Assignment 3: AIML231

Linear Regression

a)

While conducting EDA I used mpg.shape, giving out (398,9) which means 398 rows and 9 columns of data. I also used mpg.info(), mpg.describe() and mpg_num.corr().

```
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    object
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

```
mpg.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

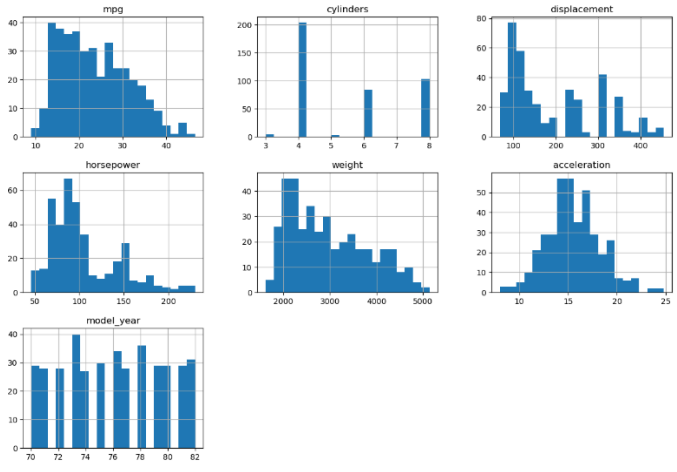
```
mpg_num = mpg[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year' ]]
mpg_num.corr()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

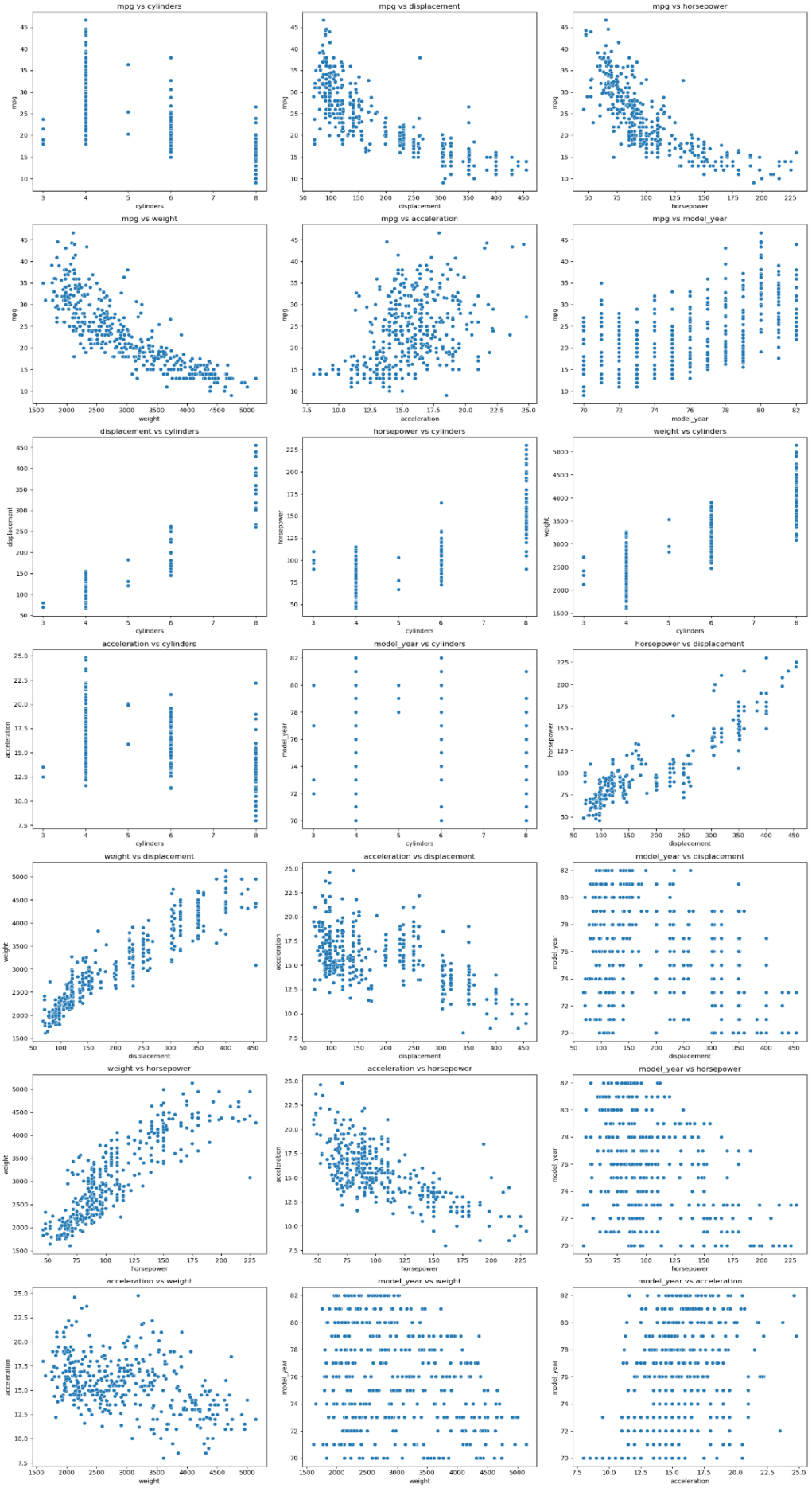
```
mpg.describe(include = object)
```

	origin	name
count	398	398
unique	3	305
top	usa	ford pinto
freq	249	6

From mpg.info(), we can see that out of the 9 features including the target class, 7 are numerical being float or integer and 2 are objects meaning categorical.
From mpg_num.corr(), we can see that the highest correlation of 0.951 (3dp) is between displacement and cylinders and the lowest correlation of 0.288 (3dp) is between model_year and acceleration.



From the histograms, we can see the frequency distribution of numeric features. Mpg, weight, displacement, and horsepower are positively skewed to the right. Acceleration appears to have a bell curve meaning it is a normal distribution and its symmetric. Model_year has a uniform distribution. Cylinders only has majority of its frequency being 4,6 and 8 cylinders.


mpg vs cylinders
mpg vs displacement
mpg vs horsepower
mpg vs weight
mpg vs acceleration
mpg vs model_year
displacement vs cylinders
horsepower vs cylinders
weight vs cylinders
