PROBABILITY AND RANDOM PROCESSES

EE325 PROGRAMMING ASSIGNMENT - 1

Aminesh Gogia - 23B1210
Rushabh Gayakwad - 23B2464
Jaswin Reddy M - 23B1289

Part-1

1. Is the opinion from the 100 people representative of the opinion of India?

- The opinion from the 100 people might not be representative of the opinion of India, for considering the large population of the country, this number seems too low to represent the country. The large number of ethnicities, age-groups and variety of job profiles means 100 would be too low a number to capture the opinion of all the people. We also do not have information on how these people were selected. Without knowing the sampling method, it's difficult to determine if the sample is unbiased or not.

2. What is your belief about the home state most IITB students come from?

- Belief of each of the team members regarding the home state of most number of students:
    1) Aminesh: Andhra Pradesh
    2) Rushabh: Telangana
    3) Jaswin: Telangana

3. How to sample the K students?
If you were allowed to choose the value of K, what value would you choose?
And how sure would you be of the actual values of the average?
What kind of quantitative measure would you use to describe your "sureness of the estimate from the single survey of K samples?

To find the optimal sampling size 'k', we generate multiple random datasets and evaluate various values of 'k' by sampling students and predicting the top three states. The goal is to maximize prediction accuracy by sampling the least possible number of students.

Procedure

1. Dataset Generation:

We randomly generate datasets where students are assigned to states; each dataset has different, randomly generated relative representations from states.

2. Sampling and Prediction:

- For each dataset, we test different 'k' values : 10, 20, 50, 100, 150, 200, 250.
- For each 'k', we simulate 50 surveys by sampling 'k' students each time.

- We identify the top three states that most frequently appear across the 50 surveys. We predict these states to be the top three states.

3. Correctness Evaluation:

- We compare the predicted top three states with the actual top three states (ignoring the order).
- We count the number of correct predictions where the predicted top three states match the actual top three states.
- We calculate the fraction of students from the top three states in those surveys where the predicted top three states were indeed correct (thus discarding info from the 'faulty' experiments).

Correctness

The **Correctness** is measured by counting the number of correct predictions:

$$\text{Correctness} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

A correct prediction occurs when the predicted top three states match the actual top three states, regardless of their order.

4. Confidence Measure:

- **Proportional Term**: This increases confidence based on the number of surveys that predicted the correct top three states.
- **Relative Arrangement**: This adjusts confidence by considering the accuracy of the order of the top three states.

$$\text{Confidence} = \frac{\text{Number of surveys with correct top 3}}{50} + \frac{1}{1 + \sum_{j=1} \left(\text{Rank}_{\text{predicted}}(j) - j\right)^2}$$

where:

- The first term represents the proportion of surveys that predicted the correct top three states.
- The second term adjusts confidence based on the accuracy of the ranking of the top three states, with lower penalties for more accurate rankings.

5. Cost Calculation:

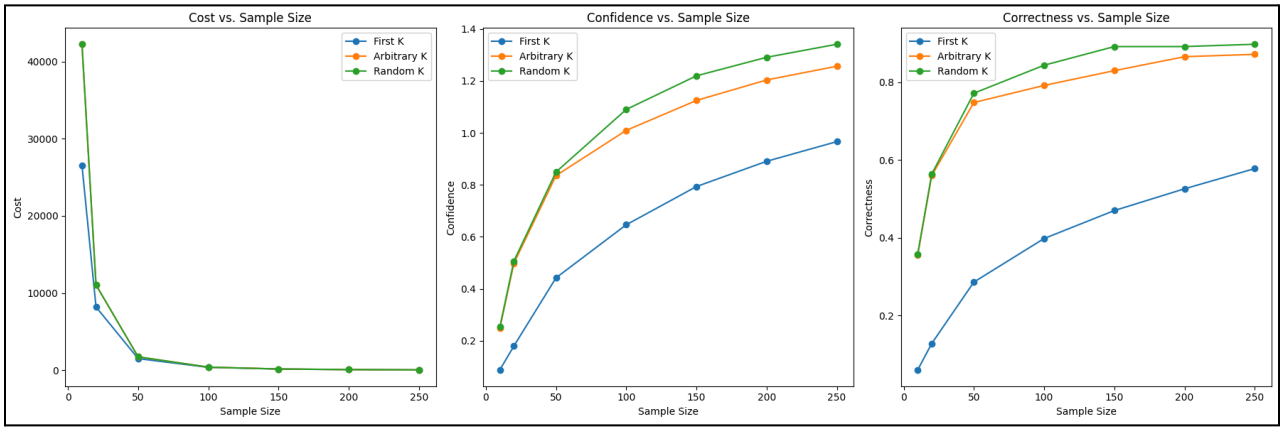
$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{3}(\text{Rank}_{\text{predicted}}(j) - j)^2 + 10 \times (\text{Predicted fraction of top 3} - \text{Actual fraction of top 3})^2 + (\frac{50 - \text{Correct Predictions}}{50})\right)$$

where:

- $\text{Rank}_{\text{predicted}}(j)$ is the predicted rank of the j^{th} state.
- n is the total number of datasets.
- The second term penalizes the difference between the predicted fraction of students from the top three states and the actual fraction.
- A correct prediction occurs when the predicted top three states match the actual top three states, regardless of their order

We calculate cost as a penalty for incorrect predictions and deviations in the fraction of students from the top three states.
Finally, we average the confidences, costs and correctness over all the

- We chose the minimum possible 'k' that makes good predictions for the top three states and the number of students from the top three states.

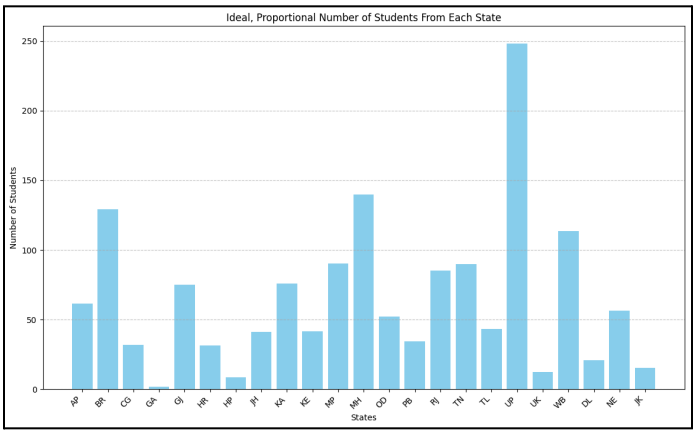


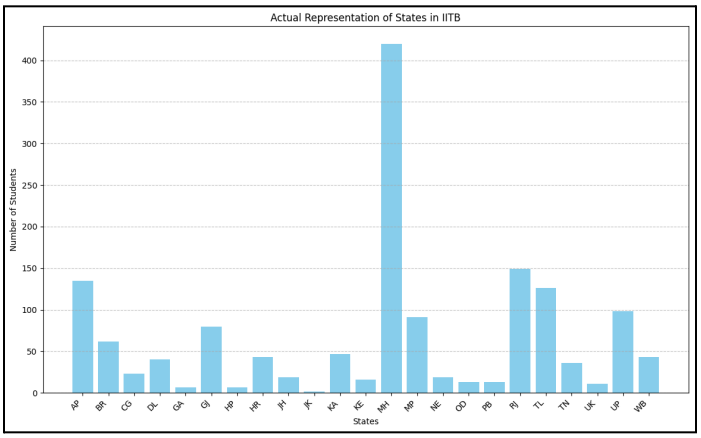
- With this **Confidence measure**, we can comment on how sure we are with regards to the prediction made using a given ‘k’, a **relative comparison** can be done between k’s by running this program using multiple k values on the same datasets.
- Seeing that the graph of accuracy and confidence gradually saturates with ‘k’, we can choose the ‘k’ to be 150 or 200. 200 offers marginally greater confidence, but surveying 50 more students to get a similar accuracy does not seem a better option.
- So, **we go for sampling 150 students**, and follow a similar data handling scheme as above; where we discard surveys where the top three predicted states are not the same as the cumulatively predicted top-three states.
- Choosing the sampling method: **Random Selection (Method - 3)** seems the best option as when comparing at the same ‘k’, cost is the least for it, and the belief and accuracy is the most. The next best method is Arbitrary Selection (Method - 2)

Part 2

1) **Is the IITB UG population a good sample of India at the granularity of states? If not, how badly is it skewed in favor of some states? Define your own measure for skew.**

   No, IIT Bombay is not a good sample of India at the granularity of states. Plotting what should have been the population proportionate representation of the states, and how the distribution actually is:

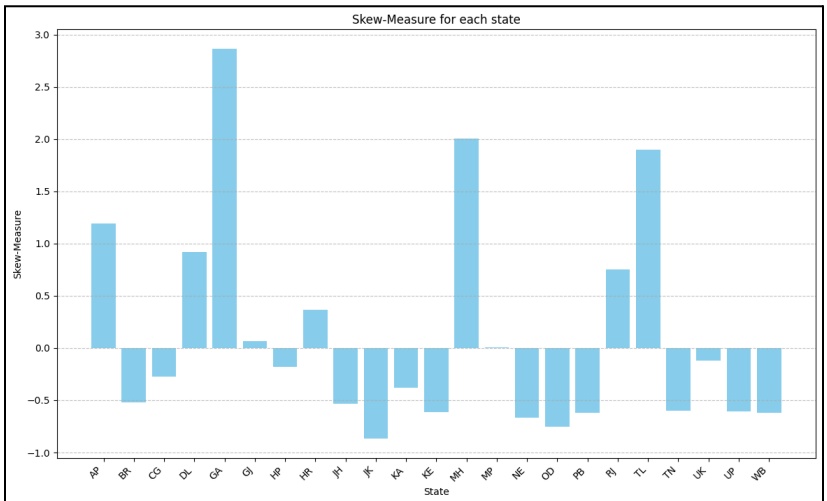




We define a measure for skew-fairness:

$$skewfairness(state) = \frac{(actual - ideal)}{ideal}$$

Where *actual* represents the number of students from a particular state, while *ideal* is the population-proportional number of students from the state. Plotting the skew measure of the states, we get:



The population weighted average of absolute values of skew-fairness of states comes out to be 0.722, which is representative of 72.2% skewfairness on an average across the country. Therefore the IITB population is not a good sample of the granularity of the states of the country. It is unfairly skewed in favour of states: Maharashtra, Rajasthan, Andhra Pradesh, Telangana and Goa.

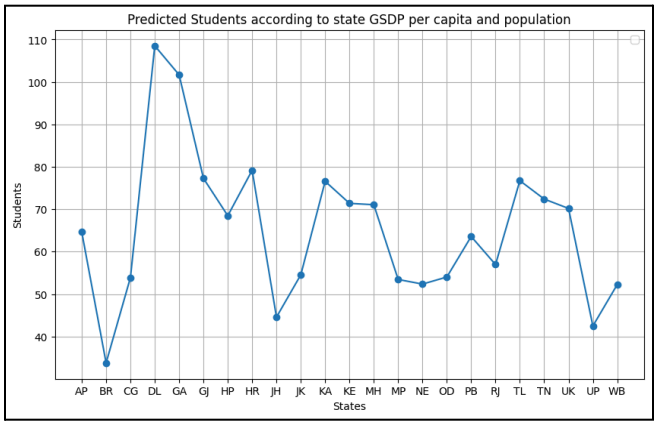
2) **Considering the population and the per capita income of the states, is the distribution of the student body among the states/regions fair? Once again, define your own measure for skewfairness and apply to the data that you have. Justify the measure.**

The measure we defined in this case was again:

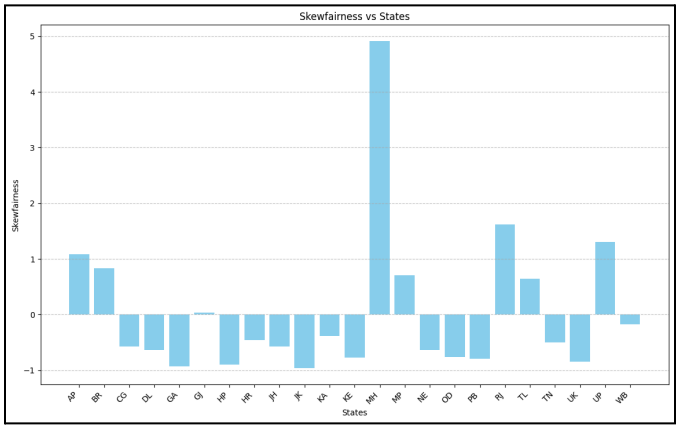
$$skewfairness(state) = \frac{(actual - ideal)}{ideal}$$

Here, *ideal* is different from the proportionate representation.

We took *ideal* to be proportional to the square root of the GSDP (Per Capita) of the state, and proportional to its population. The reason to take square root was because the perception of our team that the marginal increase in number of students with the per capita GSDP should eventually decrease. The ideal representation looked like:



The skewfairness measure plotted for each state was:

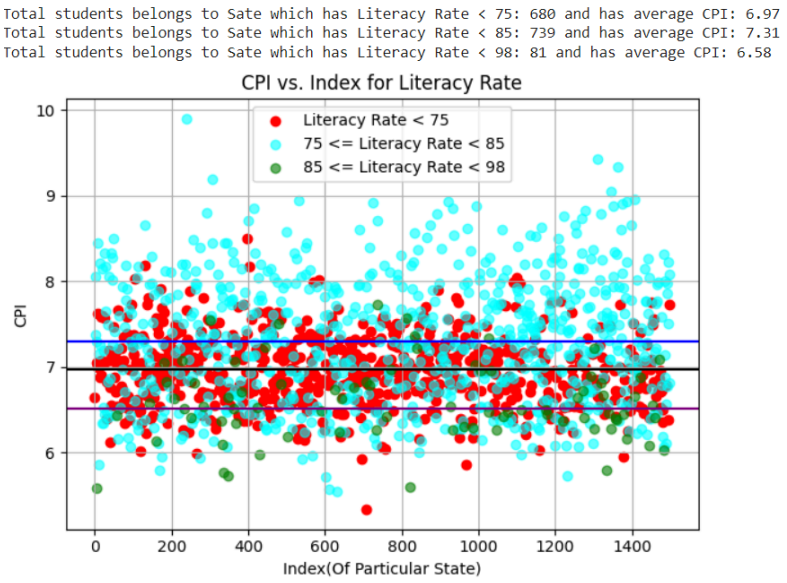


Even in this case, it appears that the IITB student dataset is not fairly distributed according to our model. Again, the stats are skewed in favour of Maharashtra, Andhra Pradesh, Telangana and Rajasthan; but they are also in favour of UP and Bihar, which expected low representation due to low Per Capita GSDP. There are several states having negative skew-fairness values, again. The population weighted average of absolute values of skew-fairness of states comes out to be 1.167 which is representative of 116.7% skewfairness on an average across the country.

3. How strongly does the state affect the JEE rank, the graduating CPI, and the first salary?

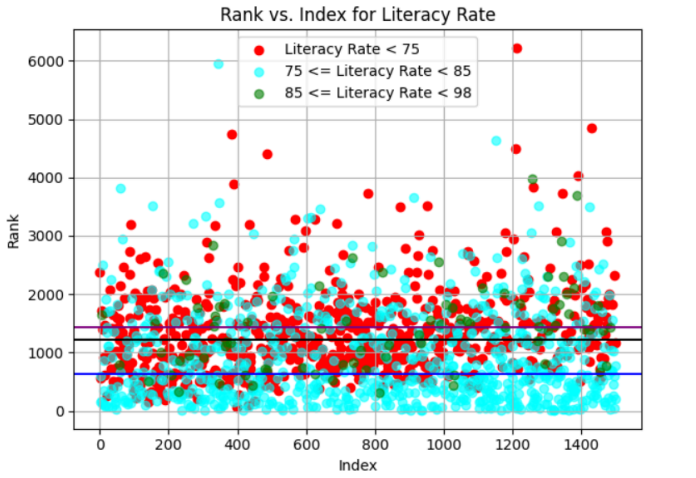
Effect of State Literacy

First Case: In our opinion, the home state's literacy affects the graduating CPI. The majority of the students having higher CPI, belong to the states with 75 - 85% literacy. Also, the average CPI of students from the highest literacy states is higher than the students of the lowest CPI band. The number of students from states in the middle Literacy band is higher than the students in the lowest band.

