



**Faculty of Engineering – Cairo University**  
**Electronics And Electrical Communications Department**  
**Fourth year - Mainstream**  
**ELC4028 Neural Networks – Assignment 1**

**Submitted by:**

ID	Sec	Name
9213327	3	محمد أيمن فاروق سيد عبد الغفار
9211027	3	محمد علاء الدين عطفت مصطفى محمد رستم
9211039	4	محمد مجدي مبروك ندا خير
9213468	4	يوسف تامر صلاح الدين السيد
9211418	4	يوسف عصام أبوبكر محمد

**Submitted to:**

Prof. Mohsen Rashwan

# Part 1: Regressing Problems

## Problem 1: Insured Persons over Years

### 1. Plotting the given data

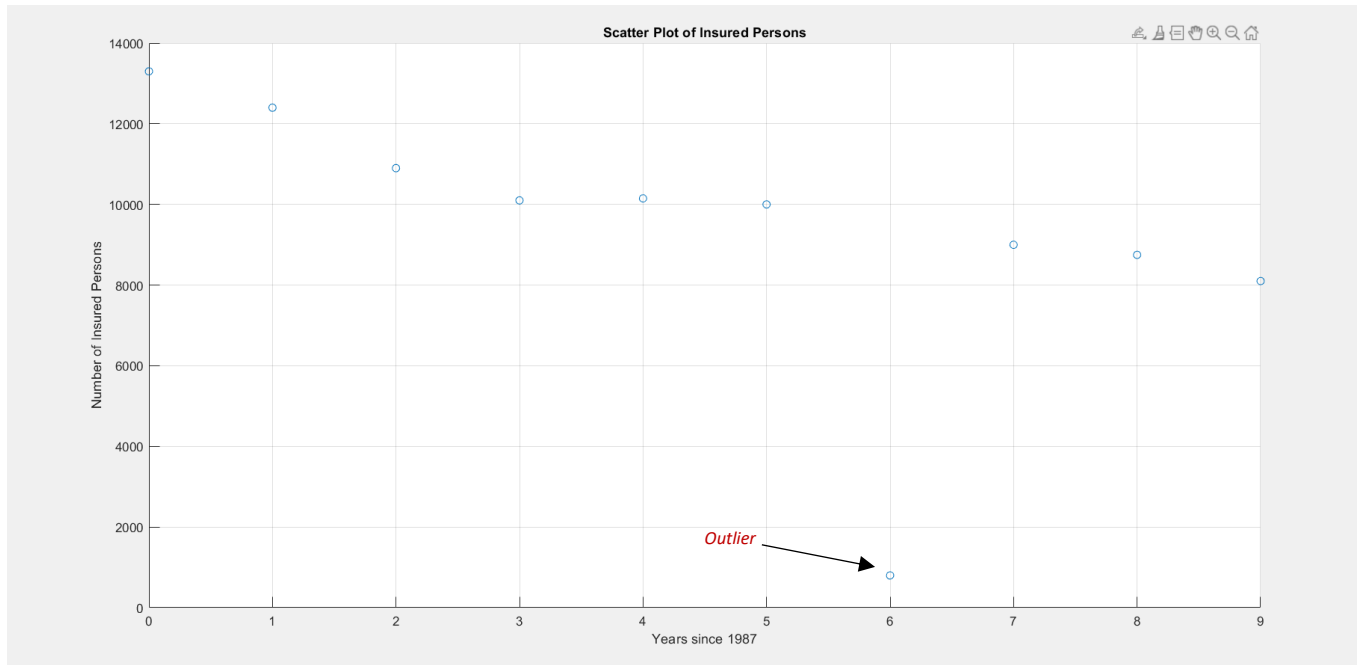


Figure 1: Scatterplot of given data

Figure 1 represents the yearly data along with the number of insured individuals. We can notice that the number of insured people was relatively low (800) in 1993 which indicates unusual behavior occurred that year, that is most likely due to a data entry error, making it an outlier.

## 2. Fitting before cleaning (With the existence of outlier)

### 2.1 Linear Model

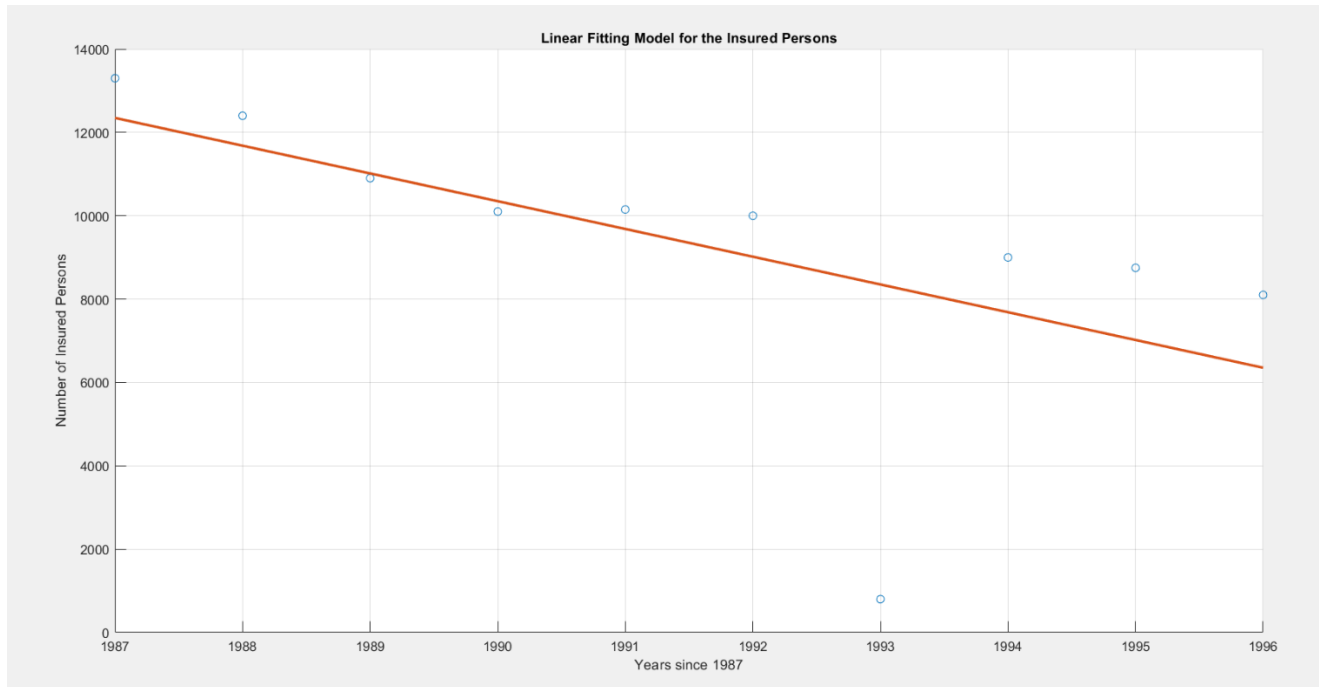


Figure 2: Linear Fitting Model for the Insured Persons

### Hand Analysis for calculating the Linear Fitting Parameters:

For Linear module analysis. The prediction equation is given by:  $\hat{y}_i = \hat{\beta}_1 x_i + \hat{\beta}_0$

Where  $\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i - \frac{(\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n}}{(\sum_{i=1}^n x_i^2) - \frac{(\sum_{i=1}^n x_i)^2}{n}}$ , and  $\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$  ( $n = 10$ )

$x_i$	$y_i$	$x_i^2$	$y_i^2$	$x_i y_i$
1987	13300	3948169	176890000	26427100
1988	12400	3952144	153760000	24651200
1989	10900	3956121	118810000	21680100
1990	10100	3960100	102010000	20099000
1991	10150	3964081	103022500	20208650
1992	10000	3968064	100000000	19920000
1993	800	3972049	640000	1594400
1994	9000	3976036	81000000	17946000
1995	8750	3980025	76562500	17456250
1996	8100	3984016	65610000	16167600
$\Sigma = 19915$	$\Sigma = 93500$	$\Sigma = 39660805$	$\Sigma = 978305000$	$\Sigma = 186150300$

Then  $\hat{\beta}_1 = -666.0606$ , ( $\bar{Y} = 9350, \bar{X} = 1991.5$ )  $\rightarrow \hat{\beta}_0 = 1335809.685$

$$\hat{y}_i = -666.0606x_i + 1335809.685$$

## MATLAB Result of the Linear Fitting Parameters:

The linear fitting equation is:  $y = -666.06x + 1335809.70$

## Hand Analysis for calculating $R^2$ Value:

$$R^2 = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}, \quad \bar{Y} = 9350, \quad n = 10, \quad \hat{Y}_i \text{ is calculated from the obtained equation}$$

$x_i$	$y_i$	$\hat{Y}_i$	$(Y_i - \hat{Y}_i)^2$	$(Y_i - \bar{Y})^2$
1987	13300	12347.27	907694.45	15602500
1988	12400	11681.21	516659.06	9302500
1989	10900	11015.15	13259.52	2402500
1990	10100	10349.09	62045.82	562500
1991	10150	9683.03	218060.98	640000
1992	10000	9016.97	966347.98	422500
1993	800	8350.91	57016241.83	73102500
1994	9000	7684.85	1729619.52	122500
1995	8750	7018.79	2997088.06	360000
1996	8100	6352.72	3052987.40	1562500
$\Sigma = 19915$	$\Sigma = 93500$	$\Sigma = 93499.99$	$\Sigma = 67480004.62$	$\Sigma = 104080000$

