

```
tukey_1 = pairwise_tukeyhsd(df.PetalLengthCm, df.Species, alpha = 0.05)
tukey_1.summary()
```

```
    Multiple Comparison of Means - Tukey HSD, FWER=0.05
    group1 group2 meandiff   p-adj      lower     upper   reject
    0       1      2.796       0.0     2.5922    2.9998    True
    0       2      4.088       0.0     3.8842    4.2918    True
    1       2      1.292       0.0     1.0882    1.4958    True
```

```
tukey_2 = pairwise_tukeyhsd(endog =df['PetalWidthCm'], groups=df['Species'], alpha = 0.05)
tukey_2.summary()
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
group1 group2 meandiff p-adj lower   upper  reject
0      1      1.082    0.0   0.9849  1.1791  True
0      2      1.782    0.0   1.6849  1.8791  True
```

```
tukey_3 = pairwise_tukeyhsd(df.SepalLengthCm, df.Species, alpha = 0.05)
tukey_3.summary()
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
group1 group2 meandiff p-adj lower  upper  reject
0      1      0.93     0.0   0.6862 1.1738 True
0      2      1.582    0.0   1.3382 1.8258 True
1      2      0.652    0.0   0.4082 0.8958 True
```

```
tukey_4 = pairwise_tukeyhsd(df.SepalWidthCm, df.Species, alpha = 0.05)
tukey_4.summary()
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
group1 group2 meandiff p-adj lower    upper   reject
0      1      -0.648   0.0   -0.8092  -0.4868  True
0      2      -0.444   0.0   -0.6052  -0.2828  True
1      2       0.204   0.009  0.0428   0.3652  True
```

Analysis of Categorical Data

CHI-SQUARED TEST OF INDEPENDENCE

Chi-square test of independence is used to determine whether two categorical variables are related. If two variables are related, the probability of one variable having a certain value is dependent on the value of the other variable.
Null hypothesis (H0): Variables are not related in the population.
Alternative hypothesis (H1): Variables are related in the population.

```
cont_table = pd.crosstab(df['PetalWidthCm'], df['SepalLengthCm'])
chi_2, p, dof, expected = chi2_contingency(cont_table)
print('Chi-square statistic:', chi_2)
print('p-value:', '%.3f' %p)
print('dof:', dof)
chi_2_table = chi2.ppf(q = 0.95, df = dof)
print('The value of chi-square from the table:', chi_2_table)
```

```
Chi-square statistic: 794.2022610326181
p-value: 0.019
dof: 714
The value of chi-square from the table: 777.2732792817321
```

```
if chi_2 > chi_2_table:
print('reject H0:')
print('Variables are independent')
```

else:
print('reject H1:')
print('Variables are dependent')

reject H0
The chi-square value obtained from the dataset is greater than the value obtained from the table using the fegrees of freedom and alpha significance level(0.05). As a result, we got reject H0 which means that the variables are independent.

Linear Regression

Linear regression is a type of statistical analysis used to predict the relationship between two variables. It assumes a linear relationship between the independent variables and the dependent variable, and aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.

X = df.iloc[:,:-1] y = df.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
model = LR() model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred) print("MSE:", mse)
R2 = r2_score(y_test, y_pred)
print('R2:', R2)

MSE: 0.03623468054768068
R2: 0.9544536138367142

After fitting the linear model, we got the MSE as 0.036 and the r_squared value as 0.9544. We will then see the average score using cross validation techniques and compare the results.

RESAMPLING METHODS

K-FOLD cross validation

It is widely used approach for estimating test error. Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
Idea is to randomly divide the data into K equal-sized parts. We leave out part k, fit the model to the other K − 1 parts (combined), and then obtain predictions for the left-out kth part.
This is done in turn for each part k = 1, 2, . . . K, and then the results are combined.

STRATIFIED K-FOLD cross validation

This is same as K-Fold CV but when there are more than two categorical variables, we use stratified K-Fold so that it divides the training and testing data in such a way that it contains all the categorical variables proportionately.

