EE-325

Programming Assignment -1 Report

Yaswanth Ram Kumar Bheesetti 23B1277
Harsha Sai Srinivas Pamarthi 23B1202
Bhavishya Singh Premi 23B1230

Part 1

1. **Is the opinion from the 100 people representative of the opinion of India? Explain**.
   Since there is no information about the 100 people surveyed, we cannot conclude that their opinions represent the entire population of India. India is highly diverse, and with such a vast range of perspectives, the views of these 100 individuals are merely a small subset of the broader spectrum of opinions across the country. Therefore, it would be inaccurate to generalise their opinions to the entire population.

2. **What is your belief about the home state of most of the IITB students? Write the belief from each member of your batch; it is reiterated that there is no shame in being wrong.**
   I believe that most IIT Bombay students are from Maharashtra because, ceteris paribus, people tend to prefer staying closer to home. When all other factors are equal, students are more likely to choose the IIT in their home state over those in other states.

Name	Roll No.	Opinion on Home state
Harshit Pendela	23B1207	Mostly from Andhra and Telangana
Jaswin Reddy	23B1289	Mostly from Maharashtra
Vishnu Bijarniya	23B1251	Mostly from Maharashtra
Lavudi Thushar	23B1224	Mostly from Maharashtra
Arshit Singh	23B1275	Mostly from Rajasthan

3. **How to select the K students to collect their data? How does the guess improve as a function of K? And hence, what is a good K?**
   The optimal selection of the K students should be such that the data should be fairly same as the actual complete Dataset, due to survey budget constraint, we want to keep K as low as possible, but it shouldn't be so low that it wouldn't be able to match the actual data. Even After fixing the value of number of samples K, the way how we pick the samples (students) also matters, they can be picked arbitrarily, randomly or just select the first K students in the available set.

   We have run some basic program to test all these possibilities and their deviation from the actual data, their plots are shown below in plot-1 . Number of samples taken is K= 10, 20, 50, 100, 200.

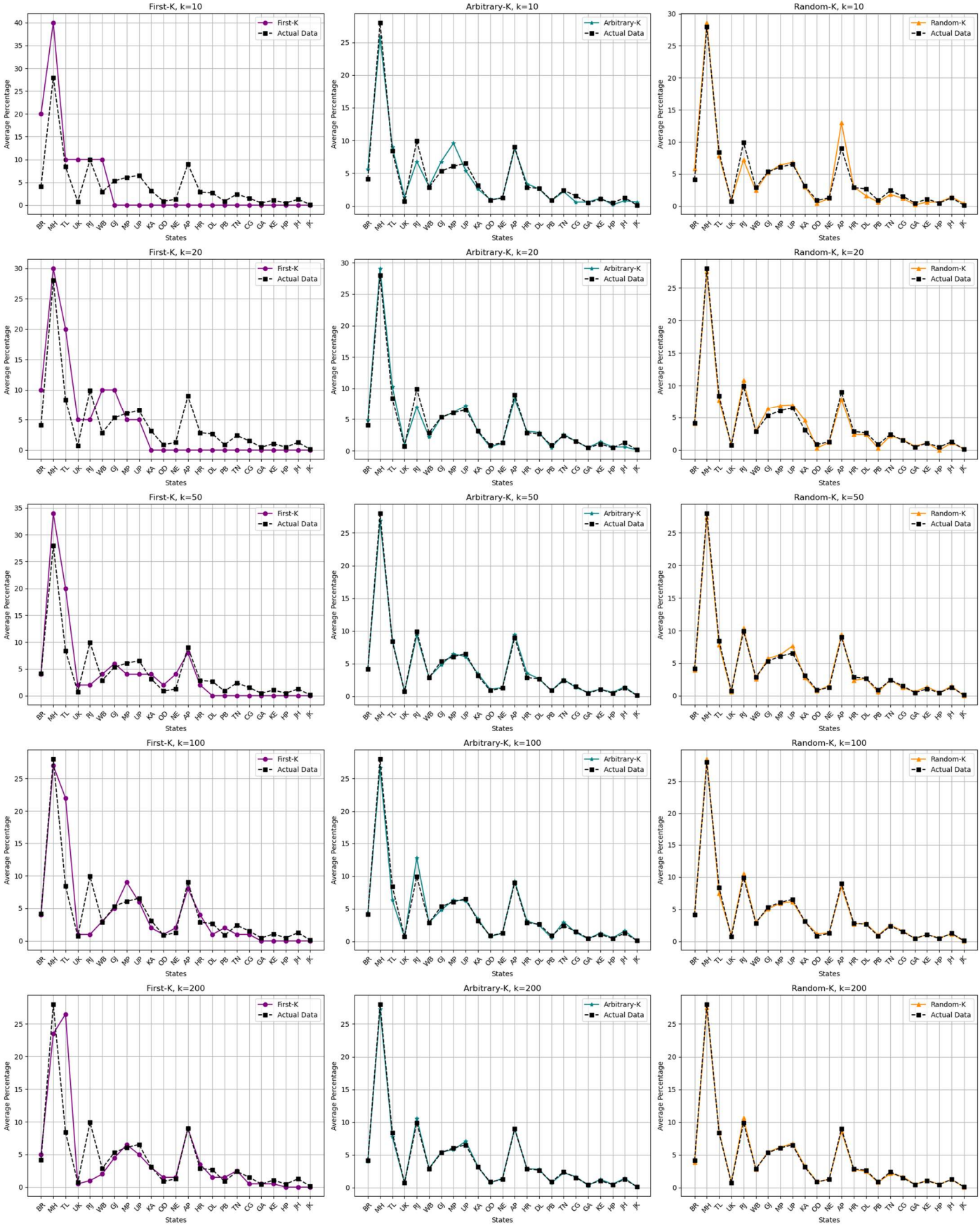
   The experiment has been repeated 50 times, for all 15 cases, it should be noted that the 50 experiments for First-K scenarios are just redundant, the percentage of each of the state in every experiment of similar kind is represented in the plots with same symbols.
   The comparison of these 15 scenarios with actual data, is given another plot-2.

   For Arbitrary scenarios, we picked a number less than 1500-K in random and picked the consecutive samples from that number.

Plot 1


First-K, k=10
Arbitrary-K, k=10
Random-K, k=10
First-K, k=20
Arbitrary-K, k=20
Random-K, k=20
First-K, k=50
Arbitrary-K, k=50
Random-K, k=50
First-K, k=100
Arbitrary-K, k=100
Random-K, k=100
First-K, k=200
Arbitrary-K, k=200
Random-K, k=200
Avg First-K
Avg Arbitrary-K
Avg Random-K
Percentage
States
BR MH TL UK RJ WB GJ MP UP KA OD NE AP HR DL PB TN CG GA KE HP JH JK

Plot 2


First-K, k=10
Arbitrary-K, k=10
Random-K, k=10
First-K, k=20
Arbitrary-K, k=20
Random-K, k=20
First-K, k=50
Arbitrary-K, k=50
Random-K, k=50
First-K, k=100
Arbitrary-K, k=100
Random-K, k=100
First-K, k=200
Arbitrary-K, k=200
Random-K, k=200
First-K
Arbitrary-K
Random-K
Actual Data
Average Percentage
States
BR MH TL UK RJ WB GJ MP UP KA OD NE AP HR DL PB TN CG GA KE HP JH JK

Observations and Conclusions

- We can observe that the actual data coincided mostly with random and arbitrary data.
- The Top 3 states and percentage of students from these states are: (from overall data)
    - Maharashtra - 28.0 % of total IITB population
    - Rajasthan - 9.93 % of total IITB population
    - Andhra Pradesh - 9.00 % of total IITB population
- For random scenarios, even at a lesser K compared to arbitrary and first-k, the data coincides perfectly.

K	MSE FOR FIRST-K	MSE FOR ARBITRARY	MSE FOR RANDOM
10	33.19	51.39	36.61
20	17.92	31.76	18.57
50	12.08	19.70	6.97
100	12.54	16.68	3.31
200	19.05	13.80	1.73

- The MSE for First-K sampling is not uniform with K, so it is not a very reliable way for sampling.
- In Random and Arbitrary sampling, MSE is fairly decreasing uniformly with increasing K.
- But random sampling is preferred over all, as the MSE is least for almost all K. Here it shouldn't be confused that at K=10, MSE is least for First-K sampling, but this is not generally always true, it so happened in this case due to the arrangement of the data in the CSV file, this shouldn't be used as a conclusion, or no generalizations should be drawn from this.
- Also, we can see that as K increases, MSE for all three cases decreases.
- So, a perfect K, and perfect sampling of all these possibilities is obtained with a **higher K**, that is not beyond the budget constraint, and a **Random Sampling**.
- So, if the maximum bearable cost is given, we can model an optimum value of K for Random sampling, a model which takes care of the trade of between accuracy and cost.

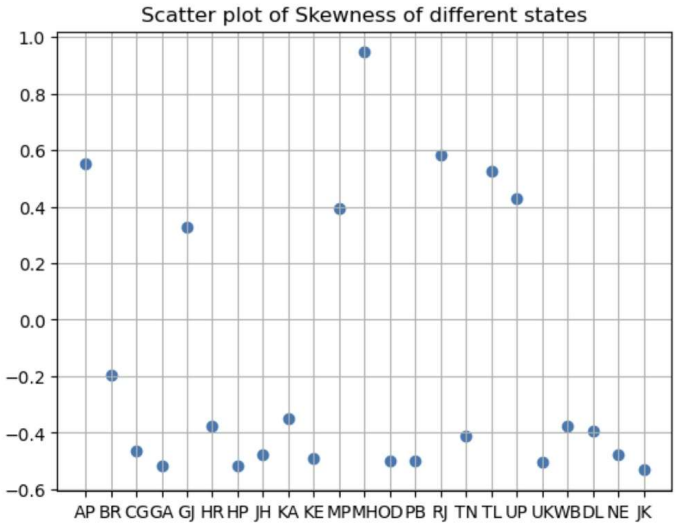
Part 2

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? (skewm)**

   To quantify skewness at every state's representation in IITB, we came up with our own relation: The skewness value is between -1 to +1, here we took the cube root to magnify the values a bit for better distinguishing.

$$\sqrt[3]{\frac{(\%\text{ of population of that state in IITB} - \text{Mean of }\%\text{ of population of all states in IITB})}{\text{Maximum value of }\%\text{ of population in IITB}}}$$

   Magnitude is the measure of over or under representation, positive skewness means that the state is over representing and negative is under representation with the mean percent representation of all states in IIT as a reference.



