Problem Solving with Statistics

This report is going to analyse the questions with data based on statistical method. By clarifying questions, identifying solution models, and processing data, the data-driven advices will be given.

Question 1. (a)

To explore whether the exposure to radioactivity is connected with the occurrence of cancer or not, we can first make a close examination of what the data look like. According to the Fig.1, by visual checking, we can see that the relationship between REI and Mortality Rate might have some pattern. When REI increases, the Mortality Rate also increases. Based on the analysis of description statistics and boxplot (Fig.2 to Fig.4), it is found that there is no outlier in the dataset. In addition, the normality test for two variables also shows that they follow normal distribution (Fig.5 to Fig.6). As REI and mortality rate are continuous variables, normal distributed, linearity, homoscedasticity, and without outlier, Pearson's correlation coefficient is applied, which is 0.9175 (Fig.8). It indicates that REI and Mortality Rate is highly positively correlated. Therefore, we start to find the pattern from linear regression model to see whether the model is well fitted and can be further used for prediction.

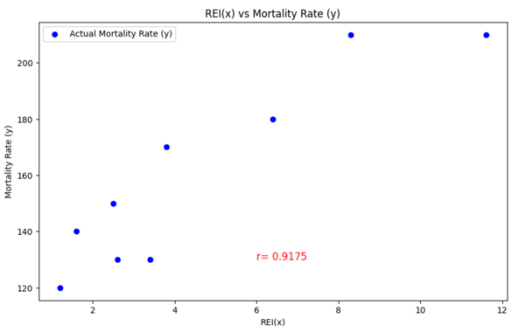


Fig.1 Scatter Plot for REI and Mortality Rate

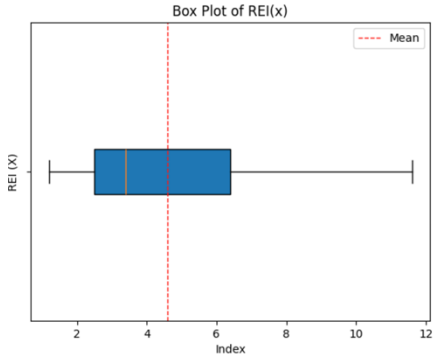


Fig.3 Box plot of REI

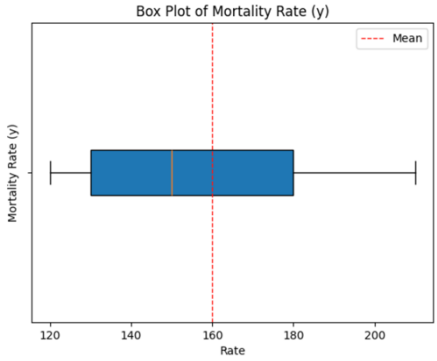


Fig.4 Box plot of Mortality Rate

REI (x)		Mortality Rate (y)	
Mean	4.6	Mean	160
Standard Error	1.160579357	Standard Error	11.426091
Median	3.4	Median	150
Mode	#N/A	Mode	210
Standard Deviation	3.481738072	Standard Deviation	34.278273
Sample Variance	12.1225	Sample Variance	1175
Kurtosis	0.626005189	Kurtosis	-1.269352648
Skewness	1.194408877	Skewness	0.52671006
Range	10.4	Range	90
Minimum	1.2	**Minimum**	120
Maximum	11.6	**Maximum**	210
Sum	41.4	Sum	1440
Q1	2.5	Q1	130
Q2	3.4	Q2	150
Q3	6.4	Q3	180
IQR	3.9	IQR	50
Lower Bound: Outlier	-3.35	**Lower Bound: Outlier**	55
Upper bound: Outlier	12.25	**Upper bound: Outlier**	255
Count	9	Count	9

Fig.2 Descriptive Statistics for REI and Mortality Rate

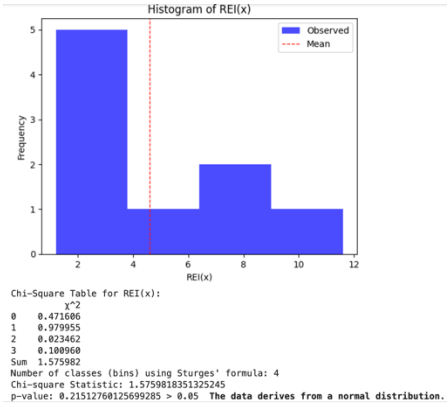


Fig.5 Histogram of REI

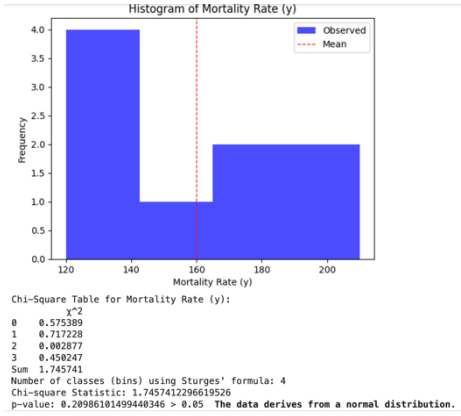


Fig.6 Histogram of Mortality Rate

To begin with, we assume that the relationship between REI and Mortality Rate can represented as $\hat{y} = \beta_0 + \beta_1 x + e_i$, which x represents REI, $\hat{y}$ represents estimated mortality rate, and e_i represents the error of observed points and predicted points. In the model, β_0 is the intercept and β_1 is the slope of the line. Additionally, we assume that each variable (x) is independent and has linearity with dependent

variable (y). And, the slope of the line should follow normal distribution. Also, the data should have homogeneity of variance. By implementing Ordinary Least Squares (OLS), we get the regression equation is $y = 9.03x + 118.45$ as shown in Fig.7. To evaluate the regression coefficient is significant, t-test is chosen due to small sample size and unknown variance of data. We can take a t-test for slope and set the hypothesis as $H_0: \beta_0 = 0,\ H_1: \beta_0 \neq 0$. According to the result (Fig.9), the p-value of the slope is about 0.0005, which is less than 0.05. As a result, we can reject the null hypothesis $H_0: \beta_0 = 0$ at 5% significance level. It implies that the independent variable has a significant impact on the dependent variable at 95% confidence. In addition, by applying t-test for the intercept, the hypothesis is set as $H_0: \beta_1 = 0,\ H_1: \beta_1 \neq 0$. Based on the Fig.9, the p-value of the intercept is smaller than 0.05 as well. This finding shows that we can reject the null hypothesis $H_0: \beta_1 = 0$ for the intercept at 95% confidence. It reflects that when REI (x) is 0, the mortality rate (y) is significantly not equal to 0. As a result, the probable limit of the regression coefficient with 95% confidence is [5.5328, 12.5328] for the slope and [98.6701, 128.2282] for the intercept. We can see that both coefficient of the regression lies in the interval. Therefore, we can make the conclusion that the data supports the claim that exposure to radioactivity is connected to cancer occurrence with linear regression.

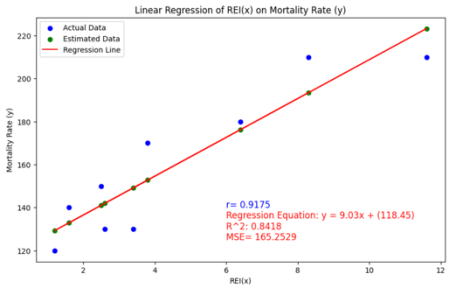


Fig.7 Linear Regression of REI and Mortality Rate

Regression Statistics	
Multiple R	**0.9175**
R Square	0.8418
Adjusted R Square	0.8192
Standard Error	14.5763
Observations	9

Fig.8 Regression Statistics of OLS

	Coefficient	Std err	t statistics	P-value	Lower Bound 95%	Upper Bound 95%
Intercept	118.4492	8.3646	14.1608	**0.0000021**	**98.6701**	**138.2282**
REI (x)	9.0328	1.4801	6.1026	**0.0005**	**5.5328**	**12.5328**

Fig.9 Coefficients of Linear Regression

Question 1. (b)

Based on the linear regression model of $y = 9.03x + 118.45$, the level of cancer mortality would be expected for an REI (x) value of 5.0 is [127.25, 199.97] at a 95% prediction level (Fig.10).

	REI(x)	t_value	St_Error_of_Prediction	Error	Lower_Bound	Point_Prediction	Upper_Bound
0	5	2.3646	15.3762	36.3588	127.2543	163.6131	199.9719

Fig.10 Estimated value based on linear regression

Question 1. (c)

To determine whether the plotted regression line would pass through the origin ($y = 0\ \ when\ x = 0$), we evaluate whether the intercept of the regression line is 0. Based on the linear regression model of $y = 9.03x + 118.45$, the result of estimated mortality rate (y) is 118.45 rather than zero when REI equals to zero ($x = 0$). Thus, the regression line does not pass through the origin.

Question 1. (d)

To test the model, it is integral to check the predictive power, significance of model and its assumptions. Firstly, we see the predictive power by calculating coefficient of determination (R^2), which is 0.8418 (Fig. 8). This number reflects that the goodness of fit is about 84%. Secondly, the level of variability within a regression should be inspected. Through ANOVA table (Fig.11), we compare the regression mean of square ($MSReg$) and residual mean of square ($MSErr$) to calculate the F statistics, which value are 7912.72, 212.47, and 37.24 respectively. Compared to the variability of the linear regression with 95% confidence, the p-value of F statistics, which is 0.0005, is less than significance level of 0.05. This evidence supports that the variability of linear regression model is statistically significant.

ANOVA

	Degree of freedom	SS	MS	F	P-value
Regression	1	7912.72427	**7912.7243**	**37.2420**	**0.0005**
Error	7	1487.27573	**212.4680**		
Sum	8	9400			

Fig.11 ANOVA Table

Thirdly, checking the assumption of normality of residual is necessary for justifying the validity of linear regression model. Therefore, we conduct the normality test on residual by chi-square. Through the process of calculating residuals for each sample, standardising residuals, and finding the lower bound and upper bound of the outliers, we can make sure that there is no outlier and can be further explore the data (Fig.12). To check the normality of standerdised residuals, we also calculate the class of histogram and its frequency (Fig.13) to plot the histogram (Fig.14).

