

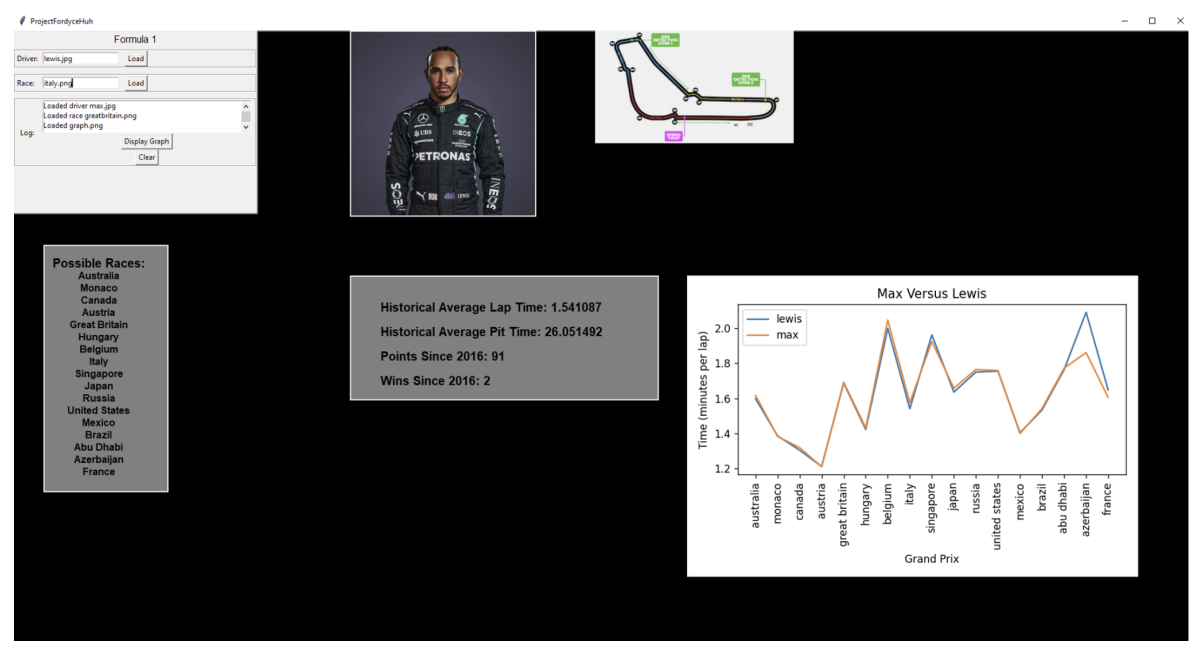
Joe Fordyce  
Jimin Huh  
BAIS:3020 Computational Thinking  
Professor Yvonne Galusha  
May 7, 2021

## Formula 1 Max Verstappen Lewis Hamilton Python Analysis

### Overall Functionality

- ❑ Our program looks at two of the top drivers in Formula 1 right now; Lewis Hamilton and Max Verstappen.
- ❑ The program does a race by race analysis of the two drivers over the last 5 years.
- ❑ This analysis includes lap times, pit stop times, points, and wins for each track in the last 5 years.
- ❑ Our goal was to determine who the faster driver is at different tracks.

