Assignment - 3 MultiLayer Perceptron and AutoEncoders

2. Multi Layer Perceptron

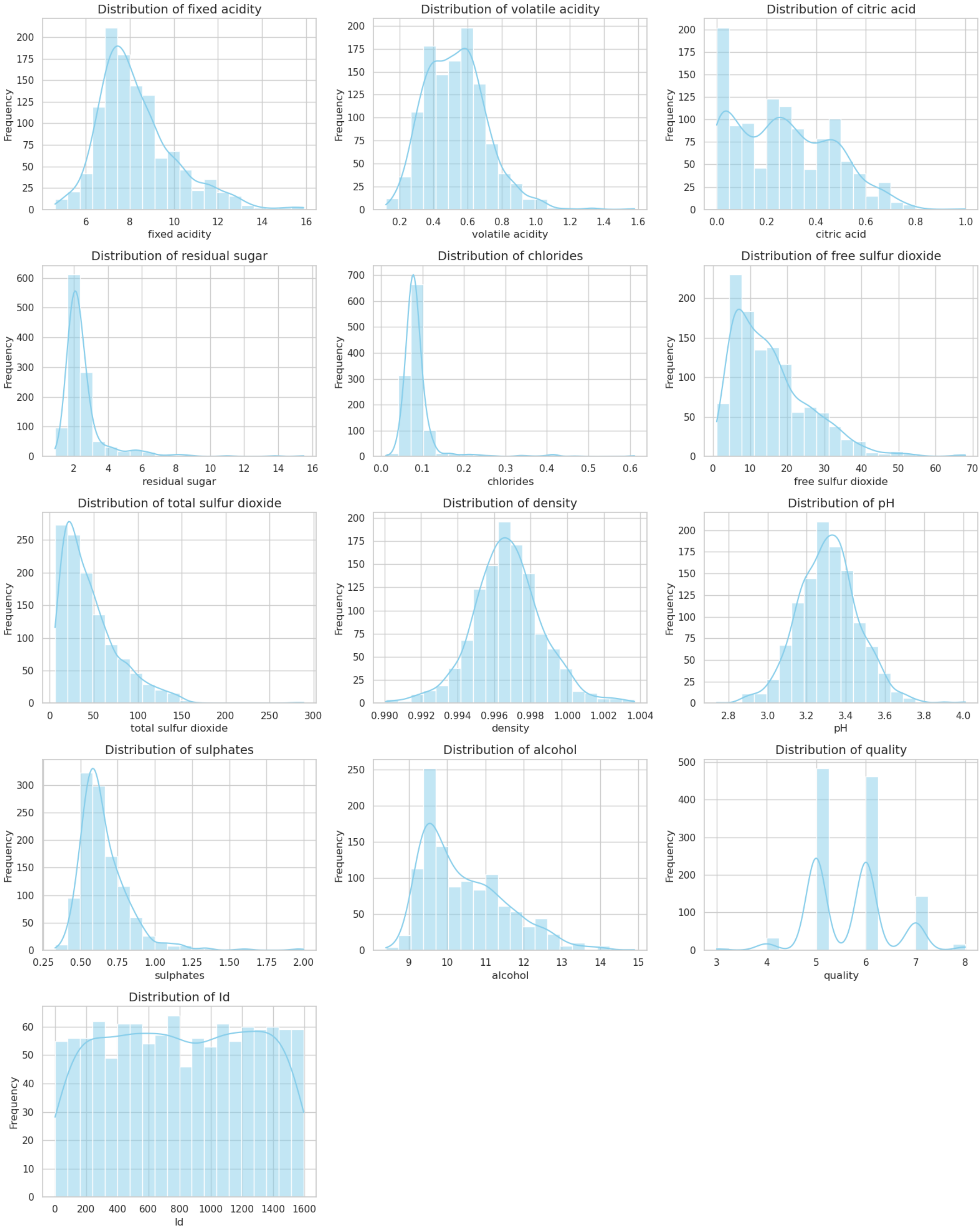
Dataset Analysis and Preprocessing

2.1.1

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.0000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311015	0.657708
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664	0.170399
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000	0.550000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000	0.620000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000	0.730000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000	2.000000

2.1.2



2.1.3

Used Label Encoder and Standard Scaler for the data and also handled missing data by filing the mean.

2.2 Model Building From Scratch

BackProp Details

Cost Function Derivative for backprop

$$J\left(a^{[3]} \mid y\right)=-y \log \left(a^{[3]}\right)-(1-y) \log \left(1-a^{[3]}\right)$$

BackProp Gradient equations

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{W}^{[3]}} & =\frac{d J}{d a^{[3]}} \frac{d a^{[3]}}{d z^{[3]}} \frac{\partial z^{[3]}}{\partial \mathbf{W}^{[3]}} \ \frac{\partial J}{\partial b^{[3]}} & =\frac{d J}{d a^{[3]}} \frac{d a^{[3]}}{d z^{[3]}} \frac{\partial z^{[3]}}{\partial b^{[3]}}\end{aligned}$$

Example of a three layered multi layer perceptron neural network.

Third Layer $$ \begin{gathered} d \mathbf{W}^{[3]}=\delta^{[3]} a^{[2] T} \ d b^{[3]}=\delta^{[3]} \ \delta^{[3]}=a^{[3]}-y \end{gathered} $$

Second Layer $$ \begin{gathered} d \mathbf{W}^{[2]}=\boldsymbol{\delta}^{[2]} \mathbf{a}^{[1] T} \ d \mathbf{b}^{[2]}=\boldsymbol{\delta}^{[2]} \ \boldsymbol{\delta}^{[2]}=\mathbf{W}^{[3] T} \delta^{[3]} * g^{\prime}\left(\mathbf{z}^{[2]}\right) \end{gathered} $$

First Layer $$ \begin{gathered} d \mathbf{W}^{[1]}=\boldsymbol{\delta}^{[1]} \mathbf{x}^T \ d \mathbf{b}^{[1]}=\boldsymbol{\delta}^{[1]} \ \boldsymbol{\delta}^{[1]}=\mathbf{W}^{[2] T} \delta^{[2]} * g^{\prime}\left(\mathbf{z}^{[1]}\right) \end{gathered} $$

Another important thing to note is that the output is a softmax function becuase we need to output probabilities of and then classify to the one which has the highest probabilities and the sum of all these should be 1 necessitating the use of a softmax function.

An interesting thing to note is that taking the derivative of the loss as both mse and binary cross entropy have given very similar results which is not surprising at the very least.

Test Run for the following hyperparamters

```
mlp = MLP_SingleLabelClassifier(input_size=X_train.shape[1], hidden_layers=[20, 20], output_size=num_classes,
          learning_rate=0.01, activation='tanh', optimizer='mini_batch_gd',
          batch_size=32, epochs=1000)
```

Early stopping at epoch 137. Best validation loss: 1.0665 Training Data Metrics: Accuracy: 0.661925601750547

Test Data Metrics: Accuracy: 0.631578947368421

Training Data Classification Report: precision recall f1-score support

```
       0       0.00      0.00      0.00         5
       1       1.00      0.03      0.07        29
       2       0.70      0.79      0.74       389
       3       0.63      0.68      0.65       369
       4       0.60      0.42      0.49       110
       5       0.00      0.00      0.00        12

accuracy                           0.66       914
```

macro avg 0.49 0.32 0.33 914 weighted avg 0.66 0.66 0.64 914

Test Data Classification Report: precision recall f1-score support

```
       1       0.00      0.00      0.00         1
       2       0.64      0.67      0.65        48
       3       0.65      0.64      0.65        53
       4       0.50      0.60      0.55        10
       5       0.00      0.00      0.00         2

accuracy                           0.63       114
```

macro avg 0.36 0.38 0.37 114 weighted avg 0.62 0.63 0.62 114

Layer 1:

Weights - Max difference: 2.718982736443082e-09

Biases - Max difference: 6.4060025441788765e-09

Relative difference (weights): 2.1347216950295636e-07

Relative difference (biases): 4.322618242263979e-07

Layer 2:

Weights - Max difference: 3.4170251552324143e-09

Biases - Max difference: 6.113528176320687e-09

Relative difference (weights): 3.9494417957312275e-07

Relative difference (biases): 9.40217072841439e-07

Layer 3:

Weights - Max difference: 7.3655226705098e-09

Biases - Max difference: 7.586337138782567e-09

Relative difference (weights): 5.345086151443521e-07

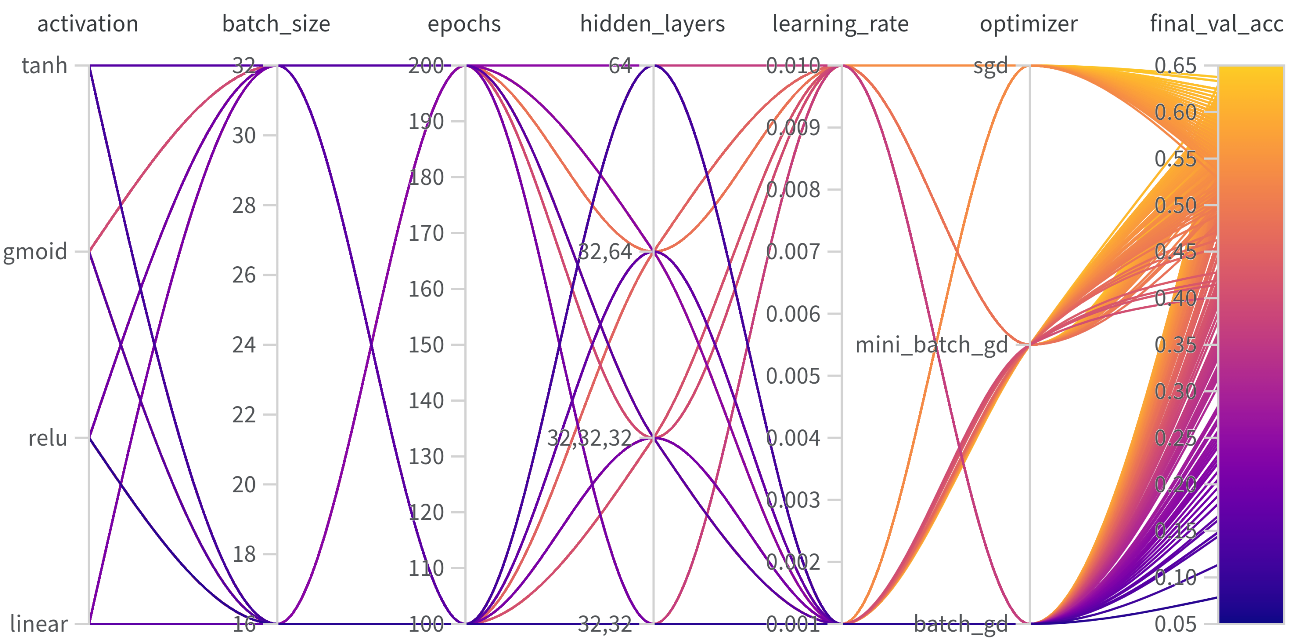
Relative difference (biases): 1.052437534849658e-06

gradient check gave a very good output of around 10^-7 using the norm verification formula, this was the formula used for calculating the numerical gradient

$$\frac{d}{d \theta} J(\theta)=\lim _{\epsilon \rightarrow 0} \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2 \epsilon}$$

2.3 Wandb Integration

```
sweep_config = {
    'method': 'grid',
    'metric': {'name': 'final_val_acc', 'goal': 'maximize'},
    'parameters': {
        'hidden_layers': {'values': [[64], [32, 32], [32, 64], [32, 32, 32]]},
        'learning_rate': {'values': [0.001, 0.01]},
        'activation': {'values': ['relu', 'sigmoid', 'linear', 'tanh']},
        'optimizer': {'values': ['sgd', 'mini_batch_gd', 'batch_gd']},
        'batch_size': {'values': [16, 32]},
        'epochs': {'values': [100, 200]}
    }
}
```



Name (384 visualized)	State	Runtim	activation	batch_size	epochs	hidden_layers	learning_rate	optimizer	best_val_acc	final_f1_score	final_precision	final_recall	final_val_a
ethereal-sweep-61	Finished	12s	relu	32	100	[32,64]	0.001	sgd	0.63755	0.628	0.62026	0.63755	0.63755
fearless-sweep-142	Finished	15s	sigmoid	16	200	[32,32,32]	0.01	sgd	0.63319	0.62112	0.61727	0.63319	0.63319
fresh-sweep-47	Finished	12s	relu	16	200	[32,32,32]	0.01	mini_batch_gd	0.62882	0.61688	0.60733	0.62882	0.62882
dutiful-sweep-43	Finished	11s	relu	16	200	[32,32,32]	0.001	sgd	0.62445	0.60965	0.60426	0.62445	0.62445
daily-sweep-301	Finished	10s	tanh	16	100	[32,64]	0.001	sgd	0.62445	0.60896	0.5975	0.62445	0.62445
daily-sweep-190	Finished	15s	sigmoid	32	200	[32,32,32]	0.01	sgd	0.62009	0.60979	0.60244	0.62009	0.62009
earthy-sweep-5	Finished	8s	relu	16	100	[64]	0.01	mini_batch_gd	0.62009	0.60519	0.59592	0.62009	0.62009
jolly-sweep-359	Finished	9s	tanh	32	100	[32,32,32]	0.01	mini_batch_gd	0.62009	0.60634	0.60055	0.62009	0.62009
dashing-sweep-317	Finished	8s	tanh	16	200	[64]	0.01	mini_batch_gd	0.62009	0.60589	0.59677	0.62009	0.62009
decent-sweep-311	Finished	8s	tanh	16	100	[32,32,32]	0.01	mini_batch_gd	0.61572	0.60513	0.60002	0.61572	0.61572
twilight-sweep-25	Finished	9s	relu	16	200	[64]	0.001	sgd	0.61572	0.60561	0.59894	0.61572	0.61572
revived-sweep-362	Finished	12s	tanh	32	200	[64]	0.001	mini_batch_gd	0.61572	0.6011	0.59035	0.61572	0.61572
giddy-sweep-71	Finished	8s	relu	32	100	[32,32,32]	0.01	mini_batch_gd	0.61572	0.60449	0.60529	0.61572	0.61572
lunar-thunder-55	Finished	5s	relu	16	200	[32,32]	0.01	mini_batch_gd	0.61135	0.59693	0.58912	0.61135	0.61135
denim-sweep-88	Finished	8s	relu	32	200	[32,64]	0.01	sgd	0.61135	0.60494	0.60857	0.61135	0.61135
mild-sweep-73	Finished	10s	relu	32	200	[64]	0.001	sgd	0.61135	0.59578	0.59216	0.61135	0.61135
azure-sweep-335	Finished	10s	tanh	16	200	[32,32,32]	0.01	mini_batch_gd	0.61135	0.59981	0.59582	0.61135	0.61135
leafy-sweep-11	Finished	7s	relu	16	100	[32,32]	0.01	mini_batch_gd	0.60699	0.59531	0.59713	0.60699	0.60699
fancy-sweep-95	Finished	13s	relu	32	200	[32,32,32]	0.01	mini_batch_gd	0.60699	0.59167	0.59204	0.60699	0.60699
mild-sweep-17	Finished	9s	relu	16	100	[32,64]	0.01	mini_batch_gd	0.60699	0.59636	0.59596	0.60699	0.60699

Runtim	activation	batch_size	epochs	hidden_layers	learning_rate	optimizer
12s	relu	32	100	[32,64]	0.001	sgd

