Introduction

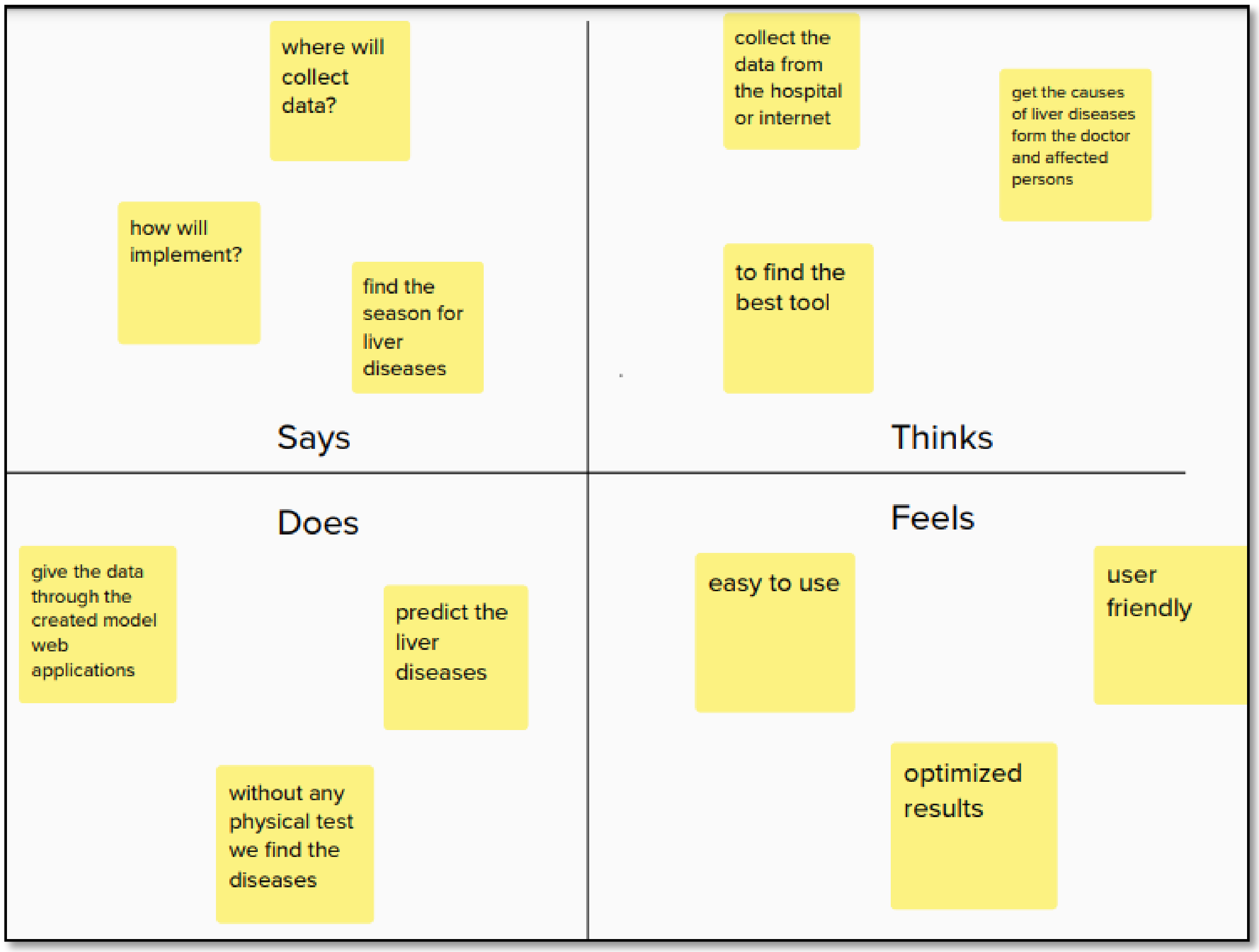
1.1 Overview

Liver diseases averts the normal function of the liver. This disease is caused by an assortment of elements that harm the liver. Diagnosis of liver infection at the preliminary stage is important for better treatment. In today's scenario devices like sensors are used for detection of infections. Accurate classification techniques are required for automatic identification of disease samples. This disease diagnosis is very costly and complicated. Therefore, the goal of this work is to evaluate the performance of different Machine Learning algorithms in order to reduce the high cost of liver disease diagnosis. Early prediction of liver disease using classification algorithms is an efficacious task that can help the doctors to diagnose the disease within a short duration of time. In this project we will analyse the parameters of various classification algorithms and compare their predictive accuracies so as to find