GUI

June 13, 2023

[1]:

```
# tkinter
import tkinter as tk
from tkinter import *
from time import strftime
from tkinter import ttk
from tkinter.filedialog import askopenfilename
from tkinter.filedialog import asksaveasfilename

# pandas and numpy
import pandas as pd
import numpy as np

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score,classification_report,
confusion_matrix
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error,mean_squared_log_error,median_absolute_error,
explained_variance_score
from tabulate import tabulate
import math
# matplotlib
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import seaborn as sns
sns.set()

from fpdf import FPDF
from tkinter import messagebox
from pandastable import Table


# creating tkinter window
master = Tk()
master.title('Machine Learning Models')
master.geometry('800x600')
master.config(bg="#CCCCCC")
```

```
feature_col = []
target_col = []
# Creating Menubar
menubar = Menu(master)

# Adding File Menu and commands
file = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='File', menu = file)
file.add_command(label ='New File', command = None)
file.add_command(label ='Open...', command = None)
file.add_command(label ='Save', command = None)
file.add_separator()
file.add_command(label ='Exit', command = master.destroy)

# Adding Edit Menu and commands
edit = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Edit', menu = edit)
edit.add_command(label ='Cut', command = None)
edit.add_command(label ='Copy', command = None)
edit.add_command(label ='Paste', command = None)
edit.add_command(label ='Select All', command = None)
edit.add_separator()
edit.add_command(label ='Find...', command = None)
edit.add_command(label ='Find again', command = None)

EDA = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='EDA', menu = EDA)


Clf = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Classification', menu = Clf)


Regreesion = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Regreesion', menu = Regreesion)

Clustering = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Clustering', menu = Clustering)
Clustering.add_command(label ='K-Means', command = None)

# Adding Help Menu
help_ = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Help', menu = help_)
help_.add_command(label ='Tk Help', command = None)
help_.add_command(label ='Demo', command = None)
help_.add_separator()
help_.add_command(label ='About Tk', command = None)


master.config(menu = menubar)
```

1 Classification

2 1) Decision Tree

[2]:

```
def DecisionTree():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Decision Tree Classifier")
    root.config(bg="lavender")

    master.withdraw()

    def data():
        global filename, file

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)

        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models',
        title="Select file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.
                nan,
                strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        colss = file.columns


    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models',
        title="Select file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,
                    strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        feature_col.extend([file.columns[i] for i in box1.curselection() if i
        not in feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in
        target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])


    def deletey():
        for i in reversed(box3.curselection()):
            target_col.remove(box3.get(i))
            box3.delete(i)


    model=None

    def fit():
        global model
        global file

        X = file[feature_col]

        y = file[target_col]


        # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y,
        test_size=float(split_size.get()))


        # Plot of Tree
        from sklearn.tree import DecisionTreeClassifier

        # Define decision tree classifier
        model = DecisionTreeClassifier(criterion=criterion.get(),
        splitter=splitter.get(),max_depth=int(max_d.get()))
        model.fit(X_train,y_train)

        # Access the best estimator and its `tree_` attribute
        tree = model.tree_
        y_pred=model.predict(X_test)
        accuracy=round(accuracy_score(y_pred,y_test),2)*100
        # Print results
        train_accuracy = round(accuracy_score(y_train, model.predict(X_train
        )), 2)
        test_accuracy = round(accuracy_score(y_test, y_pred), 2)

        Label(root, text='PDF has been generated.' , font=('Helvetica', 10
        , 'bold'), bg="light blue",
              relief="solid").place(x=450, y=240)
        accuracy=round(accuracy_score(y_pred,y_test),2)*100
        Label(root, text=f'Accuracy (in %) : {accuracy}', font=('Helvetica', 10
        , 'bold'), bg="light blue",
          relief="solid").place(x=450, y=270)
        Label(root, text=f'Train accuracy : {train_accuracy}',
        font=('Helvetica', 10, 'bold'), bg="light blue",
              relief="solid").place(x=450, y=300)
        Label(root, text=f'Test accuracy  : {test_accuracy}', font=('Helvetica',
        10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=330)


        # Plot of Tree
        from sklearn.tree import plot_tree
        # Create a PDF file with A4 size
        with PdfPages('Decision Tree Classifier.pdf') as pdf:
            pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

            ##Page 1: Descriptive statistics
            numeric_df = X.describe()

            # Create a new page for the descriptive statistics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')

            # Add row labels and round off values to two decimal places
            numeric_df.insert(0, '', numeric_df.index)
            numeric_df = numeric_df.round(2)

            table = ax.table(cellText=numeric_df.
            values, colLabels=numeric_df.columns, cellLoc='center', loc='center')
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)

            # Add a title to the table
            title = ax.set_title('Descriptive Statistics for Numeric Variables
            ', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            # Add the page to the PDF file
            pdf.savefig()

            ##Page 2: Visualise the Dataset
            plt.figure(figsize=(8.27, 5.87))
            plt.title(
            "Heatmap of Correlation Matrix for Numeric Variables",fontsize=16)
            sns.heatmap(X.corr())
            pdf.savefig()


            ##Page 3: Plot the classification report
            cr = classification_report(y_test, y_pred, target_names=target_name,
            output_dict=True)
            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            df = pd.DataFrame(cr).transpose().round(2)
            ax.axis('off')
            table = ax.table(cellText=df.values, colLabels=df.columns,
            cellLoc='center', loc='center', rowLabels=df.index)
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)
            for coord, cell in table.get_celld().items():
                if coord[0] == 0:
                    cell.set_width(0.2)
                else:
                    cell.set_width(0.2)

            # Add a title to the classification report table
            title = ax.set_title('Classification Report', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            pdf.savefig()

            # Plot the confusion matrix
            cm = confusion_matrix(y_test, y_pred, labels=target_name)

            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
            ax.set_title('Confusion Matrix', fontsize=16)
            ax.set_xticklabels(target_name)
            ax.set_yticklabels(target_name)

            pdf.savefig()

            ##Page 5: Plot the decision tree in the page 4
            plt.figure(figsize=(8.27, 11.67))
            plot_tree(model, filled=True, class_names=target_name,feature_names
            =feature_names)

            plt.title("Decision Tree",fontsize=16)

            pdf.savefig()

            # Close the plots
            plt.close('all')



    L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
    L1.place(x=200, y=460)

    def predict():
        x_dummies = s.get().split(",")
        x_tests = []

        for i in x_dummies:
            x_tests.append(float(i))

        global model  # Access the global model variable
        y_pred = model.predict([x_tests])
        L1.config(text=str(y_pred))


    listbox = Listbox(root, selectmode="multiple")
    listbox.pack

    # Create label and entry for split size
    Label(root, font="System", text="Enter the split size:").place(x=20, y=275)
    split_size = tk.StringVar()
    Entry(root, textvariable=split_size).place(x=200, y=275)

    criterion = StringVar()
    choose = ttk.Combobox(root, width=20, textvariable=criterion)
    choose['values'] = ("gini", "entropy")
    choose.place(x=200, y=300)
    Label(root, font="System", text="Choose the criterion:").place(x=20, y=300)

    splitter = StringVar()
    choose = ttk.Combobox(root, width=20, textvariable=splitter)
    choose['values'] = ("best", "random")
    choose.place(x=200, y=330)
    Label(root, font="System", text="Choose the splitter:").place(x=20, y=330)

    #Max depth
    Label(root, font="System", text="Max deapth:").place(x=20, y=360)
    max_d = StringVar()
    Entry(root, textvariable=max_d).place(x=200, y=360)

    # Create label and entry for Feature Variabel Values
    s = StringVar()
    Entry(root,text=s,width=20).place(x=200,y=430)
    Label(root,font="System",text='Feature Variable Values:').place(x=20,y=430)


    l1 = Label(root, text='Select Data File')
    l1.grid(row=0, column=0)


    e1 = Entry(root, text='')
    e1.grid(row=0, column=1)
    Button(root, text='Open',
    command=load_data,activeforeground="white",activebackground="black").
    grid(row=0, column=2)
```

```
    # To switch master window
    #Button(root, text='Close', command=lambda: switch
    _windows(root, master),activeforeground="white",activebackground="black
    ").grid(row=0, column=4)
    Button(root, text='Back', command=lambda: switch_windows(root, master),
    bg='white', fg='red').grid(row=0, column=4)


    box1 = Listbox(root, selectmode='multiple')
    box1.grid(row=11, column=0)


    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',
    command=getx,activeforeground="white",activebackground="black").grid(row=14
    , column=1)
    Button(root, text='Delete X',
    command=deletex,activeforeground="white",activebackground="black").
    grid(row=15, column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',
    command=gety,activeforeground="white",activebackground="black").grid(row=14,
    column=2)
    Button(root, text='Delete Y',
    command=deletey,activeforeground="white",activebackground="black").
    grid(row=15, column=2)

    Button(root, text="RUN MODEL",
    command=fit,activeforeground="white",activebackground="black").place(x=350,
    y=330)
    Button(root, text="PREDICT",
    command=predict,activeforeground="white",activebackground="black").
    place(x=350, y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()
Clf.add_command(label ='Decision Tree Classifier', command = DecisionTree)
```

