Name (Last, First): Kunjangada Arun Sushma

Student ID: 002473132

Assignment 2

NEU_COE_INFO6105_Fall2024

Instructions:

1. For answering **programming questions**, please use Adobe Acrobat to edit the pdf file in two steps **[See Appendix: Example Question and Answer]**:
    a. Copy and paste your R or python code as text in the box provided (so that your teaching team can run your code);
    b. Screenshot your R or python console outputs, save them as a .PNG image file, and paste/insert them in the box provided.
    c. Show all work - credit will not be given for code without showing the code in action by including the screenshot of R or python console outputs.
2. To answer **non-programming questions**, please type or handwrite your final answers clearly in the boxes. Show all work - credit will not be given for numerical solutions that appear without explanation in the space above the boxes. **You're encouraged to use R or python to graph/plot the data and produce numerical summaries; please** append **your code and screenshot of the outputs at the end of your pdf submission.**
3. **[Total 129 pts = 9 + 21 + 48 + 39 pts + 12 Extra Credit pts]**

Grading Rubric

Each question is worth 3 points and will be graded as follows:

3 points: Correct answer with work shown

2 points: Incorrect answer but attempt shows some understanding (work shown)

1 point: Incorrect answer but an attempt was made (work shown), or **correct answer without explanation (work not shown)**

0 points: Left blank or made little to no effort/work not shown

Reflective Journal [3 pts]

(Copy and paste the link to your live Google doc in the box below)

https://docs.google.com/document/d/1ptEhnYHniNtT1yxDPcvXK7LpJaPGzi80BCSGZZhom7Y/edit?usp=sharing

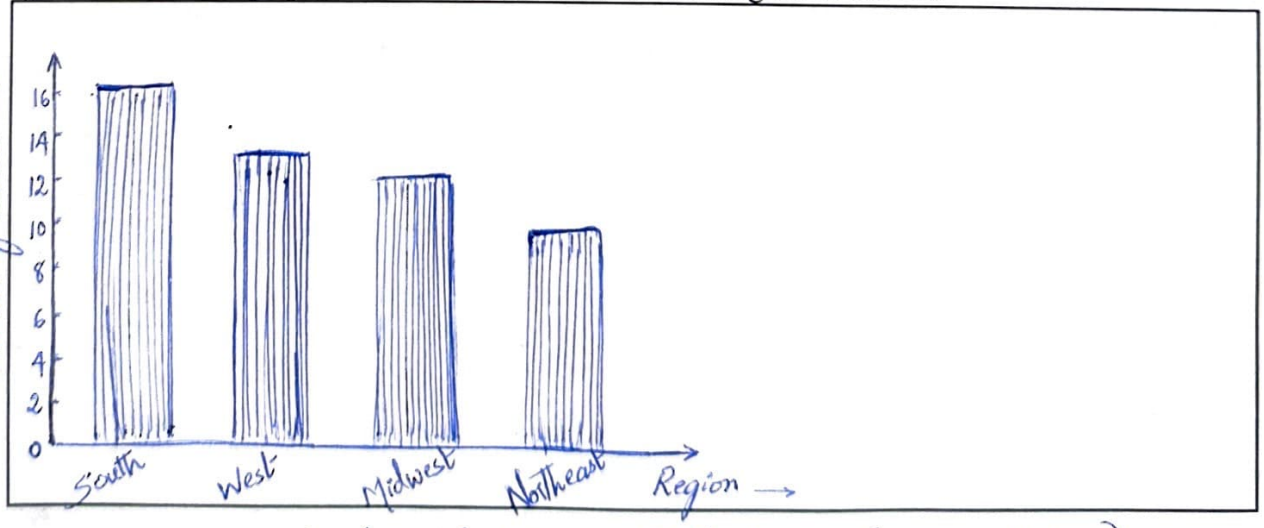
Part I. Categorical Data [9 pts = 3 x 3 pts]

1) Below is a list of unorganized data from the 2020 election. This gives the state, their geographical location, and the voting majority. [The data is accessible via a .csv file saved under the course shared folder "Course Contents".]

