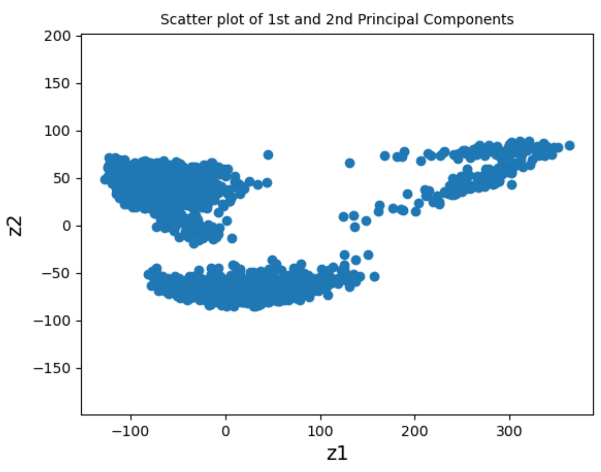
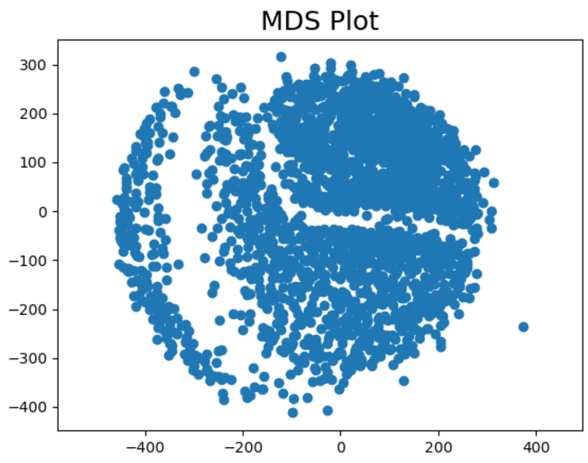
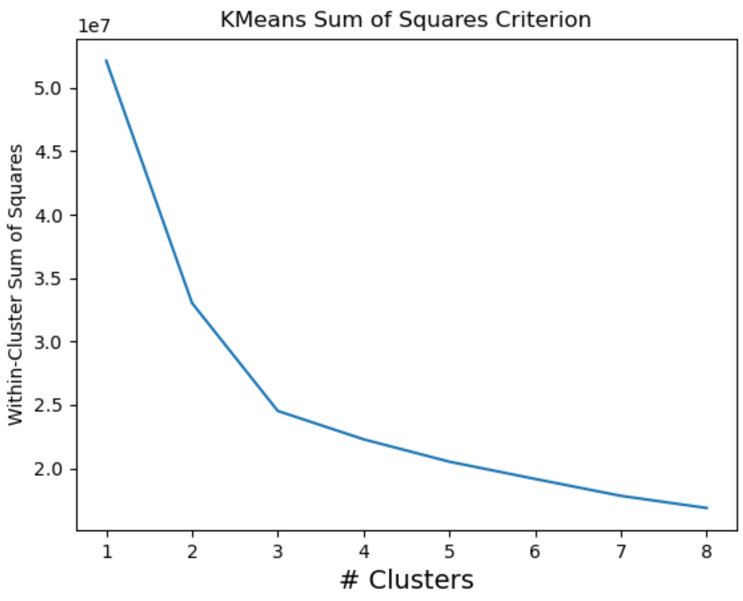
Problem 2

Part 1. Visualization

1.