out the best classifier for determining the liver disease. This project compares various classification algorithms such as Random Forest, Logistic Regression, KNN and ANN Algorithm with an aim to identify the best technique. Based on this study, Random Forest with the highest accuracy outperformed the other algorithms and can be further utilised in the prediction of liver disease and can be recommended to the user.

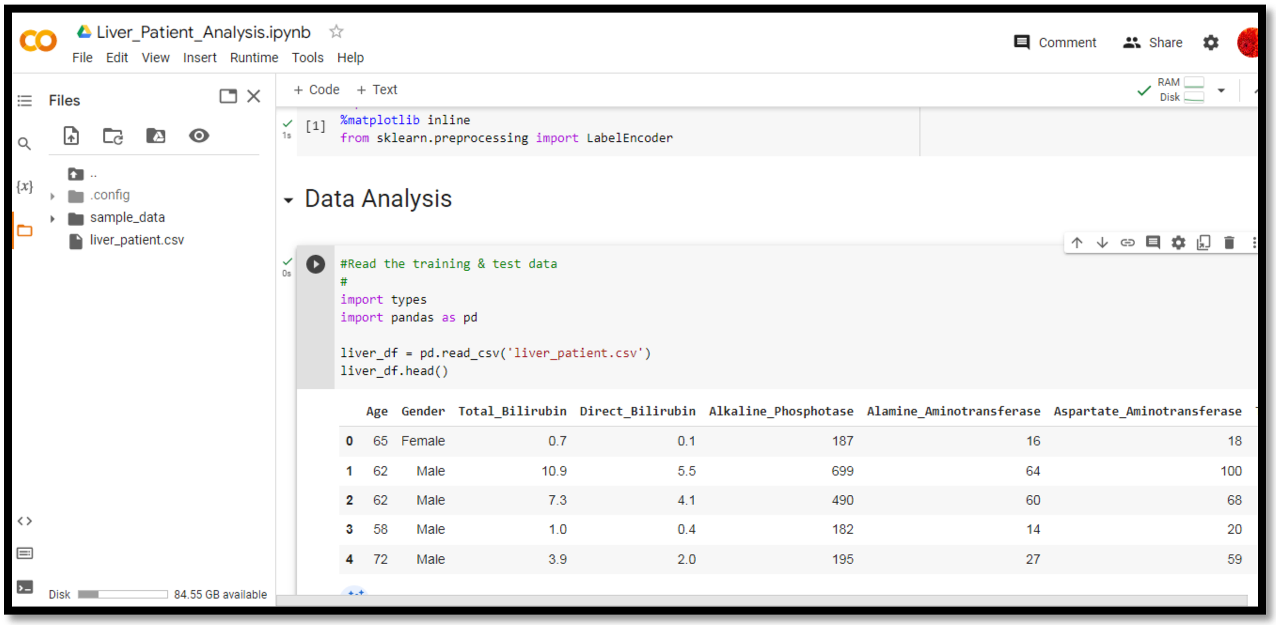
1.2 Purpose

The purpose of this project is to early prediction of Liver

2. Problem Definition & Design Thinking


where will collect data?
how will implement?
find the season for liver diseases
Says
collect the data from the hospital or internet
get the causes of liver diseases form the doctor and affected persons
to find the best tool
Thinks
Does
give the data through the created model web applications
predict the liver diseases
without any physical test we find the diseases
Feels
easy to use
user friendly
optimized results

