Monash University: Assessment Cover Sheet

Student name	Tan	Jin **Chun**	
School/Campus	**Monash University Malaysia**	**Student's I.D. number**	32194471
Unit name	ECE4076 - Computer vision - S1 2023		
Lecturer's name	**Dr. Maxine Tan**	**Tutor's name**	
Assignment name	Lab 4 Results Document Submission	**Group Assignment: No** **Note, each student must attach a coversheet**	
Lab/Tute Class: Friday Lab Session	**Lab/Tute Time: 10 a.m - 12 p.m**		**Word Count:**
Due date: 21-05-2023	**Submit Date: 21-05-2023**		**Extension granted** ☐

If an extension of work is granted, specify date and provide the signature of the lecturer/tutor. Alternatively, attach an email printout or handwritten and signed notice from your lecturer/tutor verifying an extension has been granted.

Extension granted until (date): ...../...../..... Signature of lecturer/tutor: .....

Late submissions policy	**Days late**	**Penalty applied**
Penalties apply to late submissions and may vary between faculties. Please refer to your faculty's late assessment policy for details.		

Patient/client confidentiality: Where a patient/client case study is undertaken a signed Consent Form must be obtained.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means to take and use another person's ideas and or manner of expressing them and to pass these off as one's own by failing to give appropriate acknowledgement. This includes material from any source, staff, students or the Internet - published and unpublished works.

Collusion: Collusion means unauthorised collaboration on assessable written, oral or practical work with another person. Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or nominee, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations (academic misconduct).
- I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may, for the purposes of assessment, reproduce the assignment and:

  i. provide it to another member of faculty and any external marker; and/or

  ii. submit to a text matching/originality checking software; and/or

  iii. submit it to a text matching/originality checking software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.

- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration or otherwise breached the academic integrity requirements in the Student Academic Integrity Policy.

Date: **21**/**05**/**2023** Signature: **Tan Jin Chun** *

Privacy Statement:

For information about how the University deals with your personal information go to http://privacy.monash.edu.au/guidelines/collection-personal-information.html#enrol

ECE4076 lab 4 results document

Name: Tan Jin Chun (32194471)

Task 1 (1 mark)

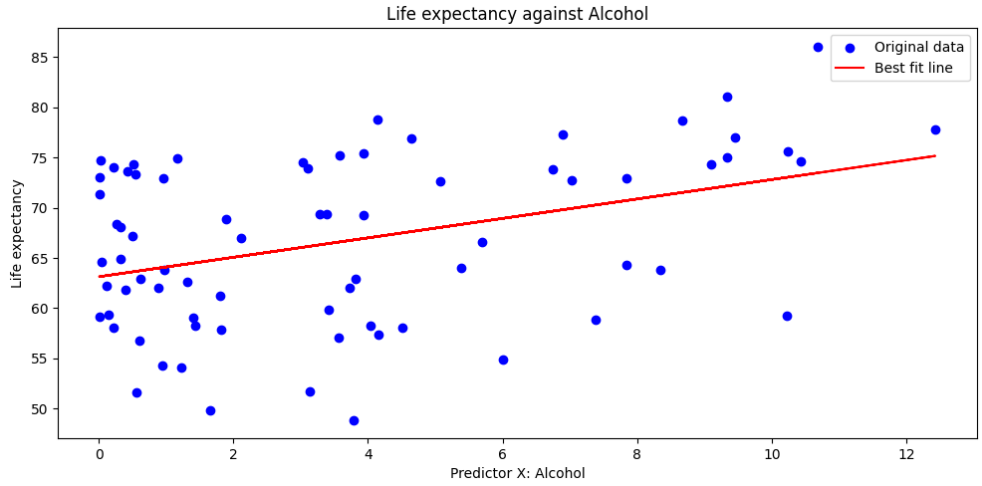
Regression coefficients for Alcohol data:
w_{0}= **63.130**
w_{1}= **0.967**

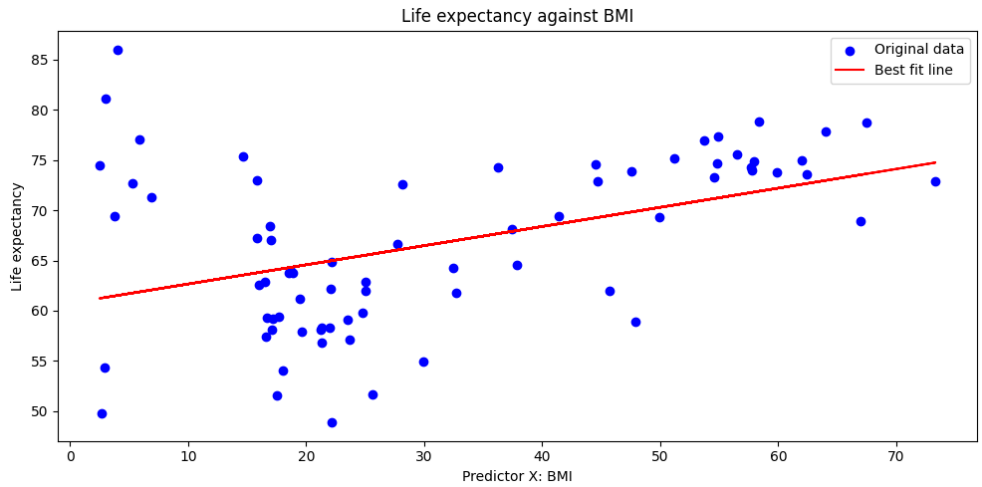
```
w[0] (intercept term): [63.13011473]
w[1] (coefficient for Alcohol): [0.96682896]
```

Regression coefficients for BMI data:
w_{0}= **60.749**
w_{1}= **0.191**

```
w[0] (intercept term): [60.74884974]
w[1] (coefficient for Alcohol): [0.19073734]
```

Insert scatter plots of Alcohol vs. Life expectancy and BMI vs. Life expectancy:





Judging by your visualisations of the data and the regression line, can you explain what the results would mean if you had to use this model to make a prediction? Does the dependency seem reasonable to you, and why / why not? What could the reason for this dependency in the data be?