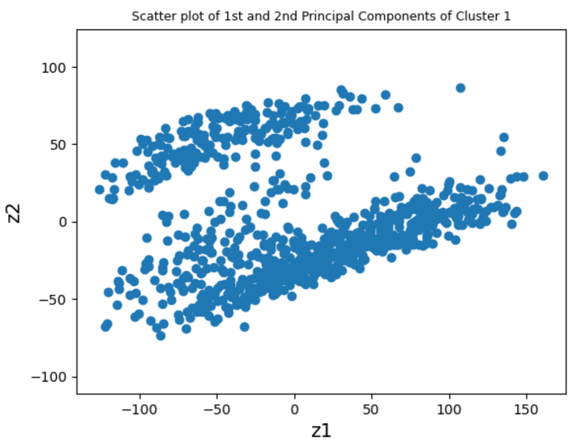


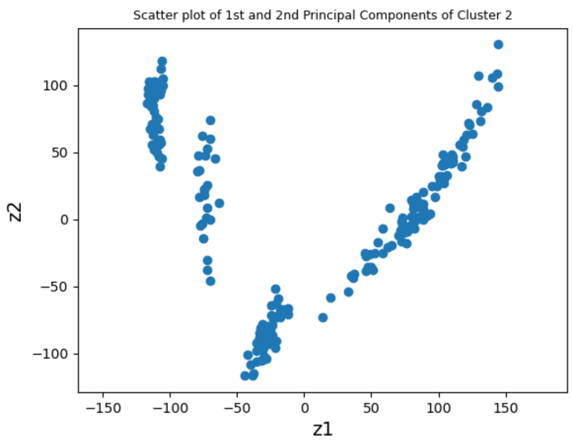


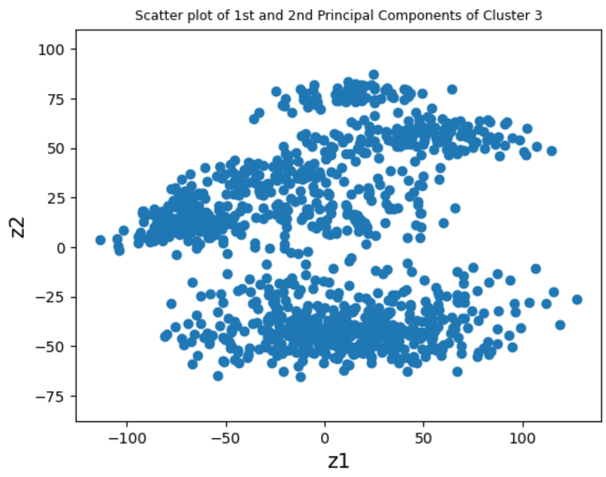


The three above figures speak in favor to 3 clusters.

2. For each of 3 clusters we perform a PCA analysis:

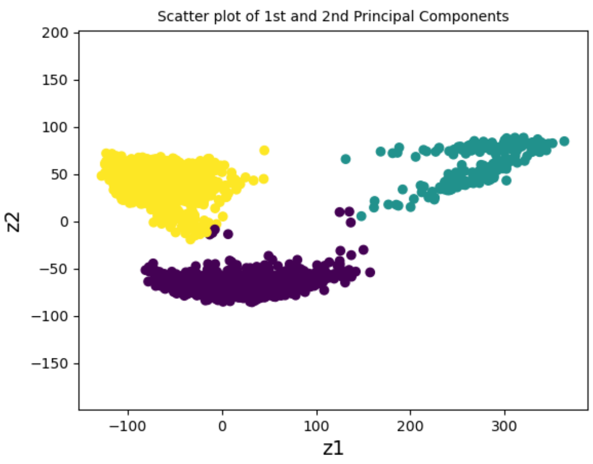


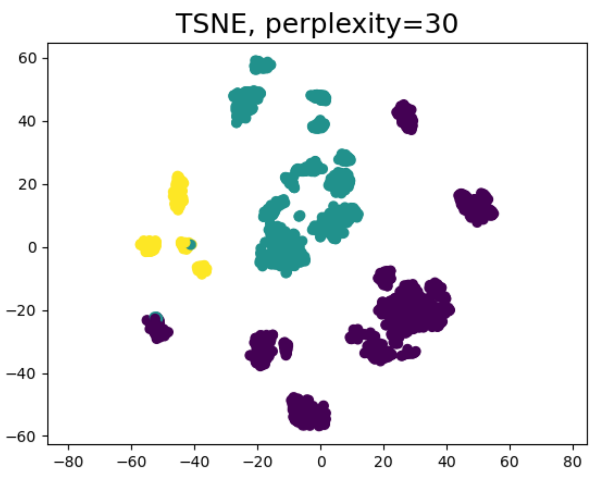




We can see that there is a substructure within each big cluster, suggesting the existence of subclusters.

We can also run k-means algorithm, to assign each point to one of 3 clusters, and then use a TSNE plot with the same coloring from k-means to find the sub-classes within each class.