   Just as observed in part 1,states like MH, RJ, AP, TL have highly skewed population(refer to the program file for measurement tables).

   As we can clearly see from the Skew Plot, the plot is not uniformly distributed across all the states, it can be due to various reasons like Population of the state in entire country, GSDP.
   We determined how much is the representation depending on the above listed factors and quantified skew fairness. A simple percentage plot could also do the work to depict the skewness.

   It should be noted that we do not have any official definitions for the terms "skewness" and "fairness." Instead, we relied on our intuition and basic understanding of these words in everyday English and attempted to quantify them in the most sensible way possible. After ideating many iterations of the quantifying functions, we selected the simplest one. We tried explain each and every assumption we made in this project. We tried to conserve a parameter "sign" which is positive if over-representation, negative for under-representation, it plays a major role in all of our skew measures.

$$\text{sign} = \frac{|\text{Percentage in IITB} - \text{Percentage in India}|}{\text{Percentage in IITB} - \text{Percentage in India}}$$

2. **Considering the population and the per capita income of the states, is the distribution of the student body among the states/regions fair?**

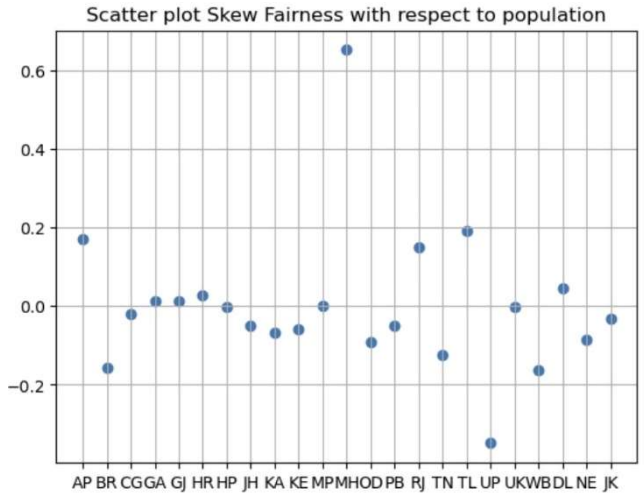
   We determined skew fairness in two situations, one in which we only considered the percentage population of the state in country, and considering both GSDP and percentage population of the state in country (just for a better picture how GSDP affects).

- **Skew Fairness only in terms of % Population of state in India (skewfmp)**
  Our definition for skew fairness in this scenario is the ratio of difference in %representation to band gap of the difference of %representation, i.e. (max % diff-min % diff).

$$\text{diff}_{\text{state}} = \text{Percentage in IITB}_{\text{state}} - \text{Percentage in India}_{\text{state}}$$

$$\text{skewfmp} = \frac{\text{diff}_{\text{state}}}{\max(\text{diff}_{\text{all states}}) - \min(\text{diff}_{\text{all states}})}$$

  Its sign indicates whether the state is over-represented(+ve) or under-represented(-ve). And the magnitude is high (close to 1) means the representation is highly unfair wrt population(Maharashtra). Magnitude closer to 0 is fair (Madhya Pradesh is an example).



Values have been tabulated below for all skew measures.

- **Skew Fairness in terms of both GSDP and % Population of state in India (skewfmg)**
  For this we first normalized the by scaling it to the interval [0 , 1]. We did this for a better definition of our skew fairness, our definition consists a sort of ratio of % diff in representation to GSDP, considering the fact that apples cannot be compared to bananas without assigning a common parameter to both of them, So, after normalizing the ratio makes more sense.

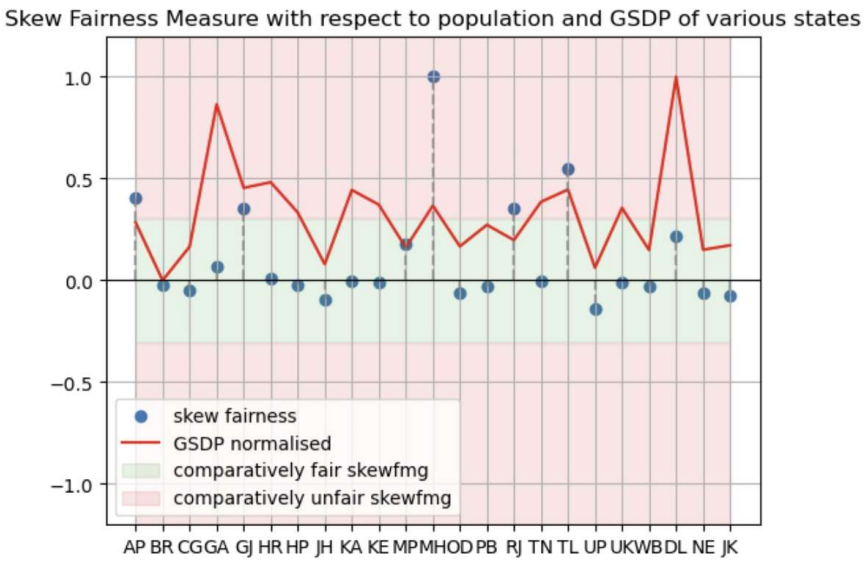
$$\text{Normalized GSDP}_{\text{state}} = \frac{\text{GSDP}_{\text{state}} - \min(\text{GSDP}_{\text{all states}})}{\max(\text{GSDP}_{\text{all states}}) - \min(\text{GSDP}_{\text{all states}})}$$

  Definition for Skew fairness considering GSDP of the states is :

$$\text{skewfmgr} = \left(\frac{\text{skewm}}{1 + \text{GSDP Normalized}}\right)^2$$

$$\text{skewfmg} = \frac{\text{skewfmgr}}{\max(\text{skewfmgr})} \times \text{sign}$$

Extra 1 in the denominator is added to account to the case where normalized GSDP is zero. Square is actually an inspiration from our various iterations of skew measure functions, it is actually redundant, sorry for that. “sign” is preserved to determine over or under representation(simple cube would’ve worked instead of this). Divided by max(skewfmgr) to bring it to the range [-1 , 1].



All Skew Measures, Normalized values :

	State Codes	Percentage in India	Percentage in IITB	skewm	skewfmp	GSDP Per Capita (million Rs.)	GSDP nor	skewfmg
0	AP	4.107602	9.000000	0.549751	0.170401	228702	0.285311	0.404623
1	BR	8.624931	4.133333	-0.197129	-0.156441	62469	0.000000	-0.022548
2	CG	2.116491	1.533333	-0.464958	-0.020311	159097	0.165846	-0.051914
3	GA	0.120845	0.466667	-0.517529	0.012045	566343	0.864816	0.069983
4	GJ	5.007597	5.333333	0.327718	0.011345	327114	0.454219	0.354778
5	HR	2.100439	2.866667	-0.375389	0.026687	343438	0.482237	0.006624
6	HP	0.568751	0.466667	-0.517529	-0.003556	256780	0.333503	-0.019650
7	JH	2.733159	1.266667	-0.479202	-0.051077	108785	0.079494	-0.092704
8	KA	5.061916	3.133333	-0.351355	-0.067172	321256	0.444165	-0.004998
9	KE	2.767786	1.066667	-0.489354	-0.059249	279132	0.371866	-0.008009
10	MP	6.017334	6.066667	0.394481	0.001718	156435	0.161277	0.179219
11	MH	9.310528	28.000000	0.945303	0.650948	276578	0.367483	1.000000
12	OD	3.477681	0.866667	-0.499101	-0.090941	159832	0.167107	-0.063955
13	PB	2.298613	0.866667	-0.499101	-0.049874	221396	0.272772	-0.029723
14	RJ	5.679430	9.933333	0.584299	0.148162	177713	0.197797	0.354922
15	TN	5.977583	2.400000	-0.411273	-0.124606	287092	0.385528	-0.000385
16	TL	2.900152	8.400000	0.525021	0.191558	322511	0.446319	0.547463
17	UP	16.555011	6.533333	0.427364	-0.349052	98819	0.062389	-0.139177
18	UK	0.835677	0.733333	-0.505394	-0.003565	269831	0.355903	-0.012967
19	WB	7.562481	2.866667	-0.375389	-0.163554	149429	0.149252	-0.029672
20	DL	1.390928	2.666667	-0.391577	0.044434	645106	1.000000	0.214795
21	NE	3.768707	1.266667	-0.479202	-0.087145	150144	0.150480	-0.062701
22	JK	1.016358	0.133333	-0.531940	-0.030755	162971	0.172495	-0.074968

Affect of the state on the JEE rank, the graduating CPI, and the first salary

- **To compare the effect of state on JEE Rank, CPI, First Salary, we have grouped students based on states. We have calculated average values of JEE Rank, CPI, First Salary of all students from all states. We found the standard deviation of these quantities.**
- **We know that if standard deviation of a quantity across different states is more, that quantity is more spread out. This means that different states have different average values of that quantity. So we can conclude whether state has an affect on that quantity.**