$$R^2 = 1 - \frac{67480004.62}{104080000} * 100\% = 35.165\%$$

## MATLAB Result for the $R^2$ Value:

For linear fitting:  $R^2 = 0.3517$

## 2.2 Quadratic Model

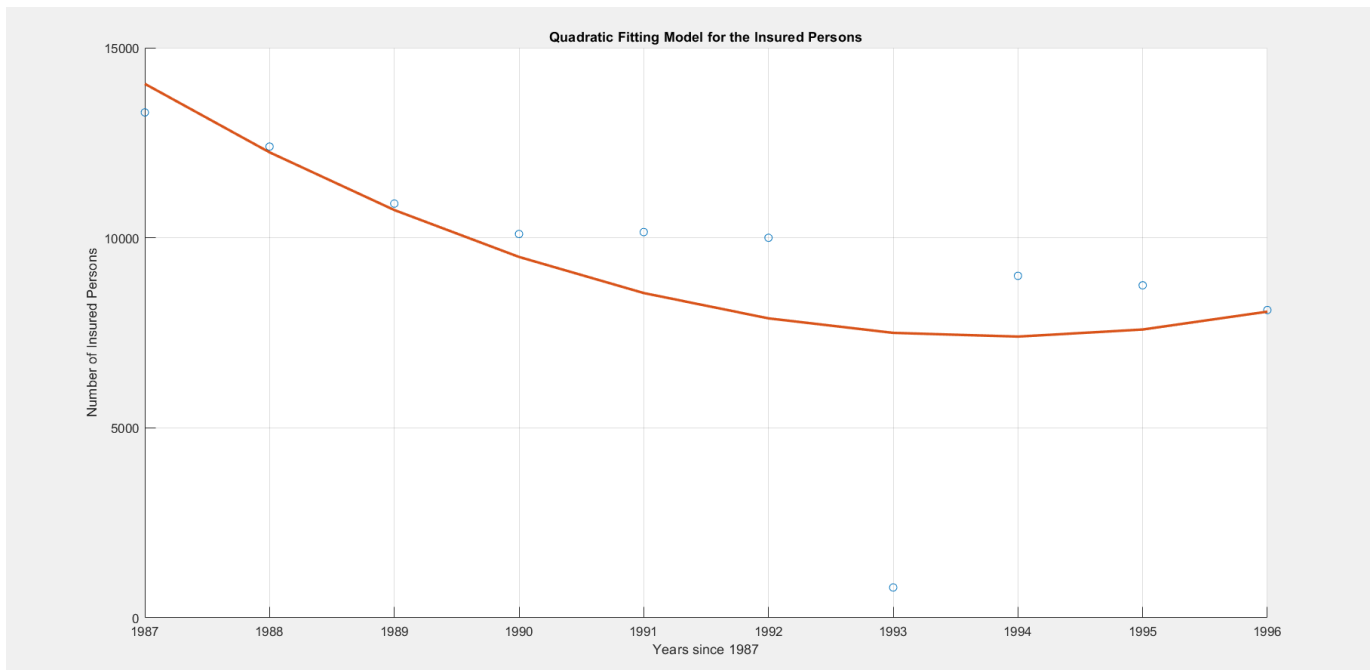


Figure 3: Quadratic Fitting Model for the Insured Persons

### MATLAB Result of the Quadratic Fitting Parameters:

The Quadratic fitting equation is:  $y = 142.05x^2 - 566433.11x + 564697173.27$

### MATLAB Result for the $R^2$ Value:

For Quadratic fitting:  $R^2 = 0.4540$

## 2.3 Cubic Model

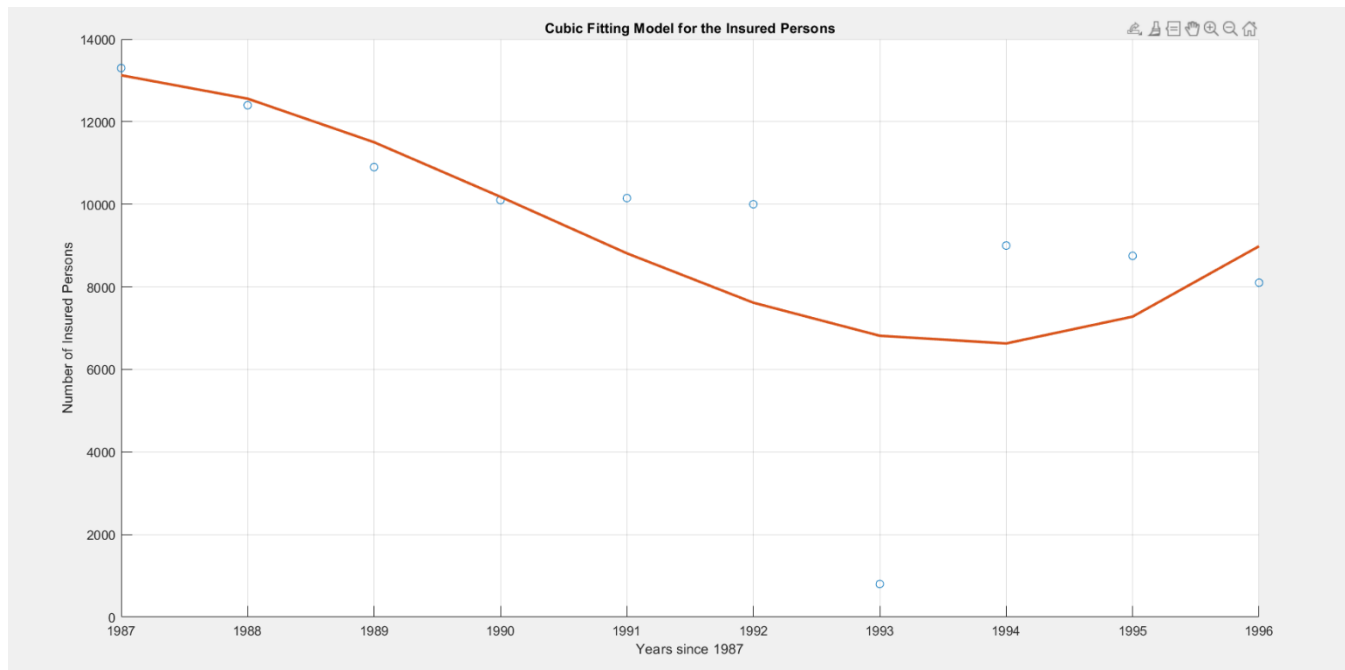


Figure 4: Cubic Fitting Model for the Insured Persons

### MATLAB Result of the Quadratic Fitting Parameters:

The Cubic fitting equation is:  $y = 36.77x^3 + -219549.65x^2 + 436949032.70x + -289871937657.24$

### MATLAB Result for the $R^2$ Value:

For Cubic fitting:  $R^2 = 0.4941$

## 2.4 The Three Models overlaid

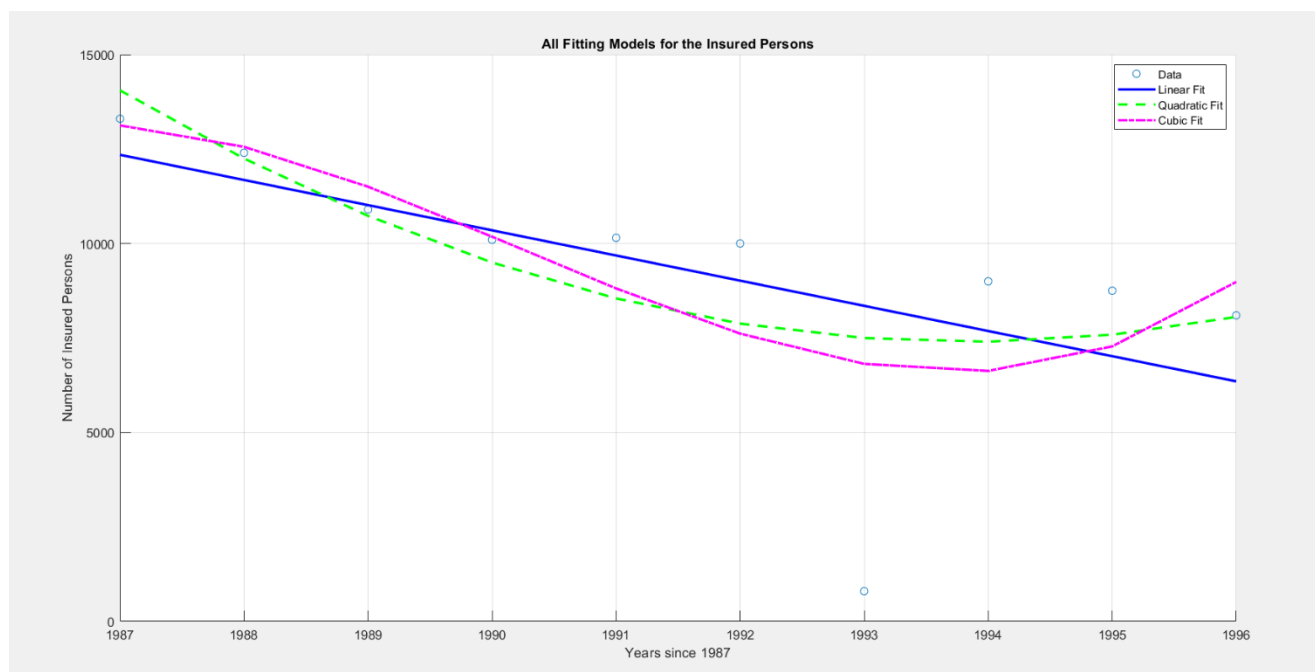


Figure 5: All the Three Fitting Model overlaid with the data

### Best Fit Model Function:

Model	Linear	Quadratic	Cubic
$R^2$	0.3516	0.454	0.4941
Parameters	2	3	4

By comparing the  $R^2$  values of the three models, it becomes clear that the **Quadratic Model** is the most suitable choice to be the best fit model. As its  $R^2$  value is only slightly lower than that of the Cubic Model, it offers a more balanced approach by maintaining a moderate number of parameters. This balance makes it a more efficient and practical option, as it provides a reasonable fit to the data without the added complexity of the Cubic Model.

### 3. Fitting After cleaning (After Removing the outlier)

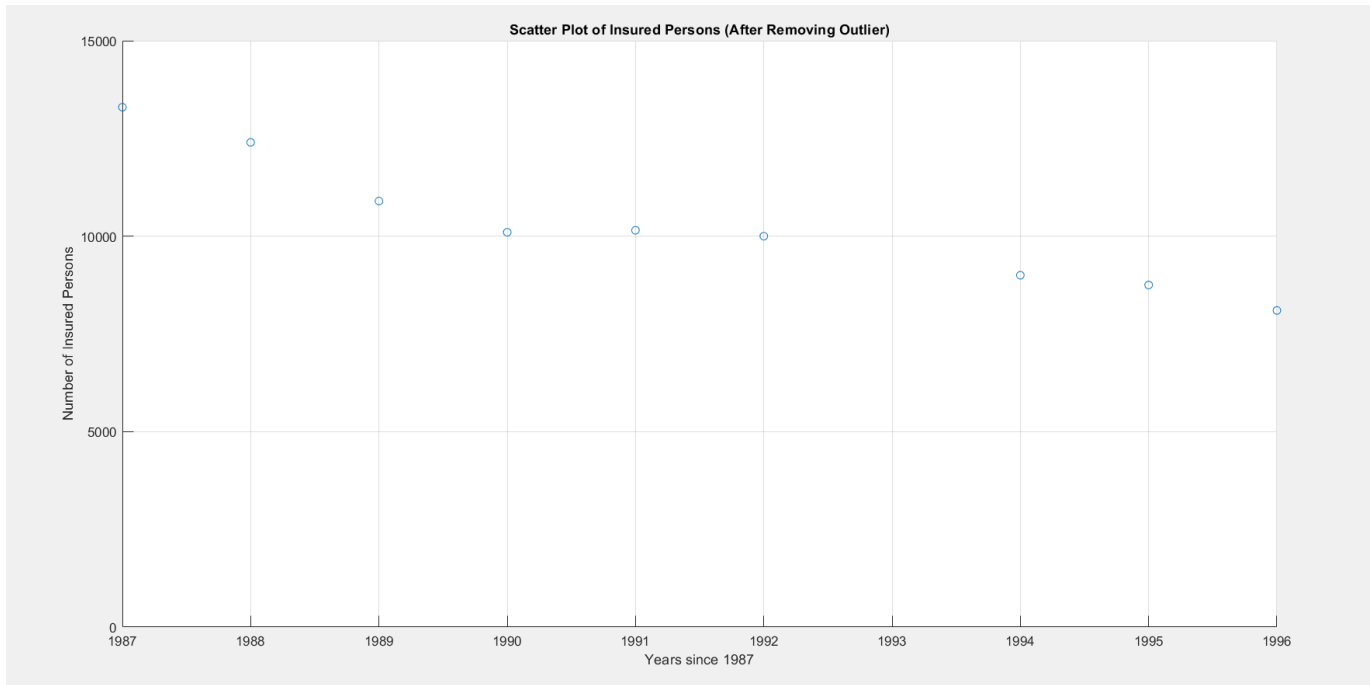


Figure 6: Scatterplot of given data after removing the outlier

The data point corresponding to the year 1993 was significantly lower than the other points on the graph, making it an outlier. To ensure more accurate and reasonable fitting curves, this outlier was excluded from the analysis as shown from Figure 6.

#### 3.1 Linear Model

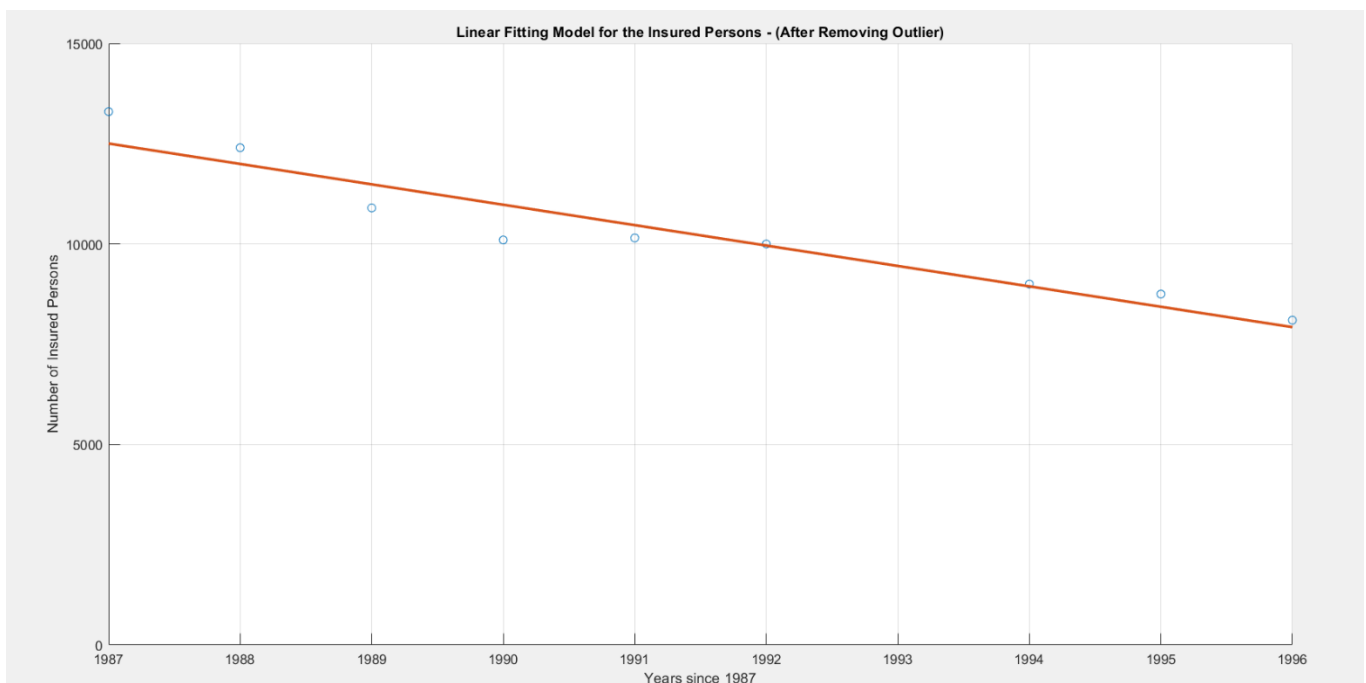


Figure 7: Linear Fitting Model for the Insured Persons after removing the outlier



## Hand Analysis for calculating the Linear Fitting Parameters:

For Linear module analysis. The prediction equation is given by:  $\hat{y}_i = \hat{\beta}_1 x_i + \hat{\beta}_0$

Where  $\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i - \frac{(\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n}}{(\sum_{i=1}^n x_i^2) - \frac{(\sum_{i=1}^n x_i)^2}{n}}$ , and  $\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$  ( $n = 9$ )

$x_i$	$y_i$	$x_i^2$	$y_i^2$	$x_i y_i$
1987	13300	3948169	176890000	26427100
1988	12400	3952144	153760000	24651200
1989	10900	3956121	118810000	21680100
1990	10100	3960100	102010000	20099000
1991	10150	3964081	103022500	20208650
1992	10000	3968064	100000000	19920000
1994	9000	3976036	81000000	17946000
1995	8750	3980025	76562500	17456250
1996	8100	3984016	65610000	16167600
$\Sigma = 17922$	$\Sigma = 92700$	$\Sigma = 35688756$	$\Sigma = 977665000$	$\Sigma = 184555900$

Then  $\hat{\beta}_1 = -508.75$ , ( $\bar{Y} = 10300$ ,  $\bar{X} = 1991.33$ )  $\rightarrow \hat{\beta}_0 = 1023389.138$

$$\hat{y}_i = -508.75x_i + 1023389.138$$

## MATLAB Result of the Linear Fitting Parameters:

The linear fitting equation is:  $y = -508.75x + 1023390.83$

## Hand Analysis for calculating $R^2$ Value:

$$R^2 = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}, \quad \bar{Y} = 10300, \quad n = 9, \quad \hat{Y}_i \text{ is calculated from the obtained equation}$$

