DMU Assignment

Question 1

Task 1

	High demand	Medium demand	Low demand		Expected Utility
Invest in A	200000	80000	-30000		83333.33333
iInvest in B	150000	100000	10000		86666.66667
Do not invest	0	0	0		0

Expected utility

Invest in A → $\frac{1}{3}(200000 + 80000 - 30000) = 83,333.33$

Invest in B → $\frac{1}{3}(150000 + 100000 + 10000) = 86,666.67$

Do not invest → $\frac{1}{3}(0 + 0 + 0) = 0$

- Maximum utility decision is Investing in B

Task 2

Regret for each market demand scenario					Max Regret for each decision
	High demand	Medium demand	Low demand		Max Regret for each decision
Invest in A	0	20000	40000		40000
iInvest in B	50000	0	0		50000
Do not invest	200000	100000	10000		200000
Optimal decision	Investing in B	(Since minimum regret is 40000)			

	High Demand	Medium Demand	Low Demand
Invest in A	200000 − 200000	100000 − 80000	10000 + 30000
Invest in B	200000 − 150000	100000 − 100000	10000 − 10000
Do not invest	200000 − 0	100000 − 0	10000 − 0

Task 3

- Code

```
import numpy as np

def expected_utility(profit_values,probabilities):
    """
    computes the expected utility of each investment
    :param profit_values: numpy array of profits for each investment
    :param probabilities: list of probabilities for each investment
    :return: a numpy array of expected utility of each investment
    """
    return np.dot(profit_values, probabilities)

def minimax_regret_decision(profit_values):
    """
```

```
    computes the regret of each investment
    :param profit_values: numpy array of profits for each investment
    :return: Decision with the minimum regret
    """
    max_regret_for_each_decision = np.max(np.max(profit_values, axis=0) - profit_values, axis=1)
    optimal_decision = f"Decision {np.where(max_regret_for_each_decision == np.min(max_regret_for_each_decision))[0][0] + 1}"
    return optimal_decision
```

- Example input

```
investments = np.array([[200000, 80000, -30000],
                        [150000, 100000, 10000],
                        [0, 0, 0]])

if __name__ == "__main__":
    x = expected_utility(investments, np.array([0.5, 0.3, 0.2]))
    print('Expected utility for investing in A: {:.2f}'.format(x[0]))
    print('Expected utility for investing in B: {:.2f}'.format(x[1]))
    print('Expected utility for not investing: {:.2f}'.format(x[2]))
    print(f"Optimal decision with the minimum regret is: "
          f"{minimax_regret_decision(investments)}")
```

- Output

```
Expected utility for investing in A: 118000.00
Expected utility for investing in B: 107000.00
Expected utility for not investing: 0.00
Optimal decision with the minimum regret is: Decision 1
```

- This means that it is best to invest in A

Task 4

- Probability is not a parameter for the minimax regret function, as such it does not affect the changes to the optimal decision
- As such, different probabilities will not affect the sensitivity of the optimal decision.
- However, since profit values is a parameter that the minimax regret function takes, it will affect the sensitivity of the optimal decision
    - Using the following profit values, we tested the sensitivity of the minimax regret function

```
investments_2 = np.array([[3, 5],
                        [1, 7]])
investments_3 = np.array([[7, 11],
                        [3, 16]])
investments_4 = np.array([[7, 11],
                        [3, 15]])
```

```
Optimal decision with the minimum regret is: Decision 1
Optimal decision with the minimum regret is: Decision 2
Optimal decision with the minimum regret is: Decision 1
```

- From the investments_3 and investments_4, we can see that a small change in one of the values (16 to 15), can change the optimal decision from decision 2 to decision 1. This makes it shows that the optimal decisions are very sensitive to changes in these parameters. Further analysis into this would enable us to achieve more certainty in this. Based on this, due to its high sensitivity, the minimax regret function may not be the most appropriate to use in cases where there are minute changes in values occurring.

Question 2

Task 1

```
import numpy as np

def sim_eco_conditions(N):
    """
    Part of two-step simulation to generate n samples of a market demand scenario based on economic conditions
    :param N: Number of samples
    :return: a list of economic conditions (strings)
    """

    # Define the probability distributions
    economic_probs = {'Good': 0.5, 'Moderate': 0.3, 'Poor': 0.2}

    # Step 1: Simulate economic conditions
    economic_conditions = np.random.choice(list(economic_probs.keys()), size=N, p=list(economic_probs.values()))

    return economic_conditions

def sim_demand_scenarios(economic_conditions, N):
    """
    Part of two-step simulation to generate n samples of a market demand scenario based on economic conditions
    :param eco_conditions: a list of economic conditions (strings)
    :param N: Number of samples
    :return: a list of market demand scenarios (strings)
    """
    demand_probs = {
        'Good': {'High': 0.6, 'Medium': 0.3, 'Low': 0.1},
        'Moderate': {'High': 0.3, 'Medium': 0.5, 'Low': 0.2},
        'Poor': {'High': 0.1, 'Medium': 0.4, 'Low': 0.5}
    }

    # Step 2: Simulate market demand scenarios based on economic condition
    demand_probs_given_eco = np.array([demand_probs[condition] for condition in economic_conditions])
    demand_scenarios = np.array(
        [np.random.choice(list(demand_probs_given_eco[i].keys()), p=list(demand_probs_given_eco[i].values())) for i in
         range(N)])

    return demand_scenarios
```