acceleration vs cylinders
model_year vs cylinders
horsepower vs displacement
weight vs displacement
acceleration vs displacement
model_year vs displacement
weight vs horsepower
acceleration vs horsepower
model_year vs horsepower
acceleration vs weight
model_year vs weight
model_year vs acceleration

These are the scatterplots amongst all pair of features. Model_year vs acceleration has the least relationship as shown in the scatterplot. Other pairs of features with low amount of relationship involve model_year vs weight, model_year vs displacement, model_year vs cylinder, acceleration vs cylinder, and more. From this, these scatterplots show that in most cases, model_year has a low relationship with other pair of features except mpg vs model_year.

Although the highest correlation was displacement vs cylinders, it's hard to see the relationship between them on scatter plot as cylinders are mainly either 4,6 or 7 so data isn't spread out and there are not many varieties. Finding relationship with cylinder and other features would be hard to do with scatterplot hence why the odd-looking cylinder scatterplots.

Visually we can see that weight vs displacement and weight vs horsepower has a strong positive relationship. This makes sense as when the weight increases, you would need more horsepower. There are also negative relationships such MPG vs displacement , MPG vs horsepower and MPG vs weight.

b)

First thing I did was drop 'name' column as it was not needed at all as it has no affect on the other features and target class. The name feature is independent of all other features entirely as it just a name.