State	*Location*	*Electoral College Vote in 2020*
AL	South	Republican
AK	West	Republican
AZ	West	Democrat
AR	South	Republican
CA	West	Democrat
CO	West	Democrat
CT	Northeast	Democrat
DE	South	Democrat
FL	South	Republican
GA	South	Democrat
HI	West	Democrat
ID	West	Republican
IL	Midwest	Democrat
IN	Midwest	Republican
IA	Midwest	Republican
KS	Midwest	Republican
KY	South	Republican
LA	South	Republican
ME	Northeast	Democrat*
MD	South	Democrat
MA	Northeast	Democrat
MI	Midwest	Democrat
MN	Midwest	Democrat
MS	South	Republican
MO	Midwest	Republican
MT	West	Republican
NE	Midwest	Republican*
NV	West	Democrat
NH	Northeast	Democrat
NJ	Northeast	Democrat
NM	West	Democrat
NY	Northeast	Democrat
NC	South	Republican
ND	Midwest	Republican
OH	Midwest	Republican
OK	South	Republican
OR	West	Democrat
PA	Northeast	Democrat
RI	Northeast	Democrat
SC	South	Republican
SD	Midwest	Republican
TN	South	Republican
TX	South	Republican
UT	West	Republican
VT	Northeast	Democrat
VA	South	Democrat
WA	West	Democrat
WV	South	Republican
WI	Midwest	Democrat
WY	West	Republican

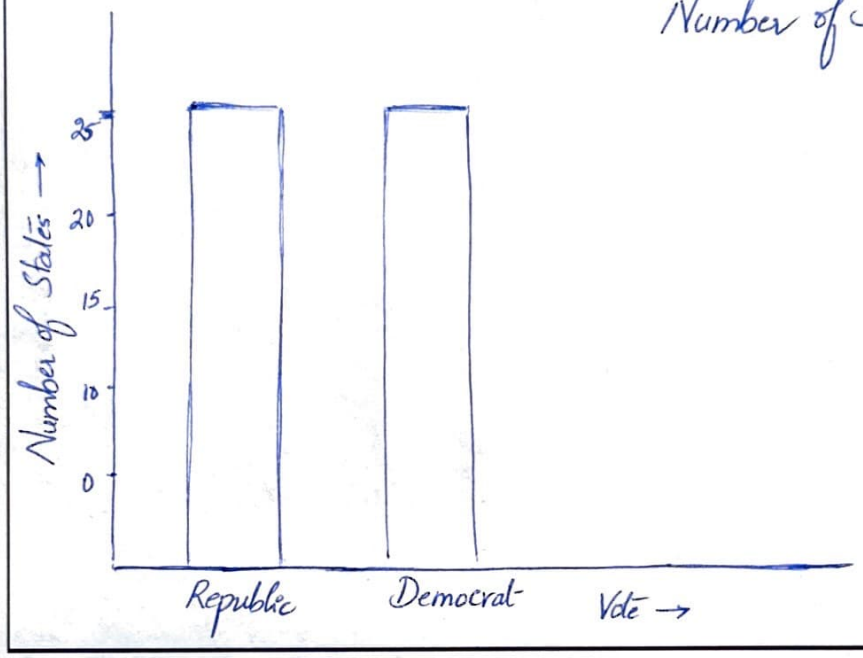
*Split Votes

a) Construct a bar graph of the amount of states in each region of the US.



Number of States in Each Region (2020 Election)

b) Construct a bar graph of the amount of states who voted either republican or democrat in the 2020 election.



The graph shows the number of states that voted either Republic or Democratic in the 2020 election.

c) Create a two way table showing the distribution of region and political party.