3. Results



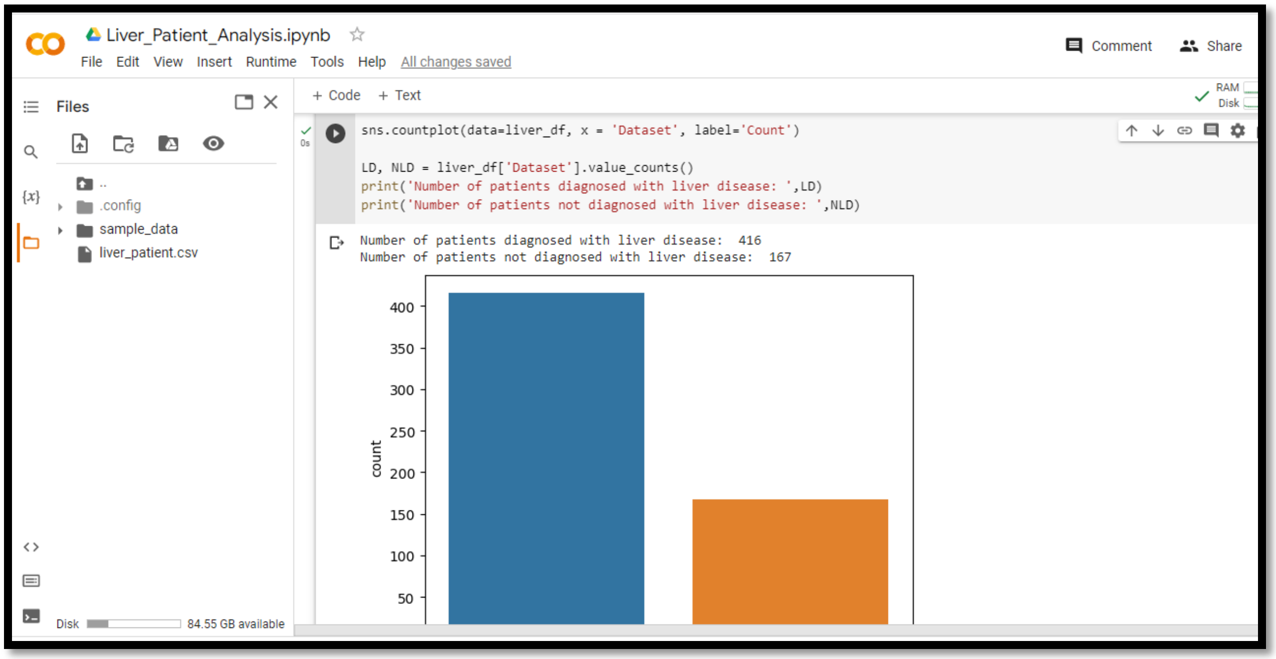
```
[1] %matplotlib inline
    from sklearn.preprocessing import LabelEncoder
```

Data Analysis

```
#Read the training & test data
#
import types
import pandas as pd

liver_df = pd.read_csv('liver_patient.csv')
liver_df.head()
```