```
missing_values = mpg.isnull().sum()
missing_values
```

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
```

Using mgp.isnull().sum(), we can examine that horsepower feature has 6 missing values. Here I implemented Simpler Imputer with the strategy of mean. Although KNN Imputer captures complex relationship but, in this case, the missing data is a small portion of the dataset hence why I used Simple Imputer as its simple and fast. Also, Simple Imputer is computationally efficient because it involves straight forward calculations like mean, median or mode whereas KNN Imputer involves calculating distances between data points which can be computationally intensive and therefore slightly more expensive.
This is why I chose Simple Imputer. I also used the mean strategy as horsepower feature is numerical, making the mean a natural and logical choice for imputation. The mean provides a central value that can be used to fill in missing data without distorting the overall distribution of the feature.

Shown by the mpg.info(), 'name' and 'origin' were the categorical features but we dropped name and so 'origin' is now the only categorical feature . Origin contains the values, Europe, Japan, USA. I chose to use the OneHotEncoder because the categorical feature origin has no order hence why I did not use OrdinalEncoder. OneHotEncoder approach preserves the individual categories of each feature without imposing any ordinal relationship between them and no

assumptions are made. The OneHotEncoder transforms categorical data into a numerical format by creating binary features for each unique category. From this, we created origin_europe, origin_japan and origin_usa.

I also normalised I chose to use MinMaxScaler rather than StandardScaler as it assumes normal distribution and does not ensure all the features will have the same range (could go from negative infinity to positive infinity). MinMaxScaler gives a range from 0 to 1 and ensures features have the same range. From the histograms, we can see that most features did not have a normal distribution and horsepower feature does not have a normal distribution hence why I chose MinMaxScaler.

c)

```
Coefficients:
cylinders       : -3.072836268639777
displacement    : 10.869748953454174
horsepower      : -2.4049507930460234
weight          : -25.498848990341074
acceleration    : 2.0232128270447314
model_year      : 8.882519840545076
origin_europe   : 0.7457218778875097
origin_japan    : 1.1142010933778534
origin_usa      : -1.8599229712653205


Training R-squared: 0.8193391903369874
Training MSE      : 10.781501110591266
Test R-squared    : 0.8339234339104407
Test MSE          : 10.632428318782665
```

From the constructed linear regression model, these are the coefficients, R-squared and mean squared error (MSE). There is an increase in displacement is associated with an increase in MPG by approximately 10.87 units. Newer vehicles (later model years) are associated with an increase in MPG by approximately 8.88 units and heavier vehicles have significantly lower MPG, with each unit increase in weight decreasing MPG by approximately 25.50 units.

The R-squared values indicate that approximately 81.93% of the variance in MPG is explained by the model in the training set, and 83.39% of the variance is explained in the test set. These values suggest a good fit of the model to the data.

The mean squared error (MSE) for the training set is 10.78, and for the test set, it is 10.63.

