**HEART DISEASE PREDICTION DOCUMENTATION**

The provided code performs a comprehensive analysis of the heart disease dataset. Here's a detailed breakdown of the code and its functionalities:

Importing Libraries:

\* pandas (pd): Used for data manipulation and analysis.

\* matplotlib.pyplot (plt): Used for creating visualizations.

\* plotly.express (px): Used for creating interactive visualizations (optional).

\* sklearn.model\_selection: Used for splitting data into training and testing sets.

\* sklearn.linear\_model: Used for training linear regression and logistic regression models.

\* sklearn.metrics: Used for evaluating the performance of machine learning models.

\* sklearn.tree: Used for training decision tree classifier models.

\* sklearn.neighbors: Used for training K-Nearest Neighbors (KNN) classifier models.

\* sklearn.cluster: Used for performing K-means clustering.

\* sklearn.discriminant\_analysis: Used for performing Linear Discriminant Analysis (LDA).

Data Loading and Exploration:

\* Reading the data: The code reads the heart disease dataset from a CSV file named 'heart.csv' using the pd.read\_csv function.

\* Exploring the data:

\* The code displays the first few rows of the data using df.head().

\* The code displays the last few rows of the data using df.tail().

\* The code gets some summary statistics of the data using df.describe().

\* The code checks for missing values in the data using df.isna().sum().

\* The code displays the data types and column names of the DataFrame.

Data Preprocessing:

\* Feature selection: The code drops three columns from the DataFrame based on their indices. These columns might not be relevant for prediction or might contain redundant information.

\* Data visualization: The code creates a histogram to visualize the distribution of the 'age' feature using px.histogram.

Model Training and Evaluation:

\* Splitting data into training and testing sets: The code splits the data into training and testing sets using train\_test\_split from sklearn.model\_selection. This helps evaluate the model's performance on unseen data.

\* Training a linear regression model:

\* The code creates a LinearRegression object from sklearn.linear\_model.

\* The model is trained on the training data using the fit method.

\* The model predicts the target variable (heart disease) for the training and testing sets using the predict method.

\* The code calculates the mean squared error (MSE) and R-squared score for both training and testing sets using mean\_squared\_error and r2\_score from sklearn.metrics to evaluate the model's performance.

\* Training a logistic regression model:

\* The code creates a LogisticRegression object with a maximum iteration limit of 1000.

\* The model is trained on the training data using the fit method.

\* The model predicts the target variable (presence or absence of heart disease) for the training and testing sets using the predict method.

\* The code calculates the accuracy, confusion matrix, precision, recall, and F1-score for the logistic regression model on the test set using functions from sklearn.metrics.

\* Training a decision tree classifier:

\* The code creates a DecisionTreeClassifier object with the entropy criterion.

\* The model is trained on the training data using the fit method.

\* The model predicts the target variable for the training and testing sets using the predict method.

\* The code calculates the accuracy, precision, recall, and F1-score for the decision tree model on the test set using functions from sklearn.metrics.

\* Training a K-nearest neighbors classifier:

\* The code creates a KNeighborsClassifier object with 5 neighbors.

\* The model is trained on the training data using the fit method.

\* The model predicts the target variable for the training and testing sets using the predict method.

\* The code calculates the accuracy, precision, recall, and F1-score for the KNN model on the test set using functions from sklearn.metrics.

Certainly, let's continue with the documentation for the provided code:

K-Means Clustering (Continued):

\* The code creates a KMeans object with three clusters (k=3) and a random state of 42 for reproducibility.

\* The fit method is called on the KMeans object to fit the model to the data.

\* The cluster labels and cluster centers are extracted from the fitted model.

\* The code visualizes the clustering results by plotting the data points with different colors for each cluster and marking the cluster centers with red 'x' markers.

K-Means Clustering Evaluation:

\* The code performs K-means clustering with k=2 clusters.

\* It calculates and prints the following evaluation metrics:

\* Inertia: A measure of within-cluster sum of squares.

\* Silhouette Score: Measures how similar a data point is to its own cluster compared to other clusters.

