Part 1

```
hyperparams = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'degree': [2, 3, 4],
}
```

In the example code I provided for training an SVM, I used the following hyper parameters:

- `C`: The regularization parameter, which controls the tradeoff between maximizing the margin and minimizing the classification error. A smaller value of C gives a wider margin but may result in misclassified examples, while a larger value of C gives a narrower margin but may result in overfitting. In the code example, I tried values of C=[0.1, 1, 10, 100] to explore a range of regularization strengths.

- `Kernel`: The kernel function used to transform the input data into a higher-dimensional feature space, where the decision boundary can be linear. The choice of kernel affects the shape of the decision boundary and can have a significant impact on the SVM's performance. In the code example, I tried three different kernel functions: 'linear', 'rbf' (radial basis function), and 'poly' (polynomial).

- `Degree`: The degree of the polynomial kernel function, if the 'poly' kernel is used. The degree determines the complexity of the feature space, and higher values of degree can lead to overfitting. In the code example, I tried values of degree=[2, 3, 4] to explore a range of polynomial degrees.

```
SVM with C=0.1, kernel=linear, degree=2:
[[449   7   1   1   5   1  13   1   4   1]
 [  8 467  14   2  14   0   3  28   2   1]
 [  6   7 500   1   8   1   4   2   7   3]
 [  4   2  10 402   6  26   6   2  15  11]
 [  5  27   2   4 437   0   7   1   4   3]
 [  6   2   6  32   2 438  21   6   4   9]
 [ 15   1   2   9  13  24 401   1  13   4]
 [  5  23  10   3   3   1   0 457   3   1]
 [ 18   5   5  48   7  16  18   1 344   9]
 [ 17   7  14  11  14   2   2   1  12 399]]
```

```
SVM with C=1, kernel=linear, degree=2:
[[449   7   1   1   5   1  13   1   4   1]
 [  8 467  14   2  14   0   3  28   2   1]
 [  6   7 500   1   8   1   4   2   7   3]
 [  4   2  10 402   6  26   6   2  15  11]
 [  5  27   2   4 437   0   7   1   4   3]
 [  6   2   6  32   2 438  21   6   4   9]
 [ 15   1   2   9  13  24 401   1  13   4]
 [  5  23  10   3   3   1   0 457   3   1]
 [ 18   5   5  48   7  16  18   1 344   9]
 [ 17   7  14  11  14   2   2   1  12 399]]
```

```
SVM with C=10, kernel=linear, degree=2:
[[449   7   1   1   5   1  13   1   4   1]
 [  8 467  14   2  14   0   3  28   2   1]
 [  6   7 500   1   8   1   4   2   7   3]
 [  4   2  10 402   6  26   6   2  15  11]
 [  5  27   2   4 437   0   7   1   4   3]
 [  6   2   6  32   2 438  21   6   4   9]
 [ 15   1   2   9  13  24 401   1  13   4]
 [  5  23  10   3   3   1   0 457   3   1]
 [ 18   5   5  48   7  16  18   1 344   9]
 [ 17   7  14  11  14   2   2   1  12 399]]
```

```
SVM with C=100, kernel=linear, degree=2:
[[449   7   1   1   5   1  13   1   4   1]
 [  8 467  14   2  14   0   3  28   2   1]
 [  6   7 500   1   8   1   4   2   7   3]
 [  4   2  10 402   6  26   6   2  15  11]
 [  5  27   2   4 437   0   7   1   4   3]
 [  6   2   6  32   2 438  21   6   4   9]
 [ 15   1   2   9  13  24 401   1  13   4]
 [  5  23  10   3   3   1   0 457   3   1]
 [ 18   5   5  48   7  16  18   1 344   9]
 [ 17   7  14  11  14   2   2   1  12 399]]
```

