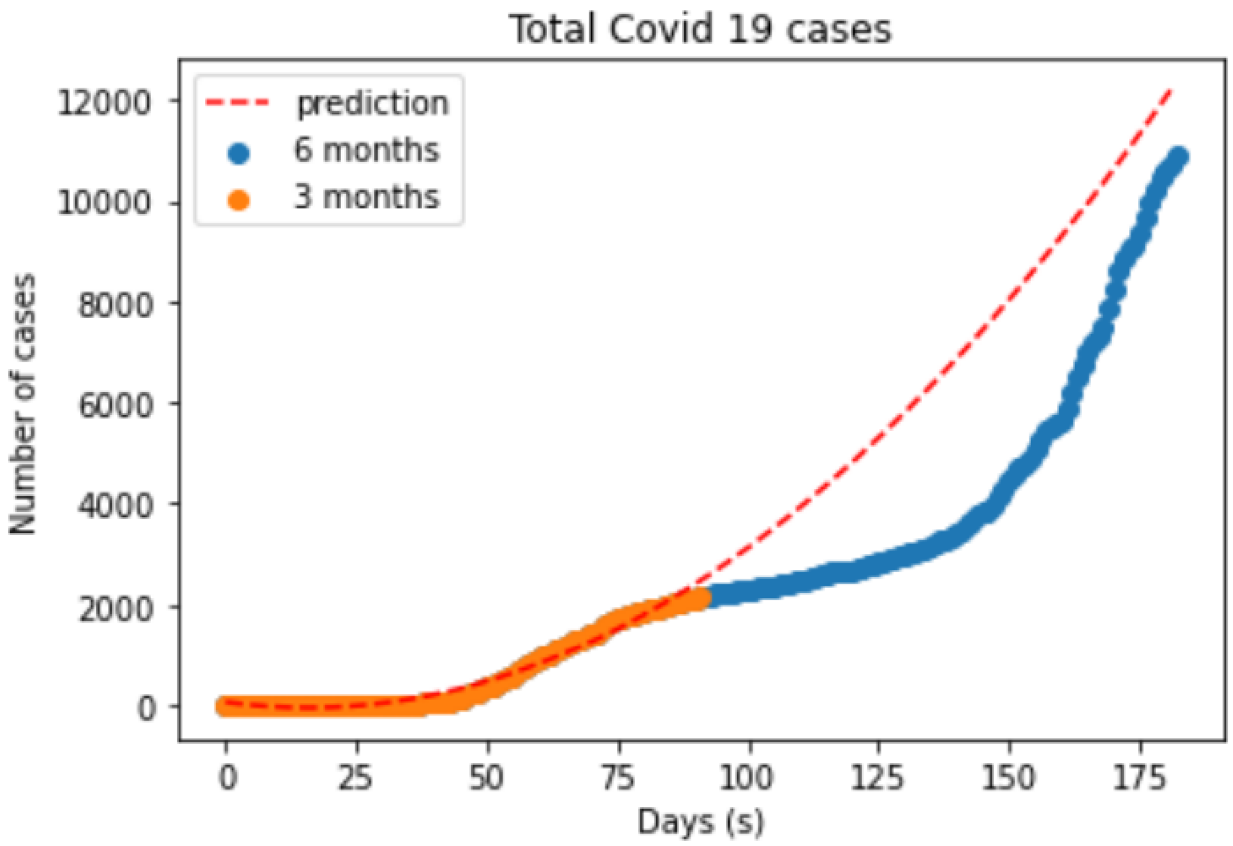
Lab 6 Documentation

Trend Prediction

This program is designed to predict the Covid 19 curve that happened within the first 7 months of 2020. This program works by first examine a CSV file with data from 1/27/2020 to 4/27/2020. This is about 90 days worth of data.

A curve fit is done where the equation of the curve fit polynomial is produced. This equation is then drawn with the number of days stretched out to predict the future curve.

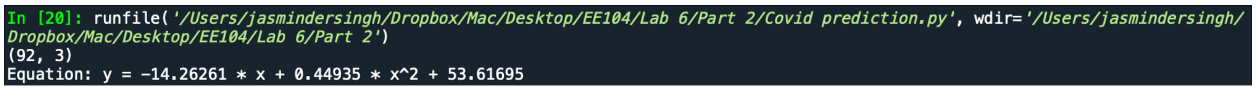
The equation was successfully obtained and we set the number of days to another 90 days extension which meant we predicted the next 3 months based on the curve fit. This curve is seen below:



As seen in the plot above, the prediction from the curve fit is somewhat really accurate. This was due to the it being the beginning stages of the pandemic and there being no vaccines. Cases were rising parabolic.

$$Y = 0.44935x^2 - 14.26261x + 53.61695$$

This equation was taken from the curve fit and it simulates is perfectly.



```
In [20]: runfile('/Users/jasmindersingh/Dropbox/Mac/Desktop/EE104/Lab 6/Part 2/Covid prediction.py', wdir='/Users/jasmindersingh/
Dropbox/Mac/Desktop/EE104/Lab 6/Part 2')
(92, 3)
Equation: y = -14.26261 * x + 0.44935 * x^2 + 53.61695
```

Risk Factor Identification

This program was designed to analyse the data to be given to financial institutions to identify the risks associated with lending money to certain people. This program is mostly built on the pandas library whereby most of the functions are based on it.