	REI(x)	Mortality Rate (y)	Fitted_value	Residual	Standardized_Res
count	9.0000	9.0000	9.0000	9.0000	9.0000
mean	4.6000	160.0000	160.0000	0.0000	0.0000
std	3.4817	34.2783	31.4498	13.6349	1.0607
min	1.2000	120.0000	129.2885	-19.1607	-1.4905
25%	2.5000	130.0000	141.0311	-11.9344	-0.9284
50%	3.4000	150.0000	149.1607	3.7410	0.2910
75%	6.4000	180.0000	176.2590	8.9689	0.6977
max	11.6000	210.0000	223.2295	17.2262	1.3400

Box Plot Values:
Q1: -0.9283820606699202, Q3: 0.6976033774318494, IQR: 1.6260725835017786
Lower Bound for Outliers: -3.367491081322597
Upper Bound for Outliers: 3.136799252684517

Fig.12 Summary of REI(x), Mortality Rate(y), Fitted Value, Residual, and Standardized residual

Rounded Number of classes: 4
class Range: 0.7076367566916284
Classes:

	Standard_Residuals	Frequency
0	-0.7829	3
1	-0.0752	1
2	0.6324	2
3	1.3400	3

Fig.13 Plotting Histogram for standerdised residual

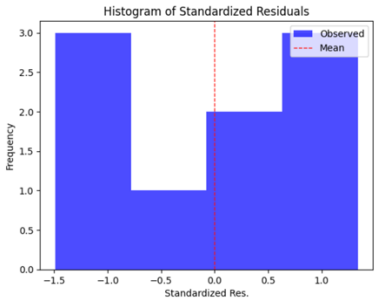


Fig.14 Histogram of standerdised residual

Then, the p-value of chi-square statistics (Fig.15) is computed, which is around 0.1483. Because the value is greater than 0.05 ($\alpha = 5\%$), it implies that the residual of the model is normally distributed at 95% confidence level.

	CDF	Bin Frequency	Expected Value
0	0.2302	0.2302	2.0720
1	0.4717	0.2415	2.1735
2	0.7245	0.2528	2.2749
3	0.8968	0.1723	1.5506
4	Sum =	0.8968	8.0710

Chi-Square Table:

	X^2
0	0.4156
1	0.6336
2	0.0332
3	1.3548
Sum	2.4372

Chi-square Statistic: 2.0897489470429065,
p-value: 0.14829043742773648

Fig.15 CDF, Expected value, and Chi-square Table

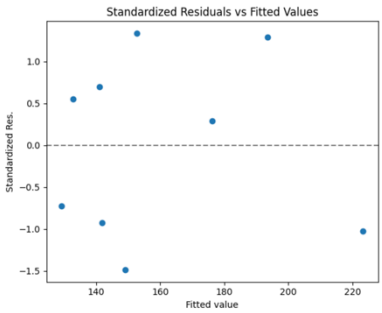


Fig.16 Standardized Residuals vs Fitted Values

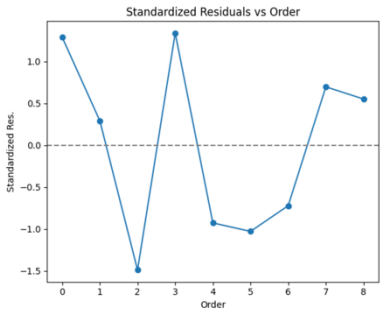


Fig.17 Standardized Residuals vs Order

In addition, to confirm the homoscedasticity assumption, by visual checking, we can see that there is no specific pattern in Fig.16. It suggests that the residuals have a constant variance. But in the Fig.17, it indicates that the standerdised residuals have a periodical trend. It is recommended that the model need further examine to make sure residuals are independent to each other and follow the assumption of linear regression.

Question 2. (i) (a)

The problem involves Poisson Distribution because the number of defects is observed in a fixed interval for measuring space and time. In addition, the event of observing defect cannot happen at the same time. The result of each test shall also be independent of other tests. According to the question, the number of trials is unknow and very large. Moreover, the events occurring with a constant probability over the time period is average of 2.6 defects per metre length. Hence, we can calculate the probability of two defects in a single metre by following the formula: $P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$. When the expected number of event (λ) is 2.6, the number of defects (x) is 2, $P(X = 2) = \frac{2.6^2 e^{-2.6}}{2!} = \frac{6.7 \times 0.0743}{2} \approx 0.251$. Thus, the probability of exactly two defects in a single metre is approximately 25.1%.

Question 2. (i) (b)

If there are more than 8 defects in a 5-metre length, we need to compute $P(X > 8)$. To make the calculation easy, we can also compute $1 - P(X \leq 8)$. In this case, the average of defects per 5-metre is 13 ($\lambda = 2.6 * 5$). Then we calculate $P(X \leq 8) = P(X = 0) + P(X = 1) + \cdots + P(X = 8) \approx 9.98\%$. The result of $1 - P(X \leq 8) = P(X > 8)$ is 0.9002. Therefore, the probability of more than 8 defects in a 5-metre length electrical conductor is about 90%.

Question 2. (i) (c)

For a 10-metre length, the average number of defects is 26 ($\lambda = 2.6 * 10$). We should calculate the probability of fewer than 20 defects by $P(X < 20)$. The value of $P(X < 20)$ is close to the value of $P(X \leq 19)$. Therefore, we can get the answer of $P(X < 20)$ is 0.0968. Thus, the probability of the electrical conductor having fewer than twenty defects in a 10-metre length is 9.68%.

Question 2. (ii) (a)

To see whether the population mean is on target, we can start from descriptive statistics and boxplot for sample data to see how the data look like. According to the Fig.18, we can get that the mean of viscosity for sample is 33.65 second and the standard deviation is 0.4392. The skewness and kurtosis are close to the theoretical values of a normal distribution 0 and can be further validated using hypothesis test. From the boxplot (Fig.19), it shows that there is no outlier in the dataset, so no need additional process for the data.

Viscosity	
Mean	**33.65**
Standard Error	0.113410338
Median	33.58
Mode	#N/A
Standard Deviation	**0.43923635**
Sample Variance	0.192928571
Kurtosis	**0.192768894**
Skewness	**0.459765495**
Range	1.62
Minimum	33
Maximum	34.62
Sum	504.75
Count	15

Fig.18 Descriptive Statistics for Viscosity

```
Box Plot Values:
Q1: 33.365, Q3: 33.92 , IQR: 0.5550
Lower Bound for Outliers: 32.5325
Upper Bound for Outliers: 34.7525
```

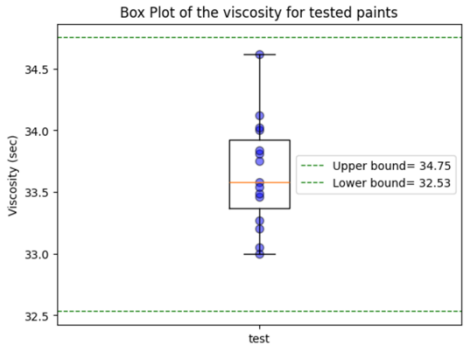


Fig.19 Box plot of Viscosity

```
Rounded Number of classes: 4
class Range: 0.40499999999999936
Classes:
   Length  Frequency
0  33.405          4
1  33.810          5
2  34.215          5
3  34.620          1
```

Fig.20 Class and Frequency for Viscosity

Because the sample size is small (n =15) and we would like to explore whether the average of the viscosity reach the standard, t-test is chosen to perform the hypothesis. The null hypothesis is set as H_0: $\mu = 34$, and the alternative hypothesis is H_1: $\mu \neq 34$. Before conducting the t-test, the prerequisite is that the distribution of the data should be normal. Thus, the normality test is conducted by chi-square. We assume that the null hypothesis (H_0) is that the data derives from a normal distribution, and the alternative hypothesis (H_1) is that the data do not follow normal distribution. By plotting histogram and calculating p-value of chi-square (Fig.20 to 23), this analysis shows that the data derives from a normal distribution because the p-value is larger than 5% significance levels. Therefore, we do not have significance evidence to reject the null hypothesis (H_0).

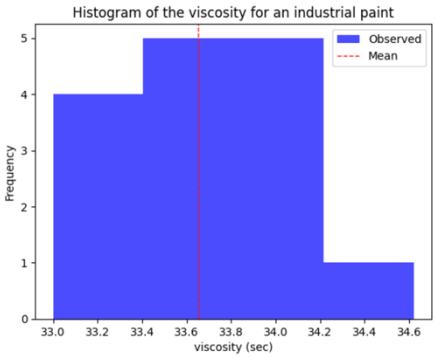


Fig.21 Histogram for Viscosity

```
        CDF  Bin Frequency  Expected Value
0  0.288495       0.288495        4.327428
1  0.642171       0.353676        5.305141
2  0.900835       0.258664        3.879954
3  0.986391       0.085556        1.283342
4      Sum =      0.986391       15.000000

Chi-Square Table:
          χ^2
0    0.024774
1    0.017551
2    0.323329
3    0.062558
Sum  0.428212
Number of classes (bins) using Sturges' formula: 4
Chi-square Statistic: 0.4282121439347843,
p-value: 0.5171334253101 48
```

Fig.22 Chi-square table of Viscosity

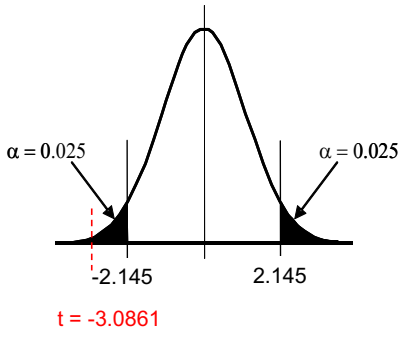


Fig.23 t-distribution of Viscosity

After checking the normality, we can continue validate that whether the population mean of the viscosity is on target or not with t-test. The t-statistics is computed as -3.0861 from the formula $t_{calc} = \frac{\bar{x} - \mu_0}{\hat{\sigma}/\sqrt{n}}$, when $\bar{x} = 33.65,\ \mu_0 = 34,\ \hat{\sigma} = 0.4392,\ and\ n = 15$. The critical value of two-tailed t-distribution with degree of freedom 14 is 2.1448 and - 2.1448 ($t_{0.025,14} = 2.1448$). Compared to critical value (Fig.23), t-statistics is in the rejection region. The p-value of the statistics is 0.0081, which is also less than 0.05. This result shows that we have 95% confidence to reject the null hypothesis (H_0: $\mu = 34$). This finding proves that we cannot accept the claim that the population mean viscosity of the industrial paint is 34 seconds at 5% significance level.