-----------RBF------------

```
SVM with C=0.1, kernel=rbf, degree=2:
[[411  27  22   1   7   2  10   3   0   0]
 [  0 466  31   1   5   0   0  35   1   0]
 [  1   7 518   0   3   0   2   5   2   1]
 [  8  11  57 336   2  45  19   3   3   0]
 [  2  58  13   0 414   0   1   2   0   0]
 [  6   5  29  16   3 425  36   4   0   2]
 [ 29  10  10   5   7  59 356   2   5   0]
 [  1  51  34   0   0   2   0 418   0   0]
 [ 19  19  53  74   4  45 107   1 147   2]
 [ 81  15 128  33  28  36  17   2  11 128]]
```

```
SVM with C=1, kernel=rbf, degree=2:
[[459   4   2   0   1   1   8   2   2   4]
 [  4 484  19   0   5   0   0  25   2   0]
 [  1   9 513   2   4   0   1   5   2   2]
 [  7   4  10 422   4  13   2   2  16   4]
 [  1  21   2   0 458   0   0   4   2   2]
 [  3   3   6  25   1 449  21   5   4   9]
 [  9   3   4   6   5  13 430   0   9   4]
 [  2  20   2   0   1   1   0 480   0   0]
 [ 14   7   3  23   1   5  24   2 382  10]
 [ 18   4  10   4  11   2   1   1  10 418]]
```

```
SVM with C=10, kernel=rbf, degree=2:
[[458   3   2   0   2   1   8   2   4   3]
 [  5 487  14   0   8   0   0  21   3   1]
 [  3   9 512   2   5   1   1   2   2   2]
 [  6   2  10 426   4  14   3   3  13   3]
 [  3  16   2   1 456   1   1   4   2   4]
 [  4   2   6  26   2 455  16   4   3   8]
 [ 12   3   2   5   7  10 432   0   9   3]
 [  3  19   7   0   1   1   0 475   0   0]
 [ 12   5   3  21   3   9  22   2 387   7]
 [ 12   5  12   3  13   3   2   1   7 421]]
```

```
SVM with C=100, kernel=rbf, degree=2:
[[458   3   2   0   2   1   8   2   4   3]
 [  5 487  14   0   8   0   0  21   3   1]
 [  3   9 512   2   5   1   1   2   2   2]
 [  6   2  10 426   4  14   3   3  13   3]
 [  3  16   2   1 456   1   1   4   2   4]
 [  4   2   6  26   2 455  16   4   3   8]
 [ 12   3   2   5   7  10 432   0   9   3]
 [  3  19   7   0   1   1   0 475   0   0]
 [ 12   5   3  21   3   9  22   2 387   7]
 [ 12   5  12   3  13   3   2   1   7 421]]
```