Judging by my visualisations of the data and the regression line, if the regression line accurately fits the data points, this suggests that the model can make reliable predictions. As we can see from the above two models, the data point in the BMI plot above follows the regression line more closely than the data point in the alcohol plot above. When a new data is placed into the model with the stronger positive correlation, we should be able to predict an accurate result. The dependency however seems a bit off, based on the visualisations of the data and the regression line, we are basically stating that when the BMI/alcohol is high, the life expectancy will be higher. However, we can counter the fact that when people's BMI/alcohol is high, it means that they very likely have the medical resources to prolong their life as well due to the country that the subject live in. This could also explain why there are huge outlier in the data as well where people with lower BMI tend to live longer.

Task 2 (1 mark)

Coefficients for multiple linear regression (on training data from Task 1):
w_{0}= **61.075**
w_{1}= **0.154**
w_{2}= **0.690**
w_{3}= **-0.005**

```
w[0] (intercept term): [61.07529735]
w[1] (coefficient for BMI): [0.15446318]
w[2] (coefficient for Alcohol): [0.6899124]
w[3] (coefficient for GDP): [-0.00529543]
```

Insert SSE for model trained on BMI only: **60.578**

```
SSE for BMI on test data: 60.5782769144324
```

Insert SSE for model trained on BMI, Alcohol and GDP: **18.682**

```
SSE for combined features on test data: 18.682307039707897
```

Thinking back to Task 1: If you could only choose one feature, either 'BMI' or 'Alcohol', which one would you choose to make a prediction? Why? Explain your reasoning!

If I could choose only one feature, I would choose the BMI Feature to make a prediction. Based on the observed scatter plot above (from task 1), we can see that the data points are closer to the best fit line. This observation would indicate that there is a strong linear relationship. If the points are widely scattered around the line, it would be a weak linear relationship. We can also observe that the total number of outliers of the data points are noticeable fewer. Based on my reasoning above, we can conclude that BMI is a good feature to make a prediction.

If you wanted to predict 'GDP' from 'BMI' and 'Alcohol', what would you do? Explain your reasoning and the steps to take.

General Explanation:
1) I will first load my data (basically making the X array as we did previously) and then I would fit a linear regression model on BMI and Alcohol to obtain the coefficients for the new model
2) After getting the coefficients, I could use them to make predictions on new data.
3) I would then use the coefficients and use the predict () function on those values.
4) I would then compute the SSE of the new model to check the discrepancy between the data and the estimated model using the newly found coefficient values and the newly predicted values

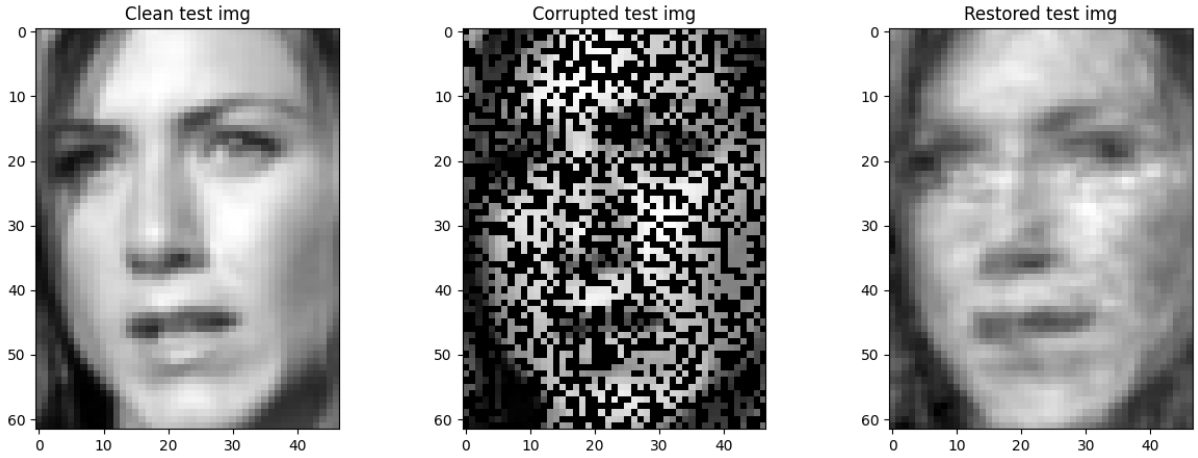
Python Code:

```
# This is the Python Code that I would write.

# Step 1: Concatenate BMI and Alcohol to form new X
X_new = np.column_stack((X_bmi, X_alc))

# Step 2: Use GDP as Y
Y_new = X_gdp

# Step 3: Obtain coefficients for new model
coefficients_new = linearRegression_cof(X_new, Y_new)

# Step 4: Generate predictions for new data
Y_pred_new = predict (X_new, coefficients_new)

# Step 5: Compute SSE for new model
sse_new = compute_sse(Y_new, Y_pred_new)

# Step 6: Printing the output
print("SSE for predicting GDP from BMI and Alcohol:", sse_new)
```

Task 3 (2 marks)

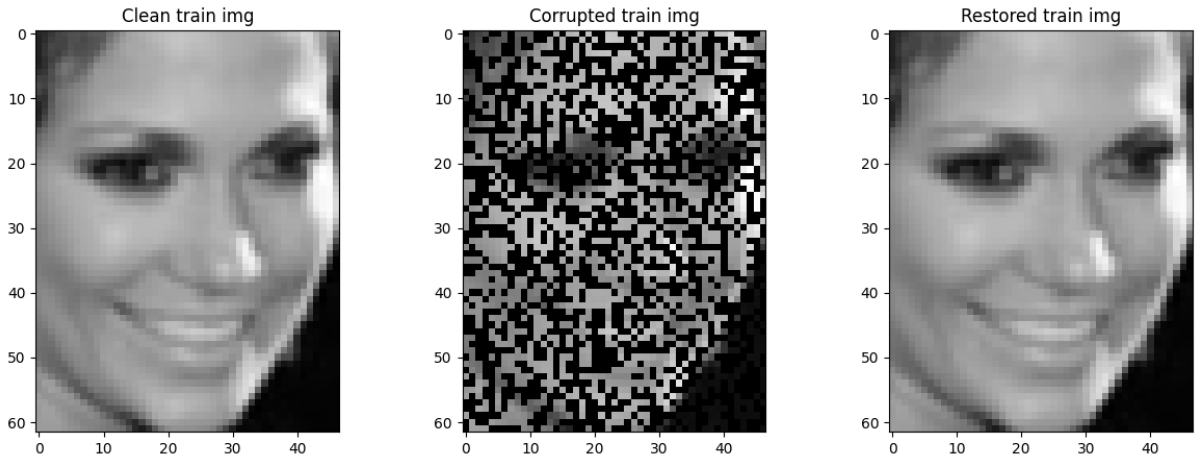
Display the reconstructed versions of the previously chosen corrupted test image (the 9th image of the test set, i.e., image[8]) alongside the original and corrupted one

Insert (Side-by-side plot of all three images: 'Clean test img', 'Corrupted test img', 'Restored test img'):



Display the reconstructed versions of the previously chosen corrupted train image (the 6th image of the training set, i.e., image[5]) alongside the original and corrupted one

Insert (Side-by-side plot of all three images: 'Clean train img', 'Corrupted train img', 'Restored train img':



What do you observe if you compare the image quality of the restored test image to the restored training image? Why does our regression model perform differently for these two sets?