$x_i$	$y_i$	$\hat{Y}_i$	$(Y_i - \hat{Y}_i)^2$	$(Y_i - \bar{Y})^2$
1987	13300	12502.89	635384.35	9000000
1988	12400	11994.14	164722.34	4410000
1989	10900	11485.39	342681.45	360000
1990	10100	10976.64	768497.69	40000
1991	10150	10467.89	101054.05	22500
1992	10000	9959.14	1669.54	90000
1994	9000	8941.64	3405.89	1690000
1995	8750	8432.89	100558.75	2402500
1996	8100	7924.14	30926.74	4840000
$\Sigma = 17922$	$\Sigma = 92700$	$\Sigma = 92684.76$	$\Sigma = 2148900.8$	$\Sigma = 22855000$

$$R^2 = 1 - \frac{2148900.8}{22855000} * 100\% = 90.598\%$$

### MATLAB Result for the $R^2$ Value:

For linear fitting:  $R^2 = 0.9060$

## 3.2 Quadratic Model

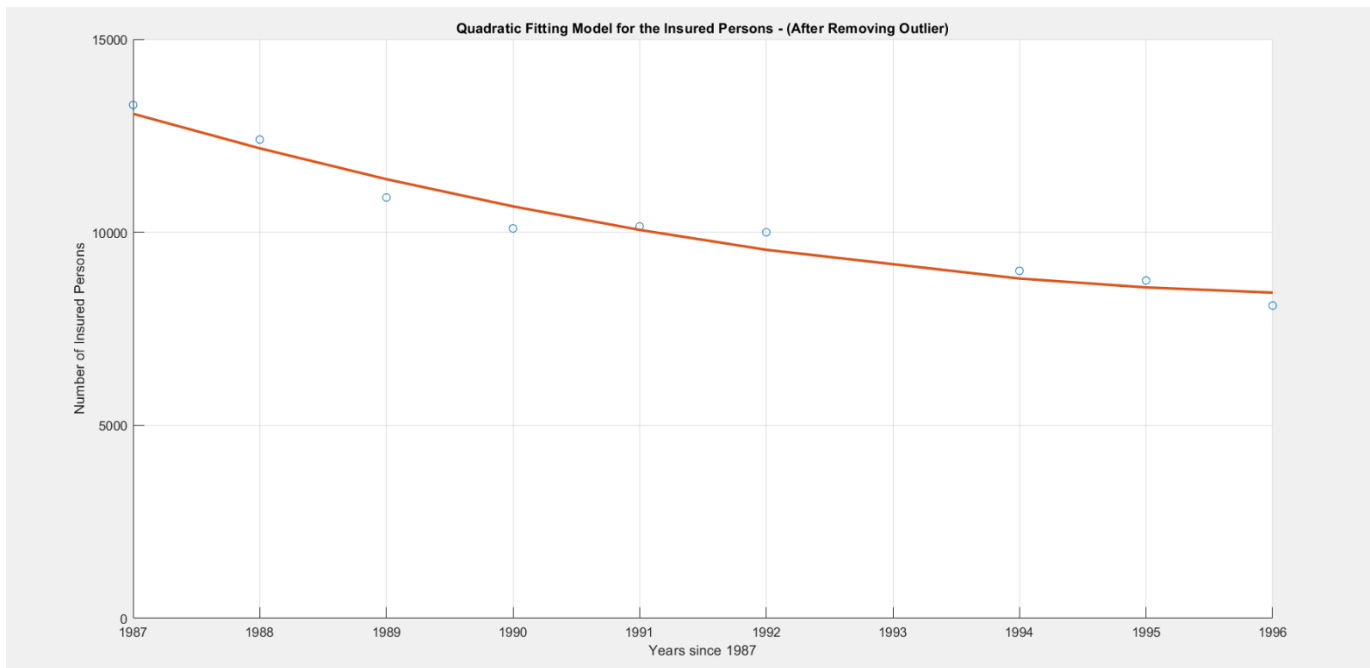


Figure 8: Quadratic Fitting Model for the Insured Persons after removing the outlier

### MATLAB Result of the Quadratic Fitting Parameters:

The Quadratic fitting equation is:  $y = 47.43x^2 + -189436.12x + 189153301.62$

### MATLAB Result for the $R^2$ Value:

For Quadratic fitting:  $R^2 = 0.9539$

### 3.3 Cubic Model

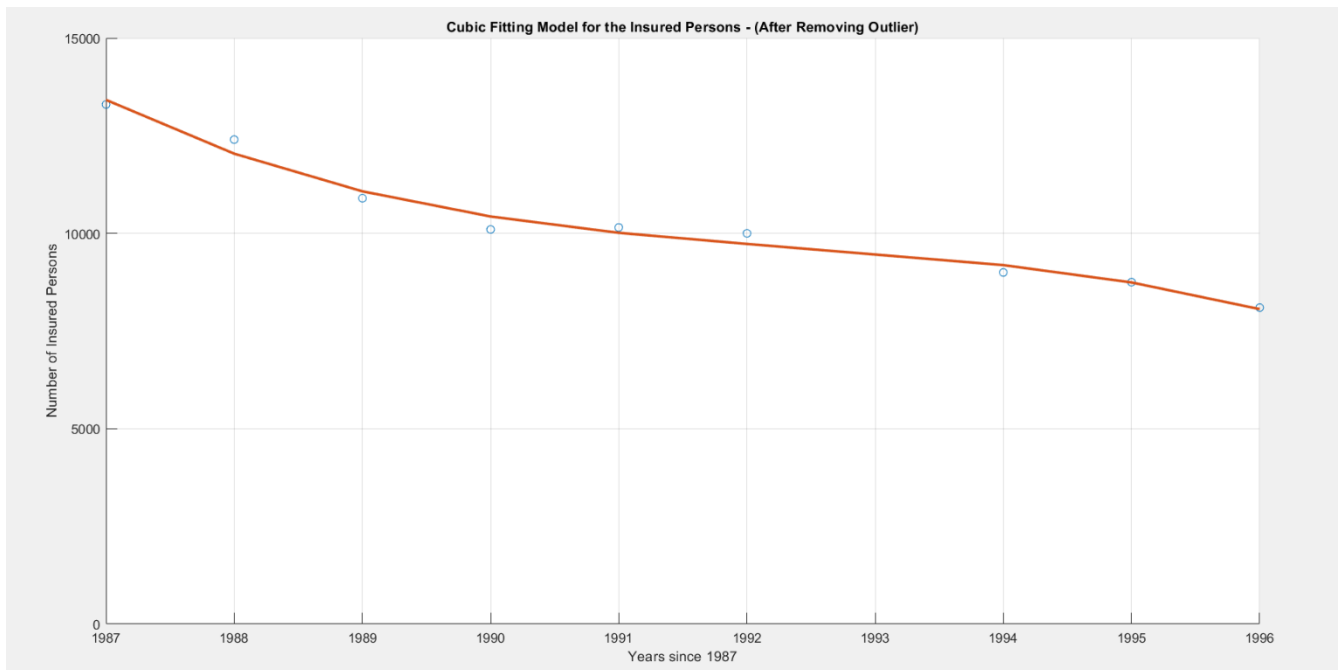


Figure 9: Cubic Fitting Model for the Insured Persons after removing the outlier

#### MATLAB Result of the Quadratic Fitting Parameters:

The Cubic fitting equation is:  $y = -15.53x^3 + 92810.09x^2 + -184917920.40x + 122812375430.57$

#### MATLAB Result for the $R^2$ Value:

For Cubic fitting:  $R^2 = 0.9819$

### 3.4 The Three Models overload

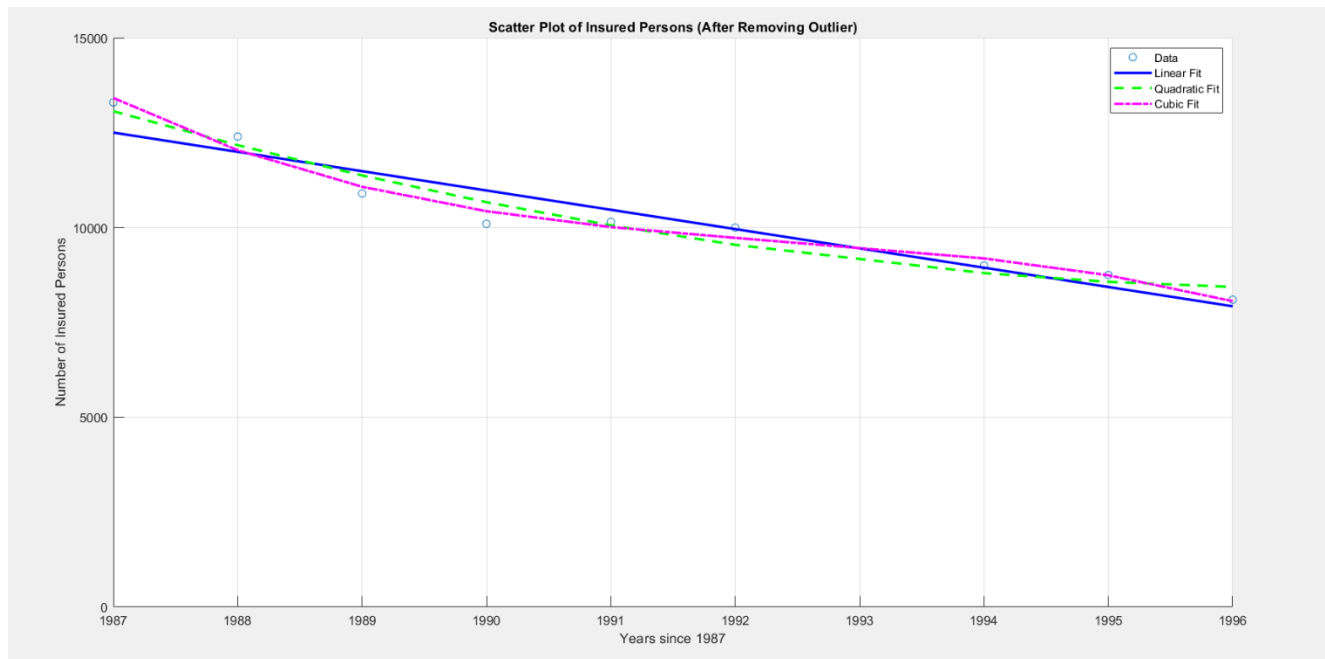


Figure 10: All the Three Fitting Model overlaid with the data after removing the outlier

#### Best Fit Model Function:

Model	Linear	Quadratic	Cubic
$R^2$	0.906	0.954	0.982
Parameters	2	3	4

By comparing the  $R^2$  values of the three models, it becomes clear that the **Quadratic Model** is the most suitable choice to be the best fit model. As its  $R^2$  value is high and it is only slightly lower than that of the Cubic Model, it offers a more balanced approach by maintaining a moderate number of parameters. This balance makes it a more efficient and practical option, as it provides a strong fit to the data without the added complexity of the Cubic Model.

#### 4. Graph the function of best fit model

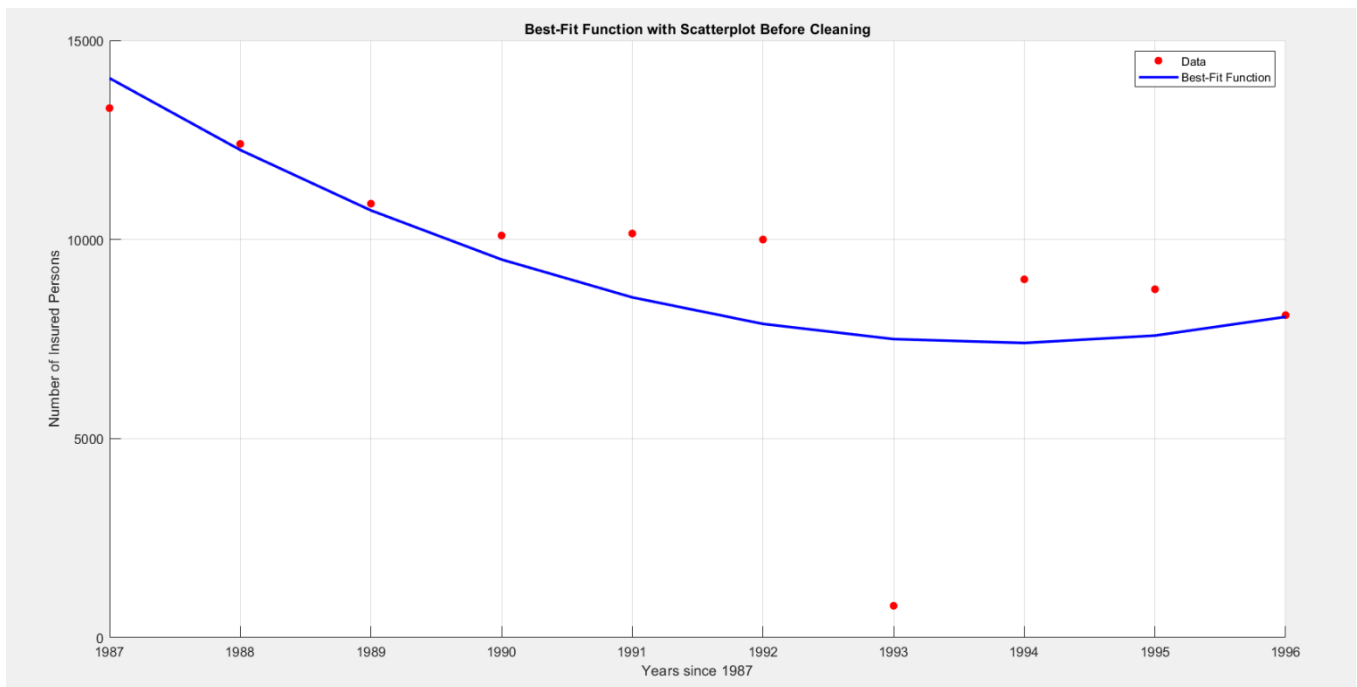


Figure 11: Best-Fit Function with Scatterplot of the data before removing the outlier