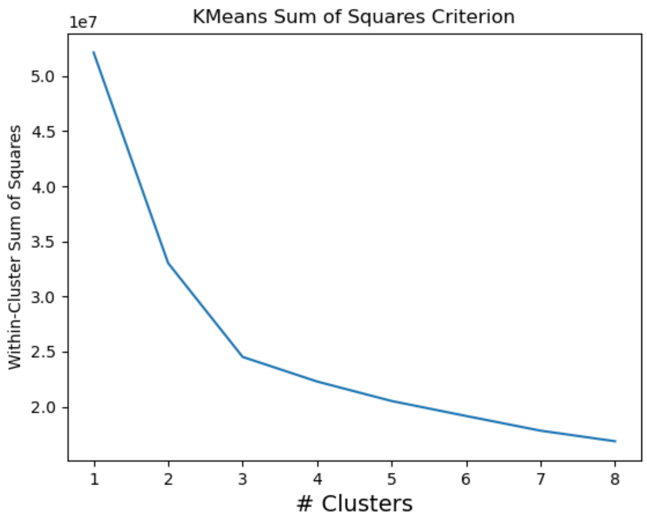


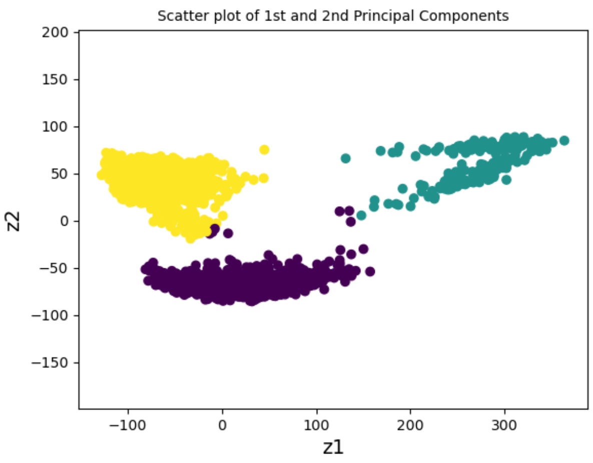


Part 2. Unsupervised Feature Selection

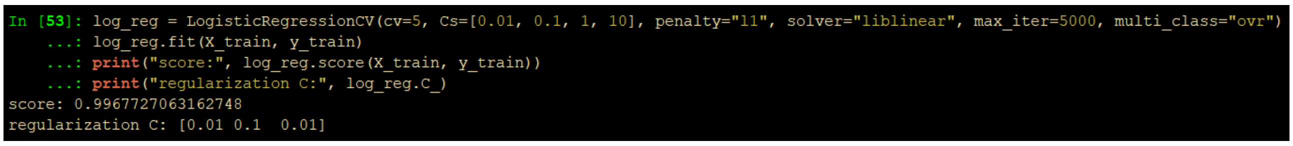
1.

From the above, we can deduce the existence of 3 clusters from the elbow of K-means sum-of-squares criterion plot:



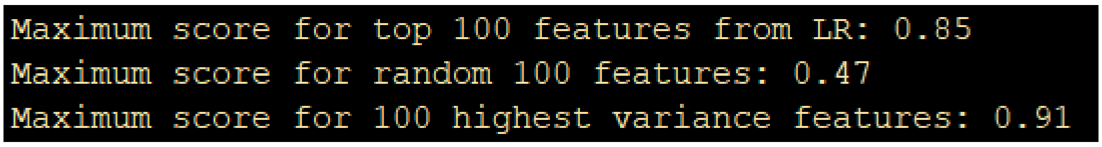


2.