The restored training image (Restored train img) looks better when compared to the restored test image (Restored test img). We can see that the resolution of the restored test image is not good and there is noise in the image. High level of noise and low resolution of the image will make the image more grainy and less detailed. This could

be due to the fact that the trained model has not been trained on the supposed test set data. This could explain why the picture looks so grainy.

Our regression model performs differently for the two sets which could be due to overfitting. Overfitting occurs when a model is trained too well on a particular set of data (training set). It will not perform well on the new unseen test set. The model would have learned the specific noise and details of the training set rather than the general pattern of the image.

Task 4 (2 marks)

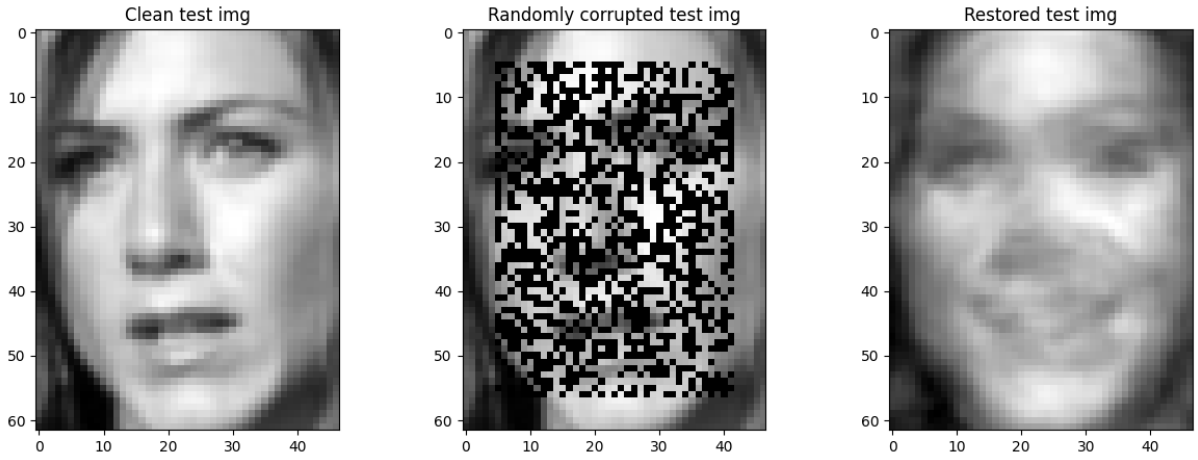
Calculate the PSNR for the 9th test image (i.e., image[8]) from Task 3: corrupted image and restored version.

PSNR for corrupted version of test_face_crpt[8]: **8.500**

PSNR for restored version of test_face_crpt[8]: **25.447**

```
Evaluating PSNR for test image with index 8 from Task 3.
PSNR for corrupted image: 8.499659422238917
PSNR for restored image: 25.447010972071613
```

Display the reconstruction results on the randomly corrupted test images. Use the same image index as previous task so you can directly compare the results:



Insert PSNR for corrupted version of test_face_crpt_rdm[8]: **9.443**

Insert PSNR for restored version of test_face_crpt_rdm[8]: **19.357**

```
Evaluating PSNR for test image with index 8 from Task 3.
PSNR for randomly corrupted image: 9.442796530924168
PSNR for restored image: 19.356665197245
```

Compare your PSNR and visual results from step 5 with the ones from step 2! Can you explain what might have changed that led to differences between the restored images, and why this happens?

By comparing the PSNR and visual results from step 5 and the ones from step 2, the changes that led to the differences between the restored images can be explained as follows:

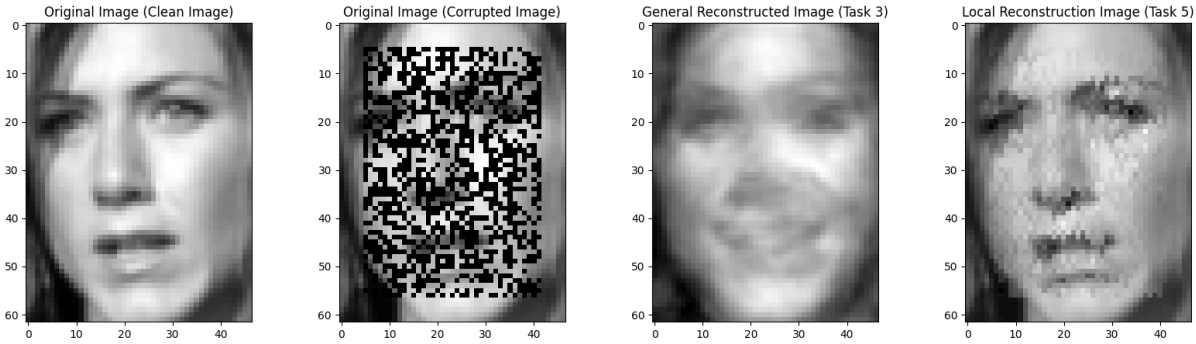
The PSNR (Peak Signal to Noise Ratio) is the measure of quality of an image or signal, comparing the maximum possible power of signal to the power of noise present. The higher the PSNR, the higher the quality image and less noise in the image.

As we have calculated above, the PSNR for the restored version of each of the test images will be higher compared to the corrupted version. We can also notice that after repeating the regression fit and prediction on our new data, the value of the PSNR will drop, meaning that the image quality has dropped. Based on the observation of the two images, we can conclude that to be true.

This could be due to the fact that linear regression is a relatively simple model. It is not able to capture the full complexity of the data, resulting in worsening of the data even if linear regression has been carried out twice. The model is just not properly trained on different test sets which would reduce its ability to generalize new data.