Question 2. (ii) (b)

At 95% confidence level, the interval of the population mean is [33.41, 33.89]. It is calculated by the formula, $\bar{x} \pm t\alpha_{/2,\nu} \frac{\hat{\sigma}}{\sqrt{n}}$. We can see that the population mean interval is not within the target interval, which is [33, 35]. Therefore, we can make a conclusion that the viscosity of an industrial paint batch would not completely fit the specification.

Question 2. (ii) (c)

In the long run, the distribution will tend to be standard normal distribution. Thus, we need to calculate the possibility of mean value less than 33 and larger than 35, which is $P(X < 33 \; or \; X > 35)$. According to Fig.24, we can draw the conclusion that the possibility of batches which likely to be outside the specification in the long run is about 1- 92.95%, which is 7.05%.

x	z-stat	P(Z <=z)
x = 33	-1.4798	0.0694578298
x = 35	3.0735	0.9989422387
P(33 < X < 35)		**0.9294844088**

Fig.24 z-statistics and p-value table of x

Question 3. (i)

To demonstrate whether there is a significant response time difference between the two operating systems disc, we start from descriptive analysis for two data set. According to the Fig.25, we can see that the mean of response time is about 65.69 microseconds for Disc 1, and 53.33 for Disc 2. The standard deviations are 23.44 and 13.47. Their coefficients of variation are 2.80 and 3.96. It suggests that the dispersity of two discs seems similar. However, it needs to be validated whether the difference is statistically significant. In addition, the skewness of Disc # 1 is close to 0, which means that the distribution is near to normal distribution. Compared to the skewness of Disc #1, the shape of distribution of Disc #2 is slightly right-skewed. Furthermore, the kurtosis of both data is less than 0. It means that the shapes of the distribution are flat.

Disc #1		Disc #2	
Mean	**65.6923077**	**Mean**	**53.3333333**
Standard Error	6.50185833	Standard Error	3.47873358
Median	65	Median	49
Mode	72	Mode	71
Standard Deviation	**23.4427836**	**Standard Deviation**	**13.4730772**
Sample Variance	549.564103	Sample Variance	181.52381
Kurtosis	**-0.541368**	**Kurtosis**	**-1.3938842**
Skewness	**0.06063408**	**Skewness**	**0.33790553**
Range	78	Range	40
Minimum	26	Minimum	36
Maximum	104	Maximum	76
Sum	854	Sum	800
Count	13	Count	15
Coefficient of Variation	**2.80224008**	**Coefficient of Variation**	**3.95851167**

Fig.25 Summary of Disc#1 and Disc #2

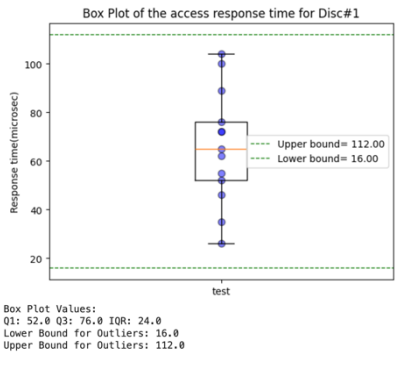


Fig.26 Box plot of Disc#1

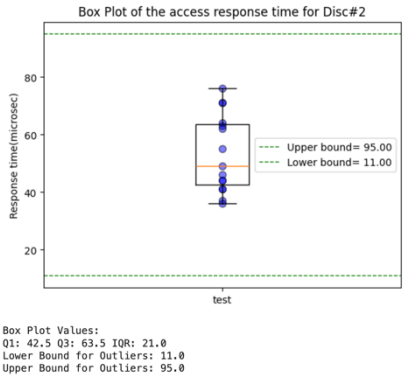


Fig.27 Box plot of Disc#2

Additionally, the boxplots of two discs are provided as above (Fig.26 & Fig.27). The figures show that there is no outlier in Disc#1 and Disc#2 data. Therefore, we can further proceed the test and no need to deal with the outlier.

Nevertheless, before we investigate the variance, the normality test is applied to check both data follow normal distribution or not. The class, frequency and histograms are calculated and plotted for its distribution as Fig.28 to Fig.30.

```
Disc #1
Rounded Number of classes: 4
class Range: 19.5
Classes:
   Length  Frequency
0    45.5          2
1    65.0          4
2    84.5          4
3   104.0          3
```

Fig.28 Class and Freq. of Disc#1

```
Disc # 2
Rounded Number of classes: 4
class Range: 10.0
Classes:
   Length  Frequency
0    46.0          6
1    56.0          3
2    66.0          3
3    76.0          3
```

Fig.29 Class and Freq. of Disc #2

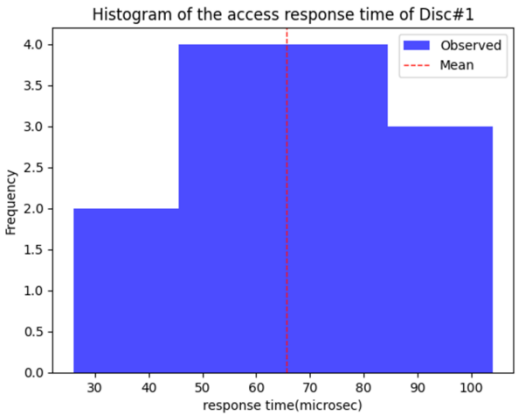


Fig.30 Histogram of the access response time. of Disc#1

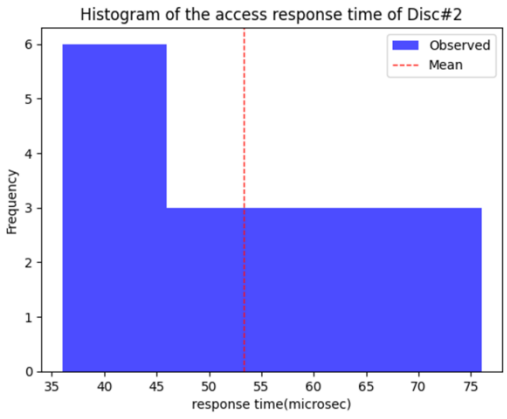


Fig.31 Histogram of the access response time. of Disc#1

In addition, by computing the cumulative distribution function, we can get the bin frequency for each class and then find the expected value for calculating the chi-square. According to the chi-square table, it is observed that the p-value of Disc # 1 is 0.4947 (Fig.32). And the p-value of Disc # 2 is 0.2035 (Fig.33). Both of the p-value are larger than 0.05, which suggests that it do not reject the normal distribution null hypothesis. As a result, we can make sure that the distribution of Disc # 1 and Disc # 2 are normal.

	CDF	Bin Frequency	Expected Value
0	0.194524	0.194524	2.528815
1	0.48822	0.293696	3.818048
2	0.788805	0.300584	3.907598
3	0.94888	0.160076	2.080984
4	Sum =	0.948880	13.000000

```
Chi-Square Table of Disc#1:
          χ^2
0    0.110584
1    0.008671
2    0.002185
3    0.405861
Sum  0.527301
Number of classes (bins) using Sturges' formula: 4
Chi-square Statistic: 0.5273010654983478,
p-value: 0.4946601437779222
```

Fig.32 CDF, Chi-square table of the access response time of Disc#1

	CDF	Bin Frequency	Expected Value
0	0.293119	0.293119	4.396787
1	0.578448	0.285329	4.279939
2	0.826429	0.247980	3.719707
3	0.953751	0.127322	1.909836
4	Sum =	0.953751	15.000000

```
Chi-Square Table of Disc#2:
          χ^2
0    0.584584
1    0.382772
2    0.139252
3    0.622282
Sum  1.728892
Number of classes (bins) using Sturges' formula: 4
Chi-square Statistic: 1.7288915127121625,
p-value: 0.2035318397851863
```

Fig.33 CDF, Chi-square table of the access response time of Disc#2

Next, due to different operating system, we cannot perform it as paired-sample. Hence, the significantly difference of variance for two sample is matter. By using F-test, we can set the hypothesis as $H_0: \sigma_1^2 = \sigma_2^2$, $H_1: \sigma_1^2 \neq \sigma_2^2$. According to the result, the p-value of two-tailed F-test (Fig.34) is one-tailed P(F<=f) multiply by 2, which is 0.0515. Consequently, the null hypothesis, $H_0: \sigma_1^2 = \sigma_2^2$ cannot be rejected. Based on the hypothesis test, we can assume that the variance of Disc # 1 and Disc # 2 has no significantly difference at 95% confidence level.

F-test	Disc #1	Disc #2
Mean	65.69230769	53.33333333
Variance	549.5641026	181.5238095
Observations	13	15
df	12	14
F	3.027504238	P(F<=f) two-tail
P(F<=f) one-tail	0.025743804	*2 = **0.05149**
F Critical one-tail	2.534243253	

Fig.34 F-test: Testing the Variances of Two Normal Populations

	Disc #1	Disc #2
Mean	65.6923077	53.3333333
Variance	549.564103	181.52381
Observations	13	15
Pooled Variance	351.38856	
Hypothesized Mean Difference	0	
df	26	
t Stat	1.73991051	
P(T<=t) one-tail	0.04685057	
t Critical one-tail	1.70561792	
P(T<=t) two-tail	**0.09370114**	
t Critical two-tail	2.05552944	

Fig.35 t-test: Testing the Difference Between Two Population Mean (Assuming Equal Variances)

Subsequently, to evaluate the mean of response time for two operating systems, two sample t-test assuming equal variances is selected due to small sample size and normal distribution. The two-tail hypothesis test is set as $H_0: \mu_1 = \mu_2$, $H_1: \mu_1 \neq \mu_2$. Based on the Fig.35, the p-value of the same mean response time is 0.0937, which is greater than 0.05. The data suggests that we do not reject the null hypothesis ($H_0: \mu_1 = \mu_2$). In conclusion, it can demonstrate that the response time has no significant difference between the two discs in two operating systems.