2.4 Evaluating Single Label Classification Model

Training Data Metrics: Accuracy: 0.7385120350109409

Test Data Metrics: Accuracy: 0.631578947368421

Training Data Classification Report: precision recall f1-score support

```
       0       1.00      0.20      0.33         5
       1       0.00      0.00      0.00        29
       2       0.77      0.82      0.79       389
       3       0.70      0.78      0.74       369
       4       0.80      0.62      0.70       110
       5       0.00      0.00      0.00        12

accuracy                           0.74       914
```

macro avg 0.54 0.40 0.43 914 weighted avg 0.71 0.74 0.72 914

Test Data Classification Report: precision recall f1-score support

```
       1       0.00      0.00      0.00         1
       2       0.65      0.65      0.65        48
       3       0.66      0.66      0.66        53
       4       0.46      0.60      0.52        10
       5       0.00      0.00      0.00         2

accuracy                           0.63       114
```

macro avg 0.35 0.38 0.37 114 weighted avg 0.62 0.63 0.62 114

2.6

Best Parameters:

- Validation Accuracy : 0.683
- Test Accuracy : 0.691
- Number of epochs : 1000
- Learning Rate : 0.001
- Optimizer : Stochastic Gradient Descent

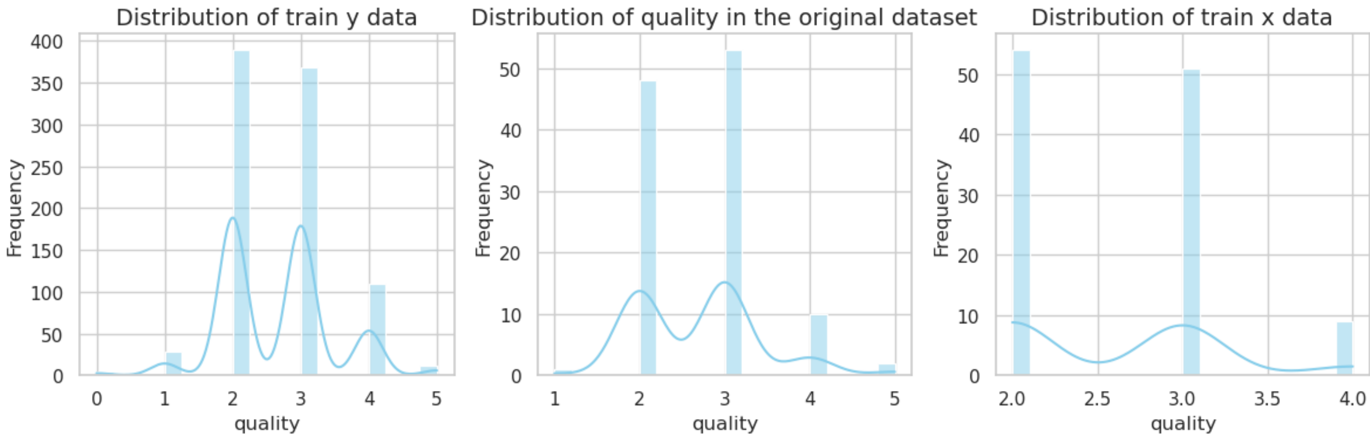
- Activation Function : ReLU

Name (26 visualized)	State	Notes	User	Tags	Created	Runtime	Sweep	activat	optimi:	acc_	acc_va	batch_	learnin	learnin	num_e	train_l
Optimizer=...ation=relu	Finished	Add notes	mjolnir		7m ago	6m 27s	-	relu	sgd	0.7	0.647	-	-	0.004	3000	2.72938
Optimiser ...:tion = relu	Finished	Add notes	mjolnir		4h ago	6m 43s	-	-	-	0.697	0.677	-	0.004	-	3000	2.72768
Optimiser ...:tion = relu	Killed	Add notes	mjolnir		6h ago	4m 31s	-	-	-	0.692	0.657	-	0.002	-	1500	2.70383
Optimizer=...ation=relu	Finished	Add notes	mjolnir		2h ago	6m 31s	-	relu	sgd	0.69	0.673	-	-	0.004	3000	2.71833
Optimiser ...ı = sigmoid	Finished	Add notes	mjolnir		3h ago	1m 18s	-	-	-	0.686	0.669	32	0.004	-	3000	2.82814
Optimiser ...:tion = relu	Finished	Add notes	mjolnir		4h ago	2m 12s	-	-	-	0.685	0.647	32	0.004	-	3000	2.75875
Optimizer=...n=sigmoid	Finished	Add notes	mjolnir		2h ago	1m 44s	-	sigmoid	mini_gd	0.682	0.658	32	-	0.004	3000	2.83258
Optimiser ...:tion = relu	Running	Add notes	mjolnir		6h ago	5h 40m 33	-	-	-	0.681	0.645	-	0.002	-	100	2.77826
Optimiser ...ı = sigmoid	Finished	Add notes	mjolnir		4h ago	12s	-	-	-	0.68	0.668	-	0.004	-	3000	2.92146
Optimizer=...ation=tanh	Finished	Add notes	mjolnir		2h ago	1m 48s	-	tanh	mini_gd	0.68	0.662	32	-	0.004	3000	2.74622
Optimiser ...tion = tanh	Finished	Add notes	mjolnir		4h ago	16s	-	-	-	0.68	0.667	-	0.004	-	3000	2.82593
Optimizer=...n=sigmoid	Finished	Add notes	mjolnir		2h ago	11s	-	sigmoid	batch_gd	0.68	0.669	-	-	0.004	3000	2.90565
Optimizer=...ation=tanh	Finished	Add notes	mjolnir		2h ago	13s	-	tanh	batch_gd	0.678	0.658	-	-	0.004	3000	2.83109
Optimizer=...ation=relu	Finished	Add notes	mjolnir		2h ago	1m 46s	-	relu	mini_gd	0.675	0.633	32	-	0.004	3000	2.7429
Optimiser ...:tion = relu	Finished	Add notes	mjolnir		4h ago	11s	-	-	-	0.674	0.66	-	0.004	-	3000	2.84114

2.7

The above table has all 5 wine quality classes and the classification report gives all metrics for all the 5 classes seperately

Most of the labels are between 2 3 and 4 as we can see from the histogram table. The model does well in classifying between 2 3 and 4 but it's perfomance is poor for classifying into the other classes as the data available for those classes is very less.