Task 5 (2 marks)

Insert visualization of all approaches. You should display all 4 images: The clean one (original image); The randomly corrupted one (original image); General reconstruction (from task 3); Local reconstruction (from task 5):



Compare the results you achieve with this local approach. What do you observe when you look at the reconstruction quality in the top half vs. the one in the lower half? Why is this the case?

When comparing the results that I have achieved with the local approach, I have observed that the reconstruction quality in the top half is significantly better than the reconstruction quality in the lower half. As we can see from the above, the image is appearing to be smiling when the original image is not. This could be due to the fact that what we have done is a raster scan, meaning that we are scanning the image from top to bottom. The algorithm will start to reconstruct the pixels from the top. As the reconstruction process moves forward (downwards), we start to rely more on the pixel that has been reconstructed. If the reconstruction is not perfect, it will lead to small errors which will accumulate quickly into larger errors when we reached the bottom of the image. Error propagation would occur.

(Optional): Insert images of further image quality improvement via averaging:

(Optional): Check PSNR of new approaches vs. previous ones:

Evaluating Images with idx8 from the test data set.

PSNR for test_face_crpt[8] w/ global: **9.443**

PSNR for test_face_crpt_rdm[8] w/ global: **19.357**

```
PSNR for randomly corrupted image: 9.442796530924168
PSNR for restored image: 19.356665197245
```

PSNR for test_face_crpt_rdm[8] w/ local: **26.983**

```
PSNR for local model: 26.982551455424755
```

PSNR for test_face_crpt_rdm[8] w/ local & filter: **28.162**

```
PSNR for filtered image: 28.162079641473653
```

Code for Task 1:

Paste your code with unanswered questions and comments in here.

```
# As always, we first import several libraries that will be helpful to solve the tasks
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
```

[1] ✓ 0.6s Python

Loading the data

```
# === Task1 ===
# Alcohol data: task1_alc.npy
# BMI data:     task1_bmi.npy
# GDP data:     task1_gdp.npy
# Life expect.: task1_lifeexpt.npy

# Load data from drive
X_alc = np.load('data/task1_alc.npy')
X_bmi = np.load('data/task1_bmi.npy')
X_gdp = np.load('data/task1_gdp.npy')
Y = np.load('data/task1_lifeexpt.npy')
```

✓ 0.0s Python

1. Regression function

```
def linearRegression_cof(X, Y):

    # You should write this linearRegression_cof function based on the multivariate case, so you can use it for later tasks (ie. tasks 2-3)
    # Inputs: X - For a univariate case, this will be m * 1, for multivariate, this will be m * n
    #         Y - For a univariate case, this will be m * 1, for multivariate, this will be m * p
    # Output: The coefficients of the linear regression model

    # This function accepts matrices X and Y as inputs, and returns the coefficients of the linear regression model.
    # The function begins by adding a column of ones to X, to account for the intercept term.
    # Then it calculates X^T * X, takes its inverse, and finally calculates the coefficients using the formula above.
    # Note: The numpy dot function is used for matrix multiplication, and linalg.inv is used to calculate the inverse of a matrix.
    # This function assumes that X and Y are numpy arrays. For the univariate case, X should be an array of shape (m, 1),
    # where m is the number of observations. For the multivariate case,
    # X can be an array of shape (m, n), where n is the number of features.
    # Similarly, Y should be an array of shape (m, 1) for the univariate case, and (m, p) for the multivariate case,
    #  where p is the number of dependent variables.
    #
    # Remember to check the conditions and assumptions for applying linear regression.
    # For instance, the relationship between the variables should be linear,
    # there should be no multicollinearity, and the errors should be normally distributed and have constant variance (homoscedasticity).

    # Add a column of ones for the intercept term
    X = np.concatenate([np.ones((X.shape[0], 1)), X], axis=1)

    # Calculate X^T * X
    XtX = np.dot(X.T, X)

    # Calculate the inverse of X^T * X
    XtX_inv = np.linalg.inv(XtX)

    # Calculate X^T * Y
    XtY = np.dot(X.T, Y)

    # Calculate the coefficients β
    beta = np.dot(XtX_inv, XtY)

    # returning the value
    return beta
```

✓ 0.0s Python

2. Obtaining regression coefficients for *Alcohol*

Hint: You need to ensure you add in a bias term, we won't remind you this everytime!

```
## Add in your code here
# Load data, already loaded from above
# Btw the hint is already accounted for in the created function above
# The bias term is the intercept of the line or hyperplane that the model is learning.
# In the equation y = mx + b, b is the bias term. This term allows the line to fit the data better by shifting it up or down.

# When setting up linear regression, need to remember to include this bias term in our model.
# In other words, it's important not to just learn a line that goes through the origin (0,0), but a line that can intersect the y-axis at any possible value,
# which usually provides a better fit to the data.

# Obtain coefficients
coefficients_alcohol = linearRegression_cof(X_alc.reshape(-1, 1), Y)

# Print coefficients
print("w[0] (intercept term):", coefficients_alcohol[0])
print("w[1] (coefficient for Alcohol):", coefficients_alcohol[1])
```

[4] ✓ 0.0s Python

```
w[0] (intercept term): [63.13011473]
w[1] (coefficient for Alcohol): [0.96682896]
```

3. Obtaining regression coefficients for *BMI*

```
## Add in your code here
# Obtain coefficients
coefficients_BMI = linearRegression_cof(X_bmi.reshape(-1, 1), Y)

# Print coefficients
print("w[0] (intercept term):", coefficients_BMI[0])
print("w[1] (coefficient for Alcohol):", coefficients_BMI[1])
```

[5] ✓ 0.0s Python

```
w[0] (intercept term): [60.74884974]
w[1] (coefficient for Alcohol): [0.19073734]
```

Hint: If you wrote the function correctly, your answer for the *BMI* data should be close to: $w_0 = 60.749, w_1 = 0.191$

4. & 5. Predicting the Life Expectancy

Write a function **predict()**, which takes a data point $\boldsymbol{x}_q$ and coefficients $\boldsymbol{w}$ and return the prediction $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}_q$.

```
def predict(X, w):

    # This is for a univariate case
    # Inputs: x - input data
    #         w - Coefficients for the linear regression model
    # Output: The predicted values based on x and w

    # Check if X is a one-dimensional array and reshape it to 2D if it is
    if len(X.shape) == 1:
        X = X.reshape(-1, 1)

    # Augment X with a constant feature 1
    X_aug = np.concatenate([np.ones((X.shape[0], 1)), X], axis=1)

    # Form y_hat by performing dot product operation between w and X_aug
    y_hat = np.dot(X_aug, w)

    # Returning back the value
    return y_hat
```