3 2) K-Nearest Neighbors

[3]:

```
def KNN():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("K-Nearest Neighbors")
    root.config(bg="lavender")
```

```
master.withdraw()

def data():
    global filename, file

    try:
        del file
        e1.delete(0, END)
        box1.delete(0, END)
    except NameError:
        pass

    filename = askopenfilename(initialdir=r'C:\Project\ML Models',
    title="Select file")
    e1.insert(0, filename)
    e1.config(text=filename)

    file = pd.read_csv(filename)

    for i in file.columns:
        box1.insert(END, i)

    for i in file.columns:
        if file[i].dtype == np.float64:
            file[i].fillna(file[i].mean(), inplace=True)
        elif file[i].dtype == np.int64:
            file[i].fillna(file[i].median(), inplace=True)
        elif file[i].dtype == object:
            imp = SimpleImputer(missing_values=np.nan,
            strategy='most_frequent')
            file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

    colss = file.columns


def load_data():
    global filename, file

    try:
        del file
        e1.delete(0, tk.END)
        box1.delete(0, tk.END)
    except NameError:
        pass

    filename = askopenfilename(initialdir=r'C:\Project\ML Models',
    title="Select file")
    e1.insert(0, filename)

    try:
        file = pd.read_csv(filename)
        for i in file.columns:
            box1.insert(tk.END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan,
                strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        # Create a new window to display the table
        top = tk.Toplevel()
        top.title("Data Table")
        f = tk.Frame(top)
        f.pack(fill=tk.BOTH, expand=1)

        # Use pandastable to display the data in a table
        pt = Table(f, dataframe=file)
        pt.show()

    except FileNotFoundError:
        messagebox.showerror("Error", "Please select a file.")

def getx():
    feature_col.extend([file.columns[i] for i in box1.curselection() if i
    not in feature_col])
    box2.insert(END, *feature_col[-len(box1.curselection()):])


def deletex():
    for i in reversed(box2.curselection()):
        feature_col.remove(box2.get(i))
        box2.delete(i)


def gety():
    global target_col
    target_col = [file.columns[i] for i in box1.curselection() if i not in
    target_col]
    box3.insert(END, *target_col[-len(box1.curselection()):])


def deletey():
    for i in reversed(box3.curselection()):
        target_col.remove(box3.get(i))
        box3.delete(i)

model=None
```

```
def fit():
    global model
    global file

    X = file[feature_col]

    y = file[target_col]


    # get target column name
    target_name = pd.unique(file[target_col].values.ravel())
    feature_names=X.columns.tolist()

    # Split data into training and testing sets
    X_train,X_test,y_train,y_test = train_test_split(X,y,
    test_size=float(split_size.get()))



    from sklearn.neighbors import KNeighborsClassifier



    # Create the KNN classifier object
    model = KNeighborsClassifier(n_neighbors=int(neighbor.get()),
    weights=weights.get(), algorithm=algorithm.get())
    model.fit(X_train, np.ravel(y_train))

    # Fit  on the training data
    model.fit(X_train, y_train)

    y_pred=model.predict(X_test)


    # Print results
    train_accuracy = round(accuracy_score(y_train, model.predict(X_train)), 2)
    test_accuracy = round(accuracy_score(y_test, y_pred), 2)


    Label(root, text=f'Train accuracy : {train_accuracy}', font=('Helvetica
    ', 10, 'bold'), bg="light blue",
          relief="solid").place(x=450, y=330)
    Label(root, text=f'Test accuracy  : {test_accuracy}', font=('Helvetica', 10
    , 'bold'), bg="light blue",
            relief="solid").place(x=450, y=300)
    accuracy=round(accuracy_score(y_pred,y_test),2)*100
    Label(root, text=f'Accuracy (in %) : {accuracy}', font=('Helvetica',
    10, 'bold'), bg="light blue",
             relief="solid").place(x=450, y=270)

    # Create a PDF file with A4 size
```

```
        with PdfPages('KNN.pdf') as pdf:
            pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

            ##Page 1: Descriptive statistics
            numeric_df = X.describe()

            # Create a new page for the descriptive statistics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')

            # Add row labels and round off values to two decimal places
            numeric_df.insert(0, '', numeric_df.index)
            numeric_df = numeric_df.round(2)

            table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
↪columns, cellLoc='center', loc='center')
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)

            # Add a title to the table
            title = ax.set_title('Descriptive Statistics ', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            # Add the page to the PDF file
            pdf.savefig()

            ##Page 2: Visualise the Dataset
            plt.figure(figsize=(8.27, 5.87))
            plt.title("Heatmap of Correlation Matrix ",fontsize=16)
            sns.heatmap(file.corr(numeric_only=True))
            pdf.savefig()


            ##Page 3: Plot the classification report
            cr = classification_report(y_test, y_pred, target_names=None,␣
↪output_dict=True)
            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            df = pd.DataFrame(cr).transpose().round(2)
            ax.axis('off')
            table = ax.table(cellText=df.values, colLabels=df.columns,␣
↪cellLoc='center', loc='center', rowLabels=df.index)
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)
            for coord, cell in table.get_celld().items():
                if coord[0] == 0:
                    cell.set_width(0.2)
                else:
                    cell.set_width(0.2)
            # Add a title to the classification report table
            title = ax.set_title('Classification Report', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)
            pdf.savefig()


            ##Page 4: # Plot the confusion matrix
            cm = confusion_matrix(y_test, y_pred, labels=target_name)

            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
            ax.set_title('Confusion Matrix', fontsize=16)
            ax.set_xticklabels(target_name)
            ax.set_yticklabels(target_name)

            pdf.savefig()

                # Close the plots
            plt.close('all')

    L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
    L1.place(x=200, y=460)

    def predict():
        x_dummies = s.get().split(",")
        x_tests = []

        for i in x_dummies:
            x_tests.append(float(i))

        global model  # Access the global model variable
        y_pred = model.predict([x_tests])
        L1.config(text=str(y_pred))

    listbox = Listbox(root, selectmode="multiple")
    listbox.pack

    # Create label and entry for split size
    Label(root, font="System", text="Enter the split size:").place(x=20, y=275)
    split_size = tk.StringVar()
    Entry(root, textvariable=split_size).place(x=200, y=275)

    # Create label and entry for the number of neighbors
    Label(root, font="System", text="The number of neighbors").place(x=20, y=300)
    neighbor = tk.StringVar()
    Entry(root, textvariable=neighbor).place(x=200, y=300)

    weights = tk.StringVar()
    choose = ttk.Combobox(root, width=20, textvariable=weights)
    choose['values'] = ('uniform', 'distance')
    choose.place(x=200, y=325)
```

```
    Label(root, font="System", text="Choose the weight type").place(x=20, y=325)

    algorithm = tk.StringVar()
    choose = ttk.Combobox(root, width=20, textvariable=algorithm)
    choose['values'] = ('auto', 'ball_tree', 'kd_tree', 'brute')
    choose.place(x=200, y=350)
    Label(root, font="System", text="Choose algorithm type").place(x=20, y=350)

    # Create label and entry for Feature Variabel Values
    s = StringVar()
    Entry(root,text=s,width=20).place(x=200,y=430)
    Label(root,font="System",text='Feature Variable Values:').place(x=20,y=430)


    l1 = Label(root, text='Select Data File')
    l1.grid(row=0, column=0)


    e1 = Entry(root, text='')
    e1.grid(row=0, column=1)
    Button(root,␣
↪text='Open',command=load_data,activeforeground="white",activebackground="black").
↪grid(row=0, column=2)

    # To switch master window
    #Button(root, text='Close', command=lambda: switch_windows(root,␣
↪master),activeforeground="white",activebackground="black").grid(row=0, column=4)
    Button(root, text='Back', command=lambda: switch_windows(root, master),␣
↪bg='white', fg='red').grid(row=0, column=4)


    box1 = Listbox(root, selectmode='multiple')
    box1.grid(row=11, column=0)


    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',␣
↪command=getx,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=1)
    Button(root, text='Delete X',␣
↪command=deletex,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',␣
↪command=gety,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=2)
```

```
    Button(root, text='Delete Y',␣
 ↪command=deletey,activeforeground="white",activebackground="black").grid(row=15,␣
 ↪column=2)

    Button(root, text="RUN MODEL",␣
 ↪command=fit,activeforeground="white",activebackground="black").place(x=350, y=330)
    Button(root, text="PREDICT",␣
 ↪command=predict,activeforeground="white",activebackground="black").place(x=350,␣
 ↪y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Clf.add_command(label ='K-Nearest Neighbors', command = KNN)
```

4 3) Support Vector classifier

[4]:
```
def SVC():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Support Vector Classifier")
    root.config(bg="lavender")

    master.withdraw()

    def data():
        global filename, file

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
 ↪file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        colss = file.columns


    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,␣
↪strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in␣
↪feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in␣
↪target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])


    def deletey():
        for i in reversed(box3.curselection()):
            target_col.remove(box3.get(i))
            box3.delete(i)

    model=None

    def fit():
        global model,file, target_names, feature_names


        X = file[feature_col]

        y = file[target_col]


        # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        #feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y,␣
↪test_size=float(split_size.get()))


        # # training model
        from sklearn.svm import SVC

        # Define decision SVM classifier
        model=SVC(C=float(Penalty.get()),kernel=Kernal.
↪get(),decision_function_shape=dca.get())
        model.fit(X_train,np.ravel(y_train))

        # Access the best estimator and its `tree_` attribute
        y_pred=model.predict(X_test)

        # Print results
        train_accuracy = round(accuracy_score(y_train, model.predict(X_train)), 2)
        test_accuracy = round(accuracy_score(y_test, y_pred), 2)

        Label(root, text='PDF has been generated.' , font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
              relief="solid").place(x=450, y=240)
        accuracy=round(accuracy_score(y_pred,y_test),2)*100
        Label(root, text=f'Accuracy (in %) : {accuracy}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
          relief="solid").place(x=450, y=270)
        Label(root, text=f'Train accuracy : {train_accuracy}', font=('Helvetica',␣
↪10, 'bold'), bg="light blue",
              relief="solid").place(x=450, y=300)
        Label(root, text=f'Test accuracy  : {test_accuracy}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=330)



        # Create a PDF file with A4 size
        with PdfPages('Support Vector Classifier.pdf') as pdf:
            pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

            ##Page 1: Descriptive statistics
            numeric_df = X.describe()

            # Create a new page for the descriptive statistics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')

            # Add row labels and round off values to two decimal places
            numeric_df.insert(0, '', numeric_df.index)
            numeric_df = numeric_df.round(2)

            table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
↪columns, cellLoc='center', loc='center')
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)

            # Add a title to the table
            title = ax.set_title('Descriptive Statistics for Numeric Variables',␣
↪fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            # Add the page to the PDF file
            pdf.savefig()
            ##Page 3: Plot the classification report
            cr = classification_report(y_test, y_pred, target_names=target_name,␣
↪output_dict=True)
            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            df = pd.DataFrame(cr).transpose().round(2)
            ax.axis('off')
            table = ax.table(cellText=df.values, colLabels=df.columns,␣
↪cellLoc='center', loc='center', rowLabels=df.index)
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)
            for coord, cell in table.get_celld().items():
                if coord[0] == 0:
                    cell.set_width(0.2)
                else:
                    cell.set_width(0.2)

            # Add a title to the classification report table
            title = ax.set_title('Classification Report', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            pdf.savefig()

            # Plot the confusion matrix
            cm = confusion_matrix(y_test, y_pred, labels=target_name)

            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
            ax.set_title('Confusion Matrix', fontsize=16)
            ax.set_xticklabels(target_name)
            ax.set_yticklabels(target_name)

            pdf.savefig()

            # Create a pandas DataFrame from the feature matrix
            df = pd.DataFrame(X, columns=feature_names)
            df['target'] = y

            # Create a scatter matrix (pair plot) with different colors for each␣
↪class
            sns.pairplot(df, vars=feature_names, hue='target', diag_kind='hist')  #␣
↪Exclude the target column from pair plot

            # Add the page to the PDF file
            pdf.savefig()
            plt.close()

            # Close the plots
            plt.close('all')
```

```python
L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
L1.place(x=200, y=460)

def predict():
    x_dummies = s.get().split(",")
    x_tests = []

    for i in x_dummies:
        x_tests.append(float(i))

    global model  # Access the global model variable
    y_pred = model.predict([x_tests])
    L1.config(text=str(y_pred))


listbox = Listbox(root, selectmode="multiple")
listbox.pack

# Create label and entry for split size
Label(root, font="System", text="Enter the split size:").place(x=20, y=275)
split_size = tk.StringVar()
Entry(root, textvariable=split_size).place(x=200, y=275)


Penalty=tk.StringVar()
choose=ttk.Combobox(root,width = 20, textvariable= Penalty)
choose['values']=('1','2','3','4','5')
choose.place(x=200,y=300)
Label(root, font="System", text="Choose Penlaty:").place(x=20, y=300)

Kernal=tk.StringVar()
choose=ttk.Combobox(root,width=20,textvariable= Kernal)
choose['values']=('linear','poly','rbf','sigmoid','precomputed')
choose.place(x=200, y=330)
Label(root, font="System", text="Choose the Kernel:").place(x=20, y=330)


dca =tk.StringVar()
choose=ttk.Combobox(root,width=20,textvariable= dca)
choose['values']=('ovo','ovr')
choose.place(x=200, y=360)
Label(root, font="System", text="Choose the dca:").place(x=20, y=360)

# Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=20).place(x=200,y=430)
Label(root,font="System",text='Feature Variable Values:').place(x=20,y=430)


l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)
```

```
    e1 = Entry(root, text='')
    e1.grid(row=0, column=1)
    Button(root,␣
↪text='Open',command=load_data,activeforeground="white",activebackground="black").
↪grid(row=0, column=2)

    # To switch master window
    Button(root, text='Back', command=lambda: switch_windows(root, master),␣
↪bg='white', fg='red').grid(row=0, column=4)


    box1 = Listbox(root, selectmode='multiple')
    box1.grid(row=11, column=0)


    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',␣
↪command=getx,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=1)
    Button(root, text='Delete X',␣
↪command=deletex,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',␣
↪command=gety,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=2)
    Button(root, text='Delete Y',␣
↪command=deletey,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=2)

    Button(root, text="RUN MODEL",␣
↪command=fit,activeforeground="white",activebackground="black").place(x=350, y=330)
    Button(root, text="PREDICT",␣
↪command=predict,activeforeground="white",activebackground="black").place(x=350,␣
↪y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Clf.add_command(label ='Support Vector Classifier', command = SVC)
```