Region	Democrat	Republic
Midwest	4	8
Northeast	9	0
South	4	12
West	8	5

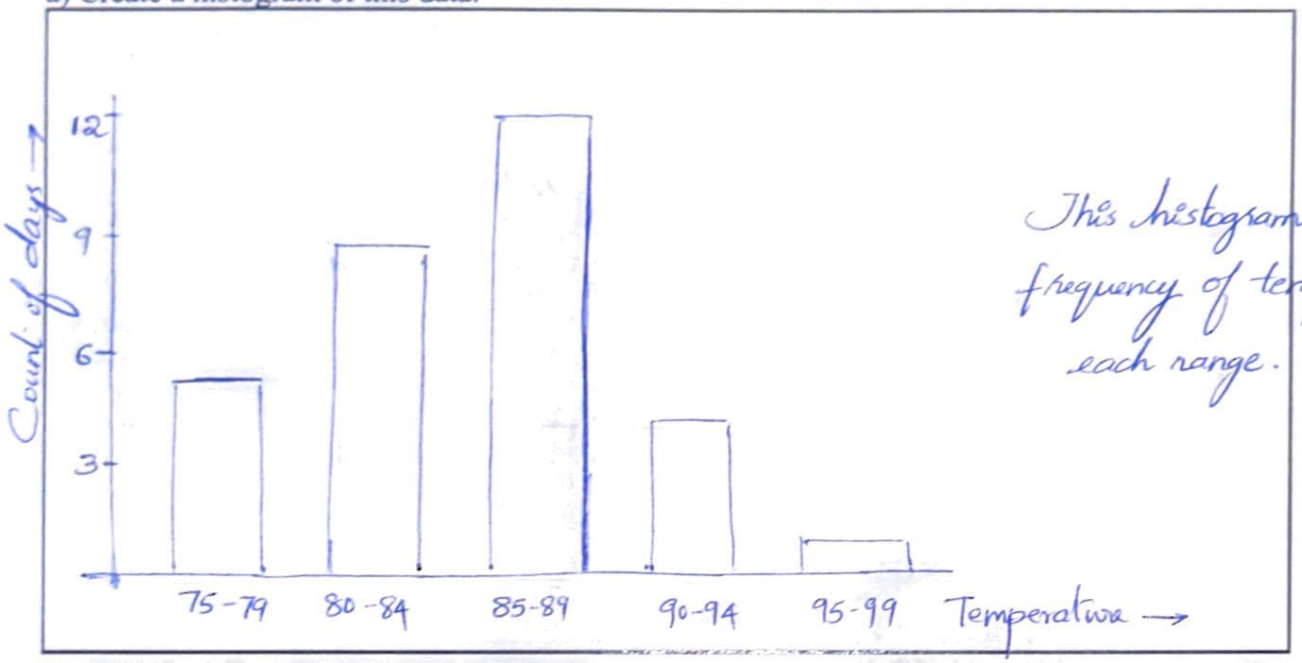
2-way table presents destribution of states by region and voting majority

Part II. Quantitative Data ([21 pts = 7 x 3 pts]

1) The following are the temperature highs for an Illinois town in August 2022.

78	75	83	85	87	91	78	91	85	84	80	79	83	85	86
80	88	94	89	86	88	84	81	85	83	80	79	87	88	91
95														

a) Create a histogram of this data.



This histogram displays the frequency of temperatures within each range.

b) Create a stem and leaf plot for this data.

7	5 8 8 9 9
8	0 0 0 1 3 3 3 4 4 5 5 5 6 6 7 7 8 8 8 9
9	1 1 1 4 5

7	8 5 8 9 9
8	3 5 7 5 4 0 3 5 6 0 8 9 6 8 4 1 5 3 0 7 8
9	1 1 4 1 5

Key: 8|3 represents 83°F, the temperature highs for an Illinois town.

c) Which graph do you think displays the data the best? Why?

Both histogram and the stem-leaf plot have their merits, but I belive stem & leaf plot displays the data set better.
- Stem and leaf plot preserves the exact values of data point, allowing us to see more datapoints.
- The stem-and-leaf plot makes it easier to see the clustering of temperatures.
- The stem-and-leaf plot makes it easier to identify outliers.
- With stem-and-leaf plot it's easier to quickly estimate the median and mode, and get the range of the data.

2) 30 people were asked how many miles, to the nearest mile, they commute to work each day:

2	5	7	3	2	10	18	15	20	7	10	18	5	12	13
12	4	5	10	7	10	12	7	3	4	11	13	14	5	6

a) Create a cumulative relative frequency graph (include your table!)

Miles	Frequency	Cummulative Frequency	Relative Frequency	Cummulative Relative Frequency
2	2	2	2/30 = 0.067	0.067
3	2	4	2/30 = 0.067	0.067+0.067=0.133
4	2	6	2/30 = 0.067	0.133+0.067=0.200
5	4	10	4/30 = 0.133	0.200+0.133 = 0.333
6	1	11	1/30 = 0.033	0.333+0.033 = 0.367
7	4	15	4/30 = 0.133	0.367+0.133 = 0.500
10	4	19	4/30 = 0.133	0.500+0.133 = 0.633
11	1	20	1/30 = 0.033	0.633+0.033 = 0.667
12	3	23	3/30 = 0.100	0.667+0.100 = 0.767
13	2	25	2/30 = 0.067	0.767+0.067 = 0.833
14	1	26	1/30 = 0.033	0.833+0.033 = 0.867
15	1	27	1/30 = 0.033	0.867+0.033 = 0.900
18	2	29	2/30 = 0.067	0.900+0.067 = 0.967
20	1	30	1/30 = 0.033	0.967+0.033 = 1.000

b) What percent of people traveled less than 12 miles?