### Visualization



## Graphical User Interface

**Main Canvas** - the overall GUI creates a black canvas that fits the screen with a menu box at the left top where a user can enter a driver's name and a specific race.

```
import PIL.Image
import PIL.ImageTk
from tkinter import *
from Utilities import *

class AllTkinterWidgets:
    def __init__(self, master):
        self.lewis_averages = {}
        self.max_averages = {}
        self.lewis_pit = {}
        self.max_pit = {}
        self.canvas = Canvas(master, width = root.winfo_screenwidth(), height = root.winfo_screenheight(), bd = 2)
        self.canvas.config(background = 'black')

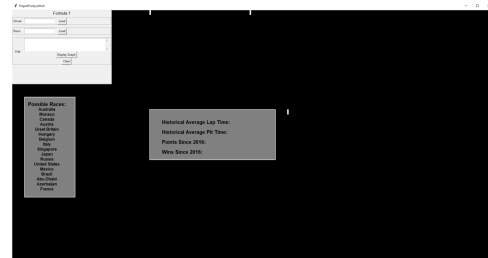
# ----- Main GUI Frame -----
self.mbar = Frame(self.canvas, relief = 'raised', width=500, bd = 2)
self.mbar.place(height = 300, width = 480)
```

```
# ----- Entry Boxes for Driver and Race, Load Buttons, Label Placements -----
self.t = StringVar()
self.t2 = StringVar()
self.ef = Frame(self.canvas, bd=2, relief='groove')
self.lb2 = Label(self.ef, text='Race: ')
self.lb2.pack(side=LEFT)
self.entry = Entry(self.ef, textvariable = self.t, bg='white')
self.bt = Button(self.ef, text = 'Load', command = self.load_race)
self.entry.pack(side = LEFT, padx = 5)
self.bt.pack(side = LEFT, padx=5)
self.ef.pack(expand=0, fill=X, pady=5, after = self.l, anchor = W)

self.l1 = Label().place(x=550,y=0)
self.l2 = Label().place(x=950,y=0)
self.l3 = Label().place(x=1100,y=400)

self.ef2 = Frame(self.canvas, bd=2, relief = 'groove')
self.lb3 = Label(self.ef2, text='Driver:')
self.lb3.pack(side=LEFT)
self.entry2 = Entry(self.ef2, textvariable=self.t2, bg='white')
self.bt3 = Button(self.ef2, text = 'Load', command = self.load_driver)
self.entry2.pack(side=LEFT, padx=5)
self.bt3.pack(side=LEFT, padx=5)
self.ef2.pack(expand=0, fill=X, pady=5, after = self.l, anchor = W)

# ----- Listbox frame, Clear Button and Display Graph Button -----
self.lf = Frame(self.canvas, bd=2, relief='groove')
self.lb = Label(self.lf, text='Log:')
self.bt1 = Button(self.lf, text = 'Display Graph', command = lambda: constructGraph(self))
self.bt2 = Button(self.lf, text = 'Clear', command = self.clear)
self.listbox = Listbox(self.lf, height=4)
self.sbl = Scrollbar(self.listbox, orient=VERTICAL, command = self.listbox.yview)
self.listbox.config(scrollcommand=self.sbl.set)
self.lb.pack(side=LEFT, padx=5)
self.bt2.pack(side = BOTTOM)
self.bt1.pack(side = BOTTOM)
self.sbl.pack(side=RIGHT, fill=Y)
self.listbox.pack(padx=5, fill = X)
self.lf.pack(expand=0, fill=X, pady=5, after = self.ef, anchor = W)
```



**Menu Box** - contains two entry boxes for a driver and a race. 'Load' buttons contain functions to load a specific picture of the driver and the race. The log area shows the actions a user did (e.g. Loaded driver max.jpg) 'Display Graph' button displays a graph of the two drivers' performances. 'Clear' button clears the images we loaded, statistics, as well as log data.

```
# ----- Formula 1 Title Creation -----
self.text = Text(self.mbar, height = 5, width =52)
self.l = Label(self.mbar, text = 'Formula 1')
self.l.config(font=('Helvetica',12))
self.l.pack(side = TOP)
```

Formula 1	
Driver:	<input type="text"/> <input type="button" value="Load"/>
Race:	<input type="text"/> <input type="button" value="Load"/>
<input type="text"/>	
Log:	<input type="button" value="Display Graph"/>
	<input type="button" value="Clear"/>