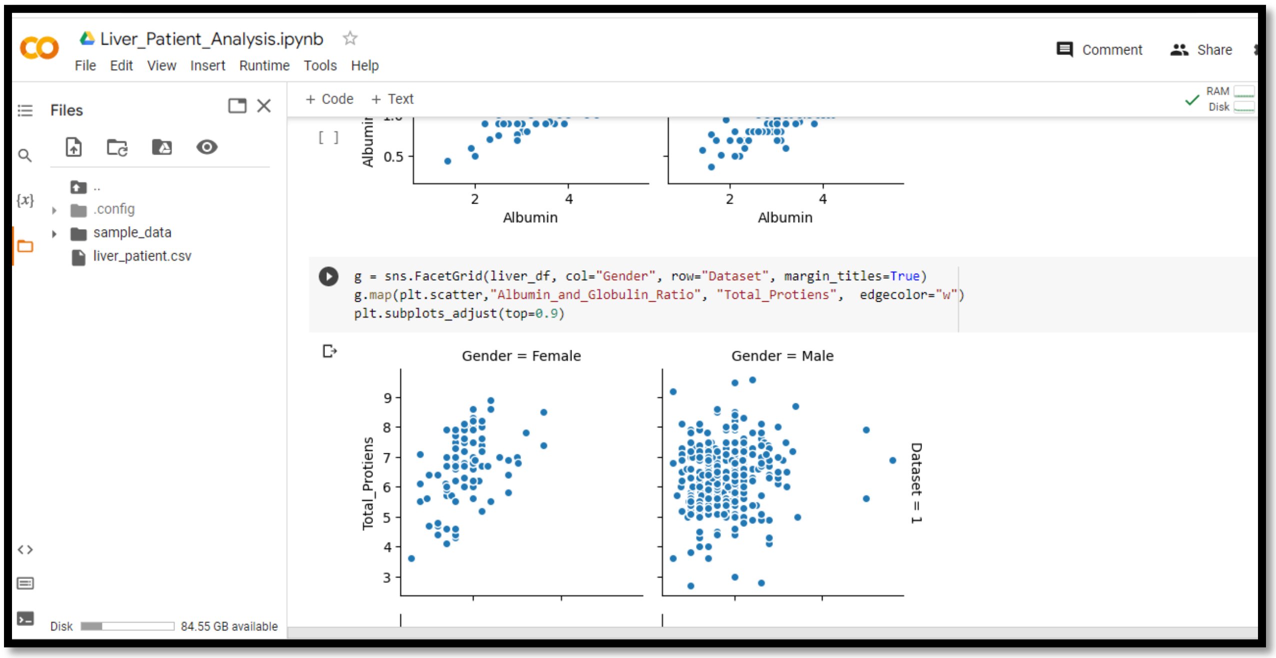
	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase
0	65	Female	0.7	0.1	187	16	18
1	62	Male	10.9	5.5	699	64	100
2	62	Male	7.3	4.1	490	60	68
3	58	Male	1.0	0.4	182	14	20
4	72	Male	3.9	2.0	195	27	59



```
sns.countplot(data=liver_df, x = 'Dataset', label='Count')

LD, NLD = liver_df['Dataset'].value_counts()
print('Number of patients diagnosed with liver disease: ',LD)
print('Number of patients not diagnosed with liver disease: ',NLD)
```

```
Number of patients diagnosed with liver disease:  416
Number of patients not diagnosed with liver disease:  167
```



```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Albumin_and_Globulin_Ratio", "Total_Protiens",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

4. Advantages and Disadvantages

- Produce high accuracy
- Easy to use
- Early Detection

5. Applications

It can be applied in medical field.

6. Conclusion

This project compares various classification algorithms such as Random Forest, Logistic Regression, KNN and ANN Algorithm with an aim to identify the best technique. Based on this study, Random Forest with the highest accuracy outperformed the other algorithms and can be further utilised in the prediction of liver disease and can be recommended to the user.

7. Future Scope

For this Model will develop Mobile Applications

8. Appendex

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# %matplotlib inline

from sklearn.preprocessing import LabelEncoder


"""# Data Analysis"""


#Read the training & test data

#

import types

import pandas as pd
```

```
liver_df = pd.read_csv('liver_patient.csv')

liver_df.head()

#Top 5 rows of the dataset

liver_df.head()

# To get a concise summary of the dataframe

liver_df.info()

# Statistical information about NUMERICAL columns in the dataset

liver_df.describe(include='all')

# Features of the dataset (Labels)

liver_df.columns

# Check for any null values
```

```
liver_df.isnull().sum()


"""- The only data that is null is the Albumin_and_Globulin_Ratio - Only 4 rows are null. Lets
see whether this is an important feature


# Data Visualization

"""


# Frequency of patients diagnosed and not diagnoised with liver disease


sns.countplot(data=liver_df, x = 'Dataset', label='Count')


LD, NLD = liver_df['Dataset'].value_counts()

print('Number of patients diagnosed with liver disease: ',LD)

print('Number of patients not diagnosed with liver disease: ',NLD)


# Frequency of patients based on their gender


sns.countplot(data=liver_df, x = 'Gender', label='Count')


M, F = liver_df['Gender'].value_counts()

print('Number of patients that are male: ',M)
print('Number of patients that are female: ',F)

liver_df[['Gender',                                    'Dataset','Age']].groupby(['Dataset','Gender'],
as_index=False).count().sort_values(by='Dataset', ascending=False)

liver_df[['Gender',                                    'Dataset','Age']].groupby(['Dataset','Gender'],
as_index=False).mean().sort_values(by='Dataset', ascending=False)

g = sns.FacetGrid(liver_df, col="Dataset", row="Gender", margin_titles=True)
g.map(plt.hist, "Age", color="red")
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Disease by Gender and Age');

g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Direct_Bilirubin", "Total_Bilirubin", edgecolor="w")
plt.subplots_adjust(top=0.9)

"""- There seems to be direct relationship between Total_Bilirubin and Direct_Bilirubin. We
have the possibility of removing one of this feature."""

g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Aspartate_Aminotransferase",                    "Alamine_Aminotransferase",