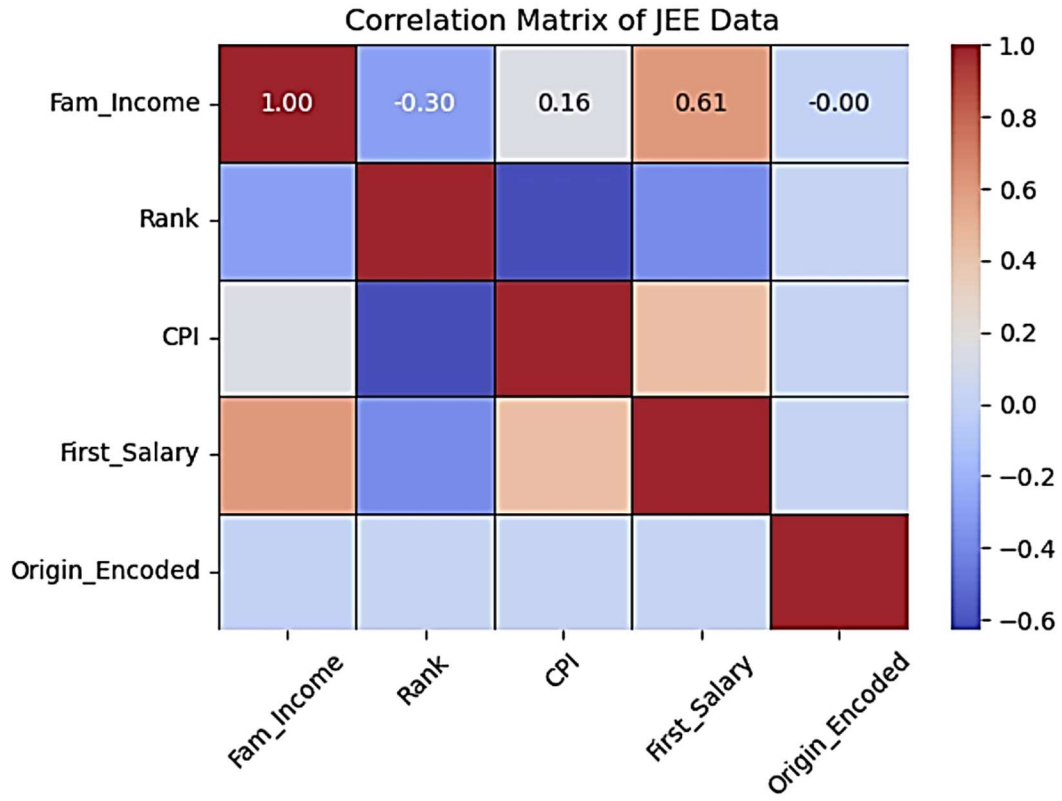
```
Mean values by State:
   Origin         Rank       CPI  First_Salary
0      AP  1078.266667  7.084815  2.383185e+06
1      BR  2030.483871  6.673871  2.301129e+06
2      CG  2325.913043  6.394783  2.228261e+06
3      DL   950.750000  6.835500  2.299250e+06
4      GA  1915.857143  6.045714  2.008571e+06
5      GJ   977.700000  6.943750  2.330625e+06
6      HP  2152.000000  6.347143  2.265714e+06
7      HR  1480.255814  6.683023  2.320233e+06
8      JH  2103.157895  6.543684  2.190000e+06
9      JK  1826.000000  6.325000  2.070000e+06
10     KA  1422.297872  6.727660  2.210000e+06
11     KE  1789.187500  6.490000  2.260625e+06
12     MH   395.126190  7.793167  2.492405e+06
13     MP  1497.912088  6.905385  2.337363e+06
14     NE  2341.631579  6.364211  2.231053e+06
15     OD  2024.923077  6.500000  2.256154e+06
16     PB  1809.538462  6.604615  2.301538e+06
17     RJ  1149.241611  7.036510  2.386846e+06
18     TL   840.682540  7.127381  2.364048e+06
19     TN  1465.777778  6.648889  2.216389e+06
20     UK  2020.727273  6.305455  2.393636e+06
21     UP  1936.530612  6.844082  2.367245e+06
22     WB  1904.627907  6.502791  2.262326e+06

Standard Deviation of JEE Rank across states: 619.0319067609205
Standard Deviation of CPI across states: 0.375536241957845
Standard Deviation of First Salary across states: 316060.662058656
```

- **As the Standard Deviation of JEE Rank across states is high, we can say that there is a moderate to strong affect of the state on JEE Rank.**
- **As the Standard Deviation of CPI across states is low, we can say that there is less affect of the state on CPI.**
- **As the Standard Deviation of First Salary across states is moderate, we can say there is a moderate affect of the state on First Salary.**

Affect of the family income on JEE rank, the graduating CPI, and the first salary

- **We are using correlation matrix in python to find the relation between different quantities.**
- **As states are alphabets, we have to encode them to numerical values in order to compute correlation matrix**
- **If two quantites are positively correlated, then an increase in one quantity causes the other to increase.**
- **If two quantites are negatively correlated, then an increase in one quantity causes the other to decrease.**
- **If there is no correlation, then change in quantity does not have any affect on the other quantity**
- **The correlation coefficient give a measure of how strongly one quantity affects the other(positively or negatively).**

```
Correlation between Family Income and JEE Rank: -0.2997994865500088
Correlation between Family Income and CPI: 0.16441222823085125
Correlation between Family Income and First Salary: 0.6116285040342762
```



- As Correlation between Family Income and JEE Rank is -0.2997994865500088, this means that there is a negative correlation between Family Income and JEE Rank . This is a positive effect of Family Income on JEE Rank as, on an average, higher the family income, better is the JEE Rank. As the correlation coefficient is 0.3, we can say **Family Income** has **low to Moderate effect** on **JEE Rank**.
- As Correlation between Family Income and CPI: 0.16441222823085125, this means that there is a positive correlation between Family Income and CPI. This is a positive effect of Family Income on CPI as, on an average, higher the family income, better is the CPI. As the correlation coefficient is 0.16, we can say **Family Income** has **Low effect** on **CPI**.
- As Correlation between Family Income and First Salary: 0.6116285040342762, this means that there is a positive correlation between Family Income and First Salary. This is a positive effect of Family Income on First Salary as, on an average, higher the family income, better is the First Salary. As the correlation coefficient is 0.61, we can say **Family Income** has **Moderate to High effect** on **First Salary**.

--- End of The Submission---

---Thank You---

q1

September 1, 2024

1 EE325 Programming Assignment-1

1.1 Done by:

Name	Roll Number
Bhavisya Singh Premi	23B1230
B Yaswanth Ram Kumar	23B1277
Harsha Sai Srinivas Pamarthi	23B1202

1.1.1 Part-1

[1]:
```
#Importing requied Libraries
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
```

[2]:
```
# Loading the data
jeed = pd.read_csv('JEEDemographics.csv')

# Sampling functions
def select_first_k_samples(data_list, K):#Selecting first k students from list
    return data_list[:K]
def select_arbitrary_k_samples(data_list, K):#Selecting a point randomly and␣
 ↪selecting k students from there.
    start_index = random.randint(0, len(data_list) - K)
    return data_list[start_index:start_index + K]
def select_random_k_samples(data_list, K):#Selecting k students in the list␣
 ↪randomly
    return random.sample(data_list, K)
```

[3]:
```
origin_state = jeed['Origin']
k_values = [10, 20, 50, 100, 200]
states = origin_state.unique()

# Different marker for each state to distinguish them
```