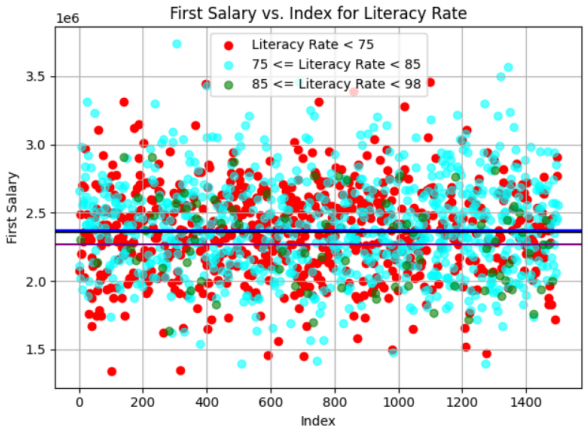


Second Case: Literacy Rate vs Rank: The top ranked students belong to the states with 75-85% literacy. Students from states with higher literacies have a better average rank than the lower ranked students.

```
Total students belongs to Sate which has Literacy Rate < 75: 680 and has avg Rank: 1345
Total students belongs to Sate which has Literacy Rate < 85: 739 and has avg Rank: 894
Total students belongs to Sate which has Literacy Rate < 98: 81 and has avg Rank: 1449
```



<u>Third Case:</u> First Salary vs Literacy Rate: The first salaries of students from different literacy states have similar averages. Also the spread in the first salaries across states is pretty similar for low and middle literacy states. So, no major impact of literacy rate of home state on first salary.



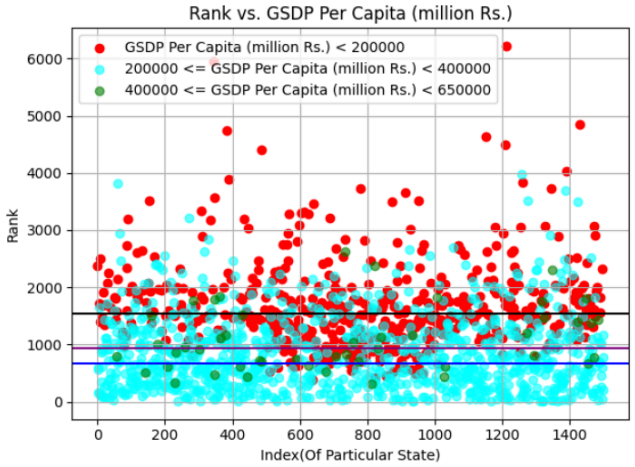
<u>Effect of STATE GSDP Per Capita</u>

<u>First Case: (CPI vs GSDP Per Capita)</u>

Students from middle GSDP per capita states form the majority of the students, have a better average CPI, and even the higher scoring students belong to these states. The

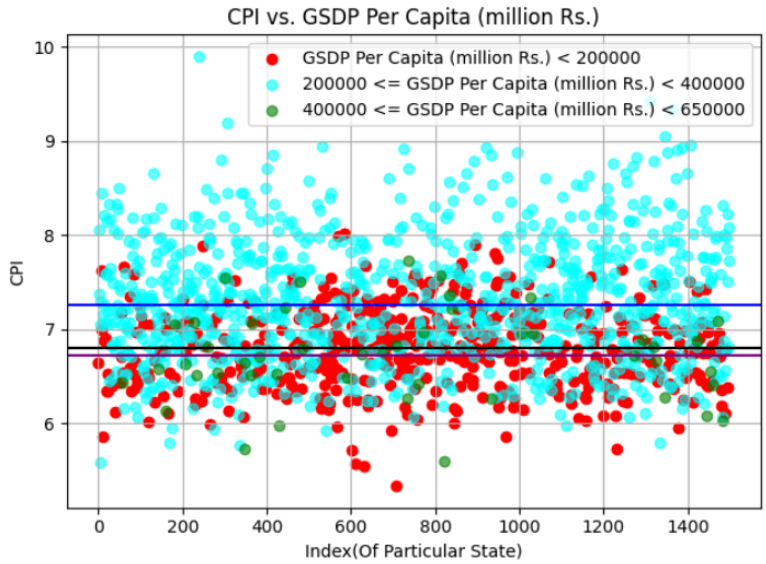
other two GSDP per capita bands have a similar average CPI, but cannot comment about their comparison, as the number of students from higher Per Capita GSDP states is very less.

```
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 200000: 519 and has avg Rank: 1682
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 400000: 934 and has avg Rank: 823
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 650000: 47 and has avg Rank: 1094
```



Second Case:

```
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 75: 519 and has average CPI: 6.80
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 85: 934 and has average CPI: 7.31
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 98: 47 and has average CPI: 6.72
```

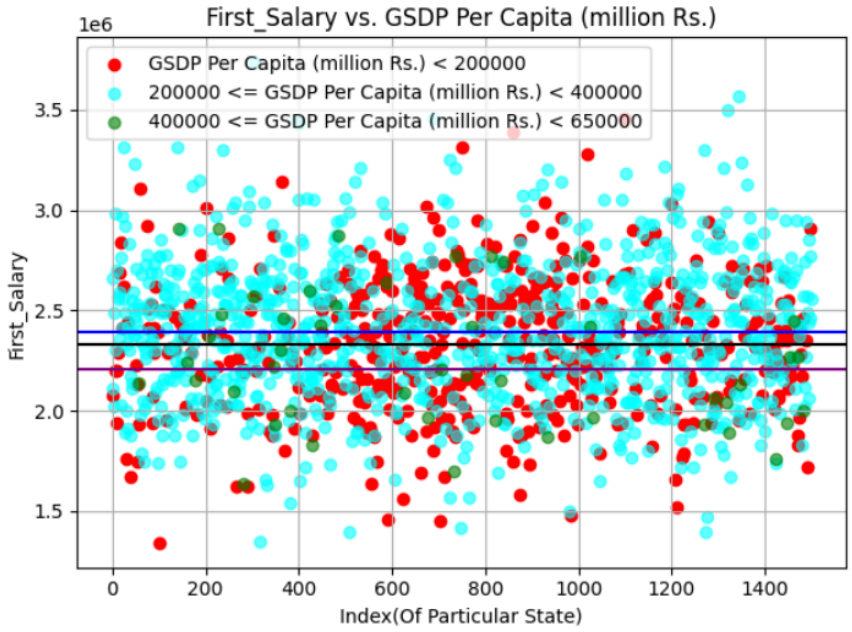


Better ranked students belong to the middle Per Capita GSDP states. The lower ranked students primarily belong to the low Per Capita GSDP states. The high per capita GSDP students are mostly in the 1000-2000 ranked range.

Third Case:

The first salaries of students from different Per Capita GSDP states have similar averages. Also the spread in the first salaries across states is pretty similar for low and middle per capita GSDP states. So, no major impact of literacy rate of home state on first salary.

```
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 200000: 519 and has First_Salaryk: 2329479.77
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 400000: 934 and has First_Salary: 2403179.87
Total students belongs to Sate which has GSDP Per Capita (million Rs.) < 650000: 47 and has First_Salary: 2255957.45
```



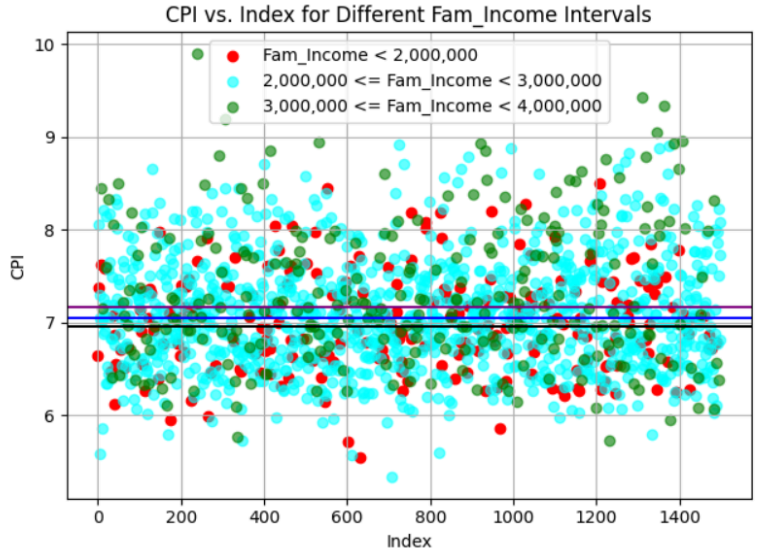
From the above we can conclude that middle literacy and middle per capita GSDP states produce better ranked students, students with higher CPI. There is no effect as such of the state on the first salary.

4. How strongly does the family income affect the JEE rank, the graduating CPI, and the first salary?

In this case, I took Family Income as the basis for the calculations.

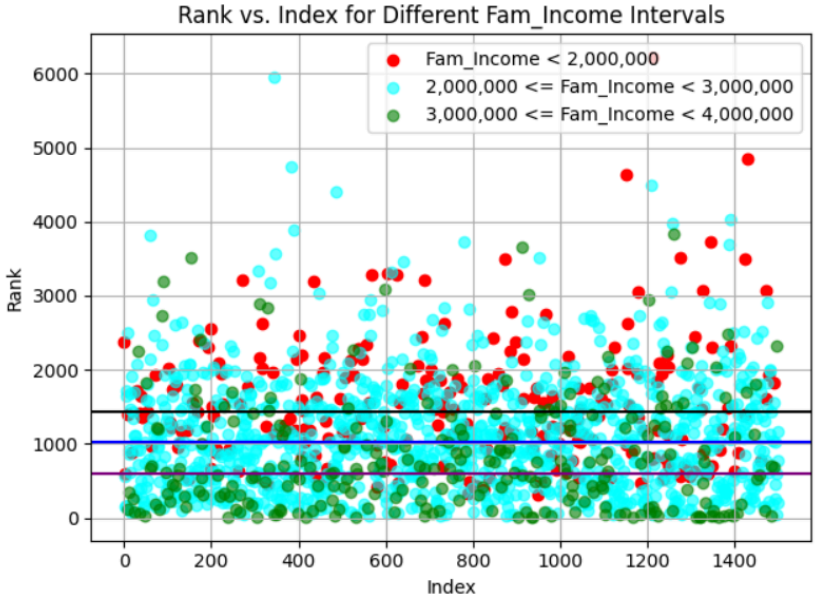
First Case: CPI increases slightly with the Family Income Levels from low to mid-income families; there is a major jump being from mid-income to high-income families. Higher CPI students belong to the mid-income and high-income families.

```
Total students has fam_income less than 2000000: 233 and has average CPI: 7.00
Total students has fam_income >= 2000000 and <= 3000000: 1011 and has average CPI: 7.09
Total students has fam_income >= 3000000 and <= 4000000: 256 and has average CPI: 7.32
```



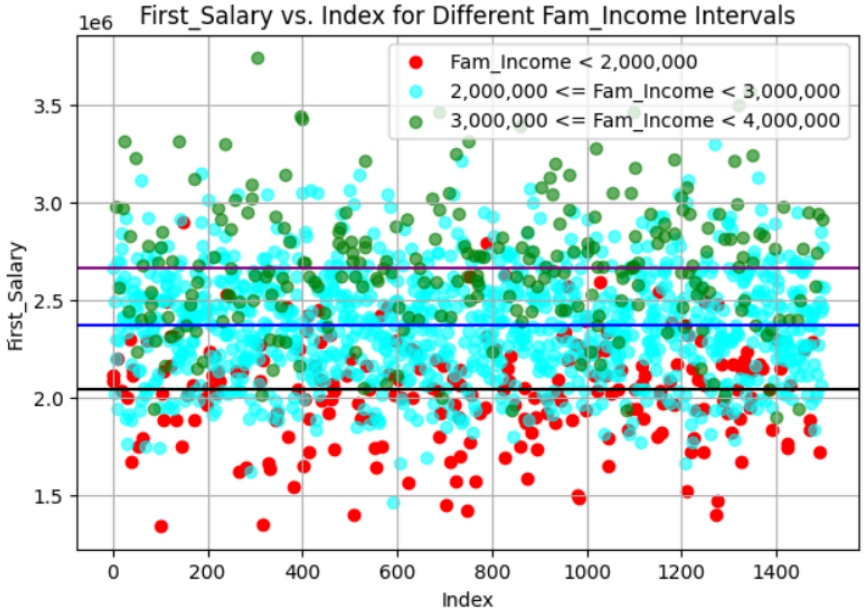
Second Case: The JEE rank improves as the family income improves, as can be seen directly from the plot.

```
Total students has fam_income less than 2000000: 233 and has average of Rank: 1542
Total students has fam_income >= 2000000 and <= 3000000 : 1011 and has average of Rank: 1112
Total students has fam_income >= 3000000 and <= 4000000: 256 and has average of Rank: 819
```



Third Case: First Salary improves with an increase in the family income:

```
Total students has fam_income less than 1500000: 233 and has avg First Salary around: 2025751.07
Total students has fam_income less than 1500000: 1011 and has avg First Salary around: 2377329.38
Total students has fam_income less than 1500000: 256 and has avg First Salary around: 2672343.75
```



Therefore, Family Income overall has a positive impact on JEE Rank, graduating CPI and First Salary.

What follows is the code, **which contains the sampling scatter plots.**

prog-assignment-1-1

September 1, 2024

[4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
```

[5]:
```
## State Information

state_info = pd.read_csv('StateInfo.csv')
state_info = state_info.to_dict()

num_of_states = len(state_info)
```

[6]:
```
## Student Information

data = pd.read_csv('JEEDemographics.csv')

home_states = data['Origin']
state_counts = home_states.value_counts()
top_states = state_counts.head(3).index.tolist()
top_states_students = state_counts.head(3).sum()
num_students = len(home_states)
state_to_rank = {state: rank for rank, state in enumerate(top_states)}

top_states_Frac = top_states_students / num_students
homeStates = home_states.tolist()

state_counts_dict = state_counts.to_dict()
```

0.1 The Sampling Functions

[7]:
```
def sample_firstK(k, custom = None):
    # Select the first k in the roll list
    if custom is None:
        return home_states[:k]
    return custom[:k]
```