Task b

- Simulation Results

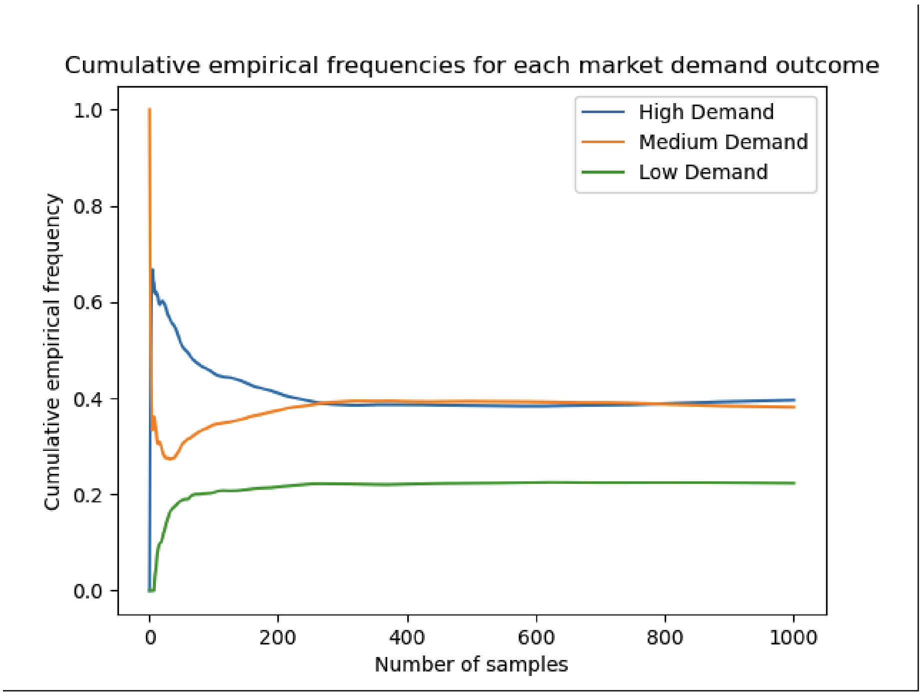
```
if __name__ == "__main__":
    # Storing the simulated outcomes in a vector y
    x = sim_eco_conditions(1000)
    y = sim_demand_scenarios(x, 1000)
    print(y)
```

```
['Medium' 'Medium' 'Medium' 'Medium' 'High' 'High' 'Medium' 'Medium'
 'High' 'Low' 'Medium' 'Low' 'High' 'Low' 'Medium' 'Low' 'High' 'Medium'
 'High' 'High' 'Low' 'Medium' 'High' 'Medium' 'High' 'High' 'Low' 'High'
 'Medium' 'Medium' 'High' 'High' 'Low' 'High' 'Medium' 'Medium' 'High'
 'High' 'Medium' 'Medium' 'High' 'Medium' 'High' 'Medium' 'Medium' 'High'
 'Low' 'High' 'Medium' 'High' 'Medium' 'Low' 'High' 'High' 'Low' 'High'
 'Medium' 'Medium' 'Low' 'Medium' 'High' 'Medium' 'Low' 'High' 'High'
 'High' 'Medium' 'Low' 'High' 'High' 'Low' 'Low' 'Medium' 'Medium' 'Low'
 'Medium' 'High' 'Medium' 'Medium' 'Low' 'High' 'High' 'Medium' 'High'
 'Medium' 'Low' 'Low' 'High' 'Medium' 'Low' 'High' 'Low' 'Low' 'Medium'
 'High' 'Medium' 'High' 'High' 'Medium' 'Medium' 'High' 'Medium' 'High'
 'High' 'High' 'Low' 'High' 'High' 'High' 'High' 'Medium' 'High' 'High'
 'Medium' 'Medium' 'Medium' 'Medium' 'Low' 'High' 'Medium' 'Medium'
 'Medium' 'High' 'Medium' 'High' 'Medium' 'High' 'Medium' 'Low' 'High'
 'Low' 'Medium' 'High' 'Medium' 'High' 'Medium' 'High' 'High' 'Medium'
 'Low' 'Low' 'Low' 'High' 'Medium' 'High' 'Medium' 'Low' 'Medium' 'High'
 'High' 'High' 'High' 'Medium' 'Low' 'Medium' 'High' 'Medium' 'Medium'
 'Medium' 'Low' 'High' 'Medium' 'Medium' 'High' 'Medium' 'Medium' 'High'
 'High' 'Medium' 'Medium' 'High' 'High' 'Medium' 'High' 'High' 'High'
 'High' 'Medium' 'Low' 'High' 'High' 'Low' 'High' 'High' 'Low' 'High'
 'High' 'High' 'Low' 'Low' 'High' 'Medium' 'High' 'Medium' 'High' 'Medium'
 'High' 'Medium' 'Low' 'High' 'High' 'Low' 'Medium' 'Medium' 'Medium'
 'High' 'Medium' 'Medium' 'Medium' 'Medium' 'High' 'High' 'High' 'Medium'
 'High' 'High' 'Low' 'Low' 'Medium' 'High' 'High' 'High' 'High' 'High'
```