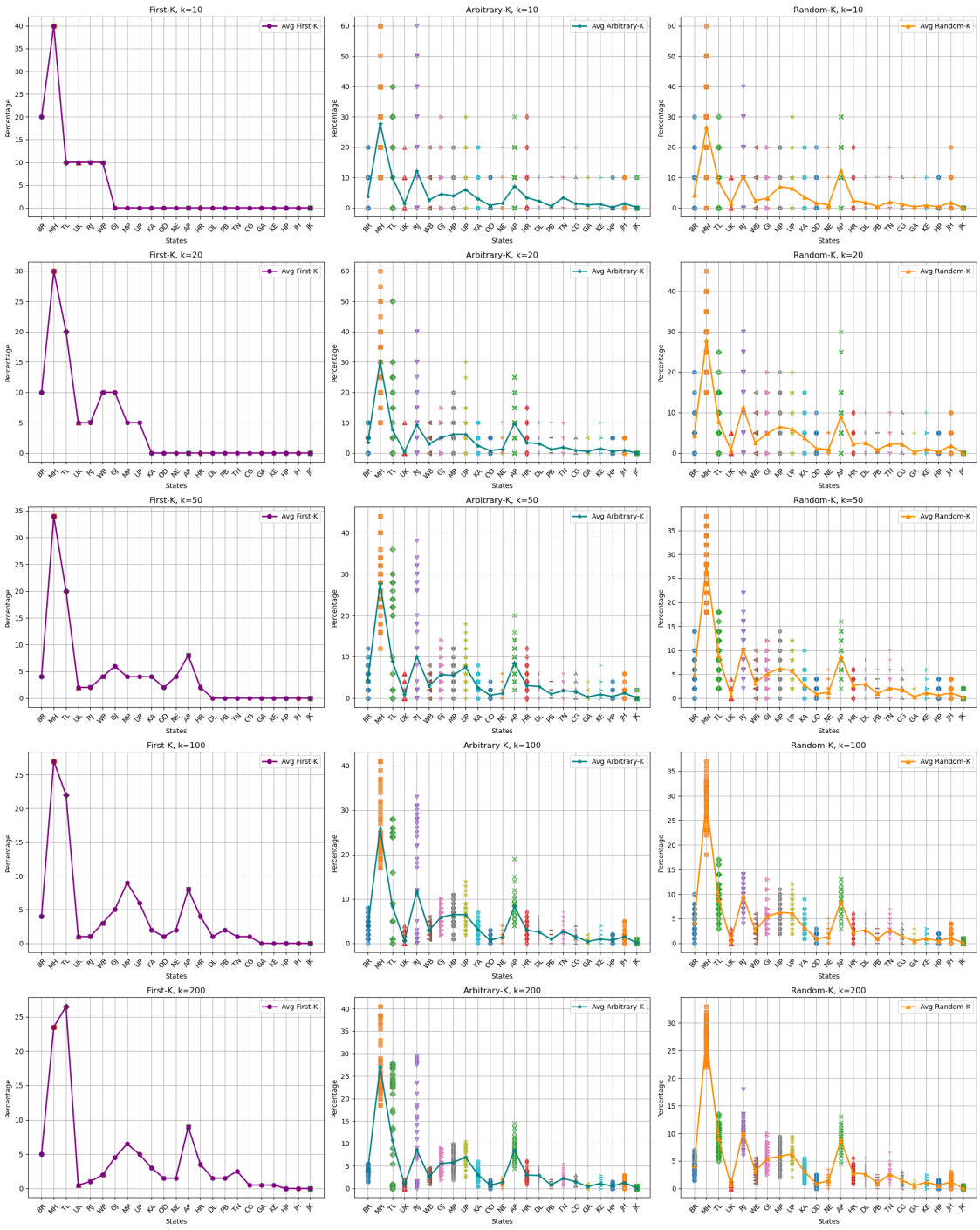
```
markers = ['o', 's', 'D', '^', 'v', '<', '>', 'p', '*', 'h', 'H', '+', 'x',␣
 ↪'d', '|', '_', '1', '2', '3', '4', '8']
state_markers = {state: markers[i % len(markers)] for i, state in␣
 ↪enumerate(states)}

# Initializing dataframes to store iteration data and cumulative sums
iteration_data_first = {k: pd.DataFrame(index=states) for k in k_values}
iteration_data_arbitrary = {k: pd.DataFrame(index=states) for k in k_values}
iteration_data_random = {k: pd.DataFrame(index=states) for k in k_values}
pd.set_option('future.no_silent_downcasting', True)
cumulative_sum_first = pd.DataFrame(index=states, columns=k_values).fillna(0).
 ↪infer_objects(copy=False)
cumulative_sum_arbitrary = pd.DataFrame(index=states, columns=k_values).
 ↪fillna(0).infer_objects(copy=False)
cumulative_sum_random = pd.DataFrame(index=states, columns=k_values).fillna(0).
 ↪infer_objects(copy=False)
```

[4]:
```
# Repeating the experiments 50 times
for v in range(50):
    for k in k_values:
        # First-K
        first_k = select_first_k_samples(origin_state, k)
        state_counts = first_k.value_counts()
        state_percentage = (state_counts / k) * 100
        iteration_data_first[k][v] = state_percentage.reindex(states,␣
 ↪fill_value=0)
        cumulative_sum_first[k] += state_percentage.reindex(states,␣
 ↪fill_value=0)

        # Arbitrary-K
        arbitrary_k = select_arbitrary_k_samples(origin_state, k)
        state_counts = arbitrary_k.value_counts()
        state_percentage = (state_counts / k) * 100
        iteration_data_arbitrary[k][v] = state_percentage.reindex(states,␣
 ↪fill_value=0)
        cumulative_sum_arbitrary[k] += state_percentage.reindex(states,␣
 ↪fill_value=0)

        # Random-K
        random_k = select_random_k_samples(origin_state.to_list(), k)
        state_counts = pd.Series(random_k).value_counts()
        state_percentage = (state_counts / k) * 100
        iteration_data_random[k][v] = state_percentage.reindex(states,␣
 ↪fill_value=0)
        cumulative_sum_random[k] += state_percentage.reindex(states,␣
 ↪fill_value=0)
```

```
# Computing average percentages
average_percentages_first = cumulative_sum_first / 50
average_percentages_arbitrary = cumulative_sum_arbitrary / 50
average_percentages_random = cumulative_sum_random / 50
```

[5]:
```
# Plotting the scatter plots with states on x-axis, percentages on y-axis, and␣
 ↪averages highlighted
fig, axs = plt.subplots(nrows=len(k_values), ncols=3, figsize=(20, 25))  # 5␣
 ↪rows for k values and 3 columns for methods
for i, k in enumerate(k_values):
    # Plot for First-K
    for state in states:
        axs[i, 0].scatter([state]*50, iteration_data_first[k].loc[state],␣
 ↪marker=state_markers[state], alpha=0.75)
    axs[i, 0].plot(states, average_percentages_first[k], color='purple',␣
 ↪marker='o', linestyle='-', linewidth=2, label='Avg First-K')
    axs[i, 0].set_title(f'First-K, k={k}')
    axs[i, 0].set_xlabel('States')
    axs[i, 0].set_ylabel('Percentage')
    axs[i, 0].grid(True)
    axs[i, 0].legend()
    axs[i, 0].tick_params(axis='x', rotation=45)

    # Plot for Arbitrary-K
    for state in states:
        axs[i, 1].scatter([state]*50, iteration_data_arbitrary[k].loc[state],␣
 ↪marker=state_markers[state], alpha=0.75)
    axs[i, 1].plot(states, average_percentages_arbitrary[k], color='teal',␣
 ↪marker='*', linestyle='-', linewidth=2, label='Avg Arbitrary-K')
    axs[i, 1].set_title(f'Arbitrary-K, k={k}')
    axs[i, 1].set_xlabel('States')
    axs[i, 1].set_ylabel('Percentage')
    axs[i, 1].grid(True)
    axs[i, 1].legend()
    axs[i, 1].tick_params(axis='x', rotation=45)

    # Plot for Random-K
    for state in states:
        axs[i, 2].scatter([state]*50, iteration_data_random[k].loc[state],␣
 ↪marker=state_markers[state], alpha=0.75)
    axs[i, 2].plot(states, average_percentages_random[k], color='darkorange',␣
 ↪marker='^', linestyle='-', linewidth=2, label='Avg Random-K')
    axs[i, 2].set_title(f'Random-K, k={k}')
    axs[i, 2].set_xlabel('States')
    axs[i, 2].set_ylabel('Percentage')
```

```
    axs[i, 2].grid(True)
    axs[i, 2].legend()
    axs[i, 2].tick_params(axis='x', rotation=45)

# Adjusting layout and showing plots
plt.tight_layout()
plt.show()
```



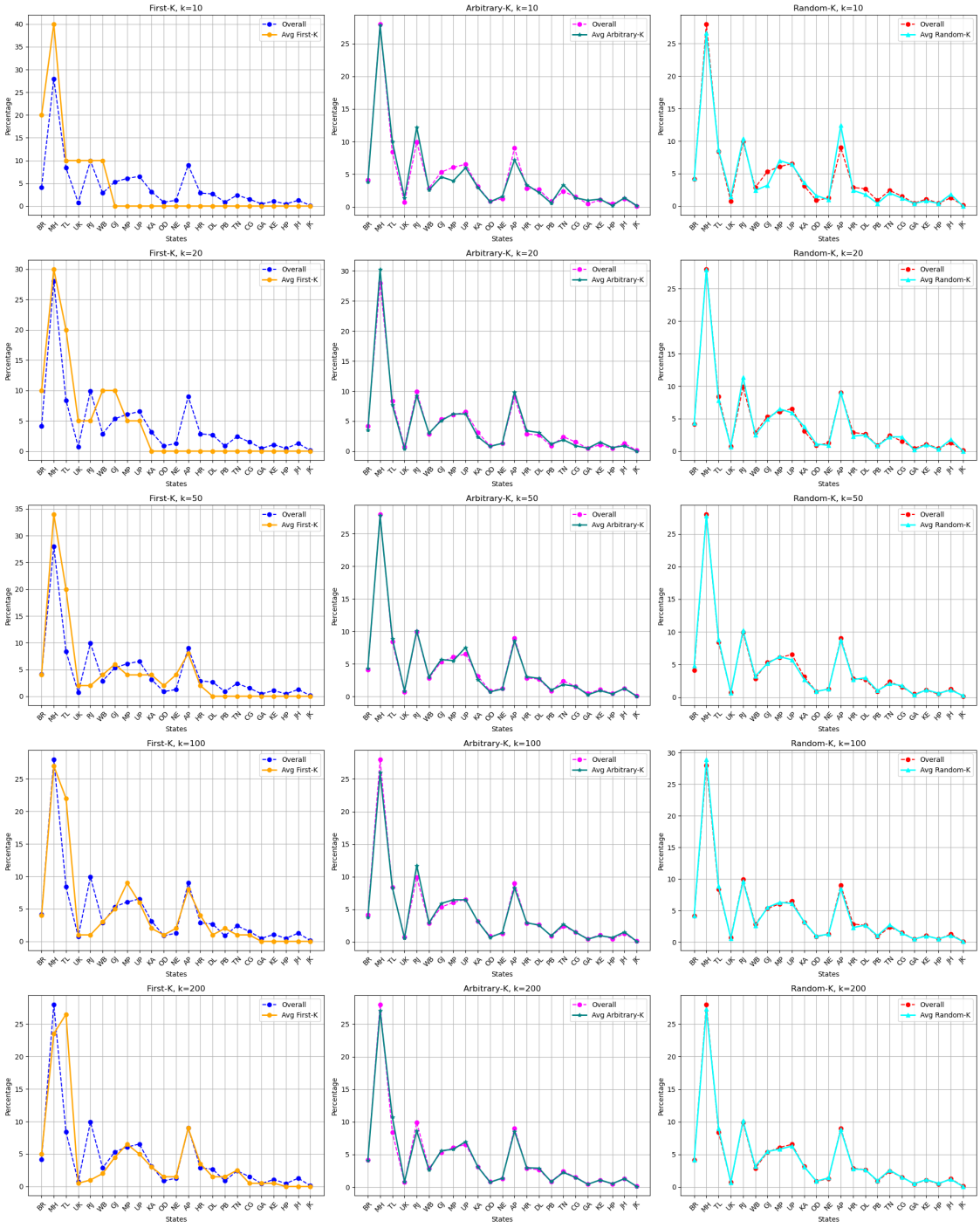
[6]:
```
state_counts=jeed['Origin'].value_counts()
print("The Top 3 states and percentage of students from these states are:")
statepercent=state_counts/15
statepercent.head(3)
```

```
The Top 3 states and percentage of students from these states are:
```

