İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ
2010
FEN BİLİMLERİ ENSTİTÜSÜ

Heart Attack Analysis Detection System Using Machine Learning Methods

Department of Software Engineering (Distance Education)

SERDAR YANIK

ORCID 0000-0000-0000-0000

Project Advisor: Dr. Öğr. Üyesi Serpil Yılmaz

January, 2024

Heart Attack Analysis Detection System Using Machine Learning Methods

Summary

In order to investigate Heart Attack Analysis and Detection Using Machine Learning Methods, models that predict the type of news in a new condition determined by using Machine Learning Models have been studied.

In particular, the performance of various classifiers, including logistic regression, K-nearest neighbour (KNN), support vector machine (SVM), Naive Bayes and decision tree, is compared. In the experiments using a real-life dataset, logistic regression and SVM gave the best results with a test accuracy of 90%. Naive Bayes achieved an accuracy of 86.67%, KNN 83.33% and decision tree 63.33%. These results suggest that logistic regression and SVM can be suitable and effective machine learning models for predicting heart attack risk.

Keywords: Engineering, artificial intelligence, machine learning, Heart Attack Analysis, Prediction

Acknowledgements

I would like to thank Dr. Serpil Yılmaz for her efforts in the emergence of the project work.

Contents

Figures List

Chapter 1

Introduction

1. Purpose of the Project

A decision tree model, which generates an algorithm that resembles human thought processes, has recently been considered a suitable statistical approach for the development of clinical prediction models [1,2]. We aimed to develop an easy-to-use prediction model for heart attack analysis by decision tree analysis.

To determine to which class a new instance belongs, the naive Bayes algorithm uses a simplified version of the Bayes formula [3]. Each class's posterior probability is calculated using the feature values present in the instance. The instance with the highest probability is then assigned to the class.

SVM has been widely utilized and routinely delivers equivalent or greater performance compared to other machine learning algorithms [4,6]. It was first created by Vapnik and colleagues [4,5]. Its principal idea is to use a kernel function to transfer data points to a high-dimensional space, from which a hyper plane can be used to segregate the data points.

Characteristics of observations are gathered for both training and test datasets, and the purpose of the kNN classifier is to categorize unlabeled observations by placing them in the class of the most similar labeled examples [7]. The diagnostic efficacy of the kNN algorithm is greatly influenced by the selection of k.thus finding a compromise between overfitting and underfitting is crucial for selecting the right k value [7, 8]. According to some writers, k should be set to the square root of the training dataset's observation count [7,9].

Logistic regression may include only one or multiple independent variables, although examining multiple variables is generally more informative because it reveals the unique contribution of each variable after adjusting for the others. [10].

The aim of this project is Heart Attack Analysis Detection System Using Machine Learning Methods. For this project, Heart Attack Analysis & Prediction Dataset containing 303 samples was used. From this dataset, machine learning models were created using sklearn, pandas, numpy and their sub-libraries written in Python language. The dataset was divided into training and test data and trained with these models. After the accuracy rates of the training and test data were determined, a new data was entered and tested and these rates were presented in the project.

2. Steps of the Project

2.1. Inclusion of Libraries in the System

First of all, I included the libraries necessary to read the Heart Attack Analysis & Prediction Dataset and create the models.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

Figure 2.1 1: Libraries required to read the data

2.2. Inclusion of the Data Set in the Project

To read the dataset, I used the read_csv() method to read the dataset and assigned it to a variable named df. I used the head() method to display the first 5 rows of the dataset.

```
df = pd.read_csv("/kaggle/input/heart-attack-analysis-prediction-dataset/heart.csv")
```
Python

```
df.head()
```
Python

