Descripción del código

```
import tkinter as tk
import pandas as pd
import numpy as np
from sklearn.metrics import r2_score
from sympy import integrate, symbols, sympify
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import plotly.graph_objs as go
import json
import sympy as sp
from tkinter import simpledialog, filedialog
```

Este es un resumen de las librerías y módulos utilizados en el programa y cómo se utilizan:

- **tkinter**: Se utiliza para crear la interfaz gráfica de usuario del programa. En particular, se importan los módulos **filedialog** y **simpledialog**, que se utilizan para mostrar cuadros de diálogo al usuario para seleccionar archivos y pedir entradas de texto.
- **scikit-learn**: Se importan varios módulos y funciones de este paquete de aprendizaje automático. r2_score se utiliza para medir la bondad de ajuste del modelo de regresión. **LinearRegression** es una clase que implementa un modelo de regresión lineal, y **PolynomialFeatures** es una clase que se utiliza para generar características polinómicas a partir de los datos de entrada. **train_test_split** es una función que se utiliza para dividir los datos en conjuntos de entrenamiento y prueba de forma aleatoria.
- **plotly.graph_objs**: Se utiliza para crear objetos de gráficos interactivos en la biblioteca **Plotly**. Se utilizan diferentes métodos y clases para crear visualizaciones de datos interactivas y personalizadas.
- **sympy**: Se utiliza para realizar cálculos matemáticos simbólicos. En particular, se importan las funciones **integrate**, symbols y **sympify**. **integrate** se utiliza para calcular integrales simbólicas, symbols se utiliza para definir símbolos para variables simbólicas y **sympify** se utiliza para convertir cadenas en expresiones simbólicas.

```
#Colores para la personalizacion
colorFondo = "#71E9F7"
lima = "#67F9A1"
negro = "#000000"
```

```
ventana = tk.Tk()
ventana.title("Calculus Consumption")
ventana.config(bg=colorFondo)
ventana.geometry("1200x720+150+10")
ventana.iconbitmap(r"C:\Users\danie\Downloads\proyecto
estr\Proyecto\icon.ico")
etiqueta_estado = tk.Label(ventana, text="")
etiqueta_estado.pack()
def cargar_archivo():
    global filename
    global nombre_persona
    nombre_persona = simpledialog.askstring("Nombre de persona", "Ingrese
el nombre de la persona:")
    if nombre_persona is not None:
        filename = filedialog.askopenfilename(initialdir='/',
title='Seleccione archivo', filetypes=(('CSV files', '*.csv'),))
        if filename:
            etiqueta_estado.config(text=f"Archivo subido para
{nombre_persona}", fg="green")
        else:
            etiqueta_estado.config(text=f"No se ha subido archivo para
{nombre_persona}", fg="red")
```

Este código tiene como objetivo crear una interfaz gráfica para la carga de archivos CSV. A continuación, se describe paso a paso lo que hace el código:

1. Define tres variables para personalizar los colores de la interfaz gráfica: colorFondo, lima y negro.
2. Crea una ventana de tkinter (tk.Tk()) con el título "Calculus Consumption", el color de fondo definido en la variable colorFondo y una resolución de 1200x720 píxeles y con una ubicación en la pantalla (posición x,y) de (150,10).
3. Establece un ícono para la ventana utilizando el método iconbitmap() y la ruta donde se encuentra el archivo de imagen del ícono.
4. Crea una etiqueta vacía (tk.Label()) para mostrar el estado de la carga del archivo.
5. Empaqueta (pack()) la etiqueta en la ventana.
6. Define una función llamada cargar_archivo() que se ejecuta cuando el usuario hace clic en el botón "Cargar archivo".
7. Dentro de la función cargar_archivo(), se declaran dos variables globales: filename y nombre_persona. filename almacenará la ruta completa del archivo CSV que el usuario cargará, mientras que nombre_persona almacenará el nombre de la persona que cargó el archivo.
8. Se muestra un cuadro de diálogo (simpledialog.askstring()) para que el usuario ingrese el nombre de la persona que cargará el archivo. El valor ingresado se guarda en la variable nombre_persona.

9. Si el valor de nombre_persona es diferente de None, se muestra un cuadro de diálogo (filedialog.askopenfilename()) para que el usuario seleccione el archivo CSV a cargar. El archivo seleccionado se guarda en la variable filename.
10. Si el valor de filename es verdadero (es decir, si el usuario seleccionó un archivo), se actualiza la etiqueta de estado (etiqueta_estado.config()) con un mensaje indicando que el archivo se ha cargado exitosamente para la persona con el nombre nombre_persona y el color de texto verde.
11. Si el valor de filename es falso (es decir, si el usuario no seleccionó un archivo), se actualiza la etiqueta de estado con un mensaje indicando que no se ha cargado ningún archivo para la persona con el nombre nombre_persona y el color de texto rojo.

```
regression_function = None
ecuacion_sympy = None
def graficador(nombre_persona,archivo_csv):
    global fig
    global regression_function
    global ecuacion_sympy
    global filename
    df = pd.read_csv(filename)
    q1 = df['Consumo'].quantile(0.25)
    q3 = df['Consumo'].quantile(0.75)
    iqr = q3 - q1
    outliers = df[(df['Consumo'] < q1 - 1.5*iqr) | (df['Consumo'] > q3 +
1.5*iqr)]
    df = df[(df['Consumo'] >= q1 - 1.5*iqr) & (df['Consumo'] <= q3 +
1.5*iqr)]
```

