This code is a Python program that provides a simple GUI interface for loading and analyzing CSV files. It uses various libraries for data visualization and machine learning, such as tkinter, pandas, plotly, and scikit-learn. Here is a detailed explanation of each line of code:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

* import tempfile
* import tkinter as tk
* import webbrowser
* from tkinter import filedialog
* from tkinter import ttk
* import pandas as pd
* import plotly.express as px
* import plotly.graph\_objs as go
* import plotly.io as pio
* from plotly.subplots import make\_subplots
* from sklearn.linear\_model import LinearRegression
* from sklearn.metrics import mean\_squared\_error, r2\_score
* from sklearn.model\_selection import train\_test\_split
* These lines import the necessary libraries for the program.
  + tempfile is used for creating temporary files,
  + tkinter is used for creating GUI interfaces,
  + webbrowser is used for opening HTML files in a web browser,
  + filedialog and .ttk are both used for creating file dialogs,
  + pandas is used for data manipulation,
  + plotly is used for data visualization, and,
  + scikit-learn is used for machine learning:
    - Specifically,
      * LinearRegression,
      * mean\_squared\_error,
      * r2\_score, and
      * train\_test\_split are ALL imported from scikit-learn.

Text

Description automatically generated

def open\_web\_view(html\_content):

with tempfile.NamedTemporaryFile(delete=False, suffix='.html') as f:

f.write(html\_content.encode())

f.flush()

webbrowser.open(f.name)

Text

Description automatically generated

* This is a function that takes HTML content as input, creates a temporary HTML file using tempfile, writes the HTML content to the file, flushes the buffer to ensure that the content is written to the file, and then opens the HTML file in a web browser using webbrowser.
* This code defines a function called open\_web\_view that takes a string parameter html\_content.
  + The first line of the function uses the tempfile module to create a temporary file with a .html suffix and returns a file object.
  + This temporary file is created using the NamedTemporaryFile method and the delete parameter set to False so that the file will not be automatically deleted when the program is finished executing.
* The second line of the function writes the html\_content string to the temporary file using the write method, which is then encoded using the encode() method.
* The flush() method is then called to make sure that the data is written to the file and it's available to be read.
* Finally, the function calls webbrowser.open(f.name), which opens the temporary file in the default web browser of the user's system. The f.name attribute returns the file name of the temporary file, which is used as the URL to open in the browser

def load\_dataset(file\_path):

df = pd.read\_csv(file\_path)

return df

Text

Description automatically generated

* This is a function that takes a file path as input, reads the CSV file using pandas, and returns a pandas DataFrame object.
* This is a Python function called load\_dataset that takes one parameter called file\_path. The purpose of this function is to load a dataset that is stored in a CSV file and return it as a pandas DataFrame.
* def load\_dataset(file\_path):
  + This line defines a new function called load\_dataset that takes one argument, file\_path.
* df = pd.read\_csv(file\_path):
  + This line loads the CSV file located at file\_path using the read\_csv function provided by the pandas library.
    - The resulting DataFrame is stored in the variable df.
* return df:
  + This line returns the DataFrame that was loaded from the CSV file. The DataFrame can be used for further analysis, processing or visualization.
* Overall, this function is a simple utility that can be used to load CSV files into a pandas DataFrame. The returned DataFrame can then be used for various purposes, such as machine learning, data exploration, or visualization.

def train\_model(X\_train, y\_train):

model = LinearRegression()

model.fit(X\_train, y\_train)

return model

Text

Description automatically generated

* This is a function that takes training data (X\_train and y\_train) as input, creates a LinearRegression model using scikit-learn, trains the model using the training data, and returns the trained model.
* The purpose of this function is to train a linear regression model on the training data and return the trained model.
* def train\_model(X\_train, y\_train):
  + This line defines a new function called train\_model that takes two arguments, X\_train and y\_train.
* model = LinearRegression():
  + This line creates an instance of the LinearRegression class from the scikit-learn library.
    - This class provides a linear regression model that can be used for regression analysis.
* model.fit(X\_train, y\_train):
  + This line trains the linear regression model on the training data. The fit method takes two arguments, X\_train and y\_train, which are the features and labels of the training data, respectively.
    - The model is trained to learn a linear relationship between the features and labels.
* return model:
  + This line returns the trained linear regression model.
    - The trained model can be used to make predictions on new data or for further analysis, such as feature importance analysis or model comparison.
* Overall, this function is a simple utility that can be used to train a linear regression model on training data. The returned model can be used to make predictions on new data or for further analysis. Linear regression is a widely used machine learning algorithm for regression tasks, such as predicting the price of a house given its features.