```
def sample_arbitraryK(k, custom = None):
    # Select a random point in the roll list, then k consecutive students from
 ↪that position
    if custom is None:
        if k <  1500:
            start = np.random.randint(0, len(home_states) - k)
            return home_states[start:start + k]
        return home_states
    if k <  1500:
            start = np.random.randint(0, len(home_states) - k)
            return custom[start:start + k]
    return custom

def sample_randomK(k, custom = None):
    # Select k random students from the roll list
    if custom is None:
        return np.random.choice(home_states, k, replace=False)
    return np.random.choice(custom, k, replace=False)
```

0.2 The Scatter Plots

[8]:
```
def calculateFrac(sample):
    counts = pd.Series(sample).value_counts()
    req_states = counts.nlargest(3)
    req_fraction = req_states.sum() / len(sample)
    return req_states.index.tolist(), req_fraction

def check(predStates, realStates):
    return set(predStates) == set(realStates)
```

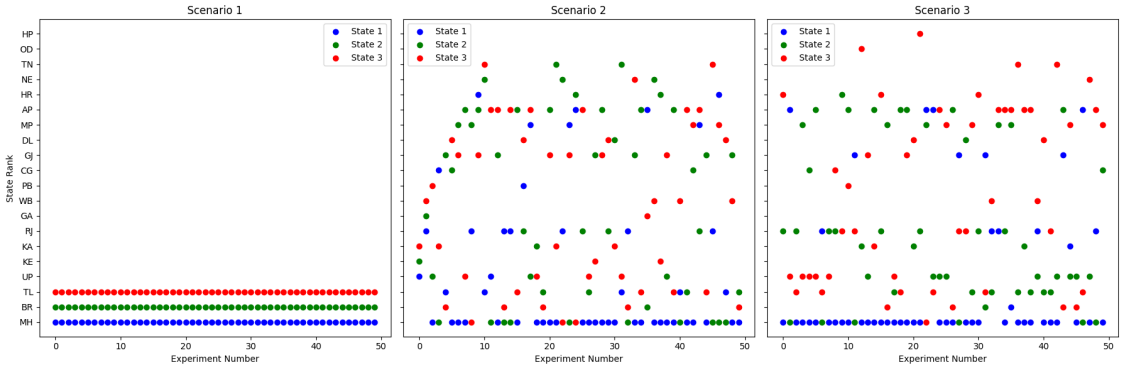
[9]:
```
import random
fractions = state_counts / len(home_states)

def top_three_fractions(sample):
    state_counts = pd.Series(sample).value_counts()
    top_three = state_counts.head(3)
    fractions = top_three / len(sample)
    return fractions.index.tolist(), fractions.tolist()

def collect_samples(home_states, K, scenario):
    top_states = []
    fractions_list = []

    for _ in range(50):
        if scenario == 1:
            # Scenario 1: First K students
            sample = home_states[:K]
```

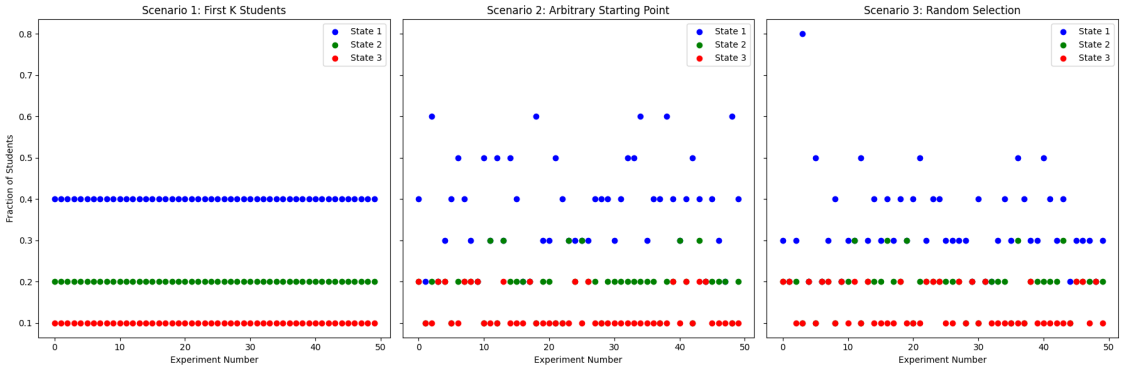
```
        elif scenario == 2:
            # Scenario 2: Arbitrary starting point
            start = random.randint(0, len(home_states) - K)
            sample = home_states[start:start + K]
        elif scenario == 3:
            # Scenario 3: Random selection
            sample = random.sample(home_states.to_list(), K)

        states, fractions = top_three_fractions(sample)
        top_states.append(states)
        fractions_list.append(fractions)

    return top_states, fractions_list

def plot_scenarios_as_subplots(fractions_list_1, fractions_list_2,␣
 ↪fractions_list_3):
    # Create a figure with 3 subplots (1 row, 3 columns)
    fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
    colors = ['blue', 'green', 'red']
    scenarios = [(fractions_list_1, 'Scenario 1: First K Students'),
                 (fractions_list_2, 'Scenario 2: Arbitrary Starting Point'),
                 (fractions_list_3, 'Scenario 3: Random Selection')]

    # Plot each scenario in its corresponding subplot
    for ax, (fractions_list, title) in zip(axes, scenarios):
        for i, fractions in enumerate(fractions_list):
            for j, fraction in enumerate(fractions):
                if j < len(colors):
                    ax.scatter(i, fraction, color=colors[j], label=f'State␣
 ↪{j+1}' if i == 0 else "")

        ax.set_title(title)
        ax.set_xlabel('Experiment Number')
        ax.set_ylabel('Fraction of Students' if ax == axes[0] else "")
        ax.legend()

    plt.tight_layout()
    plt.show()
```

[10]:
```
top_three_states, fractions = top_three_fractions(home_states)
print("Top Three States:", top_three_states)
print("Fractions:", fractions)

k_values = [10,20,50,100,200]
for K in k_values:
    print("K value:", K)
```

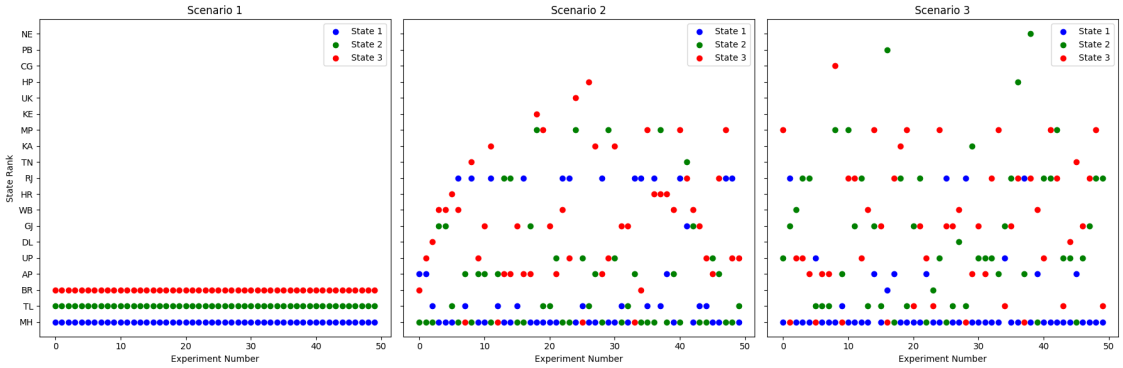
```
   top_states_1, fractions_scenario_1 = collect_samples(home_states, K,␣
↪scenario=1)
   top_states_2, fractions_scenario_2 = collect_samples(home_states, K,␣
↪scenario=2)
   top_states_3, fractions_scenario_3 = collect_samples(home_states, K,␣
↪scenario=3)

   fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
   colors = ['blue', 'green', 'red']
   for ax, top_states, title in zip(axes, [top_states_1, top_states_2,␣
↪top_states_3], ['Scenario 1', 'Scenario 2', 'Scenario 3']):
       for i, states in enumerate(top_states):
           for j, state in enumerate(states):
               if j < len(colors):
                   ax.scatter(i, state, color=colors[j], label=f'State {j+1}'␣
↪if i == 0 else "")

       ax.set_title(title)
       ax.set_xlabel('Experiment Number')
       ax.set_ylabel('State Rank' if ax == axes[0] else "")
       ax.legend()

   plt.tight_layout()
   plt.show()

   plot_scenarios_as_subplots(fractions_scenario_1, fractions_scenario_2,␣
↪fractions_scenario_3)
```

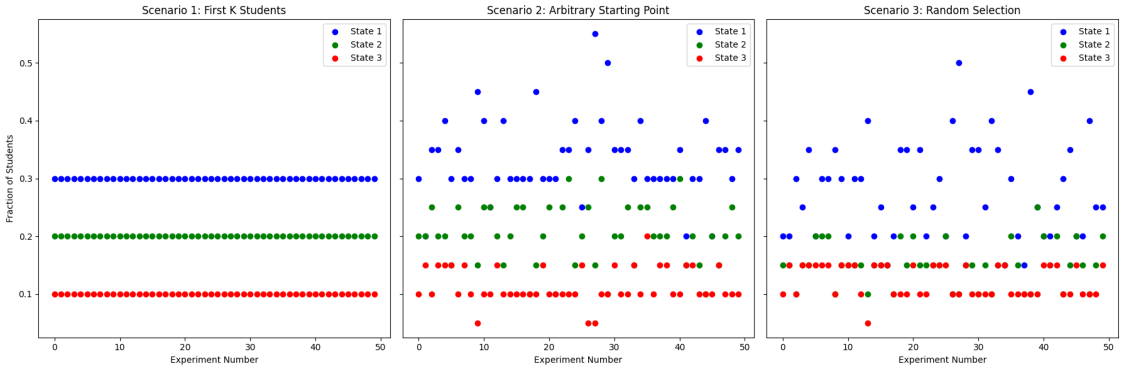
```
Top Three States: ['MH', 'RJ', 'AP']
Fractions: [0.28, 0.09933333333333333, 0.09]
K value: 10
```



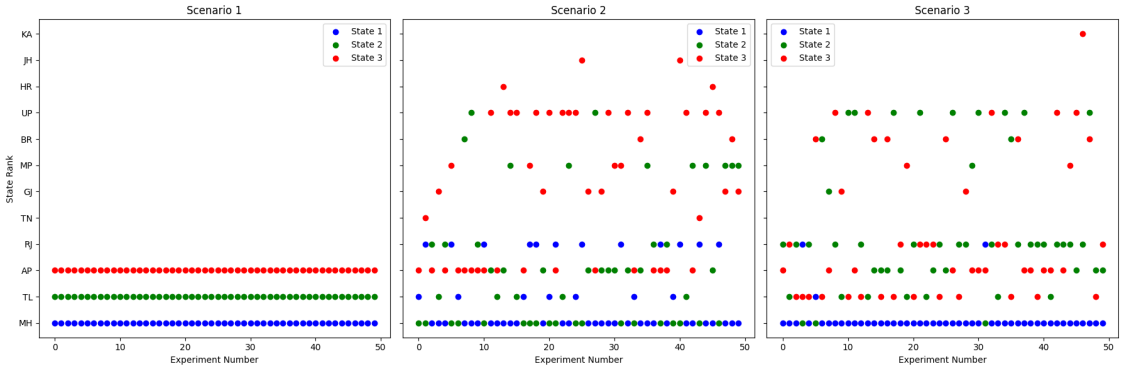


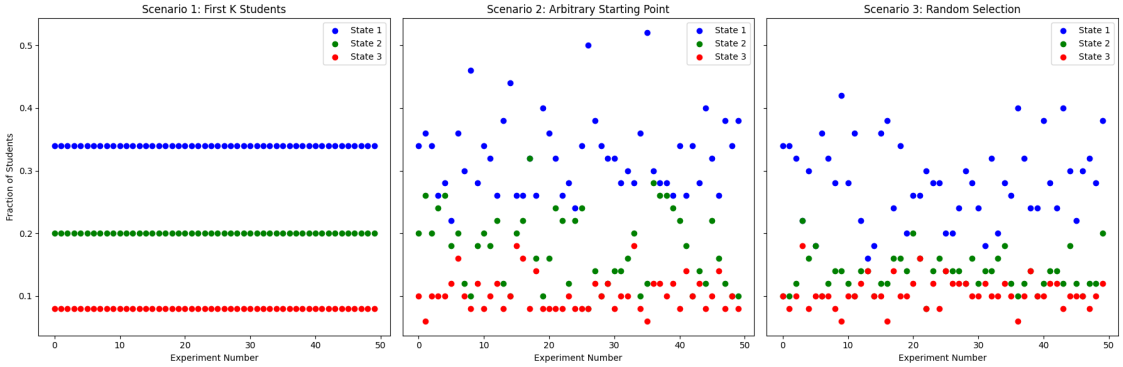
K value: 20



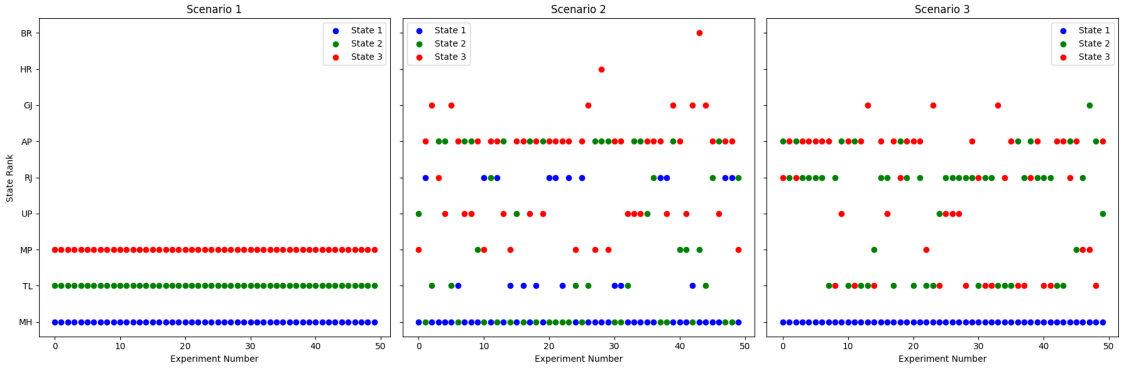


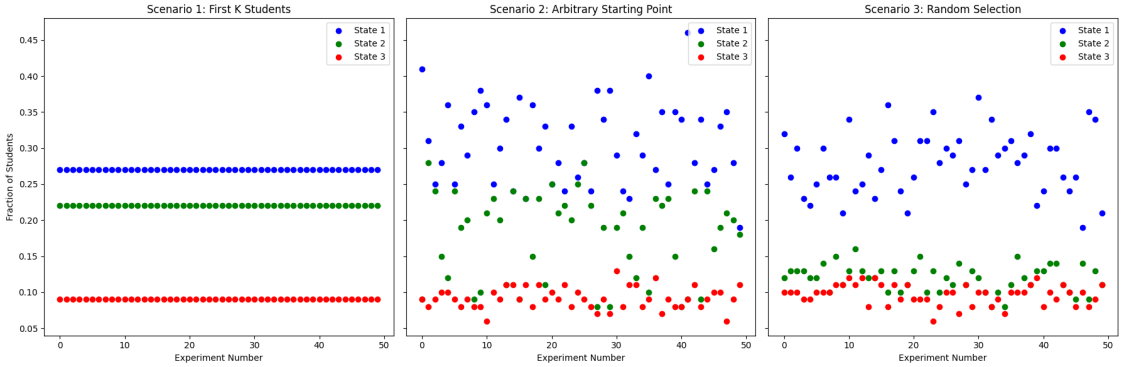
K value: 50


Scenario 1
Scenario 2
Scenario 3
State 1
State 2
State 3
KA
JH
HR
UP
BR
MP
GJ
TN
RJ
AP
TL
MH
State Rank
0
10
20
30
40
50
Experiment Number

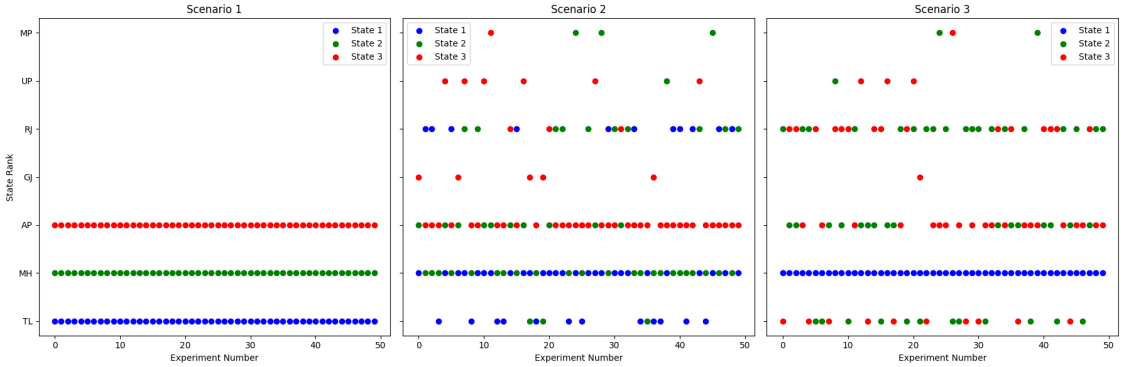

Scenario 1: First K Students
Scenario 2: Arbitrary Starting Point
Scenario 3: Random Selection
State 1
State 2
State 3
0.5
0.4
0.3
0.2
0.1
Fraction of Students
0
10
20
30
40
50
Experiment Number

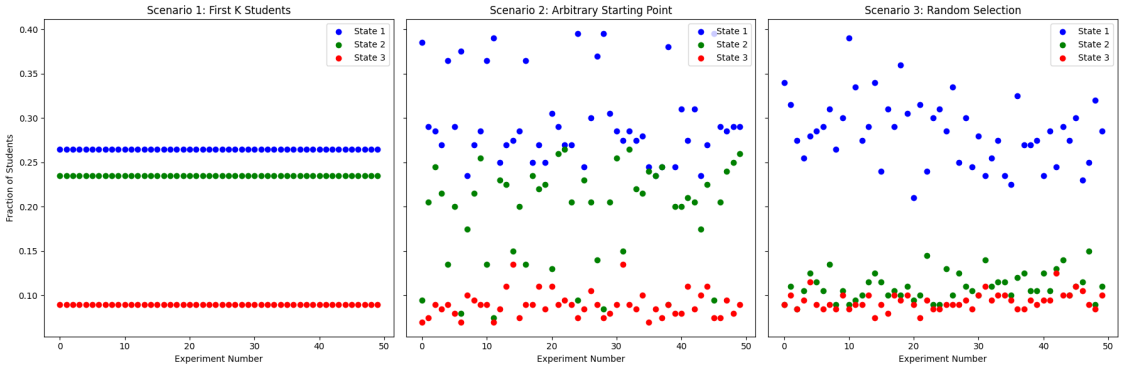
K value: 100


Scenario 1
Scenario 2
Scenario 3
State 1
State 2
State 3
BR
HR
GJ
AP
RJ
UP
MP
TL
MH
State Rank
0
10
20
30
40
50
Experiment Number


Scenario 1: First K Students
Scenario 2: Arbitrary Starting Point
Scenario 3: Random Selection
State 1
State 2
State 3
Fraction of Students
Experiment Number

K value: 200


Scenario 1
Scenario 2
Scenario 3
State 1
State 2
State 3
MP
UP
RJ
GJ
AP
MH
TL
State Rank
Experiment Number


Scenario 1: First K Students
Scenario 2: Arbitrary Starting Point
Scenario 3: Random Selection
State 1
State 2
State 3
Fraction of Students
Experiment Number

0.3 Number of Students From Top Three States

[11]:
```
## Using the sampling function, simply sampling all the students

x = sample_firstK(1500)
predStates, fraction = calculateFrac(x)

print(f'Top three states from which students belong: {predStates}')
print(f'Fraction of students from the top three top states: {fraction}')
```