Question 3. (ii) (a)

To analyse the relationship between social media engagement and e-commerce sales for an online retail company, visual checking for the data is essential. Based on the Fig.36, the graph shows that the data might have linear relationship, homoscedasity, and monotonicity.

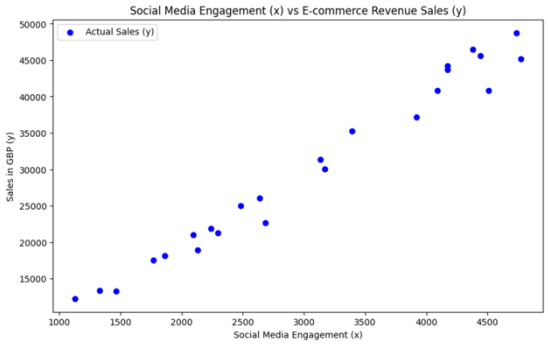


Fig.36 Scatter plot for Social Media Engagement (x) vs E-commerce Revenue Sales (y)

Social Media Engagement		E-commerce Sales (GBP)	
Mean	3041.916667	Mean	30043.025
Standard Error	240.6300224	Standard Error	2487.514074
Median	2910	Median	28030.23
Mode	#N/A	Mode	#N/A
Standard Deviation	1178.841543	Standard Deviation	12186.28042
Sample Variance	1389667.384	Sample Variance	148505430.4
Kurtosis	-1.417376716	Kurtosis	-1.511580197
Skewness	0.018278929	Skewness	0.110812155
Range	3642	Range	36511.45
Minimum	**1130**	**Minimum**	**12248.6**
Maximum	**4772**	**Maximum**	**48760.05**
Sum	73006	Sum	721032.6
Q1	2121.25	Q1	20514.385
Q2	2910	Q2	28030.23
Q3	4171.75	Q3	41584.205
IQR	2050.5	IQR	21069.82
Lower Bound: Outlier	**-954.5**	**Lower Bound: Outlier**	**-11090.345**
Upper bound: Outlier	**7247.5**	**Upper bound: Outlier**	**73188.935**
Count	24	Count	24

Fig.37 Summary of Social Media Engagement and E-commerce Revenue Sales

Therefore, to determine the type of correlation for the analysis, the assumptions of Pearson's and Spearman's correlation coefficient method need to be checked. When applying Pearson's method, the variables should be continuous, have linear relationship. Additionally, the variables should be normal distributed. Besides, the Spearman's method is usually implemented to mitigate the influence of outlier for at least ordinal data or when the dataset with outlier. Thus, to examine which approach is appropriate for the data, the type of variable is also important. As the social media engagement and e-commerce sales are both continuous variables, the Pearson's and Spearman's correlation coefficient are both suitable. From Fig.37 to Fig.40, the descriptive analysis and the boxplot both illustrates that there is no outlier in the sample data. However, the normality of variables and residual should be confirmed when adopting the Pearson's correlation coefficient.

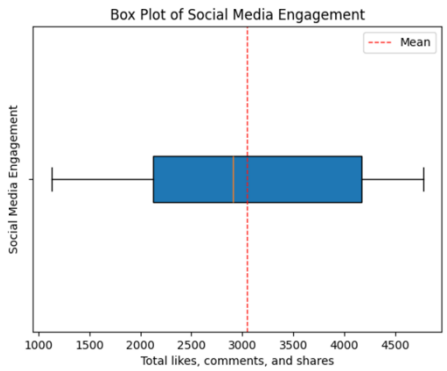


Fig.38 Box plot of Social Media Engagement

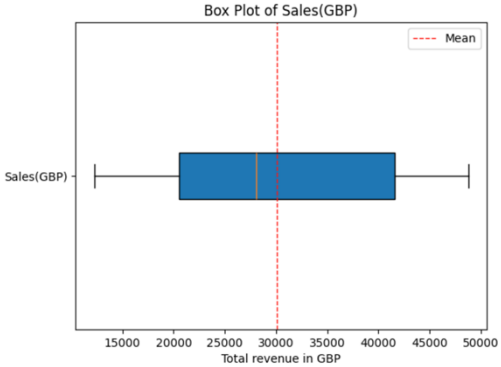


Fig.39 Box plot of E-commerce Revenue Sales

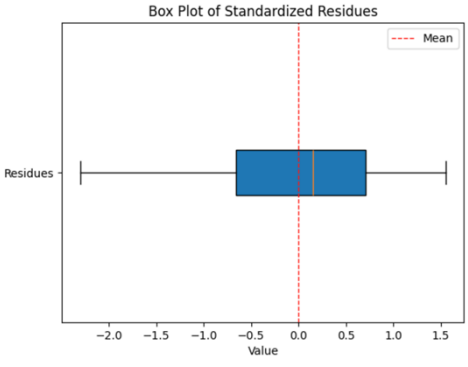


Fig.40 Box plot of residual

By calculating number of classes and frequency (Fig.41, 44, 47), the histogram is plotted as Fig.42, Fig.45 and Fig.48.

Socia Media Engagement
Rounded Number of classes: 5
class Range: 728.4
Classes:

	Length	Frequency
0	1858.4000	4
1	2586.8000	6
2	3315.2000	4
3	4043.6000	2
4	4772.0000	8

Fig.41 Class and Freq. of Social Media Engagement

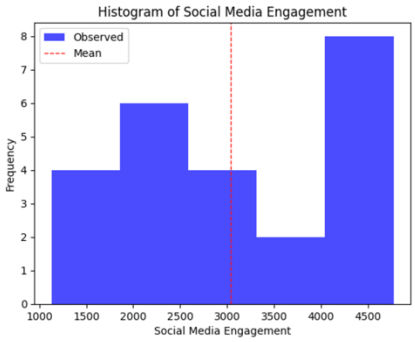


Fig.42 Histogram of Media Engagement

	CDF	Bin Frequency	Expected Value
0	0.157698	0.157698	3.784741
1	0.349722	0.192024	4.608587
2	0.591663	0.241940	5.806572
3	0.802259	0.210597	5.054320
4	0.928895	0.126636	3.039268
5	Sum =	0.928895	24.000000

Chi-Square Table for Social Media Engagement:

	χ^2
0	0.0122
1	0.4201
2	0.5621
3	1.8457
4	8.0970

Sum 10.9371
Number of classes (bins) using Sturges' formula: 5
Chi-square Statistic: 10.937097143838624,
p-value: 0.00661087837284613

Fig.43 CDF and Chi-square Table of Media Engagement

Sales Revenue in GBP
Rounded Number of classes: 5
class Range: 7302.290000000001
Classes:

	Length	Frequency
0	19550.8900	6
1	26853.1800	6
2	34155.4700	2
3	41457.7600	4
4	48760.0500	6

Fig.44 Class and Freq. of Sales Revenue

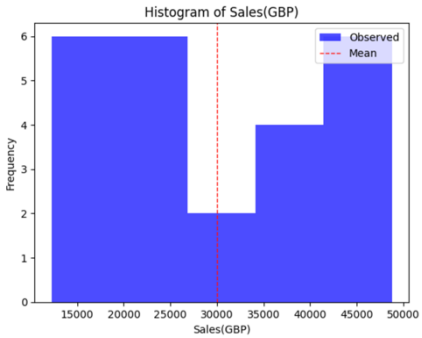


Fig.45 Histogram of Sales

	CDF	Bin Frequency	Expected Value
0	0.194625	0.194625	4.670993
1	0.396754	0.202130	4.851111
2	0.632117	0.235362	5.648700
3	0.82554	0.193423	4.642163
4	0.93772	0.112179	2.692306
5	Sum =	0.937720	24.000000

Chi-Square Table for Sales Revenue:

	χ^2
0	0.3781
1	0.2721
2	2.3568
3	0.0888
4	4.0637

Sum 7.1596
Number of classes (bins) using Sturges' formula: 5
Chi-square Statistic: 7.159626732570177,
p-value: 0.03650465103409808

Fig.46 CDF and Chi-square Table of Sales

Rounded Number of classes: 5
class Range: 0.7708082854349865
Classes:

	Standard_Residuals	Frequency
0	−1.5303	2
1	−0.7595	3
2	0.0113	3
3	0.7821	10
4	1.5529	6

Fig.47 Class and Freq. of Sales Revenue

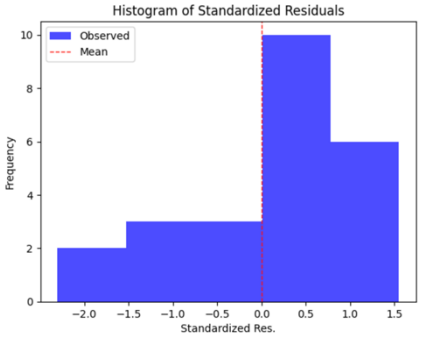


Fig.48 Histogram of Sales

	CDF	Bin Frequency	Expected Value
0	0.0671	0.0671	1.6092
1	0.2286	0.1615	3.8766
2	0.5044	0.2758	6.6198
3	0.7780	0.2736	6.5675
4	0.9358	0.1577	3.7853
5	Sum =	0.9358	22.4585

Chi-Square Table:

	χ^2
0	0.0949
1	0.1982
2	1.9794
3	1.7940
4	1.2958

Sum 5.3623
Chi-square Statistic: 4.918871264866541,
p-value: 0.08548318129124735

Fig.49 CDF and Chi-square Table of Sales

Furthermore, based on the normality test with chi-square (Fig.43, 46 and Fig.49) of social media engagement, e-commerce sales and standardized residual, it reveals that both variables are not normally distributed because their p-value of the chi-square are less than 0.05. However, because the p-value for residual is 0.085, which greater than 0.05, it indicates that the standerdised residual comes from normal distribution. As a result, the Pearson's correlation coefficient is still appropriate to evaluate the correlation between two variables.

Question 3. (ii) (b)

The Pearson's correlation coefficient of social media engagement and e-commerce sales is 0.9885, which is quite close to 1. It represents that the relationship of social media engagement and e-commerce sales is highly positively correlated. Thus, to achieve high e-commerce revenue sales, the recommended strategy could be improving the marketing activities on social media. Because when the likes, comments, and shares on social media boosts, the revenue of e-commerce sales has strong correlation to be increased.