Answer: The cummulative relative frequency for 11 miles which is just before 12 miles is 0.667 ie, 66.7% of people traveled less than 12 miles.

c) What percent of people traveled over 15 miles?

Answer: The cummulative relative frequency for 15 miles is 0.900 ie, 90% of people traveled 15 miles or less.

∴ 100% − 90% = 10% of people traveled over 15 miles

d) What percent of people traveled between 10 and 15 miles?

Answer: Cummulative relative frequency for 15 miles = 0.900 ie, 90%.
Cummulative relative frequency for 10 miles = 0.633 ie, 63.3%.

∴ 90% − 63.3% = 26.7% of people traveled between 10 and 15 miles.

Part III. Summary Statistics and Boxplots [48 pts = 16 x 3 pts]

1) Below is a list of calories and cholesterol amounts in 4 randomly selected menu items from 4 different fast food companies. [The data is accessible via a .csv file saved under the course shared folder "Course Contents" as "calories_and_cholesterol_amounts.csv"]

Company	*Menu Item*	*Calories*	*Cholesterol (mg)*
McDonald's	Bacon, Egg, & Cheese Biscuit	460	215
McDonald's	Big Mac®	590	85
McDonald's	Filet-O-Fish®	390	30
McDonald's	Cheeseburger	300	40
Burger King	Whopper® Sandwich with Cheese	740	115
Burger King	Cheeseburger	280	45
Burger King	Crispy Chicken Sandwich	670	60
Burger King	Bacon, Egg & Cheese Biscuit	400	170
Wendy's	Baconator®	960	155
Wendy's	Bacon Double Stack®	440	65
Wendy's	Classic Chicken Sandwich	490	75
Wendy's	Sausage, Egg, and Cheese Biscuit	580	285
Chick-fil-A	Chicken Biscuit	460	45
Chick-fil-A	Bacon, Egg, and Cheese Biscuit	420	180
Chick-fil-A	Grilled Chicken Sandwich	390	75
Chick-fil-A	Chicken Nuggets (8 count)	250	80

a) Using R or python, find the following summary statistics for CALORIES for each company.

$$\text{Mean} = \frac{\Sigma \text{values}}{\text{Number of values}}$$

Answer:

	Mean	Min	Q1	Med	Q3	Max	Std. Dev
McDonald's	435	300	345	425	525	590	122.34
Burger King	522.5	280	340	535	705	740	218.23
Wendy's	617.5	440	465	535	770	960	235.57
Chick-fil-A	380	250	355	405	440	460	91.29

Which company has the greatest calorie variability in the distribution?

Answer: Wendy's has the greatest calorie variability, with standard deviation of 234.62 calories.

b) There are many methods for determining outliers. Two methods frequently used are:

Rule #1: An outlier is a value greater than 1.5 × IQR above the third quartile or more than 1.5 × IQR below the first quartile.

Rule #2: An outlier is a value located 2 or more standard deviations above, or below, the mean.

Using rule #1, are there any outliers in the Wendy's distribution? Show your work.

Answer:

IQR = Q3 − Q1

For Wendy's:- Q1 = 465
Q3 = 770
IQR = 305

The minimum & maximum value ie, 440 & 960 both fall within these bounds.
∴ No outlier.

Lower bound for outliers = 465 − 1.5 × 305 = 465 − 457.5 = 7.5

Upper bound for outliers = 770 + 1.5 × 305 = 770 + 457.5 = 1227.5

Using rule #2, are there any outliers in the Wendy's distribution? Show your work.

Answer:

Mean = 617.5

Standard Deviation = 235.57

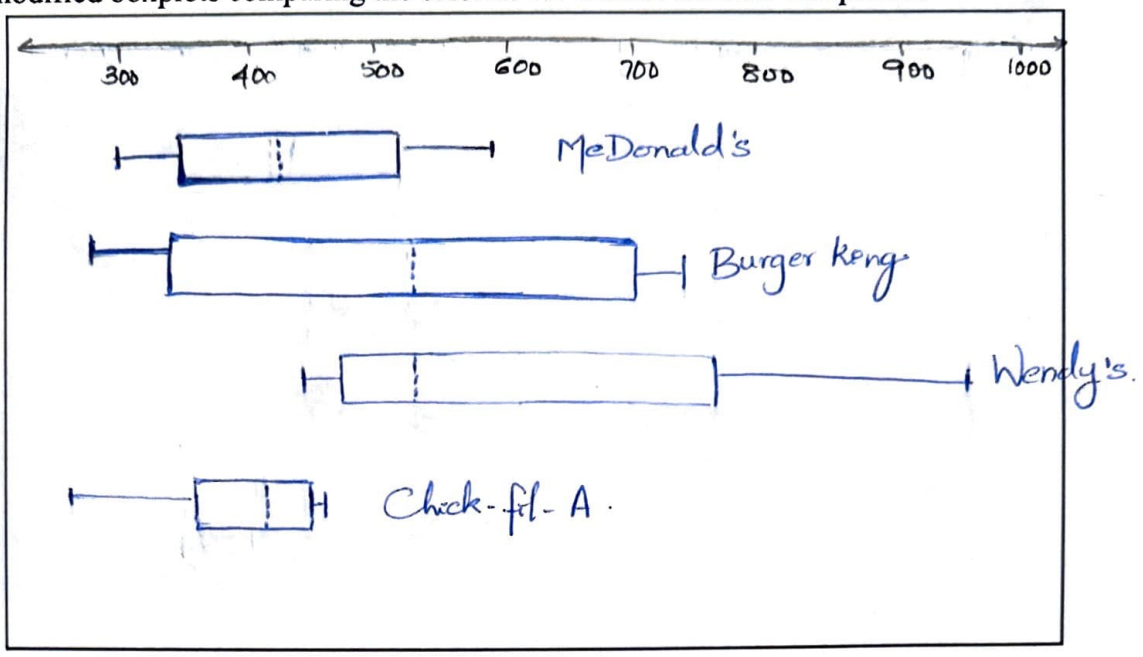
Lower bound = 617.5 − 2 × 235.57 = 146.36

Upper bound = 617.5 + 2 × 235.57 = 1088.64

All Wendy's value fall within 146.36 − 1088.64 range, so there is no outlier.

c) Draw four modified boxplots comparing the calories for each of the four companies.

Answer:



d) In your town, McDonald's and Burger King are on the Northside and Wendy's and Chick-fil-A are on the Southside. Using R or python, find the following summary statistics for CHOLESTEROL for the Northside and Southside fast food restaurants. **(6 pts = 2 x 3 pts)**

	Mean	Min	Q1	Med	Q3	Max	Std. Dev
Northside	95	30	42.5	72.5	142.5	215	67.08
Southside	120	45	70	77.5	167.5	285	81.37

Which region has the greatest cholesterol variability in the distribution?

Answer:

From the table Southside has higher standard deviation (81.37 mg) it has the highest cholesterol variability in the distribution.

e) Using rule #1, are there any outliers in the Southside's distribution? Show your work.

Answer:

For Southside: Q1 = 70 , Q3 = 167.5.

IQR = Q3 - Q1 = 167.5 - 70 = 97.5

Lower bound : 70.0 - 1.5 * 97.5 = -76.25

Upper bound : 167.5 + 1.5 * 97.5 = 313.75

The are no outlider as all the values of Southside fall between -76.25 and 313.75.

Using rule #2, are there any outliers in the Southside's distribution? Show your work.

Answer:

Mean = 120
Standard deviation = 81.37
Lower bound = 120 - 2*81.37 = -42.74
Upper bound = 120.0 + 2*81.37 = 282.74
Standard deviation method identifies 285mg as an outlier as it doesn't fall inside the boundary range.

f) Remove the value of 285mg from the Southside cholesterol's data set. Use R or python to find the following values again:

	Mean	Min	Q1	Med	Q3	Max	Std. Dev
Southside	96.43	45	70	75	117.5	180	48.99 ≈ 49

What values changed the most? What values changed the least?

Answer (Most):

The maximum value changed by 105mg (from 285 to 180)
Q3 changed by 50mg (from 167.5 to 117.5), Standard Deviation changed by ~~32.37mg (81.37 to 49)~~ & ~~mean changed by 23.57mg (120 to 96.43)~~.

Answer (Least):

Minimum there was no change. & there was no change in Q1 value.
Median minimal change 2.5mg (77.5 to 75).

Part IV. Statistical Programming (51 pts) [* Extra Credits] [Instructions in R]

1. **Sequences.** Generate the following sequences using `rep()`, `seq()` and arithmetic:

   (a) $1, 3, 5, 7, \ldots, 21$.

   (b) $1, 10, 100, \ldots, 10^9$.

   (c) $0, 1, 2, 3, 0, \ldots, 3, 0, 1, 2, 3$ [with each entry appearing 6 times]

   (d) $0, 0, 0, 1, 1, 1, 2, \ldots, 4, 4, 4$.