3. Multi Layer Perceptron Regression

3.1 Data Preprocessing

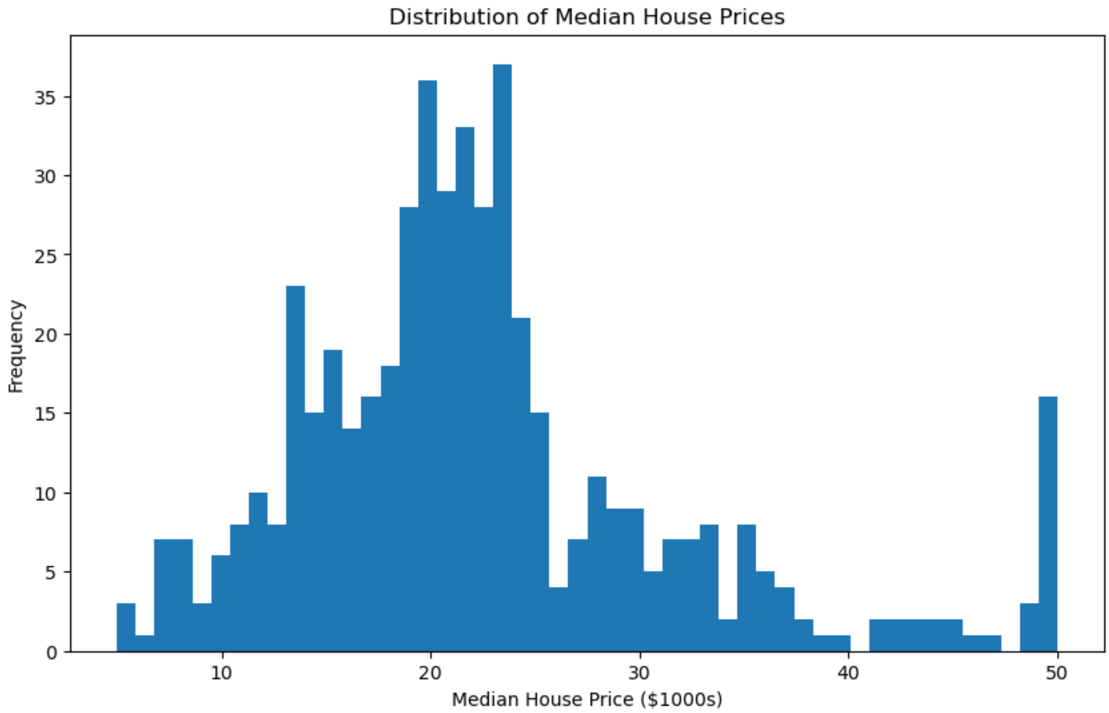
3.1.1

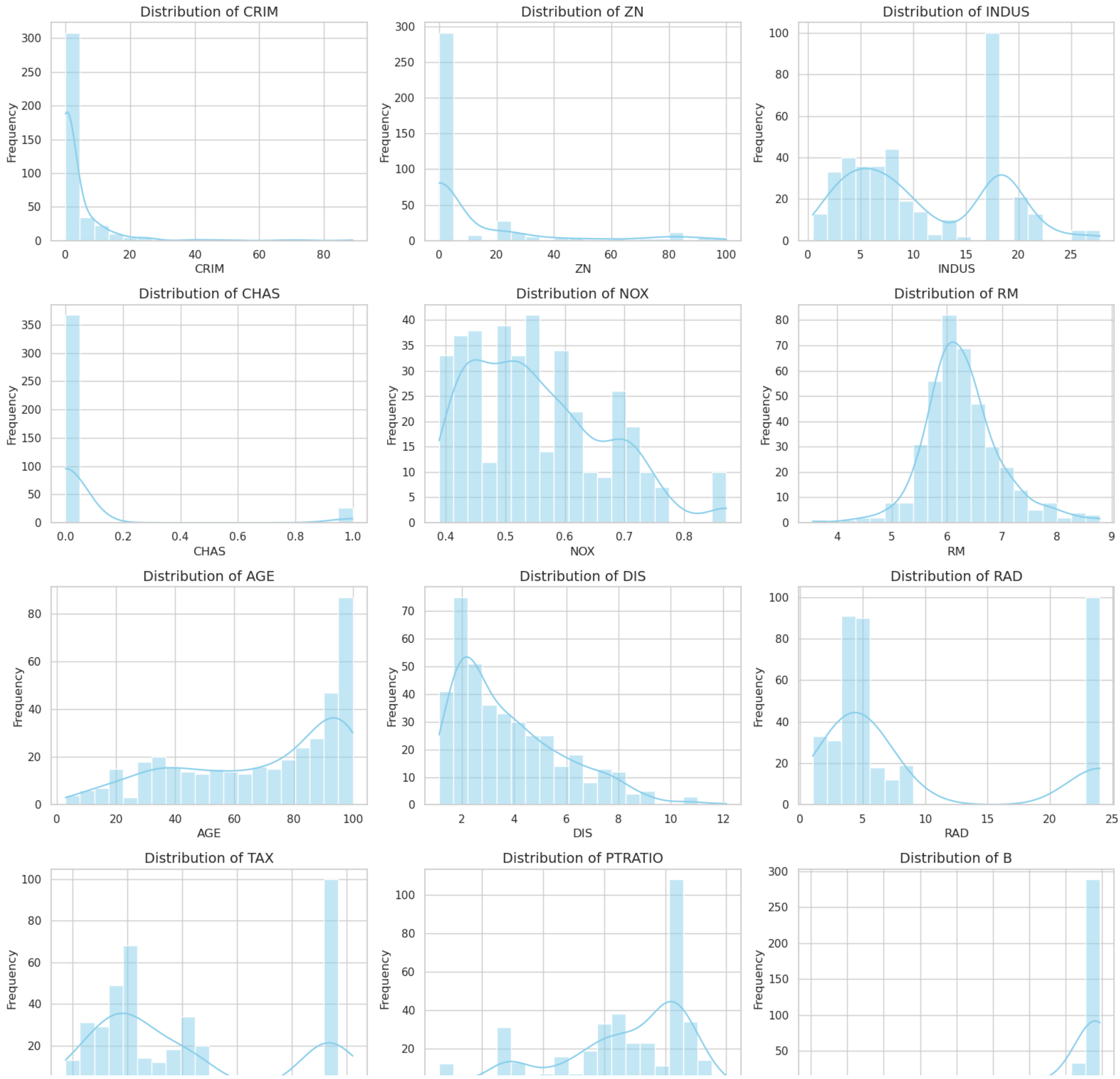
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

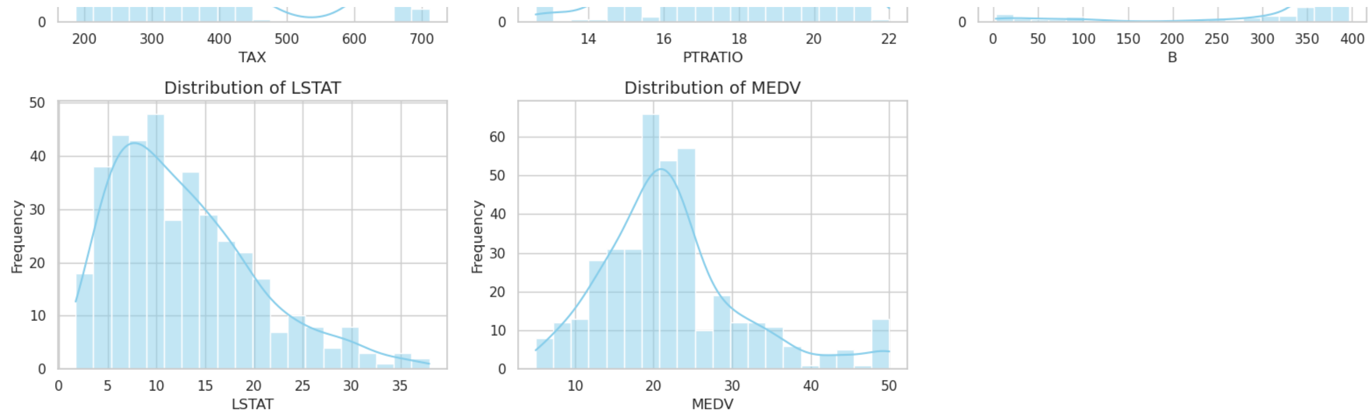
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRA
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.000000	506.000000	506.00
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237154	18.455
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.707259	168.537116	2.1649
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.000000	279.000000	17.400
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.000000	330.000000	19.050

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRA
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.000000	666.000000	20.200
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000

3.1.2


Distribution of Median House Prices
Frequency
Median House Price ($1000s)
0
5
10
15
20
25
30
35
10
20
30
40
50


Distribution of CRIM
CRIM
Distribution of ZN
ZN
Distribution of INDUS
INDUS
Distribution of CHAS
CHAS
Distribution of NOX
NOX
Distribution of RM
RM
Distribution of AGE
AGE
Distribution of DIS
DIS
Distribution of RAD
RAD
Distribution of TAX
Distribution of PTRATIO
Distribution of B
Frequency