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

return mse, r2

Text

Description automatically generated

* This is a function that takes a trained model (model), testing data (X\_test and y\_test), and evaluates the performance of the model using mean squared error (mean\_squared\_error) and R2 score (r2\_score) from scikit-learn.
  + It returns both the mean squared error and R2 score.
* The purpose of this function is to evaluate the performance of a machine learning model on a test dataset and return two performance metrics, mean squared error and R-squared.
* def evaluate\_model(model, X\_test, y\_test):
  + This line defines a new function called evaluate\_model that takes three arguments, model, X\_test, and y\_test.
* y\_pred = model.predict(X\_test):
  + This line uses the trained model to make predictions on the test data X\_test. The predicted values are stored in the variable y\_pred.
* mse = mean\_squared\_error(y\_test, y\_pred):
  + This line calculates the mean squared error (MSE) between the true values y\_test and the predicted values y\_pred. The mean squared error is a common metric used to evaluate regression models. It measures the average squared difference between the predicted and true values.
* r2 = r2\_score(y\_test, y\_pred):
  + This line calculates the R-squared (R2) coefficient between the true values y\_test and the predicted values y\_pred. The R-squared coefficient is a common metric used to evaluate regression models. It measures the proportion of variance in the dependent variable that is predictable from the independent variable(s).
* return mse, r2
  + This line returns two performance metrics, mean squared error and R-squared, as a tuple.
* Overall, this function is a simple utility that can be used to evaluate the performance of a machine learning model on a test dataset. The function calculates two common regression metrics, mean squared error and R-squared, which can be used to compare the performance of different models or to fine-tune hyperparameters. The returned metrics can be used to get an idea of how well the model is performing on new data.

def plot\_charts(df):

fig\_pie = px.pie(df, names=df.columns[-1], title='Pie Chart of Target Variable')

fig\_scatter = px.scatter\_matrix(df, dimensions=df.columns[:-1], color=df.columns[-1],

title='Scatter Matrix of Features')

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = train\_model(X\_train, y\_train)

mse = evaluate\_model(model, X\_test, y\_test)

mse\_var.set(f"MSE: {mse:.2f}")

r2\_var.set(f"R2 Score: {r2:.2f}")

fig\_regression = make\_subplots(rows=1, cols=X.shape[1], subplot\_titles=X.columns)

for idx, col in enumerate(X.columns):

fig\_regression.add\_trace(go.Scatter(x=X[col], y=y, mode='markers', name=col, showlegend=False), row=1, col=idx + 1)

X\_reg = X[[col]]

model\_reg = train\_model(X\_reg, y)

y\_reg = model\_reg.predict(X\_reg)

fig\_regression.add\_trace(go.Scatter(x=X[col], y=y\_reg, mode='lines', name=f"{col}\_reg", showlegend=False), row=1, col=idx + 1)

fig\_regression.update\_layout(title\_text='Linear Regression Plots', showlegend=False)

fig\_histograms = make\_subplots(rows=1, cols=X.shape[1], subplot\_titles=X.columns)

for idx, col in enumerate(X.columns):

fig\_histograms.add\_trace(go.Histogram(x=X[col], name=col), row=1, col=idx + 1)

fig\_histograms.update\_layout(title\_text='Histograms')

open\_web\_view(pio.to\_html(fig\_pie, full\_html=False))

open\_web\_view(pio.to\_html(fig\_scatter, full\_html=False))

open\_web\_view(pio.to\_html(fig\_regression, full\_html=False))

open\_web\_view(pio.to\_html(fig\_histograms, full\_html=False))