Appendix

This report is analysed with Excel and Python. The python code is provided as below.

Q1

https://drive.google.com/file/d/1V2jyh6Cbu-gPkbrOSnHzWDyloo66LIG6/view?usp=sharing

```python
# Calculating the boundaries for outliers
Q1 = REI_df['REI(x)'].quantile(0.25)
Q3 = REI_df['REI(x)'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)
# Displaying the results
print("Box Plot Values:")
print(f"Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
print(f"Lower Bound for Outliers: {lower_bound}")
print(f"Upper Bound for Outliers: {upper_bound}\n")

# Calculate mean and standard dviation
mean = np.mean(REI)
std_dev = np.std(REI, ddof=1)


# Box plot
plt.boxplot(REI, vert=False, patch_artist=True)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Box Plot of REI(x)')
plt.xlabel('Index')
plt.ylabel('REI (X)')
plt.yticks([1], [])
plt.legend()
plt.show()
```

```python
# Calculate Sturges' number of bins
n = len(REI)
k = int(1 + 3.322 * np.log10(n))
classrange=np.ptp(REI)/k
print("\033[1mREI(x)\033[0m")

print("Rounded Number of classes:",k)
print("class Range:",classrange)
# Histogram and expected frequencies
hist, bin_edges = np.histogram(REI, bins=k)
cdf_values = stats.norm.cdf(bin_edges+classrange, mean, std_dev)


# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Length': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values)-1):
    if i==0:
      frequency=cdf_values[0]
    else:
      frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies= np.array(bin_frequencies)
# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())
```

```python
# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized,ddof=2)
hist_df.Length=hist_df.Length+classrange
print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]

# Histogram
plt.hist(REI, bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of REI(x)')
plt.xlabel('REI(x)')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print("Chi-Square Table for REI(x):")
print(chi_square_df)

# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})
```

```python
# Calculating the boundaries for outliers
Q1 = Mortality_Rate_df['Mortality Rate (y)'].quantile(0.25)
Q3 = Mortality_Rate_df['Mortality Rate (y)'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)
# Displaying the results
print("Box Plot Values:")
print(f"Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
print(f"Lower Bound for Outliers: {lower_bound}")
print(f"Upper Bound for Outliers: {upper_bound}\n")

# Calculate mean and standard dviation
mean = np.mean(Mortality_Rate)
std_dev = np.std(Mortality_Rate, ddof=1)


# Box plot
plt.boxplot(Mortality_Rate, vert=False, patch_artist=True)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Box Plot of Mortality Rate (y)')
plt.xlabel('Rate')
plt.yticks([1], [])
plt.ylabel('Mortality Rate (y)')
plt.legend()
plt.show()
```

```python
# Calculate Sturges' number of bins
n = len(Mortality_Rate)
k = int(1 + 3.322 * np.log10(n))
classrange=np.ptp(Mortality_Rate)/k
print("\033[1mMortality Rate (y)\033[0m")

print("Rounded Number of classes:",k)
print("class Range:",classrange)
# Histogram and expected frequencies
hist, bin_edges = np.histogram(Mortality_Rate, bins=k)
cdf_values = stats.norm.cdf(bin_edges+classrange, mean, std_dev)


# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Length': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values)-1):
    if i==0:
      frequency=cdf_values[0]
    else:
      frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies= np.array(bin_frequencies)
# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())
```

```python
# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized,ddof=2)
hist_df.Length=hist_df.Length+classrange
print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]

# Histogram
plt.hist(Mortality_Rate, bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of Mortality Rate (y)')
plt.xlabel('Mortality Rate (y)')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print("Chi-Square Table for Mortality Rate (y):")
print(chi_square_df)

# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})
```

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from scipy.stats import spearmanr

data = {
    'REI(x)': [8.3, 6.4, 3.4, 3.8, 2.6, 11.6, 1.2, 2.5, 1.6],
    'Mortality Rate (y)': [210, 180, 130, 170, 130, 210, 120, 150, 140]
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)

#calculate persons correlation coefficient
pearson_corr, _ = pearsonr(df['REI(x)'], df['Mortality Rate (y)'])
print(pearson_corr)

# Calculate the Spearman's rank correlation coefficient
spearman_correlation_coefficient, _ = spearmanr(df['REI(x)'], df['Mortality Rate (y)']) #_ represent p-value,

# Plot the data and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(df['REI(x)'], df['Mortality Rate (y)'], color='blue', label='Actual Mortality Rate (y)')
plt.title('REI(x) vs Mortality Rate (y)')
plt.xlabel('REI(x)')
plt.ylabel('Mortality Rate (y)')
plt.legend()
plt.text(6, 130, f'r= {pearson_corr:.4f}', fontsize=12, color='red')
plt.show()

# Output the Spearman's rank correlation coefficient
spearman_correlation_coefficient
```

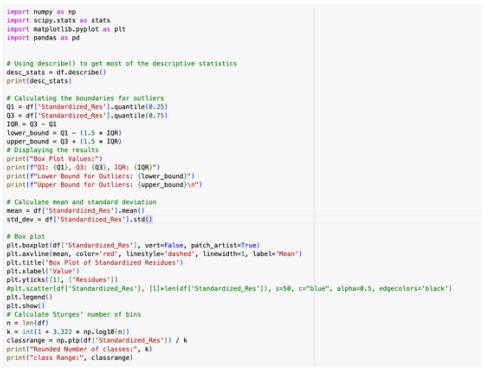
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm


data = {
    'REI(x)': [8.3, 6.4, 3.4, 3.8, 2.6, 11.6, 1.2, 2.5, 1.6],
    'Mortality Rate (y)': [210, 180, 130, 170, 130, 210, 120, 150, 140]
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Calculating descriptive statistics
descriptive_stats = df.describe()

# Adding a constant to the independent value for statsmodels
X = sm.add_constant(df['REI(x)'])
y = df['Mortality Rate (y)']

# Creating an Ordinary Least Squares (OLS) regression model with statsmodels library
model = sm.OLS(y, X).fit()
y_pred = model.predict(X)

# Getting the model parameters for the formula
intercept, slope = model.params

# Calculating MSE and R^2
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
```

```python
# Plotting the scatter plot with the regression line
plt.figure(figsize=(10, 6))
plt.scatter(df['REI(x)'], df['Mortality Rate (y)'], color='blue', label='Actual Data')
plt.scatter(df['REI(x)'], y_pred, color='green', label='Estimated Data')
plt.plot(df['REI(x)'], y_pred, color='red', label='Regression Line')
plt.title('Linear Regression of REI(x) on Mortality Rate (y)')
plt.xlabel('REI(x)')
plt.ylabel('Mortality Rate (y)')
plt.legend()

# Calculating the boundaries for REI(x)
Q1_x = df['REI(x)'].quantile(0.25)
Q3_x = df['REI(x)'].quantile(0.75)
IQR_x = Q3_x - Q1_x
lower_bound_x = Q1_x - (1.5 * IQR_x)
upper_bound_x = Q3_x + (1.5 * IQR_x)


# Calculating the boundaries for Mortality Rate (y)
Q1_y = df['Mortality Rate (y)'].quantile(0.25)
Q3_y = df['Mortality Rate (y)'].quantile(0.75)
IQR_y = Q3_y - Q1_y
lower_bound_y = Q1_y - (1.5 * IQR_y)
upper_bound_y = Q3_y + (1.5 * IQR_y)


# Adding the regression equation to the plot
plt.text(6, 140, f'r= {pearson_corr:.4f}', fontsize=12, color='blue')
plt.text(6, 135,f'Regression Equation: y = {slope:.2f}x + {(intercept:.2f})', fontsize=12, color='red')
plt.text(6, 130, f'R^2: {r2:.4f}', fontsize=12, color='red')
plt.text(6, 125, f'MSE= {mse:.4f}', fontsize=12, color='red')
plt.show()
```

```python
# Returning the descriptive statistics, the regression equation, MSE, and R-squared value
# descriptive_stats, f'Regression Equation: y = {slope:.2f}x + {intercept:.2f}', mse, r2
print(descriptive_stats)
print("\nBox Plot Values for REI(x):")
print(f"Q1: {Q1_x}, Q3: {Q3_x}, IQR: {IQR_x:.2f}")
print(f"Lower Bound for REI(X): {lower_bound_x:.2f}")
print(f"Upper Bound for REI(X): {upper_bound_x}")
print("\nBox Plot Values for Mortality Rate (y):")
print(f"Q1: {Q1_y}, Q3: {Q3_y}, IQR: {IQR_y}")
print(f"Lower Bound for Mortality Rate (y): {lower_bound_y}")
print(f"Upper Bound for Mortality Rate (y): {upper_bound_y}\n")


print(f'Regression Equation: y = {slope:.2f}x + {(intercept:.2f})')

print(f"MSE: {mse}")
print(f"R^2: {r2}")
```

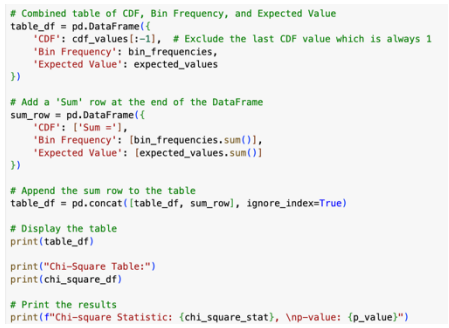
```python
# Adding a constant to the independent value
X = sm.add_constant(df['REI(x)'].values)

# Dependent variable
y = df['Mortality Rate (y)'].values

# Creating an OLS regression model
model = sm.OLS(y, X).fit()

# Getting the summary of the regression
regression_summary = model.summary()

# Get detailed tables from summary as DataFrames
tables = model.summary2().tables
regression_statistics_df = pd.DataFrame(tables[0])

regression_summary
```

```python
from scipy.stats import zscore

# Normality test for residuals:
# Using the same DataFrame 'df' from the previous analysis
# We will calculate the residuals and their standardized values (z-scores)

# Fitting the model
model = LinearRegression()
model.fit(df[['REI(x)']], df['Mortality Rate (y)'])

# Adding predictions to the dataframe
df['Fitted_value'] = model.predict(df[['REI(x)']])

# Residuals
df['Residual'] = df['Mortality Rate (y)'] - df['Fitted_value']

# Standardized Residuals
df['Standardized_Res'] = zscore(df['Residual'])

# Residual Output Table
residual_output = df[['Fitted_value', 'Standardized_Res']]

# Standardized Residual Descriptive Statistics
std_res_desc = df['Standardized_Res'].describe()
pd.options.display.float_format = '{:.4f}'.format
print(residual_output)

print("\nDescriptive statistics:")
std_res_desc

Q1 = df['Standardized_Res'].quantile(0.25)
Q3 = df['Standardized_Res'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

# Displaying the results
print("\nBox Plot Values:")
print("Q1:", Q1 , "Q3:", Q3, "IQR:", IQR)
print("Lower Bound for Outliers:", lower_bound)
print("Upper Bound for Outliers:", upper_bound)
```