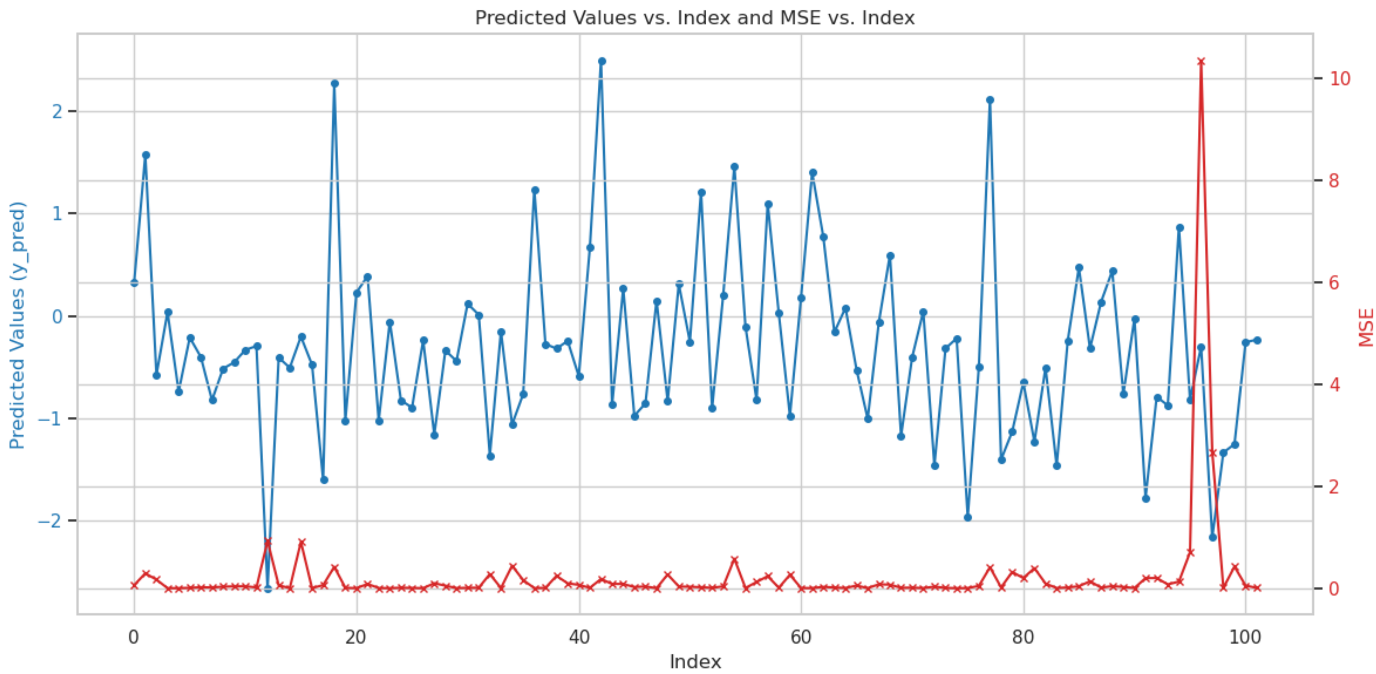


3.2 MLP Regression from Scratch

For performing the regresion task the main differences from the single label classification are as follow:

- The Final Activation layer isnt softmax as the output is regression and shouldn't be probabilities, so it is a linear function thus changing one derivative term in the backprop algorithm.
- And the loss function is the Mean Square Error function.

```
Epoch 1/1000, Train Loss: 0.8448, Val Loss: 0.7813
Epoch 101/1000, Train Loss: 0.0953, Val Loss: 0.1476
Epoch 201/1000, Train Loss: 0.0842, Val Loss: 0.1391
Early stopping at epoch 218. Best validation loss: 0.1311
Training Data Metrics:
  MSE: 0.0813445504484368, MAE: 0.20858720960733684, R2: 0.9186554495515632

Test Data Metrics:
  MSE: 0.23068735567223425, MAE: 0.28336033533464455, R2: 0.7267208243619698
```

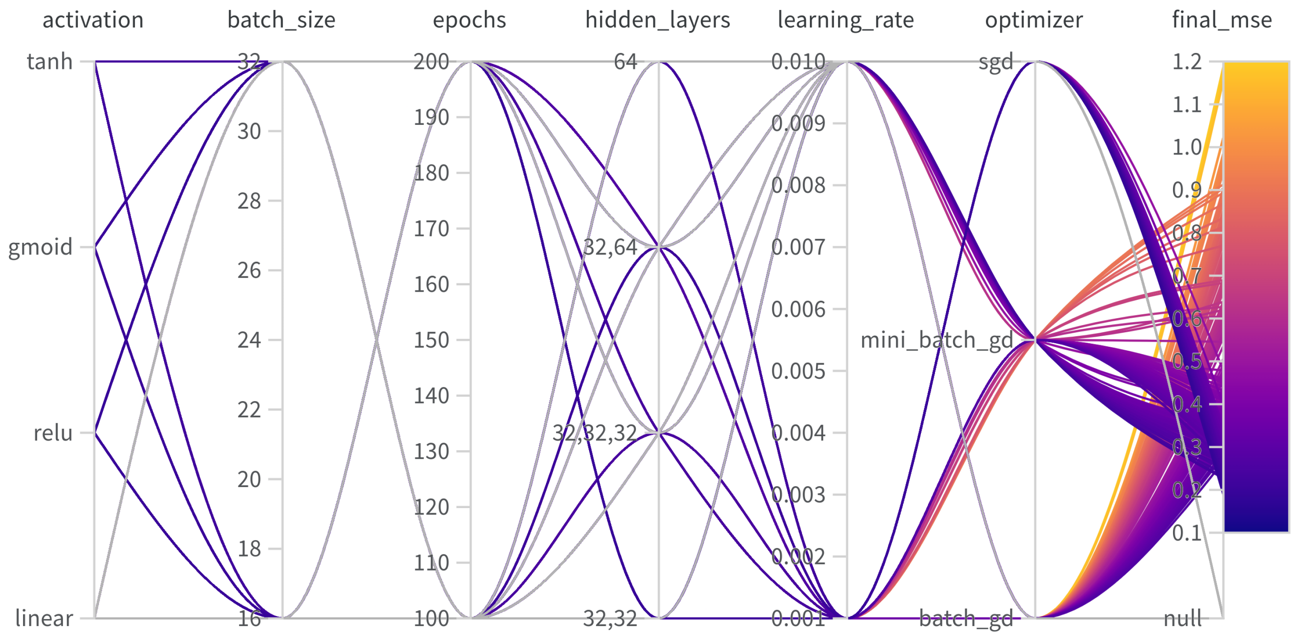


We can see that the comparision of 10 values are close too

```
y_test =  [[ 1.03034948e+00]
 [-9.86690150e-01]
 [ 3.71794749e-04]
 [-7.18466795e-01]
 [-3.00038362e-01]
 [-5.36074914e-01]
 [-9.43774413e-01]
 [-3.42954099e-01]
 [-6.43364256e-01]]
y_test_pred =  [[ 1.46753008]
 [-0.59132107]
 [ 0.19043355]
 [-0.90686476]
 [-0.60504998]
 [-0.37792748]
 [-0.8440979 ]
 [-0.00153177]
 [-0.27154073]]
```

```
model.check_gradients(X, y)
✓ 0.0s

MAE: 0.0016892924588279446
MSE: 7.265086944441281e-07

np.float64(0.02628028229447117)
```

