HEALTHCARE – CARDIOVASCULAR DATASET

COURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# read an excel file and convert
# into a dataframe object
data = pd.DataFrame(pd.read_excel("cep.xlsx"))
data.isna().sum()
data.describe()
continuousFeatures = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

def outlier_treatment(data, drop=False):
    for eachFeature in continuousFeatures:
        featureData = data[eachFeature]
        Q1, Q3 = np.percentile(featureData, [25, 75])
        IQR = Q3 - Q1
        outlierCalc = 1.5 * IQR
        outliers = featureData[~((featureData >= Q1 - outlierCalc) & (featureData <=
Q3 + outlierCalc))].index.tolist()
        if not drop:
            print('For the feature {}, No of Outliers is {}'.format(eachFeature,
len(outliers)))
        if drop:
            data.drop(outliers, inplace=True)
            print('Outliers from {} feature removed'.format(eachFeature))

outlier_treatment(data[continuousFeatures], drop=True)
```

```
# b.  Identify the data variables which are categorical
```

```
categorical_features = data.select_dtypes(include=['object']).columns.tolist()
sns.countplot(data=data, x='sex')
plt.show()
fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='sex', hue='target', data=data, palette='Set1', ax=ax)
ax.set_title("Distribution of Sex w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(['Female', 'Male'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
```

```
            color='white', weight='bold')

plt.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='cp', hue='target', data=data, palette='Set2', ax=ax)
ax.set_title("Distribution of chest pain w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(name, rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

plt.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='fbs', hue='target', data=data, palette='Set3', ax=ax)
ax.set_title("Distribution of Fasting Blood Sugar w.r.to Target", fontsize=13,
weight='bold')
ax.set_xticklabels(['True','False'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

plt.tight_layout()
plt.show()

# c.  Study the occurrence of CVD across the Age category
fig, ax = plt.subplots(figsize=(10, 6))

sns.countplot(x='age', hue='target', data=data, palette='Set2', ax=ax)
ax.set_title("Distribution of CVD across age categories", fontsize=13, weight='bold')
ax.set_xlabel("Age", fontsize=12)
ax.set_ylabel("Count", fontsize=12)
ax.legend(title="CVD", labels=["No", "Yes"])

totals = []
for i in ax.patches:
    totals.append(i.get_height())
```

```
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

plt.tight_layout()
plt.show()

#d. Study the composition of all patients with respect to the Sex category

sex_counts = data['sex'].value_counts()
labels = ['Male', 'Female']

fig, ax = plt.subplots(figsize=(6, 6))
ax.pie(sex_counts, labels=labels, autopct='%1.1f%%', startangle=90, colors=['skyblue',
'lightgreen'])
ax.set_title("Composition of Patients by Sex", fontsize=14, weight='bold')

plt.show()

# e.  Study if one can detect heart attacks based on anomalies in the resting blood
pressure (trestbps) of a patient
fig, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x='target', y='trestbps', data=data, ax=ax)
ax.set_title("Resting Blood Pressure (trestbps) vs. Heart Attack", fontsize=14,
weight='bold')
ax.set_xlabel("Heart Attack", fontsize=12)
ax.set_ylabel("Resting Blood Pressure (mm Hg)", fontsize=12)
ax.set_xticklabels(["No", "Yes"])

plt.tight_layout()
plt.show()
# f.  Describe the relationship between cholesterol levels and a target variable
plt.figure(figsize=(8, 6))
sns.histplot(data=data, x='chol', hue='target', kde=True, palette='Set2')
plt.title("Distribution of Cholesterol Levels by Target", fontsize=14, weight='bold')
plt.xlabel("Cholesterol Levels", fontsize=12)
plt.ylabel("CVD", fontsize=12)
plt.legend(title="Target", labels=["No", "Yes"])

plt.tight_layout()
plt.show()
plt.figure(figsize=(8, 6))
sns.boxplot(x='target', y='chol', data=data, palette='Set2')
plt.title("Cholesterol Levels by Target Variable", fontsize=14, weight='bold')
plt.xlabel("CVD", fontsize=12)
plt.ylabel("Cholesterol Levels", fontsize=12)
plt.xticks(ticks=[0, 1], labels=["No", "Yes"])