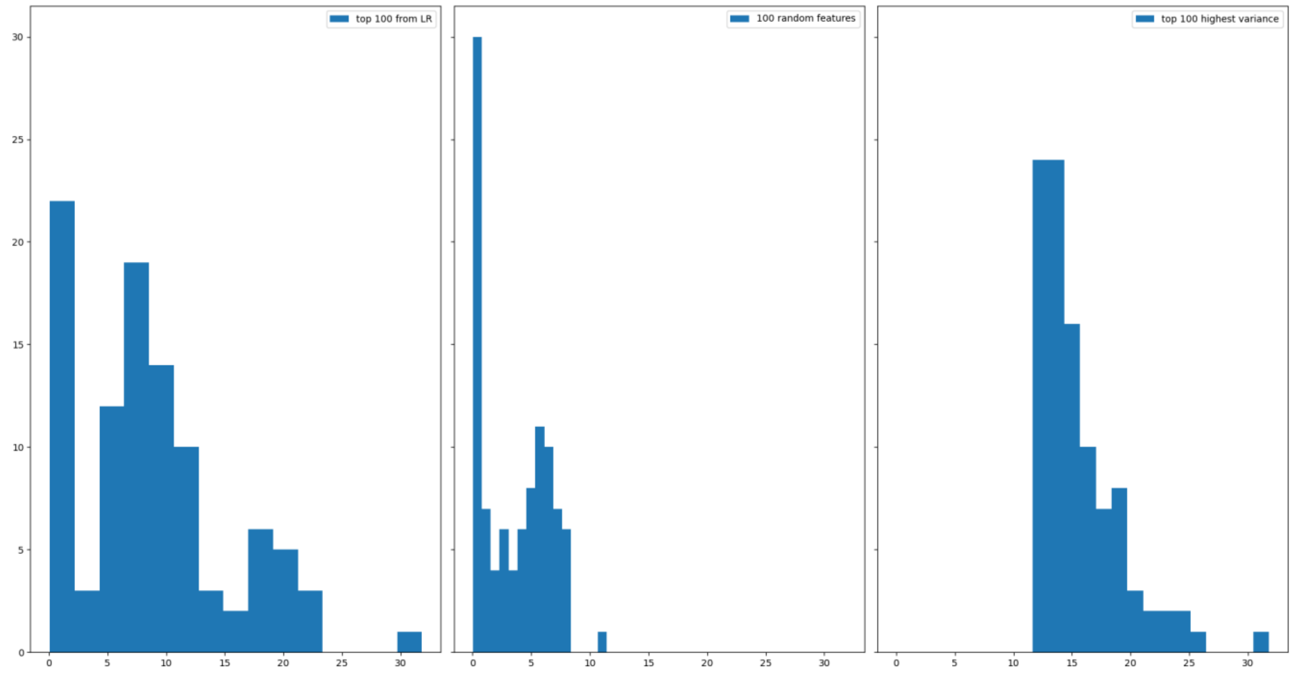


```
In [53]: log_reg = LogisticRegressionCV(cv=5, Cs=[0.01, 0.1, 1, 10], penalty="l1", solver="liblinear", max_iter=5000, multi_class="ovr")
    ...: log_reg.fit(X_train, y_train)
    ...: print("score:", log_reg.score(X_train, y_train))
    ...: print("regularization C:", log_reg.C_)
score: 0.9967727063162748
regularization C: [0.01 0.1  0.01]
```