edgecolor="w")
plt.subplots_adjust(top=0.9)
```

```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)

g.map(plt.scatter,"Alkaline_Phosphotase", "Alamine_Aminotransferase",  edgecolor="w")

plt.subplots_adjust(top=0.9)


g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)

g.map(plt.scatter,"Total_Protiens", "Albumin",  edgecolor="w")

plt.subplots_adjust(top=0.9)


g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)

g.map(plt.scatter,"Albumin", "Albumin_and_Globulin_Ratio",  edgecolor="w")

plt.subplots_adjust(top=0.9)


g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)

g.map(plt.scatter,"Albumin_and_Globulin_Ratio", "Total_Protiens",  edgecolor="w")

plt.subplots_adjust(top=0.9)


liver_df.head(3)


"""- Convert categorical variable "Gender" to indicator variables"""


pd.get_dummies(liver_df['Gender'], prefix = 'Gender').head()
```

```
liver_df = pd.concat([liver_df,pd.get_dummies(liver_df['Gender'], prefix = 'Gender')], axis=1)

liver_df.head()

liver_df.describe()

liver_df[liver_df['Albumin_and_Globulin_Ratio'].isnull()]

liver_df["Albumin_and_Globulin_Ratio"] =
liver_df.Albumin_and_Globulin_Ratio.fillna(liver_df['Albumin_and_Globulin_Ratio'].mean())

#liver_df[liver_df['Albumin_and_Globulin_Ratio'] == 0.9470639032815201]

# The input variables/features are all the inputs except Dataset.
# The prediction or label is 'Dataset' that determines whether the patient has liver disease
or not.
# Dropping Gender and Dataset

X = liver_df.drop(['Gender','Dataset'], axis=1)
X.head(3)

y = liver_df['Dataset']
```

```
# 1 for liver disease; 2 for no liver disease

# Correlation

liver_corr = X.corr()
liver_corr

plt.figure(figsize=(30, 30))
sns.heatmap(liver_corr, cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws={'size': 15},cmap= 'coolwarm')
plt.title('Correlation between features');

"""The above correlation also indicates the following correlation
- Total_Protiens & Albumin
- Alamine_Aminotransferase & Aspartate_Aminotransferase
- Direct_Bilirubin & Total_Bilirubin
- There is some correlation between Albumin_and_Globulin_Ratio and Albumin. But its not as high as Total_Protiens & Albumin

# Machine Learning
"""
```

```
# Importing modules

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report,confusion_matrix

from sklearn import linear_model

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC, LinearSVC

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
BaggingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.linear_model import Perceptron

from sklearn.linear_model import SGDClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural_network import MLPClassifier


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)

print (X_train.shape)

print (y_train.shape)

print (X_test.shape)

print (y_test.shape)
```

```
"""# Logistic Regression"""

# Create logistic regression object

logreg = LogisticRegression()

# Train the model using the training sets and check score

logreg.fit(X_train, y_train)

#Predict Output

log_predicted= logreg.predict(X_test)
logreg_score = round(logreg.score(X_train, y_train) * 100, 2)
logreg_score_test = round(logreg.score(X_test, y_test) * 100, 2)

#Equation coefficient and Intercept

print('Logistic Regression Training Score: \n', logreg_score)
print('Logistic Regression Test Score: \n', logreg_score_test)
print('Coefficient: \n', logreg.coef_)
print('Intercept: \n', logreg.intercept_)
print('Accuracy: \n', accuracy_score(y_test,log_predicted))

print('Confusion Matrix: \n', confusion_matrix(y_test,log_predicted))

print('Classification Report: \n', classification_report(y_test,log_predicted))