```
Top three states from which students belong: ['MH', 'RJ', 'AP']
Fraction of students from the top three top states: 0.4693333333333333
```

0.4 Selecting 'k'

0.4.1 Overview

To find the optimal sampling size 'k', we generate multiple random datasets and evaluate various values of 'k' by sampling students and predicting the top three states. The goal is to maximize prediction accuracy by sampling least possible number of students.

0.4.2 Procedure

1. **Dataset Generation**:
   - We randomly generate datasets where students are assigned to states; each dataset has different, randomly generated relative representations from states.
2. **Sampling and Prediction**:
   - For each dataset, we test different 'k' values : 10, 20, 50, 100, 150, 200, 250.
   - For each 'k', we simulate 50 surveys by sampling 'k' students each time.
   - We identify the top three states that most frequently appear across the 50 surveys. We predict these states to be the top three states.
3. **Correctness Evaluation**:
   - We compare the predicted top three states with the actual top three states (ignoring the order).
   - We count the number of correct predictions where the predicted top three states match the actual top three states.
   - We calculate the fraction of students from the top three states in those surveys where the predicted top three states were indeed correct.
4. **Confidence Measure**:
   - **Proportional Term**: This increases confidence based on the number of surveys that predicted the correct top three states.
   - **Relative Arrangement**: This adjusts confidence by considering the accuracy of the order of the top three states.
5. **Cost Calculation**:
   - We calculate cost as a penalty for incorrect predictions and deviations in the fraction of students from the top three states.
6. **Final Evaluation**:
   - We chose the minimum possible 'k' that makes good predictions for the top three states and the number of students from the top three states.

[12]:
```
import random
from collections import Counter
import numpy as np

def generate_student_data(num_students=1500):
    student_assignments = random.choices(tuple(state_counts_dict.keys()),␣
 ↪k=num_students, weights=np.clip(np.random.randn(len(state_counts_dict.
 ↪keys())), a_min=0.1, a_max=None))
    state_counts = {state: student_assignments.count(state) for state in␣
 ↪state_counts_dict.keys()}
    return student_assignments, state_counts

def mode_of_top_three(states_list):
    flattened_list = [state for sublist in states_list for state in sublist]
    most_common_states = [state for state, count in Counter(flattened_list).
 ↪most_common(3)]
    return most_common_states

sampling_funcs = [sample_firstK, sample_arbitraryK, sample_randomK]

corr_per_samp_size = {samp_size: [0, 0, 0] for samp_size in [10, 20, 50, 100,␣
 ↪150, 200, 250]}
cost_per_samp_size = {samp_size: [0, 0, 0] for samp_size in [10, 20, 50, 100,␣
 ↪150, 200, 250]}
confidence_per_samp_size = {samp_size: [0, 0, 0] for samp_size in [10, 20, 50,␣
 ↪100, 150, 200, 250]}
n = 500

random.seed(100)

for i in range(n):
    x, y = generate_student_data()
    desc_states = sorted(y.items(), key=lambda x: x[1], reverse=True)
    rank_to_state = {i: desc_states[i][0] for i in␣
 ↪range(len(state_counts_dict))}
    state_to_rank = {x: y for y, x in rank_to_state.items()}

    actual_num = sum(desc_states[i][1] for i in range(3))

    for samp_size in [10, 20, 50, 100, 150, 200, 250]:
        correct = [0, 0, 0]
        cost = [0, 0, 0]
        conf = [0, 0, 0]

        for j in range(3):
            top_three_states = []
```

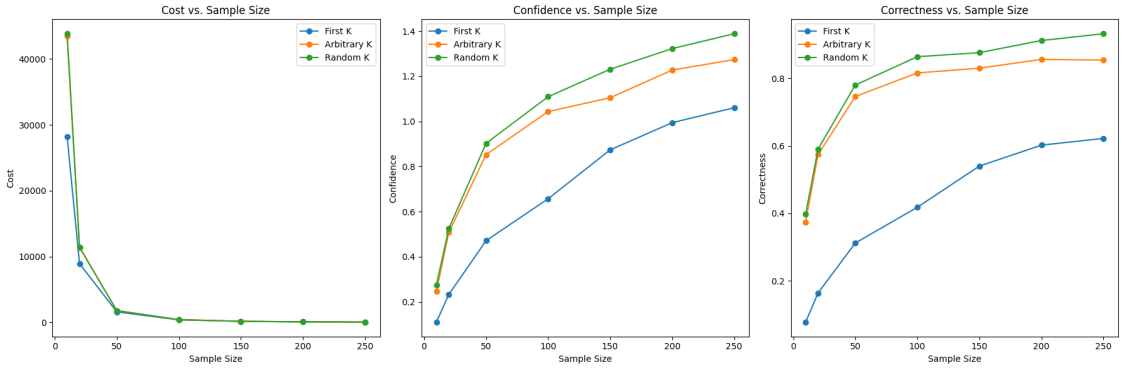
```
            for k in range(50):
                z = sampling_funcs[j](samp_size, x)
                z = {state: list(z).count(state) for state in state_counts_dict.
↪keys()}
                state_v_count_sampled = sorted(z.items(), key=lambda x: x[1],␣
↪reverse=True)
                top_three = [s[0] for s in state_v_count_sampled[:3]]
                top_three_states.append(top_three)

            mode_top_three = mode_of_top_three(top_three_states)
            count_corr_preds = sum(1 for top_three in top_three_states if␣
↪set(top_three) == set(mode_top_three))
            act_ranks_of_pred_top_3 = [state_to_rank[state] for state in␣
↪mode_top_three]

            if set(act_ranks_of_pred_top_3) == {0, 1, 2}:
                correct[j] += 1

                conf[j] += count_corr_preds / 50.0  + 1 /(1 +  sum((rank - i)␣
↪** 2 for i, rank in enumerate(act_ranks_of_pred_top_3)))
            cost_one = sum((rank - i) ** 2 for i, rank in␣
↪enumerate(act_ranks_of_pred_top_3)) + 10 * ((sum(y[state] for state in␣
↪mode_top_three)/samp_size - actual_num/1500) ** 2)
            cost[j] += cost_one

        # Averaging the results across all random datasets
        for j in range(3):
            cost_per_samp_size[samp_size][j] += cost[j]/n + (50 - correct[j])/
↪(50 * n)
            corr_per_samp_size[samp_size][j] += correct[j]/n
            confidence_per_samp_size[samp_size][j] += conf[j]/n

print(cost_per_samp_size)
print(corr_per_samp_size)
print(confidence_per_samp_size)
```

```
{10: [28210.129817991117, 43512.259220657776, 43823.489580657784], 20:
[8928.082893324441, 11384.997801991114, 11426.568002657781], 50:
[1604.4928190577773, 1794.5275507911128, 1798.95047985778], 100:
[393.89780705777827, 422.0827274577777, 422.7161618577775], 150:
[170.42756341333344, 175.86514030222241, 175.836536568889], 200:
[91.75828462444447, 92.66398809111105, 92.55147875777776], 250:
[55.87763111111108, 55.54806876444446, 55.41400369777777]}
{10: [0.07800000000000006, 0.3740000000000003, 0.3980000000000003], 20:
[0.16400000000000012, 0.5740000000000004, 0.5900000000000004], 50:
[0.3120000000000002, 0.7460000000000006, 0.7800000000000006], 100:
```

```
[0.4180000000000003, 0.8160000000000006, 0.8640000000000007], 150:
[0.5400000000000004, 0.8300000000000006, 0.8760000000000007], 200:
[0.6020000000000004, 0.8560000000000006, 0.9120000000000007], 250:
[0.6220000000000004, 0.8540000000000006, 0.9320000000000007]}
{10: [0.10996825396825394, 0.24621206349206348, 0.27445206349206364], 20:
[0.23200000000000004, 0.5069676190476189, 0.523831111111111], 50:
[0.4706666666666664, 0.8521333333333327, 0.9023688888888886], 100:
[0.6565079365079374, 1.043060952380952, 1.1090914285714288], 150:
[0.8733333333333361, 1.1045422222222212, 1.2308292063492074], 200:
[0.994095238095241, 1.227166984126985, 1.3225733333333327], 250:
[1.0603174603174628, 1.2737320634920644, 1.388342222222223]}
```

[13]:
```
# Sample sizes and labels for the sampling methods
sample_sizes = [10, 20, 50, 100, 150, 200, 250]
sampling_methods = ['First K', 'Arbitrary K', 'Random K']

# Plot Cost, Confidence, and Correctness
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharex=True)

# Cost Plot
for i in range(3):
    axes[0].plot(sample_sizes, [cost_per_samp_size[s][i] for s in␣
 ↪sample_sizes], marker='o', label=sampling_methods[i])
axes[0].set_title('Cost vs. Sample Size')
axes[0].set_xlabel('Sample Size')
axes[0].set_ylabel('Cost')
axes[0].legend()

# Confidence Plot
for i in range(3):
    axes[1].plot(sample_sizes, [confidence_per_samp_size[s][i] for s in␣
 ↪sample_sizes], marker='o', label=sampling_methods[i])
axes[1].set_title('Confidence vs. Sample Size')
axes[1].set_xlabel('Sample Size')
axes[1].set_ylabel('Confidence')
axes[1].legend()

# Correctness Plot
for i in range(3):
    axes[2].plot(sample_sizes, [corr_per_samp_size[s][i] for s in␣
 ↪sample_sizes], marker='o', label=sampling_methods[i])
axes[2].set_title('Correctness vs. Sample Size')
axes[2].set_xlabel('Sample Size')
axes[2].set_ylabel('Correctness')
axes[2].legend()

plt.tight_layout()
```

```
plt.show()
```



0.5 Is the IITB Population a Good Sample of India at the Granularity of the States?

We first calculate what the number of students should have been if the distribution of students across states was proportional to the population distribution. That would have been "fair".

We then compare it with the actual representation of each state in IITB, and define our skew fairness measure as:

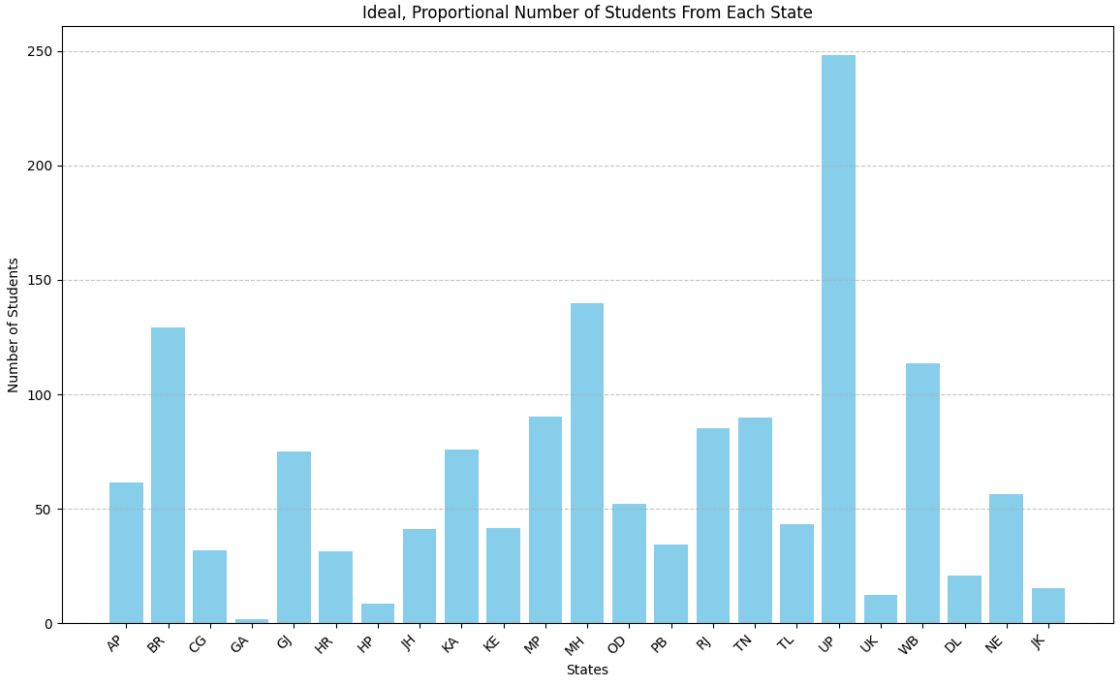
Skew Fairness = (Actual - Ideal) / Ideal

where "Actual" represents the number of students from a given state in IITB, and "Ideal" represents the ideal proportional representation.

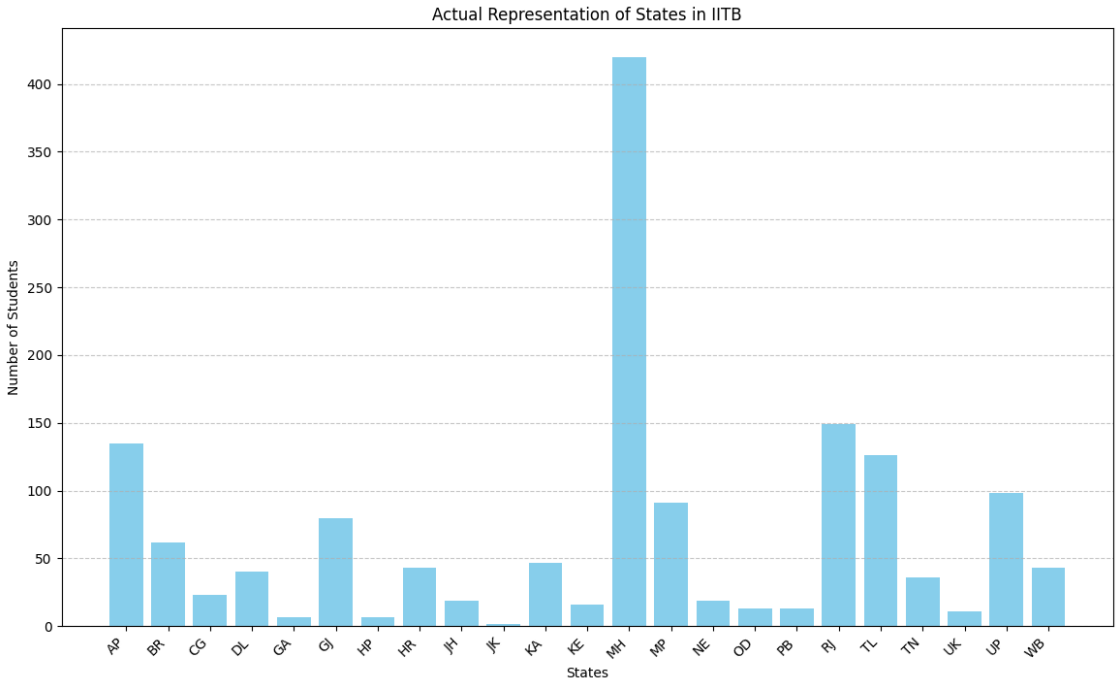
This measure indicates how deviated (by what factor of the ideal representation) the actual representation of a state is, from what it should've been. A positive value indicates more than proportionate representation, while a negative value indicates a lesser number of students than what was 'fair'.

[14]:
```
state_vs_pop = {state_info['State Codes'][i]: state_info['Population'][i] for i␣
 ↪in range(len(state_counts))}
total_pop = sum(state_vs_pop.values())

ideal_proportional_rep = {k: v * num_students/total_pop for k, v in␣
 ↪state_vs_pop.items()}

keys = list(ideal_proportional_rep.keys())
values = list(ideal_proportional_rep.values())

plt.figure(figsize=(14, 8))
plt.bar(keys, values, color='skyblue')
plt.xlabel('States')
plt.ylabel('Number of Students')
plt.title('Ideal, Proportional Number of Students From Each State')
```