[6]:
```
Origin
MH    28.000000
RJ     9.933333
AP     9.000000
Name: count, dtype: float64
```

[7]:
```
# Calculating the overall percentage of students from each state using the␣
 ↪whole dataset
total_counts = origin_state.value_counts()
overall_percentages = (total_counts / len(origin_state)) * 100

# Plotting the additional graphs
fig, axs = plt.subplots(nrows=len(k_values), ncols=3, figsize=(20, 25))  # 5␣
 ↪rows for k values and 3 columns for methods
for i, k in enumerate(k_values):
    # Plot for First-K
    axs[i, 0].plot(states, overall_percentages.reindex(states, fill_value=0),␣
 ↪color='blue', marker='o',linestyle='--', label='Overall')
    axs[i, 0].plot(states, average_percentages_first[k], color='orange',␣
 ↪marker='o', linestyle='-', linewidth=2, label='Avg First-K')
    axs[i, 0].set_title(f'First-K, k={k}')
    axs[i, 0].set_xlabel('States')
    axs[i, 0].set_ylabel('Percentage')
    axs[i, 0].grid(True)
    axs[i, 0].legend()
    axs[i, 0].tick_params(axis='x', rotation=45)

    # Plot for Arbitrary-K
    axs[i, 1].plot(states, overall_percentages.reindex(states, fill_value=0),␣
 ↪color='magenta', marker='o',linestyle='--', label='Overall')
    axs[i, 1].plot(states, average_percentages_arbitrary[k], color='teal',␣
 ↪marker='*', linestyle='-', linewidth=2, label='Avg Arbitrary-K')
    axs[i, 1].set_title(f'Arbitrary-K, k={k}')
    axs[i, 1].set_xlabel('States')
    axs[i, 1].set_ylabel('Percentage')
    axs[i, 1].grid(True)
    axs[i, 1].legend()
```

```
    axs[i, 1].tick_params(axis='x', rotation=45)

    # Plot for Random-K
    axs[i, 2].plot(states, overall_percentages.reindex(states, fill_value=0),␣
 ↪color='red', marker='o',linestyle='--', label='Overall')
    axs[i, 2].plot(states, average_percentages_random[k], color='cyan',␣
 ↪marker='^', linestyle='-', linewidth=2, label='Avg Random-K')
    axs[i, 2].set_title(f'Random-K, k={k}')
    axs[i, 2].set_xlabel('States')
    axs[i, 2].set_ylabel('Percentage')
    axs[i, 2].grid(True)
    axs[i, 2].legend()
    axs[i, 2].tick_params(axis='x', rotation=45)

# Adjusting layout and showing plots
plt.tight_layout()
plt.show()
```



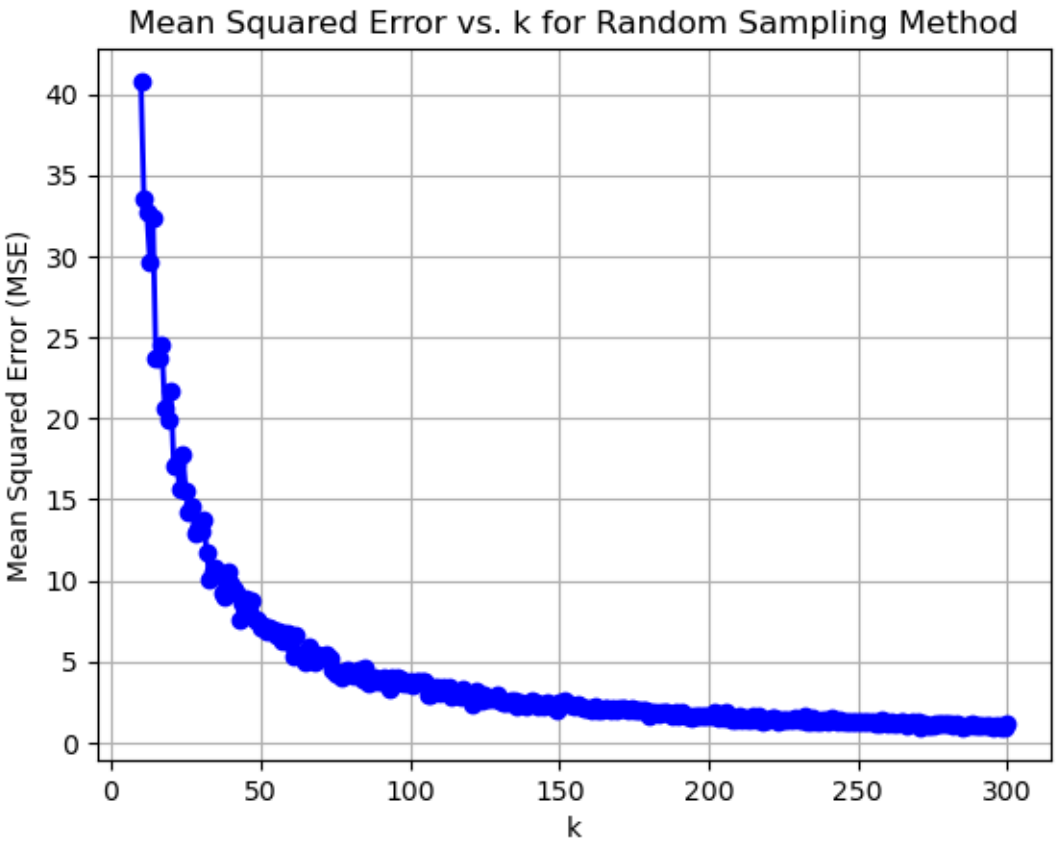
[8]:
```
# Initializing dictionaries to store the errors for different k values for each method
errors_first_k={}
errors_arbitrary_k={}
errors_random_k={}
```

```
# Calculating the errors for the First-K sampling method
for k in k_values:
    errors=[]
    for v in range(50):
        mse=((overall_percentages-iteration_data_first[k].iloc[:, v])**2).mean()
        errors.append(mse)
    errors_first_k[k]=np.mean(errors)

# Calculating the errors for the Arbitrary-K sampling method
for k in k_values:
    errors=[]
    for v in range(50):
        mse=((overall_percentages-iteration_data_arbitrary[k].iloc[:, v])**2).
 ↪mean()
        errors.append(mse)
    errors_arbitrary_k[k]=np.mean(errors)

# Calculating the errors for the Random-K sampling method
for k in k_values:
    errors=[]
    for v in range(50):
        mse=((overall_percentages - iteration_data_random[k].iloc[:, v])**2).
 ↪mean()
        errors.append(mse)
    errors_random_k[k]=np.mean(errors)

# Displaying the errors for different k values for each method
print("Mean Squared Errors for First-K sampling method:")
for k in k_values:
    print(f"k={k}: MSE={errors_first_k[k]}")

print("\nMean Squared Errors for Arbitrary-K sampling method:")
for k in k_values:
    print(f"k={k}: MSE={errors_arbitrary_k[k]}")

print("\nMean Squared Errors for Random-K sampling method:")
for k in k_values:
    print(f"k={k}: MSE={errors_random_k[k]}")
```

```
Mean Squared Errors for First-K sampling method:
k=10: MSE=33.19845410628018
k=20: MSE=17.923091787439606
k=50: MSE=12.085410628019318
k=100: MSE=12.543381642512074
k=200: MSE=19.057874396135272
```

```
Mean Squared Errors for Arbitrary-K sampling method:
k=10: MSE=51.392077294686
k=20: MSE=31.768309178743962
k=50: MSE=19.70419323671498
k=100: MSE=16.681584541062804
k=200: MSE=13.800628019323671

Mean Squared Errors for Random-K sampling method:
k=10: MSE=36.618743961352656
k=20: MSE=18.5712077294686
k=50: MSE=6.9712077294686
k=100: MSE=3.3143961352657003
k=200: MSE=1.7301932367149757
```

[10]:
```python
#Here we are plotting mean square error for different values of k using the␣
 ↪random k picking method.
# Initializing a dictionary to store the errors for each k value between 10 and␣
 ↪300
mse_random_k = {}

# Calculating the MSE for each k value from 10 to 300
for k in range(10, 301):
    errors=[]
    for v in range(50):
        random_k=select_random_k_samples(origin_state.to_list(), k)
        state_counts=pd.Series(random_k).value_counts()
        state_percentage=(state_counts/k)*100
        predicted_percentage=state_percentage.reindex(states, fill_value=0)
        mse=((overall_percentages-predicted_percentage)**2).mean()
        errors.append(mse)
    mse_random_k[k]=np.mean(errors)

# Converting the dictionary to a list of k values and corresponding MSEs
k_values_list=list(mse_random_k.keys())
mse_values_list=list(mse_random_k.values())

# Plot the MSE vs. k
plt.plot(k_values_list, mse_values_list, color='blue', marker='o', linewidth=2)
plt.title('Mean Squared Error vs. k for Random Sampling Method')
plt.xlabel('k')
plt.ylabel('Mean Squared Error (MSE)')
plt.grid(True)
plt.show()
```