Este código define una función llamada "graficador" que toma dos argumentos: "nombre_persona" y "archivo_csv". Dentro de la función, se utilizan tres variables globales: "regression_function", "ecuacion_sympy" y "filename". La función comienza leyendo el archivo csv cuyo nombre está en la variable global "filename" utilizando la biblioteca pandas y asignando el resultado a la variable "df".

Luego, se calculan los valores del primer cuartil "q1", el tercer cuartil "q3" y el rango intercuartil "iqr" utilizando el método "quantile()" de pandas. Después, se encuentran los valores atípicos, si los hay, utilizando una expresión booleana que compara los valores de la columna "Consumo" del dataframe "df" con los valores de q1 y q3, más o menos una y media veces el valor de iqr. Los resultados de esta expresión se asignan a la variable "outliers".

Finalmente, se filtran los valores del dataframe "df" utilizando una expresión booleana similar a la anterior pero con la intención de mantener solo los valores que no son atípicos. El nuevo dataframe filtrado se asigna a la variable "df".

En resumen, la función "graficador" carga un archivo csv utilizando pandas, filtra los valores atípicos utilizando los cuartiles y la regla de las 1.5 desviaciones estándar, y luego filtra los valores del dataframe para obtener solo los valores no atípicos.

```
    x = df['Mes'].values
    y = df['Consumo'].values
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=1)

    lin_reg = LinearRegression()
    poly_reg = LinearRegression()
    poly_transform = PolynomialFeatures(degree=2)
    x_train_poly = poly_transform.fit_transform(x_train.reshape(-1, 1))
    x_test_poly = poly_transform.fit_transform(x_test.reshape(-1, 1))
    lin_reg.fit(x_train.reshape(-1, 1), y_train)
    poly_reg.fit(x_train_poly, y_train)
```

Continuando con el código, después de filtrar los valores atípicos y obtener los valores del dataframe para la columna "Mes" en "x" y la columna "Consumo" en "y", se divide el conjunto de datos en dos conjuntos diferentes: uno para entrenamiento ("x_train" y "y_train") y otro para pruebas ("x_test" e "y_test"). Se utiliza la función "train_test_split" de la biblioteca scikit-learn para dividir los datos en un 70% para entrenamiento y un 30% para pruebas. Además, se establece una semilla aleatoria ("random_state=1") para asegurar que los resultados sean reproducibles.

Luego, se inicializa un objeto de la clase "LinearRegression()" de scikit-learn y se lo asigna a la variable "lin_reg". También se inicializa otro objeto de la misma clase y se lo asigna a la variable "poly_reg". Se utiliza la clase "PolynomialFeatures()" para transformar los datos de entrenamiento y prueba a un polinomio de segundo grado. Los datos transformados se asignan a las variables "x_train_poly" y "x_test_poly".

Después de esto, se ajusta un modelo de regresión lineal utilizando los datos de entrenamiento originales ("x_train" e "y_train") utilizando el método "fit()" de la variable "lin_reg". Luego, se ajusta un modelo de regresión polinómica utilizando los datos de entrenamiento transformados ("x_train_poly" e "y_train") utilizando el método "fit()" de la variable "poly_reg".

En resumen, este código divide los datos en conjuntos de entrenamiento y prueba, transforma los datos a un polinomio de segundo grado, y ajusta dos modelos de regresión (uno lineal y otro polinómico) utilizando los datos de entrenamiento.

```
    lin_r2 = r2_score(y_test, lin_reg.predict(x_test.reshape(-1, 1)))
    poly_r2 = r2_score(y_test, poly_reg.predict(x_test_poly))
```

Después de entrenar los modelos de regresión lineal y polinomial, se calculan los coeficientes de determinación R^2 de cada modelo. El coeficiente de determinación es una medida estadística que indica qué tan bien se ajustan los valores predichos por el modelo a los valores reales.

La función "r2_score()" de la biblioteca sklearn se utiliza para calcular el R^2 de cada modelo. Se le pasan dos argumentos: los valores reales (y_test) y los valores predichos por el modelo (que se obtienen aplicando el método "predict()" de cada modelo a los datos de prueba).

