```python
## this file is intended to store all necessary inputs
from ambiance import Atmosphere   ## All in SI units
import numpy as np
from math import *
from pandas import *

#   Encapsulation into a class
class Aircraft:
#Important Data
    span = 44.84  # in m
    S = 260 # wing ref area in m^2
    e = 0.85 # oswald effy
    k = 1/(pi*e*(span**2)/S)
    C_D0 = 0.016
    C_D0_OEI = 0.02
    k_OEI = 1/(pi*0.75*(span**2)/S)
    C_Lmax = 1.7
    W = 160000*9.81 # Weight in N

    Drag_div_M = 0.85  # Mach
    h_max = 30480 # in m or 100 kft intended to impose a limit on y-axis span
    steps = 5000   # number of points
    h = np.linspace (0,h_max,steps, endpoint = True)
    ft_to_m = 0.305 # conversion factor from ft to m
    lb_to_N = 4.4482216282509 # conversion factor lb to N
    metric_to_imperial_power = 0.7375621493  #from N m/s to lb ft/s
    m2_to_ft2 = 10.764
    conditions = Atmosphere (h)

    def ceiling (P_max): # the total power to get service ceiling in ft
        weight = Aircraft.W/Aircraft.lb_to_N  # unit lb
        power = P_max*Aircraft.metric_to_imperial_power # unit  lb ft/s
        atm = Atmosphere (0)
        density = atm.density
        area = Aircraft.S *Aircraft.m2_to_ft2
        num = 0.7436 * Aircraft.C_D0^0.25
        den = (Aircraft.k/pi) ^ (4/3)
        # equation from intro to flight by anderson (E 6.15.8)
        absolute_ceiling = -19867*log(weight/power)*sqrt((weight/area)*2/density)*((num)/(den))
        return absolute_ceiling

    def Zero_Lift_Drag (M, config='AEO'):    # function to return C_D0
        if config == 'AEO':
            if M<=Aircraft.Drag_div_M:
                Drag_coeff = Aircraft.C_D0
            else:
                Drag_coeff = Aircraft.C_D0*(1+(M-0.85)/0.1)

        elif config =='OEI':
            if M<=Aircraft.Drag_div_M:
                Drag_coeff = Aircraft.C_D0_OEI
            else:
                Drag_coeff = Aircraft.C_D0_OEI*(1+(M-0.85)/0.1)
        return Drag_coeff

    def CL_from_M (Mach, Height =0):
        cond = Atmosphere (Height)
        Pressure = cond.pressure
        # avoid division by zero
        if Mach == 0:
            Lift_coeff = 0
        else:
            Lift_coeff = Aircraft.W/(0.7*Pressure*Aircraft.S*(Mach**2))
        return Lift_coeff

    def M_from_CL (CL, Height =0):
        cond = Atmosphere (Height)
        Pressure = cond.pressure
        if CL <= 0.15:    # it causes error and glitching for some reason. CD will peak under 0.15 otherwise.
            Mach = 0
        else:
            Mach = sqrt ((Aircraft.W)/(0.7*(Pressure)*(Aircraft.S)*(CL)))
        return Mach

    def poly_inter (thrust_array, mach_array, degree): # extract polynomial coefficients for Thrust (y) vs Mach (x)
        coefficients = np.polyfit(mach_array, thrust_array, degree) # degreee of polynomial
        polynomial = np.poly1d(coefficients)
        return polynomial

    def Thrust_Profile (name):
        lb_to_N= Aircraft.lb_to_N
        poly_inter = Aircraft.poly_inter
        # arrays for thrust and altitude
```

```python
    def remove_nan(data, col_name):
        df = DataFrame(data, columns=[col_name])
        df = df[df[col_name].notna()]
        return df[col_name].tolist()

    class Thrust_Data1:
    # reading CSV file
        data1 = read_csv("PW JT9D-3.csv")

    # converting column data to list with no terms NAN
        mach_00   = remove_nan(data1, 'PW JT9D-3 00 X')
        thrust_00 = remove_nan(data1, 'PW JT9D-3 00 Y')
        mach_15   = remove_nan(data1, 'PW JT9D-3 15 X')
        thrust_15 = remove_nan(data1, 'PW JT9D-3 15 Y')
        mach_25   = remove_nan(data1, 'PW JT9D-3 25 X')
        thrust_25 = remove_nan(data1, 'PW JT9D-3 25 Y')
        mach_35   = remove_nan(data1, 'PW JT9D-3 35 X')