Mean Squared Error vs. k for Random Sampling Method
Mean Squared Error (MSE)
k
0
5
10
15
20
25
30
35
40
0
50
100
150
200
250
300

q2itr1

September 1, 2024

1 EE325 Programming Assignment-1

1.1 Done by:

Name	Roll Number
Bhavisya Singh Premi	23B1230
B Yaswanth Ram Kumar	23B1277
Harsha Sai Srinivas Pamarthi	23B1202

1.1.1 Part-2

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
#Calculating percentage of students from each state in India and in IITB
statepop=pd.read_csv('StateInfo.csv',usecols=['State Codes','Population'])
statepop["Percentage"]=(statepop["Population"]/statepop["Population"].sum())*100
statepop=statepop.drop("Population",axis=1)
iitbpop=pd.read_csv("JEEDemographics.csv",usecols=["Origin"])
iitbpop.rename(columns={'Origin': 'State Codes'}, inplace=True)
iitbpop=iitbpop.value_counts()/15
#Making a single dataframe
pop = pd.merge(statepop, iitbpop, on='State Codes', how='inner')
pop.rename(columns={'Percentage': 'Percentage in India','count':'Percentage in␣
 ↪IITB'}, inplace=True)
pop
```

[2]:

	State Codes	Percentage in India	Percentage in IITB
0	AP	4.107602	9.000000
1	BR	8.624931	4.133333
2	CG	2.116491	1.533333
3	GA	0.120845	0.466667
4	GJ	5.007597	5.333333
5	HR	2.100439	2.866667
6	HP	0.568751	0.466667

7	JH	2.733159	1.266667
8	KA	5.061916	3.133333
9	KE	2.767786	1.066667
10	MP	6.017334	6.066667
11	MH	9.310528	28.000000
12	OD	3.477681	0.866667
13	PB	2.298613	0.866667
14	RJ	5.679430	9.933333
15	TN	5.977583	2.400000
16	TL	2.900152	8.400000
17	UP	16.555011	6.533333
18	UK	0.835677	0.733333
19	WB	7.562481	2.866667
20	DL	1.390928	2.666667
21	NE	3.768707	1.266667
22	JK	1.016358	0.133333

1.2 We defined skewness of a particular state as

$$\sqrt[3]{\frac{(\%\text{ of population of that state in IITB} - \text{Mean of }\%\text{ of population of all states in IITB})}{\text{Maximum value of }\%\text{ of population in IITB}}}$$

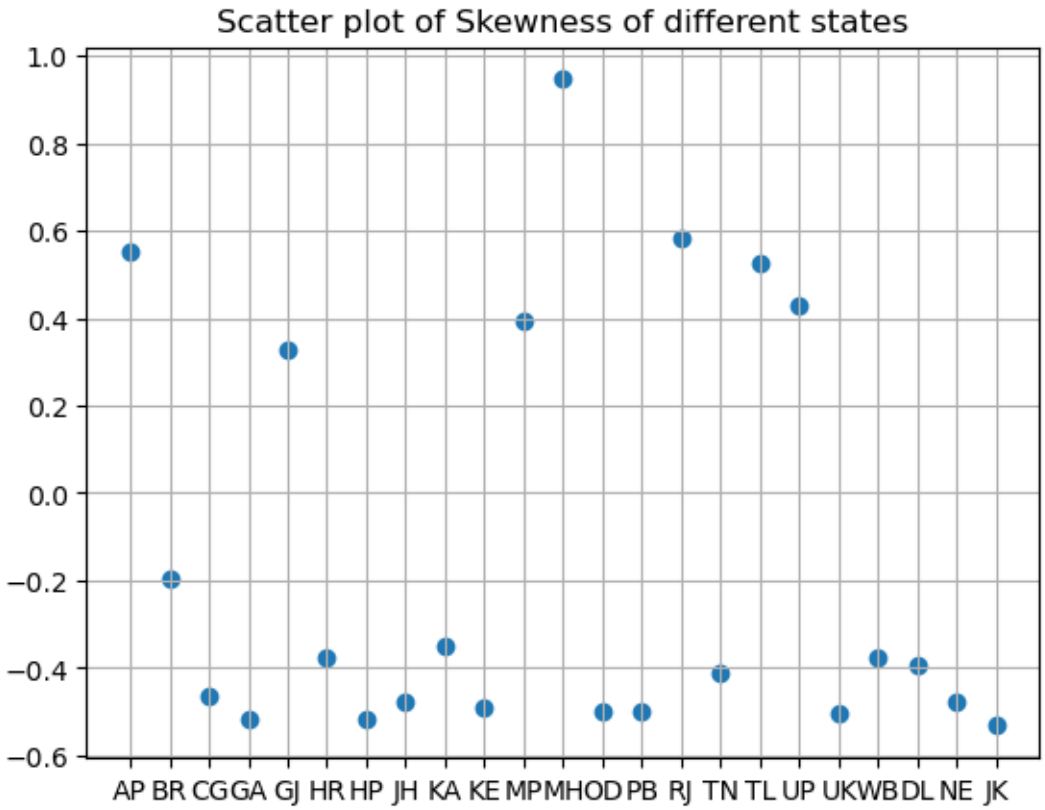
[3]:
```
#Calculating Skewness of the data
pop["skewm"]=np.cbrt((pop["Percentage in IITB"]-pop["Percentage in IITB"].
↪mean())/pop["Percentage in IITB"].max())
pop
```

[3]:

	State Codes	Percentage in India	Percentage in IITB	skewm
0	AP	4.107602	9.000000	0.549751
1	BR	8.624931	4.133333	-0.197129
2	CG	2.116491	1.533333	-0.464958
3	GA	0.120845	0.466667	-0.517529
4	GJ	5.007597	5.333333	0.327718
5	HR	2.100439	2.866667	-0.375389
6	HP	0.568751	0.466667	-0.517529
7	JH	2.733159	1.266667	-0.479202
8	KA	5.061916	3.133333	-0.351355
9	KE	2.767786	1.066667	-0.489354
10	MP	6.017334	6.066667	0.394481
11	MH	9.310528	28.000000	0.945303
12	OD	3.477681	0.866667	-0.499101
13	PB	2.298613	0.866667	-0.499101
14	RJ	5.679430	9.933333	0.584299
15	TN	5.977583	2.400000	-0.411273
16	TL	2.900152	8.400000	0.525021
17	UP	16.555011	6.533333	0.427364