Los valores de R^2 se asignan a las variables "lin_r2" y "poly_r2", respectivamente.

```
    if len(outliers) > 0:
        fig = go.Figure([
                    go.Scatter(x=outliers['Mes'], y=outliers['Consumo'],
                            name='outliers', mode='markers',
marker=dict(color='green', size=10)),
                ])
    if lin_r2 > poly_r2:

        y_pred = lin_reg.predict(x.reshape(-1, 1))

        fig = go.Figure([
                    go.Scatter(x=x_train.squeeze(), y=y_train,
                            name='train', mode='markers'),
                    go.Scatter(x=x_test.squeeze(), y=y_test,
                            name='test', mode='markers'),
                    go.Scatter(x=x[~np.isin(x, np.concatenate((x_train,
x_test)))], y=y[~np.isin(x, np.concatenate((x_train, x_test)))],
                            name='outlier', mode='markers',
marker=dict(color='red', size=10)),
                    go.Scatter(x=x[~np.isin(x, np.concatenate((x_train,
x_test, x[~np.isin(x, np.concatenate((x_train, x_test)))])))],
y=y[~np.isin(x, np.concatenate((x_train, x_test, x[~np.isin(x,
np.concatenate((x_train, x_test)))])))],
                            name='other data', mode='markers',
marker=dict(color='green', size=10)),
                    go.Scatter(x=x, y=y_pred.squeeze(),
                            name='prediction')
            ])
        fig.show()
```

Este código está utilizando la biblioteca Plotly para crear gráficos interactivos en línea.

En la primera sección del código, si hay valores atípicos en los datos, se creará una figura que muestra los valores atípicos como puntos verdes en un gráfico de dispersión.

Luego, si el coeficiente de determinación (R2) para el modelo de regresión lineal simple (lin_r2) es mayor que el R2 para el modelo de regresión polinómica (poly_r2), se creará una figura que muestra:

- Los valores de entrenamiento (x_train, y_train) como puntos verdes en un gráfico de dispersión.
- Los valores de prueba (x_test, y_test) como puntos azules en un gráfico de dispersión.
- Los valores atípicos (outliers) como puntos rojos en un gráfico de dispersión.
- Los otros valores de datos (no entrenamiento, no prueba y no atípicos) como puntos verdes en un gráfico de dispersión.
- La línea de predicción del modelo de regresión lineal simple como una línea sólida en un gráfico de línea.
- Las etiquetas, colores y modos de graficado se establecen utilizando los parámetros 'name', 'mode' y 'marker' dentro de los objetos de gráfico 'Scatter' de Plotly.

```
    else:
        # Predecir los valores con el modelo de regresión polinomial
        y_pred =
poly_reg.predict(poly_transform.fit_transform(x.reshape(-1, 1)))

        # Crear la figura con Plotly para visualizar los datos y la
predicción
        x_range = np.linspace(x.min(), x.max(), 100)
        y_range =
poly_reg.predict(poly_transform.fit_transform(x_range.reshape(-1, 1)))

        fig = go.Figure([
                    go.Scatter(x=x_train.squeeze(), y=y_train,
                            name='train', mode='markers'),
                    go.Scatter(x=x_test.squeeze(), y=y_test,
                            name='test', mode='markers'),
                    go.Scatter(x=x[~np.isin(x, np.concatenate((x_train,
x_test)))], y=y[~np.isin(x, np.concatenate((x_train, x_test)))],
                            name='outlier', mode='markers',
marker=dict(color='red', size=10)),
                    go.Scatter(x=x[~np.isin(x, np.concatenate((x_train,
x_test, x[~np.isin(x, np.concatenate((x_train, x_test)))])))],
y=y[~np.isin(x, np.concatenate((x_train, x_test, x[~np.isin(x,
np.concatenate((x_train, x_test)))])))],
```

```
                            name='other data', mode='markers',
marker=dict(color='green', size=10)),
                    go.Scatter(x=x_range, y=y_range.squeeze(),
                            name='prediction')
            ])
        fig.show()
```

En esta sección del código, si el coeficiente de determinación de la regresión lineal simple es menor que el de la regresión polinómica, se predicen los valores utilizando el modelo de regresión polinómica.

Luego se crea una figura con Plotly para visualizar los datos y la predicción. Se utiliza poly_transform.fit_transform() para transformar los datos de entrada x en una matriz de características polinómicas, y luego se utiliza poly_reg.predict() para predecir los valores de salida y.

Se crea una x_range que se utiliza para visualizar la curva de ajuste. Se utilizan diferentes colores y modos de graficado para representar diferentes conjuntos de datos: entrenamiento, prueba, valores atípicos y otros datos.