X = df.iloc[:, :-1] y = df.iloc[:, -1]
def get_score(model, X_train, X_test, y_train, y_test):
model.fit(X_train, y_train)

```
return model.score(X_test, y_test)
folds = StratifiedKFold(n_splits=5) kf = KFold(n_splits=5)

scores_kf = [] scores_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.iloc[train_index], y.iloc[test_index]
scores_kf.append(get_score(LinearRegression(), X_train, X_test, y_train, y_test))

for train_index, test_index in folds.split(X, y):
X_train, X_test, y_train, y_test = X.iloc[train_index], X.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
scores_skf.append(get_score(LinearRegression(), X_train, X_test, y_train, y_test))

print('average score of k-fold:' np.average(scores_kf))
```

```
average score of k-fold: 0.3228834275997373
average score of stratified k-fold:0.9264434246210916
```

The score of the stratified k-fold is better than k-fold as the model inputs the training data with equal proportions of categorical variables.

Linear Model Selection and Regularization

Linear model selection refers to the process of selecting a linear regression model that best fits a given dataset. This involves choosing the appropriate subset of predictor variables, as well as deciding on the degree of polynomial terms to include in the model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=100)
lr = LinearRegression() lr.fit(X_train, y_train)
y_pred = lr.predict(X_test) MSE = mean_squared_error(y_test, y_pred) RMSE = np.sqrt(MSE)
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)
```

The RMSE values of linear, lasso and ridge models are 0.2029326740428823, 0.2555455565419551 and 0.2030228959320554 respectively.
We can observe that there is no major difference between the three models used to predict the target variable.

Forward Selection

Forward selection is a technique for building a linear regression model by iteratively adding predictor variables that best improve the model's performance. The process starts with an empty model and then adds variables one at a time until a stopping criterion is met, such as reaching a desired model size or when adding new variables no longer improves the model's performance.

```
X = df.iloc[:, 1:5] y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
FFS = SFS(RFC(n_jobs=-1), k_features=(1,4), forward=True, floating=False, verbose=2,
```

```
scoring='accuracy', cv=5).fit(X_train, y_train)
FFS.k_score_
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 1.3s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 4 out of 4 | elapsed: 2.1s finished

[2023-05-06 13:44:26] Features: 1/4 -- score: 1.0[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 3 out of 3 | elapsed: 0.7s finished

[2023-05-06 13:44:27] Features: 2/4 -- score: 1.0[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 0.5s finished

[2023-05-06 13:44:28] Features: 3/4 -- score: 1.0[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s finished

[2023-05-06 13:44:28] Features: 4/4 -- score: 1.0
```

Backward Selection

Backward elimination is a technique for building a linear regression model by iteratively removing predictor variables that contribute the least to the model's performance. The process starts with a full model that includes all predictor variables, and then removes variables one at a time until a stopping criterion is met, such as reaching a desired model size or when further removals do not significantly impact the model's performance.

```
BFS = SFS(RFC(n_jobs=-1), k_features=(1,4), forward=False, floating=True, verbose=2,
scoring='accuracy', cv=5).fit(X_train, y_train)
BFS.k_score_
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 4 out of 4 | elapsed: 0.9s finished

[2023-05-06 13:44:29] Features: 3/1 -- score: 1.0[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 3 out of 3 | elapsed: 0.7s finished