[6] ✓ 0.0s Python

Now, we will use this **predict()** function to plot Y = 'Life expectancy' against the predictor X = 'Alcohol' and 'BMI'. You are asked to display both the original data as a scatter plot and your predicted best fit line.

```
# Display the scatter results
# Generate predictions
Y_pred_alc = predict(X_alc, coefficients_alcohol)
Y_pred_bmi = predict(X_bmi, coefficients_BMI)

# Create a figure with two subplots
fig, ax = plt.subplots(2, 1, figsize=(10, 10))

# Scatter plot for Alcohol
ax[0].scatter(X_alc, Y, color='blue', label='Original data')
ax[0].plot(X_alc, Y_pred_alc, color='red', label='Best fit line')
ax[0].set_xlabel('Predictor X: Alcohol')
ax[0].set_ylabel('Life expectancy')
ax[0].legend()
ax[0].set_title('Life expectancy against Alcohol')

# Scatter plot for BMI
ax[1].scatter(X_bmi, Y, color='blue', label='Original data')
ax[1].plot(X_bmi, Y_pred_bmi, color='red', label='Best fit line')
ax[1].set_xlabel('Predictor X: BMI')
ax[1].set_ylabel('Life expectancy')
ax[1].legend()
ax[1].set_title('Life expectancy against BMI')

# Show the plot
plt.tight_layout()
plt.show()
```

Code for Task 2:

Paste your code with unanswered questions and comments in here.

1. Obtaining coefficients for multiple linear regression (on training data from Task 1)

Hint: Don't forget to add in the bias term!

```
# Extending the linear regression model to fit BMI, Alcohol, and GDP is
# as simple as concatenating these features to form a new X matrix,
# and then calling the linearRegression_cof function with this new X and Y as arguments.

# Concatenate the features
X_combined = np.column_stack((X_bmi, X_alc, X_gdp))

# Obtain the coefficients
coefficients_combined = linearRegression_cof(X_combined, Y)

# Print the coefficients
print("w[0] (intercept term):", coefficients_combined[0])
print("w[1] (coefficient for BMI):", coefficients_combined[1])
print("w[2] (coefficient for Alcohol):", coefficients_combined[2])
print("w[3] (coefficient for GDP):", coefficients_combined[3])
```

✓ 0.0s Python

```
w[0] (intercept term): [61.07529735]
w[1] (coefficient for BMI): [0.15446318]
w[2] (coefficient for Alcohol): [0.6899124]
w[3] (coefficient for GDP): [-0.00529543]
```

2. Computing the Error to evaluate the performance of the linear regression models

```
#Compute the sum squared error
def compute_sse(y_estimate, y):

    # Inputs: y_estimate is your estimated y from your linear regression model
    #         y is your actual y
    # Output: Return your sum squared error

    # Compute the differences between the actual and estimated values
    diff = y - y_estimate

    # Compute the square of the differences
    diff_squared = diff ** 2

    # Sum up the squared differences to compute the SSE
    sse = np.sum(diff_squared)

    # Returning the value
    return sse
```

[9] ✓ 0.0s Python

3. Evaluating on unseen test data

```
# Load the test data for this task: Australia, year 2002
# Test BMI data:     task2_AU2002_test_bmi.npy
# Test Alcohol data: task2_AU2002_test_alc.npy
# Test GDP data:     task2_AU2002_test_gdp.npy
# Test Life expect.: task2_AU2002_test_lifeexpt.npy

# Important: Be careful NOT to confuse your training and test data!
# Training data is for fitting the model, test data for evaluation only!

# Load the test data from drive
X_bmi_test = np.load('data/task2_AU2002_test_bmi.npy')
X_alc_test = np.load('data/task2_AU2002_test_alc.npy')
X_gdp_test = np.load('data/task2_AU2002_test_gdp.npy')
Y_test = np.load('data/task2_AU2002_test_lifeexpt.npy')
```

[10] ✓ 0.0s Python

a) L_{SE} when trained on X = 'BMI' (Coeficients from Task 1)

```
# Show your SSE value
# Generate predictions for BMI test data
Y_pred_bmi_test = predict(X_bmi_test, coefficients_BMI)

# Compute SSE for BMI
sse_bmi_test = compute_sse(Y_test, Y_pred_bmi_test)

print("SSE for BMI on test data:", sse_bmi_test)
```

[11] ✓ 0.0s Python

```
SSE for BMI on test data: 60.5782769144324
```

b) L_{SE} when trained on X_1 = 'BMI', X_2 = 'Alcohol' and X_3 = 'GDP' (Coefficients from Task 1)

```
# Show your SSE value
# Concatenate test features
X_test_combined = np.column_stack((X_bmi_test, X_alc_test, X_gdp_test))

# Generate predictions for combined test data
Y_pred_test_combined = predict(X_test_combined, coefficients_combined)

# Compute SSE for combined features
sse_test_combined = compute_sse(Y_test, Y_pred_test_combined)

print("SSE for combined features on test data:", sse_test_combined)
```

[12] ✓ 0.0s Python

```
SSE for combined features on test data: 18.682307039707897
```

Code for Task 3:

Paste your code with unanswered questions and comments in here.

1. Loading and displaying data

```
# ===Task3===
# Training set: Original images:    train_face_clean.npy
# Training set: Corrupted images:   train_face_crpt.npy
# Test set: Original images:        test_face_clean.npy
# Test set: Corrupted images:       test_face_crpt.npy

# Load data from drive
train_imgs_crpt = np.load('data/train_face_crpt.npy')
train_imgs_clean = np.load('data/train_face_clean.npy')
test_imgs_crpt = np.load('data/test_face_crpt.npy')
test_imgs_clean = np.load('data/test_face_clean.npy')

# Lets see what the corrupted training data looks like
# --> Visualise e.g. img[5] of the corrupted train and img[8] of the corrupted test set (but feel free to check others as well)
fig, (ax0, ax1 )= plt.subplots(1,2, figsize = (10,20))
ax0.imshow(train_imgs_crpt[5], cmap="gray")
ax0.title.set_text("Corrupted training data")
ax1.imshow(test_imgs_crpt[8], cmap="gray")
ax1.title.set_text("Corrupted test data")

n_samples, h, w = train_imgs_crpt.shape
print(f'We have a total of {n_samples} training images, each with height {h}px and width {w}px.')
```