Finalmente, la figura se muestra con fig.show().

```
   regression_function = " "
   if lin_r2 > poly_r2:
        m, b = np.polyfit(x, y_pred, 1)
        regression_function = f'{m:.2f}*x + {b:.2f}'
    else:
        p4,p3,p2,p1, p0 = np.polyfit(x_range, y_range, 4)
        regression_function = f'{p4:2f}*x**4+{p3:2f}*x**3+{p2:.2f}*x**2 +
{p1:.2f}*x + {p0:.2f}'
    print(f'Regresión: y = {regression_function}')
    ecuacion_sympy = sympify(regression_function)
    extraer()
```

En este fragmento de código se está asignando un valor inicial vacío a la variable regression_function. Luego se realiza una comparación entre los valores de lin_r2 y poly_r2. Si lin_r2 es mayor que poly_r2, entonces se calculan los coeficientes de la regresión lineal a través de np.polyfit con los valores de x e y_pred y se asigna a regression_function la cadena de texto correspondiente. En caso contrario, se calculan los coeficientes de la regresión polinomial de cuarto grado a través de np.polyfit con los

valores de x_range e y_range y se asigna a regression_function la cadena de texto correspondiente.

Luego se imprime la ecuación de la regresión con print. Se utiliza la función sympify del módulo sympy para convertir la cadena de texto de regression_function en una ecuación simbólica, la cual se asigna a la variable ecuacion_sympy.

Finalmente, se llama a la función extraer(), la cual será definida después.

```
    with open(r'C:\Users\danie\Downloads\proyecto estr\ecuaciones.txt',
'r') as archivo:
        contenido_actual = archivo.read()
    if contenido_actual:
        resultados = json.loads(contenido_actual)
    if ecuacion_sympy is not None:
        resultados[nombre_persona] = str(ecuacion_sympy)
        with open(r"C:\Users\danie\Downloads\proyecto
estr\ecuaciones.txt", mode='w') as archivo:
            json.dump(resultados, archivo)
```

Este bloque de código se encarga de almacenar la ecuación obtenida de la regresión en un archivo de texto para su posterior uso.

En primer lugar, se abre el archivo ecuaciones.txt en modo lectura y se lee su contenido actual mediante el método read() y se almacena en la variable contenido_actual.

Luego, se verifica si hay algún contenido en contenido_actual usando una condición if. Si hay contenido, significa que el archivo ecuaciones.txt ya tiene datos guardados, entonces se cargan los datos usando la función json.loads() y se almacenan en la variable resultados.

A continuación, se verifica si ecuacion_sympy no es nulo (es decir, si se ha obtenido una ecuación de la regresión). Si es así, se añade la ecuación a la variable resultados usando como clave el valor de la variable nombre_persona y como valor la cadena de texto de la ecuación obtenida, convertida a una cadena con la función str().

Finalmente, se abre el archivo ecuaciones.txt en modo escritura y se guarda el diccionario resultados actualizado en él mediante la función json.dump(). De esta forma, se asegura que la ecuación obtenida de la regresión se guarda en el archivo de texto para su posterior uso.

```
def extraer():
    global resultados
    resultados ={}
    with open(r'C:\Users\danie\Downloads\proyecto estr\ecuaciones.txt',
'r') as archivo:
        contenido_actual = archivo.read()
```

```
    if contenido_actual:
        resultados = json.loads(contenido_actual)
    return resultados
```

Este es un bloque de código que define la función extraer(). La función se utiliza para extraer el contenido actual de un archivo de texto ecuaciones.txt.

Primero, la función define una variable resultados vacía como un diccionario global. Luego, se abre el archivo en modo lectura y se lee su contenido actual utilizando el método read(). Si el archivo tiene contenido, el contenido se carga en la variable resultados en formato JSON utilizando el método json.loads().

Finalmente, la función devuelve la variable resultados, que contiene el contenido actual del archivo en formato de diccionario Python.

```
def comparar(w):
  global resultados, resultados_evaluacion
  mes = float(w)
  ress = []
  ress.append("Resultados de la evaluación:")
  for nombre_personaa, ecuacion in resultados.items():
      evae = sympify(ecuacion)
      resultado_evaluacion = evae.evalf(subs={'x': mes})
      resultados_evaluacion[nombre_persona] =
round(resultado_evaluacion,2)
      ress.append(f"{nombre_personaa}: {round(resultado_evaluacion)}")
  valores = list(resultados_evaluacion.values())
  max_consumo = max(resultados_evaluacion, key=resultados_evaluacion.get)
  min_consumo = min(resultados_evaluacion, key=resultados_evaluacion.get)
  ress.append(f"\nEl consumo pr hola soy un cometario hecho por william
omedio en el mes {mes} es de: {round(np.mean(valores),2)} kwt")
  ress.append(f"La persona que consumió más en el mes {mes} es
{max_consumo} con {round(resultados_evaluacion[max_consumo],2)} kwt")
  ress.append(f"La persona que consumió menos en el mes {mes} es
{min_consumo} con {round(resultados_evaluacion[min_consumo],2)} kwt")
  return "\n".join(ress)
```

Esta función comparar(w) tiene como objetivo comparar y evaluar las ecuaciones de consumo eléctrico de diferentes personas en un mes específico, y determinar quién consumió más y quién consumió menos en ese mes.

Primero, se lee el archivo ecuaciones.txt y se carga su contenido en la variable resultados, que es un diccionario donde las claves son los nombres de las personas y los valores son las ecuaciones de consumo eléctrico correspondientes.

