Implementation of Logistic Regression

```
Training set Accuracy:92.72399999999999%

Validation set Accuracy:91.47999999999999%

Testing set Accuracy:92.03%
```

During the training phase the model learns various patterns and even the noise in the training data which gives higher accuracy. But when it comes to test phase the model is applied to the new unseen test data, which fails to generalize and struggles to find the patterns. But finally, it provides a reasonably good accuracy.

Total error with respect to each category:

```
 Training set Accuracy for Logistic Regression:92.72399999999999%
Error for Category 0: 2.2141%
Error for Category 1: 2.0899%
Error for Category 2: 8.7939%
Error for Category 3: 10.0955%
Error for Category 4: 6.3610%
Error for Category 5: 11.7168%
Error for Category 6: 3.6804%
Error for Category 7: 5.7740%
Error for Category 8: 12.5541%
Error for Category 9: 10.8103%

 Testing set Accuracy Logistic Regression:92.03%
Error for Category 0: 1.9388%
Error for Category 1: 1.7621%
Error for Category 2: 10.9496%
Error for Category 3: 8.6139%
Error for Category 4: 6.7210%
Error for Category 5: 14.1256%
Error for Category 6: 5.2192%
Error for Category 7: 7.6848%
Error for Category 8: 13.0390%
Error for Category 9: 10.9019%
```

Overall, we can see that the categories 0 and 1 have low error difference which makes the model to easily classify the training and test sets. However, category 5 shows largest error difference which means that the features may overlap with other categories which causes misclassifications on unseen data.

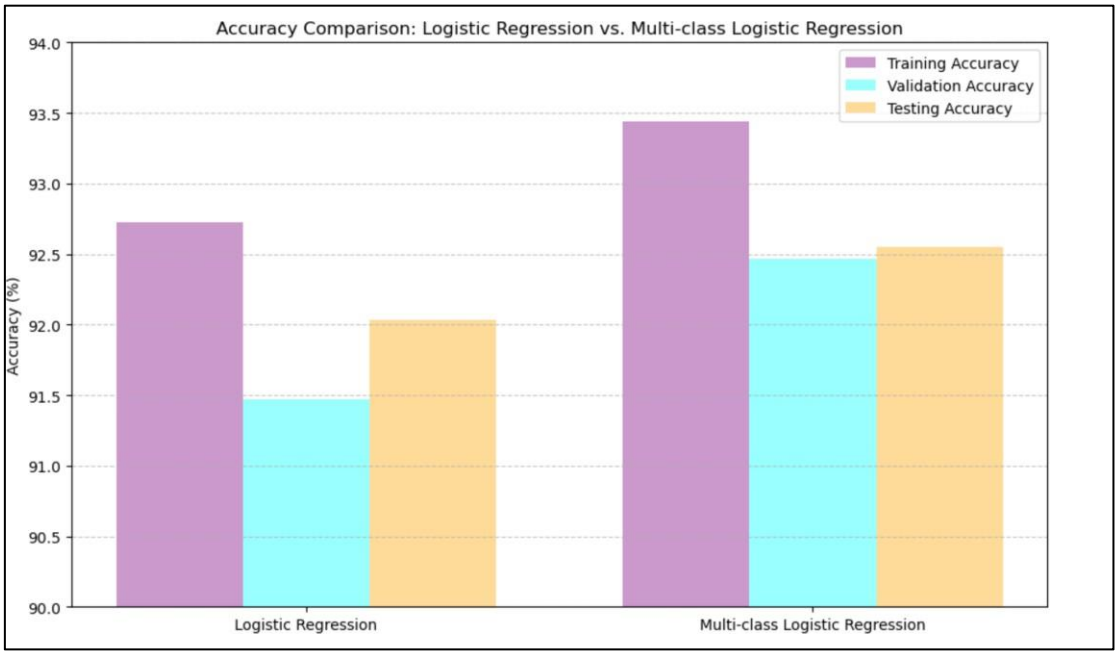
Multi-class Logistic Regression

```
Training set Accuracy:93.448%

Validation set Accuracy:92.47999999999999%

Testing set Accuracy:92.55%
```

Total error with respect to each category for MLR:

```
 Training set Accuracy for Multi-class Logistic Regression: 93.448%
Error for Category 0: 2.7829%
Error for Category 1: 2.6123%
Error for Category 2: 9.1771%
Error for Category 3: 9.2964%
Error for Category 4: 5.4936%
Error for Category 5: 10.3596%
Error for Category 6: 3.2534%
Error for Category 7: 5.2422%
Error for Category 8: 9.5650%
Error for Category 9: 8.7492%

 Testing set Accuracy for Multi-class Logistic Regression: 92.55%
Error for Category 0: 2.0408%
Error for Category 1: 2.2026%
Error for Category 2: 10.4651%
Error for Category 3: 9.5050%
Error for Category 4: 6.2118%
Error for Category 5: 13.3408%
Error for Category 6: 4.5929%
Error for Category 7: 7.3930%
Error for Category 8: 10.8830%
Error for Category 9: 8.9197%
```

Overall, we can see that the categories 0 and 1 shows lowest test errors which suggests that the model can effectively classify both training and test sets. However, category 5 shows highest error difference, which means that there is overlap with features from categories, leading to misclassifications on unseen data. Categories 2 and 8 also show relatively higher error differences, indicating some challenges in generalizing for these categories.



From the graph plotted, the multi class logistic regression shows better performance compared to logistic regression. This is because it considers all classes at once, rather than treating each class independently, which might be beneficial in cases where classes have overlapping features as it provides more balanced classification across categories.