5 4) Naive Bayes

[5]:

```
def NB():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Naive Bayes Classifier")
    root.config(bg="lavender")

    master.withdraw()

    def data():
        global filename, file

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
 ↪file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        colss = file.columns


    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass
        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,␣
↪strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in␣
↪feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in␣
↪target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])
```

```python
def deletey():
    for i in reversed(box3.curselection()):
        target_col.remove(box3.get(i))
        box3.delete(i)

model=None

def fit():
    global model
    global file

    X = file[feature_col]

    y = file[target_col]


    # get target column name
    target_name = pd.unique(file[target_col].values.ravel())
    feature_names=X.columns.tolist()

    # Split data into training and testing sets
    X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=float(split_size.get()))


    # Training Model
    from sklearn.naive_bayes import MultinomialNB


    # Define decision tree classifier
    model=MultinomialNB(alpha=float(alph.get()),fit_prior=bool(fit_p.get()))
    model.fit(X_train,np.ravel(y_train))

    y_pred=model.predict(X_test)

    # Print results
    train_accuracy = round(accuracy_score(y_train, model.predict(X_train)), 2)
    test_accuracy = round(accuracy_score(y_test, y_pred), 2)

    Label(root, text='PDF has been generated.' , font=('Helvetica', 10, 'bold'), bg="light blue",
          relief="solid").place(x=450, y=240)
    accuracy=round(accuracy_score(y_pred,y_test),2)*100
    Label(root, text=f'Accuracy (in %) : {accuracy}', font=('Helvetica', 10, 'bold'), bg="light blue",
          relief="solid").place(x=450, y=270)
    Label(root, text=f'Train accuracy : {train_accuracy}', font=('Helvetica', 10, 'bold'), bg="light blue",
          relief="solid").place(x=450, y=300)
    Label(root, text=f'Test accuracy  : {test_accuracy}', font=('Helvetica', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=330)


        # Training Model
        from sklearn.tree import plot_tree
        # Create a PDF file with A4 size
        with PdfPages('Naive Bayes Classifier.pdf') as pdf:
            pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

            ##Page 1: Plot the classification report
            cr = classification_report(y_test, y_pred, target_names=target_name,␣
↪output_dict=True)
            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            df = pd.DataFrame(cr).transpose().round(2)
            ax.axis('off')
            table = ax.table(cellText=df.values, colLabels=df.columns,␣
↪cellLoc='center', loc='center', rowLabels=df.index)
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)
            for coord, cell in table.get_celld().items():
                if coord[0] == 0:
                    cell.set_width(0.2)
                else:
                    cell.set_width(0.2)

            # Add a title to the classification report table
            title = ax.set_title('Classification Report', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            pdf.savefig()


            #Page 2: Plot the confusion matrix
            cm = confusion_matrix(y_test, y_pred, labels=target_name)

            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
            ax.set_title('Confusion Matrix', fontsize=16)
            ax.set_xticklabels(target_name)
            ax.set_yticklabels(target_name)

            pdf.savefig()
            # Close the plots
            plt.close('all')



    L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
    L1.place(x=200, y=460)
```

```
def predict():
    x_dummies = s.get().split(",")
    x_tests = []

    for i in x_dummies:
        x_tests.append(float(i))

    global model  # Access the global model variable
    y_pred = model.predict([x_tests])
    L1.config(text=str(y_pred))


listbox = Listbox(root, selectmode="multiple")
listbox.pack

# Create label and entry for split size
Label(root, font="System", text="Enter the split size:").place(x=20, y=275)
split_size = tk.StringVar()
Entry(root, textvariable=split_size).place(x=200, y=275)


alph=tk.StringVar()
choose=ttk.Combobox(root,width=20,textvariable=alph)
choose['values']=('1','2','3','4')
choose.place(x=200, y=300)
Label(root, font="System", text="Choose alpha:").place(x=20, y=300)

fit_p=tk.StringVar()
choose=ttk.Combobox(root,width=20,textvariable=fit_p)
choose['values']=('True','False')
choose.place(x=200, y=330)
Label(root, font="System", text=" Fit_p:").place(x=20, y=330)

# Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=20).place(x=200,y=430)
Label(root,font="System",text='Feature Variable Values:').place(x=20,y=430)


l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)



e1 = Entry(root, text='')
e1.grid(row=0, column=1)
Button(root,␣
↪text='Open',command=load_data,activeforeground="white",activebackground="black").
↪grid(row=0, column=2)

# To switch master window
#Button(root, text='Close', command=lambda: switch_windows(root,␣
↪master),activeforeground="white",activebackground="black").grid(row=0, column=4)
```

```
    Button(root, text='Back', command=lambda: switch_windows(root, master),␣
↪bg='white', fg='red').grid(row=0, column=4)


    box1 = Listbox(root, selectmode='multiple')
    box1.grid(row=11, column=0)


    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',␣
↪command=getx,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=1)
    Button(root, text='Delete X',␣
↪command=deletex,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',␣
↪command=gety,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=2)
    Button(root, text='Delete Y',␣
↪command=deletey,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=2)

    Button(root, text="RUN MODEL",␣
↪command=fit,activeforeground="white",activebackground="black").place(x=350, y=330)
    Button(root, text="PREDICT",␣
↪command=predict,activeforeground="white",activebackground="black").place(x=350,␣
↪y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Clf.add_command(label ='Naive Bayes', command = NB)
```

6 Regression

7 1) Decision Tree Regression

[6]:

```
def DecisionTreeRegression():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Decision Tree Regression")
    root.config(bg="lavender")

    master.withdraw()
    def data():
        global filename, file

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        colss = file.columns


    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)
            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,␣
    ↪strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in␣
    ↪feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in␣
    ↪target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])


    def deletey():
        for i in reversed(box3.curselection()):
            target_col.remove(box3.get(i))
            box3.delete(i)


    model=None

    def fit():
        global model
        global file

        X = file[feature_col]

        y = file[target_col]

         # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y,␣
↪test_size=float(split_size.get()))

        from sklearn.tree import DecisionTreeRegressor

        # Define decision tree classifier
        model = DecisionTreeRegressor(criterion=criterion.get(), splitter=splitter.
↪get(),max_depth=int(max_d.get()))

        model.fit(X_train,y_train)

        y_pred=model.predict(X_test)


        MSE=mean_squared_error(y_pred,y_test).round(2)
        r2=r2_score(y_pred,y_test).round(2)
        EV=explained_variance_score(y_pred,y_test).round(2)
        MAE=mean_absolute_error(y_pred,y_test).round(2)
        MSLE=mean_squared_log_error(y_pred,y_test).round(2)
        MeAE=median_absolute_error(y_pred,y_test).round(2)

        Label(root, text='PDF has been generated.' , font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
              relief="solid").place(x=450, y=240)
        Label(root, text=f'Mean Squared Error : {MSE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
              relief="solid").place(x=450, y=270)
        Label(root, text=f'R Square  : {r2}', font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
                relief="solid").place(x=450, y=300)
        Label(root, text=f'Explained Variance : {EV}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=330)
        Label(root, text=f'Mean Absolute Error  : {MAE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=370)
        Label(root, text=f'Mean Squared Log Error : {MSLE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=400)
```

```
       Label(root, text=f'Median Absolute Error : {MeAE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
               relief="solid").place(x=450, y=430)

       # Plot of Tree
       from sklearn.tree import plot_tree

       # Define your performance metrics
      # y_pred = model.predict(X_test)
       rmse = round(math.sqrt(MSE), 2)

       # Create a PDF file with A4 size
       with PdfPages('Decision Tree Regressin.pdf') as pdf:
           pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

           ##Page 1: Descriptive statistics
           numeric_df = file.describe()

           # Create a new page for the descriptive statistics
           fig, ax = plt.subplots(figsize=(8.27, 11.69))
           ax.axis('off')

           # Add row labels and round off values to two decimal places
           numeric_df.insert(0, '', numeric_df.index)
           numeric_df = numeric_df.round(2)

           table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
↪columns, cellLoc='center', loc='center')
           table.auto_set_font_size(False)
           table.set_fontsize(10)
           table.scale(1, 1.5)

           # Add a title to the table
           title = ax.set_title('Descriptive Statistics for Numeric Variables',␣
↪fontsize=16)
           title.set_y(0.95)
           fig.subplots_adjust(top=0.85)

           # Add the page to the PDF file
           pdf.savefig()

           # Page 2: Performance metrics
           fig, ax = plt.subplots(figsize=(8.27, 11.69))
           ax.axis('off')
           metrics_data = [['R-squared', r2], ['Mean Absolute Error', MAE], ['Mean␣
↪Squared Error', MSE], ['Root Mean Squared Error', rmse]]
           metrics_df = pd.DataFrame(metrics_data, columns=['Metric', 'Value'])
           metrics_table = ax.table(cellText=metrics_df.values,␣
↪colLabels=metrics_df.columns, cellLoc='center', loc='center')
           metrics_table.auto_set_font_size(False)
           metrics_table.set_fontsize(10)
           metrics_table.scale(1, 1.5)
```

```
        title = ax.set_title('Performance Metrics', fontsize=16)
        title.set_y(0.95)
        fig.subplots_adjust(top=0.85)
        pdf.savefig()


        ##Page 3: Plot the decision tree in the page 4
        plt.figure(figsize=(8.27, 11.67))
        plot_tree(model, filled=True,feature_names =feature_names)
        plt.title("Decision Tree",fontsize=16)

        pdf.savefig()


L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
L1.place(x=200, y=460)

def predict():
    x_dummies = s.get().split(",")
    x_tests = []

    for i in x_dummies:
        x_tests.append(float(i))

    global model  # Access the global model variable
    y_pred = model.predict([x_tests])
    L1.config(text=str(y_pred))


listbox = Listbox(root, selectmode="multiple")
listbox.pack

# Create label and entry for split size
Label(root, font="System", text="Enter the split size:").place(x=20, y=275)
split_size = tk.StringVar()
Entry(root, textvariable=split_size).place(x=200, y=275)

criterion = StringVar()
choose = ttk.Combobox(root, width=20, textvariable=criterion)
choose['values'] = ('absolute_error', 'squared_error', 'poisson', 'friedman_mse')
choose.place(x=200, y=300)
Label(root, font="System", text="Choose the criterion:").place(x=20, y=300)

splitter = StringVar()
choose = ttk.Combobox(root, width=20, textvariable=splitter)
choose['values'] = ("best", "random")
choose.place(x=200, y=330)
Label(root, font="System", text="Choose the splitter:").place(x=20, y=330)

#Max depth
Label(root, font="System", text="Max deapth:").place(x=20, y=360)
max_d = StringVar()
```

```
    Entry(root, textvariable=max_d).place(x=200, y=360)

    # Create label and entry for Feature Variabel Values
    s = StringVar()
    Entry(root,text=s,width=20).place(x=200,y=430)
    Label(root,font="System",text='Feature Variable Values:').place(x=20,y=430)


    l1 = Label(root, text='Select Data File')
    l1.grid(row=0, column=0)


    e1 = Entry(root, text='')
    e1.grid(row=0, column=1)
    Button(root,␣
↪text='Open',command=load_data,activeforeground="white",activebackground="black").
↪grid(row=0, column=2)

    # To switch master window
    #Button(root, text='Close', command=lambda: switch_windows(root,␣
↪master),activeforeground="white",activebackground="black").grid(row=0, column=4)
    Button(root, text='Back', command=lambda: switch_windows(root, master),␣
↪bg='white', fg='red').grid(row=0, column=4)


    box1 = Listbox(root, selectmode='multiple')
    box1.grid(row=11, column=0)


    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',␣
↪command=getx,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=1)
    Button(root, text='Delete X',␣
↪command=deletex,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',␣
↪command=gety,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=2)
    Button(root, text='Delete Y',␣
↪command=deletey,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=2)

    Button(root, text="RUN MODEL",␣
↪command=fit,activeforeground="white",activebackground="black").place(x=350, y=330)
```

```
    Button(root, text="PREDICT",␣
↪command=predict,activeforeground="white",activebackground="black").place(x=350,␣
↪y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()
Regreesion.add_command(label ='Decision Tree Regressor', command =␣
↪DecisionTreeRegression)
```