3. For the reduced dataset I have achieved to get these properties



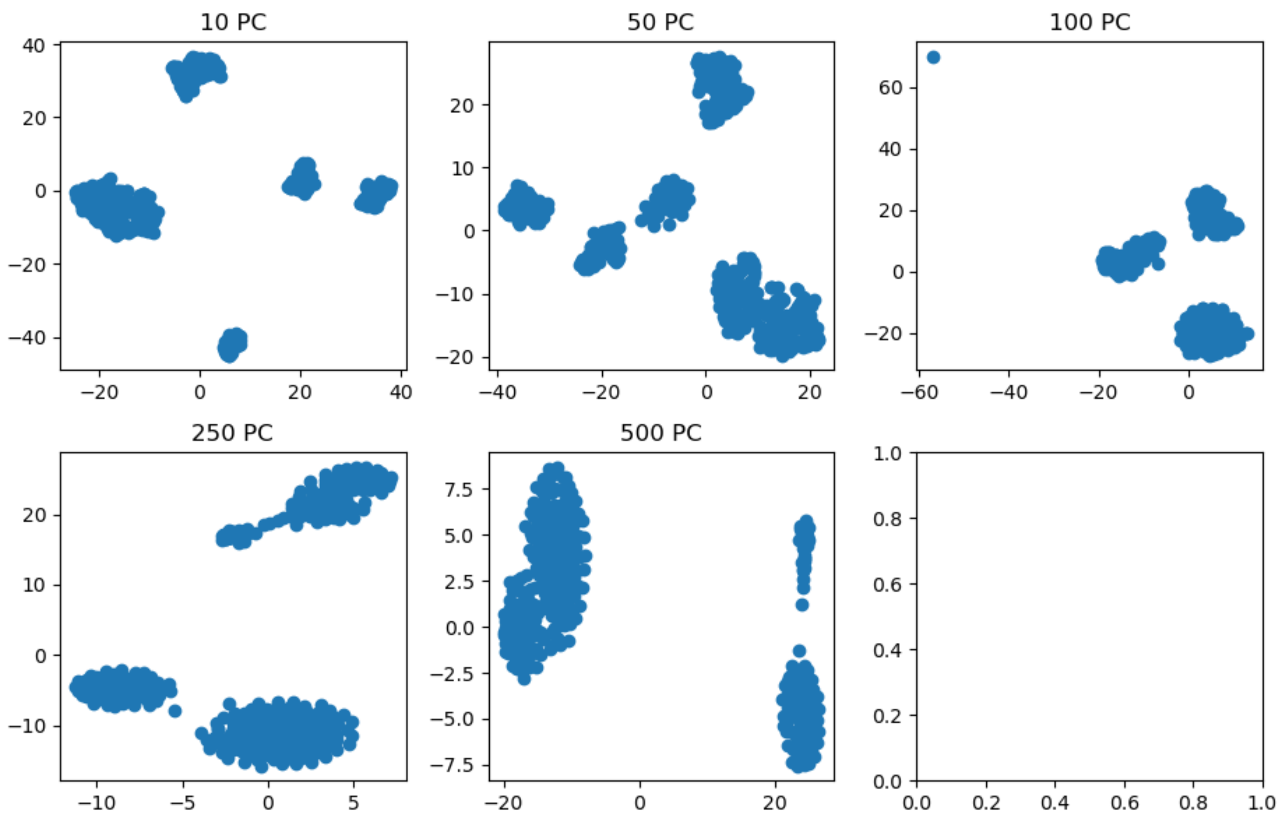
```
Maximum score for top 100 features from LR: 0.85
Maximum score for random 100 features: 0.47
Maximum score for 100 highest variance features: 0.91
```



The features with maximum variance have the highest impact on logistic regression, which is thoroghly possible – it easier to fit the line to the data which is more spread out. But LR gives us more features, which do not nesessarely have high variance but nevertheless are important features.

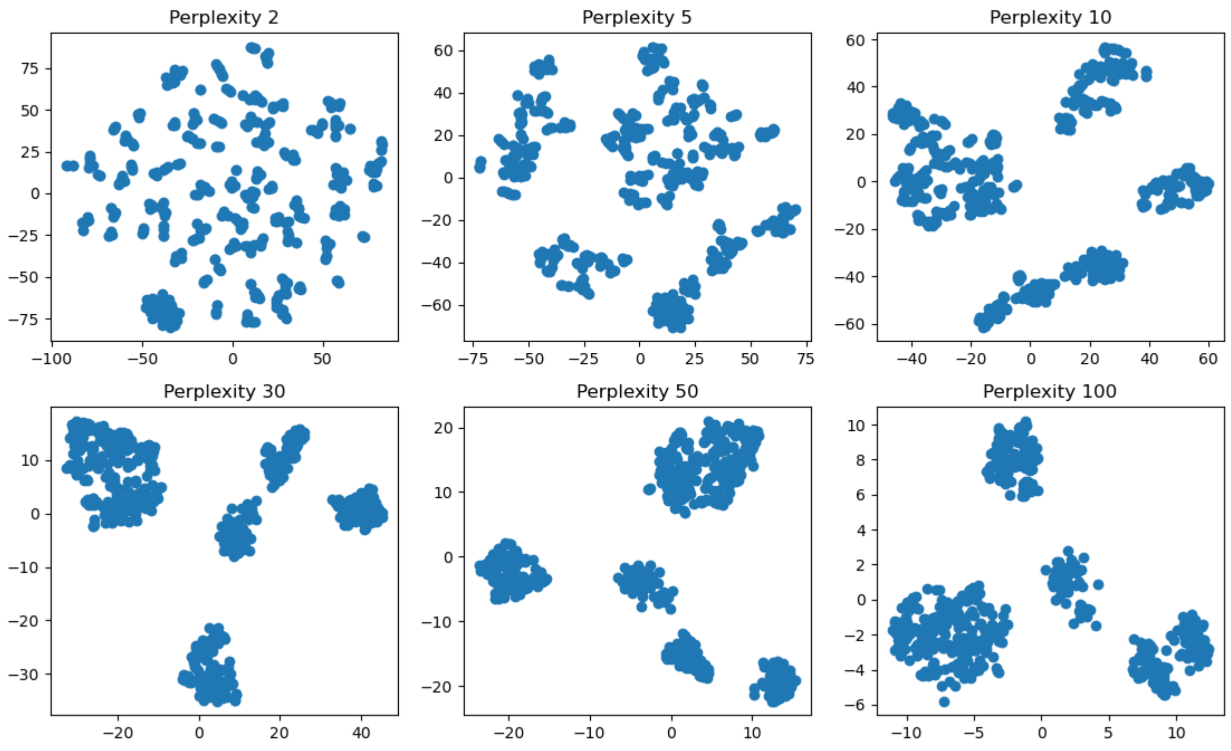
Problem 3.