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 2.2. 1: Data Table

2.3. Missing and Unique Value Analysis

To make missing value analysis I used df.isnull().sum() code.

```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

Figure 2.3. 1: Missing Value Analysis

To make missing value analysis I used the folowing code.

for i in list(df.columns):

print("{} -- {}".format(i, df[i].value_counts().shape[0]))

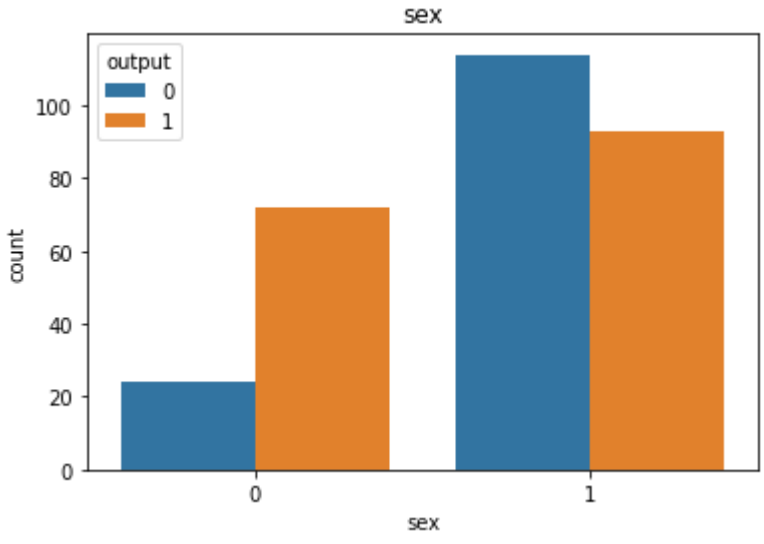
```
age -- 41
sex -- 2
cp -- 4
trtbps -- 49
chol -- 152
fbs -- 2
restecg -- 3
thalachh -- 91
exng -- 2
oldpeak -- 40
slp -- 3
caa -- 5
thall -- 4
output -- 2
```

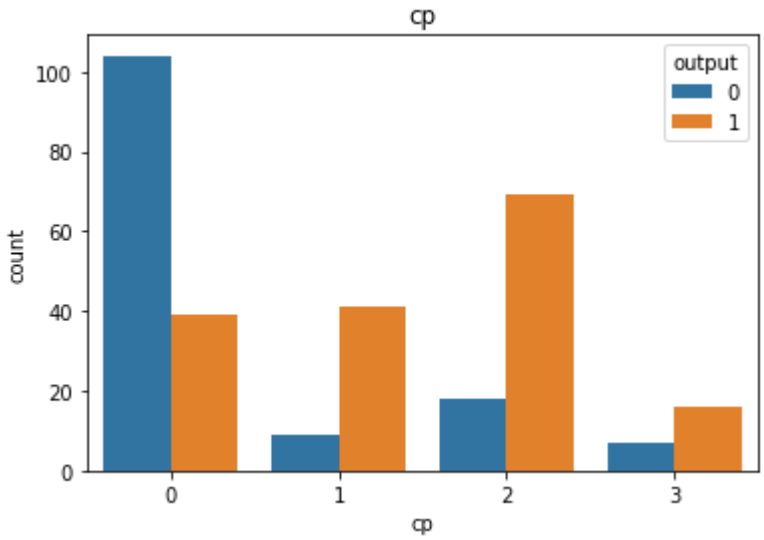
Figure 2.3. 2: Unique Value Analysis

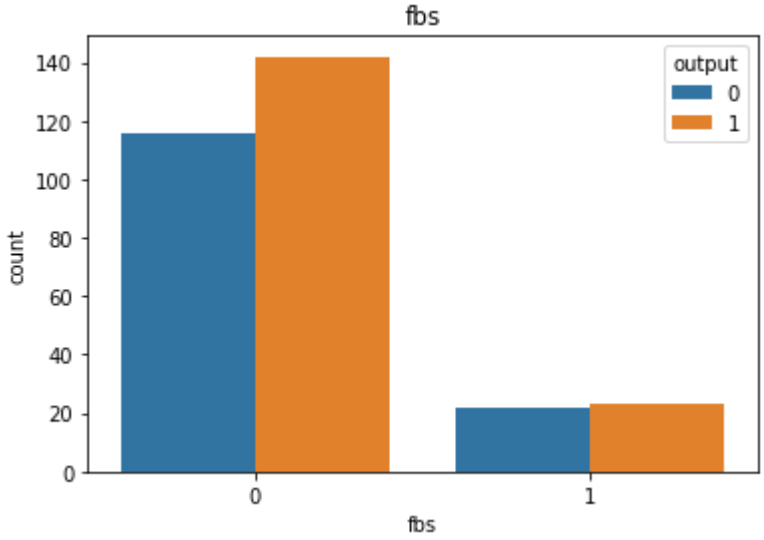
2.4. Categorical and Numeric Feature Analysis

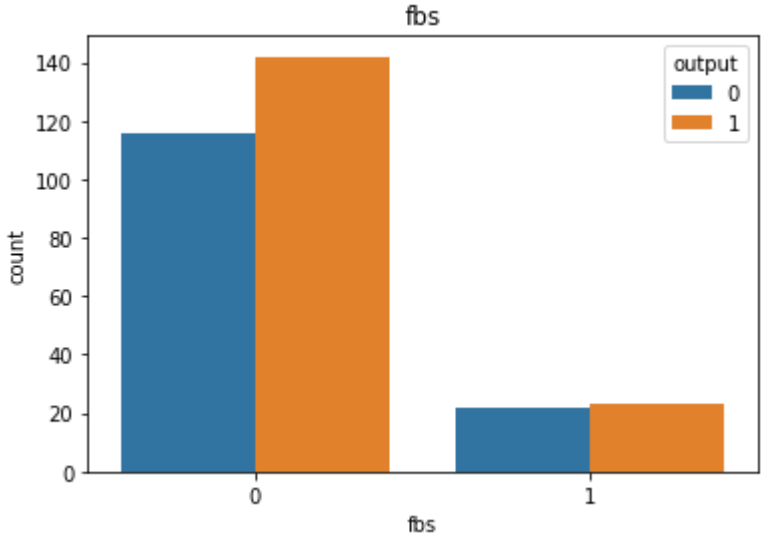
I made a list of categorical variables and visualized them with the help of the seaborn library. I used the following codes for this process.

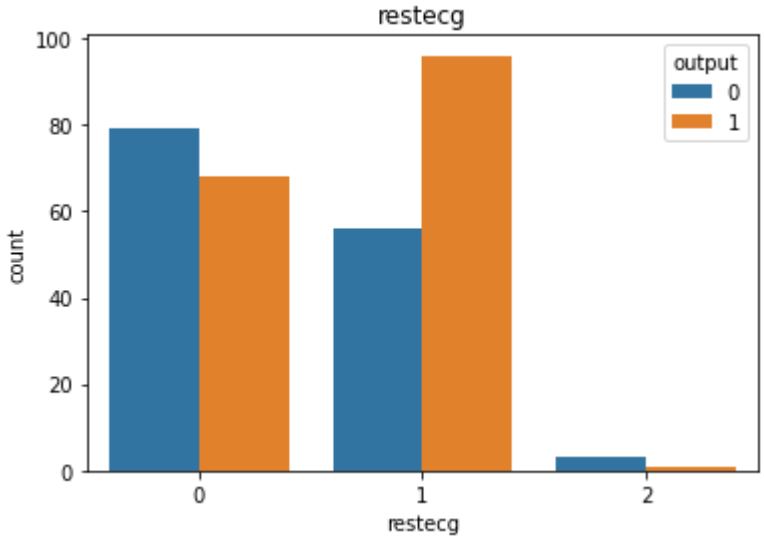
```
categorical_list = ["sex", "cp","fbs","restecg","exng","slp","caa","thall","output"]

