**Colored Text**

def textColor(color, text):

if color == "red":

print("\033[31m", text, sep="", end="")

elif color == "blue":

print("\033[34m", text, sep="", end="")

elif color == "green":

print("\033[32m", text, sep="", end="")

elif color == "purple":

print("\033[35m", text, sep="", end="")

else:

print("\033[0m", text, sep="", end="")

**Distribution plot**

def distribution\_plot\_wrt\_target(data, predictor, target):

fig, axs = plt.subplots(2, 2, figsize=(12, 10))

target\_uniq = data[target].unique()

axs[0, 0].set\_title("Distribution of target for target=" + str(target\_uniq[0]))

sns.histplot(

data=data[data[target] == target\_uniq[0]],

x=predictor,

kde=True,

ax=axs[0, 0],

color="teal",

stat="density",

)

axs[0, 1].set\_title("Distribution of target for target=" + str(target\_uniq[1]))

sns.histplot(

data=data[data[target] == target\_uniq[1]],

x=predictor,

kde=True,

ax=axs[0, 1],

color="orange",

stat="density",

)

axs[1, 0].set\_title("Boxplot w.r.t target")

sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist\_rainbow")

axs[1, 1].set\_title("Boxplot (without outliers) w.r.t target")

sns.boxplot(

data=data,

x=target,

y=predictor,

ax=axs[1, 1],

showfliers=False,

palette="gist\_rainbow",

)

plt.tight\_layout()

plt.show()

**Histogram Boxplot**

def histogram\_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):

"""

Boxplot and histogram combined

data: dataframe

feature: dataframe column

figsize: size of figure (default (12,7))

kde: whether to show the density curve (default False)

bins: number of bins for histogram (default None)

"""

data[feature].describe()

f2, (ax\_box2, ax\_hist2) = plt.subplots(

nrows=2, # Number of rows of the subplot grid= 2

sharex=True, # x-axis will be shared among all subplots

gridspec\_kw={"height\_ratios": (0.25, 0.75)},

figsize=figsize,

) # creating the 2 subplots

sns.boxplot(

data=data, x=feature, ax=ax\_box2, showmeans=True, color="violet"

) # boxplot will be created and a star will indicate the mean value of the column

sns.histplot(

data=data, x=feature, kde=kde, ax=ax\_hist2, bins=bins, palette="winter"

) if bins else sns.histplot(

data=data, x=feature, kde=kde, ax=ax\_hist2

) # For histogram

ax\_hist2.axvline(

data[feature].mean(), color="green", linestyle="--"

) # Add mean to the histogram

ax\_hist2.axvline(

data[feature].median(), color="black", linestyle="-"

) # Add median to the histogram

**Labeled Barplot**

def labeled\_barplot(data, feature, perc=False, n=None):

"""

Barplot with percentage at the top

data: dataframe

feature: dataframe column

perc: whether to display percentages instead of count (default is False)

n: displays the top n category levels (default is None, i.e., display all levels)

"""

total = len(data[feature]) # length of the column

count = data[feature].nunique()

if n is None:

plt.figure(figsize=(count + 1, 5))

else:

plt.figure(figsize=(n + 1, 5))

plt.xticks(rotation=90, fontsize=15)

ax = sns.countplot(

data=data,

x=feature,

palette="Paired",

order=data[feature].value\_counts().index[:n].sort\_values(),

)

for p in ax.patches:

if perc == True:

label = "{:.1f}%".format(

100 \* p.get\_height() / total

) # percentage of each class of the category

else:

label = p.get\_height() # count of each level of the category

x = p.get\_x() + p.get\_width() / 2 # width of the plot

y = p.get\_height() # height of the plot

ax.annotate(

label,

(x, y),

ha="center",

va="center",

size=12,

xytext=(0, 5),

textcoords="offset points",

) # annotate the percentage

plt.show() # show the plot

**Stacked Barplot**

def stacked\_barplot(data, predictor, target):

"""

Print the category counts and plot a stacked bar chart

data: dataframe

predictor: independent variable

target: target variable

"""

count = data[predictor].nunique()

sorter = data[target].value\_counts().index[-1]

tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort\_values(

by=sorter, ascending=False

)

print(tab1)

print("-" \* 120)

tab = pd.crosstab(data[predictor], data[target], normalize="index").sort\_values(

by=sorter, ascending=False

)

tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))

plt.legend(

loc="lower left", frameon=False,

)

plt.legend(loc="upper left", bbox\_to\_anchor=(1, 1))

plt.show()

**Min Max Scaler**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(df[['Oldpeak']])

df[['Oldpeak']] = scaler.transform(df[['Oldpeak']])

**Label Encoder**

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

df['ST\_Slope'] = label\_encoder.fit\_transform(df['ST\_Slope'])

Pretty Print

def prettyPrint():

os.system("clear")

print("listofEmail")

print()

counter = 1 # add a counter

for email in listOfEmail:

print(f"{counter}: {email}") # make this an f-string

counter += 1 # adds a number with each new email

time.sleep(1)