8 2) Logistic Regression

[7]:

```
def LogisticRegression():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Logistic Regression")
    root.config(bg="lavender")

    master.withdraw()

    def data():
        global filename, file

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        colss = file.columns
    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,␣
↪strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in␣
↪feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in␣
↪target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])


    def deletey():
        for i in reversed(box3.curselection()):
            target_col.remove(box3.get(i))
            box3.delete(i)


    model=None

    def fit():
        global model
        global file

        X = file[feature_col]

        y = file[target_col]

        # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y,␣
↪test_size=float(split_size.get()))


        from sklearn.linear_model import LogisticRegression

        # Define decision tree classifier
        model= LogisticRegression(C=int(C.get()),max_iter=int(max_iter.get()))
        model.fit(X_train,y_train)

        y_pred=model.predict(X_test)


        # Print results
        train_accuracy = round(accuracy_score(y_train, model.predict(X_train)), 2)
        test_accuracy = round(accuracy_score(y_test, y_pred), 2)

        Label(root, text='PDF has been generated.' , font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
              relief="solid").place(x=450, y=240)
        accuracy=round(accuracy_score(y_pred,y_test),2)*100
        Label(root, text=f'Accuracy (in %) : {accuracy}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
          relief="solid").place(x=450, y=270)
        Label(root, text=f'Train accuracy : {train_accuracy}', font=('Helvetica',␣
↪10, 'bold'), bg="light blue",
              relief="solid").place(x=450, y=300)
        Label(root, text=f'Test accuracy  : {test_accuracy}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=330)

        # Create a PDF file with A4 size
        with PdfPages('Logistic Regression.pdf') as pdf:
            pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

            ##Page 1: Descriptive statistics
            numeric_df = X.describe()

            # Create a new page for the descriptive statistics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')

            # Add row labels and round off values to two decimal places
            numeric_df.insert(0, '', numeric_df.index)
            numeric_df = numeric_df.round(2)

            table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
↪columns, cellLoc='center', loc='center')
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)

            # Add a title to the table
            title = ax.set_title('Descriptive Statistics for Numeric Variables',␣
↪fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            # Add the page to the PDF file
            pdf.savefig()

            ##Page 2: Visualise the Dataset
            plt.figure(figsize=(8.27, 5.87))
            plt.title("Heatmap of Correlation Matrix for Numeric␣
↪Variables",fontsize=16)
            plt.subplots(figsize=(8.27,5.8))
            sns.heatmap(X.corr(numeric_only=True))
            pdf.savefig()
            ##Page 3: Plot the classification report
            cr = classification_report(y_test, y_pred, target_names=None,␣
↪output_dict=True)
            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            df = pd.DataFrame(cr).transpose().round(2)
            ax.axis('off')
            table = ax.table(cellText=df.values, colLabels=df.columns,␣
↪cellLoc='center', loc='center', rowLabels=df.index)
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)
            for coord, cell in table.get_celld().items():
                if coord[0] == 0:
                    cell.set_width(0.2)
                else:
                    cell.set_width(0.2)

            # Add a title to the classification report table
            title = ax.set_title('Classification Report', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            pdf.savefig()


            ##Page 4: Plot the confusion matrix
            cm = confusion_matrix(y_test, y_pred)

            fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8.27, 5.87))
            sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
            ax.set_title('Confusion Matrix', fontsize=16)
            ax.set_xlabel('Predicted Label')
            ax.set_ylabel('True Label')

            pdf.savefig()

            # Close the plots
            plt.close('all')

    L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
    L1.place(x=200, y=460)

    def predict():
        x_dummies = s.get().split(",")
        x_tests = []

        for i in x_dummies:
            x_tests.append(float(i))

        global model  # Access the global model variable
        y_pred = model.predict([x_tests])
        L1.config(text=str(y_pred))




    listbox = Listbox(root, selectmode="multiple")
    listbox.pack


    # Create label and entry for split size
    Label(root, font="System", text="Enter the split size:").place(x=20, y=275)
    split_size = tk.StringVar()
    Entry(root, textvariable=split_size).place(x=200, y=275)

    #Max iter
    Label(root, font="System", text="Max iter:").place(x=20, y=300)
    max_iter = StringVar()
    Entry(root, textvariable=max_iter).place(x=200, y=300)

    #choose C
    Label(root, font="System", text="C:").place(x=20, y=330)
    C = StringVar()
    Entry(root, textvariable=C).place(x=200, y=330)



    # Create label and entry for Feature Variabel Values
    s = StringVar()
    Entry(root,text=s,width=20).place(x=200,y=430)
    Label(root,font="System",text='Feature Variable Values:').place(x=20,y=430)


    l1 = Label(root, text='Select Data File')
    l1.grid(row=0, column=0)


    e1 = Entry(root, text='')
    e1.grid(row=0, column=1)
    Button(root,␣
↪text='Open',command=load_data,activeforeground="white",activebackground="black").
↪grid(row=0, column=2)

    # To switch master window
    #Button(root, text='Close', command=lambda: switch_windows(root,␣
↪master),activeforeground="white",activebackground="black").grid(row=0, column=4)
    Button(root, text='Back', command=lambda: switch_windows(root, master),␣
↪bg='white', fg='red').grid(row=0, column=4)


    box1 = Listbox(root, selectmode='multiple')
    box1.grid(row=11, column=0)
    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',␣
 ↪command=getx,activeforeground="white",activebackground="black").grid(row=14,␣
 ↪column=1)
    Button(root, text='Delete X',␣
 ↪command=deletex,activeforeground="white",activebackground="black").grid(row=15,␣
 ↪column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',␣
 ↪command=gety,activeforeground="white",activebackground="black").grid(row=14,␣
 ↪column=2)
    Button(root, text='Delete Y',␣
 ↪command=deletey,activeforeground="white",activebackground="black").grid(row=15,␣
 ↪column=2)

    Button(root, text="RUN MODEL",␣
 ↪command=fit,activeforeground="white",activebackground="black").place(x=350, y=330)
    Button(root, text="PREDICT",␣
 ↪command=predict,activeforeground="white",activebackground="black").place(x=350,␣
 ↪y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()
Clf.add_command(label ='Logistic Regression', command = LogisticRegression)
```