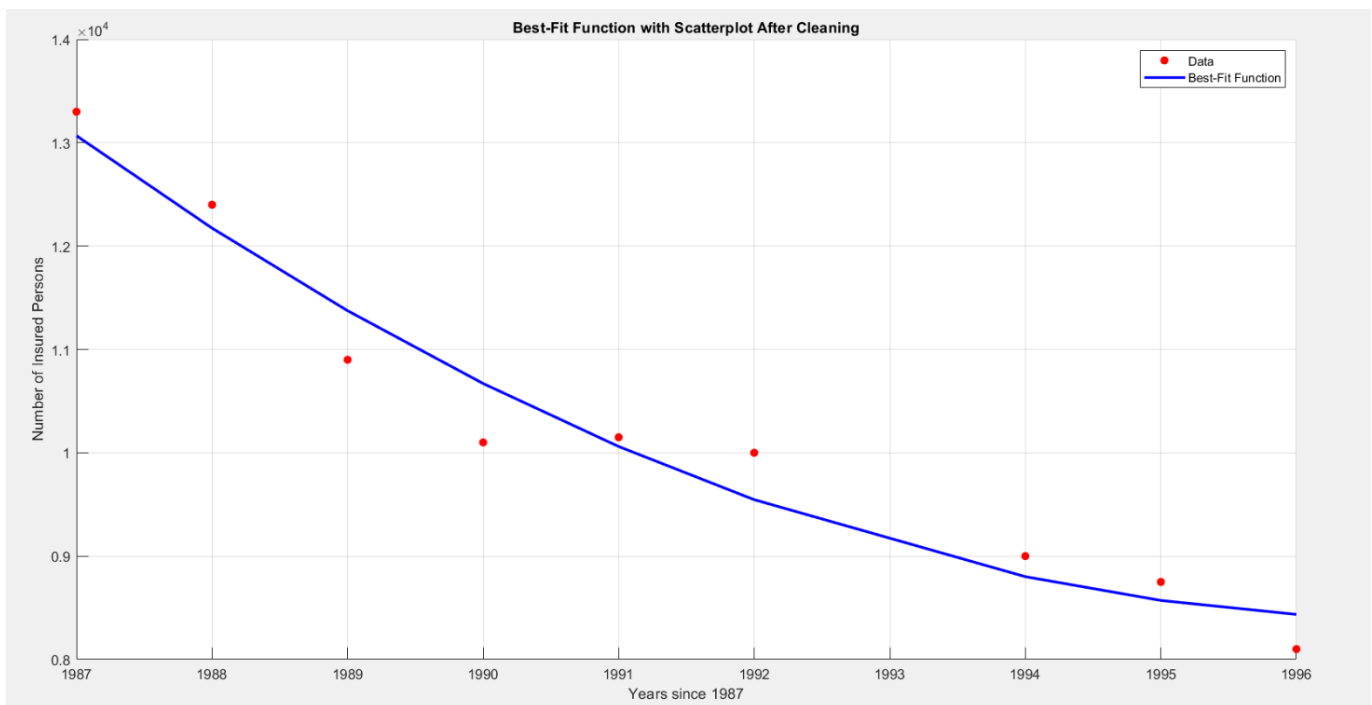


Figure 12: Best-Fit Function with Scatterplot of the data after removing the outlier

The best fit model in both cases is the Quadratic Model ( $R^2_{before\ cleaning} = 0.454$ ,  $R^2_{after\ cleaning} = 0.954$ )

## 5. Predict the average number of insured persons in 1997.

- Using the Build-in function in MATLAB (*polyval*)

```
% Predict for 1997
x_pred = 1997; % 1997
y_pred = polyval(best_p_no_outlier, x_pred);
fprintf('Predicted insured persons in 1997: %.2f\n', y_pred);
```

- Result

```
Predicted insured persons in 1997: 8395.37
```

## 6. Comparison between the Models before and after removing the outlier.

Model	Linear	Quadratic	Cubic
$R^2$ with Outlier	0.3516	0.454	0.4941
$R^2$ without Outlier	0.906	0.954	0.982

By examining the  $R^2$  values for the three models before and after outlier removal, we can observe a significant improvement in their accuracy. Initially, the presence of outliers negatively impacted the fitting process, making it unreliable and misleading, even when using more complex models with a larger number of parameters, such as the Cubic Model. Despite its increased complexity, the Cubic Model still produced an  $R^2$  value lower than 0.5, indicating that the model struggled to accurately represent the data due to the influence of the outlier.

However, after successfully identifying and removing the outlier, the accuracy of all models drastically improved, with each one achieving an  $R^2$  value greater than 0.9. This improvement was observed even in the simpler models, such as the Linear Model, which has the fewest parameters. The results highlight the crucial role of outlier detection and removal in ensuring that the fitted models provide a meaningful and accurate representation of the underlying data trends.

## Problem 2: Estimating Heating Oil

### 1. Plotting the given data

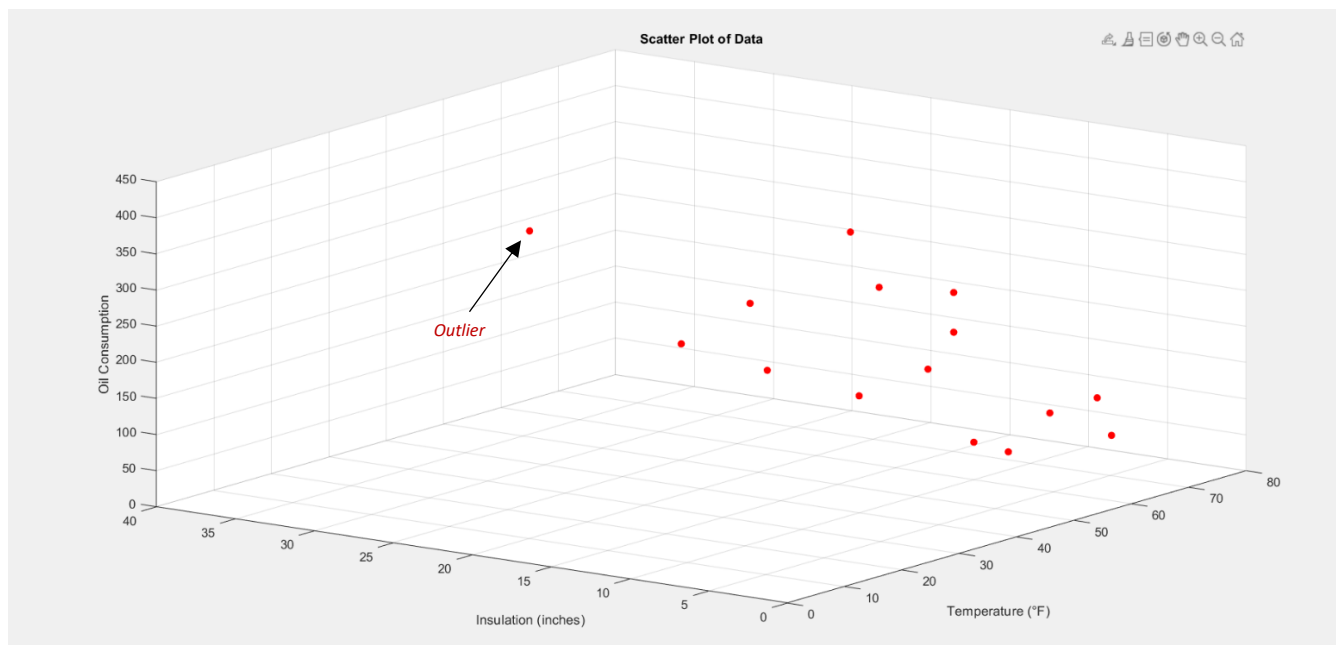


Figure 13: Scatterplot of given data

Figure 13 represents a three-dimensional plane graph that represents the heating oil consumption data for a single-family home during the month of January. This graph is based on two key independent variables: the average outdoor temperature (measured in degrees Fahrenheit) and the level of insulation (measured in inches). By analyzing the data distribution, we can observe that most data points follow a general trend; however, there is one particular point that stands out as unusual. Specifically, the data point corresponding to an oil consumption of **233 gallons**, an average temperature of **65°F**, and an insulation level of **40 inches** appears to deviate significantly from the expected pattern. This anomaly suggests an irregularity in the dataset, which is most likely the result of a data entry mistake rather than a genuine representation of heating oil usage. Consequently, this point can be classified as an outlier, potentially warranting further review or removal to ensure the accuracy of the regression analysis.

## 2. Fitting before cleaning (With the existence of outlier)

### 2.1 Linear Model

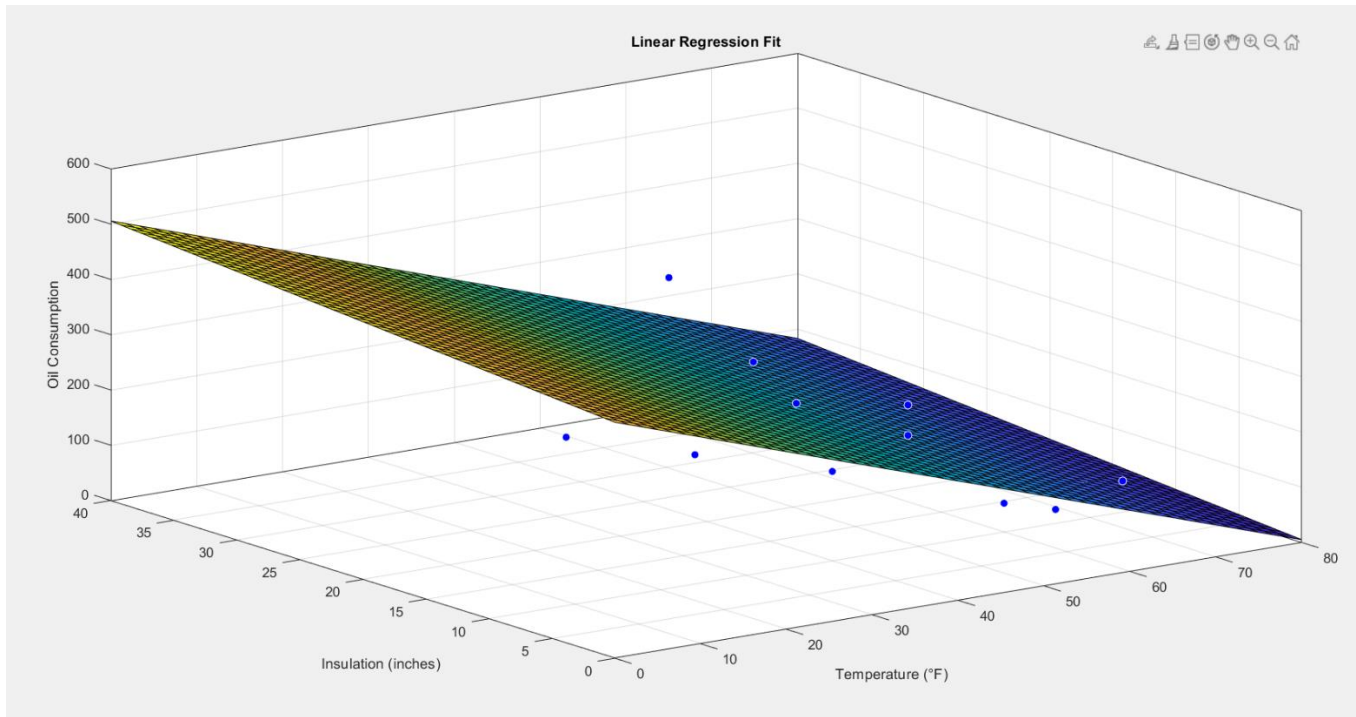


Figure 14: Linear Fitting Model for the Oil Consumption

### MATLAB Result of the Linear Fitting Parameters:

Coefficient for Temp : 426.97

Coefficient for Insulation: -5.27

Constant term : 1.97

Linear Fit Equation:  $\text{Oil} = 426.97 * \text{Temp} + -5.27 * \text{Insulation} + 1.97$