\* Davies-Bouldin Score: Measures the average similarity between each cluster and its most similar cluster.

\* Calinski-Harabasz Score: Measures the ratio of between-cluster variance to within-cluster variance.

\* Homogeneity Score: Measures the extent to which each cluster contains only members of a single class.

\* Completeness Score: Measures the extent to which all members of a given class are assigned to the same cluster.

\* V-measure Score: The harmonic mean of homogeneity and completeness.

\* The code visualizes the clustering results by plotting the data points with different colors for each cluster and marking the cluster centers with red 'x' markers.

Linear Discriminant Analysis (LDA):

\* The code performs LDA to project the data onto a single dimension that best separates the two classes (presence or absence of heart disease).

\* The code visualizes the projected data points for each class on a single line.

Key Observations and Insights:

\* The code provides a comprehensive analysis of the heart disease dataset, including data exploration, preprocessing, model training and evaluation, and clustering analysis.

\* The results can be used to gain insights into the factors that contribute to heart disease and to build predictive models for identifying individuals at risk.

\* The code demonstrates the use of various machine learning techniques, including linear regression, logistic regression, decision trees, K-nearest neighbors, K-means clustering, and LDA.

\* The code can be further extended by:

\* Performing hyperparameter tuning to optimize the performance of the models.

\* Evaluating different feature selection and preprocessing techniques.

\* Exploring other clustering algorithms, such as DBSCAN and hierarchical clustering.

\* Building more complex models, such as ensemble methods and deep learning models.

**Platform:**

Google Colab:

Google Colab is a cloud-based platform by Google that provides a Jupyter Notebook environment for writing and executing Python code. It is widely used for data science, machine learning, and deep learning tasks due to its accessibility and simplicity. Colab supports GPU and TPU acceleration, allowing users to train models faster without requiring powerful local hardware.

Users can access files directly from Google Drive, import datasets from various sources, and install custom Python libraries. Collaboration is seamless, enabling multiple users to edit and run notebooks simultaneously. With features like markdown support, code snippets, and visualization tools, Colab is a versatile tool for researchers, developers, and students.

**Code:**

import pandas as pd

df=pd.read\_csv('/content/heart (1).csv')

df

df.head()

df.tail()

df.describe()

df.shape

df.columns

df.isna().sum()

df.head()

df.tail()

df.shape

df['target'].value\_counts()

df.info()

df['target'].value\_counts()

df['chol'].value\_counts()

df['sex'].value\_counts()

df=df.drop([df.columns[3],df.columns[6],df.columns[5]],axis=1)

df.head()

import matplotlib.pyplot as plt

import plotly.express as px

fig=px.histogram(df,x='age')

fig.show()

x=df.drop(['target'],axis=1)

y=df['target']

print(x.head())

print(y.head())

from sklearn.model\_selection import train\_test\_split

train\_x,test\_x,train\_y,test\_y=train\_test\_split(x,y,test\_size=0.2,random\_state=2)

print(train\_x.head())

print(train\_y.head())

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error,r2\_score

model\_lr=LinearRegression()

model\_lr.fit(train\_x,train\_y)

y\_pred\_train\_lr=model\_lr.predict(train\_x)

y\_pred\_test\_lr=model\_lr.predict(test\_x)

y\_pred\_train\_lr=model\_lr.predict(train\_x)

y\_pred\_test\_lr=model\_lr.predict(test\_x)

train\_mse\_lr=mean\_squared\_error(train\_y,y\_pred\_train\_lr)

test\_mse\_lr=mean\_squared\_error(test\_y,y\_pred\_test\_lr)

train\_r2\_lr=r2\_score(train\_y,y\_pred\_train\_lr)

test\_r2\_lr=r2\_score(test\_y,y\_pred\_test\_lr)

print(train\_mse\_lr)

print(test\_mse\_lr)

print(train\_r2\_lr)

print(test\_r2\_lr)

from sklearn.linear\_model import LogisticRegression

model\_lr=LogisticRegression(max\_iter=1000)