9 3) Linear Regression

[8]:

```
def LinearRegression():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Linear Regression")
    root.config(bg="lavender")
    master.withdraw()

    def data():
        global filename, file,feature_names,target_names

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass
        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        feature_names = list(file.columns)
        target_names = []
        for i in file.columns:
            if file[i].dtype == object:
                target_names.append(i)
                class_names.extend(list(file[i].unique()))

    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,␣
↪strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))



            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        global feature_col
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in␣
↪feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])

        # Update feature_names
        global feature_names
        feature_names = list(file[feature_col].columns)


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)

        # Update feature_names
        global feature_names
        feature_names = list(file[feature_col].columns)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in␣
↪target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])

        # Update target_names
        global target_names
        target_names = list(file[target_col].columns)
        global class_names
        class_names = list(set(file[target_col]))
        #list(set(data['target_column']))
```

```python
def deletey():
    for i in reversed(box3.curselection()):
        target_col.remove(box3.get(i))
        box3.delete(i)

    # Update target_names
    global target_names
    target_names = list(file[target_col].columns)
    global class_names
    class_names = list(set(file[target_col]))

model=None

def fit():
    global model
    global file

    X = file[feature_col]

    y = file[target_col]

    # get target column name
    target_name = pd.unique(file[target_col].values.ravel())
    feature_names=X.columns.tolist()

    # Split data into training and testing sets
    X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=float(split_size.get()))


    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    X_train=scaler.fit_transform(X_train)
    X_test=scaler.transform(X_test)

    from sklearn.linear_model import LinearRegression
    ##cross validation
    from sklearn.model_selection import cross_val_score

    model=LinearRegression()
    model.fit(X_train,y_train)


    model_pred=model.predict(X_test)
    MSE=mean_squared_error(model_pred,y_test)
    r2=r2_score(model_pred,y_test)
    EV=explained_variance_score(model_pred,y_test)
    MAE=mean_absolute_error(model_pred,y_test)
    MSLE=mean_squared_log_error(model_pred,y_test)
    MeAE=median_absolute_error(model_pred,y_test)
    intercept=model.intercept_
```

```
        coefficients=model.coef_

        Label(root, text='PDF has been generated.' , font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
              relief="solid").place(x=450, y=240)
        Label(root, text=f'Intercept : {intercept}', font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
              relief="solid").place(x=450, y=270)
        Label(root, text=f'Coefficients  : {coefficients}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=300)
        Label(root, text=f'Mean Squared Error : {MSE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
              relief="solid").place(x=450, y=330)
        Label(root, text=f'R Square  : {r2}', font=('Helvetica', 10, 'bold'),␣
↪bg="light blue",
                relief="solid").place(x=450, y=360)
        Label(root, text=f'Explained Variance : {EV}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=390)
        Label(root, text=f'Mean Absolute Error  : {MAE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=420)
        Label(root, text=f'Mean Squared Log Error : {MSLE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=450)
        Label(root, text=f'Median Absolute Error : {MeAE}', font=('Helvetica', 10,␣
↪'bold'), bg="light blue",
                relief="solid").place(x=450, y=480)



        # Create a PDF file with A4 size
        with PdfPages('Linear Regression.pdf') as pdf:
            pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

            ##Page 1: Descriptive statistics
            numeric_df = file.describe()

            # Create a new page for the descriptive statistics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')

            # Add row labels and round off values to two decimal places
            numeric_df.insert(0, '', numeric_df.index)
            numeric_df = numeric_df.round(2)

            table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
↪columns, cellLoc='center', loc='center')
            table.auto_set_font_size(False)
            table.set_fontsize(10)
            table.scale(1, 1.5)

            # Add a title to the table
            title = ax.set_title('Descriptive Statistics for Numeric Variables',␣
↪fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            # Add the page to the PDF file
            pdf.savefig()




            # Page 2: Performance metrics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')
            metrics_data = [['Intercept', intercept], ['Coefficient',␣
↪coefficients],['R Square', r2], ['Mean Squared Error', MSE],
                            ['Explained Variance', EV],['Mean Squared Log Error',␣
↪MSLE],['Mean Absolute Error', MAE],['Median Absolute Error', MeAE]]
            metrics_df = pd.DataFrame(metrics_data, columns=['Metric', 'Value'])
            metrics_table = ax.table(cellText=metrics_df.values,␣
↪colLabels=metrics_df.columns, cellLoc='center', loc='center')
            metrics_table.auto_set_font_size(False)
            metrics_table.set_fontsize(10)
            metrics_table.scale(1, 1.5)
            title = ax.set_title('Performance Metrics', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)
            pdf.savefig()

            ##Page 3: Visualise the Dataset
            plt.figure(figsize=(8.27, 5.87))
            plt.title("Heatmap of Correlation Matrix for Numeric␣
↪Variables",fontsize=16)
            sns.heatmap(file.corr())
            pdf.savefig()

            ##Page 4:
            plt.figure(figsize=(8.27, 5.87))
            plt.title("Distribution plot of Difference between Actual and Predicted␣
↪Responce",fontsize=16)
            plt.subplots(figsize=(8.27,5.8))
            sns.displot(model_pred-y_test,kind='kde')
            pdf.savefig()



            # Close the plots
            plt.close('all')
```

```
L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
L1.place(x=200, y=430)

def predict():
    x_dummies = s.get().split(",")
    x_tests = []

    for i in x_dummies:
        x_tests.append(float(i))

    global model  # Access the global model variable
    y_pred = model.predict([x_tests])
    L1.config(text=str(y_pred))

listbox = Listbox(root, selectmode="multiple")
listbox.pack

# Create label and entry for split size
Label(root, font="System", text="Enter the split size:").place(x=20, y=270)
split_size = tk.StringVar()
Entry(root, textvariable=split_size).place(x=200, y=270)



 # Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=30).place(x=200,y=360)
Label(root,font="System",text='Feature Variable Values').place(x=20, y=360)
l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)


e1 = Entry(root, text='')
e1.grid(row=0, column=1)
Button(root,␣
↪text='Open',command=load_data,activeforeground="white",activebackground="black").
↪grid(row=0, column=2)

# To switch master window
Button(root, text='Back', command=lambda: switch_windows(root,␣
↪master),activeforeground="white",activebackground="black").grid(row=0, column=4)


box1 = Listbox(root, selectmode='multiple')
box1.grid(row=11, column=0)


Label(root, text='Features').grid(row=10, column=1)
box2 = Listbox(root,selectmode='multiple')
box2.grid(row=11, column=1)
```

```
    Button(root, text='Select X',␣
↪command=getx,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=1)
    Button(root, text='Delete X',␣
↪command=deletex,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',␣
↪command=gety,activeforeground="white",activebackground="black").grid(row=14,␣
↪column=2)
    Button(root, text='Delete Y',␣
↪command=deletey,activeforeground="white",activebackground="black").grid(row=15,␣
↪column=2)

    Button(root, text="RUN MODEL",␣
↪command=fit,activeforeground="white",activebackground="black").place(x=350, y=270)
    Button(root, text="PREDICT",␣
↪command=predict,activeforeground="white",activebackground="black").place(x=350,␣
↪y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Regreesion.add_command(label ='Linear Regression', command = LinearRegression)
```

10 4)Polynomial Regression

[9]:

```
def PolynomialRegression():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Polynomial Regression")
    root.config(bg="lavender")
    master.withdraw()

    def data():
        global filename, file,feature_names,target_names

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        feature_names = list(file.columns)
        target_names = []
        for i in file.columns:
            if file[i].dtype == object:
                target_names.append(i)
                class_names.extend(list(file[i].unique()))

    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title="Select␣
↪file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,␣
↪strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))
            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        global feature_col
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in␣
↪feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])

        # Update feature_names
        global feature_names
        feature_names = list(file[feature_col].columns)


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)

        # Update feature_names
        global feature_names
        feature_names = list(file[feature_col].columns)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in
        target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])

        # Update target_names
        global target_names
        target_names = list(file[target_col].columns)
        global class_names
        class_names = list(set(file[target_col]))
        #list(set(data['target_column']))

    def deletey():
        for i in reversed(box3.curselection()):
            target_col.remove(box3.get(i))
            box3.delete(i)

        # Update target_names
        global target_names
        target_names = list(file[target_col].columns)
        global class_names
        class_names = list(set(file[target_col]))

    model=None

    def fit():
        global model
        global file

        X = file[feature_col]

        y = file[target_col]

        # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y,
        test_size=float(split_size.get()))


        from sklearn.linear_model import LinearRegression
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.metrics import r2_score,mean_squared_error,explained_variance_score,mean_absolute_error,mean_squared_log_err

        poly_features = PolynomialFeatures(degree=int(degree.get()))

        # transforms the existing features to higher degree features.
        X_train_poly = poly_features.fit_transform(X_train)

        # fit the transformed features to Linear Regression
        model = LinearRegression()
        model.fit(X_train_poly, y_train)


        model_pred=model.predict(poly_features.fit_transform(X_test))
        MSE=mean_squared_error(model_pred,y_test)
        r2=r2_score(model_pred,y_test)
        EV=explained_variance_score(model_pred,y_test)
        MAE=mean_absolute_error(model_pred,y_test)
        MSLE=mean_squared_log_error(model_pred,y_test)
        MeAE=median_absolute_error(model_pred,y_test)
        intercept=model.intercept_
        coefficients=model.coef_
```

```
Label(root, text='PDF has been generated.' , font=('Helvetica',
10, 'bold'), bg="light blue",
      relief="solid").place(x=450, y=240)
Label(root, text=f'Intercept : {intercept}', font=('Helvetica',
10, 'bold'), bg="light blue",
      relief="solid").place(x=450, y=270)
Label(root, text=f'Coefficients  : {coefficients}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=300)
Label(root, text=f'Mean Squared Error : {MSE}', font=('Helvetica',
10, 'bold'), bg="light blue",
      relief="solid").place(x=450, y=330)
Label(root, text=f'R Square  : {r2}', font=('Helvetica', 10, 'bold'),
bg="light blue",
        relief="solid").place(x=450, y=360)
Label(root, text=f'Explained Variance : {EV}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=390)
Label(root, text=f'Mean Absolute Error  : {MAE}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=420)
Label(root, text=f'Mean Squared Log Error : {MSLE}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=450)
Label(root, text=f'Median Absolute Error : {MeAE}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=480)




# Create a PDF file with A4 size
with PdfPages('Polynomial Regression.pdf') as pdf:
    pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

    ##Page 1: Descriptive statistics
    numeric_df = file.describe()

    # Create a new page for the descriptive statistics
    fig, ax = plt.subplots(figsize=(8.27, 11.69))
    ax.axis('off')

    # Add row labels and round off values to two decimal places
    numeric_df.insert(0, '', numeric_df.index)
    numeric_df = numeric_df.round(2)

    table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
    columns, cellLoc='center', loc='center')
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1, 1.5)
```

```
            # Add a title to the table
            title = ax.set_title('Descriptive Statistics for Numeric Variables',
            fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)

            # Add the page to the PDF file
            pdf.savefig()


            # Page 2: Performance metrics
            fig, ax = plt.subplots(figsize=(8.27, 11.69))
            ax.axis('off')
            metrics_data = [['Intercept', intercept], ['Coefficient',
            coefficients],['R Square', r2], ['Mean Squared Error', MSE],
                            ['Explained Variance', EV],['Mean Squared Log Error',
                            MSLE],['Mean Absolute Error', MAE],
                            ['Median Absolute Error', MeAE]]
            metrics_df = pd.DataFrame(metrics_data, columns=['Metric', 'Value'])
            metrics_table = ax.table(cellText=metrics_df.values,
            colLabels=metrics_df.columns, cellLoc='center', loc='center')
            metrics_table.auto_set_font_size(False)
            metrics_table.set_fontsize(10)
            metrics_table.scale(1, 1.5)
            title = ax.set_title('Performance Metrics', fontsize=16)
            title.set_y(0.95)
            fig.subplots_adjust(top=0.85)
            pdf.savefig()

            ##Page 3: Visualise the Dataset
            plt.figure(figsize=(8.27, 5.87))
            plt.title("Heatmap of Correlation Matrix for Numeric␣
↪Variables",fontsize=16)
            sns.heatmap(file.corr())
            pdf.savefig()


            # Page 4

            plt.figure(figsize=(8.27, 5.87))
            plt.title("Distribution plot of Difference between Actual and Predicted␣
↪Response",fontsize=16)
            plt.subplots(figsize=(8.27,5.8))
            sns.displot(model_pred-y_test,kind='kde')
            pdf.savefig()



            # Close the plots
            plt.close('all')
```

```
L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
L1.place(x=200, y=430)

def predict():
    x_dummies = s.get().split(",")
    x_tests = []

    for i in x_dummies:
        x_tests.append(float(i))

    global model  # Access the global model variable
    y_pred = model.predict([x_tests])
    L1.config(text=str(y_pred))

listbox = Listbox(root, selectmode="multiple")
listbox.pack

# Create label and entry for split size
Label(root, font="System", text="Enter the split size:").place(x=20, y=270)
split_size = tk.StringVar()
Entry(root, textvariable=split_size).place(x=200, y=270)

#Degree
Label(root, font="System", text="Degree:").place(x=20, y=300)
degree = StringVar()
Entry(root, textvariable=degree).place(x=200, y=300)

# Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=30).place(x=200,y=330)
Label(root,font="System",text='Feature Variable Values').place(x=20, y=330)



l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)


e1 = Entry(root, text='')
e1.grid(row=0, column=1)
Button(root,
text='Open',command=load_data,activeforeground="white",activebackground="black").
grid(row=0, column=2)

# To switch master window
Button(root, text='Back', command=lambda: switch_windows(root,
master),activeforeground="white",activebackground="black").grid(row=0, column=4)


box1 = Listbox(root, selectmode='multiple')
box1.grid(row=11, column=0)
```

```
    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',
    command=getx,activeforeground="white",activebackground="black").grid(row=14,
    column=1)
    Button(root, text='Delete X',
    command=deletex,activeforeground="white",activebackground="black").grid(row=15,
    column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',
    command=gety,activeforeground="white",activebackground="black").grid(row=14,
    column=2)
    Button(root, text='Delete Y',
    command=deletey,activeforeground="white",activebackground="black").grid(row=15,
    column=2)

    Button(root, text="RUN MODEL",
    command=fit,activeforeground="white",activebackground="black").place(x=350,

    y=270)
    Button(root, text="PREDICT",
    command=predict,activeforeground="white",activebackground="black").place(x=350,
    y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Regreesion.add_command(label ='Polynomial Regression', command
= PolynomialRegression)
```