**Possible Races List** - shows possible races a user can enter. The codes create a canvas and place names of the races inside the canvas.

```
#----- Entry Boxes for Driver and Race, Load Buttons, Label Placements -----
self.t = StringVar()
self.t2 = StringVar()
self.ef = Frame(self.canvas, bd=2, relief='groove')
self.lb2 = Label(self.ef, text='Race: ')
self.lb2.pack(side= LEFT)
self.entry = Entry(self.ef, textvariable = self.t, bg='white')
self.bt = Button(self.ef, text = 'Load', command = self.load_race)
self.entry.pack(side = LEFT, padx = 5)
self.bt.pack(side = LEFT, padx= 5)
self.ef.pack(expand=0, fill=X, pady=5, after = self.l, anchor = W)

self.l1 = Label().place(x=550,y=0)
self.l2 = Label().place(x=950,y=0)
self.l3 = Label().place(x=1100,y=400)

self.ef2 = Frame(self.canvas, bd=2, relief = 'groove')
self.lb3 = Label(self.ef2, text='Driver:')
self.lb3.pack(side=LEFT)
self.entry2 = Entry(self.ef2,textvariable=self.t2, bg='white')

self.bt3 = Button(self.ef2, text = 'Load', command = self.load_driver)
self.entry2.pack(side=LEFT, padx=5)
self.bt3.pack(side=LEFT, padx=5)
self.ef2.pack(expand=0, fill=X, pady=5, after = self.l, anchor = W)

#----- Listbox Frame, Clear Button and Display Graph Button-----
self.lf = Frame(self.canvas, bd=2, relief='groove')
self.lb = Label(self.lf, text='Log:')
self.bt1 = Button(self.lf, text = 'Display Graph', command = lambda: constructGraph(self))
self.bt2 = Button(self.lf, text = 'Clear', command = self.clear)
self.listbox = Listbox(self.lf, height=4)
self.sbl = Scrollbar(self.listbox, orient=VERTICAL, command= self.listbox.yview)
self.listbox.configure(yscrollcommand=self.sbl.set)
self.lb.pack(side=LEFT, padx=5)
self.bt2.pack(side = BOTTOM)
self.lf.pack(side = BOTTOM)
self.sbl.pack(side=RIGHT, fill=Y)
self.listbox.pack(padx=5, fill = X)
self.lf.pack(expand=0, fill=X, pady=5, after = self.ef, anchor = W)
```

#### Possible Races:

Australia  
Monaco  
Canada  
Austria  
Great Britain  
Hungary  
Belgium  
Italy  
Singapore  
Japan  
Russia  
United States  
Mexico  
Brazil  
Abu Dhabi  
Azerbaijan  
France

**Historical Statistics Display** - contains texts of historical average lap time, pit time, points, and wins. The codes, just like Possible Races List, creates a canvas and place information there.

```
#----- Historical Statistic Display Area Creation-----
self.canvas2 = Canvas(master, width = 500, height = 200)
self.canvas2.place(x=550, y=400)
self.canvas2.configure(background = 'grey')
self.canvas2.create_text(50,40, text = 'Historical Average Lap Time:', fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
self.canvas2.create_text(50,80, text = 'Historical Average Pit Time:', fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
self.canvas2.create_text(50,120, text = 'Points Since 2016: ', fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
self.canvas2.create_text(50,160, text = 'Wins Since 2016: ', fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
```

**Historical Average Lap Time:**

**Historical Average Pit Time:**

**Points Since 2016:**

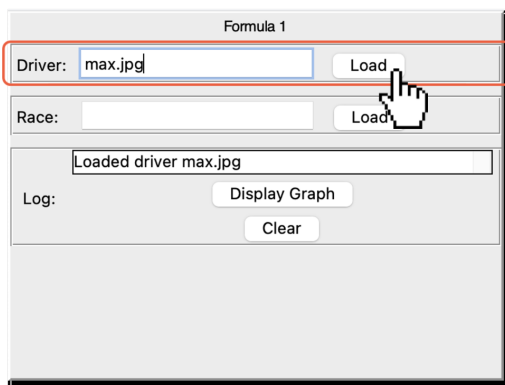
**Wins Since 2016:**

## Main Functions

these are the functions that actually interact with the GUI and make changes on the screen.