model\_lr.fit(train\_x,train\_y)

y\_pred\_train=model\_lr.predict(train\_x)

y\_pred\_test=model\_lr.predict(test\_x)

print(y\_pred\_test)

print(y\_pred\_train)

from sklearn.metrics import accuracy\_score,confusion\_matrix

accuracy\_train=accuracy\_score(train\_y,y\_pred\_train)

print("Accuracy of the Logistic Regression model on your train",accuracy\_train)

accuracy\_test=accuracy\_score(test\_y,y\_pred\_test)

print("Accuracy of the Logistic Regression model on your test dataset",accuracy\_test)

confusion\_matrix(test\_y,y\_pred\_test)

tn,fp,fn,tp=confusion\_matrix(test\_y,y\_pred\_test).ravel()

print("TN:",tn)

print("FN:",fn)

print("FP:",fp)

print("TP:",tp)

accuracy=(tp+tn)/(tp+tn+fp+fn)

accuracy

confusion\_matrix(train\_y,y\_pred\_train)

Recall=tp/(tp+fn)

print("Recall of Logistic Regression is:",Recall)

Precision=tp/(tp+fp)

print("Precision of Logistic Regression is",Precision)

F1\_score=(2\*Recall+Precision)/(Recall+Precision)

print("F1\_score of Logistic Regression is:",F1\_score)

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier(criterion='entropy')

dt.fit(train\_x,train\_y)

y\_pred\_test\_dt=dt.predict(test\_x)

y\_pred\_train\_dt=dt.predict(train\_x)

accuracy\_train = accuracy\_score(train\_y,y\_pred\_train\_dt)

print("Accuracy of the Decision tree model on your train dataset,accuracy\_train")

accuracy\_test = accuracy\_score(test\_y,y\_pred\_test\_dt)

print("Accuracy of the Decision tree model on your test dataset,accuracy\_test")

from sklearn.metrics import precision\_score, recall\_score, f1\_score # Importing necessary functions

precision\_test=precision\_score(test\_y,y\_pred\_test\_dt)

print("precision of the Desion Tree on your test dataset",precision\_test)

recall\_test=recall\_score(test\_y,y\_pred\_test\_dt)

print("recall of the Decision tree on your test dataset",recall\_test)

f1\_score\_test=f1\_score(test\_y,y\_pred\_test\_dt)

print("F1 score of the Decision Tree on your test dataset",f1\_score\_test)

precision\_train=precision\_score(train\_y,y\_pred\_train\_dt)

print("precision of the DT on your train dataset",precision\_train)

recall\_train=recall\_score(train\_y,y\_pred\_train\_dt)

print("recall of the DT on your train dataset",recall\_train)

f1\_score\_train=f1\_score(train\_y,y\_pred\_train\_dt)

print("F1 score of the DT on your train dataset",f1\_score\_train)

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier # Corrected the class name

model\_knn = KNeighborsClassifier(n\_neighbors=5) # Corrected the class name

model\_knn.fit(train\_x, train\_y)

y\_pred\_train\_knn = model\_knn.predict(train\_x)

y\_pred\_test\_knn = model\_knn.predict(test\_x)

print(y\_pred\_train\_knn)

print(y\_pred\_test\_knn)

accuracy\_test = accuracy\_score(test\_y,y\_pred\_test\_knn)

print("Accuracy of the knn model on your test dataset,accuracy\_test")

accuracy\_train = accuracy\_score(train\_y,y\_pred\_train\_knn)

print("Accuracy of the knn model on your train dataset,accuracy\_train")

precision\_test=precision\_score(test\_y,y\_pred\_test\_knn)

print("precision of the knn on your test dataset",precision\_test)

recall\_test=recall\_score(test\_y,y\_pred\_test\_knn)

print("recall of the knn on your test dataset",recall\_test)

f1\_score\_test=f1\_score(test\_y,y\_pred\_test\_knn)

print("F1 score of the knn on your test dataset",f1\_score\_test)

print("Accuracy of the KNN model on your train dataset,accuracy\_train")