-----Poly-----

```
SVM with C=0.1, kernel=poly, degree=2:
[[388  63  14   0   3   2   9   1   2   1]
 [  0 499  20   1   0   0   0  19   0   0]
 [  0  33 496   1   1   0   2   2   2   2]
 [  5  59  40 338   2  29   7   1   3   0]
 [  0 114   6   0 369   0   0   1   0   0]
 [  3  30  21  12   2 425  28   2   0   3]
 [ 18  48   9   4   6  38 348   1   9   2]
 [  0 123  18   0   0   1   0 364   0   0]
 [  8  70  22  61   7  26  74   0 201   2]
 [ 40  67  65  23  19  21   9   2  14 219]]
SVM with C=0.1, kernel=poly, degree=3:
[[280 183  14   0   0   0   6   0   0   0]
 [  0 520  12   0   0   0   0   7   0   0]
 [  0 102 431   0   1   0   1   2   2   0]
 [  1 222  53 169   1  31   6   0   1   0]
 [  0 231   2   0 257   0   0   0   0   0]
 [  1 159  15   1   0 325  25   0   0   0]
 [ 14 167   2   0   2  28 264   0   6   0]
 [  0 287  17   0   0   1   0 201   0   0]
 [  5 230  23   4   4  39  51   0 115   0]
 [ 22 234  62   2   8  29   9   1   9 103]]
SVM with C=0.1, kernel=poly, degree=4:
[[144 326  11   0   0   0   2   0   0   0]
 [  0 530   9   0   0   0   0   0   0   0]
 [  0 182 354   0   0   0   0   2   1   0]
 [  0 334  51  79   0  19   0   0   1   0]
 [  0 334   4   0 152   0   0   0   0   0]
 [  0 346   9   0   0 162   9   0   0   0]
 [  7 302   3   0   1   5 165   0   0   0]
 [  0 380  15   0   0   0   0 111   0   0]
 [  1 334  27   0   2  11  27   0  69   0]
 [  7 375  30   0   2   8   3   0   8  46]]
```

```
SVM with C=1, kernel=poly, degree=2:
[[452   9   4   0   2   0   6   1   4   5]
 [  3 486  19   1   7   0   0  18   3   2]
 [  2  12 510   3   3   0   1   3   2   3]
 [  6   9  16 413   3  15   4   1  15   2]
 [  2  20   4   0 459   0   0   1   2   2]
 [  3   4   8  30   1 443  19   4   5   9]
 [ 13   8   3   6   7  13 416   0  13   4]
 [  2  30   5   0   0   1   0 468   0   0]
 [ 14  11   5  11   6  10  17   3 385   9]
 [ 14   6  11   7  11   4   2   1   6 417]]
SVM with C=1, kernel=poly, degree=3:
[[434  23   7   2   3   0   8   1   4   1]
 [  2 488  19   1   3   0   0  24   2   0]
 [  0  20 506   2   3   0   2   2   2   2]
 [  4  24  18 406   3   9   4   2  13   1]
 [  1  37   4   0 443   0   0   1   3   1]
 [  2  12  11  29   0 442  17   1   7   5]
 [ 12  28   3   2   5  14 403   0  13   3]
 [  3  47   6   1   0   1   0 448   0   0]
 [  9  28   5  15   1  11  17   2 376   7]
 [ 17  24  15   6  11   6   2   1  12 385]]
SVM with C=1, kernel=poly, degree=4:
[[395  69   6   1   3   0   6   0   2   1]
 [  1 502  16   0   2   0   0  17   1   0]
 [  0  51 478   1   2   0   2   1   2   2]
 [  3  96  13 347   1   5   4   2  12   1]
 [  0  90   1   0 396   0   0   1   2   0]
 [  1  68   6  21   0 403  18   1   6   2]
 [  9  70   3   0   3  18 365   0  12   3]
 [  1 110   5   0   0   1   0 389   0   0]
 [  7  89   2   8   0   6  13   0 343   3]
 [ 13  80  14   5   5   5   2   2  14 339]]
```

```
SVM with C=10, kernel=poly, degree=2:
[[451   3   5   1   2   2   7   2   5   5]
 [  5 482  15   2   8   0   1  18   6   2]
 [  2  11 504   2   5   2   2   3   5   3]
 [  5   5  13 412   4  19   5   0  16   5]
 [  5  15   4   1 455   1   0   3   2   4]
 [  6   4   9  29   1 437  19   6   5  10]
 [  9   2   3   7   7  14 426   1  10   4]
 [  2  29   8   1   0   2   0 464   0   0]
 [ 15   8   7  12   5  13  24   3 375   9]
 [ 17   8  11   5  11   8   3   0   7 409]]
SVM with C=10, kernel=poly, degree=3:
[[441   7   8   1   3   1   8   3   7   4]
 [  6 482  20   1   3   0   0  20   5   2]
 [  1   9 514   1   5   1   2   2   2   2]
 [  7   7  15 411   3  14   4   3  16   4]
 [  0  17   6   0 460   0   1   2   3   1]
 [  6   5   8  24   2 449  18   4   4   6]
 [ 13   7  10   2   7  18 410   1  11   4]
 [  3  29  12   2   0   1   0 459   0   0]
 [ 14   7   6  14   3  16  20   2 380   9]
 [ 18   9  17   9   8   6   1   0  10 401]]
SVM with C=10, kernel=poly, degree=4:
[[425  32   6   0   2   1   7   2   6   2]
 [  3 484  20   1   4   0   1  22   4   0]
 [  1  21 502   3   3   0   1   2   4   2]
 [  5  25  21 397   3  13   3   1  13   3]
 [  1  37   8   0 439   0   0   1   3   1]
 [  5  20   9  28   1 432  19   1   6   5]
 [ 15  33   5   0   3  18 387   0  20   2]
 [  2  55  14   0   0   1   0 434   0   0]
 [ 11  31  11   8   0  12  21   2 366   9]
 [ 24  31  16   8  10   8   1   2  10 369]]
```

```
SVM with C=100, kernel=poly, degree=2:
[[450   3   5   1   2   2   8   2   5   5]
 [  5 480  14   2   9   0   1  20   6   2]
 [  3  11 503   2   5   3   2   3   4   3]
 [  5   5  14 413   4  18   5   0  16   4]
 [  5  17   5   1 452   1   0   3   2   4]
 [  6   3  10  30   1 439  17   6   5   9]
 [ 10   2   3   7   7  14 425   1  10   4]
 [  4  27  10   1   0   2   0 462   0   0]
 [ 15   8   8  12   5  13  24   3 374   9]
 [ 17   8  12   5  10   7   3   0   6 411]]
SVM with C=100, kernel=poly, degree=3:
[[443   5   7   1   2   1  10   3   7   4]
 [  4 479  22   3   3   1   1  20   4   2]
 [  0   6 516   2   5   1   2   3   2   2]
 [  7   5  19 407   3  15   5   3  16   4]
 [  0  16   7   0 456   1   2   2   3   3]
 [  7   5   6  25   2 449  18   4   4   6]
 [ 14   4  10   1   5  19 414   1  11   4]
 [  3  28  15   2   0   1   0 457   0   0]
 [ 12   5   9  14   1  15  21   2 380  12]
 [ 18   6  18   8  11   6   2   0  11 399]]
SVM with C=100, kernel=poly, degree=4:
[[432  17   9   1   3   1   8   3   6   3]
 [  4 479  20   2   4   1   1  23   4   1]
 [  0  15 510   3   2   1   2   1   3   2]
 [  7  20  29 387   3  15   7   1  12   3]
 [  2  31   9   0 441   0   0   2   3   2]
 [  5  17  11  25   2 432  20   3   6   5]
 [ 14  24   9   1   3  19 393   0  18   2]
 [  2  43  20   0   0   1   0 440   0   0]
 [ 11  16  14   8   2  17  26   0 367  10]
 [ 28  19  27   5   7   6   2   2  13 370]]
```

