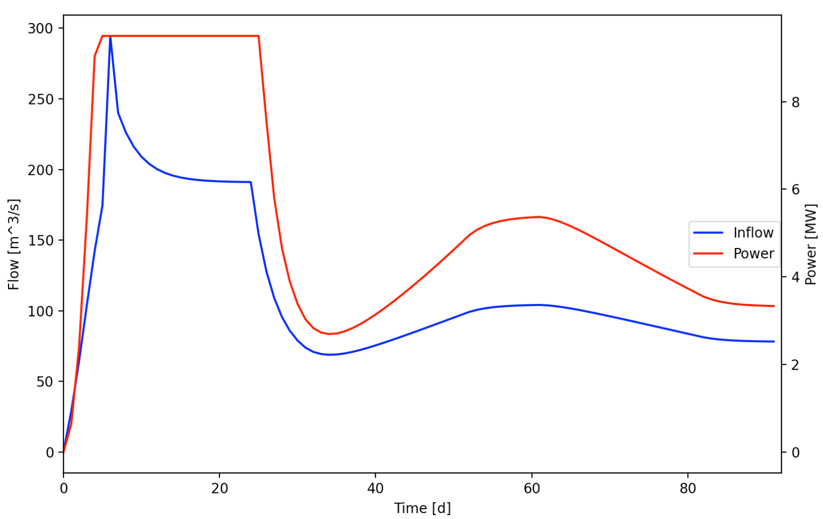
```python
import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

turbine_output = 9500 # kW
turbine_efficiency =  0.92 # %92 efficiency
watershed_area =  2950000000 # m^2
sec_per_day = 24 * 60 * 60
snow_amount = 200 # cm

# runoff forecast model based on model from HydroA/part6.py

t_1 = 0 # d
t_2 = 1 # d
Q_1 = 0 # m^3/s
Q_2 = 0 # initializing
A_neg = -1/3 # d

R_forecast = np.array([0, 0, 0, 0, 0, 15, 0]) # mm per hour
R_avg = np.zeros(85) # 92 days - 7 forecast days
R_avg[:24] = 1.74
R_avg[24:54] = 3
R_avg[54:] = 2.23
R_avg_sum = np.sum(R_avg)

T_forecast = np.array([14, 18, 21, 23, 24, 14, 15])
T_avg = np.zeros(85) # 92 days - 7 forecast days
T_avg[:24] = 17
T_avg[24:54] = 20
T_avg[54:] = 22
T_avg_sum = np.sum(T_avg)

# smoothing out rainfall averages, averaging over a sliding ~21 day window
R_df = pd.DataFrame({'Rainfall': R_avg})
R_df = R_df.rolling(21, min_periods=1).mean()
R_avg = R_df['Rainfall'].values

# adding back missing data from smoothing
R_avg_delta = R_avg_sum - np.sum(R_avg)
R_avg_delta_avg = np.full(len(R_avg), (R_avg_delta / len(R_avg)))
R_avg = R_avg + R_avg_delta_avg
R = np.concatenate([R_forecast, R_avg])

# finding bias
df = pd.read_csv('21dayforecast.csv')
df_bias = df.copy()
for day in range(1,8):
    column = "D" + str(day)
    for i in range(21):
        df_bias.iloc[i, df_bias.columns.get_loc(column)] = df.iloc[i, df.columns.get_loc(column)] - df.iloc[i, df.columns.get_loc('Observed')]

#averaging biases
df_bias = df_bias.sum(axis=0)
for i in range(1,8):
    day = "D" + str(i)
    df_bias[day] = df_bias[day] / 21

# applying bias to 7 day forecast
for day in range(1,8):
    column = "D" + str(day)
    T_forecast[day - 1] = T_forecast[day - 1] - df_bias[column]

# smoothing out temperature averages, averaging over a sliding ~21 day window
T_df = pd.DataFrame({'Temp.': T_avg})
T_df = T_df.rolling(21, min_periods=1).mean()
T_avg = T_df['Temp.'].values
# adding back missing data from smoothing
T_avg_delta = T_avg_sum - np.sum(T_avg)
T_avg_delta_avg = np.full(len(T_avg), (T_avg_delta / len(T_avg)))
T_avg = T_avg + T_avg_delta_avg
T = np.concatenate([T_forecast, T_avg])
T = T - 10 # subtracting 10 C threshold to find difference to make runoff in mm
```

```python
# calculate melt from 200 cm of snow
# remove melt amounts from T once 200 cm has been exceeded
cur_melt_total = 0
last_melt_day = len(T)
for i in range(len(T)):
    cur_melt_total += T[i]
    if (cur_melt_total >= snow_amount):
        T[i] = T[i] - (cur_melt_total - snow_amount)
        last_melt_day = i
        break

T[last_melt_day:] = [0] * (len(T) - last_melt_day)

# R = R + T # adding melt and rainfall mm totals together

# converting rainfall to flow for that day
R = [r / 1000 for r in R] # mm to m conversion
R = [r / sec_per_day for r in R] # day to s conversion
R = [r * watershed_area for r in R] # water amount converter to volume (m^3/s)

# converting snowmelt to flow for that day
T = [t / 1000 for t in T] # mm to m conversion
T = [t / sec_per_day for t in T] # day to s conversion
T = [t * watershed_area * 0.5 for t in T]   # water amount converter to volume (m^3/s)
                                            # and accounting for 50% of area

R = np.array(R) + np.array(T) # adding melt and rainfall mm totals together

# calculate Q/ inflow
inflow = [0] * len(R) # m^3/s

for i in range(len(R)):
    inflow[i] = Q_1
    Q_2 = Q_1 * math.exp(A_neg * (t_2 - t_1)) + R[i] * (1 - math.exp(A_neg * (t_2 - t_1)))
    Q_1 = Q_2                        # ^ d in t_1 & t_2 cancel out with d^-1 in A_neg ^
    t_1 += 1
    t_2 += 1

# calculate river velocity
B = 2.25
velocity = np.array([B * q ** (1/3) for q in inflow])

#calculate power output
power = np.zeros(len(inflow))
for i in range(len(inflow)):
    power[i] = 0.5 * turbine_efficiency * inflow[i] * (velocity[i] ** 2) / 1000 # MW
    if (power[i] > 9.5):            # ^^ removed density as part of W to MW converson
        power[i] = 9.5

revenue_day = np.array([p * 1000 * 24 * 0.25 for p in power]) # daily revenue
revenue_total = np.sum(revenue_day)
print("Total revenue: " + str(revenue_total))

x = list(range(len(inflow)))

fig, host = plt.subplots(1, 1, figsize=(10,6))

par = host.twinx()

p1, = host.plot(x, inflow, label='Inflow', color='blue')
p2, = par.plot(x, power, label='Power', color='red')

host.set_xlim(0,len(R))

host.set_ylabel('Flow [m^3/s]')
host.set_xlabel('Time [d]')
par.set_ylabel('Power [MW]')

lines = [p1, p2]

host.legend(lines, [l.get_label() for l in lines], loc="center right", borderaxespad=0.1)

plt.subplots_adjust(right=0.85)

plt.show()
```



I found the total revenue to be $2952465.62.

So I decided to choose C 2.7 mill because most of the values were based off highs and it would not make sense to over estimate revenue for this scenario. Additionally we used smoother values for rainfall and temperatures if the we more erratic then we could see more scenarios where the power generated is maxed and then a quick drop do to lack of rain compared to a high intensity precipitation event. This would cause less revenue in the dryer days and wasted revenue on the high precipitation days.

The area risks of floods and avalanches/slides in the area with high intensity precipitation events.