sns.heatmap(confusion_matrix(y_test,log_predicted),annot=True,fmt="d")


coeff_df = pd.DataFrame(X.columns)

coeff_df.columns = ['Feature']

coeff_df["Correlation"] = pd.Series(logreg.coef_[0])

pd.Series(logreg.coef_[0])


coeff_df.sort_values(by='Correlation', ascending=False)


"""# Gaussian Naive Bayes"""


# Create gaussian object


gaussian = GaussianNB()

gaussian.fit(X_train, y_train)


#Predict Output

gauss_predicted = gaussian.predict(X_test)
```

```
gauss_score = round(gaussian.score(X_train, y_train) * 100, 2)

gauss_score_test = round(gaussian.score(X_test, y_test) * 100, 2)

print('Gaussian Score: \n', gauss_score)

print('Gaussian Test Score: \n', gauss_score_test)

print('Accuracy: \n', accuracy_score(y_test, gauss_predicted))

print(confusion_matrix(y_test,gauss_predicted))

print(classification_report(y_test,gauss_predicted))


sns.heatmap(confusion_matrix(y_test,gauss_predicted),annot=True,fmt="d")


"""# Random Forest"""


# create random_forest object


random_forest                                                                   =
RandomForestClassifier(max_depth=3,n_estimators=56,criterion='entropy')

random_forest.fit(X_train, y_train)


#Predict Output


rf_predicted = random_forest.predict(X_test)
```

```
random_forest_score = round(random_forest.score(X_train, y_train) * 100, 2)

random_forest_score_test = round(random_forest.score(X_test, y_test) * 100, 2)

print('Random Forest Score: \n', random_forest_score)

print('Random Forest Test Score: \n', random_forest_score_test)

print('Accuracy: \n', accuracy_score(y_test,rf_predicted))

print(confusion_matrix(y_test,rf_predicted))

print(classification_report(y_test,rf_predicted))


finX = liver_df[['Total_Protiens','Albumin', 'Gender_Male']]

finX.head(4)


"""# Logistic Regression"""


X_train, X_test, y_train, y_test = train_test_split(finX, y, test_size=0.30, random_state=101)


# Create logistic regression object


logreg = LogisticRegression()


# Train the model using the training sets and check score
```

```
logreg.fit(X_train, y_train)

# Predict Output

log_predicted= logreg.predict(X_test)

logreg_score = round(logreg.score(X_train, y_train) * 100, 2)
logreg_score_test = round(logreg.score(X_test, y_test) * 100, 2)

# Equation coefficient and Intercept

print('Logistic Regression Training Score: \n', logreg_score)
print('Logistic Regression Test Score: \n', logreg_score_test)
print('Coefficient: \n', logreg.coef_)
print('Intercept: \n', logreg.intercept_)
print('Accuracy: \n', accuracy_score(y_test,log_predicted))
print('Confusion Matrix: \n', confusion_matrix(y_test,log_predicted))
print('Classification Report: \n', classification_report(y_test,log_predicted))

sns.heatmap(confusion_matrix(y_test,log_predicted),annot=True,fmt="d")

"""# Decision Tree Classifier"""
```

```
# Create decision tree object

dt=DecisionTreeClassifier()

# Train the model using the training sets and check score

dt.fit(X_train,y_train)

# Predict Output

y_pred=dt.predict(X_test)

dt_score = round(dt.score(X_train, y_train) * 100, 2)
dt_test = round(dt.score(X_test, y_test) * 100, 2)

from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

```
"""# Model evaluation"""


# We can now rank our evaluation of all the models to choose the best one for our problem.


models = pd.DataFrame({
    'Model': [ 'Logistic Regression', 'Gaussian Naive Bayes','Random Forest','Decision Tree'],
    'Score': [ logreg_score, gauss_score, random_forest_score,dt_score],
    'Test Score': [ logreg_score_test, gauss_score_test, random_forest_score_test,dt_test]})
models.sort_values(by='Test Score', ascending=False)
```