df_categoric = df.loc[:, categorical_list]
for i in categorical_list:
    plt.figure()
    sns.countplot(x = i, data = df_categoric, hue = "output")
    plt.title(i)
```

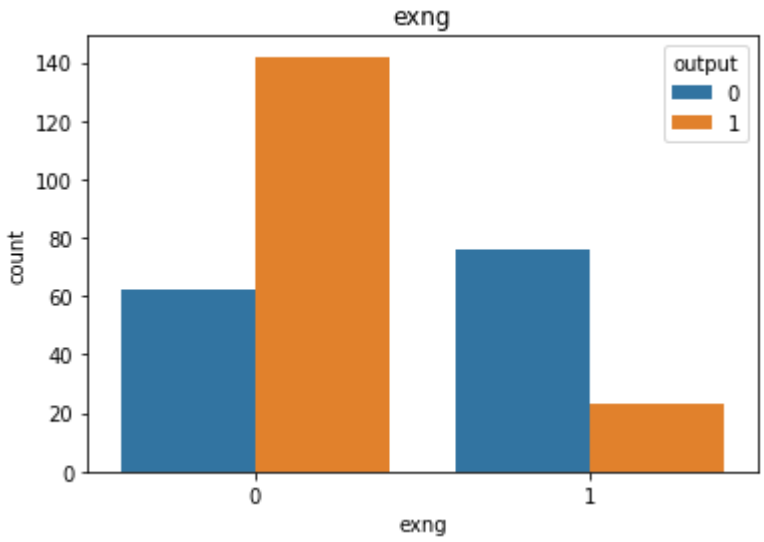


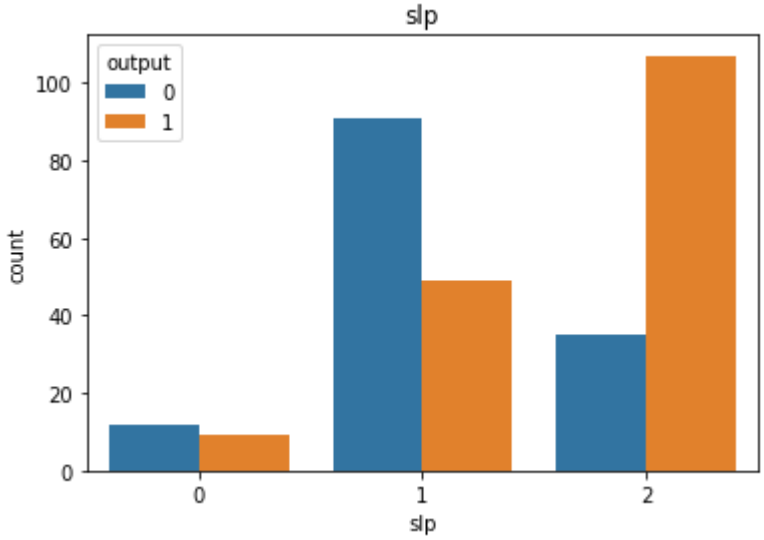

cp
output
0
1
count
0
20
40
60
80
100
0
1
2
3
cp

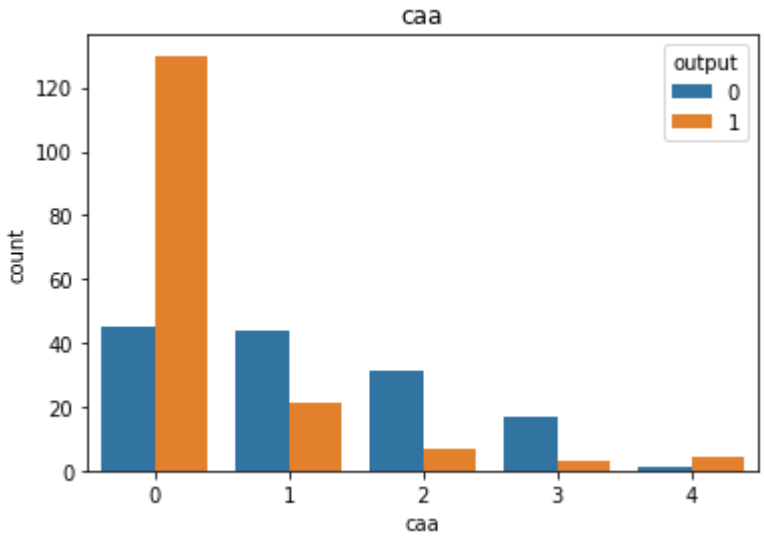

fbs
output
0
1
count
0
20
40
60
80
100
120
140
0
1
fbs

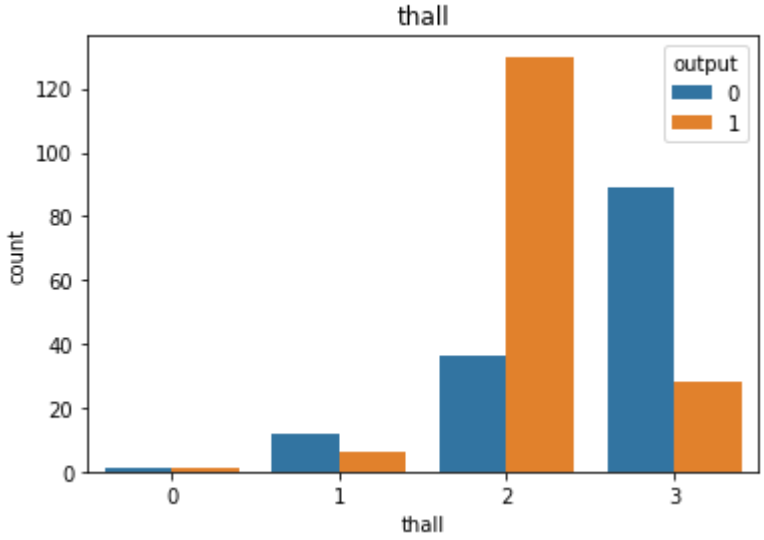

fbs
output
0
1
count
0
20
40
60
80
100
120
140
0
1
fbs

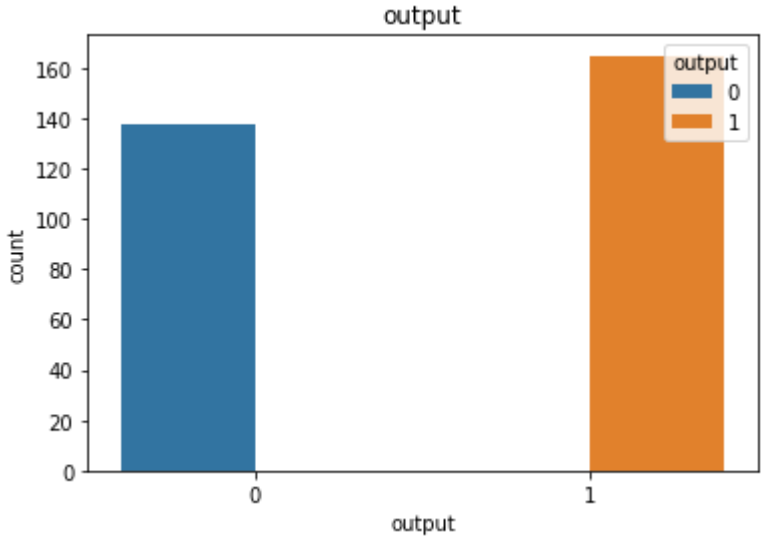

restecg
output
0
1