Clustering

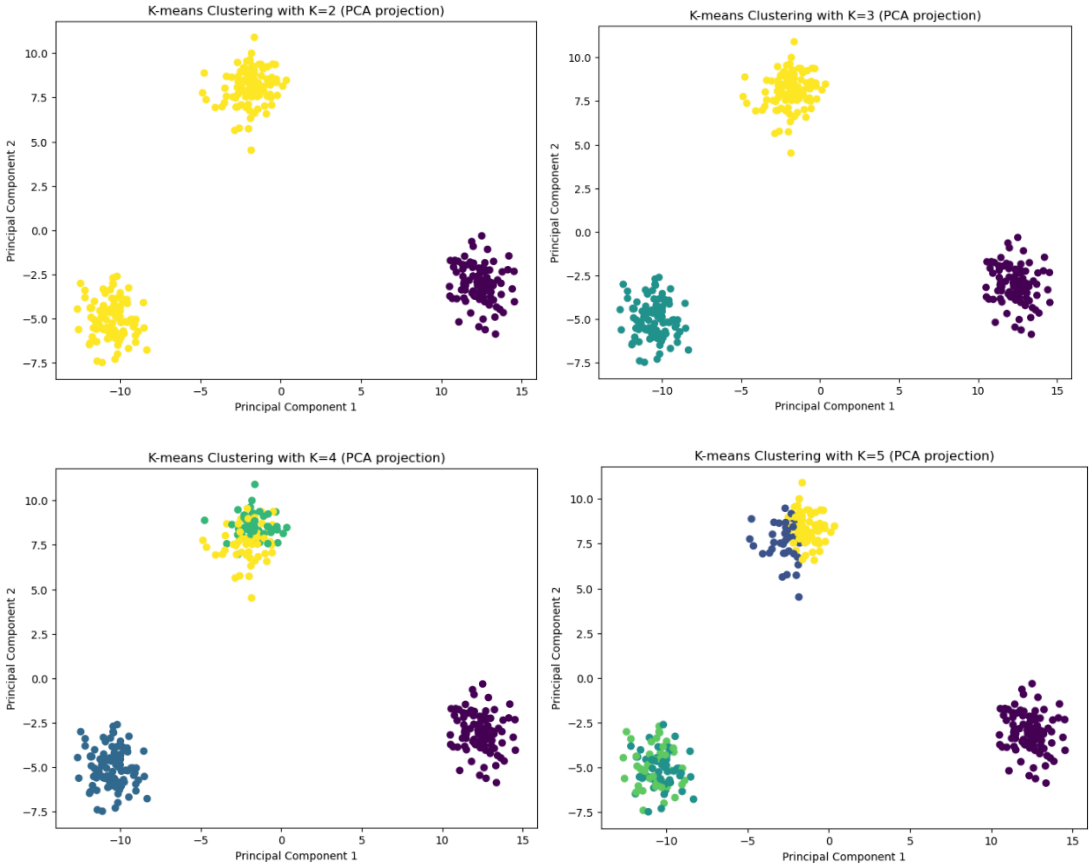
a)

```
When k = 2: silhouette score is 0.6515279371492652
When k = 3: silhouette score is 0.838307214726647
When k = 4: silhouette score is 0.6243095614369332
When k = 5: silhouette score is 0.42444835947679366
The best K based on silhouette score is: 3
```

When K=3, the silhouette score is the highest at 0.838307214726647. This indicates that clustering the dataset into 3 clusters results in the most well-defined and separated clusters compared to other values of K.

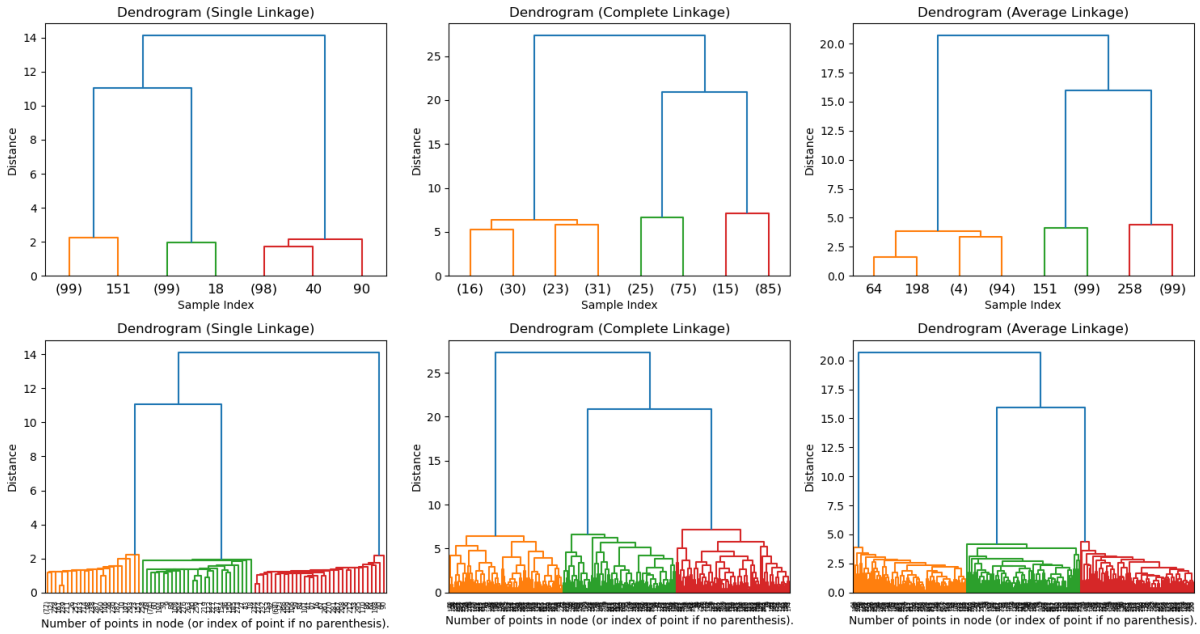
When K=5, the silhouette score is the lowest at 0.42444835947679366, indicating poorly defined clusters with significant overlap or incorrect assignments.