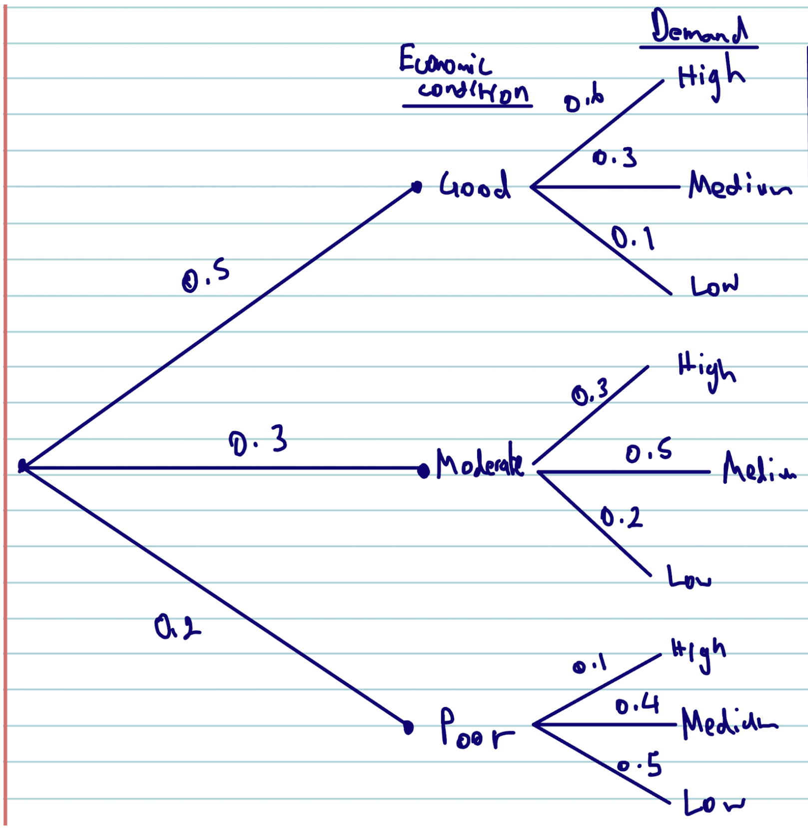
```
 'Medium' 'Low' 'High' 'Medium' 'Medium' 'High' 'Medium' 'High' 'Medium'
 'Medium' 'Medium' 'High' 'Medium' 'Low' 'Low' 'Low' 'High' 'High'
 'Medium' 'Low' 'Medium' 'Low' 'High' 'Medium' 'High' 'High' 'Low' 'High'
 'Medium' 'High' 'Medium' 'Low' 'High' 'High' 'Medium' 'Low' 'High'
 'Medium' 'High' 'Medium' 'High' 'High' 'Medium' 'Medium' 'High' 'High'
 'High' 'Low' 'High' 'Medium' 'High' 'High' 'Medium' 'Low' 'Low' 'High'
 'Low' 'Low' 'Low' 'Medium' 'High' 'Low' 'Low' 'Medium' 'Medium' 'Low'
 'High' 'Medium' 'Medium' 'High' 'Medium' 'Medium' 'Medium' 'Medium'
 'High' 'High' 'Medium' 'High' 'High' 'High' 'Medium' 'Medium' 'Medium'
 'Low' 'Medium' 'Low' 'High' 'Low' 'High' 'Medium' 'Low' 'Low' 'High'
 'Low' 'Low' 'Medium' 'High' 'Low' 'High' 'Medium' 'High' 'High' 'Low'
 'Low' 'High' 'Medium' 'Medium' 'High' 'High' 'Medium' 'Low' 'Medium'
 'Low' 'Low' 'Medium' 'Medium' 'Medium' 'High' 'Low' 'High' 'High' 'High'
 'High' 'Low' 'High' 'Low' 'Medium' 'High' 'Low' 'Low' 'Medium' 'High'
 'Low' 'High' 'Medium' 'Low' 'Medium' 'High' 'Medium' 'Medium' 'High'
 'High' 'Medium' 'Medium' 'High' 'Medium' 'High' 'Low' 'Medium' 'Medium'
 'Medium' 'Low' 'Low' 'Medium' 'High' 'High' 'Medium' 'Low' 'High'
 'Medium' 'High' 'High' 'Medium' 'High' 'Low' 'High' 'Medium' 'Low'
 'Medium' 'High' 'Low' 'High' 'High' 'High' 'High' 'High' 'High' 'Medium'
 'Medium' 'Low' 'Medium' 'High' 'Medium' 'High' 'High' 'Medium' 'Low'
 'Medium' 'High' 'High' 'High' 'Medium' 'Medium' 'Medium' 'Low' 'Low'
 'Medium' 'Medium' 'High' 'High' 'High' 'High' 'High' 'Low' 'Medium' 'Low'
 'High' 'High' 'Low' 'High' 'High' 'Low' 'Low' 'Medium' 'Low' 'Medium'
 'High' 'High' 'Medium' 'Medium' 'High' 'Low' 'High' 'Medium' 'Low' 'High'
 'High' 'Medium' 'Medium' 'Medium' 'Medium' 'Medium' 'High' 'Low' 'Medium'
 'High' 'Low' 'Medium' 'High' 'High' 'Low' 'Medium' 'High' 'Medium' 'High'
 'High' 'High' 'Low' 'High' 'Low' 'High' 'Medium' 'Medium' 'High' 'Medium'