[2023-05-06 13:44:30] Features: 2/1 -- score: 1.0[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 0.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 0.5s finished
```

[2023-05-06 13:44:31] Features: 1/1 -- score: 1.0

As we can see that both forward and the backward feature selection methods gave the score as 1.0 which means that the model is overfitting the data as it contains only 150 samples in the dataset.
We will use principal component analysis to interpret the data points by reducing the dimensionality.

Principal Component Analysis

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

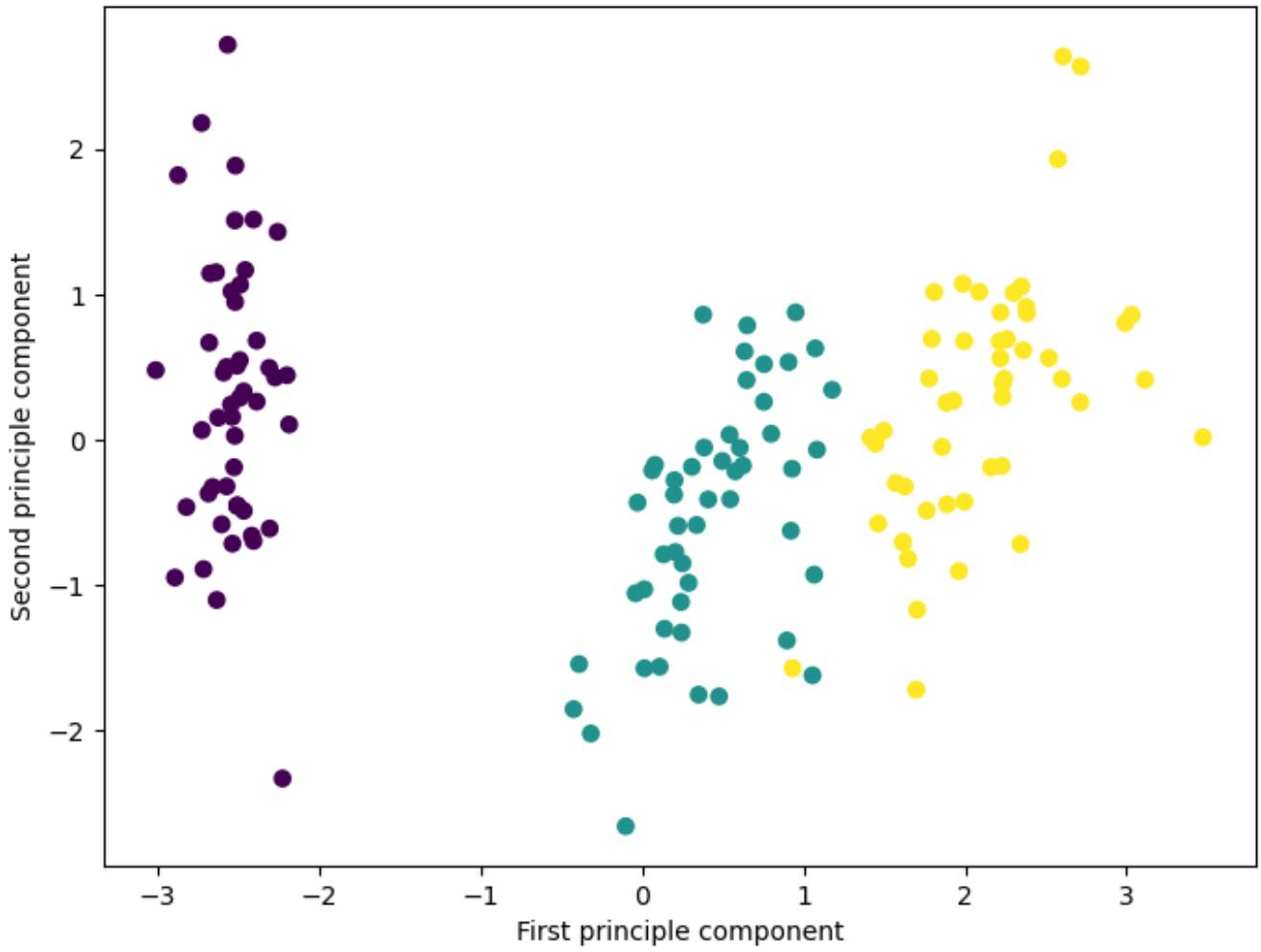
Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.

```
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)
x_pca.shape
```

(150,2)

We can observe that the dimensions of the input variables is reduced to 2 from 4.

```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c = df['Species'])
plt.xlabel('First principle component')
plt.ylabel('Second principle component')
```



We will implement non-linear regression to check if it predicts the target variable more accurately than the linear regression model.

Moving Beyond Linearity

```
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 degree 2: 0.927700134984149
R-squared for degree 3: 0.9038504178983079
R-squared for degree 4: -1.3962081402508932

As we can see that the R-squared value is decresing when the degree of the polynomial increases. This indicates that the non-linear regression is not the best model for the prediction of the iris-species.

FITTING THE MODEL

Multinomial Logistic Regression

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
classifier = LogisticRegression(random_state = 0, multi_class='multinomial', solver='lbfgs')
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_pred, y_test)
print('The accuracy of the logistic regression model is:', '%.3f' %(accuracy*100) + '%')
```

The accuracy of the logistic regression model is: 96.667%

CONCLUSION

We have successfully used various statistical methods for analysing the data. By using the non-parametric tests to check the relation (check if their mean are equal or not) between the input variables ('SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm') and the output categorical variable ('Species'). We used chi-square test of independence to determine whether the variables are independent or not. We then used some feature selection techniques to get the features that predicts the target variable accurately but the data got overfitted the model. We used multinomial logistic regression model to predict the target variable and obtained accuracy of 96.667%.

REFERENCES

[1] https://www.kaggle.com/datasets/uciml/iris (https://www.kaggle.com/datasets/uciml/iris)