b)



When k = 3, the clusters are separated correctly with appropriate colour.

c)



Here I have provided dendrograms generated with different linkage methods for both low-dimensional (p = 2) and high-dimensional (p = 20) which provide more insight. The top 3 dendrograms are when p = 2 and the bottom 3 dendrograms are when p = 20. When p =20, we can see the dendrogram in a more detailed view of the clustering structure as there is increased complexity.

Agglomerative Clustering merges clusters based on distance between clusters defined by linkage methods. Single linkage computes the minimum pairwise dissimilarity where one observation is in cluster A and the other is in cluster B. Complete linkage computes the maximum pairwise dissimilarity where one observation is in cluster A and the other is in cluster B. Average linkage computes the average pairwise dissimilarity where one observation is in cluster A and the other is in cluster B. These are the different types of linkage shown above in the dendrograms above.

When looking at the dendrograms when p =20, complete and average linkage produced more balance dendrograms. Complete linkage tends to produce compact clusters and average linkage may result in clusters with varying levels of compactness. They are both visually similar. Although single linkage could be computationally efficient, it can produce trailing clusters in which single observation are merged one at a time leading to chaining, shown in the single linkage dendrogram above (p = 20). This makes it difficult to interpret and analyse the resulting clusters. Single linkage dendrogram also looks different to complete and average linkage dendrograms

d)

Advantages of hierarchical clustering:
With hierarchical clustering, you do not need to specify the K (the initial selection of cluster centroids) in advance whereas in K-means clustering you need to. If you have no information about the data, knowing what K to use can be hard.
Hierarchical clustering doesn't not need to re-run to obtain clustering with different number of clusters unlike K-means clustering where you need to.
Hierarchical clustering can handle numerical and categorical data whereas K-means is not suitable for categorical data.
Hierarchical clustering is an deterministic algorithm, meaning that given the same input data and parameters, it will produce the same clustering result every time it is run. Unlike K-means which is stochastic.

Disadvantages of hierarchical clustering:
Clustering obtained by cutting the dendrogram at certain height is necessarily nested within the clustering obtained by cutting at a greater height. This hierarchical structure might not always align with the natural grouping of the data. For example, if there are clear subgroups within a larger cluster, hierarchical clustering may not be able to capture these subgroups effectively.
Hierarchical clustering is also more computationally expensive given a large number of sample due to pair-wise distance unlike K-means.
K-means is more simple and flexible then hierarchal clustering and also can scale well with large number of samples and features.

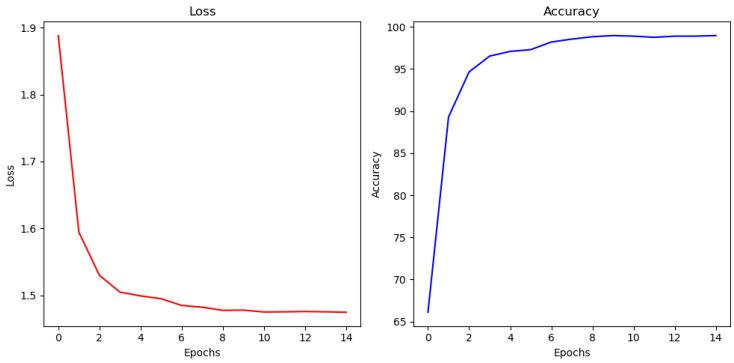
Neural Networks

a)

Below I used ReLU activation function, SoftMax activation function, Adam optimiser, batch size of thirty-two and learning rate of 0.01. Created class MLP(nn.Module) shown in the code.