        thrust_35 = remove_nan(data1, 'PW JT9D-3 35 Y')
        mach_45   = remove_nan(data1, 'PW JT9D-3 45 X')
        thrust_45 = remove_nan(data1, 'PW JT9D-3 45 Y')

        #3 or 2 degree polynomial should suffice
        polynomial_00 = poly_inter(thrust_00, mach_00, 3)
        polynomial_15 = poly_inter(thrust_15, mach_15, 3)
        polynomial_25 = poly_inter(thrust_25, mach_25, 3)
        polynomial_35 = poly_inter(thrust_35, mach_35, 3)
        polynomial_45 = poly_inter(thrust_45, mach_45, 3)

        alt = [0,15000,25000,35000,45000]    # altitude in ft
        bounds = 5

    class Thrust_Data2:
        data2 = read_csv("PW 4056.csv")
        # converting column data to list
        mach_00   = remove_nan(data2,'PW 4056 00 X')
        thrust_00 = remove_nan(data2,'PW 4056 00 Y')
        mach_05   = remove_nan(data2,'PW 4056 05 X')
        thrust_05 = remove_nan(data2,'PW 4056 05 Y')
        mach_10   = remove_nan(data2,'PW 4056 10 X')
        thrust_10 = remove_nan(data2,'PW 4056 10 Y')
        mach_15   = remove_nan(data2,'PW 4056 15 X')
        thrust_15 = remove_nan(data2,'PW 4056 15 Y')
        mach_20   = remove_nan(data2,'PW 4056 20 X')
        thrust_20 = remove_nan(data2,'PW 4056 20 Y')
        mach_25   = remove_nan(data2,'PW 4056 25 X')
        thrust_25 = remove_nan(data2,'PW 4056 25 Y')
        mach_30   = remove_nan(data2,'PW 4056 30 X')
        thrust_30 = remove_nan(data2,'PW 4056 30 Y')
        mach_35   = remove_nan(data2,'PW 4056 35 X')
        thrust_35 = remove_nan(data2,'PW 4056 35 Y')
        mach_40   = remove_nan(data2,'PW 4056 40 X')
        thrust_40 = remove_nan(data2,'PW 4056 40 Y')
        mach_45   = remove_nan(data2,'PW 4056 45 X')
        thrust_45 = remove_nan(data2,'PW 4056 45 Y')

        #3 or 2 degree polynomial should suffice
        polynomial_00 = poly_inter(thrust_00, mach_00, 3)
        polynomial_05 = poly_inter(thrust_05, mach_05, 3)
        polynomial_10 = poly_inter(thrust_10, mach_10, 3)
        polynomial_15 = poly_inter(thrust_15, mach_15, 3)
        polynomial_20 = poly_inter(thrust_20, mach_20, 3)
        polynomial_25 = poly_inter(thrust_25, mach_25, 3)
        polynomial_30 = poly_inter(thrust_30, mach_30, 3)
        polynomial_35 = poly_inter(thrust_35, mach_35, 3)
        polynomial_40 = poly_inter(thrust_40, mach_40, 3)
        polynomial_45 = poly_inter(thrust_45, mach_45, 3)

        alt = [0,5000,10000,15000,20000,25000,30000,35000,40000,45000]    # altitude in ft
        bounds = 10

    # or maybe it shoudl return mmax and hv max
    if name == 'Profile1':
        return Thrust_Data1

    elif name == 'Profile2':
        return Thrust_Data2

def Thrust (alt,profile, config):     # intended to get return a thrust value at a specific altitude
    # retrieve thrust info
    Thrust_Data = Aircraft.Thrust_Profile(profile)
    # partition the thrust profile into sections and bounds would be either alt1, alt2
```

```python
        # m is the mach number
        # start doing the mach thing here
        Atm = Atmosphere(alt*Aircraft.ft_to_m)
        mstall = sqrt(Aircraft.W/(0.7*Atm.pressure*Aircraft.S*Aircraft.C_Lmax))
        m = np.linspace(mstall,0.6, Aircraft.steps)
        Thrust = np.linspace(mstall,0.6, Aircraft.steps)
        j = 0
        mmax = np.linspace(0,1,1)
        hvmax = np.linspace(0,1,1)
        for i in m:
            if profile == 'Profile1':
                thrust_from_mach_00 = Thrust_Data.polynomial_00(i)
                thrust_from_mach_15 = Thrust_Data.polynomial_15(i)
                thrust_from_mach_25 = Thrust_Data.polynomial_25(i)
                thrust_from_mach_35 = Thrust_Data.polynomial_35(i)
                thrust_from_mach_45 = Thrust_Data.polynomial_45(i)
                if   (0==alt):
                    thrust_val = thrust_from_mach_00