 'High' 'High' 'High' 'Low' 'High' 'Medium' 'High' 'Low' 'High' 'Medium'
 'High' 'High' 'Low' 'High' 'High' 'Medium' 'High' 'Medium' 'Low' 'High'
 'Low' 'Medium' 'Medium' 'Medium' 'Medium' 'High' 'Medium' 'Medium' 'High'
 'Low' 'High' 'Medium' 'Medium' 'Low' 'High' 'High' 'High' 'Medium' 'High'
 'Low' 'Medium' 'High' 'Medium' 'High' 'Medium' 'Low' 'High' 'Low' 'Low'
 'High' 'High' 'High' 'High' 'Medium' 'High' 'High' 'High' 'High' 'Medium'
 'Medium' 'Medium' 'Medium' 'High' 'High' 'High' 'High' 'Low' 'Medium'
 'Medium' 'Medium' 'Low' 'High' 'High' 'High' 'Medium' 'Medium' 'High'
 'High' 'High' 'High' 'Medium' 'Low' 'Medium' 'High' 'Medium' 'High' 'Low'
 'High' 'High' 'High' 'Low' 'High' 'Medium' 'Low' 'Medium' 'High' 'Medium'
 'Medium' 'High' 'Medium' 'Low' 'High' 'Low' 'Medium' 'High' 'Low' 'High'
 'Medium' 'Medium' 'High' 'High' 'High' 'High' 'High' 'Medium' 'Low'
 'Medium' 'Low' 'Medium' 'Medium' 'High' 'High' 'Medium' 'High' 'Medium'
 'Medium' 'Low' 'High' 'Medium' 'Medium' 'Medium' 'High' 'High' 'High'
 'Medium' 'High' 'High' 'High' 'High' 'Medium' 'Medium' 'Low' 'Low' 'High'
 'Low' 'Medium' 'Low' 'Medium' 'High' 'High' 'High' 'High' 'Medium' 'High'
 'High' 'Medium' 'High' 'High' 'High' 'Medium' 'Low' 'High' 'High' 'High'
 'Medium' 'Low' 'Medium' 'Low' 'High' 'Medium' 'High' 'High' 'Low' 'High'
 'High' 'High' 'Low' 'Medium' 'High' 'High' 'High' 'Low' 'Low' 'High'
 'Medium' 'Low' 'Low' 'Low' 'High' 'Medium' 'High' 'High' 'High' 'Low'
 'High' 'Medium' 'High' 'Low' 'Medium' 'High' 'Low' 'High' 'High' 'Low'
 'Medium' 'Medium' 'Low' 'High' 'High' 'High' 'Low' 'Low' 'Medium' 'High'
 'Medium' 'High' 'High' 'High' 'High' 'High' 'High' 'Medium' 'Medium'
 'Medium' 'Medium' 'High' 'High' 'Low' 'High' 'High' 'Medium' 'High'
 'High' 'High' 'Low' 'Medium' 'Medium' 'High' 'High' 'High' 'High' 'High'
 'Low' 'Medium' 'Medium' 'Low' 'Low' 'High' 'High' 'Low' 'Medium' 'Low'
 'Low' 'Medium' 'High' 'High' 'Medium' 'High' 'Medium' 'Medium' 'Medium'