The program below imports all the necessary libraries:

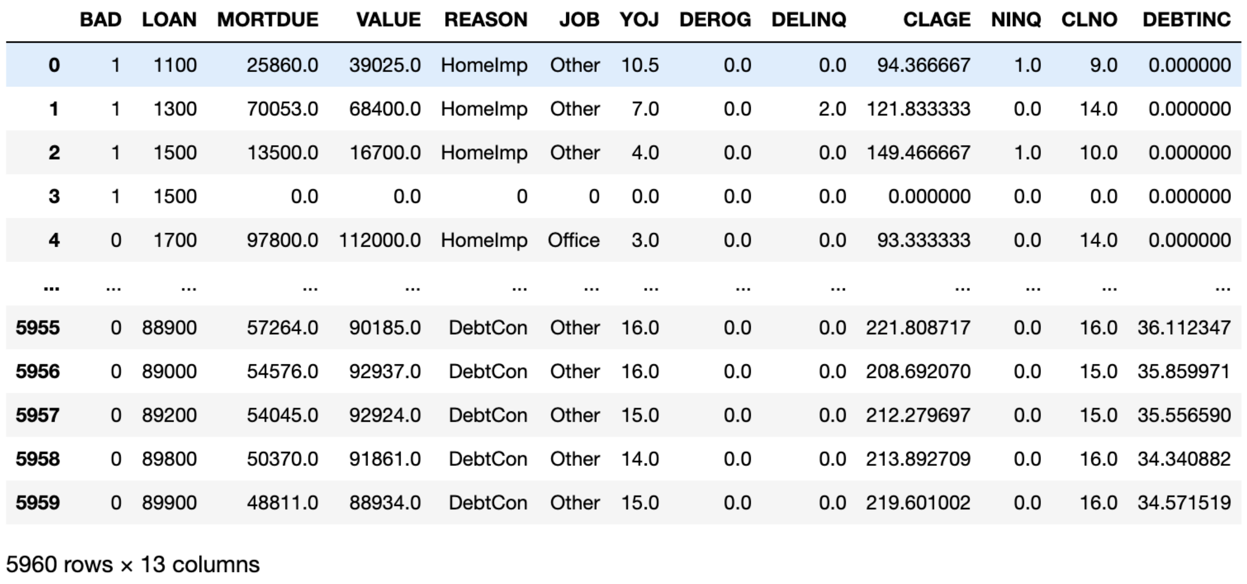
pandas: pandas is a software library written for the Python programming language for data manipulation and analysis

numpy: library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
```

The code below imports the data from the hmeq.csv file. The data is then scanned for NULL/NaN where it is then filled with 0.0