count
restecg
0
1
2
0
20
40
60
80
100


exng
output
0
1
count
exng
0
1
0
20
40
60
80
100
120
140


slp
output
0
1
count
slp
0
1
2
0
20
40
60
80
100


caa
output
0
1
count
0
20
40
60
80
100
120
0
1
2
3
4
caa


thall
output
0
1
count
0
20
40
60
80
100
120
0
1
2
3
thall


output
output
0
1
count
0
20
40
60
80
100
120
140
160
0
1
output

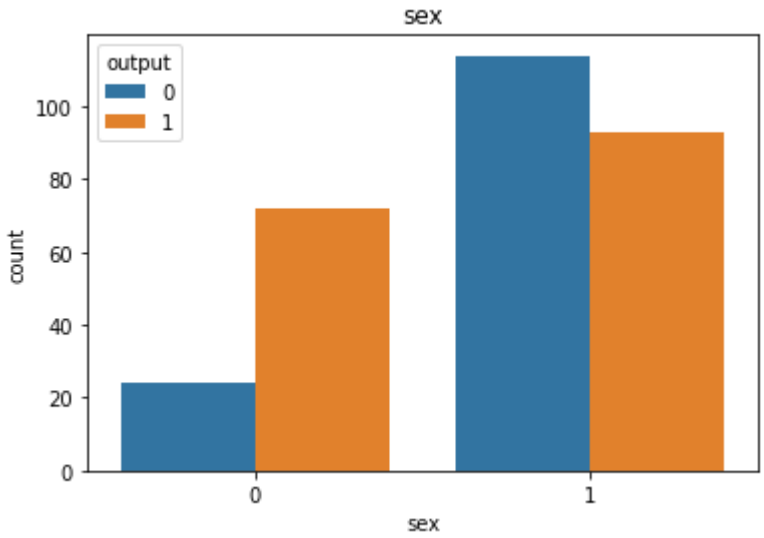


Figure 2.4. 1: Categorical Feature Analysis

I made a list of numeric variables and visualized them with the help of the seaborn library. I used the following codes for this process.