 'Low' 'High' 'Low' 'High' 'High' 'High' 'Medium' 'Medium' 'Low' 'Medium'
 'Medium' 'High' 'Medium' 'Low' 'Low' 'Medium' 'High' 'High' 'Medium'
 'High' 'High' 'High' 'Low' 'High' 'High' 'High' 'Low' 'Low' 'High' 'High'
 'High' 'Low' 'High' 'Medium' 'Medium' 'Medium' 'High' 'Medium' 'Low'
 'Medium' 'Low' 'Low' 'Low' 'High' 'Medium' 'High' 'High' 'Medium'
 'Medium' 'Low' 'Medium' 'High' 'High' 'High' 'Medium' 'High' 'Low'
 'Medium' 'Low' 'Medium' 'Low' 'Low' 'Medium' 'High' 'Medium' 'Low' 'High'
 'High' 'Low' 'Medium' 'High' 'High' 'Medium' 'High' 'High' 'Medium'
 'High' 'High' 'Low' 'High' 'High' 'High' 'High' 'High' 'Medium' 'Low'
 'Medium' 'High' 'Medium' 'Low' 'Medium' 'High' 'High' 'Medium' 'High'
 'Medium' 'Medium' 'Medium' 'Medium' 'High' 'Low' 'High' 'Medium' 'Medium'
 'Medium' 'Low' 'Medium' 'High' 'Medium' 'Medium' 'High' 'Medium' 'High'
 'Low' 'Low' 'Medium' 'Medium' 'Low' 'High' 'High' 'High' 'High' 'Medium'
 'Low' 'Low' 'High' 'Low' 'Medium' 'High' 'Low' 'High' 'Medium' 'Medium'
 'Low' 'Medium' 'Low' 'Low' 'High' 'High' 'Medium' 'High' 'High' 'High'
 'Low' 'Medium' 'Low' 'High' 'High' 'Low' 'Low' 'Medium' 'Low' 'High'
 'Low' 'Medium' 'Medium' 'Low' 'High' 'Medium' 'High' 'High' 'Low' 'High'
 'Low' 'High' 'Medium' 'Low' 'Medium' 'High' 'Medium' 'Medium' 'Medium'
 'High' 'High' 'Medium' 'Medium' 'Medium' 'High' 'Medium' 'Medium' 'High'
 'Medium' 'Low' 'High' 'Medium' 'High' 'Medium' 'High' 'Medium' 'Low'
 'High' 'Low' 'Medium' 'High' 'Medium' 'Medium' 'High' 'Medium' 'Low'
 'Low' 'High' 'High' 'High' 'High' 'High' 'Low' 'Low' 'Medium' 'Low'
 'Medium' 'Medium' 'Medium' 'High' 'Medium' 'High' 'Low' 'Medium' 'Medium'
 'Medium' 'High' 'Medium' 'Medium' 'High' 'Low' 'Low' 'Medium' 'Low'
 'High' 'High' 'Medium' 'Low' 'Medium' 'High' 'High' 'Low' 'Medium'
 'Medium' 'Medium' 'Low' 'High' 'Medium' 'Medium' 'Low' 'Low' 'Medium'
```

```
'High' 'Low' 'Medium' 'High' 'Low' 'High' 'Low' 'Low' 'High' 'High'
'Medium' 'Low' 'Low' 'Low' 'High']
```