Support Vector Machines

Linear kernel:

```
--------------SVM-------------------

SVM with linear kernel

 Training set Accuracy:97.286

 Validation set Accuracy:93.64

 Testing set Accuracy:93.78
```

The linear kernel worked well but has its limits because it assumes the data can be separated with a straight line, which isn't always true for complex datasets like MNIST. The small difference between training and testing accuracy means it slightly focuses too much on the training data but still does a decent job on new data.

RBF Kernel (Gamma = 1):

```
SVM with RBF kernel (gamma = 1)

 Training set Accuracy:100.0

 Validation set Accuracy:15.479999999999999

 Testing set Accuracy:17.14
```

With gamma=1, the RBF kernel did perfectly on the training data, but this shows it was overfitting a lot. It didn't do well on the validation or testing data, meaning it couldn't handle new examples. This happened because gamma=1 made the model focus too much on memorizing the training data instead of learning the general patterns.

RBF Kernel (Default Gamma):

```
Training Accuracy (Default Gamma): 98.98%
Validation Accuracy (Default Gamma): 97.89%
Testing Accuracy (Default Gamma): 97.87%
```

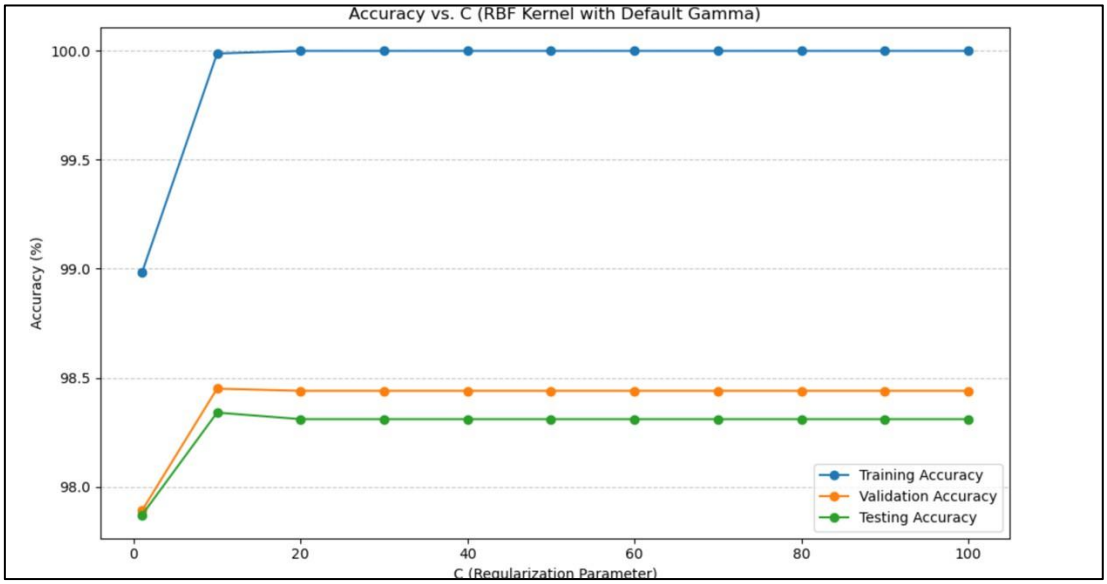
With the default gamma setting, the RBF kernel worked well. The gamma value automatically adjusted to the data, helping the model focus on overall patterns instead of getting stuck on details. The validation and testing accuracies were very close to the training accuracy, showing that the model handled new, unseen data effectively without overfitting.

Impact of C Parameter:

C	Train Acc (%)	Val Acc (%)	Test Acc (%)
1	98.98	97.89	97.87
10	99.99	98.45	98.34
20	100.00	98.44	98.31
30	100.00	98.44	98.31
40	100.00	98.44	98.31
50	100.00	98.44	98.31
60	100.00	98.44	98.31
70	100.00	98.44	98.31
80	100.00	98.44	98.31
90	100.00	98.44	98.31
100	100.00	98.44	98.31

As the C value increases, the training accuracy improves, starting at 98.98% for C=1 and reaching 100% for C=20 and higher. The validation and testing accuracies are highest at C=10, with 98.45% and 98.34%, and stay about the same for larger C values. When C is low, the model is better at generalizing to new data because it's more regularized. As C gets higher, the model focuses too much on the training data, leading to overfitting. Overall, C=10 gives the best balance between accuracy and generalization.

After checking the accuracies for different C values, the graph below shows how the training, validation, and testing accuracies change as C gets bigger.



Conclusion for SVM:

- The linear kernel did a decent job but struggled with the complex patterns in the MNIST dataset.
- The RBF kernel with gamma=1 focused too much on the training data and didn't work well with new data.
- The RBF kernel with default gamma (scale) gave the best results, balancing accuracy for both training and testing.
- The C parameter had a big impact, and C=10 gave the best balance between fitting the training data and working well on new data.

Overall, the RBF kernel with default gamma and C=10 was the best choice for this dataset.

CONCLUSION

- Logistic regression worked well overall but had trouble with overlapping features, especially in categories like 5 and 8. Multi-class logistic regression did better by looking at all the classes together, which helped it generalize better and give more balanced results.
- SVM turned out to be the most effective method. The linear kernel was fast but could not handle the non-linear patterns in the dataset. The RBF kernel with gamma=1 overfit the training data but adjusting the C value (with C=10 working best) gave the right balance between accuracy and generalization.
- Multi-class logistic regression outperformed one-vs-all logistic regression by handling overlapping features better. It looked at all the classes at the same time, which made its classifications more balanced and effective.