Luego, se itera a través de cada nombre de persona y su ecuación correspondiente en el diccionario resultados, y se utiliza la función sympify para convertir la ecuación de texto a una expresión simbólica de SymPy. Se evalúa la expresión simbólica de cada persona con un valor mes específico (representando el mes a evaluar), y se redondea el resultado a 2 decimales.

El resultado de la evaluación para cada persona se almacena en un nuevo diccionario llamado resultados_evaluacion, donde las claves son los nombres de las personas y los valores son los resultados de la evaluación de sus ecuaciones de consumo eléctrico en el mes específico.

Luego se realiza una serie de operaciones en el diccionario resultados_evaluacion para determinar quién consumió más y quién consumió menos en el mes específico. Se calcula la media de los resultados de evaluación, se obtiene el nombre de la persona con el valor más alto y el valor más bajo, y se almacena todo en la lista ress.

Finalmente, se devuelve una cadena de texto con los resultados de la evaluación y las comparaciones.

```
def mesdet(mes):
  global nombre_persona
  global ecuacion_sympy
  ecuacion = ecuacion_sympy
  x = symbols('x')
  expr = sympify(ecuacion)
  valor_evaluar = float(mes)
  resultado_evaluacion = expr.evalf(subs={x: valor_evaluar})
  resultado = f"El consumo de  {nombre_persona} en el mes {valor_evaluar}
es: {round(resultado_evaluacion)} kwt "
  print(resultado)
  return resultado
```

La función mesdet(mes) se encarga de evaluar la ecuación de consumo de energía de una persona en un mes específico. Toma como entrada el mes que se quiere evaluar en formato numérico.

Primero, la función asigna a las variables globales nombre_persona y ecuacion_sympy los valores correspondientes. Luego, se utiliza la librería sympy para convertir la cadena de la ecuación a una expresión simbólica expr.

A continuación, se evalúa la expresión simbólica con el valor del mes a través de la función evalf() y se redondea el resultado a un entero con la función round(). El resultado se almacena en la variable resultado_evaluacion.

Finalmente, se construye una cadena de texto que incluye el nombre de la persona evaluada, el mes y el consumo de energía redondeado. Esta cadena se retorna.

```
def variosmes(p,z):
  global ecuacion_sympy
  global regression_function
  x = symbols('x')
  menor = float(p)
  mayor = float(z)
  integral = integrate(ecuacion_sympy, (x, menor, mayor))
  prom_consu = round(integral*(1/(z-p)))
  total_a_pagar = round(374.24*integral)
  total_a_pagar_str = "{:,.2f}".format(float(total_a_pagar))
  resultado = [
      f'Tendencia: y = {regression_function}',
      f'El total de consumo en estos se espera que sea de:
{round(integral)} kwt',
      f"El total a pagar por {round(integral)} kwt es de
{total_a_pagar_str} pesos",
      f"El consumo promedio se espera que sea de {prom_consu} kwt en esos
meses"
  ]
  return "\n".join(resultado)
```

Esta función toma dos parámetros, p y z, que son los valores del mes menor y mayor respectivamente para los cuales se desea calcular el consumo de energía eléctrica. La función utiliza la ecuación de regresión polinómica previamente calculada ecuacion_sympy y la función de regresión regression_function para realizar los cálculos.

Primero, se convierten p y z en números de punto flotante y se utiliza la función integrate de la biblioteca sympy para calcular la integral de la ecuación de regresión polinómica en el rango de valores de p a z. El resultado se redondea y se almacena en la variable integral.

A continuación, se utiliza el valor de integral para calcular el total a pagar por el consumo de energía eléctrica en los meses dados. El cálculo se realiza multiplicando integral por una tarifa de energía eléctrica de 374.24 pesos por kilovatio hora. El resultado se redondea y se formatea como una cadena de caracteres con dos decimales y comas separando los miles.

Después, se calcula el consumo promedio esperado en los meses dados. Esto se hace dividiendo el valor de integral por la diferencia entre z y p y redondeando el resultado.

Por último, se crea una lista de cadenas de caracteres que resumen los resultados y se devuelve la lista como una cadena de caracteres separada por saltos de línea mediante la función join.

```
def pruebader(a,b):
  global ecuacion_sympy
  x = sp.Symbol('x')
  inf = int(a)
  sup = int(b)
  derivada = sp.diff(ecuacion_sympy, x)
  primera_derivada_en_a = derivada.subs(x, inf)
  primera_derivada_en_b = derivada.subs(x, sup)

  if primera_derivada_en_a > 0 and primera_derivada_en_b > 0:
      resultado = f"El consumo de energía aumenta en el intervalode meses
desde el mes {inf} a {sup}"
      resultado = f"El consumo de energía aumenta en el intervalo "
  elif primera_derivada_en_a < 0 and primera_derivada_en_b < 0:
      resultado = f"El consumo de energía disminuye en el intervalo de
meses desde el mes {inf} a {sup}"
  elif primera_derivada_en_a == 0 and primera_derivada_en_b == 0:
      resultado = f"El consumo de energía se mantiene constante en el
intervalo de meses desde el mes {inf} a {sup}"
  else:
      resultado = f"El consumo de energía varía en esos meses, es decir
no tiene una tendencia marcada"
  return resultado
```