3.3 Wandb Integration



Name (387 visualized)	activat	batch_	epochs	hidden	learning_rate	optimizer	best_loss	best_val_mse	epoch	final_mae	final_mse	final_r2	train_loss	val_l
stellar-sweep-133	sigmoid	16	200	[32,32]	0.01	sgd	0.2866	0.19197	200	0.27463	0.19197	0.78282	0.21247	0.286
peachy-sweep-181	sigmoid	32	200	[32,32]	0.01	sgd	0.29212	0.19478	200	0.2645	0.19478	0.77965	0.2058	0.291
fragrant-sweep-175	sigmoid	32	200	[64]	0.01	sgd	0.29802	0.2041	200	0.29254	0.2041	0.7691	0.18875	0.297
smart-sweep-4	relu	16	100	[64]	0.01	sgd	0.30235	0.2085	80	0.28066	0.2085	0.76413	0.13538	0.302
efficient-sweep-84	relu	32	200	[32,32]	0.01	sgd	0.30296	0.1881	37	0.27295	0.1881	0.7872	0.22312	0.461
woven-sweep-325	tanh	16	200	[32,32]	0.01	sgd	0.30739	0.22367	200	0.31712	0.22367	0.74696	0.11689	0.307
confused-sweep-295	tanh	16	100	[64]	0.01	sgd	0.30971	0.20844	100	0.25484	0.20844	0.76419	0.14691	0.309
silver-sweep-10	relu	16	100	[32,32]	0.01	sgd	0.31433	0.19512	51	0.25926	0.19512	0.77927	0.13919	0.315
deft-sweep-60	relu	32	100	[32,32]	0.01	sgd	0.31463	0.20267	45	0.27401	0.20267	0.77072	0.14209	0.325
easy-sweep-127	sigmoid	16	200	[64]	0.01	sgd	0.31911	0.22874	200	0.28487	0.22874	0.74123	0.17935	0.318
pious-sweep-301	tanh	16	100	[32,32]	0.01	sgd	0.31987	0.2197	100	0.29349	0.2197	0.75145	0.15413	0.319
unique-sweep-187	sigmoid	32	200	[32,64]	0.01	sgd	0.32239	0.20068	200	0.28492	0.20068	0.77298	0.25332	0.321
blooming-surf-83	relu	32	200	[32,32]	0.001	sgd	0.32554	0.21374	200	0.25294	0.21374	0.7582	0.15254	0.325
rich-sweep-331	tanh	16	200	[32,64]	0.01	sgd	0.32777	0.21872	200	0.28262	0.21872	0.75257	0.13597	0.327
peach-sweep-139	sigmoid	16	200	[32,64]	0.01	sgd	0.33202	0.2078	200	0.29145	0.2078	0.76492	0.25564	0.331
trim-sweep-34	relu	16	200	[32,32]	0.01	sgd	0.33351	0.21385	39	0.2869	0.21385	0.75807	0.15404	0.336
whole-sweep-223	linear	16	200	[64]	0.01	sgd	0.33595	0.31199	200	0.34052	0.31199	0.64705	0.24924	0.335

We can see the sgd performs much much better compared to that of the other optimizers, sigmoid and relu are better than tanh when others are hyperparamters are fixed, learning rate is optimial at 0.01 and 2 layers > 1layer > 3 layers here .

Name (387 visualized)	activat	batch_	epochs	hidden	learning_rate	optimizer	best_loss	best_val_mse	epoch	final_mae	final_mse	final_r2	train_loss	val_l
stellar-sweep-133	sigmoid	16	200	[32,32]	0.01	sgd	0.2866	0.19197	200	0.27463	0.19197	0.78282	0.21247	0.286

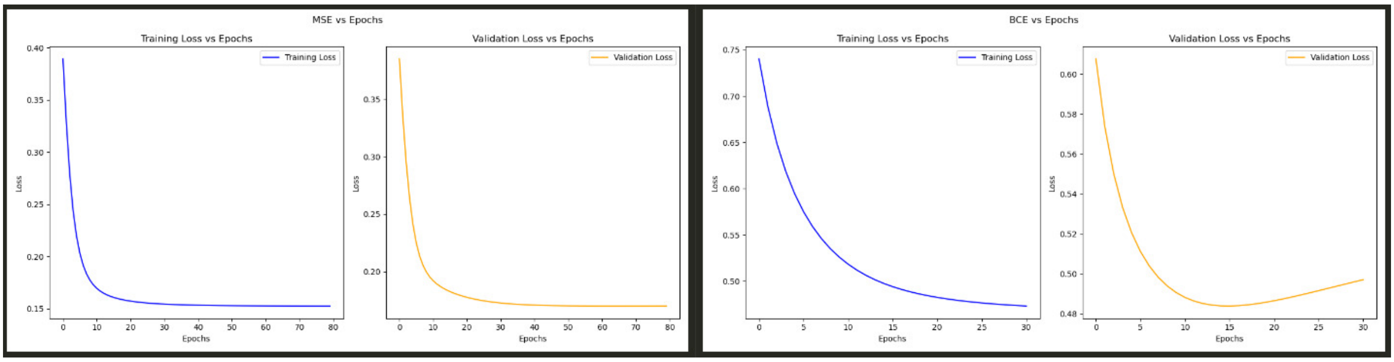
These are the best parameters

3.4

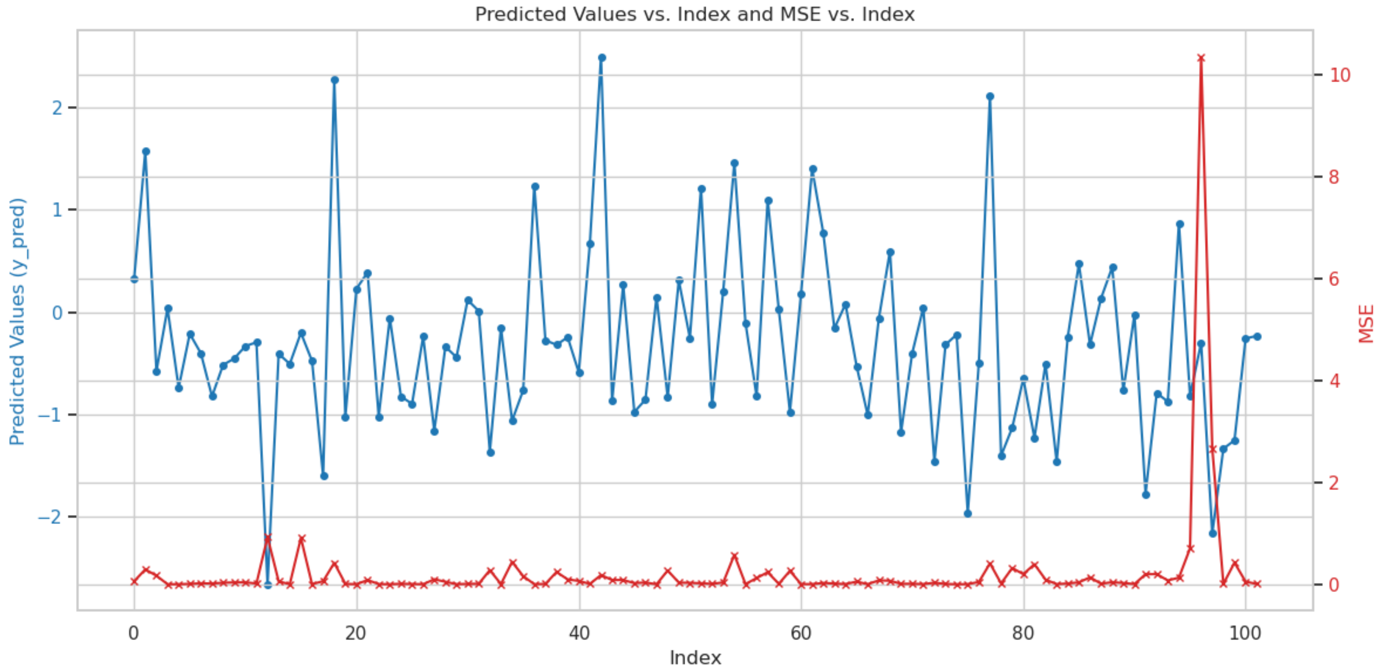
```
mlp = MLP_Regressor(input_size=X_train.shape[1], hidden_layers=[32, 32], output_size=1,
                    learning_rate=0.01, activation='sigmoid', optimizer='sgd',
                    batch_size=32, epochs=200,  patience=10)
```

```
Epoch 1/200, Train Loss: 0.8739, Val Loss: 0.6794
Epoch 101/200, Train Loss: 0.1362, Val Loss: 0.1759
Training Data Metrics:
  MSE: 0.09559504392853287, MAE: 0.22174151526212468, R2: 0.9044049560714671

Test Data Metrics:
  MSE: 0.14179505945108242, MAE: 0.233021284983614, R2: 0.8320253104318659
```

3.5 MSE vs BCE



3.6 Analysis



There is difference we can see in the rate of convergence the mse convergence is much quicker and decays faster as compared to that of the bce loss which looks more logarithmic than that of the the BCE loss plot.