[13] ✓ 0.2s Python

```
We have a total of 185 training images, each with height 62px and width 47px.
```

2. Fitting the regression model / Determining the parameters

```
# Implement regression_fit()
def linearRegression_cof(X, y, alpha=0.1):
    """
    This function computes the coefficients of the multivariate linear regression model using the normal equation.
    X: np.array of shape (N, D) - features (corrupted pixels)
    y: np.array of shape (N,) - targets (clean pixels)
    Returns:
    theta: np.array of shape (D,) - the computed coefficients
    """
    # Append a column of ones for the bias term
    X = np.hstack([np.ones((X.shape[0], 1)), X])

    # # Compute the normal equation
    # theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

    # Compute the ridge regression (normal equation with regularization)
    theta = np.linalg.inv(X.T.dot(X) + alpha * np.identity(X.shape[1])).dot(X.T).dot(y)

    # Returning the value of theta
    return theta
```

```
def regression_fit_vectorized(samples_X, samples_Y):
    """
    This function fits a linear regression model to the data, using the normal equation method.
    samples_X: np.array of shape (N, D) - input features
    samples_Y: np.array of shape (N, P) - target features
    Returns:
    reg_cof: np.array of shape (D+1, P) - regression coefficients for each pixel
    """
    # Add a column of ones for the bias term
    X = np.hstack([np.ones((samples_X.shape[0], 1)), samples_X])

    # Compute the normal equation in a vectorized manner
    reg_cof = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(samples_Y)

    return reg_cof
```

[14] ✓ 0.0s Python

```
# Retrieve the regression parameters
# reg_cof = regression_fit(train_imgs_crpt, train_imgs_clean)
# Reshape the training images to 2D arrays
train_imgs_crpt_2d = train_imgs_crpt.reshape((n_samples, h * w))
train_imgs_clean_2d = train_imgs_clean.reshape((n_samples, h * w))

# Compute the regression coefficients
reg_cof = regression_fit_vectorized(train_imgs_crpt_2d, train_imgs_clean_2d)
```

[15] ✓ 8.9s Python

+ Code + Markdown

3. Completing the corrupted images of the unseen test data by using linear regression to predict the missing pixels

```
# # Predict the missing pixels for all test images
# def predict(samples, reg_cof):

#     # Remember, we are dealing with the case where we have multivariate inputs AND outputs (so each pixel has its own output value)

#     # input arguments:
#     #    samples: np.array with image data, shape (N, h, w)
#     #    reg_cof: regression coefficients / parameters
#     # output:
#     #    recon_samples: np.array with image data, shape (N, h, w)

#     # Flatten each image in the samples to make them 1D
#     samples_flat = samples.reshape((samples.shape[0], -1))

#     # Add a bias term to the flattened samples
#     samples_flat_aug = np.hstack((np.ones((samples_flat.shape[0], 1)), samples_flat))

#     # Predict the missing pixels using the regression coefficients
#     recon_samples_flat = np.dot(samples_flat_aug, reg_cof)

#     # Reshape the predicted samples back to the original image dimensions
#     recon_samples_res = recon_samples_flat.reshape(samples.shape)

#     return recon_samples_res
```

```
def predict_vectorized(samples, reg_cof):
    """
    This function applies the regression model to the corrupted images to predict the missing pixels.
    samples: np.array of shape (N, h, w) - corrupted images
    reg_cof: list of np.array of shape (D+1,) - regression coefficients for each pixel
    Returns:
    recon_samples: np.array of shape (N, h, w) - reconstructed images
    """
    # Add a column of ones for the bias term
    samples_2d = np.hstack([np.ones((samples.shape[0], 1)), samples])

    # Compute the reconstructed samples using the regression model
    recon_samples_2d = samples_2d.dot(reg_cof)

    # Reshape the reconstructed samples to the original shape
    recon_samples = recon_samples_2d.reshape(samples.shape)

    return recon_samples
```

[16] ✓ 0.0s Python

```
# Predict / Restore images of the test set
test_imgs_crpt_2d = test_imgs_crpt.reshape((-1, h * w))
test_imgs_recon_2d = predict_vectorized(test_imgs_crpt_2d, reg_cof)
test_imgs_recon = test_imgs_recon_2d.reshape(test_imgs_crpt.shape)

# Display the reconstructed versions of the previously chosen corrupted test image alongside the original and corrupted one
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(test_imgs_clean[8], cmap="gray")
ax1.title.set_text('Clean test img')
ax2.imshow(test_imgs_crpt[8], cmap="gray")
ax2.title.set_text('Corrupted test img')
ax3.imshow(test_imgs_recon[8], cmap="gray")
ax3.title.set_text('Restored test img')
plt.show()
```

4. Completing the corrupted training images via linear regression

```
# Predict / Restore images of the training set
train_imgs_crpt_2d = train_imgs_crpt.reshape((-1, h * w))
train_imgs_recon_2d = predict_vectorized(train_imgs_crpt_2d, reg_cof)
train_imgs_recon = train_imgs_recon_2d.reshape(train_imgs_crpt.shape)

# Display the reconstructed versions of the previously chosen corrupted training image alongside the original and corrupted one
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(train_imgs_clean[5], cmap="gray")
ax1.title.set_text('Clean train img')
ax2.imshow(train_imgs_crpt[5], cmap="gray")
ax2.title.set_text('Corrupted train img')
ax3.imshow(train_imgs_recon[5], cmap="gray")
ax3.title.set_text('Restored train img')
plt.show()
```

[18] ✓ 0.2s Python

Code for Task 4:

Paste your code with unanswered questions and comments in here.