La función pruebader(a,b) toma dos argumentos a y b, que representan los límites inferiores y superiores de un intervalo de meses respectivamente. La función calcula la primera derivada de la ecuación de consumo de energía en función del tiempo, en los puntos a y b. Si la primera derivada es positiva en ambos puntos, la función devuelve un mensaje que indica que el consumo de energía aumenta en el intervalo de meses desde el mes a a b. Si la primera derivada es negativa en ambos puntos, la función devuelve un mensaje que indica que el consumo de energía disminuye en el intervalo de meses desde el mes a a b. Si la primera derivada es cero en ambos puntos, la función devuelve un mensaje que indica que el consumo de energía se mantiene constante en el intervalo de meses desde el mes a a b. Si la primera derivada es positiva en un punto y negativa en el otro, la función devuelve un mensaje que indica que el consumo de energía varía en esos meses, es decir, no tiene una tendencia marcada. El resultado se almacena en la variable resultado y se devuelve al final de la función.

```
def nuevomes(mess):
  global filename
  df = pd.read_csv(filename)
  ultimo_mes = df["Mes"].iloc[-1]
  nuevo_mes = ultimo_mes + 1
  nuevo_consumo = float(mess)
  nueva_fila = pd.DataFrame({'Mes': [nuevo_mes], 'Consumo':
[nuevo_consumo]})
  df = pd.concat([df, nueva_fila], ignore_index=True)
```

```
    df.to_csv(filename, index=False)
    resul = "El archivo ha sido actuañlizado con exito"
    return resul
```

La función nuevomes recibe un parámetro mess que se refiere al consumo de energía de un nuevo mes que se quiere agregar al archivo de datos. La función comienza leyendo un archivo CSV (cuyo nombre está almacenado en la variable global filename) utilizando la biblioteca Pandas de Python.

Luego, la función encuentra el último mes registrado en el archivo, lo que le permite agregar el nuevo mes en secuencia. A continuación, convierte el parámetro mess en un número de punto flotante y crea una nueva fila de datos que contiene el mes y su consumo correspondiente.

Luego, se concatena la nueva fila con el dataframe existente utilizando la función pd.concat() de Pandas, que agrega la nueva fila a la parte inferior del dataframe. El parámetro ignore_index=True indica que Pandas debe ignorar los índices existentes y crear nuevos índices para las filas concatenadas.

Finalmente, la función guarda el archivo actualizado en disco usando el método to_csv() de Pandas y devuelve una cadena de texto que indica que el archivo ha sido actualizado con éxito.

```
boton_cargar_archivo = tk.Button(ventana, text="Cargar archivo",
relief='ridge', overrelief="raised",
                            bg="white", font="gregoria", cursor="hand2",
borderwidth=2, command=cargar_archivo)
boton_cargar_archivo.pack()

campo_salida = tk.Text(ventana, width=50, height=10)
campo_salida.pack()


def limipiar_campos():
    campo_entrada_min.delete(0, tk.END)
    campo_entrada_max.delete(0, tk.END)
    mess.delete(0,tk.END)
    campo_salida.delete('1.0', tk.END)

def borrar_texto(evento):
    if evento.widget.get()  == evento.widget.get():
       evento.widget.delete(0, tk.END)

def poner_texto(evento):
    if evento.widget.get() == "":
        evento.widget.insert(0, evento.widget.campo)
```

Esta parte del código corresponde a la interfaz gráfica de usuario de la aplicación. Primero, se crea un botón llamado "boton_cargar_archivo" con el texto "Cargar archivo", que al hacer clic en él llama a la función "cargar_archivo".

Luego, se crea un campo de texto llamado "campo_salida" con una anchura de 50 y una altura de 10, que se utiliza para mostrar el resultado de las operaciones realizadas por la aplicación.

A continuación, se define una función llamada "limpiar_campos" que se utiliza para borrar los campos de entrada de la interfaz gráfica.