11 5) Ridge Regression

[10]:

```
def RidgeRegression():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Ridge Regression")
    root.config(bg="lavender")
    master.withdraw()


    def data():
        global filename, file,feature_names,target_names

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title
        ="Select file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        feature_names = list(file.columns)
        target_names = []
        for i in file.columns:
            if file[i].dtype == object:
                target_names.append(i)
                class_names.extend(list(file[i].unique()))

    def load_data():
        global filename, file

        try:
            del file
            e1.delete(0, tk.END)
            box1.delete(0, tk.END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models', title=
        "Select file")
        e1.insert(0, filename)

        try:
            file = pd.read_csv(filename)

            for i in file.columns:
                box1.insert(tk.END, i)

            for i in file.columns:
                if file[i].dtype == np.float64:
                    file[i].fillna(file[i].mean(), inplace=True)
                elif file[i].dtype == np.int64:
                    file[i].fillna(file[i].median(), inplace=True)
                elif file[i].dtype == object:
                    imp = SimpleImputer(missing_values=np.nan,
                    strategy='most_frequent')
                    file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))



            # Create a new window to display the table
            top = tk.Toplevel()
            top.title("Data Table")
            f = tk.Frame(top)
            f.pack(fill=tk.BOTH, expand=1)

            # Use pandastable to display the data in a table
            pt = Table(f, dataframe=file)
            pt.show()

        except FileNotFoundError:
            messagebox.showerror("Error", "Please select a file.")

    def getx():
        global feature_col
        feature_col.extend([file.columns[i] for i in box1.curselection() if i not in
        feature_col])
        box2.insert(END, *feature_col[-len(box1.curselection()):])

        # Update feature_names
        global feature_names
        feature_names = list(file[feature_col].columns)


    def deletex():
        for i in reversed(box2.curselection()):
            feature_col.remove(box2.get(i))
            box2.delete(i)

        # Update feature_names
        global feature_names
        feature_names = list(file[feature_col].columns)


    def gety():
        global target_col
        target_col = [file.columns[i] for i in box1.curselection() if i not in
        target_col]
        box3.insert(END, *target_col[-len(box1.curselection()):])

        # Update target_names
```

```
        global target_names
        target_names = list(file[target_col].columns)
        global class_names
        class_names = list(set(file[target_col]))
        #list(set(data['target_column']))

    def deletey():
        for i in reversed(box3.curselection()):
            target_col.remove(box3.get(i))
            box3.delete(i)

        # Update target_names
        global target_names
        target_names = list(file[target_col].columns)
        global class_names
        class_names = list(set(file[target_col]))

    model=None

    def fit():
        global model
        global file

        X = file[feature_col]

        y = file[target_col]

        # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y,
        test_size=float(split_size.get()))


        from sklearn.linear_model import Ridge
        from sklearn.metrics import␣
↪r2_score,mean_squared_error,explained_variance_score,
        mean_absolute_error,mean_squared_log_error,median_absolute_error

        model = Ridge(alpha=float(alpha.get()),max_iter=int(max_iter.get()))
        model.fit(X_train,y_train)

        model_pred=model.predict(X_test)
        MSE=mean_squared_error(model_pred,y_test)
        r2=r2_score(model_pred,y_test)
        EV=explained_variance_score(model_pred,y_test)
        MAE=mean_absolute_error(model_pred,y_test)
        MSLE=mean_squared_log_error(model_pred,y_test)
        MeAE=median_absolute_error(model_pred,y_test)
        intercept=model.intercept_
```

```
coefficients=model.coef_

Label(root, text='PDF has been generated.' , font=('Helvetica', 10, 'bold'),
bg="light blue",
      relief="solid").place(x=450, y=240)
Label(root, text=f'Intercept : {intercept}', font=('Helvetica', 10, 'bold'),
bg="light blue",
      relief="solid").place(x=450, y=270)
Label(root, text=f'Coefficients  : {coefficients}', font=('Helvetica', 10,
'bold'), bg="light blue",
        relief="solid").place(x=450, y=300)
Label(root, text=f'Mean Squared Error : {MSE}', font=('Helvetica', 10,
'bold'), bg="light blue",
      relief="solid").place(x=450, y=330)
Label(root, text=f'R Square  : {r2}', font=('Helvetica', 10, 'bold'),
bg="light blue",
        relief="solid").place(x=450, y=360)
Label(root, text=f'Explained Variance : {EV}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=390)
Label(root, text=f'Mean Absolute Error  : {MAE}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=420)
Label(root, text=f'Mean Squared Log Error : {MSLE}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=450)
Label(root, text=f'Median Absolute Error : {MeAE}', font=('Helvetica',
10, 'bold'), bg="light blue",
        relief="solid").place(x=450, y=480)


# Create a PDF file with A4 size
with PdfPages('Ridge Regression.pdf') as pdf:
    pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

    ##Page 1: Descriptive statistics
    numeric_df = file.describe()

    # Create a new page for the descriptive statistics
    fig, ax = plt.subplots(figsize=(8.27, 11.69))
    ax.axis('off')

    # Add row labels and round off values to two decimal places
    numeric_df.insert(0, '', numeric_df.index)
    numeric_df = numeric_df.round(2)

    table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
    columns, cellLoc='center', loc='center')
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1, 1.5)
```

```
# Add a title to the table
title = ax.set_title('Descriptive Statistics for Numeric Variables',
fontsize=16)
title.set_y(0.95)
fig.subplots_adjust(top=0.85)

# Add the page to the PDF file
pdf.savefig()



# Page 2: Performance metrics
fig, ax = plt.subplots(figsize=(8.27, 11.69))
ax.axis('off')
metrics_data = [['Intercept', intercept], ['Coefficient',
coefficients],['R Square', r2], ['Mean Squared Error', MSE],
                ['Explained Variance', EV],['Mean Squared Log Error',
                MSLE],['Mean Absolute Error', MAE],
                ['Median Absolute Error', MeAE]]
metrics_df = pd.DataFrame(metrics_data, columns=['Metric', 'Value'])
metrics_table = ax.table(cellText=metrics_df.values,
colLabels=metrics_df.columns, cellLoc='center', loc='center')
metrics_table.auto_set_font_size(False)
metrics_table.set_fontsize(10)
metrics_table.scale(1, 1.5)
title = ax.set_title('Performance Metrics', fontsize=16)
title.set_y(0.95)
fig.subplots_adjust(top=0.85)
pdf.savefig()

##Page 3: Visualise the Dataset
plt.figure(figsize=(8.27, 5.87))
plt.title(
"Heatmap of Correlation Matrix for Numeric Variables",fontsize=16)
sns.heatmap(file.corr())
pdf.savefig()

## Page 4 :
plt.figure(figsize=(8.27, 5.87))
alphas = np.linspace(0.01,500,100)
ridge = Ridge(max_iter=10000)
coefs = []

for a in alphas:
    ridge.set_params(alpha=a)
    ridge.fit(X_train, y_train)
    coefs.append(ridge.coef_)

ax = plt.gca()

ax.plot(alphas, coefs)
ax.set_xscale('log')
```

```
            plt.axis('tight')
            plt.xlabel('alpha')
            plt.ylabel('Standardized Coefficients')
            plt.title('Ridge coefficients as a function of alpha');


            pdf.savefig()


            # Page 5

            plt.figure(figsize=(8.27, 5.87))
            plt.title(
            "Distribution plot of Difference between Actual and Predicted␣
↪Response",fontsize=16)
            plt.subplots(figsize=(8.27,5.8))
            sns.displot(model_pred-y_test,kind='kde')
            pdf.savefig()



            # Close the plots
            plt.close('all')

    L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
    L1.place(x=200, y=430)

    def predict():
        x_dummies = s.get().split(",")
        x_tests = []

        for i in x_dummies:
            x_tests.append(float(i))

        global model  # Access the global model variable
        y_pred = model.predict([x_tests])
        L1.config(text=str(y_pred))

    listbox = Listbox(root, selectmode="multiple")
    listbox.pack

    # Create label and entry for split size
    Label(root, font="System", text="Enter the split size:").place(x=20, y=270)
    split_size = tk.StringVar()
    Entry(root, textvariable=split_size).place(x=200, y=270)

    #Max iter
    Label(root, font="System", text="Max iter:").place(x=20, y=300)
    max_iter = StringVar()
    Entry(root, textvariable=max_iter).place(x=200, y=300)

    #choose Alpha
```

```
Label(root, font="System", text="Alpha").place(x=20, y=330)
alpha= StringVar()
Entry(root, textvariable=alpha).place(x=200, y=330)

 # Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=30).place(x=200,y=360)
Label(root,font="System",text='Feature Variable Values').place(x=20, y=360)

l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)


e1 = Entry(root, text='')
e1.grid(row=0, column=1)
Button(root,
text='Open',command=load_data,activeforeground="white",activebackground="black").
grid(row=0, column=2)

# To switch master window
Button(root, text='Back', command=lambda: switch_windows(root,
master),activeforeground="white",activebackground="black").grid(row=0, column=4)


box1 = Listbox(root, selectmode='multiple')
box1.grid(row=11, column=0)


Label(root, text='Features').grid(row=10, column=1)
box2 = Listbox(root,selectmode='multiple')
box2.grid(row=11, column=1)
Button(root, text='Select X',
command=getx,activeforeground="white",activebackground="black").grid(row=14,
column=1)
Button(root, text='Delete X',
command=deletex,activeforeground="white",activebackground="black").grid(row=15,
column=1)

Label(root, text='Respose').grid(row=10, column=2)
box3 = Listbox(root,selectmode='multiple')
box3.grid(row=11, column=2)
Button(root, text='Select Y',
command=gety,activeforeground="white",activebackground="black").grid(row=14,
column=2)
Button(root, text='Delete Y',
command=deletey,activeforeground="white",activebackground="black").grid(row=15,
column=2)

Button(root, text="RUN MODEL",
command=fit,activeforeground="white",activebackground="black").place(x=350,
y=270)
Button(root, text="PREDICT",
    command=predict,activeforeground="white",activebackground="black").place(x=350
    , y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Regreesion.add_command(label ='Ridge Regression', command = RidgeRegression)
```