```
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45, ha='right')

plt.show()
```

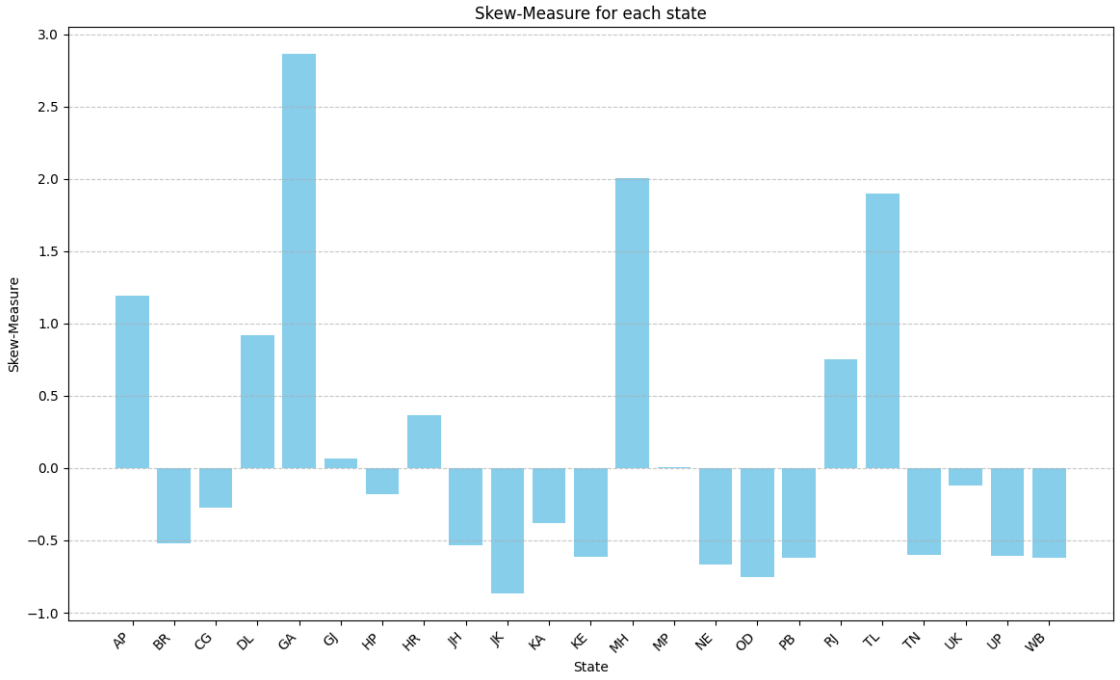


[15]:
```
keys = list(sorted(state_counts_dict.keys(), key = lambda x: x, reverse= False))
values = [state_counts_dict[k] for k in keys]

plt.figure(figsize=(14, 8))
plt.bar(keys, values, color='skyblue')
plt.xlabel('States')
plt.ylabel('Number of Students')
plt.title('Actual Representation of States in IITB')
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45, ha='right')

plt.show()
```



[16]:
```
def state_skew(state_id):
    actual = state_counts_dict[state_id]
    ideal = ideal_proportional_rep[state_id]

    return (actual - ideal)/ideal
```

[17]:
```
keys = list(sorted(state_counts_dict.keys(), key = lambda x: x, reverse= False))
values = [state_skew(k) for k in keys]

plt.figure(figsize=(14, 8))
plt.bar(keys, values, color='skyblue')
plt.xlabel('State')
plt.ylabel('Skew-Measure')
plt.title('Skew-Measure for each state')
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45, ha='right')

plt.show()

rms_skew = np.sum([abs(state_skew(state)) * ideal_proportional_rep[state]/1500␣
 ↪for state in state_counts_dict.keys()])
print(f'State Population Weighted Average of Absolute Value of Skew-Measure␣
 ↪across states: {rms_skew}')
```




```
State Population Weighted Average of Absolute Value of Skew-Measure across
states: 0.7219695831974697
```

0.5.1 How skewed from the population proportional distribution?

The representation in IITB does not appear to be distributed fairly according to population, as the skewfairness measure has high values for several states. States like Maharashtra, Andhra Pradesh and Telangana show high positive values of skewfairness, while Jammu and Kashmir, West Bengal, Bihar and Odisha have highly negative skewfairness values. So, no, the IITB population is not a good example of the granularity of India at the granularity of states.

0.6 Fairness with Respect to Population and Per-Capita Income of States

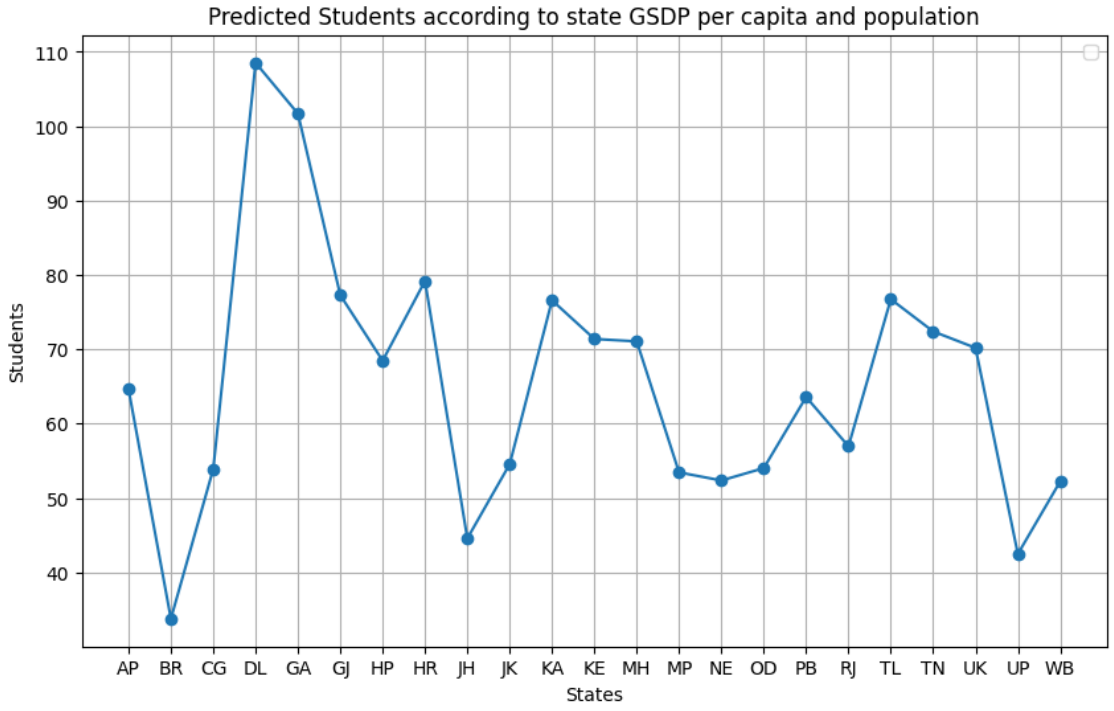
Assumption: Ideally, the number of students should be √(per capita GSDP) (assumed to be not linear)
The number of students should also be state population.
Therefore, the number of students should be √(per capita GSDP) * state population

Then, to measure skew-fairness, we again define the measure as: **Skew Fairness = (Actual - Ideal) / Ideal**

[18]:
```
per_cap_vs_state = dict(zip(state_info['State Codes'].values(),␣
 ↪state_info['GSDP Per Capita (million Rs.)'].values()))
per_cap_inc_vs_count = {per_cap_vs_state[state]:state_counts_dict[state] for␣
 ↪state in state_counts_dict.keys()}
```

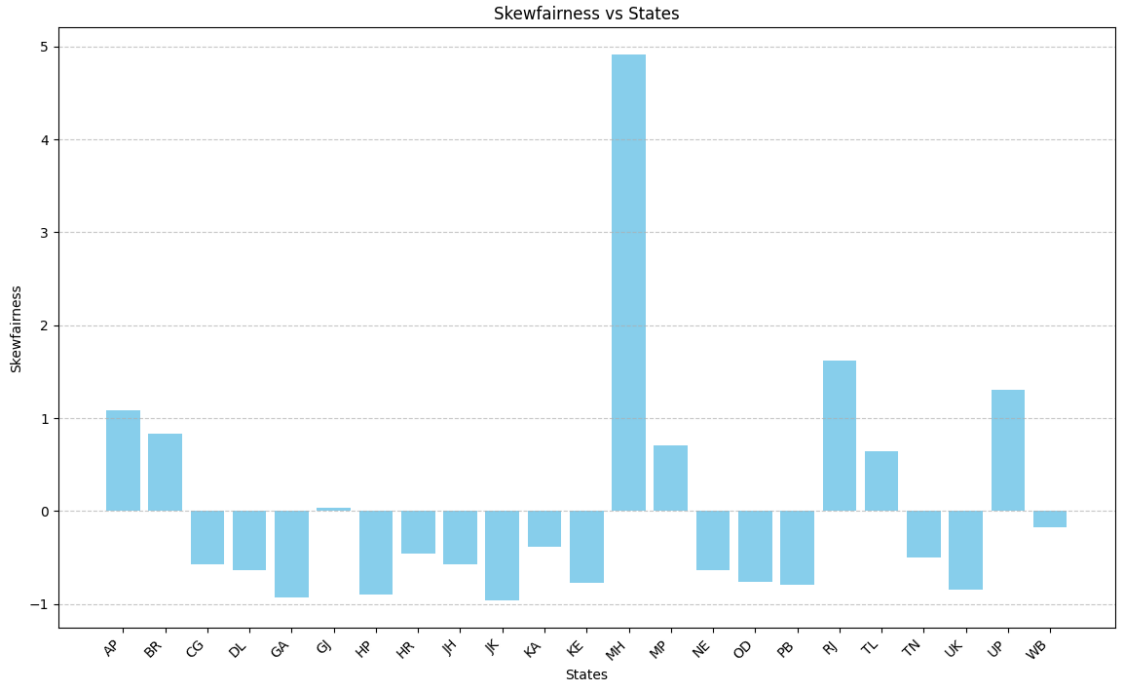
```
ideal_per_capita_rep = {s: 1500 * state_vs_pop[s]*(per_cap_vs_state[s] ** 0.5/␣
 ↪sum(per_cap_vs_state[u] ** 0.5 * state_vs_pop[s] for u in state_vs_pop.
 ↪keys()))  for s in state_vs_pop.keys()}
plt.figure(figsize=(10, 6))

sorted_stuff3 = sorted(ideal_per_capita_rep.items(), key = lambda x: x[0])
sorted_stuff3 = {x[0] : x[1] for x in sorted_stuff3}
plt.plot(sorted_stuff3.keys(), sorted_stuff3.values(), marker='o')
plt.ylabel('Students')
plt.xlabel('States')
plt.title('Predicted Students according to state GSDP per capita and␣
 ↪population')
plt.legend()
plt.grid(True)
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
```



[19]:
```
def state_skew_per_cap_pop(state):
    ideal = ideal_per_capita_rep[state]
    actual = state_counts_dict[state]
    return (actual - ideal)/ideal
```

```
keys = list(sorted(state_counts_dict.keys(), key = lambda x: x, reverse= False))
values = [state_skew_per_cap_pop(k) for k in keys]

plt.figure(figsize=(14, 8))
plt.bar(keys, values, color='skyblue')


plt.xlabel('States')
plt.ylabel('Skewfairness')
plt.title('Skewfairness vs States')
plt.grid(True, axis='y', linestyle='--', alpha=0.7)

plt.xticks(rotation=45, ha='right')


plt.show()

rms_skew_gsdp = np.sum([abs(state_skew_per_cap_pop(state)) *␣
 ↪ideal_proportional_rep[state]/1500 for state in state_counts_dict.keys()])
print(f'State Population Weighted Average of Absolute Value of Skew-Measure␣
 ↪across states: {rms_skew_gsdp}')
```




```
State Population Weighted Average of Absolute Value of Skew-Measure across
states: 1.1672535145236824
```

0.6.1 How skewed from the model?

The representation in IITB does not appear to be distributed fairly according to population and GSDP per capita, as the skewfairness measure has high absolute values for several states. So, the IITB population is highly skewed from the 'fair' distribution model w.r.t. Per Capita GSDP and the Population.

0.6.2 State affect the JEE rank, the graduating CPI, and the first salary?

[20]:
```
df1 = data
df2 = pd.read_csv('StateInfo.csv')
```

[21]:
```
state_code = df2['State Codes'].tolist()
origin_counts = df1['Origin'].value_counts()

cpi = df1['CPI'].median()
print(cpi)

l_r = df2['Literacy %'].median()
print(l_r)

print(df1['Fam_Income'].median())
print(df2['GSDP Per Capita (million Rs.)'].median())
```

```
7.06
80.4
2500495.5
228702.0
```

[22]:
```
gsdp_states =  df2['State Codes'][df2['GSDP Per Capita (million Rs.)'] >=
 ↪228702.0]  # This finds GSDP per capita of state that is greater than median
 ↪value i.e. 228702.0

lit_rate = df2['State Codes'][df2['Literacy %'] >= 80.40]   # This finds
 ↪literacy rate of states greater than meadian value i.e. 80.40

req_states = set(gsdp_states).intersection(set(lit_rate))
req_states = list(req_states)

print(req_states)

median = df1['Rank'].median()
print(median)

states = np.array(req_states)

mask = df1['Origin'].isin(states)
```

```
# In this Dataframe these are states which are following above condition
origin_condition = df1['Origin'].isin(states)

print(df1[origin_condition])
```

```
['TN', 'KE', 'GA', 'HP', 'MH', 'HR', 'UK', 'GJ', 'DL']
1028.0
      Unnamed: 0 Origin  Fam_Income  Score  Rank   CPI  First_Salary
1              1     MH     1711319    152   597  7.37       2110000
2              2     MH     2670466    223   155  8.06       2660000
4              4     MH     2241793    212   178  7.29       2020000
5              5     UK     2384692     66  1653  5.58       2300000
7              7     MH     3019608    226   110  8.45       2980000
…            …     …          …     …    …   …            …
1493        1493     GJ     2724808    143  1028  6.77       2440000
1494        1494     MH     2788977    250    51  8.22       2850000
1495        1495     MH     2421113    192   595  7.50       2060000
1498        1498     GJ     2427495    146   769  6.81       2460000
1499        1499     MH     2608313    201   213  8.08       2560000

[660 rows x 7 columns]
```

[23]:
```
cpi_condition = df1['CPI'] >= 7.08
rank_condition = df1['Rank'] < 1028
fst_sal_condition = df1['First_Salary'] >= 2500495.5

# Intersection of above cases
combined_condition = (
    origin_condition &
    fst_sal_condition &
    cpi_condition &
    rank_condition
)

filtered_df = df1[combined_condition]
state_counts = filtered_df['Origin'].value_counts()

print(filtered_df)
print(state_counts)

print(f'\n Since out of 1500 students, only these number of students have good␣
 ↪CPI, good first salary and good rank: "{filtered_df.shape[0]}".')
```