```
18          UK              0.835677            0.733333 -0.505394
19          WB              7.562481            2.866667 -0.375389
20          DL              1.390928            2.666667 -0.391577
21          NE              3.768707            1.266667 -0.479202
22          JK              1.016358            0.133333 -0.531940
```

[4]:
```
#Scatter plot of Skewness of different states
plt.title("Scatter plot of Skewness of different states")
plt.scatter(pop["State Codes"],pop["skewm"])
plt.grid()
plt.show()
```



1.3 We Defined the Skew Fairness Measure with Respect to the Population of a Particular State

$$\text{diff}_{\text{state}} = \text{Percentage in IITB}_{\text{state}} - \text{Percentage in India}_{\text{state}}$$

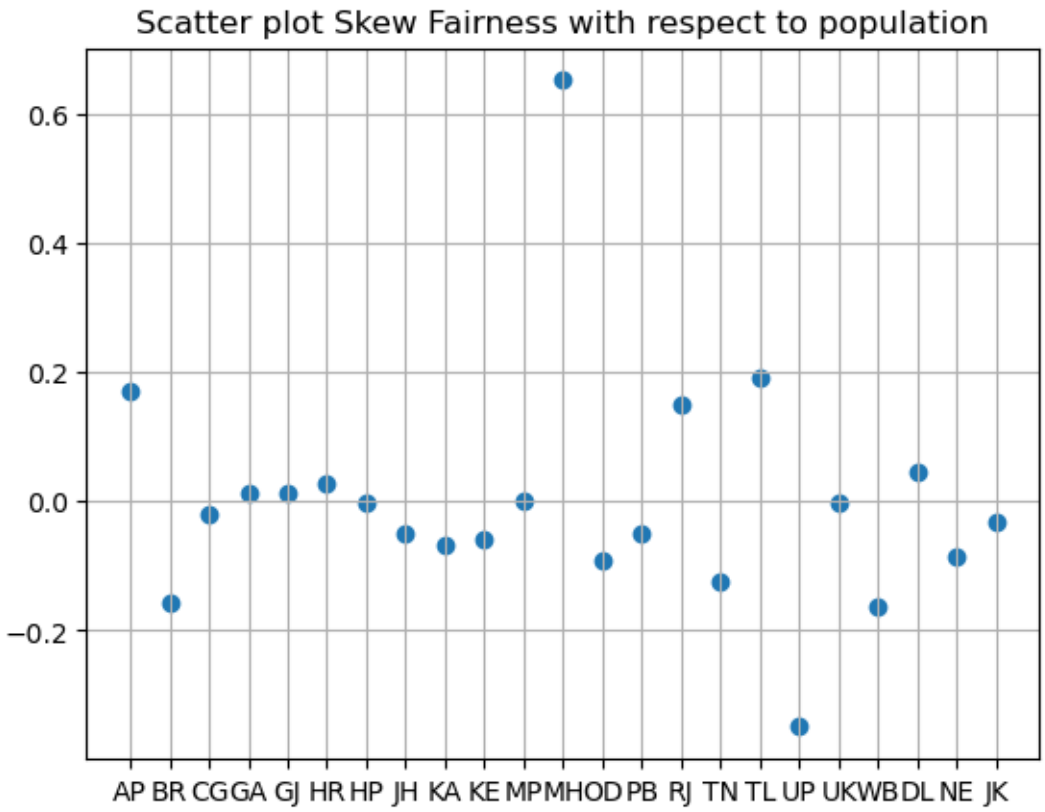
$$\text{skewfmp} = \frac{\text{diff}_{\text{state}}}{\max(\text{diff}_{\text{all states}}) - \min(\text{diff}_{\text{all states}})}$$

[5]:
```
pop["skewfmp"]=(pop["Percentage in IITB"]-pop["Percentage in India"])
pop["skewfmp"]=pop["skewfmp"]/(pop["skewfmp"].max()-pop["skewfmp"].min())
pop
```

[5]:

	State Codes	Percentage in India	Percentage in IITB	skewm	skewfmp
0	AP	4.107602	9.000000	0.549751	0.170401
1	BR	8.624931	4.133333	-0.197129	-0.156441
2	CG	2.116491	1.533333	-0.464958	-0.020311
3	GA	0.120845	0.466667	-0.517529	0.012045
4	GJ	5.007597	5.333333	0.327718	0.011345
5	HR	2.100439	2.866667	-0.375389	0.026687
6	HP	0.568751	0.466667	-0.517529	-0.003556
7	JH	2.733159	1.266667	-0.479202	-0.051077
8	KA	5.061916	3.133333	-0.351355	-0.067172
9	KE	2.767786	1.066667	-0.489354	-0.059249
10	MP	6.017334	6.066667	0.394481	0.001718
11	MH	9.310528	28.000000	0.945303	0.650948
12	OD	3.477681	0.866667	-0.499101	-0.090941
13	PB	2.298613	0.866667	-0.499101	-0.049874
14	RJ	5.679430	9.933333	0.584299	0.148162
15	TN	5.977583	2.400000	-0.411273	-0.124606
16	TL	2.900152	8.400000	0.525021	0.191558
17	UP	16.555011	6.533333	0.427364	-0.349052
18	UK	0.835677	0.733333	-0.505394	-0.003565
19	WB	7.562481	2.866667	-0.375389	-0.163554
20	DL	1.390928	2.666667	-0.391577	0.044434
21	NE	3.768707	1.266667	-0.479202	-0.087145
22	JK	1.016358	0.133333	-0.531940	-0.030755

[6]:
```
#Scatter plot Skew Fairness with respect to population
plt.title("Scatter plot Skew Fairness with respect to population")
plt.scatter(pop["State Codes"],pop["skewfmp"])
plt.grid()
plt.show()
```



1.4 We normalized GSDP per capita of all states

Normalization of GSDP per capita:

$$\text{Normalized GSDP}_{\text{state}} = \frac{\text{GSDP}_{\text{state}} - \min(\text{GSDP}_{\text{all states}})}{\max(\text{GSDP}_{\text{all states}}) - \min(\text{GSDP}_{\text{all states}})}$$

We defined the Skew Fairness Measure with respect to population and GSDP (skewfmg):

$$\text{sign} = \frac{|\text{Percentage in IITB} - \text{Percentage in India}|}{\text{Percentage in IITB} - \text{Percentage in India}}$$

$$\text{skewfmgr} = \left(\frac{\text{skewm}}{1 + \text{GSDP Normalized}}\right)^2$$

$$\text{skewfmg} = \frac{\text{skewfmgr}}{\max(\text{skewfmgr})} \times \text{sign}$$

[7]:
```
#Adding the GSDP per capita data into the population table
stategdp=pd.read_csv('StateInfo.csv',usecols=['State Codes','GSDP Per Capita␣
 ↪(million Rs.)'])
stategdp=pd.merge(pop, stategdp , on='State Codes', how='inner')
#Normalizing the GDP using min-max normalisation
stategdp["GSDP nor"]=(stategdp['GSDP Per Capita (million Rs.)']-stategdp['GSDP␣
 ↪Per Capita (million Rs.)'].min())/(stategdp['GSDP Per Capita (million Rs.)'].
 ↪max()-stategdp['GSDP Per Capita (million Rs.)'].min())
sign=(abs(pop["Percentage in IITB"]-pop["Percentage in India"]))/
 ↪(pop["Percentage in IITB"]-pop["Percentage in India"])
#Computing the Skew Fairness Measure with respect to population and␣
 ↪GSDP(skewfmg)
stategdp["skewfmg"]=(stategdp["skewm"]/1+stategdp["GSDP nor"])**2
stategdp["skewfmg"]=(stategdp["skewfmg"])/(stategdp["skewfmg"].max())*sign
stategdp
```

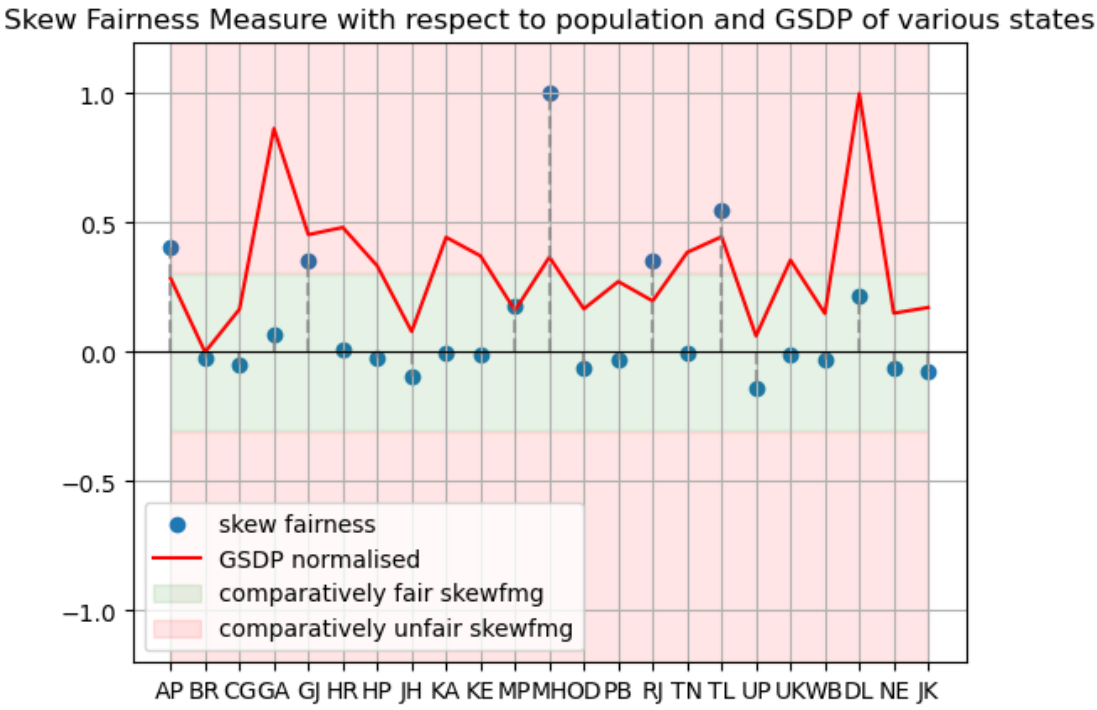
[7]:

	State Codes	Percentage in India	Percentage in IITB	skewm	skewfmp	\
0	AP	4.107602	9.000000	0.549751	0.170401	
1	BR	8.624931	4.133333	-0.197129	-0.156441	
2	CG	2.116491	1.533333	-0.464958	-0.020311	
3	GA	0.120845	0.466667	-0.517529	0.012045	
4	GJ	5.007597	5.333333	0.327718	0.011345	
5	HR	2.100439	2.866667	-0.375389	0.026687	
6	HP	0.568751	0.466667	-0.517529	-0.003556	
7	JH	2.733159	1.266667	-0.479202	-0.051077	
8	KA	5.061916	3.133333	-0.351355	-0.067172	
9	KE	2.767786	1.066667	-0.489354	-0.059249	
10	MP	6.017334	6.066667	0.394481	0.001718	
11	MH	9.310528	28.000000	0.945303	0.650948	
12	OD	3.477681	0.866667	-0.499101	-0.090941	
13	PB	2.298613	0.866667	-0.499101	-0.049874	
14	RJ	5.679430	9.933333	0.584299	0.148162	
15	TN	5.977583	2.400000	-0.411273	-0.124606	
16	TL	2.900152	8.400000	0.525021	0.191558	
17	UP	16.555011	6.533333	0.427364	-0.349052	
18	UK	0.835677	0.733333	-0.505394	-0.003565	
19	WB	7.562481	2.866667	-0.375389	-0.163554	
20	DL	1.390928	2.666667	-0.391577	0.044434	
21	NE	3.768707	1.266667	-0.479202	-0.087145	
22	JK	1.016358	0.133333	-0.531940	-0.030755	

	GSDP Per Capita (million Rs.)	GSDP nor	skewfmg
0	228702	0.285311	0.404623
1	62469	0.000000	-0.022548
2	159097	0.165846	-0.051914
3	566343	0.864816	0.069983
4	327114	0.454219	0.354778