```
numeric_list = ["age", "trtbps","chol","thalachh","oldpeak","output"]

 df_numeric = df.loc[:, numeric_list]

sns.pairplot(df_numeric, hue = "output", diag_kind = "kde")
plt.show()
```

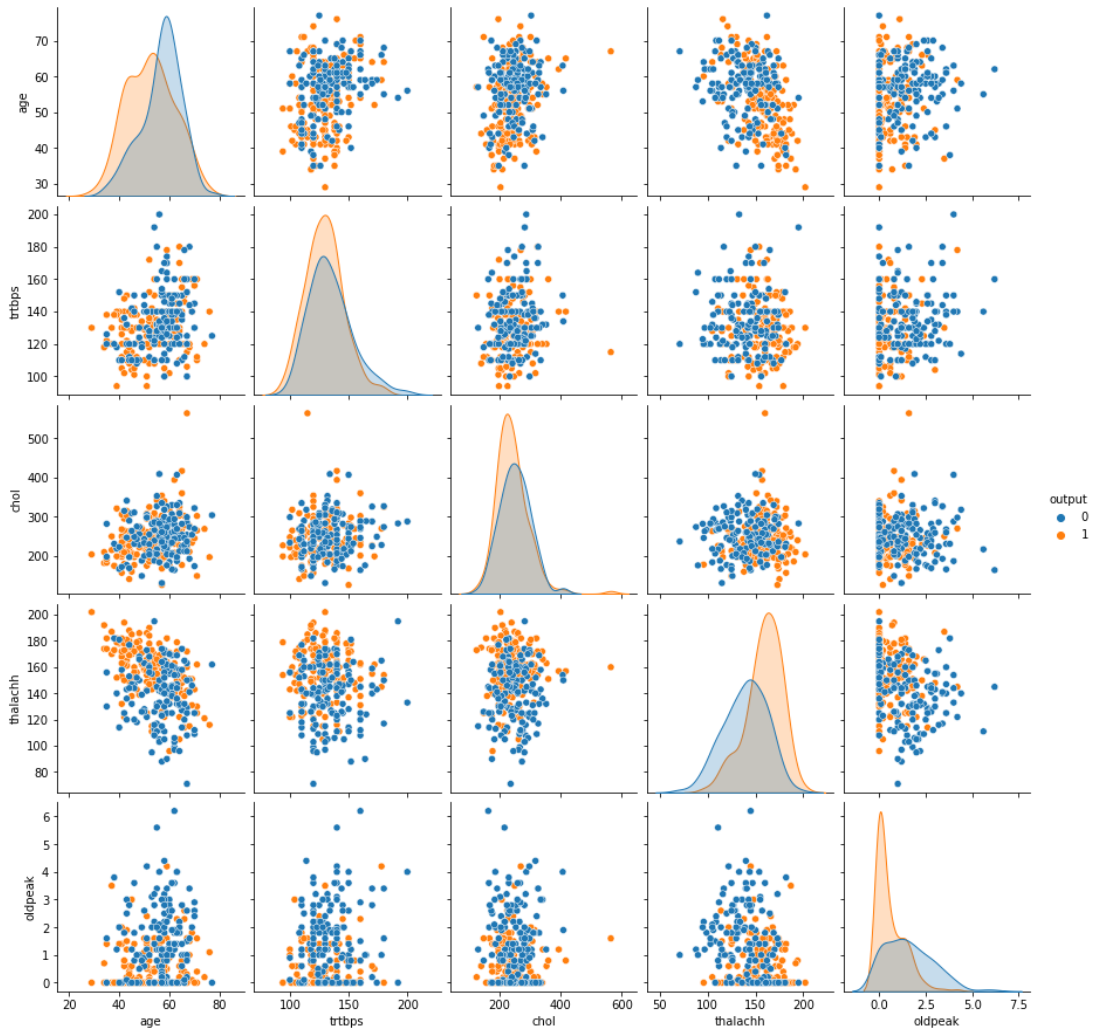


Figure 2.4. 2: Bivariate data analysis with scatter plot

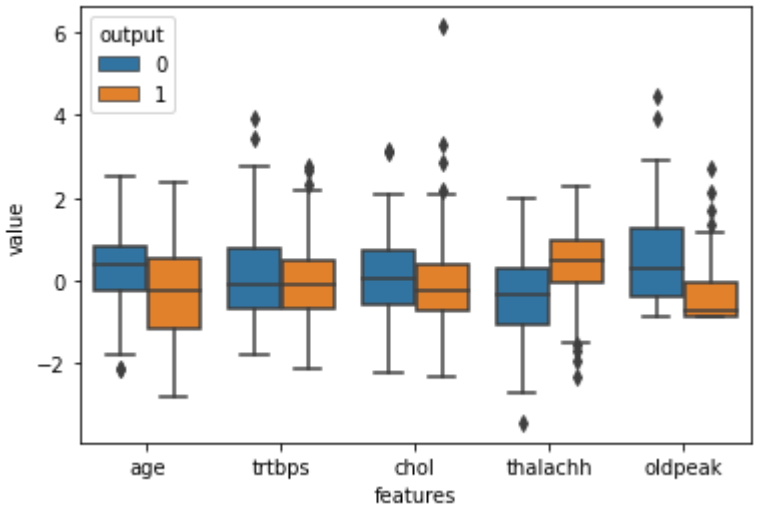