```
      Unnamed: 0 Origin  Fam_Income  Score  Rank   CPI  First_Salary
2              2     MH     2670466    223   155  8.06       2660000
7              7     MH     3019608    226   110  8.45       2980000
11            11     GJ     3683618    186   345  7.15       2560000
12            12     GJ     2354626    128   858  7.26       2570000
```

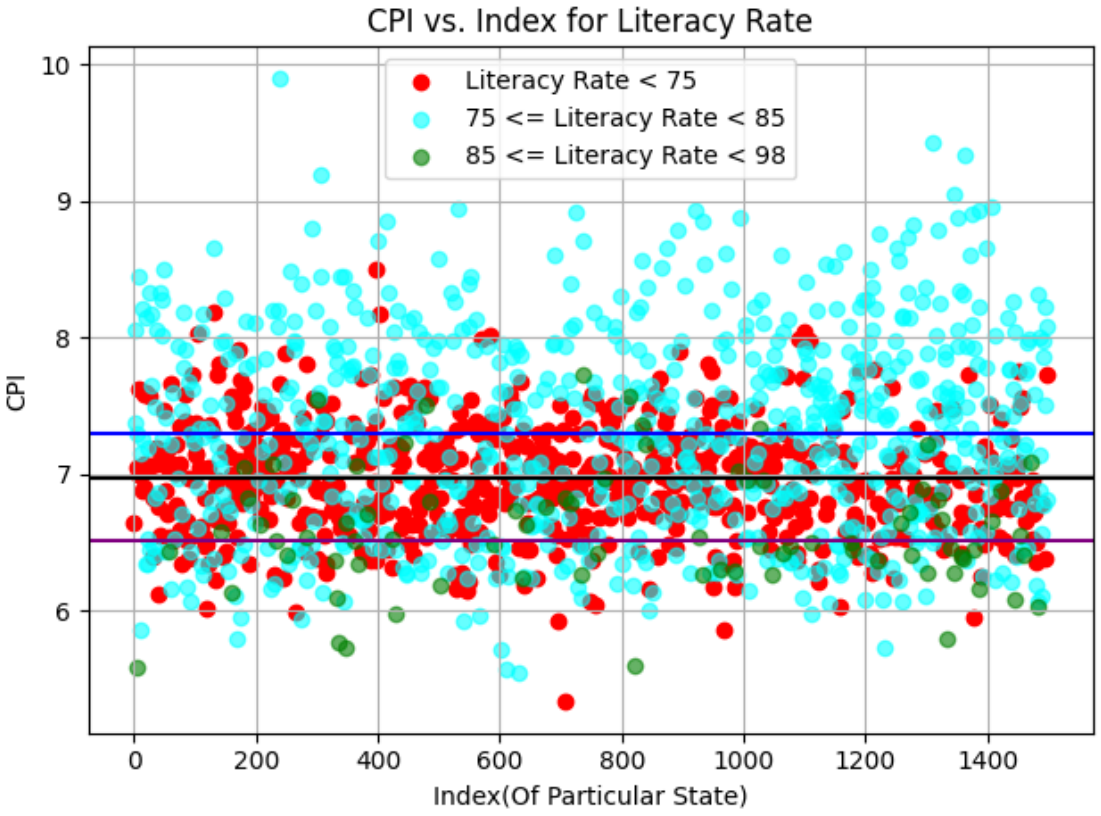
```
14            14     MH     2704665    233    78  8.21       2590000
…            …    …          …    …   …   …            …
1469        1469     MH     3131210    224   108  7.99       2950000
1483        1483     MH     3048780    259    34  8.31       2570000
1492        1492     MH     2352230    169   458  7.86       2570000
1494        1494     MH     2788977    250    51  8.22       2850000
1499        1499     MH     2608313    201   213  8.08       2560000

[212 rows x 7 columns]
Origin
MH    197
GJ     11
DL      4
Name: count, dtype: int64
```

Since out of 1500 students, only these number of students have good CPI, good first salary and good rank: "212".

[24]:
```
lr_75 =  df2[df2['Literacy %'] < 75]['State Codes']
lr_75 = lr_75.replace('BH', 'BR')
lr_85 = df2[(df2['Literacy %'] >=75) & (df2['Literacy %'] < 85)]['State Codes']
lr_98 = df2[(df2['Literacy %'] >=85) & (df2['Literacy %'] < 98)]['State Codes']

cpi_lr_75 = df1[df1['Origin'].isin(lr_75)]['CPI']
cpi_lr_85 = df1[df1['Origin'].isin(lr_85)]['CPI']
cpi_lr_98 = df1[df1['Origin'].isin(lr_98)]['CPI']

plt.scatter(cpi_lr_75.index, cpi_lr_75, color='red', label='Literacy Rate < 75')
plt.scatter(cpi_lr_85.index, cpi_lr_85, color='cyan', label='75 <= Literacy␣
 ↪Rate < 85', alpha=0.6)
plt.scatter(cpi_lr_98.index, cpi_lr_98, color='green', label='85 <= Literacy␣
 ↪Rate < 98', alpha=0.6)

plt.axhline(cpi_lr_75.median(),color='black')
plt.axhline(cpi_lr_85.median(),color='blue')
plt.axhline(cpi_lr_98.median(),color='purple')

plt.xlabel('Index(Of Particular State)')
plt.ylabel('CPI')
plt.title('CPI vs. Index for Literacy Rate')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students belongs to Sate which has Literacy Rate < 75: {cpi_lr_75.
 ↪count()} and has average CPI: {cpi_lr_75.mean():.2f}')
```

```
print(f'Total students belongs to Sate which has Literacy Rate < 85: {cpi_lr_85.
  ↪count()} and has average CPI: {cpi_lr_85.mean():.2f}')
print(f'Total students belongs to Sate which has Literacy Rate < 98: {cpi_lr_98.
  ↪count()} and has average CPI: {cpi_lr_98.mean():.2f}')
plt.show()
```

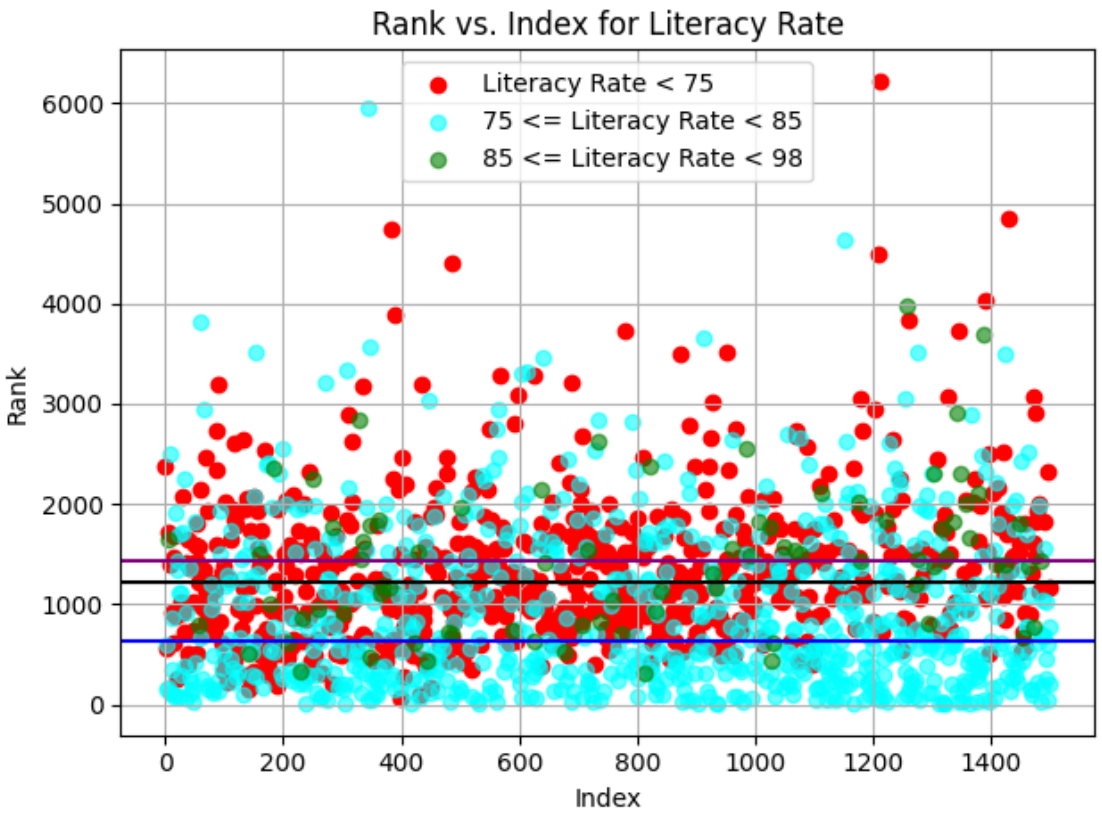
```
Total students belongs to Sate which has Literacy Rate < 75: 680 and has average
CPI: 6.97
Total students belongs to Sate which has Literacy Rate < 85: 739 and has average
CPI: 7.31
Total students belongs to Sate which has Literacy Rate < 98: 81 and has average
CPI: 6.58
```



[25]:
```
lr_75 =  df2[df2['Literacy %'] < 75]['State Codes']
lr_75 = lr_75.replace('BH', 'BR')
lr_85 = df2[(df2['Literacy %'] >=75) & (df2['Literacy %'] < 85)]['State Codes']
lr_98 = df2[(df2['Literacy %'] >=85) & (df2['Literacy %'] < 98)]['State Codes']

rank_lr_75 = df1[df1['Origin'].isin(lr_75)]['Rank']
rank_lr_85 = df1[df1['Origin'].isin(lr_85)]['Rank']
rank_lr_98 = df1[df1['Origin'].isin(lr_98)]['Rank']
```

```
plt.scatter(rank_lr_75.index, rank_lr_75, color='red', label='Literacy Rate <␣
 ↪75')
plt.scatter(rank_lr_85.index, rank_lr_85, color='cyan', label='75 <= Literacy␣
 ↪Rate < 85', alpha=0.6)
plt.scatter(rank_lr_98.index, rank_lr_98, color='green', label='85 <= Literacy␣
 ↪Rate < 98', alpha=0.6)

plt.axhline(rank_lr_75.median(),color='black')
plt.axhline(rank_lr_85.median(),color='blue')
plt.axhline(rank_lr_98.median(),color='purple')

plt.xlabel('Index')
plt.ylabel('Rank')
plt.title('Rank vs. Index for Literacy Rate')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students belongs to State which has Literacy Rate < 75:␣
 ↪{rank_lr_75.count()} and has First_Salary: {rank_lr_75.mean():.0f}')
print(f'Total students belongs to State which has Literacy Rate < 85:␣
 ↪{rank_lr_85.count()} and has First_Salary: {rank_lr_85.mean():.0f}')
print(f'Total students belongs to State which has Literacy Rate < 98:␣
 ↪{rank_lr_98.count()} and has First_Salary: {rank_lr_98.mean():.0f}')
plt.show()
```

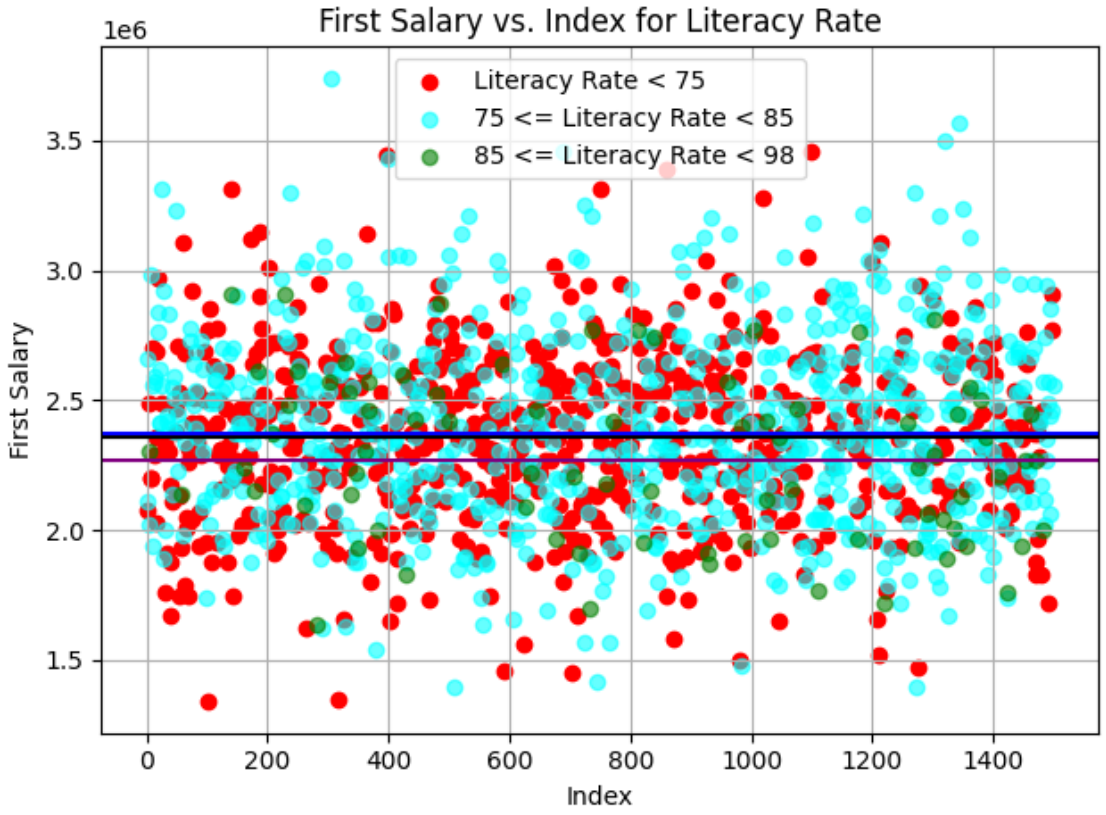
```
Total students belongs to State which has Literacy Rate < 75: 680 and has
First_Salary: 1345
Total students belongs to State which has Literacy Rate < 85: 739 and has
First_Salary: 894
Total students belongs to State which has Literacy Rate < 98: 81 and has
First_Salary: 1449
```



[26]:
```
lr_75 =  df2[df2['Literacy %'] < 75]['State Codes']
lr_75 = lr_75.replace('BH', 'BR')
lr_85 = df2[(df2['Literacy %'] >=75) & (df2['Literacy %'] < 85)]['State Codes']
lr_98 = df2[(df2['Literacy %'] >=85) & (df2['Literacy %'] < 98)]['State Codes']

fs_lr_75 = df1[df1['Origin'].isin(lr_75)]['First_Salary']
fs_lr_85 = df1[df1['Origin'].isin(lr_85)]['First_Salary']
fs_lr_98 = df1[df1['Origin'].isin(lr_98)]['First_Salary']

plt.scatter(fs_lr_75.index, fs_lr_75, color='red', label='Literacy Rate < 75')
plt.scatter(fs_lr_85.index, fs_lr_85, color='cyan', label='75 <= Literacy Rate␣
 ↪< 85', alpha=0.6)
plt.scatter(fs_lr_98.index, fs_lr_98, color='green', label='85 <= Literacy Rate␣
 ↪< 98', alpha=0.6)

plt.axhline(fs_lr_75.median(),color='black')
plt.axhline(fs_lr_85.median(),color='blue')
plt.axhline(fs_lr_98.median(),color='purple')

plt.xlabel('Index')
plt.ylabel('First Salary')
```

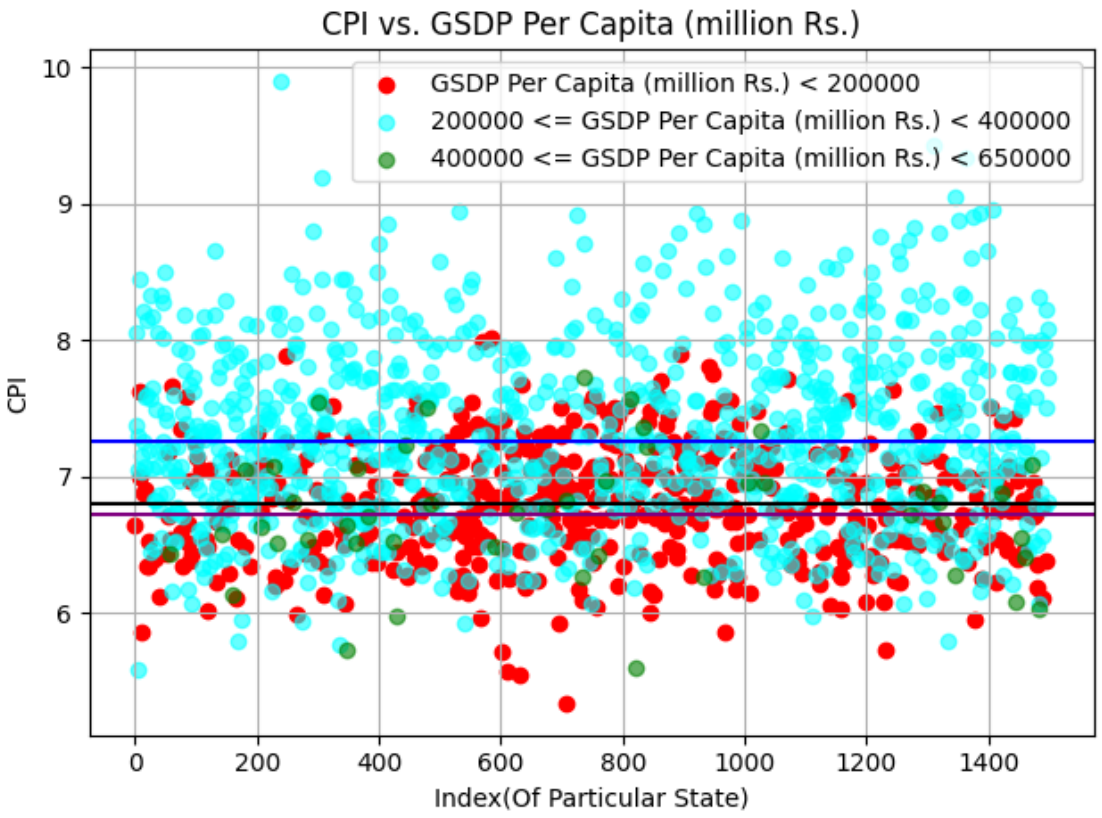
```
plt.title('First Salary vs. Index for Literacy Rate')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students belongs to State which has Literacy Rate < 75: {fs_lr_75.
  ↪count()} and has First_Salary: {fs_lr_75.mean():.2f}')
print(f'Total students belongs to State which has Literacy Rate < 85: {fs_lr_85.
  ↪count()} and has First_Salary: {fs_lr_85.mean():.2f}')
print(f'Total students belongs to State which has Literacy Rate < 98: {fs_lr_98.
  ↪count()} and has First_Salary: {fs_lr_98.mean():.2f}')
plt.show()
```