   (e)* $50, 47, 44, \ldots, 14, 11$.

   (f)* $1, 2, 5, 10, 20, 50, 100, \ldots, 5 \times 10^4$.

   Can any of your answers be simplified using recycling?

2. **Arithmetic.** Create a vector containing each of the following sequences:

   (a) $\cos\left(\frac{\pi n}{3}\right)$, for $n = 0, \ldots, 10$.

   (b) $1, 9, 98, 997, \ldots, 999994$.

   (c) $e^n - 3n$, for $n = 0, \ldots, 10$.

   (d)* $3n \mod 7$, for $n = 0, \ldots, 10$.

   Let

   $$S_n = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{2i-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^{n+1}}{2n-1}.$$

   You will recall that $\lim_n S_n = \pi/4$.

   (e) Evaluate $4S_{10}$, $4S_{100}$ and $4S_{1000}$. *[Hint: use the `sum()` function.]*

   (f) Create a vector with entries $S_i - \frac{\pi}{4}$, for $i = 1, \ldots, 1000$. *[Hint: try creating the vector with entries S_i first; the function `cumsum()` may be useful.]*

3. **Subsetting**

   Create a vector `x` of normal random variables as follows:

```
> set.seed(123)
> x <- rnorm(100)
```

   The `set.seed()` fixes the random number generator so that we all obtain the same `x`; changing the argument `123` to something else will give different results. This is useful for replication.

   Give commands to select a vector containing:

   (a) the 25th, 50th and 75th elements;

   (b) the first 25 elements;

   (c) all elements except those from the 31st to the 40th.

   Recall the logical operators `|`, `&` and `!`. Give commands to select:

   (d) all values larger than 1.5 (how many are there?);

   (e) what about the entries that are either > 1.5 or < -1?

Answer: Copy and paste your R or phthon code in the box below (not an image but the text).

```
#1. Sequences
import numpy as np

# (a) 1, 3, 5, 7, ..., 21
a = np.arange(1, 22, 2)
print("(a):", a)

# (b) 1, 10, 100, ..., 10^9
b = 10 ** np.arange(0, 10)
print("(b):", b)

# (c) 0, 1, 2, 3, 0, ..., 3, 0, 1, 2, 3 [with each entry appearing 6 times]
c = np.tile(np.arange(4), 6)
print("(c):", c)

# (d) 0, 0, 0, 1, 1, 1, 2, ..., 4, 4, 4
d = np.repeat(np.arange(5), 3)
print("(d):", d)

# (e) 50, 47, 44, ..., 14, 11
e = np.arange(50, 10, -3)
print("(e):", e)

# (f) 1, 2, 5, 10, 20, 50, 100, ..., 5 x 10^4
f = np.array([1, 2, 5])
while f[-1] < 5e4:
    f = np.append(f, f[-1] * 2)
print("(f):", f)

# Can any of your answers be simplified using recycling?
# Yes, (c) and (d) use recycling with np.tile and np.repeat respectively.
```

```
(a): [ 1  3  5  7  9 11 13 15 17 19 21]
(b): [         1         10        100       1000      10000     100000
    1000000   10000000  100000000 1000000000]
(c): [0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]
(d): [0 0 0 1 1 1 2 2 2 3 3 3 4 4 4]
(e): [50 47 44 41 38 35 32 29 26 23 20 17 14 11]
(f): [    1     2     5    10    20    40    80   160   320   640  1280  2560
  5120 10240 20480 40960 81920]
```