precision\_train=precision\_score(train\_y,y\_pred\_train\_knn)

print("precision of the knn on your train dataset",precision\_train)

recall\_train=recall\_score(train\_y,y\_pred\_train\_knn)

print("recall of the knn on your train dataset",recall\_train)

f1\_score\_test=f1\_score(train\_y,y\_pred\_train\_knn)

print("F1 score of the knn on your train dataset",f1\_score\_train)

from sklearn.cluster import KMeans

k=3

model\_kmeans=KMeans(n\_clusters=k,random\_state=42)

columns\_for\_cllustering=["age","chol"]

df\_for\_clustering=df[columns\_for\_cllustering]

model\_kmeans.fit(df\_for\_clustering)

from sklearn.cluster import KMeans

k=3

model\_kmeans=KMeans(n\_clusters=k,random\_state=42)

columns\_for\_cllustering=["age","chol"]

df\_for\_clustering=df[columns\_for\_cllustering]

model\_kmeans.fit(df\_for\_clustering)

cluster\_labels=model\_kmeans.labels\_

cluster\_centers=model\_kmeans.cluster\_centers\_

plt.figure(figsize=(8,6))

plt.scatter(df\_for\_clustering['age'],df\_for\_clustering['chol'],c=cluster\_labels,cmap='viridis',alpha=0.5)

plt.scatter(cluster\_centers[:,0],cluster\_centers[:,1],marker='x',s=100,color='red')

plt.xlabel("Age")

plt.ylabel("Chol")

plt.title("K-means Clustering")

#plt.legend() # legend might not be necessary if not specifying specific labels for the scatter plots

plt.show()

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score,davies\_bouldin\_score,calinski\_harabasz\_score,homogeneity\_score,completeness\_score,v\_measure\_score

import matplotlib.pyplot as plt

k=2

model\_KMeans = KMeans(n\_clusters=k,random\_state=42)

model\_KMeans.fit(x)

cluster\_labels=model\_KMeans.labels\_

inertia=model\_KMeans.inertia\_

print("Inertia:",inertia)

silhouette=silhouette\_score(x,cluster\_labels)

print("Silhouette\_Score:",silhouette)

davies\_bouldin=davies\_bouldin\_score(x,cluster\_labels)

print("Davies\_Bouldin Score:",davies\_bouldin)

calinski\_harabasz=calinski\_harabasz\_score(x,cluster\_labels)

print("Calinski-Harabasz Score:",calinski\_harabasz)

homogeneity=homogeneity\_score(y,cluster\_labels)

completeness=completeness\_score(y,cluster\_labels)

v\_measure=v\_measure\_score(y,cluster\_labels)

print("Homogeneity Score:",homogeneity)

print("Completeness Score:",completeness)

print("v\_measure:",v\_measure)

plt.figure(figsize=(8,6))

colors=['navy','turquoise']

lw=2

plt.figure(figsize=(8,6))

plt.scatter(df\_for\_clustering['age'],df\_for\_clustering['chol'],c=cluster\_labels,cmap='viridis',alpha=0.7)

plt.scatter(cluster\_centers[:,0],cluster\_centers[:,1],c='red',marker='x',s=100,label='centroid')

plt.xlabel('age')

plt.ylabel('chol')

plt.title('K-means Clustering')

plt.legend()

plt.show()

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

import matplotlib.pyplot as plt

import numpy as np

lda =LinearDiscriminantAnalysis(n\_components=1)

x\_lda=lda.fit\_transform(x,y)

plt.figure(figsize=(8,6))

colors=['navy','turquoise']

lw=2

for color,i,target\_name in zip(colors,[0,1],['No Heart Disease','Heart Disease']):

plt.scatter(x\_lda[y==1,0],np.zeros\_like(x\_lda[y==1,0]),color=color,alpha=.8,lw=lw,label=target\_name)

plt.legend(loc="best",shadow=False,scatterpoints=1)

plt.title("LDA of Heart Disease Dataset")

plt.xlabel("LD1")

plt.show()

I hope this documentation is helpful!