```
Total students belongs to State which has Literacy Rate < 75: 680 and has
First_Salary: 2359132.35
Total students belongs to State which has Literacy Rate < 85: 739 and has
First_Salary: 2396481.73
Total students belongs to State which has Literacy Rate < 98: 81 and has
First_Salary: 2276419.75
```



[27]:
```
# GSDP of State

gp_20 =  df2[df2['GSDP Per Capita (million Rs.)'] < 200000]['State Codes']
gp_20 = gp_20.replace('BH', 'BR')
gp_40 = df2[(df2['GSDP Per Capita (million Rs.)'] >=200000) & (df2['GSDP Per␣
 ↪Capita (million Rs.)'] < 400000)]['State Codes']
gp_65 = df2[(df2['GSDP Per Capita (million Rs.)'] >=400000) & (df2['GSDP Per␣
 ↪Capita (million Rs.)'] < 650000)]['State Codes']

cpi_gp_20 = df1[df1['Origin'].isin(gp_20)]['CPI']
cpi_gp_40 = df1[df1['Origin'].isin(gp_40)]['CPI']
cpi_gp_65 = df1[df1['Origin'].isin(gp_65)]['CPI']

plt.scatter(cpi_gp_20.index, cpi_gp_20, color='red', label='GSDP Per Capita␣
 ↪(million Rs.) < 200000')
plt.scatter(cpi_gp_40.index, cpi_gp_40, color='cyan', label='200000 <= GSDP Per␣
 ↪Capita (million Rs.) < 400000', alpha=0.6)
plt.scatter(cpi_gp_65.index, cpi_gp_65, color='green', label='400000 <= GSDP␣
 ↪Per Capita (million Rs.) < 650000', alpha=0.6)

plt.axhline(cpi_gp_20.median(),color='black')
plt.axhline(cpi_gp_40.median(),color='blue')
plt.axhline(cpi_gp_65.median(),color='purple')

plt.xlabel('Index(Of Particular State)')
plt.ylabel('CPI')
plt.title('CPI vs. GSDP Per Capita (million Rs.)')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
 ↪< 75: {cpi_gp_20.count()} and has average CPI: {cpi_gp_20.mean():.2f}')
print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
 ↪< 85: {cpi_gp_40.count()} and has average CPI: {cpi_gp_40.mean():.2f}')
print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
 ↪< 98: {cpi_gp_65.count()} and has average CPI: {cpi_gp_65.mean():.2f}')
plt.show()
```

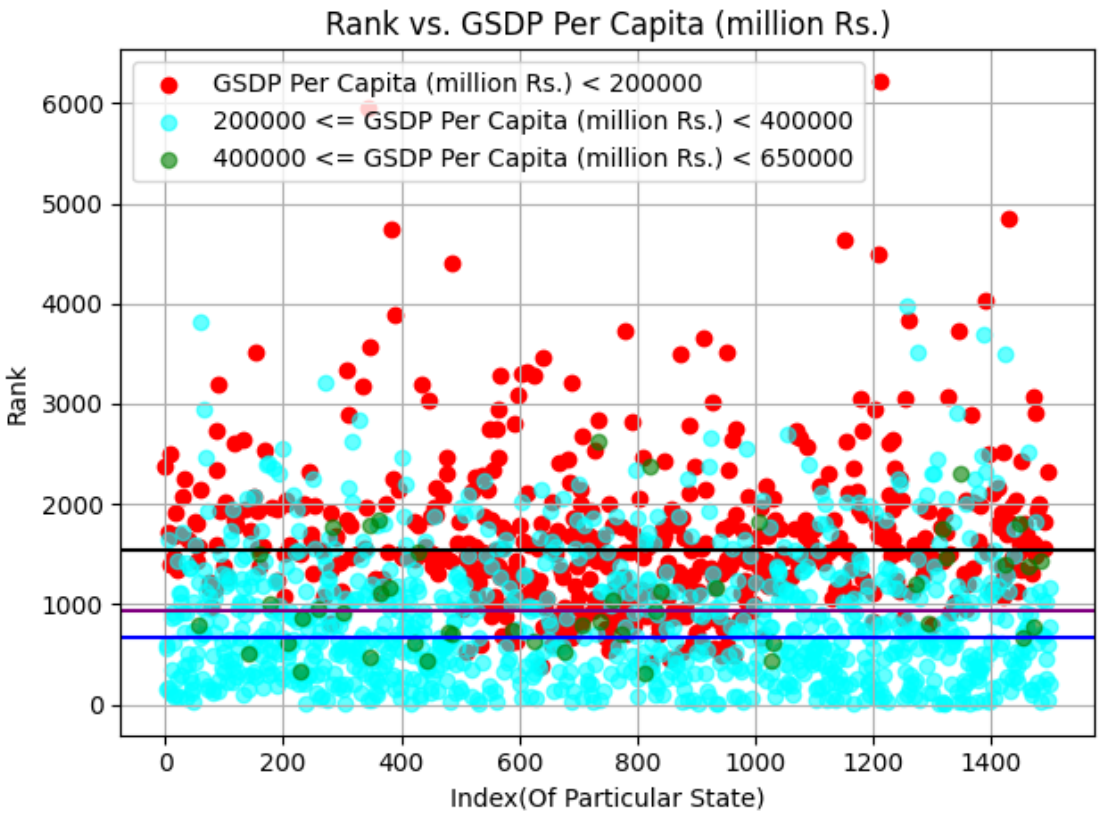
```
Total students belongs to State which has GSDP Per Capita (million Rs.) < 75:
519 and has average CPI: 6.80
Total students belongs to State which has GSDP Per Capita (million Rs.) < 85:
934 and has average CPI: 7.31
Total students belongs to State which has GSDP Per Capita (million Rs.) < 98: 47
and has average CPI: 6.72
```



[28]:
```
# GSDP of State

gp_20 =  df2[df2['GSDP Per Capita (million Rs.)'] < 200000]['State Codes']
gp_20 = gp_20.replace('BH', 'BR')
gp_40 = df2[(df2['GSDP Per Capita (million Rs.)'] >=200000) & (df2['GSDP Per␣
 ↪Capita (million Rs.)'] < 400000)]['State Codes']
gp_65 = df2[(df2['GSDP Per Capita (million Rs.)'] >=400000) & (df2['GSDP Per␣
 ↪Capita (million Rs.)'] < 650000)]['State Codes']

rank_gp_20 = df1[df1['Origin'].isin(gp_20)]['Rank']
rank_gp_40 = df1[df1['Origin'].isin(gp_40)]['Rank']
rank_gp_65 = df1[df1['Origin'].isin(gp_65)]['Rank']

plt.scatter(rank_gp_20.index, rank_gp_20, color='red', label='GSDP Per Capita␣
 ↪(million Rs.) < 200000')
plt.scatter(rank_gp_40.index, rank_gp_40, color='cyan', label='200000 <= GSDP␣
 ↪Per Capita (million Rs.) < 400000', alpha=0.6)
plt.scatter(rank_gp_65.index, rank_gp_65, color='green', label='400000 <= GSDP␣
 ↪Per Capita (million Rs.) < 650000', alpha=0.6)
```

```
plt.axhline(rank_gp_20.median(),color='black')
plt.axhline(rank_gp_40.median(),color='blue')
plt.axhline(rank_gp_65.median(),color='purple')

plt.xlabel('Index(Of Particular State)')
plt.ylabel('Rank')
plt.title('Rank vs. GSDP Per Capita (million Rs.)')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
  ↪< 200000: {rank_gp_20.count()} and has avg Rank: {rank_gp_20.mean():.0f}')
print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
  ↪< 400000: {rank_gp_40.count()} and has avg Rank: {rank_gp_40.mean():.0f}')
print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
  ↪< 650000: {rank_gp_65.count()} and has avg Rank: {rank_gp_65.mean():.0f}')
plt.show()
```

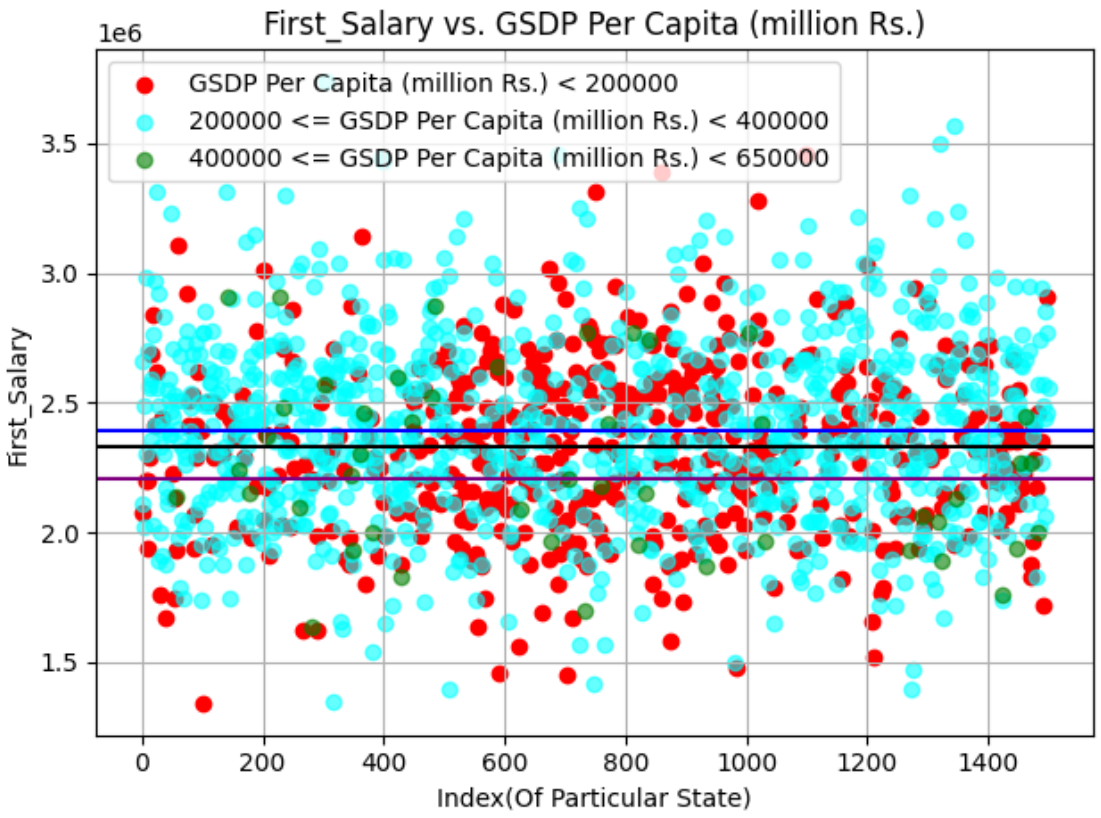
```
Total students belongs to State which has GSDP Per Capita (million Rs.) <
200000: 519 and has avg Rank: 1682
Total students belongs to State which has GSDP Per Capita (million Rs.) <
400000: 934 and has avg Rank: 823
Total students belongs to State which has GSDP Per Capita (million Rs.) <
650000: 47 and has avg Rank: 1094
```



[29]:
```
# GSDP of State

gp_20 =  df2[df2['GSDP Per Capita (million Rs.)'] < 200000]['State Codes']
gp_20 = gp_20.replace('BH', 'BR')
gp_40 = df2[(df2['GSDP Per Capita (million Rs.)'] >=200000) & (df2['GSDP Per␣
 ↪Capita (million Rs.)'] < 400000)]['State Codes']
gp_65 = df2[(df2['GSDP Per Capita (million Rs.)'] >=400000) & (df2['GSDP Per␣
 ↪Capita (million Rs.)'] < 650000)]['State Codes']

fs_gp_20 = df1[df1['Origin'].isin(gp_20)]['First_Salary']
fs_gp_40 = df1[df1['Origin'].isin(gp_40)]['First_Salary']
fs_gp_65 = df1[df1['Origin'].isin(gp_65)]['First_Salary']

plt.scatter(fs_gp_20.index, fs_gp_20, color='red', label='GSDP Per Capita␣
 ↪(million Rs.) < 200000')
plt.scatter(fs_gp_40.index, fs_gp_40, color='cyan', label='200000 <= GSDP Per␣
 ↪Capita (million Rs.) < 400000', alpha=0.6)
plt.scatter(fs_gp_65.index, fs_gp_65, color='green', label='400000 <= GSDP Per␣
 ↪Capita (million Rs.) < 650000', alpha=0.6)
```

```
plt.axhline(fs_gp_20.median(),color='black')
plt.axhline(fs_gp_40.median(),color='blue')
plt.axhline(fs_gp_65.median(),color='purple')

plt.xlabel('Index(Of Particular State)')
plt.ylabel('First_Salary')
plt.title('First_Salary vs. GSDP Per Capita (million Rs.)')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
  ↪< 200000: {fs_gp_20.count()} and has First_Salaryk: {fs_gp_20.mean():.2f}')
print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
  ↪< 400000: {fs_gp_40.count()} and has First_Salary: {fs_gp_40.mean():.2f}')
print(f'Total students belongs to State which has GSDP Per Capita (million Rs.)␣
  ↪< 650000: {fs_gp_65.count()} and has First_Salary: {fs_gp_65.mean():.2f}')
plt.show()
```