The mse is high when the data stops following a certain patern Also the mse appears shoots up when the peaks are too close together in the sequence of data

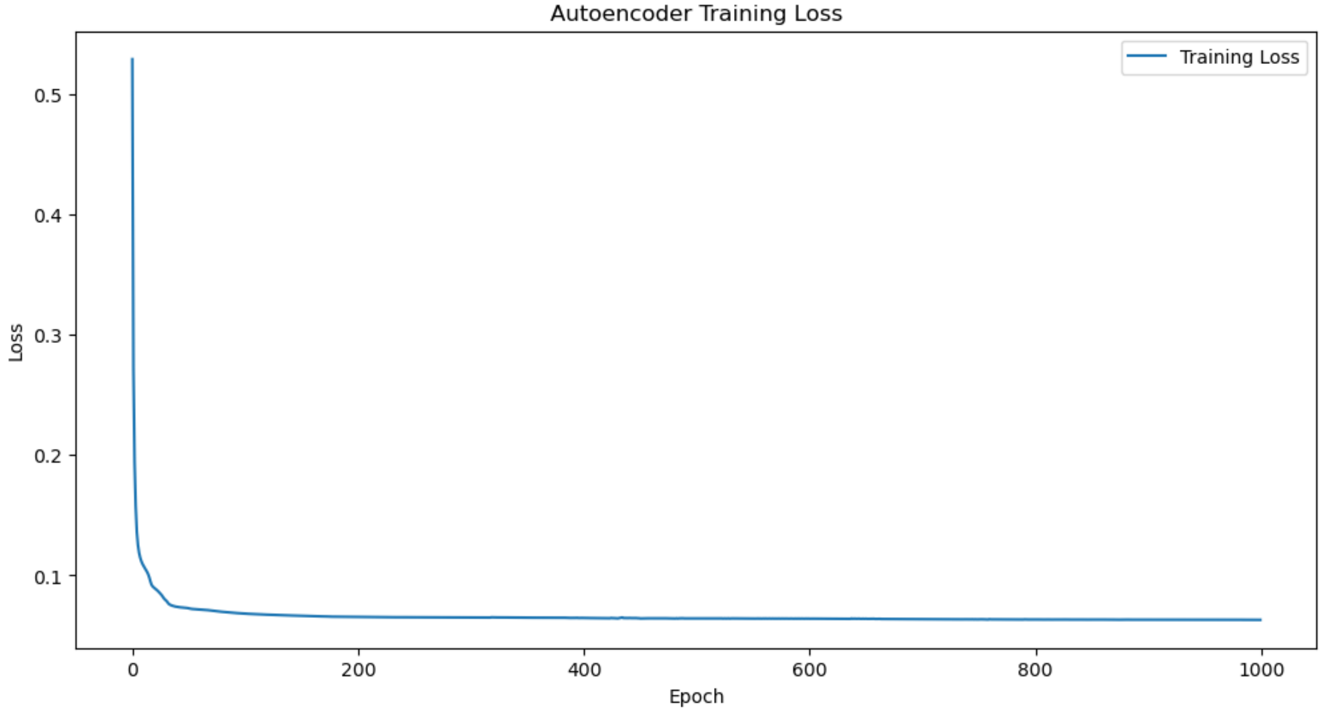
3.7 Bonus

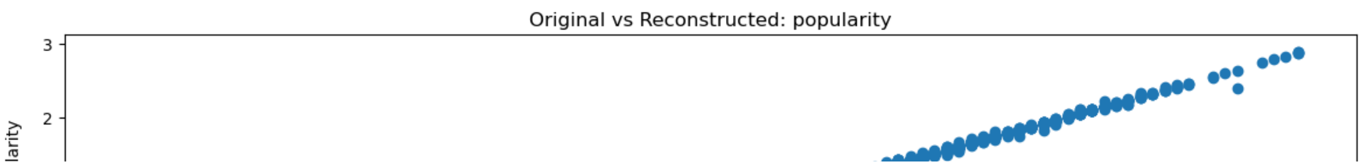
I have integrated both the MLP Regressor and Classifier into the MLP Class where the onyl changes are argument has a new 'classifer/regressor' based on whcih there are if conditions for the output layer and its activation along with some changes in gradient function under the same if condition. Also the loss is different for both so even the loss and loss derivative have the if condition.

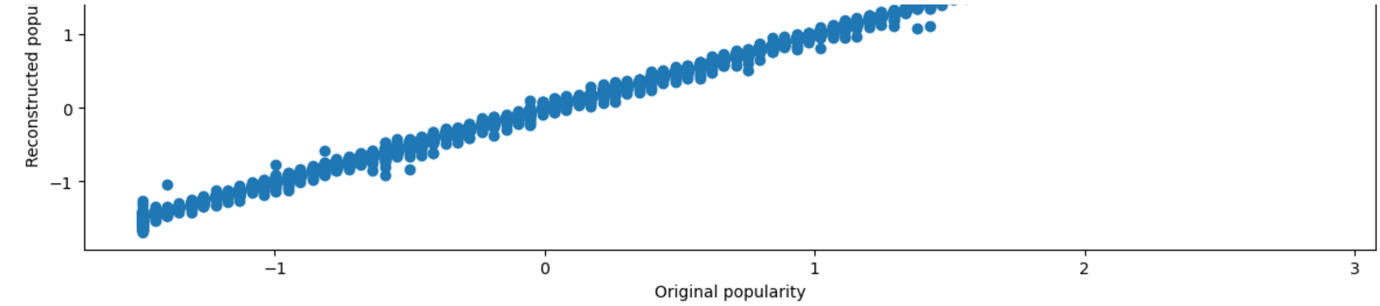
4. AutoEncoders

4.1

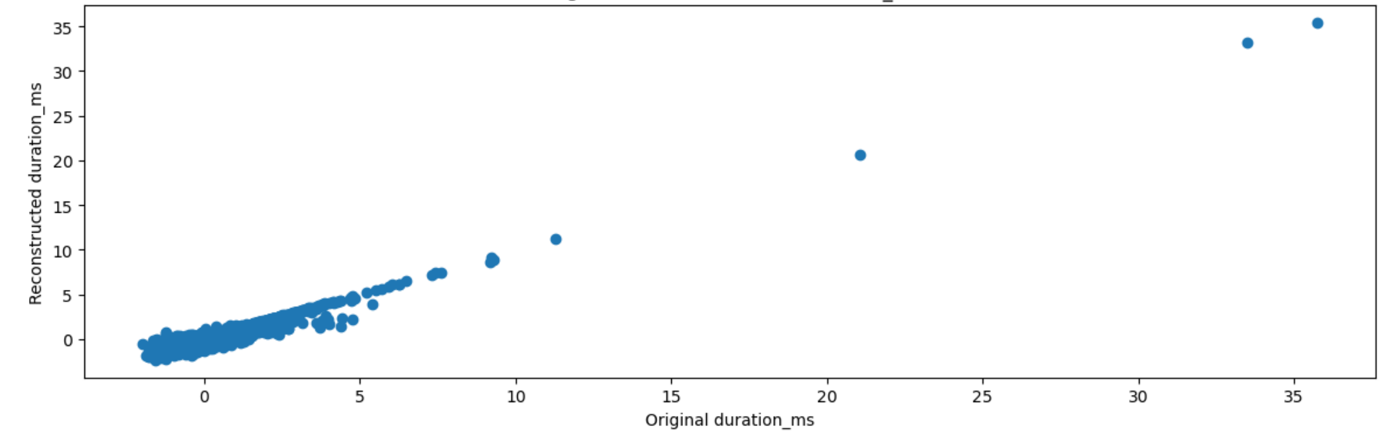
For the AutoEncoder we make a multi layer output mlp regressor with the input and output dimensions equal to the dimensions of the present dataset and the central layer is the latent layer which learns the reduced dataset In out case it is 15 -> 12 -> 9 -> 12 -> 15 where 9 is latent layer containing reduced data.