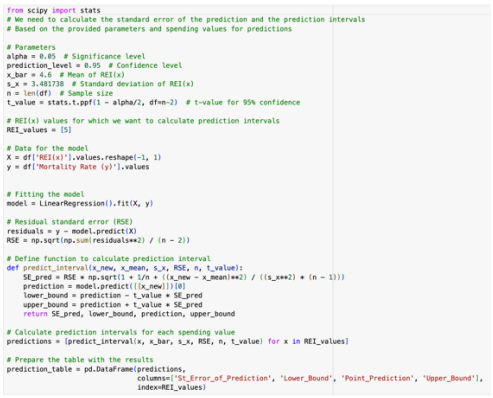


```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd


# Using describe() to get most of the descriptive statistics
desc_stats = df.describe()
print(desc_stats)

# Calculating the boundaries for outliers
Q1 = df['Standardized_Res'].quantile(0.25)
Q3 = df['Standardized_Res'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)
# Displaying the results
print("Box Plot Values:")
print(f"Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
print(f"Lower Bound for Outliers: {lower_bound}")
print(f"Upper Bound for Outliers: {upper_bound}\n")

# Calculate mean and standard deviation
mean = df['Standardized_Res'].mean()
std_dev = df['Standardized_Res'].std()

# Box plot
plt.boxplot(df['Standardized_Res'], vert=False, patch_artist=True)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Box Plot of Standardized Residues')
plt.xlabel('Value')
plt.yticks([1], ['Residues'])
#plt.scatter(df['Standardized_Res'], [1]*len(df['Standardized_Res']), s=50, c="blue", alpha=0.5, edgecolors='black')
plt.legend()
plt.show()
# Calculate Sturges' number of bins
n = len(df)
k = int(1 + 3.322 * np.log10(n))
classrange = np.ptp(df['Standardized_Res']) / k
print("Rounded Number of classes:", k)
print("Class Range:", classrange)
```

```
# Histogram and expected frequencies
hist, bin_edges = np.histogram(df['Standardized_Res'], bins=k)
cdf_values = stats.norm.cdf(bin_edges + classrange, mean, std_dev)

# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Standard_Residuals': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values) - 1):
    if i == 0:
        frequency = cdf_values[0]
    else:
        frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies = np.array(bin_frequencies)

# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())

# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized, ddof=2)
hist_df.Standard_Residuals = hist_df.Standard_Residuals + classrange

print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]
# Histogram
plt.hist(df['Standardized_Res'], bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of Standardized Residuals')
plt.xlabel('Standardized Res.')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



```
# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})

# Add a 'Sum' row at the end of the DataFrame
sum_row = pd.DataFrame({
    'CDF': ['Sum ='],
    'Bin Frequency': [bin_frequencies.sum()],
    'Expected Value': [expected_values.sum()]
})

# Append the sum row to the table
table_df = pd.concat([table_df, sum_row], ignore_index=True)

# Display the table
print(table_df)

print("Chi-Square Table:")
print(chi_square_df)

# Print the results
print(f"Chi-square Statistic: {chi_square_stat}, \np-value: {p_value}")
```



```
from scipy import stats
# We need to calculate the standard error of the prediction and the prediction intervals
# Based on the provided parameters and spending values for predictions

# Parameters
alpha = 0.05  # Significance level
prediction_level = 0.95  # Confidence level
x_bar = 4.6  # Mean of REI(x)
s_x = 3.481738  # Standard deviation of REI(x)
n = len(df)  # Sample size
t_value = stats.t.ppf(1 - alpha/2, df=n-2)  # t-value for 95% confidence

# REI(x) values for which we want to calculate prediction intervals
REI_values = [5]

# Data for the model
X = df['REI(x)'].values.reshape(-1, 1)
y = df['Mortality Rate (y)'].values


# Fitting the model
model = LinearRegression().fit(X, y)

# Residual standard error (RSE)
residuals = y - model.predict(X)
RSE = np.sqrt(np.sum(residuals**2) / (n - 2))

# Define function to calculate prediction interval
def predict_interval(x_new, x_mean, s_x, RSE, n, t_value):
    SE_pred = RSE * np.sqrt(1 + 1/n + ((x_new - x_mean)**2) / ((s_x**2) * (n - 1)))
    prediction = model.predict([[x_new]])[0]
    lower_bound = prediction - t_value * SE_pred
    upper_bound = prediction + t_value * SE_pred
    return SE_pred, lower_bound, prediction, upper_bound

# Calculate prediction intervals for each spending value
predictions = [predict_interval(x, x_bar, s_x, RSE, n, t_value) for x in REI_values]

# Prepare the table with the results
prediction_table = pd.DataFrame(predictions,
                                columns=['St_Error_of_Prediction', 'Lower_Bound', 'Point_Prediction', 'Upper_Bound'],
                                index=REI_values)
```

```
# Add t-value and error columns
prediction_table['t_value'] = t_value
prediction_table['Error'] = RSE

# Reorder columns for the final table
prediction_table = prediction_table[['t_value', 'St_Error_of_Prediction', 'Error', 'Lower_Bound', 'Point_Prediction', 'Upper_Bound']]

prediction_table.reset_index().rename(columns={'index': 'REI(x)'})

# Calculate prediction intervals and error based on t-value * St. Error of Prediction
prediction_table['Error'] = prediction_table['t_value'] * prediction_table['St_Error_of_Prediction']

# Reorder columns for the final table after updating the Error column
prediction_table = prediction_table[['t_value', 'St_Error_of_Prediction', 'Error', 'Lower_Bound', 'Point_Prediction', 'Upper_Bound']]

# Prepare the residual plots again with the line connecting points on the second scatter plot
plt.figure(figsize=(12, 5))

# Residuals vs Fitted values
plt.subplot(1, 2, 1)
plt.scatter(df['Fitted_value'], df['Standardized_Res'])
plt.axhline(0, color='grey', linestyle='--')
plt.xlabel('Fitted value')
plt.ylabel('Standardized Res.')
plt.title('Standardized Residuals vs Fitted Values')

# Residuals vs Order of data with line connecting points
plt.subplot(1, 2, 2)
plt.plot(df.index, df['Standardized_Res'], marker='o', linestyle='-')  # Line added here
plt.axhline(0, color='grey', linestyle='--')
plt.xlabel('Order')
plt.ylabel('Standardized Res.')
plt.title('Standardized Residuals vs Order')

plt.tight_layout()
plt.show()

# Show the updated prediction table
prediction_table.reset_index().rename(columns={'index': 'REI(x)'})
```

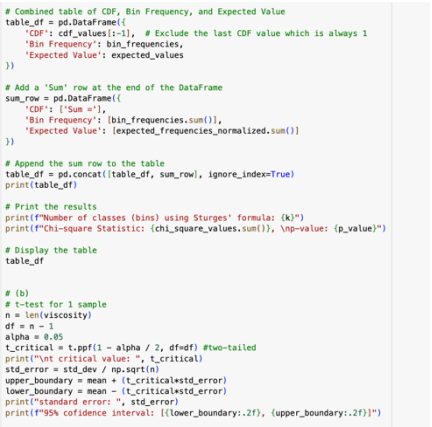
Q2

https://colab.research.google.com/drive/1LeJzxL3xsMiKr4yjKgvgOMEOTjfiK2dz?usp=sharing

```
# Q2 (ii)

import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import t
from scipy.stats import ttest_1samp

viscosity = np.array([
    33.75, 33.85, 34.80, 33.81, 33.46, 34.02, 33.58, 33.27, 33.49, 33.20, 34.62, 33.00, 33.54, 34.12, 33.84
])
viscosity_df = pd.DataFrame(viscosity, columns=['viscosity(sec)'])

# Using describe() to get the descriptive statistics
desc_stats_viscosity = viscosity_df.describe()
print("Descriptive statistics of tested paints:\n", desc_stats_viscosity)

# (a)
# Calculate mean and standard dviation
mean = np.mean(viscosity)
std_dev = np.std(viscosity, ddof=1)
min = np.min(viscosity)
max = np.max(viscosity)


# Scatter
plt.scatter([1]*len(viscosity), viscosity, s=50, c='blue', alpha=0.6, edgecolors='black')
target = 34
upper_target = 34 + 1
lower_target = 34 - 1
plt.axhline(mean, color='red', linestyle='--', linewidth=1, label='Mean')
plt.axhline(upper_target, color='green', linestyle='--', linewidth=1, label=f'Upper Target = {upper_target:.2f}')
plt.axhline(lower_target, color='green', linestyle='--', linewidth=1, label=f'Lower Target = {lower_target:.2f}')
plt.title("Scatter plot of the tested paints")
plt.xlabel("test")
plt.ylabel("Viscosity (sec)")
plt.xticks(np.arange(1, 1, 1))
plt.legend(loc='center right')
plt.show()
```

```
# Calculating the boundaries for outliers
Q1 = viscosity_df['viscosity(sec)'].quantile(0.25)
Q3 = viscosity_df['viscosity(sec)'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

# Displaying the results
print("\nBox Plot Values:")
print(f"Q1: {Q1}, Q3: {Q3} , IQR: {IQR:.4f}")
print("Lower Bound for Outliers:", lower_bound)
print("Upper Bound for Outliers:", upper_bound)