Answer: Copy and paste your R or phthon code in the box below (not an image but the text).

```
#2. Arithmetic section
import numpy as np

# (a) cos(nπ/3), for n = 0, ..., 10
a = np.cos(np.arange(11) * np.pi / 3)
print("(a):", a)

# (b) 1, 9, 98, 997, ..., 999994
b = np.array([int('9' * i + '1' * (6-i)) for i in range(6)])
print("(b):", b)

# (c) e^n - 3n, for n = 0, ..., 10
c = np.exp(np.arange(11)) - 3 * np.arange(11)
print("(c):", c)

# (d) 3n mod 7, for n = 0, ..., 10
d = (3 * np.arange(11)) % 7
print("(d):", d)

# (e) Evaluate 4S_10, 4S_100 and 4S_1000
def S(n):
    return np.sum((-1)**(np.arange(1, n+1) + 1) / (2 * np.arange(1, n+1) - 1))

print("(e):")
print("4S_10 =", 4 * S(10))
print("4S_100 =", 4 * S(100))
print("4S_1000 =", 4 * S(1000))

# (f) Create a vector with entries S_i - π/4, for i = 1, ..., 1000
f = np.cumsum((-1)**(np.arange(1, 1001) + 1) / (2 * np.arange(1, 1001) - 1)) - np.pi/4
np.set_printoptions(threshold=np.inf)
print("(f):", f)
```

```
(a): [ 1.   0.5 -0.5 -1.  -0.5  0.5  1.   0.5 -0.5 -1.  -0.5]
(b): [111111 911111 991111 999111 999911 999991]
(c): [ 1.00000000e+00 -2.81718172e-01  1.38905610e+00  1.10855369e+01
  4.25981500e+01  1.33413159e+02  3.85428793e+02  1.07563316e+03
  2.95695799e+03  8.07608393e+03  2.19964658e+04]
(d): [0 3 6 2 5 1 4 0 3 6 2]
(e):
4S_10 = 3.0418396189294024
4S_100 = 3.131592903558552
4S_1000 = 3.140592653839792
(f): [ 0.21460184 -0.1187315   0.0812685  -0.06158864  0.04952247 -0.04138662
  0.03553646 -0.03113021  0.02769332 -0.02493826  0.02268079 -0.02079747
  0.01920253 -0.01783451  0.01664825 -0.01560981  0.01469322 -0.01387821
  0.01314881 -0.01249221  0.01189803 -0.01135778  0.01086444 -0.01041216
  0.00999601 -0.00961184  0.00925609 -0.00892573  0.00861813 -0.00833102
  0.00806242 -0.00781059  0.00757402 -0.00735135  0.0071414  -0.00694311
  0.00675552 -0.00657781  0.0064092  -0.00624902  0.00609665 -0.00595154
  0.00581317 -0.00568108  0.00555487 -0.00543414  0.00531855 -0.00520777
```

```
 0.00027933 -0.00027902  0.00027871 -0.0002784   0.00027809 -0.00027778
 0.00027747 -0.00027716  0.00027685 -0.00027655  0.00027624 -0.00027594
 0.00027563 -0.00027533  0.00027503 -0.00027473  0.00027442 -0.00027412
 0.00027382 -0.00027352  0.00027322 -0.00027293  0.00027263 -0.00027233
 0.00027203 -0.00027174  0.00027144 -0.00027115  0.00027086 -0.00027056
 0.00027027 -0.00026998  0.00026969 -0.0002694   0.00026911 -0.00026882
 0.00026853 -0.00026824  0.00026795 -0.00026767  0.00026738 -0.00026709
 0.00026681 -0.00026652  0.00026624 -0.00026596  0.00026567 -0.00026539
 0.00026511 -0.00026483  0.00026455 -0.00026427  0.00026399 -0.00026371
 0.00026344 -0.00026316  0.00026288 -0.0002626   0.00026233 -0.00026205
 0.00026178 -0.00026151  0.00026123 -0.00026096  0.00026069 -0.00026042
 0.00026015 -0.00025988  0.00025961 -0.00025934  0.00025907 -0.0002588
 0.00025853 -0.00025826  0.000258   -0.00025773  0.00025747 -0.0002572
 0.00025694 -0.00025667  0.00025641 -0.00025615  0.00025589 -0.00025562
 0.00025536 -0.0002551   0.00025484 -0.00025458  0.00025432 -0.00025406
 0.00025381 -0.00025355  0.00025329 -0.00025304  0.00025278 -0.00025253
 0.00025227 -0.00025202  0.00025176 -0.00025151  0.00025126 -0.000251
 0.00025075 -0.0002505   0.00025025 -0.00025   ]
```