```
Total students belongs to State which has GSDP Per Capita (million Rs.) <
200000: 519 and has First_Salaryk: 2329479.77
Total students belongs to State which has GSDP Per Capita (million Rs.) <
400000: 934 and has First_Salary: 2403179.87
Total students belongs to State which has GSDP Per Capita (million Rs.) <
650000: 47 and has First_Salary: 2255957.45
```



From the above cases, there is no strong proof that State affects JEE Rank, graduating CPI and First Salary.

0.6.3 Family income affect the JEE rank, the graduating CPI, and the first salary?

[30]:
```
# Conditons to distribute students who have Family Income, CPI, Rank And First␣
↪Salary above average (MEDIAN)

fam_income_condition = df1['Fam_Income'] >= 2291998
rank_condition = df1['Rank'] < 1028
cpi_condition = df1['CPI'] >= 7.08
fst_sal_condition = df1['First_Salary'] >= 2540000

combined_condition = (
    cpi_condition &
    fst_sal_condition &
    fam_income_condition &
    rank_condition
)

filtered_df_2 = df1[combined_condition]
```

```
state_counts_2 = filtered_df_2['Origin'].value_counts()

print(filtered_df_2)

print(state_counts_2)
print(f'\n Since out of 1500 students, only these number of students have Good␣
 ↪Family Income, good CPI, good first salary and rank "{filtered_df_2.
 ↪shape[0]}" .')
```

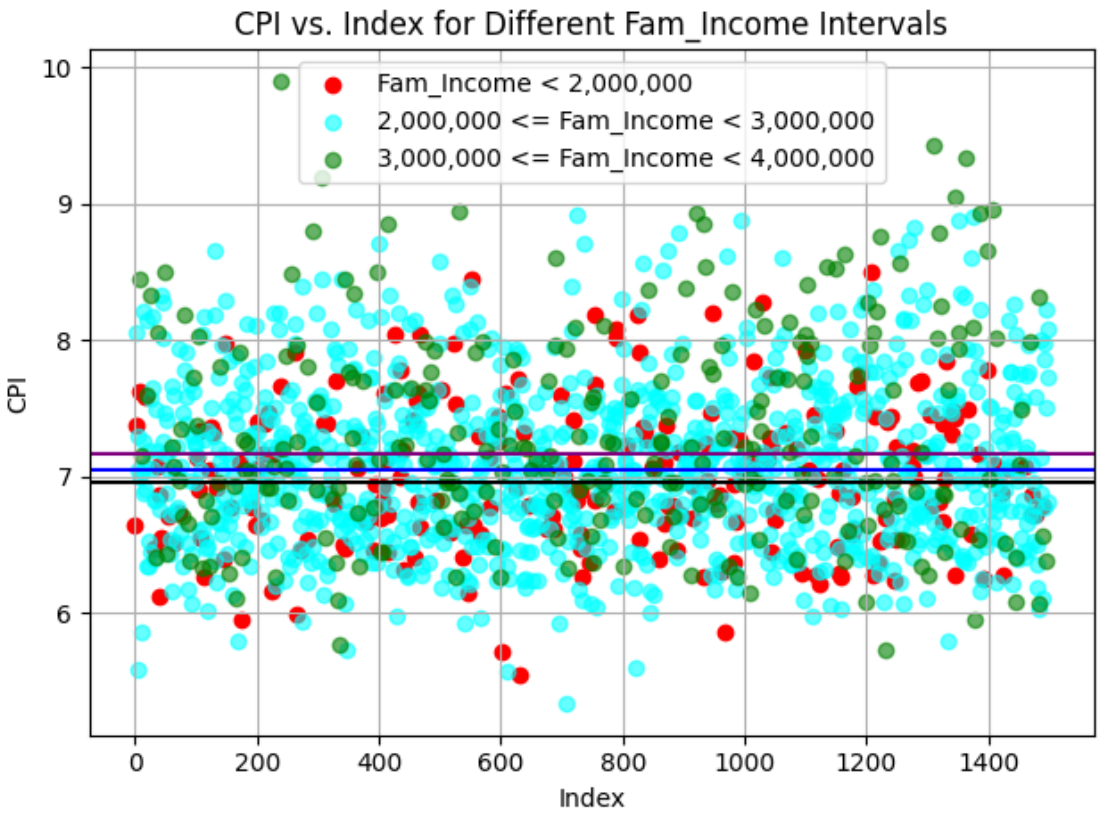
```
      Unnamed: 0 Origin  Fam_Income  Score  Rank   CPI  First_Salary
2              2     MH     2670466    223   155  8.06       2660000
7              7     MH     3019608    226   110  8.45       2980000
11            11     GJ     3683618    186   345  7.15       2560000
12            12     GJ     2354626    128   858  7.26       2570000
14            14     MH     2704665    233    78  8.21       2590000
…            …     …          …     …    …   …           …
1469        1469     MH     3131210    224   108  7.99       2950000
1483        1483     MH     3048780    259    34  8.31       2570000
1492        1492     MH     2352230    169   458  7.86       2570000
1494        1494     MH     2788977    250    51  8.22       2850000
1499        1499     MH     2608313    201   213  8.08       2560000

[240 rows x 7 columns]
Origin
MH    159
TL     24
AP     20
RJ     19
GJ     10
MP      4
DL      3
KA      1
Name: count, dtype: int64

 Since out of 1500 students, only these number of students have Good Family
Income, good CPI, good first salary and rank "240" .
```

This code is between Fam_Income and Graduating CPI.

[31]:
```
fam_income_lt_15 =  df1[df1['Fam_Income'] < 2000000]['CPI']
fam_income_lt_30 = df1[(df1['Fam_Income'] >= 2000000) & (df1['Fam_Income'] <␣
 ↪3000000)]['CPI']
fam_income_lt_40 = df1[(df1['Fam_Income'] >= 3000000) & (df1['Fam_Income'] <␣
 ↪4000000)]['CPI']

plt.scatter(fam_income_lt_15.index, fam_income_lt_15, color='red',␣
 ↪label='Fam_Income < 2,000,000')
```

```
plt.scatter(fam_income_lt_30.index, fam_income_lt_30, color='cyan',␣
 ↪label='2,000,000 <= Fam_Income < 3,000,000', alpha=0.6)
plt.scatter(fam_income_lt_40.index, fam_income_lt_40, color='green',␣
 ↪label='3,000,000 <= Fam_Income < 4,000,000', alpha=0.6)

plt.axhline(fam_income_lt_15.median(),color='black')
plt.axhline(fam_income_lt_30.median(),color='blue')
plt.axhline(fam_income_lt_40.median(),color='purple')

plt.xlabel('Index')
plt.ylabel('CPI')
plt.title('CPI vs. Index for Different Fam_Income Intervals')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students has fam_income less than 2000000: {fam_income_lt_15.
 ↪count()} and has average CPI: {fam_income_lt_15.mean():.2f}')
print(f'Total students has fam_income >= 2000000 and <= 3000000:␣
 ↪{fam_income_lt_30.count()} and has average CPI: {fam_income_lt_30.mean():.
 ↪2f}')
print(f'Total students has fam_income >= 3000000 and <= 4000000:␣
 ↪{fam_income_lt_40.count()} and has average CPI: {fam_income_lt_40.mean():.
 ↪2f}')

plt.show()
```

```
Total students has fam_income less than 2000000: 233 and has average CPI: 7.00
Total students has fam_income >= 2000000 and <= 3000000: 1011 and has average
CPI: 7.09
Total students has fam_income >= 3000000 and <= 4000000: 256 and has average
CPI: 7.32
```

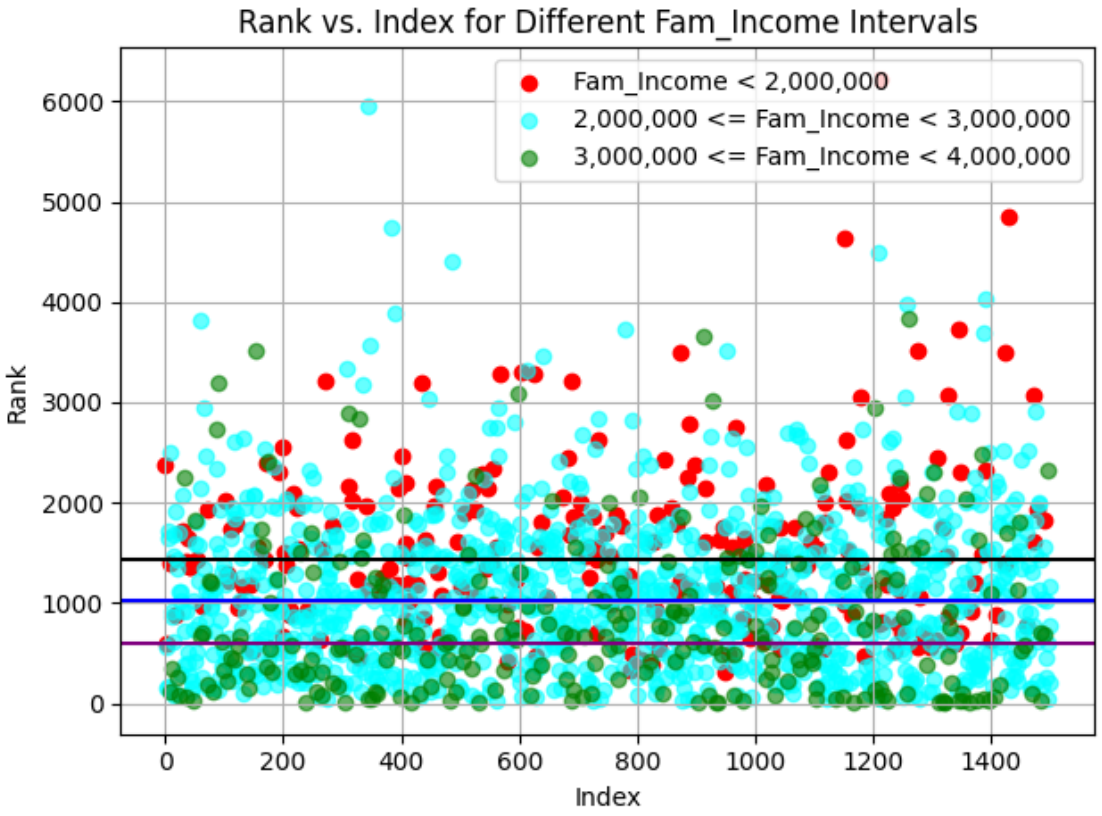


This code is between Fam_Income and JEE Rank.

[32]:
```
fam_income_lt_15 =  df1[df1['Fam_Income'] < 2000000]['Rank']
fam_income_lt_30 = df1[(df1['Fam_Income'] >= 2000000) & (df1['Fam_Income'] <␣
↪3000000)]['Rank']
fam_income_lt_40 = df1[(df1['Fam_Income'] >= 3000000) & (df1['Fam_Income'] <␣
↪4000000)]['Rank']

plt.scatter(fam_income_lt_15.index, fam_income_lt_15, color='red',␣
↪label='Fam_Income < 2,000,000')
plt.scatter(fam_income_lt_30.index, fam_income_lt_30, color='cyan',␣
↪label='2,000,000 <= Fam_Income < 3,000,000', alpha=0.6)
plt.scatter(fam_income_lt_40.index, fam_income_lt_40, color='green',␣
↪label='3,000,000 <= Fam_Income < 4,000,000', alpha=0.6)

plt.axhline(fam_income_lt_15.median(),color='black')
plt.axhline(fam_income_lt_30.median(),color='blue')
plt.axhline(fam_income_lt_40.median(),color='purple')

plt.xlabel('Index')
plt.ylabel('Rank')
```

```
plt.title('Rank vs. Index for Different Fam_Income Intervals')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students has fam_income less than 2000000: {fam_income_lt_15.
  ↪count()} and has average of Rank: {fam_income_lt_15.mean():.0f}')
print(f'Total students has fam_income >= 2000000 and <= 3000000 :␣
  ↪{fam_income_lt_30.count()} and has average of Rank: {fam_income_lt_30.mean():
  ↪.0f}')
print(f'Total students has fam_income >= 3000000 and <= 4000000:␣
  ↪{fam_income_lt_40.count()} and has average of Rank: {fam_income_lt_40.mean():
  ↪.0f}')

plt.show()
```

```
Total students has fam_income less than 2000000: 233 and has average of Rank:
1542
Total students has fam_income >= 2000000 and <= 3000000 : 1011 and has average
of Rank: 1112
Total students has fam_income >= 3000000 and <= 4000000: 256 and has average of
Rank: 819
```

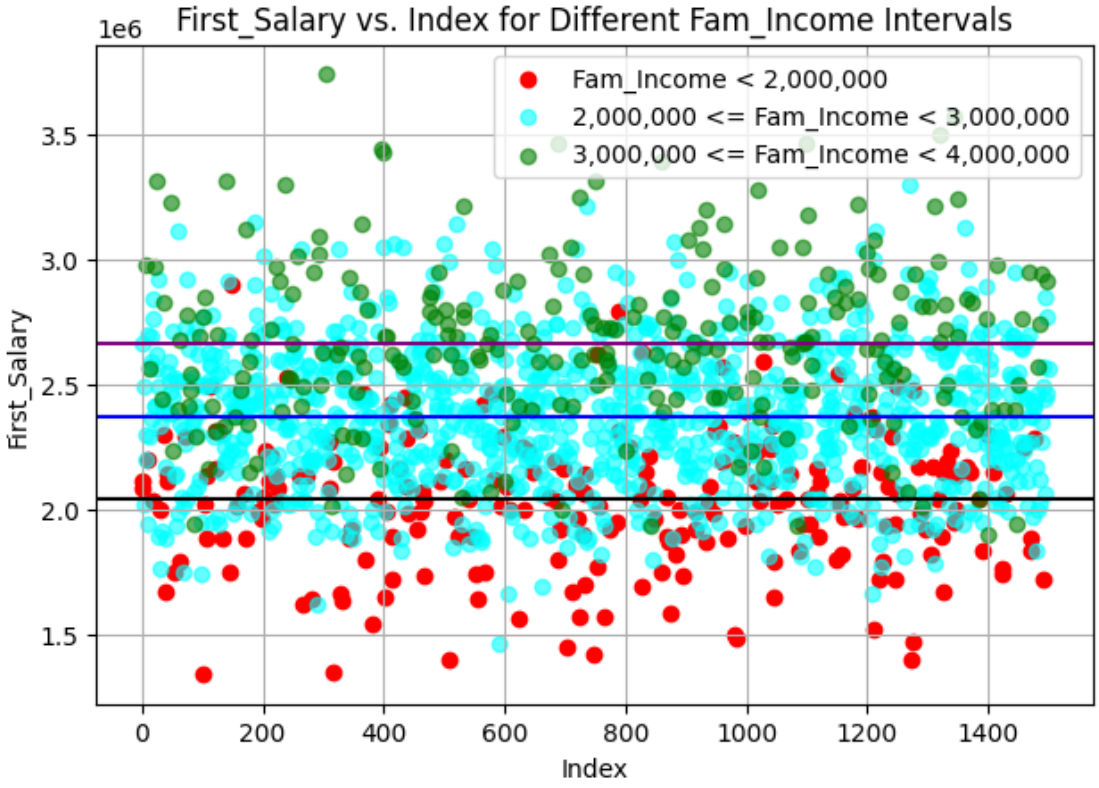


This code is between Fam_Income and First Salary.

[33]:
```
fam_income_lt_15 =  df1[df1['Fam_Income'] < 2000000]['First_Salary']
fam_income_lt_30 = df1[(df1['Fam_Income'] >= 2000000) & (df1['Fam_Income'] <␣
 ↪3000000)]['First_Salary']
fam_income_lt_40 = df1[(df1['Fam_Income'] >= 3000000) & (df1['Fam_Income'] <␣
 ↪4000000)]['First_Salary']


plt.scatter(fam_income_lt_15.index, fam_income_lt_15, color='red',␣
 ↪label='Fam_Income < 2,000,000')
plt.scatter(fam_income_lt_30.index, fam_income_lt_30, color='cyan',␣
 ↪label='2,000,000 <= Fam_Income < 3,000,000', alpha=0.6)
plt.scatter(fam_income_lt_40.index, fam_income_lt_40, color='green',␣
 ↪label='3,000,000 <= Fam_Income < 4,000,000', alpha=0.6)

# Median for fam_income_lt_15
plt.axhline(fam_income_lt_15.median(),color='black')
# Median for fam_income_lt_30
plt.axhline(fam_income_lt_30.median(),color='blue')
# Median for fam_income_lt_40
plt.axhline(fam_income_lt_40.median(),color='purple')

plt.xlabel('Index')
plt.ylabel('First_Salary')
plt.title('First_Salary vs. Index for Different Fam_Income Intervals')

plt.legend()
plt.grid(True)
plt.tight_layout()

print(f'Total students has fam_income less than 1500000: {fam_income_lt_15.
 ↪count()} and has avg First Salary around: {fam_income_lt_15.mean():.2f}')
print(f'Total students has fam_income less than 1500000: {fam_income_lt_30.
 ↪count()} and has avg First Salary around: {fam_income_lt_30.mean():.2f}')
print(f'Total students has fam_income less than 1500000: {fam_income_lt_40.
 ↪count()} and has avg First Salary around: {fam_income_lt_40.mean():.2f}')

plt.show()
```

```
Total students has fam_income less than 1500000: 233 and has avg First Salary
around: 2025751.07
Total students has fam_income less than 1500000: 1011 and has avg First Salary
around: 2377329.38
Total students has fam_income less than 1500000: 256 and has avg First Salary
```

```
around: 2672343.75
```



From above data Family Income plays significant role in JEE Rank, graduating CPI and First Salary.