Task 3



- From the graph above, it seems that the the cumulative frequencies seem to converge on specific values of probability. Both High and Medium demand curves seem to converge just above and below 0.4, while the Low demand curve seems to converge at around 0.2. This is relatively close the probabilities as obtained from the hand calculation below, demonstrating that the simulation accurately represents the underlying probability distribution.



$$P(\text{High}) = (0.5 \cdot 0.6) + (0.3 \cdot 0.3) + (0.2 \cdot 0.1) = 0.41$$

$$P(\text{Medium}) = (0.5 \times 0.3) + (0.3 \times 0.5) + (0.2 \times 0.4) = 0.38$$

$$P(\text{Low}) = (0.5 \times 0.1) + (0.3 \times 0.2) + (0.2 \times 0.5) = 0.21$$

Task 4

```
import numpy as np
import matplotlib.pyplot as plt
from question1 import expected_utility

def sim_eco_conditions(N):
    """
    Simulate the economic conditions for N samples and return the appropriate
    cumulative emperical proportions.
    :param N: number of samples
    :return:
        high_demand_cum_freqs: array containing the emperical proportions for
                              high market demand
        medium_demand_cum_freqs: array containing the emperical proportions for
                                medium market demand
        low_demand_cum_freqs: array containing the emperical proportions for
                             low market demand
    """
    # Define the probability distributions
    economic_probs = {'Good': 0.5, 'Moderate': 0.3, 'Poor': 0.2}

    demand_probs = {
        'Good': {'High': 0.6, 'Medium': 0.3, 'Low': 0.1},
        'Moderate': {'High': 0.3, 'Medium': 0.5, 'Low': 0.2},
        'Poor': {'High': 0.1, 'Medium': 0.4, 'Low': 0.5}
    }

    # Step 1: Simulate economic conditions
    economic_conditions = np.random.choice(list(economic_probs.keys()),
                                           size=N, p=list(economic_probs.values()))

    # Step 2: Simulate market demand scenarios based on economic condition
    demand_probs_given_eco = np.array([demand_probs[condition]
                                       for condition in economic_conditions])
    demand_scenarios = np.array(
        [np.random.choice(list(demand_probs_given_eco[a].keys()),
                          p=list(demand_probs_given_eco[a].values())) for a in
         range(N)])

    # Initialize counters for each market demand outcome
    demand_counters = {'High': 0, 'Medium': 0, 'Low': 0}
```

```
    # Initialize arrays to store the cumulative empirical frequencies for each outcome
    high_demand_cum_freqs = np.zeros(N)
    medium_demand_cum_freqs = np.zeros(N)
    low_demand_cum_freqs = np.zeros(N)

    # Update the counters and compute the empirical frequencies for each outcome
    for i in range(1, N+1):
        demand_counters['High'] += np.sum(demand_scenarios[:i] == 'High')
        demand_counters['Medium'] += np.sum(demand_scenarios[:i] == 'Medium')
        demand_counters['Low'] += np.sum(demand_scenarios[:i] == 'Low')


        high_demand_emp_freq = demand_counters['High'] / \
                               sum(demand_counters.values())
        medium_demand_emp_freq = demand_counters['Medium'] / \
                                 sum(demand_counters.values())
        low_demand_emp_freq = demand_counters['Low'] / \
                              sum(demand_counters.values())

        # Update the cumulative empirical frequencies
        high_demand_cum_freqs[i - 1] = high_demand_emp_freq
        medium_demand_cum_freqs[i - 1] = medium_demand_emp_freq
        low_demand_cum_freqs[i - 1] = low_demand_emp_freq

    # Plot the cumulative empirical frequencies
    x = np.arange(1, N+1)
    plt.plot(x, high_demand_cum_freqs, label='High Demand')
    plt.plot(x, medium_demand_cum_freqs, label='Medium Demand')
    plt.plot(x, low_demand_cum_freqs, label='Low Demand')
    plt.legend()
    plt.xlabel('Number of samples')
    plt.ylabel('Cumulative empirical frequency')
    plt.title('Cumulative empirical frequencies for each market demand outcome')
    plt.show()

    return high_demand_cum_freqs, medium_demand_cum_freqs, low_demand_cum_freqs

if __name__ == "__main__":
    high, med, low = sim_eco_conditions(1000)
    investments = np.array([[200000, 80000, -30000],
                            [150000, 100000, 10000],
                            [0, 0, 0]])
    x = expected_utility(investments, np.array([high[-1], med[-1], low[-1]]))
    print('Expected utility for investing in A: {:.2f}'.format(x[0]))
    print('Expected utility for investing in B: {:.2f}'.format(x[1]))
    print('Expected utility for not investing: {:.2f}'.format(x[2]))
```

```
Expected utility for investing in A: 109850.31
Expected utility for investing in B: 103710.19
Expected utility for not investing: 0.00
```

Task 5

- The optimal decision here is Investing in A since it has the max expected utility

Task 6

- Compared to the scenario where the probabilities were equal, the optimal decision seems to have not have changed in this instance from “Investing in B” to “Investing in A”. In regards to the expected utilities, the expected utility for A in the equal probabilities scenario is approximately 6.9% higher in value than what it is after running the simulation. Similarly, the expected utility for “investing in B” is approximately 3.1% higher in the equal probabilities scenario. However, the expected utility for the “Do not invest” decision remains to the be the same at 0 in both scenarios.
- From the above information, it seems that assuming equal probability, in this case, would be be reasonable. This is because the differences between the results from the equal probabilities scenario and the simulation seem to only be in the range 0% through to 10%. Despite there being differences, there is no change in the optimal decision. This shows that the probabilities of each outcome from the simulation do not differ significantly from the assumed equal probabilities and also, in this case, do not have a significant impact for decision making under risk. To obtain more accurate results, it may be better to conduct another simulation with much more samples.

Question 3

Task 1

$$\frac{d}{d\theta}\left[\sum_{i=1}^{N} w_i(\theta - x_i)^2\right] = 0$$

$$\Rightarrow \sum_{i=1}^{N} 2w_i(\theta - x_i) = 0$$

$$\Rightarrow 2\sum_{i=1}^{N} w_i(\theta - x_i) = 0$$

$$\Rightarrow \sum_{i=1}^{N} w_i\theta - w_i x_i = 0$$

$$\sum_{i=1}^{N} w_i\theta - \sum_{i=1}^{N} w_i x_i = 0$$

Since θ is independent of i:

$$\Rightarrow \theta^* \sum_{i=1}^{N} w_i = \sum_{i=1}^{N} w_i x_i$$

$$\Rightarrow \theta^* = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} \quad //$$

Task 2

Initialise $\theta \leftarrow \theta_0$ for some starting guss, set a small fudge parameter $\delta > 0$

repeat
  for $i = 1 : N$ do
    Update weights, $w_i \leftarrow 1/\max\{|\theta - x_i|, \delta\}$
  end for

  Compute new solution, $\theta^* = \dfrac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}$