### MATLAB Result for the $R^2$ Value:

$R^2$  for Linear Model: 0.696686



## 2.2 Quadratic Model

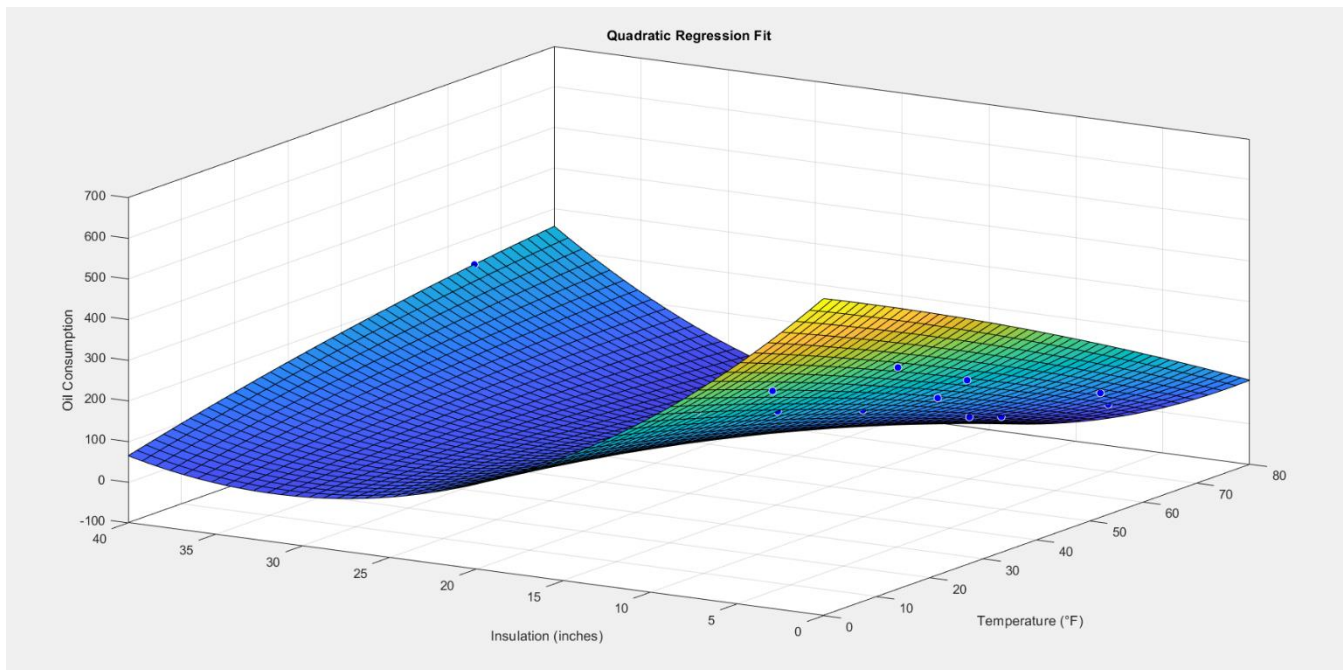


Figure 15: Quadratic Fitting Model for the Oil

### MATLAB Result of the Quadratic Fitting Parameters:

```
Coefficient for Temp^2      : 680.03
Coefficient for Insulation^2 : -6.17
Coefficient for Temp*Insulation: -41.72
Coefficient for Temp        : -0.01
Coefficient for Insulation   : 0.24
Constant term              : 0.66
Quadratic Fit Equation: Oil = 680.03 * Temp^2 + -6.17 * Insulation^2 + -41.72 * (Temp * Insulation)
                        + -0.01 * Temp + 0.24 * Insulation + 0.66
```

### MATLAB Result for the $R^2$ Value:

$R^2$  for Quadratic Model: 0.974152

### 3. Fitting After cleaning (After Removing the outlier)

#### 3.1 Linear Model

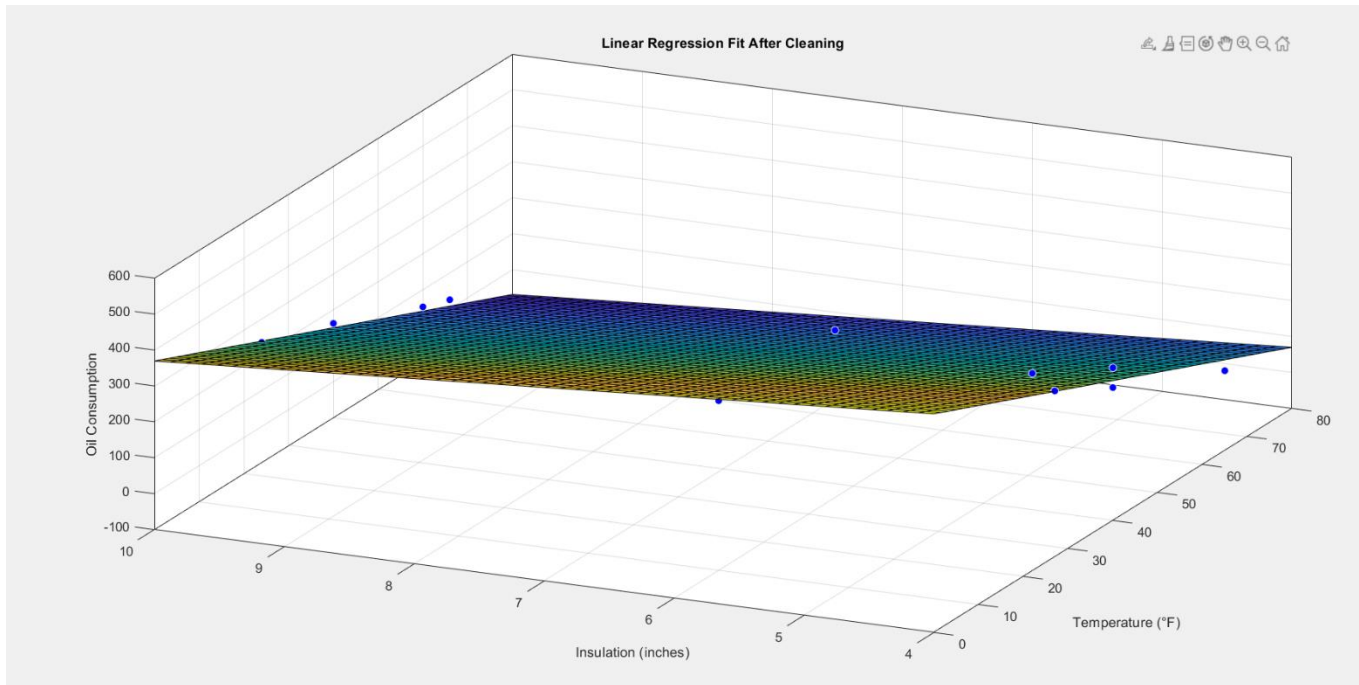


Figure 16: Linear Fitting Model for the Oil Consumption after removing the outlier

#### MATLAB Result of the Linear Fitting Parameters:

Coefficient for Temp : 601.60

Coefficient for Insulation: -5.48

Constant term : -23.23

Linear Fit Equation:  $\text{Oil} = 601.60 * \text{Temp} + -5.48 * \text{Insulation} + -23.23$

#### MATLAB Result for the $R^2$ Value:

$R^2$  for Linear Model: 0.960609

## 3.2 Quadratic Model

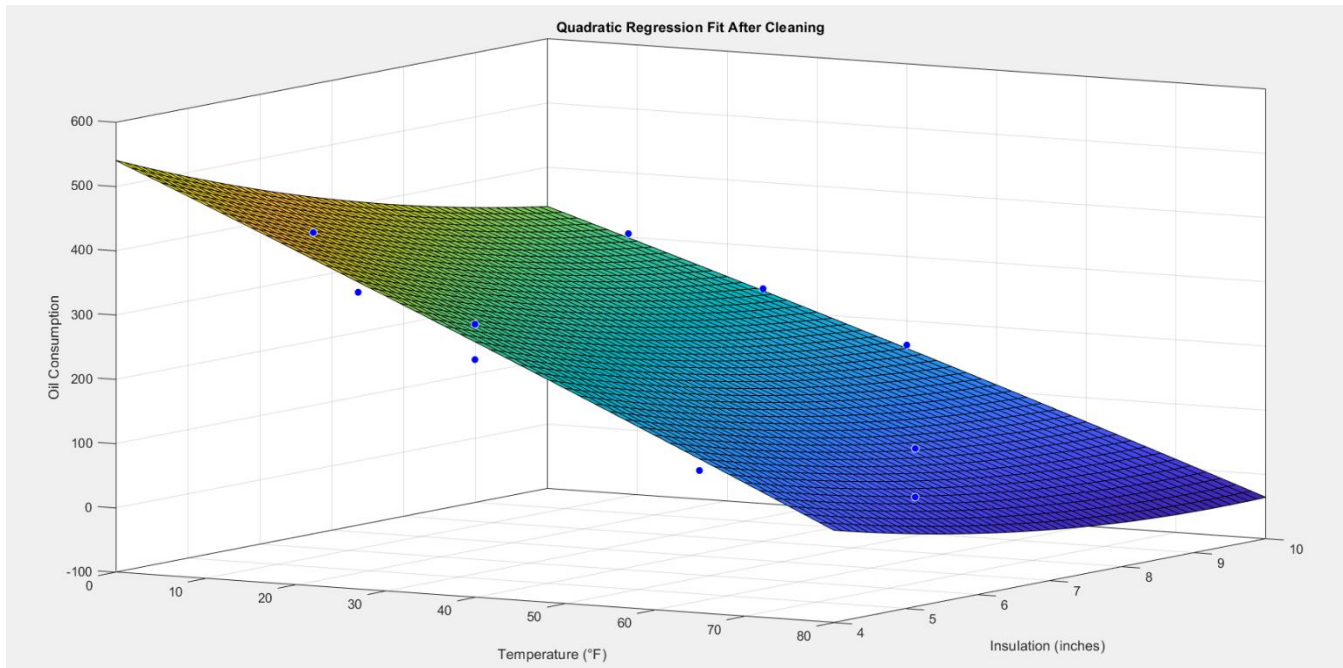


Figure 17: Quadratic Fitting Model for the Oil Consumption after removing the

### MATLAB Result of the Quadratic Fitting Parameters:

```
Coefficient for Temp^2      : 785.52
Coefficient for Insulation^2 : -6.96
Coefficient for Temp*Insulation: -72.16
Coefficient for Temp        : -0.00
Coefficient for Insulation   : 0.26
Constant term              : 2.74
Quadratic Fit Equation: Oil = 785.52 * Temp^2 + -6.96 * Insulation^2 + -72.16 * (Temp * Insulation)
                          + -0.00 * Temp + 0.26 * Insulation + 2.74
```

### MATLAB Result for the $R^2$ Value:

```
R^2 for Quadratic Model: 0.976889
```

#### 4. Graph the function of best fit model

##### Before Cleaning:

Model	Linear	Quadratic
$R^2$	0.6967	<b>0.9742</b>
Parameters	2	<b>3</b>

By comparing the  $R^2$  values of the two models, it becomes clear that the **Quadratic Model** is the most suitable choice to be the best fit model. As its  $R^2$  value is much higher than that of the Linear Model indicating better fitting model.

##### After Cleaning:

Model	Linear	Quadratic
$R^2$	<b>0.9606</b>	0.9769
Parameters	<b>2</b>	3

By comparing the  $R^2$  values of the two models, it becomes clear that the **Linear Model** is the most suitable choice to be the best fit model. As its  $R^2$  value is high and it is only slightly lower than that of the Quadratic Model, it offers a more balanced approach by maintaining a low number of parameters. This balance makes it a more efficient and practical option, as it provides a strong fit to the data without the added complexity of the Quadratic Model.

#### 5. Predict the needed oil if the temperature is 15 Fahrenheit and the insulation is 5 attic insulations inches.

##### - MATLAB Code

```
% Predict the needed oil for temperature is 15 Fahrenheit and the insulation is 5 attic
predicted_oil_linear_cleaned = linearModel_cleaned(15,5);
predicted_oil_quad_cleaned = quadraticModel_cleaned(15,5);
fprintf('\n\nPredicted needed oil for temp = 15 F and insulation = 5 attic:\n');
fprintf('Linear Cleaned: %f\n', predicted_oil_linear_cleaned);
fprintf('Quadratic Cleaned: %f\n\n', predicted_oil_quad_cleaned);
```

##### - Result

```
Predicted needed oil for temp = 10 F and insulation = 5 attic:
Linear Cleaned: 403.215051
Quadratic Cleaned: 407.351114
```

## 6. Comparison between the Models before and after removing the outlier.

Model	Linear	Quadratic
$R^2$ with Outlier	0.6967	0.9742
$R^2$ without Outlier	0.9606	0.9769

By carefully analyzing the  $R^2$  values of both models before and after the outlier was removed, we can observe a significant enhancement in the accuracy of the Linear Model, whereas the Quadratic Model exhibits only a minor change. This difference in behavior can be attributed to the fact that the Linear Model is inherently the most appropriate and best-fit model for this particular dataset.

Before outlier removal, the Linear Model struggled to capture the underlying pattern effectively due to the strong influence of the outlier, leading to a considerably lower  $R^2$  value. However, once the outlier was detected and eliminated, the Linear Model experienced a dramatic improvement in its predictive accuracy, achieving an  $R^2$  value exceeding 0.95, which signifies an excellent fit.

On the other hand, the Quadratic Model maintained a high  $R^2$  value both before and after the outlier was removed. This is because the presence of the outlier initially made the dataset resemble a quadratic trend, allowing the Quadratic Model to achieve a relatively high  $R^2$  value, even when the outlier was included. As a result, the impact of the outlier removal on the Quadratic Model was minimal, as it was already able to fit the data well in both cases. This further confirms that while the Quadratic Model may provide a reasonable fit, the Linear Model is the true best-fit model once the dataset is properly cleaned from misleading points.

## ***Part 2: Some classical ML modeling***

### **Introduction:**

#### **DCT Features:**

- Discrete Cosine Transform (DCT) features are extracted by transforming an image from the spatial domain to the frequency domain, where most important information is concentrated in a few low-frequency coefficients. For digit classification in ReducedMNIST, DCT helps by reducing redundancy and emphasizing dominant frequency components while discarding less significant ones, leading to a more compact representation with 225 dimensions. These features are widely used in image compression and recognition tasks, as they help improve classification performance while reducing computational complexity.