Figure 2.4. 3: Box Plot Analysis d6

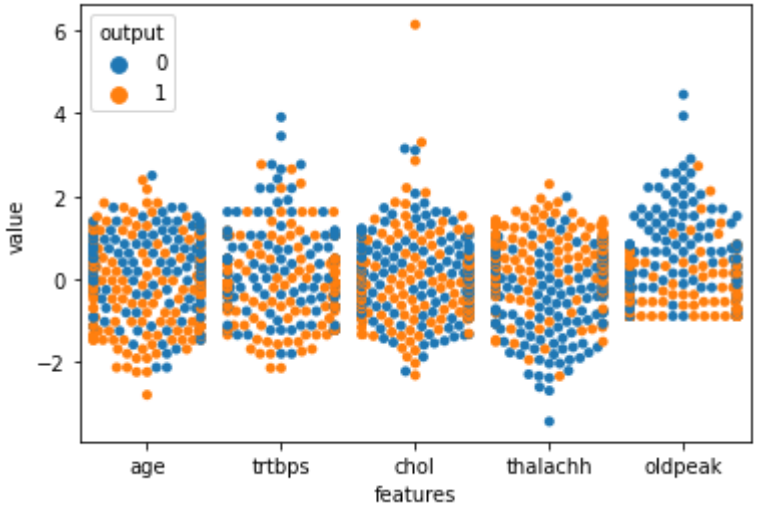


Figure 2.4. 4: Swarm Plot Analysis

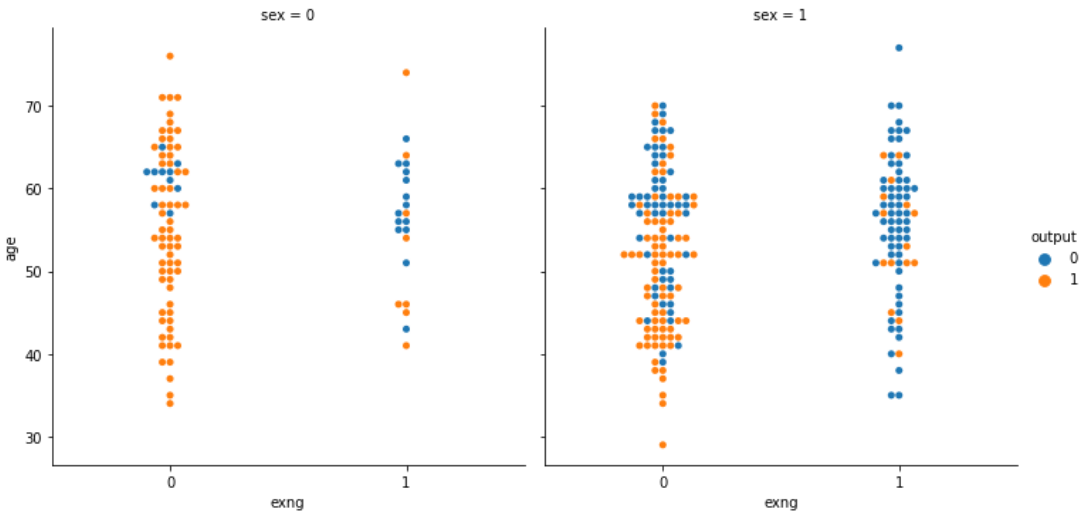


Figure 2.4. 5: Cat Plot Analysis

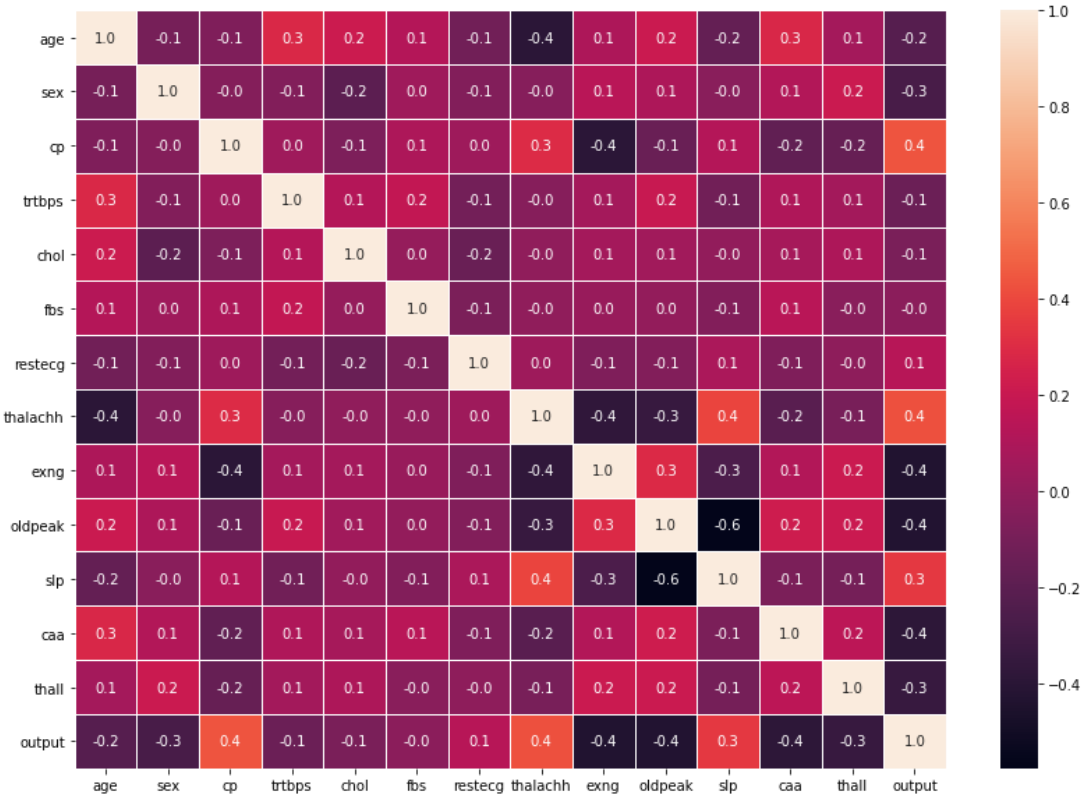


Figure 2.4. 6: Correlation Analysis

2.5. Outlier Detection

I used the following code to find outlier values in the data set.