  Update parameter estimate (decision), $\theta \leftarrow \theta^*$

until θ estimate converged or maximum iterations reached

Task 3

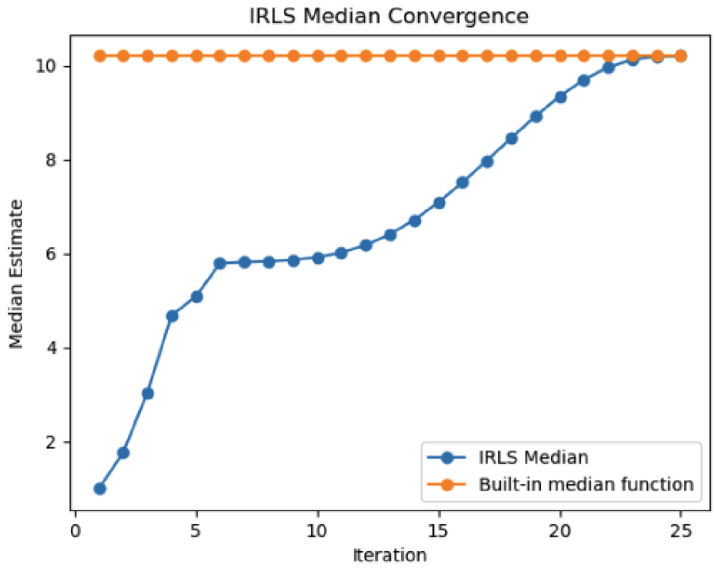
```
def irls_median(x, tol, max_iter, init_guess):
    """
    Implementation of the IRLS function for finding the median of a set of N numbers
    :param x: array of numbers
    :param tol: error tolerance
    :param max_iter: maximum number of iterations
    :param init_guess: initial guess for median
```

```
    :return: array of median estimates calculate via the IRLS method
    """
    # Initialize median estimate
    median_est = list()
    median_est.append(init_guess)

    # Set a small numerical fudge parameter
    delta = 1e-6

    # Iterate until convergence or maximum iterations reached
    median_est_new = median_est[-1]
    for i in range(max_iter):
        # Update weights
        w = np.zeros(len(x))
        for j in range(len(x)):
            w[j] = 1 / np.maximum(np.abs(median_est_new - x[j]), delta)

        # Compute median estimate
        median_est_new = np.average(x, weights=w)

        # Check convergence
        if np.abs(median_est_new - median_est[-1]) < tol:
            break

        # Update median estimate
        median_est.append(median_est_new)

    return np.array(median_est)
```

Task 4

```
if __name__ == "__main__":
    x = [1.1, 4.5, 2.1, 10.2, 16.1, 5.8, 900, 11, 15.6]
    tol = 1e-3
    max_iter = 30
    init_guess = 1
    z = irls_median(x, tol, max_iter, init_guess)
    print(z[-1])
    a = np.repeat(np.median(x),  len(z))


    plt.plot(range(1, len(z) + 1), z, marker='o', label='IRLS Median')
    plt.plot(range(1, len(a) + 1), a, marker='o', label='Built-in median function')
    plt.xlabel('Iteration')
    plt.ylabel('Median Estimate')
    plt.title('IRLS Median Convergence')
    plt.legend()
    plt.show()
```

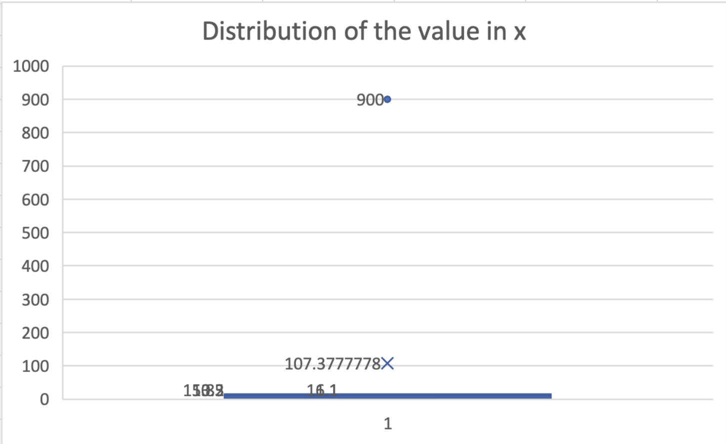
- The result (median) of the array x
    - 10.20 (10.199737909437054)
- Plot of solution



Task 5

- The mean of the array x is 107.38 (107.37777777777778)

Lower quartile	3.3
Upper Quartile	15.85



- This goes to show how skewed the mean is compared to the median. In this case, the 900 value in the array cause the distribution of the data to be right skewed. We can see that the mean of the array, x is much further than the upper quartile of the box and whisker plot. This shows how the mean is impacted by the skewed data. Unlike the mean, the median (10.2) is within the interquartile range.