12 6) Lasso Regression

[11]:

```
def LassoRegression():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Lasso Regression")
    root.config(bg="lavender")
    master.withdraw()

    def data():
        global filename, file,feature_names,target_names

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models
        ', title="Select file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan,
                strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

        feature_names = list(file.columns)
        target_names = []
        for i in file.columns:
```

```
        if file[i].dtype == object:
            target_names.append(i)
            class_names.extend(list(file[i].unique()))

def load_data():
    global filename, file

    try:
        del file
        e1.delete(0, tk.END)
        box1.delete(0, tk.END)
    except NameError:
        pass

    filename = askopenfilename(initialdir=r'C:\Project\ML Models
    ', title="Select file")
    e1.insert(0, filename)

    try:
        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(tk.END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan,
                strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))



        # Create a new window to display the table
        top = tk.Toplevel()
        top.title("Data Table")
        f = tk.Frame(top)
        f.pack(fill=tk.BOTH, expand=1)

        # Use pandastable to display the data in a table
        pt = Table(f, dataframe=file)
        pt.show()

    except FileNotFoundError:
        messagebox.showerror("Error", "Please select a file.")

def getx():
    global feature_col
    feature_col.extend([file.columns[i] for i in box1.curselection(
    ) if i not in feature_col])
    box2.insert(END, *feature_col[-len(box1.curselection()):])

    # Update feature_names
    global feature_names
    feature_names = list(file[feature_col].columns)


def deletex():
    for i in reversed(box2.curselection()):
        feature_col.remove(box2.get(i))
        box2.delete(i)

    # Update feature_names
    global feature_names
    feature_names = list(file[feature_col].columns)


def gety():
    global target_col
    target_col = [file.columns[i] for i in box1.curselection() if i not in
    target_col]
    box3.insert(END, *target_col[-len(box1.curselection()):])

    # Update target_names
    global target_names
    target_names = list(file[target_col].columns)
    global class_names
    class_names = list(set(file[target_col]))
    #list(set(data['target_column']))

def deletey():
    for i in reversed(box3.curselection()):
        target_col.remove(box3.get(i))
        box3.delete(i)

    # Update target_names
    global target_names
    target_names = list(file[target_col].columns)
    global class_names
    class_names = list(set(file[target_col]))

model=None

def fit():
    global model
    global file

    X = file[feature_col]

    y = file[target_col]
```

```
        # get target column name
        target_name = pd.unique(file[target_col].values.ravel())
        feature_names=X.columns.tolist()

        # Split data into training and testing sets
        X_train,X_test,y_train,y_test = train_test_split(X,y
        , test_size=float(split_size.get()))


        from sklearn.linear_model import Lasso
        from sklearn.metrics import
        r2_score,mean_squared_error,explained_variance_score,mean_absolute_error,
        mean_squared_log_error,median_absolute_error

        model = Lasso(alpha=float(alpha.get()),max_iter=int(max_iter.get()))
        model.fit(X_train,y_train)
        model_pred=model.predict(X_test)
        MSE=mean_squared_error(model_pred,y_test)
        r2=r2_score(model_pred,y_test)
        EV=explained_variance_score(model_pred,y_test)
        MAE=mean_absolute_error(model_pred,y_test)
        MSLE=mean_squared_log_error(model_pred,y_test)
        MeAE=median_absolute_error(model_pred,y_test)


        Label(root, text='PDF has been generated.' , font=('Helvetica
        ', 10, 'bold'), bg="light blue",
              relief="solid").place(x=450, y=240)
        Label(root, text=f'Intercept : {model.intercept_}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
              relief="solid").place(x=450, y=270)
        Label(root, text=f'Coefficients  : {model.coef_}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=300)
        Label(root, text=f'Mean Squared Error : {MSE}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
              relief="solid").place(x=450, y=330)
        Label(root, text=f'R Square  : {r2}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=360)
        Label(root, text=f'Explained Variance : {EV}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=390)
        Label(root, text=f'Mean Absolute Error  : {MAE}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=420)
        Label(root, text=f'Mean Squared Log Error : {MSLE}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=450)
        Label(root, text=f'Median Absolute Error : {MeAE}', font=('Helvetica
        ', 10, 'bold'), bg="light blue",
                relief="solid").place(x=450, y=480)
```

```
# Create a PDF file with A4 size
with PdfPages('Losso Regression.pdf') as pdf:
    pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

    ##Page 1: Descriptive statistics
    numeric_df = file.describe()

    # Create a new page for the descriptive statistics
    fig, ax = plt.subplots(figsize=(8.27, 11.69))
    ax.axis('off')

    # Add row labels and round off values to two decimal places
    numeric_df.insert(0, '', numeric_df.index)
    numeric_df = numeric_df.round(2)

    table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
    columns, cellLoc='center', loc='center')
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1, 1.5)

    # Add a title to the table
    title = ax.set_title('Descriptive Statistics for Numeric Variables',
    fontsize=16)
    title.set_y(0.95)
    fig.subplots_adjust(top=0.85)

    # Add the page to the PDF file
    pdf.savefig()

    ##Page 2: Visualise the Dataset
    plt.figure(figsize=(8.27, 5.87))
    plt.title("Heatmap of Correlation Matrix for
    Numeric Variables",fontsize=16)
    plt.subplots(figsize=(8.27, 5.87))
    sns.heatmap(file.corr(numeric_only=True))
    pdf.savefig()



    ## Page 3 :
    plt.figure(figsize=(8.27, 5.87))
    alphas = np.linspace(0.01,500,100)
    lasso = Lasso(max_iter=10000)
    coefs = []

    for a in alphas:
        lasso.set_params(alpha=a)
        lasso.fit(X_train, y_train)
        coefs.append(lasso.coef_)
```

```
            ax = plt.gca()

            ax.plot(alphas, coefs)
            ax.set_xscale('log')
            plt.axis('tight')
            plt.xlabel('alpha')
            plt.ylabel('Standardized Coefficients')
            plt.title('Lasso coefficients as a function of alpha');
            pdf.savefig()



            # Page 4

            plt.figure(figsize=(8.27, 5.87))
            plt.title(
            "Distribution plot of Difference between Actual and Predicted␣
↪Response",fontsize=16)
            plt.subplots(figsize=(8.27,5.8))
            sns.displot(model_pred-y_test,kind='kde')
            pdf.savefig()

            # Close the plots
            plt.close('all')


    L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
    L1.place(x=200, y=430)

    def predict():
        x_dummies = s.get().split(",")
        x_tests = []

        for i in x_dummies:
            x_tests.append(float(i))

        global model  # Access the global model variable
        y_pred = model.predict([x_tests])
        L1.config(text=str(y_pred))

    listbox = Listbox(root, selectmode="multiple")
    listbox.pack

    # Create label and entry for split size
    Label(root, font="System", text="Enter the split size:").place(x=20, y=270)
    split_size = tk.StringVar()
    Entry(root, textvariable=split_size).place(x=200, y=270)

    #Max iter
    Label(root, font="System", text="Max iter:").place(x=20, y=300)
    max_iter = StringVar()
```

```
Entry(root, textvariable=max_iter).place(x=200, y=300)

#choose Alpha
Label(root, font="System", text="Alpha").place(x=20, y=330)
alpha= StringVar()
Entry(root, textvariable=alpha).place(x=200, y=330)

 # Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=30).place(x=200,y=360)
Label(root,font="System",text='Feature Variable Values').place(x=20, y=360)


l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)


e1 = Entry(root, text='')
e1.grid(row=0, column=1)
Button(root, text='Open',
command=load_data,activeforeground="white",activebackground="black").
grid(row=0, column=2)

# To switch master window
Button(root, text='Back', command=lambda: switch_windows(root,
master),activeforeground="white",activebackground="black").
grid(row=0,
column=4)


box1 = Listbox(root, selectmode='multiple')
box1.grid(row=11, column=0)


Label(root, text='Features').grid(row=10, column=1)
box2 = Listbox(root,selectmode='multiple')
box2.grid(row=11, column=1)
Button(root, text='Select X',
command=getx,activeforeground="white",activebackground="black").grid(row=14,
column=1)
Button(root, text='Delete X',
command=deletex,activeforeground="white",activebackground="black").
grid(row=15, column=1)

Label(root, text='Respose').grid(row=10, column=2)
box3 = Listbox(root,selectmode='multiple')
box3.grid(row=11, column=2)
Button(root, text='Select Y',
command=gety,activeforeground="white",activebackground="black").grid(row=14,
column=2)
Button(root, text='Delete Y',
command=deletey,activeforeground="white",activebackground="black").
```

```
    grid(row=15, column=2)


    Button(root, text="RUN MODEL",
    command=fit,activeforeground="white",activebackground="black").place(x=350,
    y=270)
    Button(root, text="PREDICT",
    command=predict,activeforeground="white",activebackground="black").
    place(x=350, y=430)
def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()

Regreesion.add_command(label ='Lasso Regression', command = LassoRegression)
```