```
numeric_list = ["age", "trtbps","chol","thalachh","oldpeak"]
df_numeric = df.loc[:, numeric_list]

for i in numeric_list:


    Q1 = np.percentile(df.loc[:, i],25)
    Q3 = np.percentile(df.loc[:, i],75)

    IQR = Q3 - Q1

    print("Old shape: ", df.loc[:, i].shape)


    upper = np.where(df.loc[:, i] >= (Q3 +2.5*IQR))


    lower = np.where(df.loc[:, i] <= (Q1 - 2.5*IQR))

    print("{} -- {}".format(upper, lower))

    try:
        df.drop(upper[0], inplace = True)
    except: print("KeyError: {} not found in axis".format(upper[0]))

    try:
        df.drop(lower[0], inplace = True)
    except:  print("KeyError: {} not found in axis".format(lower[0]))
```

```
    print("New shape: ", df.shape)
```

```
Old shape:  (303,)
(array([], dtype=int64),) -- (array([], dtype=int64),)
New shape:  (303, 14)
Old shape:  (303,)
(array([223, 248]),) -- (array([], dtype=int64),)
New shape:  (301, 14)
Old shape:  (301,)
(array([85]),) -- (array([], dtype=int64),)
New shape:  (300, 14)
Old shape:  (300,)
(array([], dtype=int64),) -- (array([], dtype=int64),)
New shape:  (300, 14)
Old shape:  (300,)
(array([203, 220]),) -- (array([], dtype=int64),)
New shape:  (298, 14)
```

Figure 2.5. 1: Outlier Detection

2.6. Encoding Categorical Columns, Scaling and Separating Data into Training and Test Data

Encoding categorical columns is an important step in preparing data for machine learning. By converting categorical values to numerical values, machine learning models are able to understand and learn from the data more effectively.

A machine learning model is trained to predict the risk of heart attack based on a number of factors, including age, sex, blood pressure, cholesterol levels, and smoking status.

The smoking status column is a categorical column with two possible values: "smoker" and "non-smoker". To encode this column, one-hot encoding is used. This creates two new binary columns: "smoker" and "non-smoker". The value of each column is 1 if the observation is a smoker/non-smoker and 0 otherwise.

The encoded smoking status columns are then added to the dataset along with the other features. The model is then trained on the dataset.

When the model is evaluated on a new dataset, it is found to perform better than a model that was trained on the dataset without encoding the categorical columns. This shows how encoding categorical columns can improve the performance of machine learning models.

I used the following code for encoding categorical columns, scaling and separating data into training and test data

df1 = pd.get_dummies(df1, columns = categorical_list[:-1], drop_first = True)

X = df1.drop(["output"], axis = 1)

y = df1[["output"]]

scaler = StandardScaler()

scaler

X[numeric_list[:-1]] = scaler.fit_transform(X[numeric_list[:-1]])

X.head()

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 3)
print("X_train: {}".format(X_train.shape))
print("X_test: {}".format(X_test.shape))
print("y_train: {}".format(y_train.shape))
print("y_test: {}".format(y_test.shape))
```

```
X_train: (268, 22)
X_test: (30, 22)
y_train: (268, 1)
y_test: (30, 1)
```

Figure 2.6. 1: Separation of Training and Test Data

2.7. Building, Training and Testing Models

I include the libraries needed to build the model.

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve
```

Figure 2.7. 1: Required Libraries for Model

2.7.1. Logistic Regression Model

The test accuracy rate of the data trained for the Logistic Regression Model and the codes are as follows.

```
logreg = LogisticRegression()
logreg
logreg.fit(X_train, y_train)
y_pred_prob = logreg.predict_proba(X_test)
y_pred_prob
y_pred = np.argmax(y_pred_prob, axis = 1)
y_pred
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.9
```

Figure 2.7.1. 1: Logistic Regression Model

```
plt.plot([0,1],[0,1],"k--")
plt.plot(fpr, tpr, label = "Logistic Regression")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Logistic Regression ROC Curve")
plt.show()
```

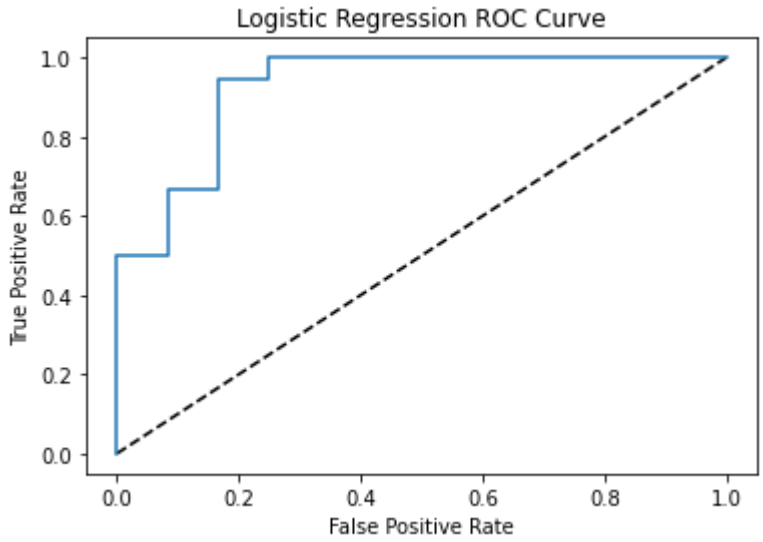


Figure 2.7.1. 2: ROC Curve

2.7.2. K-Nearest Neighbors (KNN) Algorithm Model

The test accuracy rate of the data trained for the K-Nearest Neighbors (KNN) Algoritm Model and the codes are as follows.

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.8333333333333334
```