Task 6

```
def test_median(x, tol):

    """
    Function that takes a vector of numbers and user tolerances and
    checks answer from irls_median against the in-built
    function for the median
    :param x: array of numbers
    :param tol: error tolerance
    """

    # Check if input vector has odd length
    if len(x) % 2 == 0:
        print("Input vector must have odd length!")
        return
```

```
    # Iterate over tolerance values
    for t in tol:
        # Compute median using IRLS
        median_irls = irls_median(x, tol=t, max_iter=30, init_guess=1)[-1]

        # Compute median using np.median
        median_np = np.median(x)

        # Check if difference between medians is within tolerance
        if np.abs(median_irls - median_np) < t:
            print("IRLS and np.median agree (tol={}): {}".format(t, median_irls))
        else:
            print("IRLS and np.median do not agree (tol={}): {}".format(t, median_irls))

if __name__ == "__main__":
    x = [1.1, 4.5, 2.1, 10.2, 16.1, 5.8, 900, 11, 15.6]
    test_median(x, [1e-3, 2e-3, 3e-3, 4e-3, 5e-3, 6e-3, 7e-3, 8e-3])
```

- Results

```
IRLS and np.median agree (tol=0.001): 10.199737909437054
IRLS and np.median agree (tol=0.002): 10.199737909437054
IRLS and np.median agree (tol=0.003): 10.199737909437054
IRLS and np.median agree (tol=0.004): 10.199737909437054
IRLS and np.median agree (tol=0.005): 10.199737909437054
IRLS and np.median agree (tol=0.006): 10.199737909437054
IRLS and np.median agree (tol=0.007): 10.199737909437054
IRLS and np.median agree (tol=0.008): 10.199737909437054
```

Question 4

Task 1

```
def geometric_median(x, tol, max_iter, init_guess):
    """
    Function for finding the geometric median vector for a set of N vectors
    :param x: array of numbers
    :param tol: error tolerance
    :param max_iter: maximum number of iterations
    :param init_guess: initial guess for median
    :return: array of median estimates calculate via the geometric median method
    """

    # Set a small numerical fudge parameter
    delta = 1e-6

    # Iterate until convergence or maximum iterations reached
    median_est_new = init_guess
    for i in range(max_iter):
        # Update weights
        distances = np.maximum(np.sqrt(np.sum((median_est_new - x)**2, axis=1)), delta)
        w = 1.0 / distances

        w /= np.sum(w)

        median_est_new_2 = np.average(x, axis=0, weights=w)

        # Check convergence
        if np.all(distances < tol):
            break

        # Update median estimate
        median_est_new = median_est_new_2

    return np.array(median_est_new)
```

Task 2

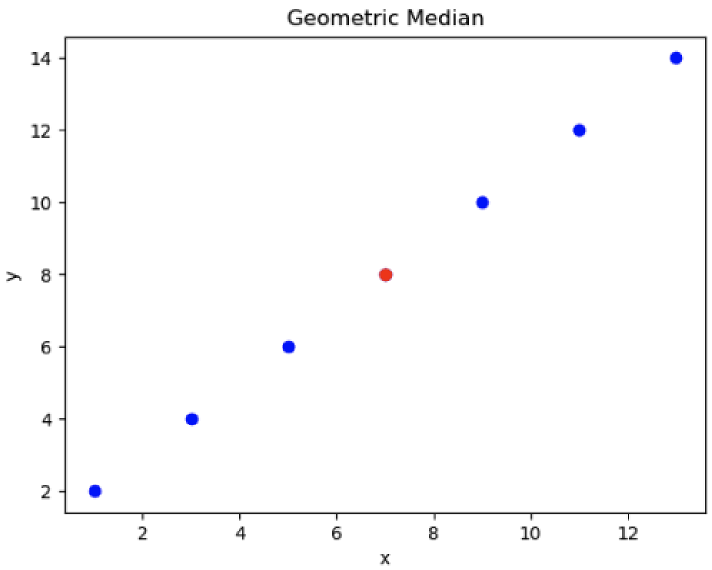
```
if __name__ == '__main__':
    x = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9,10], [11,12], [13,14]])
```

```
    tol = 1e-3
    max_iter = 30
    init_guess = [1,2]

    median_estimates = geometric_median(x, tol, max_iter, init_guess)
    print(median_estimates)

    # plot the input vectors and the geometric median estimate
    plt.scatter(x[:, 0], x[:, 1], c='blue')
    plt.scatter(median_estimates[0], median_estimates[1], c='red')
    plt.title("Geometric Median")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```

- Median estimates

```
[7. 8.]
```

- plot of geometric median estimate



Task 3

```
if __name__ == '__main__':
    x_2 = np.array([[1.1], [4.5], [2.1], [10.2], [16.1], [5.8], [900], [11], [15.6]])
    tol_2 = 1e-3
    max_iter_2 = 30
    init_guess_2 = [1]

    median_estimates_2 = geometric_median(x_2, tol_2, max_iter_2, init_guess_2)
    print(median_estimates_2)
```

- Median estimate

```
[10.2]
```

- This shows that the code gives the same answer as the code from the previous problem