13 7) Support Vector Regression

[12]:

```
def SVR():
    root=Toplevel(master)
    root.geometry('800x600')
    root.title("Support Vector Regression")
    root.config(bg="lavender")
    master.withdraw()


    def data():
        global filename, file,feature_names,target_names

        try:
            del file
            e1.delete(0, END)
            box1.delete(0, END)
        except NameError:
            pass

        filename = askopenfilename(initialdir=r'C:\Project\ML Models',
        title="Select file")
        e1.insert(0, filename)
        e1.config(text=filename)

        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
```

```
            imp = SimpleImputer(missing_values=np.nan,
            strategy='most_frequent')
            file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))

    feature_names = list(file.columns)
    target_names = []
    for i in file.columns:
        if file[i].dtype == object:
            target_names.append(i)
            class_names.extend(list(file[i].unique()))

def load_data():
    global filename, file

    try:
        del file
        e1.delete(0, tk.END)
        box1.delete(0, tk.END)
    except NameError:
        pass

    filename = askopenfilename(initialdir=r'C:\Project\ML Models',
    title="Select file")
    e1.insert(0, filename)

    try:
        file = pd.read_csv(filename)

        for i in file.columns:
            box1.insert(tk.END, i)

        for i in file.columns:
            if file[i].dtype == np.float64:
                file[i].fillna(file[i].mean(), inplace=True)
            elif file[i].dtype == np.int64:
                file[i].fillna(file[i].median(), inplace=True)
            elif file[i].dtype == object:
                imp = SimpleImputer(missing_values=np.nan,
                strategy='most_frequent')
                file[i] = imp.fit_transform(file[i].values.reshape(-1, 1))



        # Create a new window to display the table
        top = tk.Toplevel()
        top.title("Data Table")
        f = tk.Frame(top)
        f.pack(fill=tk.BOTH, expand=1)

        # Use pandastable to display the data in a table
        pt = Table(f, dataframe=file)
        pt.show()
    except FileNotFoundError:
        messagebox.showerror("Error", "Please select a file.")

def getx():
    global feature_col
    feature_col.extend([file.columns[i] for i in box1.curselection() if i
    not in feature_col])
    box2.insert(END, *feature_col[-len(box1.curselection()):])

    # Update feature_names
    global feature_names
    feature_names = list(file[feature_col].columns)


def deletex():
    for i in reversed(box2.curselection()):
        feature_col.remove(box2.get(i))
        box2.delete(i)

    # Update feature_names
    global feature_names
    feature_names = list(file[feature_col].columns)


def gety():
    global target_col
    target_col = [file.columns[i] for i in box1.curselection() if i not in
    target_col]
    box3.insert(END, *target_col[-len(box1.curselection()):])

    # Update target_names
    global target_names
    target_names = list(file[target_col].columns)
    global class_names
    class_names = list(set(file[target_col]))
    #list(set(data['target_column']))

def deletey():
    for i in reversed(box3.curselection()):
        target_col.remove(box3.get(i))
        box3.delete(i)

    # Update target_names
    global target_names
    target_names = list(file[target_col].columns)
    global class_names
    class_names = list(set(file[target_col]))

model=None

def fit():
    global model
    global file

    X = file[feature_col]

    y = file[target_col]

    # get target column name
    target_name = pd.unique(file[target_col].values.ravel())
    feature_names=X.columns.tolist()

    # Split data into training and testing sets
    X_train,X_test,y_train,y_test = train_test_split(X,y,
    test_size=float(split_size.get()))



    from sklearn.svm import SVR
    from sklearn.metrics import
    r2_score,mean_squared_error,explained_variance_score,mean_absolute_error,
    mean_squared_log_error,median_absolute_error

    model = SVR(epsilon=float(eps.get()),kernel=Kernal.get())
    #fiting model_cv
    model.fit(X_train,np.ravel(y_train))

    model_pred=model.predict(X_test)
    MSE=mean_squared_error(model_pred,y_test)
    r2=r2_score(model_pred,y_test)
    EV=explained_variance_score(model_pred,y_test)
    MAE=mean_absolute_error(model_pred,y_test)
    MSLE=mean_squared_log_error(model_pred,y_test)
    MeAE=median_absolute_error(model_pred,y_test)
    intercept=model.intercept_
    coefficients=model.coef_

    Label(root, text='PDF has been generated.' , font=('Helvetica', 10,
    'bold'), bg="light blue",
          relief="solid").place(x=450, y=240)
    Label(root, text=f'Intercept : {intercept}', font=('Helvetica',
    10, 'bold'), bg="light blue",
          relief="solid").place(x=450, y=270)
    Label(root, text=f'Coefficients  : {coefficients}', font=('Helvetica',
    10, 'bold'), bg="light blue",
            relief="solid").place(x=450, y=300)
    Label(root, text=f'Mean Squared Error : {MSE}', font=('Helvetica', 10,
    'bold'), bg="light blue",
          relief="solid").place(x=450, y=330)
    Label(root, text=f'R Square  : {r2}', font=('Helvetica', 10, 'bold'),
    bg="light blue",
            relief="solid").place(x=450, y=360)
    Label(root, text=f'Explained Variance : {EV}', font=('Helvetica', 10,
    'bold'), bg="light blue",
            relief="solid").place(x=450, y=390)
    Label(root, text=f'Mean Absolute Error  : {MAE}', font=('Helvetica', 10,
    'bold'), bg="light blue",
            relief="solid").place(x=450, y=420)
    Label(root, text=f'Mean Squared Log Error : {MSLE}', font=('Helvetica',
    10, 'bold'), bg="light blue",
            relief="solid").place(x=450, y=450)
    Label(root, text=f'Median Absolute Error : {MeAE}', font=('Helvetica',
    10, 'bold'), bg="light blue",
            relief="solid").place(x=450, y=480)




    # Create a PDF file with A4 size
    with PdfPages('Support Vector Regression.pdf') as pdf:
        pdf.infodict()['_pagesize'] = (842, 595)  # A4 size in points

        ##Page 1: Descriptive statistics
        numeric_df = file.describe()

        # Create a new page for the descriptive statistics
        fig, ax = plt.subplots(figsize=(8.27, 11.69))
        ax.axis('off')

        # Add row labels and round off values to two decimal places
        numeric_df.insert(0, '', numeric_df.index)
        numeric_df = numeric_df.round(2)

        table = ax.table(cellText=numeric_df.values, colLabels=numeric_df.
        columns, cellLoc='center', loc='center')
        table.auto_set_font_size(False)
        table.set_fontsize(10)
        table.scale(1, 1.5)

        # Add a title to the table
        title = ax.set_title('Descriptive Statistics for Numeric Variables'
        , fontsize=16)
        title.set_y(0.95)
        fig.subplots_adjust(top=0.85)

        # Add the page to the PDF file
        pdf.savefig()


        # Page 2: Performance metrics
        fig, ax = plt.subplots(figsize=(8.27, 11.69))
        ax.axis('off')
        metrics_data = [['Intercept', intercept], ['Coefficient'
        , coefficients],['R Square', r2], ['Mean Squared Error', MSE],
                        ['Explained Variance', EV]
```

```
                        ,['Mean Squared Log Error', MSLE],
                        ['Mean Absolute Error', MAE],
                        ['Median Absolute Error', MeAE]]
        metrics_df = pd.DataFrame(metrics_data, columns=['Metric', 'Value'])
        metrics_table = ax.table(cellText=metrics_df.values,
        colLabels=metrics_df.columns, cellLoc='center', loc='center')
        metrics_table.auto_set_font_size(False)
        metrics_table.set_fontsize(10)
        metrics_table.scale(1, 1.5)
        title = ax.set_title('Performance Metrics', fontsize=16)
        title.set_y(0.95)
        fig.subplots_adjust(top=0.85)
        pdf.savefig()

        ##Page 2: Visualise the Dataset
        plt.figure(figsize=(8.27, 5.87))
        plt.title
        ("Heatmap of Correlation Matrix for Numeric Variables",fontsize=16)
        sns.heatmap(file.corr())
        pdf.savefig()

        ##Page 3: Visualizing the SVR results
        plt.figure(figsize=(8.27, 5.87))
        plt.scatter(X_test, y_test, color = 'red')
        plt.plot(X_test,model.predict(X_test), color = 'blue')
        plt.title('Support Vectore Regression Model')
        plt.xlabel('X')
        plt.ylabel('y')
        pdf.savefig()




        # Close the plots
        plt.close('all')


L1 = Label(root, font=('Helvetica', 10, 'bold'), bg="light blue")
L1.place(x=200, y=430)

def predict():
    x_dummies = s.get().split(",")
    x_tests = []

    for i in x_dummies:
        x_tests.append(float(i))

    global model  # Access the global model variable
    y_pred = model.predict([x_tests])
    L1.config(text=str(y_pred))

listbox = Listbox(root, selectmode="multiple")
```

```
listbox.pack

# Create label and entry for split size
Label(root, font="System", text="Enter the split size:").place(x=20, y=270)
split_size = tk.StringVar()
Entry(root, textvariable=split_size).place(x=200, y=270)



Label(root,font="System",text="Epsilon").place(x=20,y=300)
Label(root,font="System",text="Kernal").place(x=20,y=330)


eps=tk.StringVar()
choose=ttk.Combobox(root,width = 30, textvariable= eps)
choose['values']=(".1",".2",".3",".4")
choose.place(x=200,y=300)

Kernal=tk.StringVar()
choose=ttk.Combobox(root,width=30,textvariable= Kernal)
choose['values']=('linear','poly','rbf','sigmoid','precomputed')
choose.place(x=200,y=330)




 # Create label and entry for Feature Variabel Values
s = StringVar()
Entry(root,text=s,width=30).place(x=200,y=390)
Label(root,font="System",text='Feature Variable Values').place(x=20, y=390)


l1 = Label(root, text='Select Data File')
l1.grid(row=0, column=0)


e1 = Entry(root, text='')
e1.grid(row=0, column=1)
Button(root,
text='Open',command=load_data,activeforeground="white",activebackground=
"black").grid(row=0, column=2)

# To switch master window
Button(root, text='Back', command=lambda: switch_windows(root
, master),activeforeground="white",activebackground="black").grid(row=0,
column=4)


box1 = Listbox(root, selectmode='multiple')
box1.grid(row=11, column=0)
```

```
    Label(root, text='Features').grid(row=10, column=1)
    box2 = Listbox(root,selectmode='multiple')
    box2.grid(row=11, column=1)
    Button(root, text='Select X',
    command=getx,activeforeground="white",activebackground="black").grid(row=14,
    column=1)
    Button(root, text='Delete X',
    command=deletex,activeforeground="white",activebackground="black").
    grid(row=15, column=1)

    Label(root, text='Respose').grid(row=10, column=2)
    box3 = Listbox(root,selectmode='multiple')
    box3.grid(row=11, column=2)
    Button(root, text='Select Y',
    command=gety,activeforeground="white",activebackground="black").grid(row=14,
    column=2)
    Button(root, text='Delete Y',
    command=deletey,activeforeground="white",activebackground="black").
    grid(row=15,
    column=2)


    Button(root, text="RUN MODEL",
    command=fit,activeforeground="white",activebackground="black").place(x=350,
    y=270)
    Button(root, text="PREDICT",
    command=predict,activeforeground="white",activebackground="black").
    place(x=350, y=430)

def switch_windows(from_window, to_window):
    from_window.withdraw()
    to_window.deiconify()
Regreesion.add_command(label ='Support Vector Regression', command = SVR)
```

```
[ ]: master.mainloop()
```