Answer: Copy and paste your R or phthon code in the box below (not an image but the text).

```
#3. Subsetting section
import numpy as np

# Set seed for reproducibility
np.random.seed(123)

# Create vector of normal random variables
x = np.random.normal(size=100)

# (a) Select the 25th, 50th and 75th elements
a = x[[24, 49, 74]]  # Python uses 0-based indexing
print("(a):", a)

# (b) Select the first 25 elements
b = x[:25]
print("(b): First 25 elements of result:")
print("(b):", b)

# (c) Select all elements except those from the 31st to the 40th
c = np.concatenate((x[:30], x[40:]))
print("(c): Length of result:", len(c))
print("(c):", c)

# (d) Select all values larger than 1.5
d = x[x > 1.5]
print("(d): Number of values larger than 1.5:", len(d))
print("(d) Values:", d)

# (e) Select entries that are either > 1.5 or < -1
e = x[(x > 1.5) | (x < -1)]
print("(e): Number of entries > 1.5 or < -1:", len(e))
print("(e) Values:", e)xt
```

```
(a): [-1.25388067  2.23814334 -2.12310035]
(b): First 25 elements of result:
(b): [-1.0856306   0.99734545  0.2829785  -1.50629471 -0.57860025  1.65143654
 -2.42667924 -0.42891263  1.26593626 -0.8667404  -0.67888615 -0.09470897
  1.49138963 -0.638902   -0.44398196 -0.43435128  2.20593008  2.18678609
  1.0040539   0.3861864   0.73736858  1.49073203 -0.93583387  1.17582904
 -1.25388067]
(c): Length of result: 90
(c): [-1.0856306   0.99734545  0.2829785  -1.50629471 -0.57860025  1.65143654
 -2.42667924 -0.42891263  1.26593626 -0.8667404  -0.67888615 -0.09470897
  1.49138963 -0.638902   -0.44398196 -0.43435128  2.20593008  2.18678609
  1.0040539   0.3861864   0.73736858  1.49073203 -0.93583387  1.17582904
 -1.25388067 -0.6377515   0.9071052  -1.4286807  -0.14006872 -0.8617549
 -0.80536652 -1.72766949 -0.39089979  0.57380586  0.33858905 -0.01183049
  2.39236527  0.41291216  0.97873601  2.23814334 -1.29408532 -1.03878821
  1.74371223 -0.79806274  0.02968323  1.06931597  0.89070639  1.75488618
  1.49564414  1.06939267 -0.77270871  0.79486267  0.31427199 -1.32626546
  1.41729905  0.80723653  0.04549008 -0.23309206 -1.19830114  0.19952407
  0.46843912 -0.83115498  1.16220405 -1.09720305 -2.12310035  1.03972709
 -0.40336604 -0.12602959 -0.83751672 -1.60596276  1.25523737 -0.68886898
  1.66095249  0.80730819 -0.31475815 -1.0859024  -0.73246199 -1.21252313
  2.08711336  0.16444123  1.15020554 -1.26735205  0.18103513  1.17786194
 -0.33501076  1.03111446 -1.08456791 -1.36347154  0.37940061 -0.37917643]
(d): Number of values larger than 1.5: 9
(d) Values: [1.65143654 2.20593008 2.18678609 2.39236527 2.23814334 1.74371223
 1.75488618 1.66095249 2.08711336]
(e): Number of entries > 1.5 or < -1: 29
(e) Values: [-1.0856306  -1.50629471  1.65143654 -2.42667924  2.20593008  2.18678609
 -1.25388067 -1.4286807  -2.79858911 -1.7715331  -1.72766949  2.39236527
  2.23814334 -1.29408532 -1.03878821  1.74371223  1.75488618 -1.32626546
 -1.19830114 -1.09720305 -2.12310035 -1.60596276  1.66095249 -1.0859024
 -1.21252313  2.08711336 -1.26735205 -1.08456791 -1.36347154]
```

Appendix: Example Question and Answer for R or python programming questions:

Calculate the sum $\sum_{j=0}^{n} r^j$, where r has been assigned the value 1.08, and compare with $(1 - r^{n+1})/(1 - r)$, for $n = 10, 20, 30, 40$.

Answer: Copy and paste your R or python code in the box below (not an image but the text).

```
r <- 1.08
n <- c(10, 20, 30, 40)
sum1 <- c()
for(i in n){
  x <- 0:i
  sum1 <- c(sum1, sum(r^x))
}
sum1   # This gives the calculated sums for n = 10, 20, 30, 40.

sum2 <- (1 - r^(n + 1)) / (1 - r)
sum2

sum2 - sum1   # The formula works.
```

Screenshot of your R or python console outputs and paste/insert the image in the box below

```
> r <- 1.08
> n <- c(10, 20, 30, 40)
> sum1 <- c()
> for(i in n){
+   x <- 0:i
+   sum1 <- c(sum1, sum(r^x))
+ }
> sum1   # This gives the calculated sums for n = 10, 20, 30, 40.
[1]  16.64549  50.42292 123.34587 280.78104
> sum2 <- (1 - r^(n + 1)) / (1 - r)
> sum2
[1]  16.64549  50.42292 123.34587 280.78104
> sum2 - sum1   # The formula works.
[1] 0 0 0 0
```

THE END