b)

```
Epoch 1: Loss = 1.888, Accuracy = 66.11%
Epoch 2: Loss = 1.595, Accuracy = 89.28%
Epoch 3: Loss = 1.530, Accuracy = 94.64%
Epoch 4: Loss = 1.505, Accuracy = 96.52%
Epoch 5: Loss = 1.499, Accuracy = 97.08%
Epoch 6: Loss = 1.495, Accuracy = 97.29%
Epoch 7: Loss = 1.485, Accuracy = 98.19%
Epoch 8: Loss = 1.482, Accuracy = 98.54%
Epoch 9: Loss = 1.478, Accuracy = 98.82%
Epoch 10: Loss = 1.478, Accuracy = 98.96%
Epoch 11: Loss = 1.475, Accuracy = 98.89%
Epoch 12: Loss = 1.476, Accuracy = 98.75%
Epoch 13: Loss = 1.476, Accuracy = 98.89%
Epoch 14: Loss = 1.476, Accuracy = 98.89%
Epoch 15: Loss = 1.475, Accuracy = 98.96%
```



As we can see that for the first epoch, Accuracy is 66.11% and loss is 1.888. Then for second epoch, accuracy is 89.28% and loss is 1.596. Then from here onwards, the accuracy jumps to 94.64% and keeps on improving slightly until epoch 15. The loss also decreases slightly.
As we can see the graphs above, around epoch 6, the acccuracy doesn't increase much at all hence this could be early signs of overfitting where the model starts to memorize the training data instead of learning generalizable patterns. Also loss isn't decreasing much after 6 epoch aswell.

c)

```
Test Accuracy: 95.56%

Test Image 1: Predicted Label= 4, Actual Label= 4
Test Image 2: Predicted Label= 5, Actual Label= 5
Test Image 3: Predicted Label= 1, Actual Label= 1
Test Image 4: Predicted Label= 4, Actual Label= 4
Test Image 5: Predicted Label= 1, Actual Label= 1
```

The test accuracy of 95.56% is very high. When training, the 15th epoch's accuracy was 98.96%. This difference between the test and train accuracy is very minimal which indicates that the model is not overfitting significantly. Also, this is proven by the five examples of test image all being classified correctly (the predicted label equal the actual label).

d)

Using different activation functions.
ReLU Testing Accuracy: 95.56%
Sigmoid Testing Accuracy: 94.17%
Tanh Testing Accuracy: 95.56%

ReLU function and Tanh function have the highest test accuracy of 95.56% and are the same.

While Sigmoid has the lowest test accuracy of 94.17%. They are all similar and they all took the same amount of time to train the model.

ReLU is known for its effectiveness in deep neural networks due to its ability to address the vanishing gradient problem and accelerate convergence.
Sigmoid often used in binary classification tasks where the output needs to be interpreted as probabilities. Maybe that's why it got the lowest test accuracy.
Tanh is better at capturing non-linear relationships compared to Sigmoid, but still suffers from the vanishing gradient problem in deep networks.

But because all the test accuracies were all very high and similar, even with different activation functions, this could mean the neural network might be robust and can perform well with various activation functions. The hyperparameters such as learning rate, batch size and number of epochs result in high performance either way. This means that the model is trained for an adequate number of epochs with a good optimization (Adam). This could also be due to the problem (image classification - digits) being relatively simple for the model as well.

e)
More hidden layers or increased number of neurons in a layer allows the network to learn more complex representation and features from the data hence improving the model's ability to capture patterns and accuracy.
But more layers and increasing the numbers of neurons could lead to overfitting especially if the training data is limited. Regularization techniques can help reduce overfitting.
Adding more layers increases the computational complexity and memory usage and therefore causes longer training times and require more computation resources. This all would lead to the model being more expensive.
The effect of changing the number of neurons can depend on the specific layer. For example, increasing neurons in initial layers might help learn more low-level features, while increasing neurons in deeper layers might help learn more abstract, high-level features.
When thinking about changing the number of neurons, this could mean decreasing the number of neurons. If this happens, the network might lose the ability to learn complex representations and capture patterns. Accuracy could decrease too. Also, the decrease in number of neurons could lead to underfitting.