Text

Description automatically generated



* This function takes a pandas DataFrame (df) as input, creates various plots using plotly, trains a LinearRegression model using train\_test\_split from scikit-learn, evaluates the model using evaluate\_model, and then displays the plots and evaluation metrics in the web browser using open\_web\_view.
* Specifically, it creates a pie chart (fig\_pie) and scatter matrix (fig\_scatter) using plotly.express, creates regression plots (fig\_regression) and histograms (fig\_histograms) using plotly.subplots and go.
* Scatter and go.Histogram from plotly.graph\_objs.
* It then updates the layouts of the regression and histogram plots and finally, calls open\_web\_view to display all the figures in the web browser.
* def plot\_charts(df):
  + This line defines a new function called plot\_charts that takes one argument, df, which is a pandas DataFrame containing the dataset.
* fig\_pie = px.pie(df, names=df.columns[-1], title='Pie Chart of Target Variable'):
  + This line creates a pie chart using the px.pie function from the Plotly Express library. The chart shows the distribution of the target variable, which is assumed to be in the last column of the dataset. The chart is stored in the variable fig\_pie.
* fig\_scatter = px.scatter\_matrix(df, dimensions=df.columns[:-1], color=df.columns[-1],
* title='Scatter Matrix of Features'):
  + This line creates a scatter matrix plot using the px.scatter\_matrix function from the Plotly Express library. The plot shows the pairwise relationships between the features in the dataset. The color of the points is determined by the target variable, which is assumed to be in the last column of the dataset. The plot is stored in the variable fig\_scatter.
* X = df.iloc[:, :-1]

y = df.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = train\_model(X\_train, y\_train)

mse = evaluate\_model(model, X\_test, y\_test)

mse\_var.set(f"MSE: {mse:.2f}")

r2\_var.set(f"R2 Score: {r2:.2f}"):

* + These lines split the dataset into training and testing sets, train a linear regression model on the training set, and evaluate the model on the testing set. The training set consists of all the features in the dataset except for the last column, which is assumed to be the target variable. The testing set consists of 20% of the data. The train\_model and evaluate\_model functions are called to train and evaluate the model. The mean squared error (MSE) and R-squared (R2) are computed and stored in mse variable. The MSE and R2 values are set to variables mse\_var and r2\_var respectively, which are later used to display these values on the GUI.
* fig\_regression = make\_subplots(rows=1, cols=X.shape[1], subplot\_titles=X.columns):
  + This line creates a subplot using the make\_subplots function from the Plotly library. The subplot consists of a row of charts, one for each feature in the dataset. The subplot is stored in the variable fig\_regression.
* for idx, col in enumerate(X.columns)

fig\_regression.add\_trace(go.Scatter(x=X[col], y=y, mode='markers', name=col, showlegend=False), row=1, col=idx + 1)

X\_reg = X[[col]]

model\_reg = train\_model(X\_reg, y)

y\_reg = model\_reg.predict(X\_reg)

fig\_regression.add\_trace(go.Scatter(x=X[col], y=y\_reg, mode='lines', name=f"{col}\_reg", showlegend=False), row=1, col=idx + 1):

* + These lines create scatter plots for each feature in the dataset using the go.Scatter function from the Plotly library. For each feature, two plots are created. The first plot is a scatter plot of the feature values versus the target variable. The second plot is a line plot of the linear regression fit for the feature. The regression line is trained using only the feature and target variable. The plots are added to the subplot fig\_regression using the add\_trace method.
* fig\_regression.update\_layout(title\_text='Linear Regression Plots', showlegend=False):
  + This line updates the layout of the fig\_regression subplot by setting the title to "Linear Regression Plots" and hiding the legend.
* fig\_histograms = make\_subplots(rows=1, cols=X.shape[1], subplot\_titles=X.columns):
  + This line creates another subplot using the make\_subplots function from the Plotly library. The subplot consists of a row of histograms, one for each feature in the dataset. The subplot is stored in the variable fig\_histograms.
* for idx, col in enumerate(X.columns)
* fig\_histograms.add\_trace(go.Histogram(x=X[col], name=col), row=1, col=idx + 1):
  + These lines create histograms for each feature in the dataset using the go.Histogram function from the Plotly library. The plots are added to the subplot fig\_histograms using the add\_trace method.
* fig\_histograms.update\_layout(title\_text='Histograms'):
  + This line updates the layout of the fig\_histograms subplot by setting the title to "Histograms".
* open\_web\_view(pio.to\_html(fig\_pie, full\_html=False))

open\_web\_view(pio.to\_html(fig\_scatter, full\_html=False))

open\_web\_view(pio.to\_html(fig\_regression, full\_html=False))