# Box plot
plt.boxplot(viscosity, vert=True, patch_artist=False)
plt.scatter([1]*len(viscosity), viscosity, s=50, c="blue", alpha=0.5, edgecolors='black')
plt.axhline(upper_bound, color='green', linestyle="--", linewidth=1, label=f"Upper bound= {upper_bound:.2f}")
plt.axhline(lower_bound, color='green', linestyle="--", linewidth=1, label=f"Lower bound= {lower_bound:.2f}")
plt.title("Box Plot of the viscosity for tested paints")
plt.ylabel('Viscosity (sec)')
plt.xticks([1], ['test'])
plt.legend(loc='center right')
plt.show()

# Calculate Sturges' number of bins
n = len(viscosity)
k = int(1 + 3.322 * np.log10(n))
classrange=np.ptp(viscosity)/k
print("\nRounded Number of classes:",k)
print("Class Range:",classrange)

# Histogram and expected frequencies
hist, bin_edges = np.histogram(viscosity, bins=k)
cdf_values = stats.norm.cdf(bin_edges+classrange, mean, std_dev)
```

```
# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Length': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values)-1):
    if i==0:
        frequency=cdf_values[0]
    else:
        frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies= np.array(bin_frequencies)
# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())

# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized,ddof=2)
hist_df.Length=hist_df.Length+classrange
print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]

# Histogram
plt.hist(viscosity_df, bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of the viscosity for an industrial paint')
plt.xlabel('viscosity (sec)')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print("Chi-Square Table:")
print(chi_square_df)
```



```
# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})

# Add a 'Sum' row at the end of the DataFrame
sum_row = pd.DataFrame({
    'CDF': ['Sum ='],
    'Bin Frequency': [bin_frequencies.sum()],
    'Expected Value': [expected_frequencies_normalized.sum()]
})

# Append the sum row to the table
table_df = pd.concat([table_df, sum_row], ignore_index=True)
print(table_df)

# Print the results
print(f"Number of classes (bins) using Sturges' formula: {k}")
print(f"Chi-square Statistic: {chi_square_values.sum()}, \np-value: {p_value}")

# Display the table
table_df


# (b)
# t-test for 1 sample
n = len(viscosity)
df = n - 1
alpha = 0.05
t_critical = t.ppf(1 - alpha / 2, df=df) #two-tailed
print("\nt critical value: ", t_critical)
std_error = std_dev / np.sqrt(n)
upper_boundary = mean + (t_critical*std_error)
lower_boundary = mean - (t_critical*std_error)
print("standard error: ", std_error)
print(f"95% cofidence interval: [{lower_boundary:.2f}, {upper_boundary:.2f}]")
```

```
# (c)
tstat_33 = (33 - mean) / std_error
tstat_35 = (35 - mean) / std_error


p_value_33 = t.sf(abs(tstat_33), df)
p_value_35 = t.sf(abs(tstat_35), df)
print("t-stat of 33 = ", tstat_33," P(x < 33)= ",p_value_33)
print("t-stat of 35 = ", tstat_35,", P(x < 5)= ",p_value_35)
prob =  p_value_35 + p_value_33
print("The probability of not in the specification: ", prob)

t_stats_df_33 = pd.DataFrame([int(33), tstat_33, p_value_33], index=['x', 't-stat', 'P(T > |t|)one-tailed']).T
t_stats_df_35 = pd.DataFrame([int(35), tstat_35, p_value_35], index=['x', 't-stat', 'P(T > |t|)one-tailed']).T
t_stats_df_sum = pd.DataFrame(["Sum: ", prob]).T
result = pd.concat([t_stats_df_33, t_stats_df_35, t_stats_df_sum], ignore_index=True)
print(result)
```

Q3

(i) https://colab.research.google.com/drive/1YLnssFbef7yDTWSt9KAR0F8vzlBYEVFL?usp=sharing

```python
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import t

# Q3 (i)
disc1 = np.array([55, 72, 26, 65, 104, 62, 89, 52, 76, 46, 100, 72, 35])
disc1_df = pd.DataFrame(disc1, columns=['disc1(microsec)'])

# Using describe() to get the descriptive statistics
desc_stats_disc1 = disc1_df.describe()
print("Descriptive statistics for the access response time of Disc#1:\n", desc_stats_disc1)

# Calculate mean and standard dviation
mean = np.mean(disc1)
std_dev = np.std(disc1, ddof=1)
min = np.min(disc1)
max = np.max(disc1)

# Calculating the boundaries for outliers
Q1 = disc1_df['disc1(microsec)'].quantile(0.25)
Q3 = disc1_df['disc1(microsec)'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

# Box plot
plt.boxplot(disc1, vert=True, patch_artist=False)
plt.scatter([1]*len(disc1), disc1, s=50, c="blue", alpha=0.5, edgecolors='black')
plt.axhline(upper_bound, color='green', linestyle="--", linewidth=1, label=f"Upper bound= {upper_bound:.2f}")
plt.axhline(lower_bound, color='green', linestyle="--", linewidth=1, label=f"Lower bound= {lower_bound:.2f}")
plt.title("Box Plot of the access response time for Disc#1")
plt.ylabel('Response time(microsec)')
plt.xticks([1], ['test'])
plt.legend(loc='center right')
plt.show()
# Displaying the results
print("Box Plot Values:")
print("Q1:", Q1 , "Q3:", Q3, "IQR:", IQR)
print("Lower Bound for Outliers:", lower_bound)
print("Upper Bound for Outliers:", upper_bound)
```

```python
# Calculate Sturges' number of bins
n = len(disc1)
k = int(1 + 3.322 * np.log10(n))
classrange=np.ptp(disc1)/k
print("\nDisc #1")
print("Rounded Number of classes:",k)
print("class Range:",classrange)

# Histogram and expected frequencies
hist, bin_edges = np.histogram(disc1, bins=k)
cdf_values = stats.norm.cdf(bin_edges+classrange, mean, std_dev)

# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Length': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values)-1):
    if i==0:
      frequency=cdf_values[0]
    else:
      frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies= np.array(bin_frequencies)
# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())

# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized,ddof=2)
hist_df.Length=hist_df.Length+classrange
print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]
```

```python
# Histogram
plt.hist(disc1_df, bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of the access response time of Disc#1')
plt.xlabel('response time(microsec)')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print("Chi-Square Table of Disc#1:")
print(chi_square_df)

# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})

# Add a 'Sum' row at the end of the DataFrame
sum_row = pd.DataFrame({
    'CDF': ['Sum ='],
    'Bin Frequency': [bin_frequencies.sum()],
    'Expected Value': [expected_frequencies_normalized.sum()]
})

# Append the sum row to the table
table_df = pd.concat([table_df, sum_row], ignore_index=True)


# Print the results
print(f"Number of classes (bins) using Sturges' formula: {k}")
print(f"Chi-square Statistic: {chi_square_values.sum()}, \np-value: {p_value}")


# Display the table
table_df
```

```python
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import t

# Q3 (i)-continued
disc2 = np.array([71, 62, 37, 46, 36, 44, 63, 71, 49, 44, 41, 76, 55, 64, 41])
disc2_df = pd.DataFrame(disc2, columns=['disc2(microsec)'])

# Using describe() to get the descriptive statistics
desc_stats_disc2 = disc2_df.describe()
print("Descriptive statistics for the access response time of Disc#2:\n", desc_stats_disc2)

# Calculate mean and standard dviation
mean = np.mean(disc2)
std_dev = np.std(disc2, ddof=1)
min = np.min(disc2)
max = np.max(disc2)

# Calculating the boundaries for outliers
Q1 = disc2_df['disc2(microsec)'].quantile(0.25)
Q3 = disc2_df['disc2(microsec)'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

# Box plot
plt.boxplot(disc2, vert=True, patch_artist=False)
plt.scatter([1]*len(disc2), disc2, s=50, c="blue", alpha=0.5, edgecolors='black')
plt.axhline(upper_bound, color='green', linestyle="--", linewidth=1, label=f"Upper bound= {upper_bound:.2f}")
plt.axhline(lower_bound, color='green', linestyle="--", linewidth=1, label=f"Lower bound= {lower_bound:.2f}")
plt.title("Box Plot of the access response time for Disc#2")
plt.ylabel('Response time(microsec)')
plt.xticks([1], ['test'])
plt.legend(loc='center right')
plt.show()
# Displaying the results
print("\nBox Plot Values:")
print("Q1:", Q1 , "Q3:", Q3, "IQR:", IQR)
print("Lower Bound for Outliers:", lower_bound)
print("Upper Bound for Outliers:", upper_bound)
```

```python
# Calculate Sturges' number of bins
n = len(disc2)
k = int(1 + 3.322 * np.log10(n))
classrange=np.ptp(disc2)/k
print("\nDisc # 2")
print("Rounded Number of classes:",k)
print("class Range:",classrange)

# Histogram and expected frequencies
hist, bin_edges = np.histogram(disc2, bins=k)
cdf_values = stats.norm.cdf(bin_edges+classrange, mean, std_dev)

# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Length': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values)-1):
    if i==0:
      frequency=cdf_values[0]
    else:
      frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies= np.array(bin_frequencies)
# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())

# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized,ddof=2)
hist_df.Length=hist_df.Length+classrange
print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]
```

```python
# Histogram
plt.hist(disc2_df, bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of the access response time of Disc#2')
plt.xlabel('response time(microsec)')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print("Chi-Square Table of Disc#2:")
print(chi_square_df)

# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})

# Add a 'Sum' row at the end of the DataFrame
sum_row = pd.DataFrame({
    'CDF': ['Sum ='],
    'Bin Frequency': [bin_frequencies.sum()],
    'Expected Value': [expected_frequencies_normalized.sum()]
})

# Append the sum row to the table
table_df = pd.concat([table_df, sum_row], ignore_index=True)


# Print the results
print(f"Number of classes (bins) using Sturges' formula: {k}")
print(f"Chi-square Statistic: {chi_square_values.sum()}, \np-value: {p_value}")