                elif (0<alt & alt<=15000):
                    thrust_val = ((alt-0)/(15000-0))*(thrust_from_mach_15-thrust_from_mach_00) + thrust_from_mach_00
                elif (15000<alt & alt<=25000):
                    thrust_val = ((alt-15000)/(25000-15000))*(thrust_from_mach_25-thrust_from_mach_15) + thrust_from_mach_15
                elif (25000<alt & alt<=35000):
                    thrust_val = ((alt-25000)/(35000-25000))*(thrust_from_mach_35-thrust_from_mach_25) + thrust_from_mach_25
                elif (35000<alt & alt<=45000):
                    thrust_val = ((alt-35000)/(45000-35000))*(thrust_from_mach_45-thrust_from_mach_35) + thrust_from_mach_35

            elif profile == 'Profile2':
                thrust_from_mach_00 = Thrust_Data.polynomial_00(i)
                thrust_from_mach_05 = Thrust_Data.polynomial_05(i)
                thrust_from_mach_10 = Thrust_Data.polynomial_10(i)
                thrust_from_mach_15 = Thrust_Data.polynomial_15(i)
                thrust_from_mach_20 = Thrust_Data.polynomial_20(i)
                thrust_from_mach_25 = Thrust_Data.polynomial_25(i)
                thrust_from_mach_30 = Thrust_Data.polynomial_30(i)
                thrust_from_mach_35 = Thrust_Data.polynomial_35(i)
                thrust_from_mach_40 = Thrust_Data.polynomial_40(i)
                thrust_from_mach_45 = Thrust_Data.polynomial_45(i)

                if   (0==alt):
                    thrust_val = thrust_from_mach_00
                elif (0<alt & alt<=5000):
                    thrust_val = ((alt-0)/(5000-0))*(thrust_from_mach_05-thrust_from_mach_00) + thrust_from_mach_00
                elif (5000<alt & alt<=10000):
                    thrust_val = ((alt-5000)/(10000 -5000))*(thrust_from_mach_10-thrust_from_mach_05) + thrust_from_mach_05
                elif (10000<alt & alt<=15000):
                    thrust_val = ((alt-10000)/(15000-10000))*(thrust_from_mach_15-thrust_from_mach_10) + thrust_from_mach_10
                elif (15000<alt & alt<=20000):
                    thrust_val = ((alt-15000)/(20000-15000))*(thrust_from_mach_20-thrust_from_mach_15) + thrust_from_mach_15
                elif (20000<alt & alt<=25000):
                    thrust_val = ((alt-20000)/(25000-20000))*(thrust_from_mach_25-thrust_from_mach_20) + thrust_from_mach_20
                elif (25000<alt & alt<=30000):
                    thrust_val = ((alt-25000)/(30000-25000))*(thrust_from_mach_30-thrust_from_mach_25) + thrust_from_mach_25
                elif (30000<alt & alt<=35000):
                    thrust_val = ((alt-30000)/(35000-30000))*(thrust_from_mach_35-thrust_from_mach_30) + thrust_from_mach_30
                elif (35000<alt & alt<=40000):
                    thrust_val = ((alt-35000)/(40000-35000))*(thrust_from_mach_40-thrust_from_mach_35) + thrust_from_mach_35
                elif (40000<alt & alt<=45000):
                    thrust_val = ((alt-40000)/(45000-40000))*(thrust_from_mach_45-thrust_from_mach_40) + thrust_from_mach_40

            CL = Aircraft.W/(0.7*Atm.pressure*Aircraft.S*(i**2))
            if config == 'AEO':
                CD = Aircraft.Zero_Lift_Drag(i, config) + Aircraft.k*(CL**2)
                if abs (thrust_val*Aircraft.lb_to_N*2 - 0.7* Atm.pressure* (i**2)*Aircraft.S*CD) < 75000:
                    maxalt = alt
            else:
                CD = Aircraft.Zero_Lift_Drag(i, config) + Aircraft.k_OEI*(CL**2)
                if abs (thrust_val*Aircraft.lb_to_N - 0.7* Atm.pressure* (i**2)*Aircraft.S*CD) < 75000:
                    maxalt = alt


                if m[j]>mmax[0]:
                     mmax[0] = m[j]
                     hvmax[0] = alt

            Thrust [j] = thrust_val
            j = j + 1
        return Thrust, m, mmax, hvmax
```