Después, se definen dos funciones más: "borrar_texto" y "poner_texto". La primera función se utiliza para borrar el texto que se encuentra en un campo de entrada cuando se hace clic en él, y la segunda función se utiliza para escribir un texto predeterminado en un campo de entrada cuando está vacío.

```
mess = tk.Entry(ventana, width=13, justify="center", font="gregoria, 11",
fg="green")
mess.campo = "Ingrese un mes"
mess.insert(0, mess.campo)
mess.bind("<FocusIn>", borrar_texto)
mess.bind("<FocusOut>", poner_texto)
mess.pack(pady=10)
campo_entrada_min = tk.Entry(ventana, width=22, justify="center",
font="gregoria, 12", fg="red")
campo_entrada_min.campo = "Primer mes o limite inferior"
campo_entrada_min.insert(0, campo_entrada_min.campo)
campo_entrada_min.bind("<FocusIn>", borrar_texto)
campo_entrada_min.bind("<FocusOut>", poner_texto)
campo_entrada_min.pack(pady=4)
campo_entrada_max = tk.Entry(ventana, width=22, justify="center",
font="gregoria, 12", fg="red")
campo_entrada_max.campo = "Ultimo mes o limite superior"
campo_entrada_max.insert(0, campo_entrada_max.campo)
campo_entrada_max.bind("<FocusIn>", borrar_texto)
campo_entrada_max.bind("<FocusOut>", poner_texto)
campo_entrada_max.pack(pady=4)
```

Se están creando tres campos de entrada en la ventana principal de la interfaz gráfica. El primero es un widget Entry llamado mess, que se utiliza para ingresar el consumo de energía de un nuevo mes. Se establece una longitud de 13 caracteres y se justifica al centro. También se establece una fuente gregoria de tamaño 11 y un color de texto verde. El texto predeterminado en este campo es "Ingrese un mes". Para borrar el texto predeterminado cuando el usuario hace clic en el campo, se utiliza la función borrar_texto() que se define

en el código anterior. De manera similar, cuando el campo pierde el enfoque, se utiliza la función poner_texto() para restablecer el texto predeterminado si no se ha ingresado nada.

Los siguientes dos campos de entrada son widgets Entry llamados campo_entrada_min y campo_entrada_max, que se utilizan para ingresar los límites inferior y superior del intervalo de tiempo para el que se desea realizar el análisis. Ambos campos tienen una longitud de 22 caracteres y se justifican al centro. Se establece la misma fuente gregoria de tamaño 12 y color de texto rojo para ambos campos. El texto predeterminado para el campo campo_entrada_min es "Primer mes o limite inferior" y el texto predeterminado para el campo campo_entrada_max es "Ultimo mes o limite superior". Se utilizan las mismas funciones borrar_texto() y poner_texto() para manejar la eliminación y restablecimiento del texto predeterminado en ambos campos.

```
def describe():
    df = pd.read_csv(filename)
    descripcion =
df.describe().style.background_gradient(cmap='Blues').to_string()
    campo_salida.delete("1.0",tk.END)
    campo_salida.insert("1.0", descripcion)
    return campo_salida

def opcion1():
    describe()

def opcion2():
    global boton_opcion2
    boton_opcion2 = True
    graficador(nombre_persona,filename)
def calcular_variosmes():
    if boton_opcion2:
        minimo = float(campo_entrada_min.get())
        maximo = float(campo_entrada_max.get())
        resultado = variosmes(minimo, maximo)
        campo_salida.delete('1.0', tk.END)
        campo_salida.insert('1.0', resultado)
    else:
        campo_salida.delete('1.0', tk.END)
        campo_salida.insert('1.0', "Debes presionar el botón de graficar
funciones primero.")
def calcular__un_mes():
    if boton_opcion2:
        valor = float(mess.get())
        resultado = mesdet(valor)
        campo_salida.delete("1.0", tk.END)
        campo_salida.insert("1.0", resultado)
    else:
        campo_salida.delete('1.0', tk.END)
        campo_salida.insert('1.0', "Debes presionar el boton de graficar
funciones primero.")
```

```
def usuacompa():
    if boton_opcion2:
        detmes = float(mess.get())
        com = comparar(detmes)
        campo_salida.delete("1.0", tk.END)
        campo_salida.insert("1.0", com)
    else:
        campo_salida.delete('1.0', tk.END)
        campo_salida.insert('1.0', "Debes presionar el boton de graficar
funciones primero.")

def concA():
    if boton_opcion2:
        inf = float(campo_entrada_min.get())
        sup = float(campo_entrada_max.get())
        prue = pruebader(inf, sup)
        campo_salida.delete("1.0", tk.END)
        campo_salida.insert("1.0", prue)
    else:
        campo_salida.delete('1.0', tk.END)
        campo_salida.insert('1.0', "Debes presionar el boton de graficar
funciones primero.")
def añadirmes():
    if boton_opcion2:
        mese = float(mess.get())
        nue = nuevomes(mese)
        campo_salida.delete("1.0", tk.END)
        campo_salida.insert("1.0", nue)
    else:
        campo_salida.delete('1.0', tk.END)
        campo_salida.insert('1.0', "Debes presionar el boton de graficar
funciones primero.")
```

Estas son las funciones que se ejecutan cuando el usuario hace clic en diferentes botones de la interfaz gráfica:

1. describe(): esta función lee el archivo cargado y genera una descripción estadística del conjunto de datos usando el método describe() de Pandas. Luego, utiliza la función style.background_gradient() para aplicar un gradiente de color a la tabla y convierte la descripción en una cadena de texto que se muestra en el campo de salida de la interfaz.
2. opcion1(): esta función simplemente llama a la función describe() cuando el usuario hace clic en el botón "Descripcion de datos" de la interfaz.
3. opcion2(): esta función establece una variable global boton_opcion2 en True y llama a la función graficador() con el nombre de archivo y el nombre de la persona seleccionados por el usuario. La variable boton_opcion2 se utiliza en otras

funciones para verificar si el usuario ha ejecutado previamente la función graficador() antes de realizar ciertas operaciones.