1.

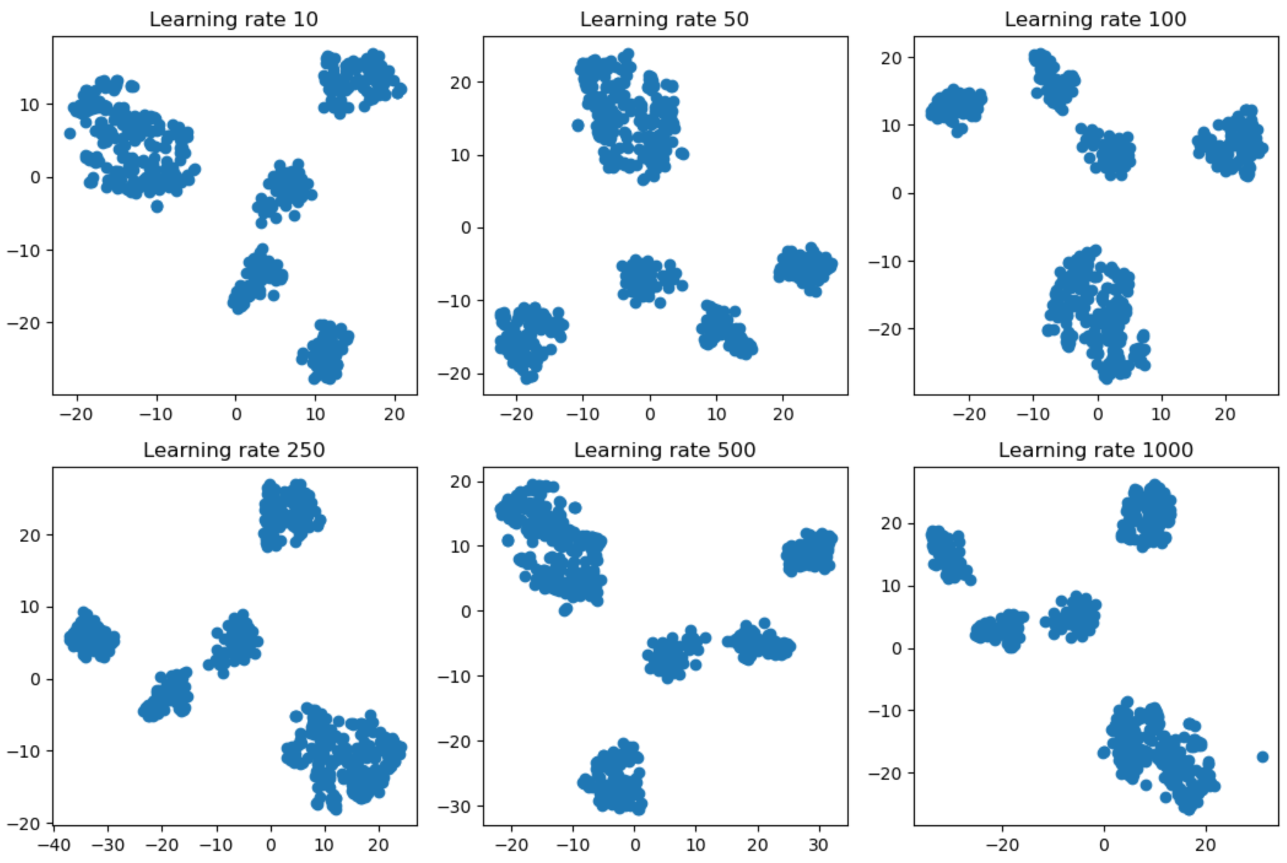


With the increase of PC, the tSNE plot tends to show less separate clusters.