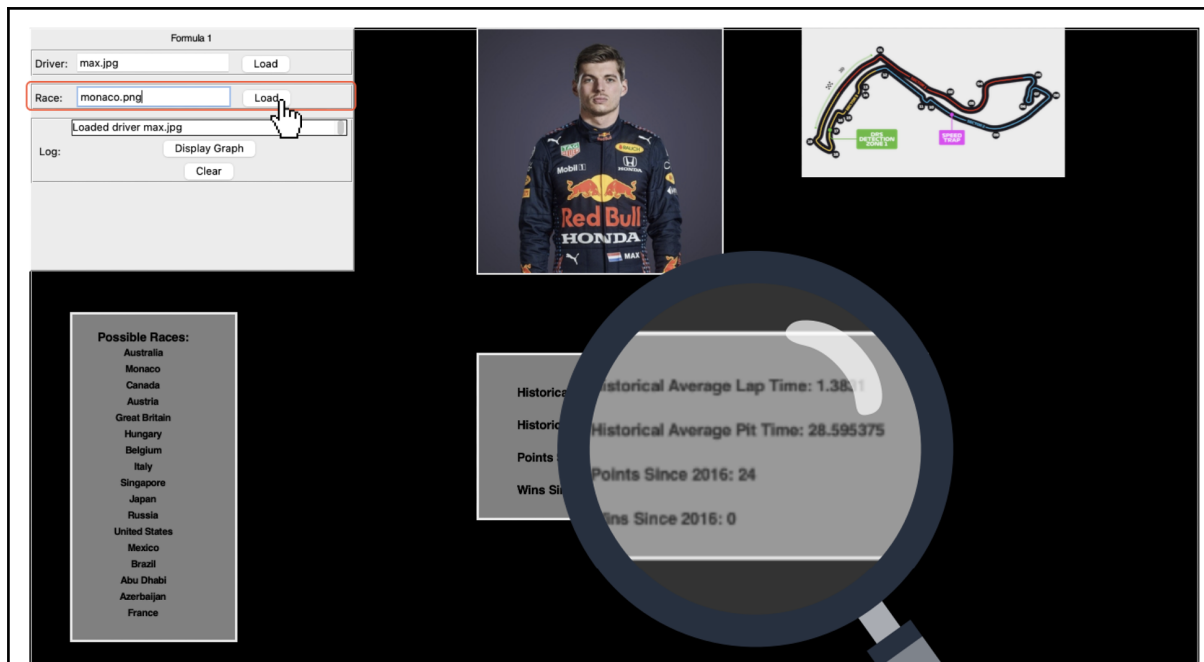
**load\_driver** - loads and displays the driver image onto the screen and loads that driver's data in the back end of the program

```
def load_driver(self):
    '''Loads driver image and loads race and pit data in the back end'''
    try:
        self.updateTimes()
        getDriverData(self)
        getTeamData(self)
        self.listbox.insert(END, 'Loaded driver ' + self.t2.get())
        self.fp = open(self.t2.get(), 'rb')
        self.img = PIL.Image.open(self.fp)
        self.photo = PIL.ImageTk.PhotoImage(self.img)
        self.l1 = Label(image=self.photo)
        self.l1.place(x=550,y=0)
    except:
        self.listbox.insert(END, 'Error Loading driver ' + self.t2.get())
```



**load\_race** - loads and displays the racetrack image and displays the loaded data from the load\_driver function in the statistics display area for that specific race.

```
def load_race(self):
    '''Loads race image and displays loaded data in statistics display area for given race'''
    try:
        self.updateTimes()
        performance = getPerformances(self)
        race = getRaceTime(self)
        pit = getPitTime(self)
        self.updateTimes(LapTime=str(race), Points=str(performance['points']), Wins=str(performance['wins']), PitTime=str(pit))
        self.listbox.insert(END, 'Loaded race ' + self.t.get())
        self.fp = open(self.t.get(), 'rb')
        self.img = PIL.Image.open(self.fp)
        self.photo1 = PIL.ImageTk.PhotoImage(self.img)
        self.l2 = Label(image=self.photo1)
        self.l2.place(x=950,y=0)
    except:
        self.listbox.insert(END, 'Error Loading race ' + self.t.get())
```