To compare the confusion matrices, we need to calculate some metrics that can help us evaluate the performance of the different models. Here are some common metrics used for evaluating classification models:

- Accuracy: the proportion of correctly classified instances out of the total number of instances.
- Precision: the proportion of true positives out of the total number of positive predictions.
- Recall: the proportion of true positives out of the total number of actual positives.
- F1-score: the harmonic mean of precision and recall.

To calculate these metrics, we need to count the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) for each class. Here is a function that can help us calculate these values:

```

SVM with C=0.1, kernel=linear:

{'accuracy': 0.8588, 'precision': 0.8594749080582591, 'recall': 0.8588, 'f1_score': 0.8582208070828447}

SVM with C=0.1, kernel=rbf:

{'accuracy': 0.7238, 'precision': 0.7629682235519027, 'recall': 0.7238, 'f1_score': 0.7048014399551918}

SVM with C=1, kernel=linear:

{'accuracy': 0.8588, 'precision': 0.8594749080582591, 'recall': 0.8588, 'f1_score': 0.8582208070828447}

SVM with C=1, kernel=rbf:

{'accuracy': 0.899, 'precision': 0.8994616675583684, 'recall': 0.899, 'f1_score': 0.8985672085251084}

SVM with C=10, kernel=linear:

{'accuracy': 0.8588, 'precision': 0.8594749080582591, 'recall': 0.8588, 'f1_score': 0.8582208070828447}

SVM with C=10, kernel=rbf:

{'accuracy': 0.9018, 'precision': 0.9021918067095356, 'recall': 0.9018, 'f1_score': 0.9014535439128074}

SVM with C=100, kernel=linear:

{'accuracy': 0.8588, 'precision': 0.8594749080582591, 'recall': 0.8588, 'f1_score': 0.8582208070828447}
```

SVM with C=100, kernel=rbf:

{'accuracy': 0.9018, 'precision': 0.9021918067095356, 'recall': 0.9018, 'f1_score': 0.9014535439128074}

SVM with C=10, kernel=poly:

accuracy': 0.883, 'precision': 0.8833438392216775, 'recall': 0.883, '}
```{'f1_score': 0.8825644418358177

Based on these results, the SVM model with **RBF kernel and C=10 has the highest overall accuracy**, precision, recall, and F1-score, indicating that it has the best overall performance on the classification task. the performance of the linear kernel models is relatively lower compared to the RBF kernel models, which is expected given that the dataset may have more complex decision boundaries that the linear kernel may not be able to capture as effectively as the RBF kernel. Additionally, increasing the C value generally leads to less regularization and more flexible models, which can improve performance, but may also lead to overfitting if the value is too high. As always, it's important to consider the specific dataset and the desired trade-offs between different metrics when selecting a model for a given task.

the SVM with C=10 and kernel=poly achieved the highest F1 score of 0.8826. The SVM with C=1 and kernel=rbf achieved the highest precision of 0.8995, while the SVM with C=10 and kernel=rbf achieved the highest recall of 0.9018 but **for our problem accuracy is most commonly used**.
```