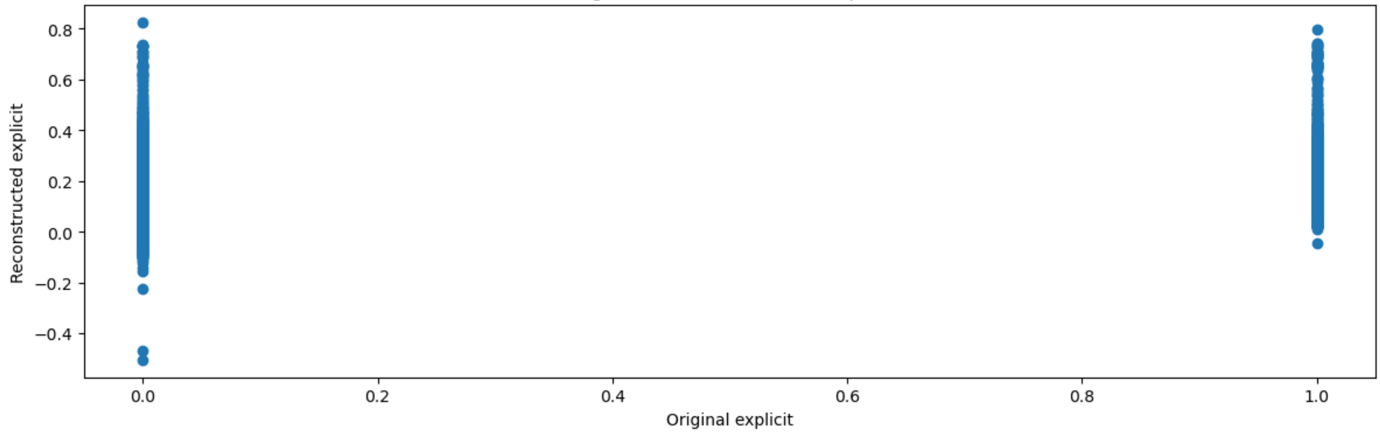




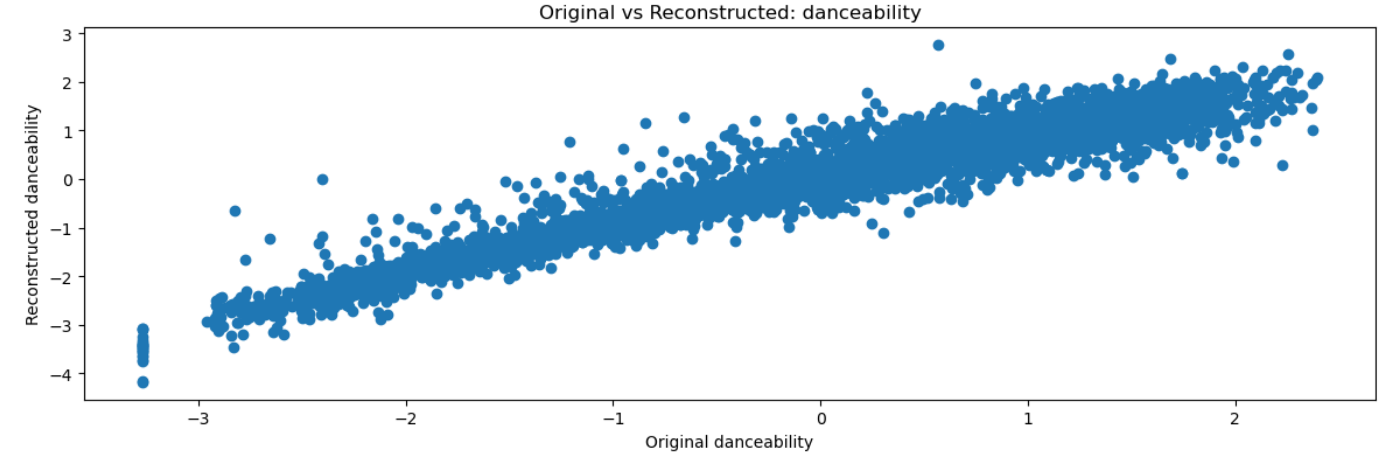
Original vs Reconstructed: duration_ms



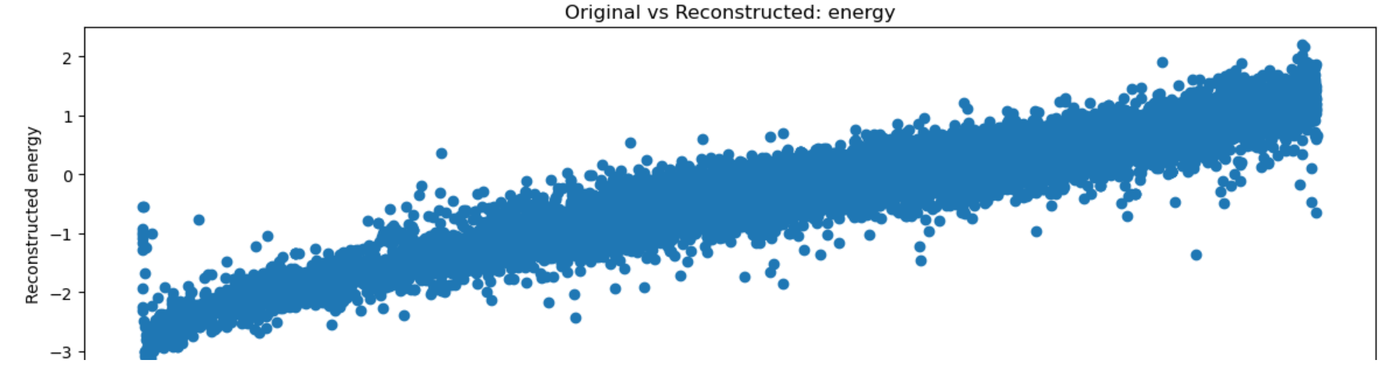
Original vs Reconstructed: explicit

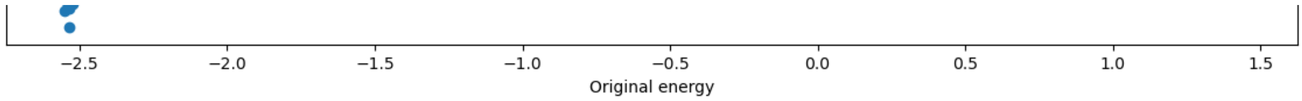


Original vs Reconstructed: danceability



Original vs Reconstructed: energy






```
Original shape: (91200, 15)
Reduced shape: (91200, 9)
Train MSE: 0.05237875614032062
Test MSE: 0.05310546740315962
Train R2: 0.8937088168166328
Test R2: 0.8923706762685729
```

4.3

This is on the reduced dimensions of the AutoEncoder

```
k     Distance     Acc       Prec      Recall    F1        Time (s)
23    manhattan    0.2024    0.1010    0.0926    0.0966    118.3068
```

this is on the reduced dimensions of the PCA

Fit KNN on this reduced dataset

Best k: 23 **Best Distance Metric**: Manhattan

Comparing the Metrics and Time

```
Without PCA
k     Distance     Acc       Prec      Recall    F1        Time (s)
23    manhattan    0.2510    0.2222    0.1667    0.1905    139.0210

With PCA (7 dimensions)
k     Distance     Acc       Prec      Recall    F1        Time (s)
23    manhattan    0.1868    0.1183    0.1019    0.1095    112.8799
```

4.4 MLP Classifier on spotify data

```
k     Distance     Acc       Prec      Recall    F1        Time (s)
23    manhattan    0.2261    0.1618    0.1019    0.1250    187.9299
```

The metrics are as good as the knn results we have acheived in the assignment-1

The accuracy is very simialr but once the mlp is trained the prediction time is very small. The only time going to training the weights and knn increases expoenentially with time which is not the case with MLP assuming same hyperparameters.