# Display the table
table_df
```

(ii) https://colab.research.google.com/drive/1L0JvkLG2ynQHnOh-CAWcLmWT_ErPh7La?usp=sharing

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


social_media = np.array([
    4174, 4507, 1860, 2294, 2130, 2095, 4772, 4092, 2638, 3169, 1466, 2238, 1330, 2482, 3135, 4444, 4171, 3919, 4735, 1130, 2685, 4380, 1769, 3391
    ])
social_media_df = pd.DataFrame(social_media, columns=['Socia Media Engagement'])

# Using describe() to get most of the descriptive statistics
desc_stats = social_media_df.describe()
print(desc_stats)

# Calculating the boundaries for outliers
Q1 = social_media_df['Socia Media Engagement'].quantile(0.25)
Q3 = social_media_df['Socia Media Engagement'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)
# Displaying the results
print("Box Plot Values:")
print(f"Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
print(f"Lower Bound for Outliers: {lower_bound}")
print(f"Upper Bound for Outliers: {upper_bound}\n")

# Calculate mean and standard dviation
mean = np.mean(social_media)
std_dev = np.std(social_media, ddof=1)


# Box plot
plt.boxplot(social_media, vert=False, patch_artist=True)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Box Plot of Social Media Engagement')
plt.xlabel('Total likes, comments, and shares')
plt.yticks([1], [])
plt.ylabel('Social Media Engagement')
plt.legend()
plt.show()
```

```python
# Calculate Sturges' number of bins
n = len(social_media)
k = int(1 + 3.322 * np.log10(n))
classrange=np.ptp(social_media)/k
print("\033[1mSocia Media Engagement\033[0m")

print("Rounded Number of classes:",k)
print("class Range:",classrange)
# Histogram and expected frequencies
hist, bin_edges = np.histogram(social_media, bins=k)
cdf_values = stats.norm.cdf(bin_edges+classrange, mean, std_dev)


# Create a DataFrame for histogram data
hist_df = pd.DataFrame({'Length': bin_edges[:-1], 'Frequency': hist})

bin_frequencies = []
for i in range(0, len(cdf_values)-1):
    if i==0:
      frequency=cdf_values[0]
    else:
      frequency = cdf_values[i] - cdf_values[i - 1]
    bin_frequencies.append(frequency)
bin_frequencies= np.array(bin_frequencies)
# Expected values
expected_values = bin_frequencies * n

# Normalize expected frequencies
expected_frequencies_normalized = expected_values * (hist.sum() / expected_values.sum())

# Calculate chi-square values for each bin
chi_square_values = (hist - expected_values) ** 2 / expected_values
chi_square_df = pd.DataFrame({'χ^2': chi_square_values})
# Add a row for the sum of chi-square values
chi_square_df.loc['Sum'] = chi_square_df.sum()

# Chi-square goodness of fit test
chi_square_stat, p_value = stats.chisquare(hist, f_exp=expected_frequencies_normalized,ddof=2)
hist_df.Length=hist_df.Length+classrange
print("Classes:")
print(hist_df, "\n")
bin_centers = bin_edges[1:]
```

```python
# Histogram
plt.hist(social_media, bins=bin_edges, alpha=0.7, color='blue', label='Observed')
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.title('Histogram of Social Media Engagement')
plt.xlabel('Social Media Engagement')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print("Chi-Square Table for Social Media Engagement:")
print(chi_square_df)

# Combined table of CDF, Bin Frequency, and Expected Value
table_df = pd.DataFrame({
    'CDF': cdf_values[:-1],  # Exclude the last CDF value which is always 1
    'Bin Frequency': bin_frequencies,
    'Expected Value': expected_values
})

# Add a 'Sum' row at the end of the DataFrame
sum_row = pd.DataFrame({
    'CDF': ['Sum ='],
    'Bin Frequency': [bin_frequencies.sum()],
    'Expected Value': [expected_frequencies_normalized.sum()]
})

# Append the sum row to the table
table_df = pd.concat([table_df, sum_row], ignore_index=True)

# Print the results
print(f"Number of classes (bins) using Sturges' formula: {k}")
print(f"Chi-square Statistic: {chi_square_values.sum()}, \np-value: {p_value}")


# Display the table
print("\033[1mSocia Media Engagement (x)\033[0m")
table_df
print(table_df)
```

```python
# Correlations
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from scipy.stats import spearmanr

data = {
    'Social Media Engagement (x)': [4174, 4507, 1860, 2294, 2130, 2095, 4772, 4092, 2638, 3169, 1466, 2238, 1330, 2482, 3135, 4444, 4171, 3919, 4735, 1130, 2685, 4380, 1769, 3391],
    'Sales (y)': [illegible]
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)

#calculate persons correlation coefficient
pearson_corr, _ = pearsonr(df['Social Media Engagement (x)'], df['Sales (y)'])
print("Pearson's correlation coefficient=", pearson_corr)

#Calculate the Spearman's rank correlation coefficient
spearman_correlation_coefficient, _ = spearmanr(df['Social Media Engagement (x)'], df['Sales (y)']) #_ represent p-value, but in this questions no need p-value, so put "_"

# Plot the data and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(df['Social Media Engagement (x)'], df['Sales (y)'], color='blue', label='Actual Sales (y)')
plt.title('Social Media Engagement (x) vs E-commerce Revenue Sales (y)')
plt.xlabel('Social Media Engagement (x)')
plt.ylabel('Sales in GBP (y)')
plt.legend()
#plt.text(3500, 20000, f"Pearson = {pearson_corr:.4f}", fontsize=12, color='red')
#plt.text(3500, 18000, f"Spearman = {spearman_correlation_coefficient:.4f}", fontsize=12, color='red')
plt.show()
```

```python
# Adding a constant to the independent value for statsmodels
X = sm.add_constant(df['Social Media Engagement (x)'])
y = df['Sales (y)']

# Creating an Ordinary Least Squares (OLS) regression model with statsmodels library
model = sm.OLS(y, X).fit()
y_pred = model.predict(X)

# Getting the model parameters for the formula
intercept, slope = model.params

# Calculating MSE and R^2
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# Plotting the scatter plot with the regression line
plt.figure(figsize=(10, 6))
plt.scatter(df['Social Media Engagement (x)'], df['Sales (y)'], color='blue', label='Actual Data')
plt.plot(df['Social Media Engagement (x)'], y_pred, color='red', label='Regression Line')
plt.title('Linear Regression of Social Media Engagement (x) on Sales (y)')
plt.xlabel('Social Media Engagement (x)')
plt.ylabel('Sales (y)')
plt.legend()


# Calculating the boundaries for REI(x)
Q1_x = df['Social Media Engagement (x)'].quantile(0.25)
Q3_x = df['Social Media Engagement (x)'].quantile(0.75)
IQR_x = Q3_x - Q1_x
lower_bound_x = Q1_x - (1.5 * IQR_x)
upper_bound_x = Q3_x + (1.5 * IQR_x)



# Calculating the boundaries for Mortality Rate (y)
Q1_y = df['Sales (y)'].quantile(0.25)
Q3_y = df['Sales (y)'].quantile(0.75)
IQR_y = Q3_y - Q1_y
lower_bound_y = Q1_y - (1.5 * IQR_y)
upper_bound_y = Q3_y + (1.5 * IQR_y)
```

```python
# Adding the regression equation to the plot
plt.text(2800, 16000, f'r= {pearson_corr:.4f}', fontsize=12, color='blue')
plt.text(2800, 14000,f'Regression Equation: y = {slope:.2f}x + ({intercept:.2f})', fontsize=12, color='red')
plt.text(2800, 12000, f'R^2: {r2:.4f}', fontsize=12, color='red')
plt.text(2800, 10000, f'MSE= {mse:.4f}', fontsize=12, color='red')

plt.show()


# Returning the descriptive statistics, the regression equation, MSE, and R-squared value
# descriptive_stats, f'Regression Equation: y = {slope:.2f}x + {intercept:.2f}', mse, r2
print(descriptive_stats)
print("\nBox Plot Values for Social Media Engagement (x):")
print(f"Q1: {Q1_x}, Q3: {Q3_x}, IQR: {IQR_x:.2f}")
print(f"Lower Bound for Social Media Engagement (x): {lower_bound_x:.2f}")
print(f"Upper Bound for Social Media Engagement (x): {upper_bound_x}")
print("\nBox Plot Values for Sales (y):")
print(f"Q1: {Q1_y}, Q3: {Q3_y}, IQR: {IQR_y}")
print(f"Lower Bound for Sales (y): {lower_bound_y}")
print(f"Upper Bound for Sales (y): {upper_bound_y}\n")


print(f'Regression Equation: y = {slope:.2f}x + ({intercept:.2f})')

print(f"MSE: {mse}")
print(f"R^2: {r2}")
```

```python
# Adding a constant to the independent value
X = sm.add_constant(df['Social Media Engagement (x)'].values)

# Dependent variable
y = df['Sales (y)'].values

# Creating an OLS regression model
model = sm.OLS(y, X).fit()

# Getting the summary of the regression
regression_summary = model.summary()

# Get detailed tables from summary as DataFrames
tables = model.summary2().tables
regression_statistics_df = pd.DataFrame(tables[0])

regression_summary
```

```python
from scipy.stats import zscore

# Normality test for residuals:
# Using the same DataFrame 'df' from the previous analysis
# We will calculate the residuals and their standardized values (z-scores)

# Fitting the model
model = LinearRegression()
model.fit(df[['Social Media Engagement (x)']], df['Sales (y)'])

# Adding predictions to the dataframe
df['Fitted_value'] = model.predict(df[['Social Media Engagement (x)']])

# Residuals
df['Residual'] = df['Sales (y)'] - df['Fitted_value']

# Standardized Residuals
df['Standardized_Res'] = zscore(df['Residual'])

# Residual Output Table
residual_output = df[['Fitted_value', 'Standardized_Res']]

# Standardized Residual Descriptive Statistics
std_res_desc = df['Standardized_Res'].describe()
pd.options.display.float_format = '{:.4f}'.format

print(residual_output)

print("\nDescriptive statistics:")
std_res_desc

Q1 = df['Standardized_Res'].quantile(0.25)
Q3 = df['Standardized_Res'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

# Displaying the results
print("\nBox Plot Values:")
print("Q1:", Q1 , "Q3:", Q3, "IQR:", IQR)
print("Lower Bound for Outliers:", lower_bound)
print("Upper Bound for Outliers:", upper_bound)
```