**updateTimes** - clears the statistics display area and populates the area with the statistics for the selected driver and selected race. (statistics being the average lap time, average pit time, wins, and points)

```
def updateTimes(self, LapTime='', PitTime='', Points='', Wins=''):
    '''Populates statistic display area'''
    self.canvas2.delete('all')
    self.canvas2.create_text(50, 40, text = 'Historical Average Lap Time: '+LapTime, fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
    self.canvas2.create_text(50, 80, text = 'Historical Average Pit Time: '+PitTime, fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
    self.canvas2.create_text(50, 120, text = 'Points Since 2016: '+Points, fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
    self.canvas2.create_text(50, 160, text = 'Wins Since 2016: '+Wins, fill = 'black', font = 'Helvetica 15 bold', anchor = NW)
```

**Clear** - clears our listbox log, our driver photo, racetrack photo, and the statistics display area; we recommend that the user clears after each driver/race combination is loaded, however the program will still function properly if the user loads drivers and racetracks over one another. We used labels to display and clear our images.

```
# ----- FUNCTION GETS -----
def clear(self):
    '''Clears driver's photo, racetrack photo, and resets statistic display area'''
    self.updateTimes()
    self.listbox.delete(0, END)
    self.l1.destroy()
    self.l2.destroy()
```

## Utilities Functions

these functions are more of the 'back end' functions that work with the data using pandas and matplotlib.

**loadData** - loads the given driver's lap data by reading the respective csv file and stores all of the racetrack lap averages in a dictionary, averages {}

```
def loadData(self,driver):  
    '''  
    Loads the given driver's race data  
    Parameters: driver : String  
    Returns: averages : Dictionary  
    '''  
    df = ''  
    if(driver == 'Lewis'):  
        df = pd.read_csv('LewisTest.csv')  
    else:  
        df = pd.read_csv('MaxTest.csv')  
    df.loc['Average'] = df.mean(skipna=True)  
    averages = {}  
    for col in range(len(df.columns)):  
        name = list(df.columns)[col]  
        race_avg = df.at['Average', name]  
        if('Unnamed' not in name):  
            averages[name.lower()] = race_avg  
    return averages
```

**getDriverData** - this function simply calls the loadData function twice, once for Max and once for Lewis, and stores the data in a new dictionary that is in self.

```
def getDriverData(self):  
    '''Stores race data in self'''  
    self.lewis_averages = loadData(self,'Lewis')  
    self.max_averages = loadData(self,'max')
```

**getRaceTime** - this function retrieves the average lap time (in minutes) for the track specified by the user;

```
def getRaceTime(self):  
    '''  
    Retrieves specified lap time for given track  
    Returns: Average lap time in minutes  
    '''  
    current_driver = self.t2.get()  
    driver_times = {}  
    if(current_driver == 'Lewis.jpg'):  
        driver_times = self.lewis_averages  
    else:  
        driver_times = self.max_averages  
  
    current_race = self.t.get().replace('.png', '')  
    if(current_race == 'greatbritain'):  
        current_race = 'great britain'  
    elif(current_race == 'abudhabi'):  
        current_race = 'abu dhabi'  
    elif(current_race == 'unitedstates'):  
        current_race = 'united states'  
  
    return round(driver_times[current_race],6)
```