4. calcular_variosmes(): esta función verifica si el usuario ha ejecutado previamente la función graficador() al verificar el valor de la variable global boton_opcion2. Si se ha ejecutado graficador(), se obtienen los valores de los límites inferior y superior ingresados por el usuario y se llama a la función variosmes() con estos valores. El resultado se muestra en el campo de salida de la interfaz.
5. calcular__un_mes(): esta función verifica si el usuario ha ejecutado previamente la función graficador() al verificar el valor de la variable global boton_opcion2. Si se ha ejecutado graficador(), se obtiene el valor del mes ingresado por el usuario y se llama a la función mesdet() con este valor. El resultado se muestra en el campo de salida de la interfaz.
6. usuacompa(): esta función verifica si el usuario ha ejecutado previamente la función graficador() al verificar el valor de la variable global boton_opcion2. Si se ha ejecutado graficador(), se obtiene el valor del mes ingresado por el usuario y se llama a la función comparar() con este valor. El resultado se muestra en el campo de salida de la interfaz.
7. concA(): esta función verifica si el usuario ha ejecutado previamente la función graficador() al verificar el valor de la variable global boton_opcion2. Si se ha ejecutado graficador(), se obtienen los valores de los límites inferior y superior ingresados por el usuario y se llama a la función pruebader() con estos valores. El resultado se muestra en el campo de salida de la interfaz.
8. añadirmes(): esta función verifica si el usuario ha ejecutado previamente la función graficador() al verificar el valor de la variable global boton_opcion2. Si se ha ejecutado graficador(), se obtiene el valor del mes ingresado por el usuario y se llama a la función nuevomes() con este valor. El resultado se muestra en el campo de salida de la interfaz.

```
boton_opcion1 = tk.Button(ventana, text="1. Descripción de datos",
relief='ridge', overrelief="raised",
                          bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                          cursor="hand2", borderwidth=2, command=opcion1)
boton_opcion1.pack(pady=4)
boton_opcion1.pack()
boton_opcion2 = tk.Button(ventana, text="2. Gráfico de relación de
datos", relief='ridge', overrelief="raised",
                          bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                          cursor="hand2", borderwidth=2, command=opcion2)
boton_opcion2.pack(pady=4)
boton_opcion2.pack()
boton_opcion2 = False
```

```
boton3 = tk.Button(ventana, text="3. Consumo de energia en varios meses",
relief='ridge', overrelief="raised",
                           bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                           cursor="hand2", borderwidth=2,
command=calcular_variosmes)
boton3.pack()
boton4 = tk.Button(ventana, text="4. Consumo de energía en un mes
determinado", relief='ridge', overrelief="raised",
                           bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                           cursor="hand2", borderwidth=2,
command=calcular__un_mes)
boton4.pack(pady=4)
boton5 = tk.Button(ventana, text="5. Comparar consumo de energía con
otros usuarios", relief='ridge', overrelief="raised",
                           bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                           cursor="hand2", borderwidth=2,
command=usuacompa)
boton5.pack(pady=4)
boton6 = tk.Button(ventana, text="6. Análisis de intervalos de consumo",
relief='ridge', overrelief="raised",
                           bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                           cursor="hand2", borderwidth=2, command=concA)
boton6.pack(pady=4)
boton7 = tk.Button(ventana, text="7. Añadir información de un nuevo mes",
relief='ridge', overrelief="raised",
                           bg="white", font="gregoria",
activebackground=lima, activeforeground=negro,
                           cursor="hand2", borderwidth=2,
command=añadirmes)
boton7.pack(pady=4)
boton_salir = tk.Button(ventana, text="Salir", relief='ridge',
overrelief="raised",
                           bg="white", font="gregoria",
activebackground="red", activeforeground=negro,
                           cursor="hand2", borderwidth=2,
command=ventana.quit)


# Añadir botones a la ventana
boton_salir.pack()

ventana.mainloop()
```

Este fragmento del código crea una interfaz gráfica de usuario que incluye varios botones con diferentes opciones para analizar los datos de consumo de energía. Cada botón se crea con la función tk.Button() y se configura con diferentes parámetros como el texto que

muestra, el tipo de relieve, el color de fondo, la fuente, el color activo, el cursor, el ancho del borde y la función que se ejecutará al hacer clic en el botón. Los botones se empaquetan en la ventana con la función pack() para que aparezcan en la pantalla. El último botón es un botón de salida que cerrará la ventana cuando se haga clic en él. La ventana se muestra con la función mainloop().