plt.tight_layout()
plt.show()
fig, ax = plt.subplots(figsize=(8, 5))
```

```
sns.countplot(x='slope', hue='target', data=data, palette='Set1', ax=ax)
ax.set_title("Slope Distribution w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(['Upsloping','Flat','Downsloping'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

plt.tight_layout()
plt.show()
fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='thal', hue='target', data=data, palette='Set1', ax=ax)
ax.set_title(" Effect of Thalassemia on CVD", fontsize=13, weight='bold')
ax.set_xticklabels(['reversable_defect','normal','fixed_defect'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

plt.tight_layout()
plt.show()
sns.set(style="ticks")
sns.pairplot(data, hue="target", palette="Set2")
plt.suptitle("Pair Plot of All Features", fontsize=14, fontweight='bold')
plt.show()
from sklearn.preprocessing import OneHotEncoder

data[categorical_features] = data[categorical_features].astype(str)

categorical_data = data[categorical_features]
numerical_data = data.drop(categorical_features, axis=1)

encoder = OneHotEncoder(sparse_output=False,drop='first')

categorical_encoded = encoder.fit_transform(categorical_data)

categorical_encoded_df = pd.DataFrame(categorical_encoded,
columns=encoder.get_feature_names_out(categorical_features))

encoded_data = pd.concat([numerical_data, categorical_encoded_df], axis=1)

#Seperate data as features and label
features = encoded_data.iloc[:,:-1].values
```

```
label = encoded_data.iloc[:,-1].values
from sklearn.preprocessing import StandardScaler
X_std=StandardScaler().fit_transform(encoded_data)
#Create train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(features,label,test_size=0.2,random_state=4)
#Apply LogisticRegression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)

print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
from sklearn.metrics import f1_score,precision_score,recall_score

predictTrain = model.predict(X_train)
predictTest = model.predict(X_test)

print("F1 Score of Training Set : ",f1_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",f1_score(y_test,predictTest,average=None))
print("F1 Score of Training Set :
",precision_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",precision_score(y_test,predictTest,average=None))
print("F1 Score of Training Set : ",recall_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",recall_score(y_test,predictTest,average=None))
from sklearn.metrics import classification_report
print(classification_report(y_train,predictTrain))
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score

# random forest algorithm
modelrf = RandomForestClassifier(n_estimators=100, random_state=42)
modelrf.set_params(min_samples_leaf=15)
modelrf.set_params(max_depth=6)
cv_scores = cross_val_score(modelrf, X_train, y_train, cv=10)
modelrf.fit(X_train,y_train)

from sklearn.metrics import accuracy_score

y_pred = modelrf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Accuracy, Precision, f1 score for RFC
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(accuracy_score(y_test,y_pred ))
print(classification_report(y_test,y_pred))

print("Cross-validation scores:", cv_scores)
# Create Confusion Matrix
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
# Evaluate random forest model
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_auc = roc_auc_score(y_test, rf_predictions)


modellr= LogisticRegression()
modellr.fit(X_train, y_train)
print(modellr.score(X_train,y_train))
print(modellr.score(X_test,y_test))
#Evaluate model

from sklearn.metrics import f1_score,precision_score,recall_score

predictTrain = modellr.predict(X_train)
predictTest = modellr.predict(X_test)

print("F1 Score of Training Set : ",f1_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",f1_score(y_test,predictTest,average=None))
print("F1 Score of Training Set :
",precision_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",precision_score(y_test,predictTest,average=None))
print("F1 Score of Training Set : ",recall_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",recall_score(y_test,predictTest,average=None))

import statsmodels.api as sm

X = encoded_data[['age', 'thalach', 'restecg_Abnormal', 'thal_reversable_defect']]
y = encoded_data['target_Possible CVD']

X = sm.add_constant(X)

logistic_model = sm.Logit(y, X)
result = logistic_model.fit()
summary = result.summary()
p_values = result.pvalues

print(summary)
print(p_values)
```