Figure 2.7.2: K-Nearest Neighbors (KNN) Algorithm Model

2.7.3. Support Vector Machine (SVM) Algorithm Model

The test accuracy rate of the data trained for the Support Vector Machine (SVM) Algoritm Model and the codes are as follows.

```
svm_classifier = SVC(probability=True)
svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.9
```

Figure 2.7.3: Support Vector Machine (SVM) Algoritm Model

2.7.4. Naive Bayes Algorithm Model

The test accuracy rate of the data trained for the Naive Bayes Algorithm Model and the codes are as follows.

```
from sklearn.naive_bayes import GaussianNB
naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train, y_train)
y_pred = naive_bayes_classifier.predict(X_test)
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.8666666666666667
```

Figure 2.7.4: Naive Bayes Algorithm Model

2.7.5. Decision Tree Algorithm Model

The test accuracy rate of the data trained for the Decision Tree Algorithm Model and the codes are as follows.

```
decision_tree_classifier = DecisionTreeClassifier()
decision_tree_classifier.fit(X_train, y_train)
y_pred = decision_tree_classifier.predict(X_test)
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.6333333333333333
```

Figure 2.7.5: Decision Tree Algorithm

3. Conclusion

In this study, various machine learning models were used to predict the risk of heart attack. Logistic regression and SVM models gave the best results with a test accuracy of 90%, while Naive Bayes achieved 86.67%, KNN 83.33% and decision tree 63.33% accuracy.

The results show that logistic regression and SVM are highly effective methods for predicting heart attack risk. The high accuracy rates of these models can enable early diagnosis and intervention by accurately predicting the risk of heart attack.

The excellent results of SVM and logistic regression can be attributed to a number of factors. First off, a variety of factors that influence the likelihood of a heart attack can be considered by these models. Secondly, these models excel at simulating intricate interactions.

The superior performance of logistic regression and SVM over Naive Bayes, KNN, and decision tree models can be attributed to multiple factors. The Naive Bayes model, for instance, makes the assumption that characteristics are independent. This might not always be the case. The neighbors chosen can have an impact on the KNN model. When the decision tree model gets complicated, it can be challenging to understand.

This study has shown that machine learning methods can be highly effective tools for predicting heart attack risk. In particular, logistic regression and SVM models stand out with their high accuracy rates.

References

1. Joung J, Oh JS, Yoon JM, Ko KO, Yoo GH, Cheon EJ. A decision tree model for predicting intravenous immunoglobulin resistance and coronary artery involvement in Kawasaki disease. BMC Pediatr. 2022 Aug 5;22(1):474. doi: 10.1186/s12887-022-03533-6. PMID: 35931986; PMCID: PMC9354345.
2. Sleeper LA, Minich LL, McCrindle BM, Li JS, Mason W, Colan SD, Atz AM, et al. Evaluation of Kawasaki disease riskscoring systems for intravenous immunoglobulin resistance. *J Pediatr.* 2011;**158**:831–5. doi: 10.1016/j.jpeds.2010.10.031.
3. Jiang W, Shen Y, Ding Y, Ye C, Zheng Y, Zhao P, Liu L, Tong Z, Zhou L, Sun S, Zhang X, Teng L, Timko MP, Fan L, Fang W. A naive Bayes algorithm for tissue origin diagnosis (TOD-Bayes) of synchronous multifocal tumors in the hepatobiliary and pancreatic system. Int J Cancer. 2018 Jan 15;142(2):357-368. doi: 10.1002/ijc.31054. Epub 2017 Oct 16. Erratum in: Int J Cancer. 2018 Jul 1;143(1):E2. PMID: 28921531.
4. Zhang D, Xiao J, Zhou N, Zheng M, Luo X, Jiang H, Chen K. A Genetic Algorithm Based Support Vector Machine Model for Blood-Brain Barrier Penetration Prediction. Biomed Res Int. 2015;2015:292683. doi: 10.1155/2015/292683. Epub 2015 Oct 4. PMID: 26504797; PMCID: PMC4609370.
5. Vapnik V. N. *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer; 1995.
6. Heikamp K., Bajorath J. Support vector machines for drug discovery. *Expert Opinion on Drug Discovery.* 2014;9(1):93–104. doi: 10.1517/17460441.2014.866943.
7. Zhang Z. Introduction to machine learning: k-nearest neighbors. Ann Transl Med. 2016 Jun;4(11):218. doi: 10.21037/atm.2016.03.37. PMID: 27386492; PMCID: PMC4916348

8. Zhang Z. Too much covariates in a multivariable model may cause the problem of overfitting. *J Thorac Dis* 2014;6:E196-7
9. Lantz B. Machine learning with R. 2nd ed. Birmingham: Packt Publishing; 2015:1.
10. Stoltzfus JC. Logistic regression: a brief primer. Acad Emerg Med. 2011 Oct;18(10):1099-104. doi: 10.1111/j.1553-2712.2011.01185.x. PMID: 21996075.)