#### **PCA Features:**

- Principal Component Analysis (PCA) features are obtained by linearly transforming high-dimensional data into a lower-dimensional space while preserving as much variance as possible. In this assignment, PCA is applied to ReducedMNIST images to reduce dimensionality while ensuring at least 95% of the total variance is retained, allowing for an efficient and noise-robust representation. This transformation helps classifiers focus on the most informative variations in the dataset, improving performance and reducing overfitting while making computations more efficient. It is found that 152 components represent the 95% variance from the 784 dimensions as shown in (Figure #)

## Results:

### K-means Clustering:

- DCT Results:

```
Training Using DCT features
```

```
Training with 1 clusters per class...
```

```
For 1 clusters per class:
```

```
Accuracy: 80.50%
```

```
Training time: 0.24 seconds
```

```
Testing time: 0.03 seconds
```

```
Total processing time: 0.27 seconds
```

```
Training with 4 clusters per class...
```

```
For 4 clusters per class:
```

```
Accuracy: 87.10%
```

```
Training time: 0.13 seconds
```

```
Testing time: 0.03 seconds
```

```
Total processing time: 0.17 seconds
```

```
Training with 16 clusters per class...
```

```
For 16 clusters per class:
```

```
Accuracy: 91.95%
```

```
Training time: 0.21 seconds
```

```
Testing time: 0.08 seconds
```

```
Total processing time: 0.29 seconds
```

```
Training with 32 clusters per class...
```

```
For 32 clusters per class:
```

```
Accuracy: 92.55%
```

```
Training time: 0.30 seconds
```

```
Testing time: 0.14 seconds
```

```
Total processing time: 0.44 seconds
```

```
Best model: 32 clusters per class with 92.55% accuracy
```

## Results Summary (DCT Features):

Clusters	Accuracy (%)	Processing Time (seconds)
1	80.50	0.27
4	87.10	0.17
16	91.95	0.29
32	92.55	0.44

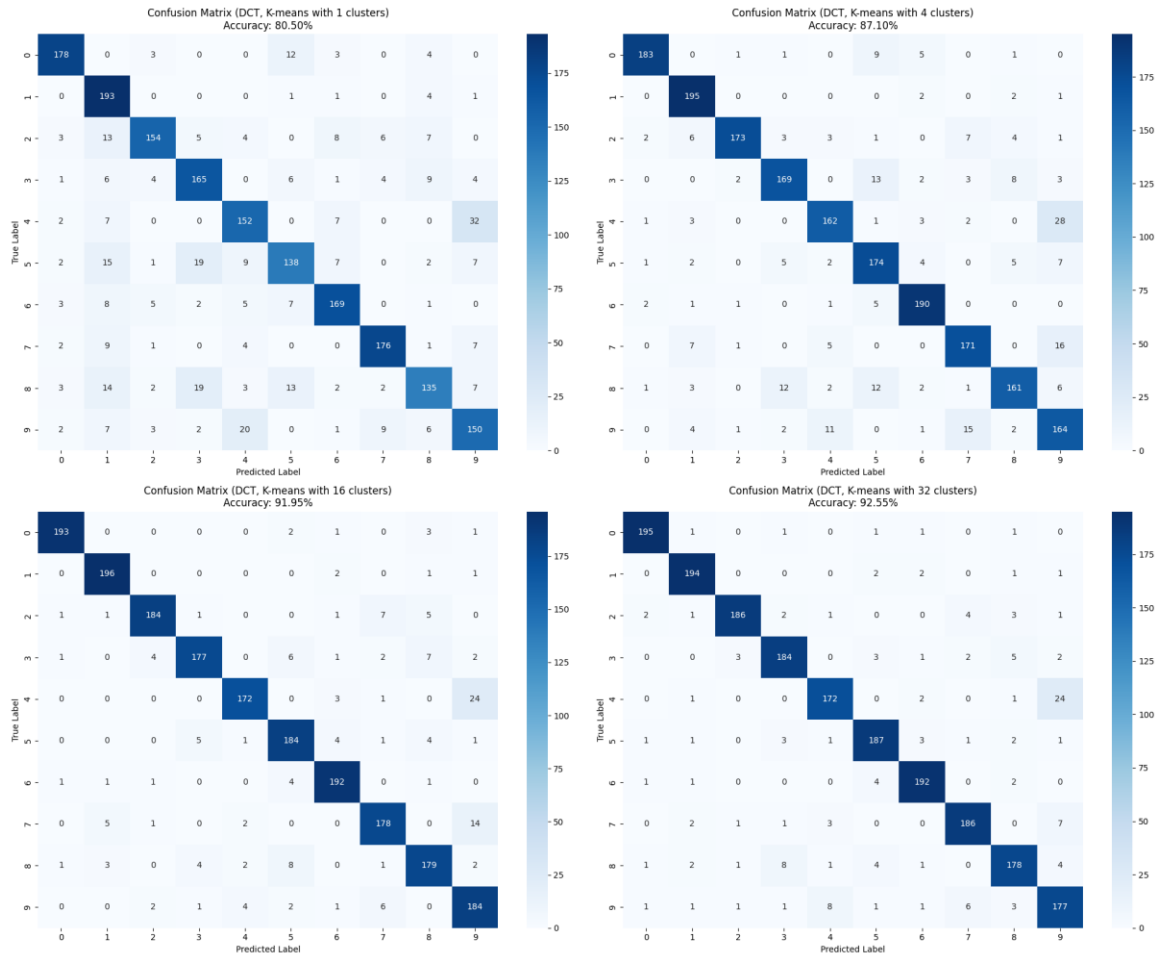


Figure 18: DCT, K-means Simulation Results



## - PCA Results:

### Training Using PCA features

Training with 1 clusters per class...

For 1 clusters per class:

Accuracy: 80.55%

Training time: 0.24 seconds

Testing time: 0.02 seconds

Total processing time: 0.26 seconds

Training with 4 clusters per class...

For 4 clusters per class:

Accuracy: 87.40%

Training time: 0.10 seconds

Testing time: 0.03 seconds

Total processing time: 0.13 seconds

Training with 16 clusters per class...

For 16 clusters per class:

Accuracy: 92.35%

Training time: 0.20 seconds

Testing time: 0.07 seconds

Total processing time: 0.27 seconds

Training with 32 clusters per class...

For 32 clusters per class:

Accuracy: 92.65%

Training time: 0.34 seconds

Testing time: 0.11 seconds

Total processing time: 0.45 seconds

Best model: 32 clusters per class with 92.65% accuracy

### Results Summary (PCA Features):

-----			
Clusters	Accuracy (%)	Processing Time (seconds)	
-----			
1	80.55	0.26	
4	87.40	0.13	
16	92.35	0.27	
32	92.65	0.45	
-----			

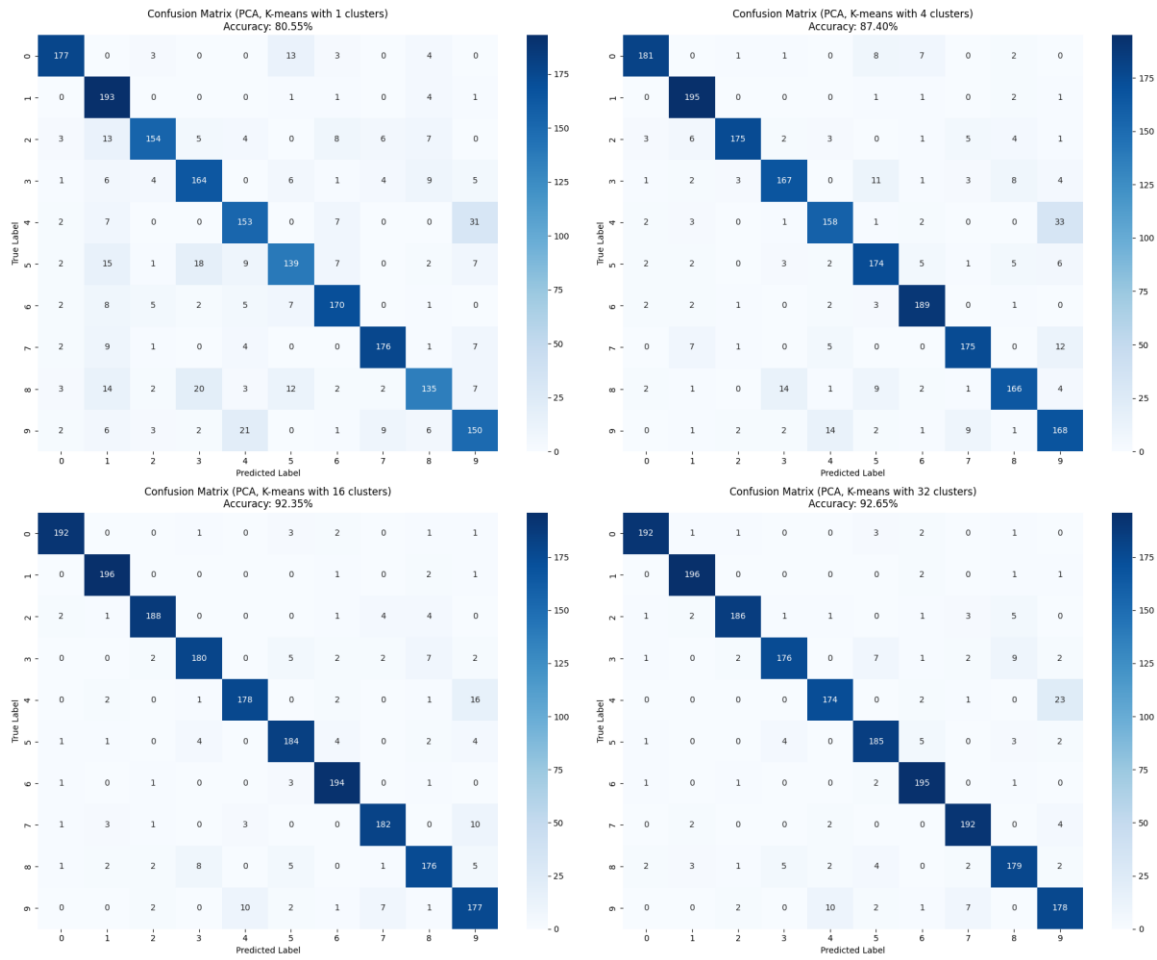


Figure 19: PCA, K-means Simulation Results

## SVM:

### - DCT Results:

Training Using DCT features with SVM classifiers

Training with linear kernel...

For linear kernel SVM:

Accuracy: 91.00%

Training time: 1.09 seconds

Testing time: 0.22 seconds

Total processing time: 1.31 seconds

Training with rbf kernel...

For rbf kernel SVM:

Accuracy: 95.55%

Training time: 1.33 seconds

Testing time: 1.22 seconds

Total processing time: 2.55 seconds

Best model: SVM with rbf kernel - 95.55% accuracy

Results Summary (DCT Features with SVM):

-----			
Kernel	Accuracy (%)	Processing Time (seconds)	
-----			
linear	91.00	1.31	
rbf	95.55	2.55	
-----			

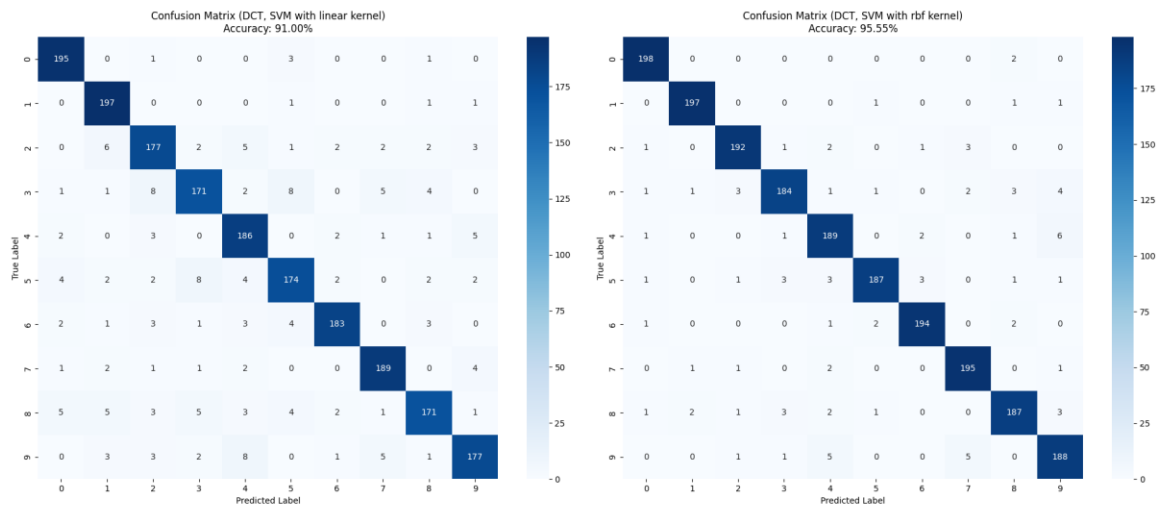


Figure 20: DCT, SVM Simulation Results

## - PCA Results:

```
Extracting PCA features...
Number of components needed for 95.0% variance: 152
PCA features shape - Train: (10000, 152), Test: (2000, 152)
PCA features extraction completed and saved.
```

## Training Using PCA features with SVM classifiers

### Training with linear kernel...

For linear kernel SVM:

Accuracy: 90.55%

Training time: 0.94 seconds

Testing time: 0.17 seconds

Total processing time: 1.11 seconds

### Training with rbf kernel...

For rbf kernel SVM:

Accuracy: 96.45%

Training time: 1.41 seconds

Testing time: 1.02 seconds

Total processing time: 2.43 seconds

Best model: SVM with rbf kernel - 96.45% accuracy

Results Summary (PCA Features with SVM):		
Kernel	Accuracy (%)	Processing Time (seconds)
linear	90.55	1.11
rbf	96.45	2.43

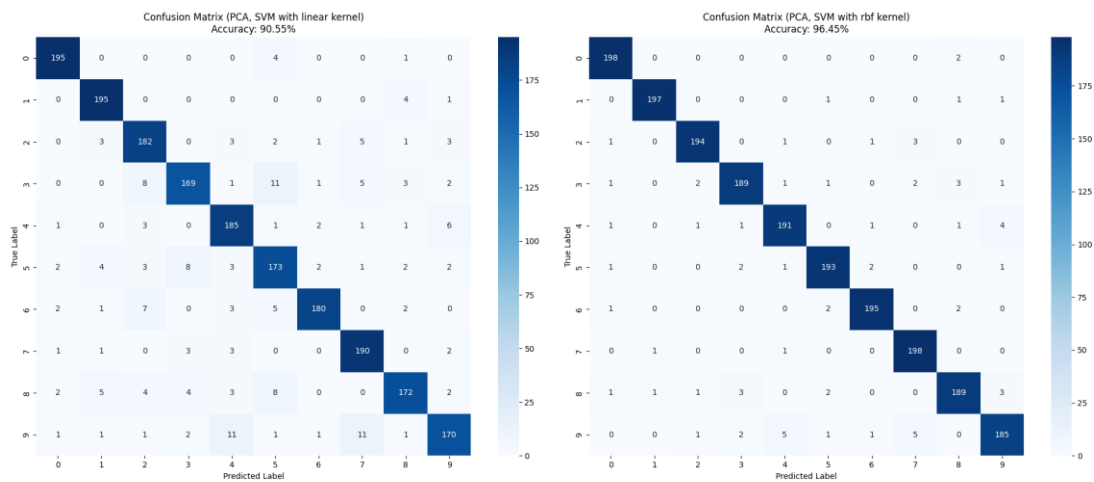


Figure 21: PCA, SVM Simulation Results

Results Summary Table:

Classifier		Features			
		DCT		PCA	
		Accuracy	Processing Time	Accuracy	Processing Time
K-means Clustering	1	80.50%	0.27s	80.55%	0.26s
	4	87.10%	0.17s	87.40%	0.13s
	16	91.95%	0.29s	92.35%	0.27s
	32	92.55%	0.44s	<b>92.65%</b>	<b>0.45s</b>
SVM	Linear	91.00%	1.31s	90.55%	1.11s
	Nonlinear (RBF)	95.55%	2.55s	<b>96.45%</b>	<b>2.43s</b>

**Note:** The non-linear SVM kernel we used is the **Radial Basis-function kernel (RBF)**.

The best model for each classifier is:

### 1. K-means Clustering:

- The highest accuracy is **92.65%** with PCA features and 32 clusters.
- Processing time is **0.45s**, which is reasonable.
- Best choice: **K-means with 32 clusters using PCA features.**

### 2. SVM:

- The highest accuracy is **96.45%** with PCA features and Nonlinear (RBF) kernel.
- Processing time is **2.43s**, which is longer but acceptable for higher accuracy.
- Best choice: **SVM with Nonlinear (RBF) kernel using PCA features.**

## Confusion matrices:

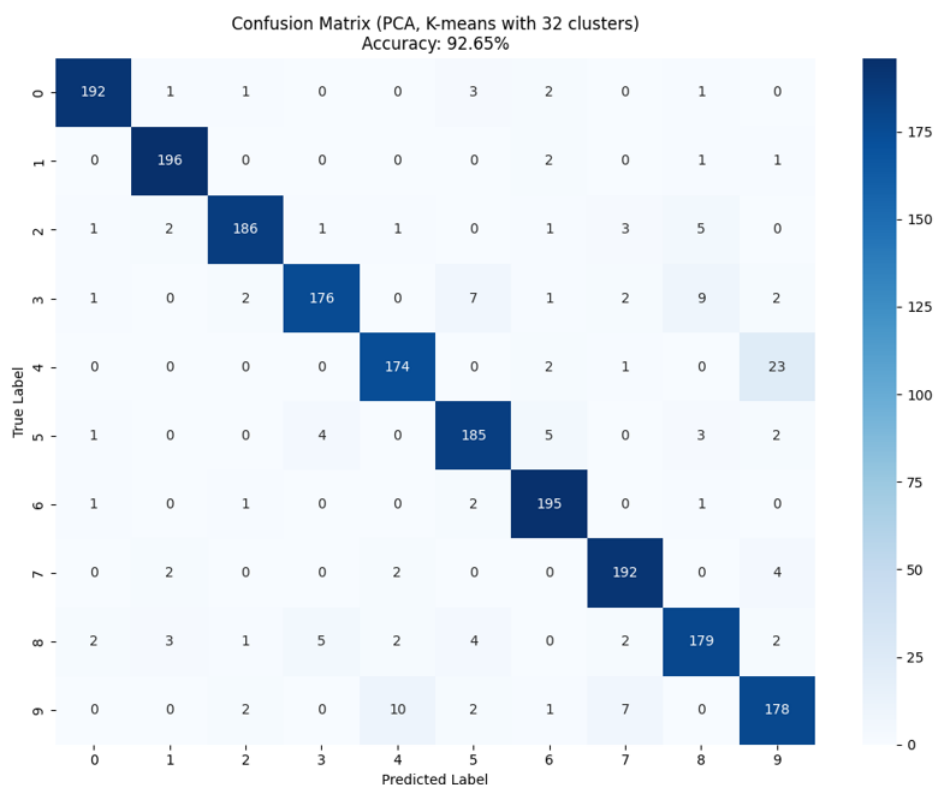


Figure 22: PCA, K-means with 32 clusters

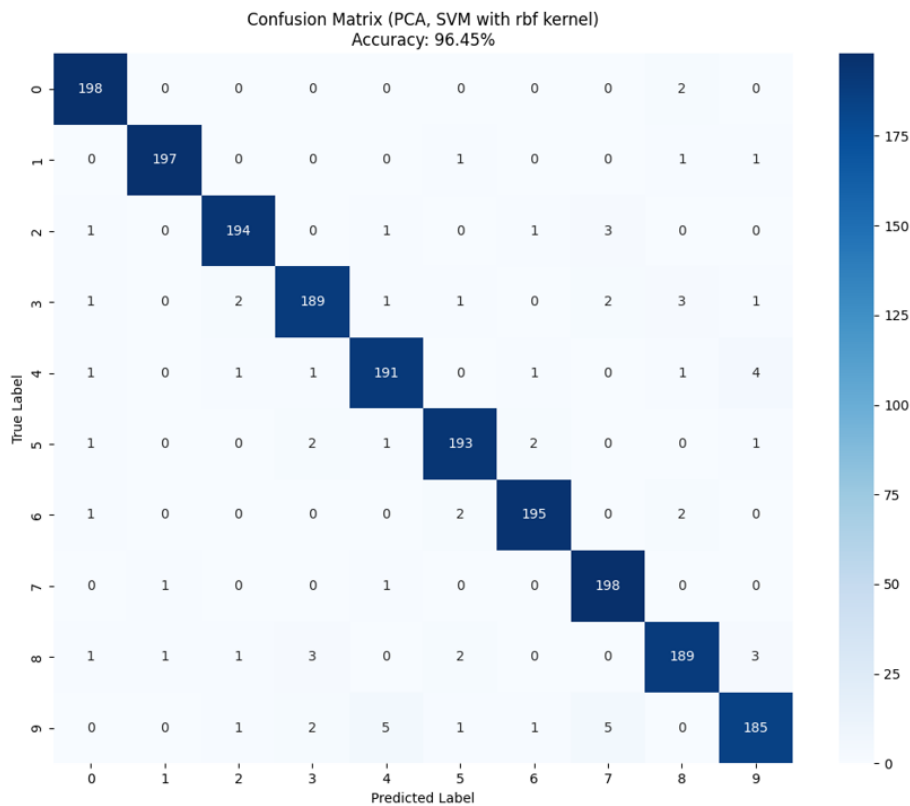


Figure 23: PCA, SVM with rbf kernel

## Conclusions:

Upon close inspection of the confusion matrices for the SVM, and K-means Clustering with small clusters per class, for both the DCT and PCA features, it can be noticed that large portion of the inaccuracies arise due to confusion between the class of digit 4 and class of digit 9. This makes some sense since the drawing of both numbers closely resemble each other sometimes which leads to an overlap between the two classes. Therefore, clustering with small values for clusters per class does not manage to capture the non-linearities in the decision boundaries required between the classes. This is the case with linear SVM models as well. Therefore, larger K for clustering and non-linear SVM models manage to achieve better accuracy due to capturing the non-linearities between the classes better.

It is important to note that the 225 DCT coefficients and 95% variance PCA components “which represent 152 components of the 784 dimensions” almost capture the same amount of required information to describe the images which is reflected in the almost equal accuracies obtained by training our models based on these features.

## Part 3

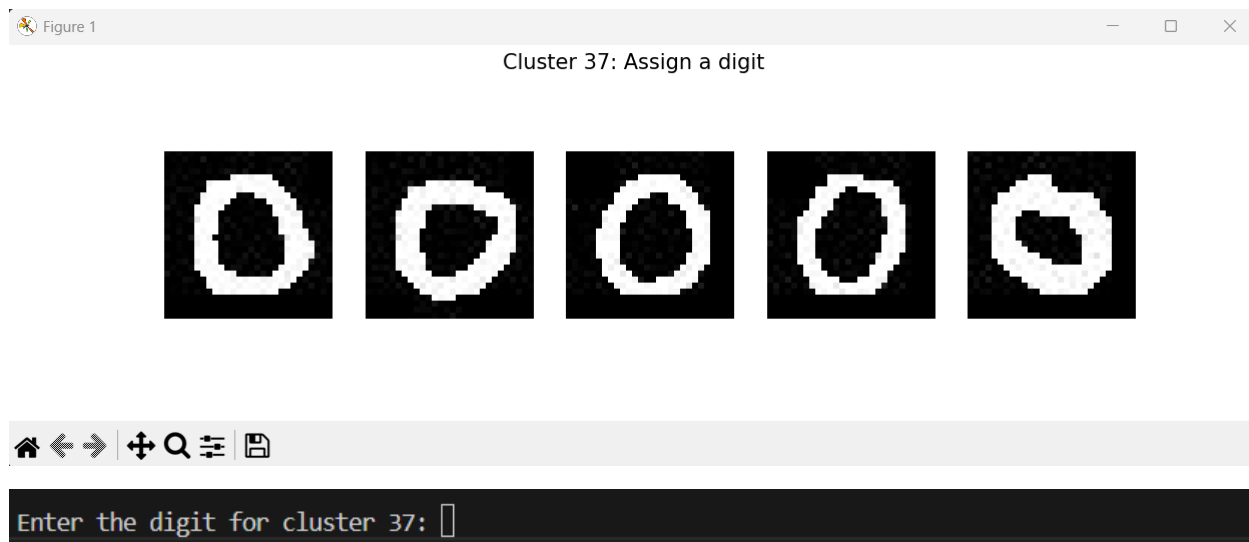
### Pipeline1: Labelling Using Clustering and SVM

#### 1. Data Preparation:

- Load training and testing images from directories.
- Extract labels from folder names (each folder represents a digit).
- Form flatten images from 28x28 to 784 (1D vector) as a feature.

#### 2. K-Means Clustering for Initial Label Assignment:

- Apply **K-Means clustering** to group training images into **100** clusters.
- Make the user assign labels to each cluster:
  - Pick **5** random samples per cluster and show them in one tap to act as one image in assigning time for more time efficiency.
  - User assign a number from 0 to 9.
- **(OR)** Assign labels to each cluster using a majority-voting approach **(for simulation only)**:
  - Pick **5** random samples per cluster.
  - Assign the most frequent label among them to the entire cluster to act as manual assignment.



#### 3. Train SVM on Clustered Labels:

- **Train** an **SVM classifier** using the features (flatten images) and the cluster-based labels.
- **Predict** the training and testing sets **by the trained SVM**.
- Evaluate SVM accuracy on the training set and on the testing set.



#### 4. We go through 2 approaches:

##### A. manually assign labels for the uncertain samples and iterate for annotation accuracy

- Compare SVM predictions with cluster-based labels.
- Identify images where **SVM disagrees with clustering**
- Calculate percentage of images with changed labels.
- Make the user assign labels to each uncertain image
- **(OR)** For each uncertain image, update its label using correct labels to act as manual assignment **(for simulation only)**.
- Retrain SVM on the updated labels.
- Repeat the process for **5 iterations**:
  - Identify uncertain images.
  - Correct their labels.
  - Retrain SVM.
  - Measure accuracy improvement.

Number of training images: 10000 Number of testing images: 2000 Shape of X_train: (10000, 28, 28) Shape of X_test: (2000, 28, 28) Shape of X_train_flattened: (10000,	Enter the digit for cluster 4: 1 Enter the digit for cluster 99: 2 Enter the digit for cluster 77: 1 Enter the digit for cluster 90: 1 Enter the digit for cluster 76: 1 Enter the digit for cluster 7: 1 Enter the digit for cluster 72: 1 Enter the digit for cluster 52: 8 Enter the digit for cluster 59: 2 Enter the digit for cluster 61: 6 Enter the digit for cluster 87: 2 Enter the digit for cluster 28: 2 Enter the digit for cluster 51: 6 Enter the digit for cluster 0: 3 Enter the digit for cluster 49: 5 Enter the digit for cluster 48: 5 Enter the digit for cluster 55: 2 Enter the digit for cluster 43: 2 Enter the digit for cluster 27: 2	Enter the digit for cluster 45: 6 Enter the digit for cluster 75: 7 Enter the digit for cluster 89: 7 Enter the digit for cluster 70: 7 Labeling Accuracy: 86.88% SVM Training Accuracy: 88.12% SVM Test Accuracy: 91.35% Percentage of images that changed clusters: 2.35% Found 235 uncertain images. Now retraining SVM... Updated SVM Training Accuracy: 88.90% Updated SVM Test Accuracy: 91.75% Percentage of images that changed labels: 1.21% Found 121 uncertain images. Now retraining SVM... Updated SVM Training Accuracy: 89.10% Updated SVM Test Accuracy: 91.75% Percentage of images that changed labels: 0.83% Found 83 uncertain images. Now retraining SVM... Updated SVM Training Accuracy: 89.27% Updated SVM Test Accuracy: 92.05% Percentage of images that changed labels: 0.74% Found 74 uncertain images. Now retraining SVM... Updated SVM Training Accuracy: 89.38% Updated SVM Test Accuracy: 91.95% Percentage of images that changed labels: 0.69% Found 69 uncertain images. Now retraining SVM... Updated SVM Training Accuracy: 89.43% Updated SVM Test Accuracy: 92.05% Percentage of images that changed labels: 0.69% Found 69 uncertain images. Total Execution Time: 384.31 seconds
---	--	--

##### B. iterate on SVM training and predicting for annotation accuracy.

- Retrain SVM on the updated labels.
- Repeat the process for **5 iterations**:
  - Identify uncertain images.
  - Correct their labels.
  - Retrain SVM.