**getTeamData** - similar to the loadData function, this function loads and stores the pit data in self; it stores these averages in two dictionaries, one for Max and one for Lewis.

```
def getTeamData(self):
    '''Loads and Stores pit data in self'''
    current_driver = self.t2.get()
    df = ''
    if(current_driver == 'Lewis.jpg'):
        df = pd.read_csv('LewisPit.csv')
    else:
        df = pd.read_csv('MaxPit.csv')

    df.loc['Average'] = df.mean(skipna=True)
    pit_times = {}

    for col in range(len(df.columns)):
        name1 = list(df.columns)[col]
        pit_avg = df.at['Average', name1]
        if('Unnamed' not in name1):
            pit_times[name1.lower()] = pit_avg

    if(current_driver == 'Lewis.jpg'):
        self.lewis_pit = pit_times
    else:
        self.max_pit = pit_times
```

**getPitTime** - similar to the getRaceTime function, the getPitTime function retrieves the average time spent in the pit lane for the racetrack specified by the user.

```
def getPitTime(self):
    '''
    Retrieves specified pit time for given track
    Returns: Average pit time in seconds
    '''
    current_driver = self.t2.get()
    pit_times = {}
    if(current_driver == 'Lewis.jpg'):
        pit_times = self.lewis_pit
    else:
        pit_times = self.max_pit

    current_race = self.t.get().replace('.png', '')
    if(current_race == 'greatbritain'):
        current_race = 'great britain'
    elif(current_race == 'abudhabi'):
        current_race = 'abu dhabi'
    elif(current_race == 'unitedstates'):
        current_race = 'united states'

    return round(pit_times[current_race],6)
```



**getPerformances** - getPerformances retrieves the points and wins for the specified driver and racetrack since 2016, it returns this data in a dictionary.

```
def getPerformances(self):  
    """  
    Retrieves points and wins for specified grand prix since 2016  
    Returns: Performance data : Dictionary  
    """  
    df = pd.read_csv('Performances.csv', index_col=0)  
  
    current_race = self.t.get().replace('.png', '').capitalize()  
    current_driver = self.t2.get().replace('.jpg', '').capitalize()  
  
    if(current_race == 'Greatbritain'):  
        current_race = 'Great Britain'  
    elif(current_race == 'Abudhabi'):  
        current_race = 'Abu Dhabi'  
    elif(current_race == 'Unitedstates'):  
        current_race = 'United States'  
  
    points = df.at[current_race, current_driver + ' Points']  
  
    wins = df.at[current_race, current_driver + ' Wins']  
  
    return {'points':points, 'wins':wins}
```

**constructGraph** - this function works primarily in matplotlib to construct our graph which displays the average lap times for both Max and Lewis over all of the 17 different races.

- Since these two are widely regarded as the two fastest drivers currently, their times are very close to one another, especially when looking at the past 5 years which contains thousands of laps, and this is shown on the graph. However, the graph is very useful as you can see which racetracks have the biggest speed discrepancies with a quick glance.

```
def constructGraph(self):  
    """ Constructs Matplotlib graph of Max versus Lewis"""  
    getDriverData(self)  
  
    lewisRace = self.lewis_averages.keys()  
    raceTimeLewis = self.lewis_averages.values()  
  
    maxRace = self.max_averages.keys()  
    raceTimeMax = self.max_averages.values()  
    fig = plt.figure()  
    plt.plot(lewisRace, raceTimeLewis, label = 'Lewis')  
    plt.plot(maxRace, raceTimeMax, label = 'max')  
    plt.legend()  
    plt.xticks(rotation='vertical')  
    plt.xlabel('Grand Prix')  
    plt.ylabel('Time (minutes per lap)')  
    plt.title('Max Versus Lewis')  
    plt.tight_layout()  
    mydpi = 122  
    plt.savefig('maxvsLewis.png', dpi=mydpi)  
  
    self.listbox.insert(END, 'Loaded graph.png')  
    self.fp = open('maxvsLewis.png', 'rb')  
    self.img = PIL.Image.open(self.fp)  
    self.photo2 = PIL.ImageTk.PhotoImage(self.img)  
    self.l3 = Label(image=self.photo2)  
    self.l3.place(x=1100,y=400)
```