1. PSNR Implementation

```
## Implement PSNR to measure quantitative difference
# Another way of doing PSNR
def PSNR(img1, img2):

    # Inputs: img1 - corrupted / reconstructed image (depending on what you are calculating)
    #         img2 - clean image
    # Output: The calculated PSNR value
    # img1 = img1.astype(float)
    # img2 = img2.astype(float)
    # Making sure that
    img1 = np.around(img1).astype(int)
    img2 = np.around(img2).astype(int)
    # img1 = img1.astype(int)
    # img2 = img2.astype(int)

    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return 100
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse))

    return psnr
```

✓ 0.0s Python

2. Calculate the PSNR for the 9th test image (i.e., image[8]) from Task 3: corrupted image and restored version

```
img_idx_test = 8
print(f'Evaluating PSNR for test image with index {img_idx_test} from Task 3.')

## Print both PSNRs for corrupted and reconstructed versions
psnr_corrupted = PSNR(test_imgs_clean[img_idx_test], test_imgs_crpt[img_idx_test])
psnr_restored = PSNR(test_imgs_clean[img_idx_test], test_imgs_recon[img_idx_test])

print(f'PSNR for corrupted image: {psnr_corrupted}')
print(f'PSNR for restored image: {psnr_restored}')
```

[20] ✓ 0.0s Python

```
Evaluating PSNR for test image with index 8 from Task 3.
PSNR for corrupted image: 8.500059687176659
PSNR for restored image: 25.45142483179692
```

3. Load new data corrupted with random patterns

If you closely inspect the corrupted data and compare it to the one from the previous task, you will notice that apart from the obvious uncorrupted 'frame' we added, the corruption pattern now differs from image to image!

```
# Load data from drive
# Train images corrupted with random pattern: train_face_crpt_rdm.npy
# Test  images corrupted with random pattern: test_face_crpt_rdm.npy
train_imgs_crpt_rdm = np.load('data/train_face_crpt_rdm.npy')
test_imgs_crpt_rdm = np.load('data/test_face_crpt_rdm.npy')

# Lets see what the corrupted training data looks like
# --> Visualise e.g. img[5] of the corrupted train and img[8] of the corrupted test set (but feel free to check others as well)
fig, (ax0, ax1 )= plt.subplots(1,2, figsize = (10,10))
ax0.imshow(train_imgs_crpt_rdm[5], cmap="gray")
ax0.title.set_text("Rdm. crpt. train data")
ax1.imshow(test_imgs_crpt_rdm[8], cmap="gray")
ax1.title.set_text("Rdm. crpt. test data")

n_samples, h, w = train_imgs_crpt_rdm.shape
print(f'We have a total of {n_samples} training images, each with height {h}px and width {w}px.')
```

[21] ✓ 0.2s Python

```
We have a total of 185 training images, each with height 62px and width 47px.
```

4. Repeat regression fit and prediction on new data

```
# Retrieve the regression coefficients to reconstruct images from corrupted ones
# Reshape the data
train_imgs_crpt_rdm_2d = train_imgs_crpt_rdm.reshape((-1, h * w))

# Retrieve the regression coefficients to reconstruct images from corrupted ones
reg_cof_rdm = regression_fit_vectorized(train_imgs_crpt_rdm_2d, train_imgs_clean_2d)
```

[22] ✓ 11.8s Python

```
# Predict the missing pixels for all test images
test_imgs_crpt_rdm_2d = test_imgs_crpt_rdm.reshape((-1, h * w))
test_imgs_recon_rdm_2d = predict_vectorized(test_imgs_crpt_rdm_2d, reg_cof_rdm)
test_imgs_recon_rdm = test_imgs_recon_rdm_2d.reshape(test_imgs_crpt_rdm.shape)
```

[23] ✓ 0.0s Python

```
# Display the reconstruction results on the randomly corrupted test images
# Use the same image index as previous task so you can directly compare the results

img_idx = 8
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(test_imgs_clean[img_idx_test], cmap="gray")
ax1.title.set_text('Clean test img')
ax2.imshow(test_imgs_crpt_rdm[img_idx_test], cmap="gray")
ax2.title.set_text('Randomly corrupted test img')
ax3.imshow(test_imgs_recon_rdm[img_idx_test], cmap="gray")
ax3.title.set_text('Restored test img')
plt.show()
```

[24] ✓ 0.4s Python

4. Calculate the PSNR for the 9th test image (i.e., image[8]): corrupted image and restored version

```
[25]  img_idx_test = 8
      ✓ 0.0s                                                                    Python
```

```
[26]  print(f'Evaluating PSNR for test image with index {img_idx_test} from Task 3.')

      ## Show the two PSNR values for both corrupted and restored versions (for the randomly corrupted image)
      psnr_corrupted_rdm = PSNR(test_imgs_clean[img_idx_test], test_imgs_crpt_rdm[img_idx_test])
      psnr_restored_rdm = PSNR(test_imgs_clean[img_idx_test], test_imgs_recon_rdm[img_idx_test])

      print(f'PSNR for randomly corrupted image: {psnr_corrupted_rdm}')
      print(f'PSNR for restored image: {psnr_restored_rdm}')
      ✓ 0.0s                                                                    Python
```

```
...  Evaluating PSNR for test image with index 8 from Task 3.
     PSNR for randomly corrupted image: 9.44395057070683
     PSNR for restored image: 19.357894781600926
```

Code for Task 5:

Paste your code with unanswered questions and comments in here.

1. Patch extraction

Create a function to extract a patch of size 'patch_size' (for us (5,11)) above the current pixel, symmetric to the left and right. A 5×11 patch should e.g. go 5 pixels above, 5 to the left and 5 to the right of the current pixel.

```
def extract_patch(img, row, col, patch_size):

    # Inputs: img - the actual image
    #         row - which row we are extracting from
    #         col - which col we are extracting from
    #         patch_size - the size of the patch that we want to extract
    # Output : The patch

    half_width = patch_size[1] // 2
    patch = img[row - patch_size[0]:row, col - half_width:col + half_width + 1]
    return patch
```

[27] ✓ 0.0s Python

2. Extract training patches for corrupted pixels

Theoretically, we could extract a patch for each individual pixel of all images (that have a sufficiently big neighbourhood). To reduce the use of data and speed up our computations, we will just use the first 10 images of the training set. We will further only extract patches for pixels that are currently corrupted in these training images. Since the corruption pattern is random, this should cover a sufficient number of different pixels.