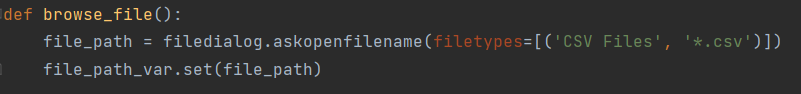
open\_web\_view(pio.to\_html(fig\_histograms, full\_html=False)):

* + These lines display the plots in a web view using the open\_web\_view function. The pio.to\_html function is used to convert the plots to HTML code, which is then displayed in the web view. Four plots are displayed: the pie chart, the scatter matrix, the linear regression plots, and the histograms.
* Overall, this function is a utility that can be used to visualize the given dataset using various charts and plots. It uses several libraries such as Plotly, Pandas, and scikit-learn to create and display the charts. The function is intended to be used in a GUI application or Jupyter notebook to provide an interactive way to explore and analyze the dataset.

def browse\_file():

file\_path = filedialog.askopenfilename(filetypes=[('CSV Files', '\*.csv')])

file\_path\_var.set(file\_path)



* This is a function that is called when the user clicks the "Browse" button.
  + It opens a file dialog using filedialog from tkinter and returns the file path of the selected CSV file.
  + The file path is then set as the value of the file\_path\_var variable.

def analyze\_data():

file\_path = file\_path\_var.get()

df = load\_dataset(file\_path)

plot\_charts(df)

Text

Description automatically generated

* This is a function that is called when the user clicks the "Analyze" button.
  + It gets the file path from the file\_path\_var variable, loads the CSV file using load\_dataset, and then calls plot\_charts to create and display the plots.

root = tk.Tk()

root.title("Data Analysis")

file\_path\_var = tk.StringVar()

mse\_var = tk.StringVar()

r2\_var = tk.StringVar()

ttk.Label(root, text="Select CSV file:").grid(column=0, row=0)

ttk.Entry(root, textvariable=file\_path\_var, width=50).grid

(column=1, row=0)

ttk.Button(root, text="Browse", command=browse\_file).grid(column=2, row=0)

ttk.Button(root, text="Analyze", command=analyze\_data).grid(column=1, row=1)

ttk.Label(root, textvariable=mse\_var).grid(column=1, row=2)

ttk.Label(root, textvariable=r2\_var).grid(column=1, row=3)

Text

Description automatically generated

* mse\_explanation = “MSE (Mean Squared Error) measures the average squared difference between predicted and actual values. Lower values indicate better model performance."
* r2\_explanation = "R2 Score represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It ranges from 0 to 1, with higher values indicating better model performance."

ttk.Label(root, text=mse\_explanation, wraplength=400).grid(column=1, row=4)

ttk.Label(root, text=r2\_explanation, wraplength=400).grid(column=1, row=5)

root.mainloop()

Text

Description automatically generated

* These lines create the GUI interface using tkinter and ttk.
  + It creates a window with the title "Data Analysis" (root.title), and adds various widgets to the window using ttk.Label, ttk.Entry, ttk.Button, and ttk.Label.
    - Specifically, it creates a label for selecting the CSV file (ttk.Label), an entry box for displaying the file path (ttk.Entry), buttons for browsing the file and analyzing the data (ttk.Button), labels for displaying the evaluation metrics (ttk.Label).
    - It then sets the values of the mse\_explanation and r2\_explanation variables, which contain explanations of the evaluation metrics.
    - Finally, it enters the event loop using root.mainloop(), which waits for the user to interact with the GUI.