```
data = pd.read_csv("hmeq.csv")
print(data.shape)
data.head()
data.fillna(0)
```



	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
0	1	1100	25860.0	39025.0	HomeImp	Other	10.5	0.0	0.0	94.366667	1.0	9.0	0.000000
1	1	1300	70053.0	68400.0	HomeImp	Other	7.0	0.0	2.0	121.833333	0.0	14.0	0.000000
2	1	1500	13500.0	16700.0	HomeImp	Other	4.0	0.0	0.0	149.466667	1.0	10.0	0.000000
3	1	1500	0.0	0.0	0	0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000
4	0	1700	97800.0	112000.0	HomeImp	Office	3.0	0.0	0.0	93.333333	0.0	14.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5955	0	88900	57264.0	90185.0	DebtCon	Other	16.0	0.0	0.0	221.808717	0.0	16.0	36.112347
5956	0	89000	54576.0	92937.0	DebtCon	Other	16.0	0.0	0.0	208.692070	0.0	15.0	35.859971
5957	0	89200	54045.0	92924.0	DebtCon	Other	15.0	0.0	0.0	212.279697	0.0	15.0	35.556590
5958	0	89800	50370.0	91861.0	DebtCon	Other	14.0	0.0	0.0	213.892709	0.0	16.0	34.340882
5959	0	89900	48811.0	88934.0	DebtCon	Other	15.0	0.0	0.0	219.601002	0.0	16.0	34.571519

5960 rows × 13 columns

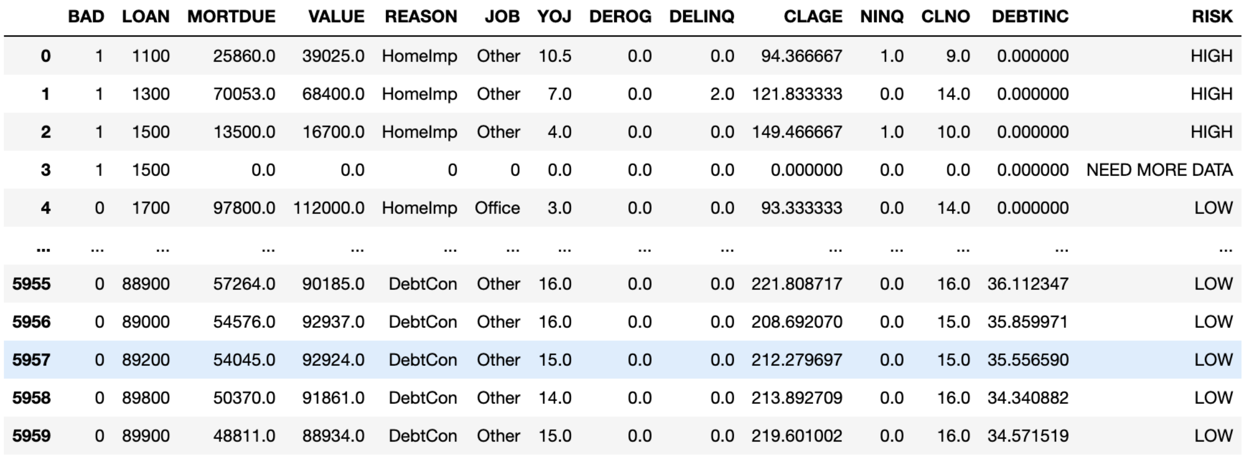
The part of the cpde below sets conditions, and based onn the conditions it go theough them one by one. So it identifies if a row is LOW risk, MEDIUM risk, HIGH risk and then if it doesnt satisfy any of these, it “NEEDS MORE DATA”

```
# create a list of our conditions
conditions = [
    (data['BAD'] != 1),
    ((data['CLAGE'] > 150) | (data['DELINQ'] > 2) & (data['DEBTINC'] > 28)),
    ((data['CLAGE'] <= 150) | (data['DEROG'] >= 1)),
    ((data['BAD'] == 1))]

# create a list of the values we want to assign for each condition
values = ['LOW', 'MEDIUM', 'HIGH', 'NEED MORE DATA']

# create a new column and use np.select to assign values to it using our lists as arguments
data['RISK'] = np.select(conditions, values)
data.fillna(0)
```

Here, it is seen to have filled the



	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC	RISK
0	1	1100	25860.0	39025.0	HomeImp	Other	10.5	0.0	0.0	94.366667	1.0	9.0	0.000000	HIGH
1	1	1300	70053.0	68400.0	HomeImp	Other	7.0	0.0	2.0	121.833333	0.0	14.0	0.000000	HIGH
2	1	1500	13500.0	16700.0	HomeImp	Other	4.0	0.0	0.0	149.466667	1.0	10.0	0.000000	HIGH
3	1	1500	0.0	0.0	0	0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	NEED MORE DATA
4	0	1700	97800.0	112000.0	HomeImp	Office	3.0	0.0	0.0	93.333333	0.0	14.0	0.000000	LOW
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5955	0	88900	57264.0	90185.0	DebtCon	Other	16.0	0.0	0.0	221.808717	0.0	16.0	36.112347	LOW
5956	0	89000	54576.0	92937.0	DebtCon	Other	16.0	0.0	0.0	208.692070	0.0	15.0	35.859971	LOW
5957	0	89200	54045.0	92924.0	DebtCon	Other	15.0	0.0	0.0	212.279697	0.0	15.0	35.556590	LOW
5958	0	89800	50370.0	91861.0	DebtCon	Other	14.0	0.0	0.0	213.892709	0.0	16.0	34.340882	LOW
5959	0	89900	48811.0	88934.0	DebtCon	Other	15.0	0.0	0.0	219.601002	0.0	16.0	34.571519	LOW

GAME

In this program, I changed 2 main components of the game:

1) Increased the number of dots from 10 to 15

The code below is shows the for loop that repeats 15 times. Each time it repeats, it drops a new dit

```
for dot in range(0,15):                 #loops 15 times
    actor = Actor("dot")

    actor.pos = randint(20, WIDTH-20), randint(20, HEIGHT-20)

    dots.append(actor)
```

2) Game ends when the user doesnt get the correct click

This feature doesnt give the user a chance and it exits immediately. This is seen in the code below where the if else statement within the on_moiuse_down

```
def on_mouse_down(pos):
    global next_dot
    global lines
```

```
if dots[next_dot].collidepoint(pos):
    if next_dot:
        lines.append((dots[next_dot-1].pos, dots[next_dot].pos))
    next_dot = next_dot + 1
else:
    quit()
```

Hardware Development

Code were ran based on the Lab 6 manual that was shared.

README