There are different ways to extract the patches, the comments below highlight one way

```
# For all images, go through each individual image (rows and cols) and find all corrupted pixels (value == 0)
# Then, extract the 5x11 patch directly above the corrupted pixel from the same image of the uncorrupted training set,
# and add it to the new patch-based training set X;
# For the following prediction, also store the 'true' value of the pixel we want to reconstruct in the training labels set Y
patch_size = (5,11)

# Initialising the variables
X_patches = []
Y_patches = []

 # use the first 10 images
for i in range(10):
    img_crpt = train_imgs_crpt[i]
    img_clean = train_imgs_clean[i]
    for row in range(patch_size[0], h):
        for col in range(patch_size[1]//2, w - patch_size[1]//2):
            # corrupted pixel
            if img_crpt[row, col] == 0:
                patch = extract_patch(img_clean, row, col, patch_size)
                X_patches.append(patch.flatten())
                Y_patches.append(img_clean[row, col])
```

[28] ✓ 0.1s Python

```
# Add all the samples to an array (similar to the original training data)
X_patches = np.array(X_patches)
Y_patches = np.array(Y_patches)
```

[29] ✓ 0.0s Python

3. Fitting the regression model to the information of the image patches

```
# # Retrieve coefficients using our linear regression fit function from Task 3

### Testing using our own built in linear regression model
def new_linearRegression_cof(X, Y):

    # You should write this linearRegression_cof function based on the multivariate case, so you can use it for later tasks (ie. tasks 2-3)
    # Inputs: X - For a univariate case, this will be m * 1, for multivariate, this will be m * n
    #         Y - For a univariate case, this will be m * 1, for multivariate, this will be m * p
    # Output: The coefficients of the linear regression model

    num = np.matmul(X.T,X)
    num = np.linalg.pinv(num)
    rem = np.matmul(X.T,Y)
    W = np.matmul(num,rem)

    # returning the value
    return W
```

```python
def new_regression_fit(samples_X, samples_Y):
    # input arguments:
    #    samples_X: training samples
    #    samples_Y: training 'labels' (uncorrupted samples)
    # output:
    #    reg_cof: regression coefficients (parameters)

    # You may choose to call your linearRegression_cof function from task 1 if you already implemented a multivariate case

    # Reshaping my input samples
    samples_X = np.reshape(samples_X, (len(samples_X), -1))
    samples_Y = np.reshape(samples_Y, (len(samples_Y), -1))

    # Adding ones to X
    X = np.concatenate((samples_X, np.ones((samples_X.shape[0], 1))), axis = 1)

    # Calling the function
    reg_cof = new_linearRegression_cof(X,samples_Y)

    # Returning the values
    return reg_cof

# Retrieve the regression coefficients to reconstruct images from corrupted ones
reg_cof_patches = new_regression_fit(X_patches, Y_patches)

# ### JUST FOR CHECKING PURPOSES
# ### Testing using the built in linear regression model
# from sklearn.linear_model import LinearRegression

# # Create an instance of the LinearRegression class
# regressor = LinearRegression()

# # Fit the model to the training data
# regressor.fit(X_patches, Y_patches)
```

4. Prediction / Reconstruction for ONE image of your choice

Prediction function is slightly different, since we cannot just directly predict ALL y-values! To predict a y-value, we need to have access to the window of size 5x11 right above the pixel we want to reconstruct. Intuitively, this can only be done for the first row of corrupted pixels, but not further (since the windows for the 2nd row would include corrupted pixels of the first row)! --> To solve this problem, we will reconstruct the image pixel-by-pixel using the reconstructed results of row1 to reconstruct row 2, and so on.

```python
# def predict_vectorized(samples, reg_cof):
#     # Add the constant to each sample
#     samples_2d = np.hstack((samples, np.ones((samples.shape[0], 1))))

#     # Apply the linear regression
#     recon_samples_2d = samples_2d.dot(reg_cof)

#     return recon_samples_2d[0, 0]
def predict_vectorized(samples, reg_cof):
    # Add the constant to each sample
    samples_2d = np.hstack((samples, np.ones((samples.shape[0], 1))))

    # Apply the linear regression
    recon_samples = samples_2d.dot(reg_cof)

    return recon_samples[0]

# Let's predict for one specific test image!
img_idx_test = 8
img_crpt_rdm = test_imgs_crpt_rdm[img_idx_test].copy()  # use copy to avoid overwriting original

for row in range(patch_size[0], h):
    for col in range(patch_size[1]//2, w - patch_size[1]//2):
        if img_crpt_rdm[row, col] == 0:  # corrupted pixel
            patch = extract_patch(img_crpt_rdm, row, col, patch_size)
            img_crpt_rdm[row, col] = predict_vectorized(patch.flatten().reshape(1, -1), reg_cof_patches)
```

```
            img_crpt_rdm[row, col] = predict_vectorized(patch.flatten().reshape(1, -1), reg_cof_patches)
            # img_crpt_rdm[row, col] = predict_vectorized(patch.flatten().reshape(1, -1), reg_cof_patches)[0, 0]
            # img_crpt_rdm[row, col] = predict_vectorized(patch.flatten().reshape(1, -1), reg_cof_patches)
[31]  ✓ 0.0s                                                                                                Python
```

5. Visualisation of the results

```python
# Visualise all approaches
# You should display at 4 images:
# The clean one (original image)
# The randomly corrupted one (original image)
# General reconstruction (from task 3)
# Local reconstruction (from task 5)
fig, axs = plt.subplots(1, 4, figsize=(20, 5))

axs[0].imshow(test_imgs_clean[img_idx_test], cmap='gray')
axs[0].set_title('Original')

axs[1].imshow(test_imgs_crpt_rdm[img_idx_test], cmap='gray')
axs[1].set_title('Corrupted')

axs[2].imshow(test_imgs_recon_rdm[img_idx_test], cmap='gray')
axs[2].set_title('Restored (naive model)')

axs[3].imshow(img_crpt_rdm, cmap='gray')
axs[3].set_title('Restored (local model)')

plt.show()

psnr_naive = PSNR(test_imgs_clean[img_idx_test], test_imgs_recon_rdm[img_idx_test])
psnr_local = PSNR(test_imgs_clean[img_idx_test], img_crpt_rdm)

print(f'PSNR for naive model: {psnr_naive}')
print(f'PSNR for local model: {psnr_local}')
```

```
3]  ✓ 0.4s                                                                                                  Python
```

(Optional): Further image quality improvement via averaging

```python
from scipy import ndimage

# You can use the ndimage.uniform_filter to filter the results
#  Apply a mean filter over the reconstructed image and calculate the PSNR of the new reconstructed results
img_filtered = ndimage.uniform_filter(img_crpt_rdm, size=5)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].imshow(img_crpt_rdm, cmap='gray')
axs[0].set_title('Restored (local model)')

axs[1].imshow(img_filtered, cmap='gray')
axs[1].set_title('Filtered')

plt.show()

psnr_filtered = PSNR(test_imgs_clean[img_idx_test], img_filtered)
print(f'PSNR for filtered image: {psnr_filtered}')
```

[34] ✓ 0.2s Python