```
5                           343438  0.482237  0.006624
6                           256780  0.333503 -0.019650
7                           108785  0.079494 -0.092704
8                           321256  0.444165 -0.004998
9                           279132  0.371866 -0.008009
10                          156435  0.161277  0.179219
11                          276578  0.367483  1.000000
12                          159832  0.167107 -0.063955
13                          221396  0.272772 -0.029723
14                          177713  0.197797  0.354922
15                          287092  0.385528 -0.000385
16                          322511  0.446319  0.547463
17                           98819  0.062389 -0.139177
18                          269831  0.355903 -0.012967
19                          149429  0.149252 -0.029672
20                          645106  1.000000  0.214795
21                          150144  0.150480 -0.062701
22                          162971  0.172495 -0.074968
```

[8]:
```
#Plotting skewfmg for various states
plt.scatter(stategdp["State Codes"],stategdp["skewfmg"],label='skew fairness')
plt.plot(stategdp["State Codes"],stategdp["GSDP nor"],color='red',label='GSDP␣
 ↪normalised')
plt.vlines(stategdp["State Codes"], ymin=0, ymax=stategdp["skewfmg"],␣
 ↪colors='gray', linestyles='--', alpha=0.7)
plt.ylim(-1.2, +1.2)
plt.axhline(0, color='black', linewidth=0.8)
plt.title("Skew Fairness Measure with respect to population and GSDP of various␣
 ↪states")
plt.fill_between(x=range(len(stategdp)), y1=0.3, y2=-0.3, color='green',␣
 ↪alpha=0.1,label='comparatively fair skewfmg')
plt.fill_between(x=range(len(stategdp)), y1=0.31, y2=1.2, color='red', alpha=0.
 ↪1,label='comparatively unfair skewfmg')
plt.fill_between(x=range(len(stategdp)), y1=-0.31, y2=-1.2, color='red',␣
 ↪alpha=0.1)
plt.grid()
plt.legend()
plt.show()
```



1.5 Affect of the state on the JEE rank, the graduating CPI, and the first salary

- **To compare the effect of state on JEE Rank, CPI, First Salary, we have grouped students based on states. We have calculated average values of JEE Rank, CPI, First Salary of all students from all states. We found the standard deviation of these quantities.**
- **We know that if standard deviation of a quantity across different states is more, that quantity is more spread out. This means that different states have different average values of that quantity. So we can conclude whether state has an affect on that quantity.**

[9]:
```
jeed=pd.read_csv("JEEDemographics.csv")

# Grouping the data by 'Origin' and calculate the mean for Rank, CPI, and First␣
 ↪Salary
grouped = jeed.groupby('Origin').agg({
    'Rank': 'mean',
    'CPI': 'mean',
    'First_Salary': 'mean'
}).reset_index()

# Calculating the overall standard deviation across states
rank_std_dev = np.sqrt(jeed.groupby('Origin')['Rank'].var().mean())
cpi_std_dev = np.sqrt(jeed.groupby('Origin')['CPI'].var().mean())
```

```
salary_std_dev = np.sqrt(jeed.groupby('Origin')['First_Salary'].var().mean())

# Printing the mean values grouped by state
print("Mean values by State:")
print(grouped)

# Printing the standard deviation for each factor
print("\nStandard Deviation of JEE Rank across states:", rank_std_dev)
print("Standard Deviation of CPI across states:", cpi_std_dev)
print("Standard Deviation of First Salary across states:", salary_std_dev)
```

```
Mean values by State:
   Origin         Rank       CPI  First_Salary
0      AP  1078.266667  7.084815  2.383185e+06
1      BR  2030.483871  6.673871  2.301129e+06
2      CG  2325.913043  6.394783  2.228261e+06
3      DL   950.750000  6.835500  2.299250e+06
4      GA  1915.857143  6.045714  2.008571e+06
5      GJ   977.700000  6.943750  2.330625e+06
6      HP  2152.000000  6.347143  2.265714e+06
7      HR  1480.255814  6.683023  2.320233e+06
8      JH  2103.157895  6.543684  2.190000e+06
9      JK  1826.000000  6.325000  2.070000e+06
10     KA  1422.297872  6.727660  2.210000e+06
11     KE  1789.187500  6.490000  2.260625e+06
12     MH   395.126190  7.793167  2.492405e+06
13     MP  1497.912088  6.905385  2.337363e+06
14     NE  2341.631579  6.364211  2.231053e+06
15     OD  2024.923077  6.500000  2.256154e+06
16     PB  1809.538462  6.604615  2.301538e+06
17     RJ  1149.241611  7.036510  2.386846e+06
18     TL   840.682540  7.127381  2.364048e+06
19     TN  1465.777778  6.648889  2.216389e+06
20     UK  2020.727273  6.305455  2.393636e+06
21     UP  1936.530612  6.844082  2.367245e+06
22     WB  1904.627907  6.502791  2.262326e+06

Standard Deviation of JEE Rank across states: 619.0319067609205
Standard Deviation of CPI across states: 0.375536241957845
Standard Deviation of First Salary across states: 316060.662058656
```

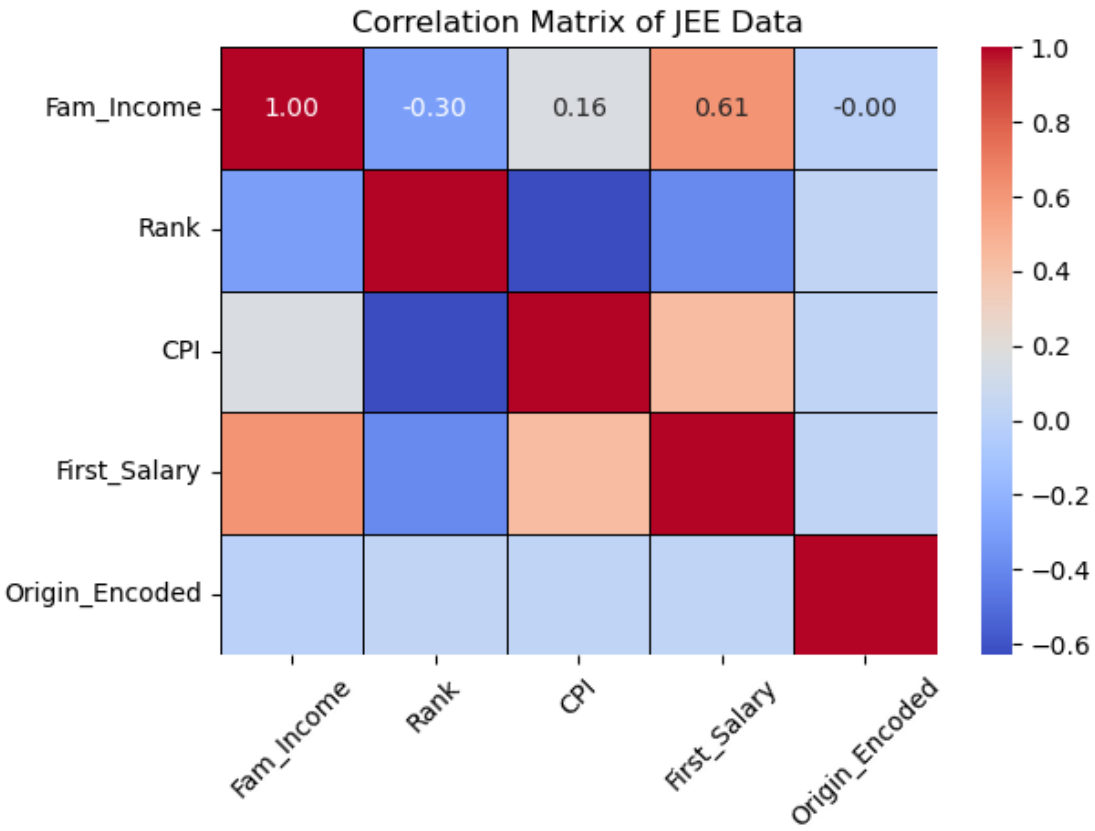
- **As the Standard Deviation of JEE Rank across states is high, we can say that there is a moderate to strong affect of the state on JEE Rank.**
- **As the Standard Deviation of CPI across states is low, we can say that there is less affect of the state on CPI.**
- **As the Standard Deviation of First Salary across states is moderate, we can say there is a moderate affect of the state on First Salary.**

1.6 Affect of the family income on JEE rank, the graduating CPI, and the first salary

- **We are using correlation matrix in python to find the relation between different quantities.**
- **As states are alphabets, we have to encode them to numerical values in order to compute correlation matrix**
- **If two quantites are positively correlated, then an increase in one quantity causes the other to increase.**
- **If two quantites are negatively correlated, then an increase in one quantity causes the other to decrease.**
- **If there is no correlation, then change in quantity does not have any affect on the other quantity**
- **The correlation coefficient give a measure of how strongly one quantity affects the other(positively or negatively).**

[10]:
```
# Encoding the 'Origin' column to numerical values
jeed['Origin_Encoded'] = jeed['Origin'].astype('category').cat.codes

# Computing the correlation matrix
correlation_matrix = jeed[['Fam_Income', 'Rank', 'CPI', 'First_Salary',␣
 ↪'Origin_Encoded']].corr()

# Printing specific correlations
print("Correlation between Family Income and JEE Rank:", correlation_matrix.
 ↪loc['Fam_Income', 'Rank'])
print("Correlation between Family Income and CPI:", correlation_matrix.
 ↪loc['Fam_Income', 'CPI'])
print("Correlation between Family Income and First Salary:", correlation_matrix.
 ↪loc['Fam_Income', 'First_Salary'])

# Visualizing the correlation matrix using a heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5, linecolor='black')
plt.title('Correlation Matrix of JEE Data')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
Correlation between Family Income and JEE Rank: -0.2997994865500088
Correlation between Family Income and CPI: 0.16441222823085125
Correlation between Family Income and First Salary: 0.6116285040342762
```



- As Correlation between Family Income and JEE Rank is -0.2997994865500088, this means that there is a negative correlation between Family Income and JEE Rank . This is a positive effect of Family Income on JEE Rank as, on an average, higher the family income, better is the JEE Rank. As the correlation coefficient is 0.3, we can say **Family Income** has **low to Moderate effect** on **JEE Rank**.
- As Correlation between Family Income and CPI: 0.16441222823085125, this means that there is a positive correlation between Family Income and CPI. This is a positive effect of Family Income on CPI as, on an average, higher the family income, better is the CPI. As the correlation coefficient is 0.16, we can say **Family Income** has **Low effect** on **CPI**.
- As Correlation between Family Income and First Salary: 0.6116285040342762, this means that there is a positive correlation between Family Income and First Salary. This is a positive effect of Family Income on First Salary as, on an average, higher the family income, better is the First Salary. As the correlation coefficient is 0.61, we can say **Family Income** has **Moderate to High effect** on **First Salary**.