- Measure accuracy improvement.

```

Number of training images: 10000
Number of testing images: 2000
Shape of X_train: (10000, 28, 28)
Shape of X_test: (2000, 28, 28)
Shape of X_train_flattened: (10000, 784)

Enter the digit for cluster 56: 0
Enter the digit for cluster 25: 0
Enter the digit for cluster 37: 0
Enter the digit for cluster 2: 6
Enter the digit for cluster 57: 0
Enter the digit for cluster 19: 0
Enter the digit for cluster 62: 0
Enter the digit for cluster 21: 0
Enter the digit for cluster 86: 0
Enter the digit for cluster 14: 0
Enter the digit for cluster 6: 0
Enter the digit for cluster 44: 2
Enter the digit for cluster 79: 0
Enter the digit for cluster 74: 8
Enter the digit for cluster 42: 8
Enter the digit for cluster 98: 0
Enter the digit for cluster 88: 8

Enter the digit for cluster 91: 6
Enter the digit for cluster 17: 6
Enter the digit for cluster 20: 8
Enter the digit for cluster 47: 8
Enter the digit for cluster 11: 2
Enter the digit for cluster 68: 8
Enter the digit for cluster 54: 4
Enter the digit for cluster 58: 5
Enter the digit for cluster 36: 3
Enter the digit for cluster 94: 3
Enter the digit for cluster 1: 1
Enter the digit for cluster 4: 1
Enter the digit for cluster 99: 2
Enter the digit for cluster 77: 1
Enter the digit for cluster 90: 1
Enter the digit for cluster 76: 1
Enter the digit for cluster 7: 1
Enter the digit for cluster 72: 1
Enter the digit for cluster 52: 8

Enter the digit for cluster 53: 9
Enter the digit for cluster 22: 9
Enter the digit for cluster 35: 6
Enter the digit for cluster 26: 6
Enter the digit for cluster 81: 5
Enter the digit for cluster 85: 5
Enter the digit for cluster 45: 6
Enter the digit for cluster 75: 9
Enter the digit for cluster 89: 7
Enter the digit for cluster 70: 7
Labeling Accuracy: 88.69%
SVM Training Accuracy: 89.85%
SVM Test Accuracy: 93.10%
Found 201 uncertain images.
Now retraining SVM...
Updated SVM Training Accuracy: 89.85%
Updated SVM Test Accuracy: 93.10%
Now retraining SVM...
Updated SVM Training Accuracy: 90.13%
Updated SVM Test Accuracy: 93.55%
Now retraining SVM...
Updated SVM Training Accuracy: 90.17%
Updated SVM Test Accuracy: 93.55%
Now retraining SVM...
Updated SVM Training Accuracy: 90.21%
Updated SVM Test Accuracy: 93.60%
Now retraining SVM...
Updated SVM Training Accuracy: 90.25%
Updated SVM Test Accuracy: 93.45%
Total Execution Time: 367.32 seconds

```

Pipeline 1 – A “not required”	Manual Time (Estimated)	Manual Time (Experimented)	Simulation Time	Accuracy
Iteration 1	3350 sec	625 sec	273.05 sec (include manual cluster labelling)	91.75 %
Iteration 2	1210 sec	109 sec	28.21 sec	91.75 %
Iteration 3	830 sec	75 sec	27.82 sec	92.05 %
Iteration 4	740 sec	67 sec	27.68 sec	91.95 %
Iteration 5	690 sec	62 sec	27.55 sec	92.05 %
Total	6820 sec	938 sec	384.31 sec	

Pipeline 1 – B “required”	Manual Time (Estimated)	Manual Time (Experimented)	Simulation Time	Accuracy
Iteration 1	1000 sec	170 sec	257.8 sec (include manual cluster labelling)	93.10 %
Iteration 2	-	-	27.73 sec	93.55 %
Iteration 3	-	-	27.16 sec	93.55 %
Iteration 4	-	-	27.32 sec	93.60 %
Iteration 5	-	-	27.31 sec	93.45 %
Total	1000 sec	170 sec	367.32 sec	

## Pipeline 2: Training based on Augmentation

### 1. Code Procedures

The code implements a random sampling of 40 images followed by augmentation and trains the model (SVM-1) followed by iterative refinement of the model (SVM-2) until the accuracy converges.

```
Loading MNIST dataset...
Initial Model Training:
- Iteration 0 Accuracy: 92.66%
Iteration 0 Time: 15.649891138076782s
Iterative Refinement:
- Iteration 1 Accuracy: 93.23%
Iteration 1 Time: 28.05673313140869s
- Iteration 2 Accuracy: 93.4%
Iteration 2 Time: 27.457985162734985s
- Iteration 3 Accuracy: 93.42%
Iteration 3 Time: 28.18378496170044s
- Iteration 4 Accuracy: 93.47%
Iteration 4 Time: 27.723519325256348s
- Iteration 5 Accuracy: 93.47%
Iteration 5 Time: 27.349474668502808s
Accuracy based on Test Set: 92.35%
Testing Time: 3.769460678100586s
Test set Accuracy after training on 10,000 labeled set: 95.7%
```

### 2. Results Analysis

The initial model achieved [92.66%] accuracy on the training set. Through iterative refinement, the accuracy improved to [93.47%] by iteration 5. When evaluated on the test set, our semi-supervised approach achieved [92.35%] accuracy.

For comparison, a fully supervised approach trained on all 10,000 manually labelled images achieved [95.7%] accuracy on the test set. This represents a performance gap of only [3.35%] while requiring just 4% of the manual labelling effort (67 minutes vs. 27.8 hours).

The most significant improvement occurred during the first [3] iterations, after which the accuracy began to plateau, indicating convergence of the model.

In the following table are summarized results for Pipeline 2 Iterations. The reported time is reported in 3 different ways: - Manual Time estimated by roughly stating that 10 seconds are spent for manual labelling of each image. - Manual Time estimated by experimenting directly for manual labelling of 400 images – Simulation Time for each iteration.

Pipeline 2	Manual Time (Estimated)	Manual Time (Experimented)	Simulated Time	Accuracy
Iteration-1 Results	66.6 min	10 min	28.06 sec	93.23%
Iteration-2 Results	66.6 min	10 min	27.46 sec	93.4%
Iteration-3 Results	66.6 min	10 min	28.18 sec	93.42%
Iteration-4 Results	66.6 min	10 min	27.72 sec	93.47%
Iteration-5 Results	66.6 min	10 min	27.35 sec	93.47%

### 3. Conclusion

Pipeline 2 demonstrates that semi-supervised learning with data augmentation can achieve near-competitive performance while drastically reducing the manual labelling burden. The approach reduced the required manual labelling time from approximately 27.8 hours to just 67 minutes, while maintaining  $[92.35/95.7 \times 100\% \approx 96.5\%]$  of the accuracy of a fully supervised model.

This approach would be particularly valuable in scenarios where obtaining labelled data is expensive or time-consuming, allowing for efficient allocation of human resources while still producing high-quality classification models.

## Pipeline 3: Active Learning with SVM

### 1. Dataset load and prepare

- Read image files from training and testing directories.
- Assign labels based on directory names (digits 0-9).
- Form flatten images from 28x28 to 784 (1D vector) as a feature.

### 2. Initialize Active Learning Process

- Randomly select an initial small labeled subset (**20** samples per class).
- Train an initial **weak** Support Vector Machine (SVM) classifier using this subset.

### 3. Active Learning Loop (5 iterations)

- Identify **uncertain** samples using SVM's predicted probabilities (samples with the **lowest confidence**).
- Select the **50** most uncertain samples and add them to the labeled dataset **manually**.
- **Retrain** the **SVM** model with the **expanded** dataset.
- Evaluate model performance on the train and test datasets after each iteration.

### 4. Final Evaluation

- Display the total number of manually labeled samples used at the end (50 for each iteration).

```
Iteration 1:
Train Accuracy after iteration 1: 87.80%
Test Accuracy after iteration 1: 89.75%
Total Execution Time: 4.22 seconds
Iteration 2:
Train Accuracy after iteration 2: 88.86%
Test Accuracy after iteration 2: 89.45%
Total Execution Time: 2.35 seconds
Iteration 3:
Train Accuracy after iteration 3: 90.09%
Test Accuracy after iteration 3: 90.50%
Total Execution Time: 2.74 seconds
Iteration 4:
Train Accuracy after iteration 4: 90.64%
Test Accuracy after iteration 4: 90.95%
Total Execution Time: 3.20 seconds
Iteration 5:
Train Accuracy after iteration 5: 91.99%
Test Accuracy after iteration 5: 92.05%
Total Execution Time: 3.64 seconds
```

Pipeline 3 "optional"	Manual Time (Estimated)	Manual Time (Experimented)	Simulation Time	Accuracy
Iteration 1	2500 sec	232 sec	4.22 sec	89.75 %
Iteration 2	500 sec	50 sec	2.35 sec	89.45 %
Iteration 3	500 sec	46 sec	2.74 sec	90.5 %
Iteration 4	500 sec	47 sec	3.20 sec	90.95 %
Iteration 5	500 sec	45 sec	3.64 sec	92.05 %
Total	4500 sec	420 sec	16.15 sec	

# Pipeline 4: CNN-Based Handwritten Digit Classification

## 1. Load Dataset

- Define the **load\_images** function to read images from specified directories.
- Convert images to grayscale and resize them to **28x28** pixels.
- Store images and their corresponding labels (digits 0-9).

## 2. Preprocessing

- Normalize pixel values to the range [0,1] by dividing by 255.
- Reshape images to **(28,28,1)** to fit CNN input format.

## 3. Data Augmentation

- Use ``ImageDataGenerator`` to apply **random rotations, shifts, and zooms** to training images.

## 4. Define the CNN Model





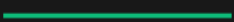
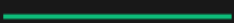
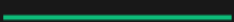
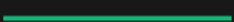
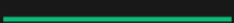





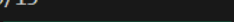
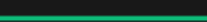
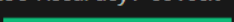
- **Convolutional Layers:**
  - Conv2D (32 filters, 3x3, ReLU)
  - MaxPooling (2x2)
  - Conv2D (64 filters, 3x3, ReLU)
  - MaxPooling (2x2)
  - Conv2D (128 filters, 3x3, ReLU)
- **Flatten layer:** Converts feature maps into a 1D vector.
- **Fully Connected Layer** (Dense 128, ReLU)
- **Dropout** (50%): Prevents overfitting.
- **Output Layer** (Softmax, 10 classes): Predicts digit class (0-9).

## 5. Train the Model

- Use **Adam optimizer** and **Sparse Categorical Crossentropy loss**.
- Train using augmented data (**datagen.flow**).
- Run for **15 epochs** with a batch size of **64**.
- Use **validation\_data = (X\_test, y\_test)** to track performance.

## 6. Evaluate the Model

- Compute **test accuracy** on unseen data.
- Compute **train accuracy** to check for overfitting.

```
Epoch 1/15
157/157  4s 19ms/step - accuracy: 0.3953 - loss: 1.6901 - val_accuracy: 0.9445 - val_loss: 0.1926
Epoch 2/15
157/157  3s 18ms/step - accuracy: 0.8433 - loss: 0.5064 - val_accuracy: 0.9715 - val_loss: 0.0829
Epoch 3/15
157/157  3s 18ms/step - accuracy: 0.9032 - loss: 0.3174 - val_accuracy: 0.9835 - val_loss: 0.0570
Epoch 4/15
157/157  3s 18ms/step - accuracy: 0.9171 - loss: 0.2662 - val_accuracy: 0.9800 - val_loss: 0.0669
Epoch 5/15
157/157  3s 19ms/step - accuracy: 0.9396 - loss: 0.2029 - val_accuracy: 0.9820 - val_loss: 0.0548
Epoch 6/15
157/157  3s 18ms/step - accuracy: 0.9480 - loss: 0.1829 - val_accuracy: 0.9820 - val_loss: 0.0515
Epoch 7/15
157/157  3s 18ms/step - accuracy: 0.9543 - loss: 0.1574 - val_accuracy: 0.9810 - val_loss: 0.0515
Epoch 8/15
157/157  3s 19ms/step - accuracy: 0.9593 - loss: 0.1316 - val_accuracy: 0.9880 - val_loss: 0.0408
Epoch 9/15
157/157  3s 19ms/step - accuracy: 0.9611 - loss: 0.1355 - val_accuracy: 0.9875 - val_loss: 0.0370
Epoch 10/15
157/157  3s 19ms/step - accuracy: 0.9674 - loss: 0.1129 - val_accuracy: 0.9860 - val_loss: 0.0504
Epoch 11/15
157/157  3s 18ms/step - accuracy: 0.9691 - loss: 0.1061 - val_accuracy: 0.9890 - val_loss: 0.0415
Epoch 12/15
157/157  3s 19ms/step - accuracy: 0.9685 - loss: 0.1026 - val_accuracy: 0.9865 - val_loss: 0.0391
Epoch 13/15
157/157  3s 18ms/step - accuracy: 0.9641 - loss: 0.1181 - val_accuracy: 0.9895 - val_loss: 0.0285
Epoch 14/15
157/157  3s 19ms/step - accuracy: 0.9724 - loss: 0.0930 - val_accuracy: 0.9905 - val_loss: 0.0323
Epoch 15/15
157/157  3s 19ms/step - accuracy: 0.9727 - loss: 0.0924 - val_accuracy: 0.9900 - val_loss: 0.0359
63/63  0s 3ms/step - accuracy: 0.9885 - loss: 0.0368
Final Test Accuracy: 99.00%
313/313  1s 3ms/step - accuracy: 0.9883 - loss: 0.0335
Final Train Accuracy: 98.71%
```