2. With the increase of perplexity level, clusters tend to form more accurately, approaching to the true clustering of the data

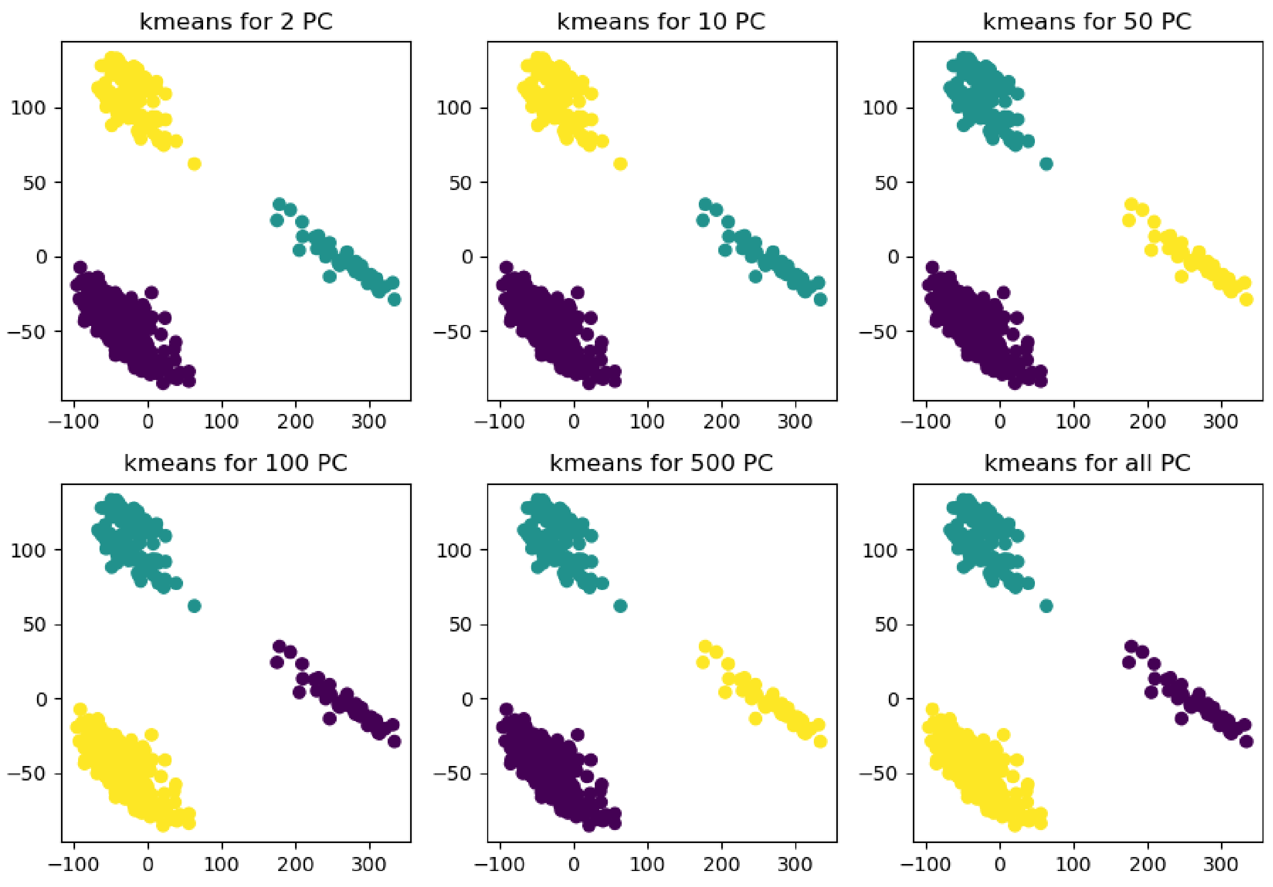


Not found difference in varying learning rate for this dataset, probably due to the fact that the default 1000 iterations suffice for convergence.



Category B (clustering/feature selection)

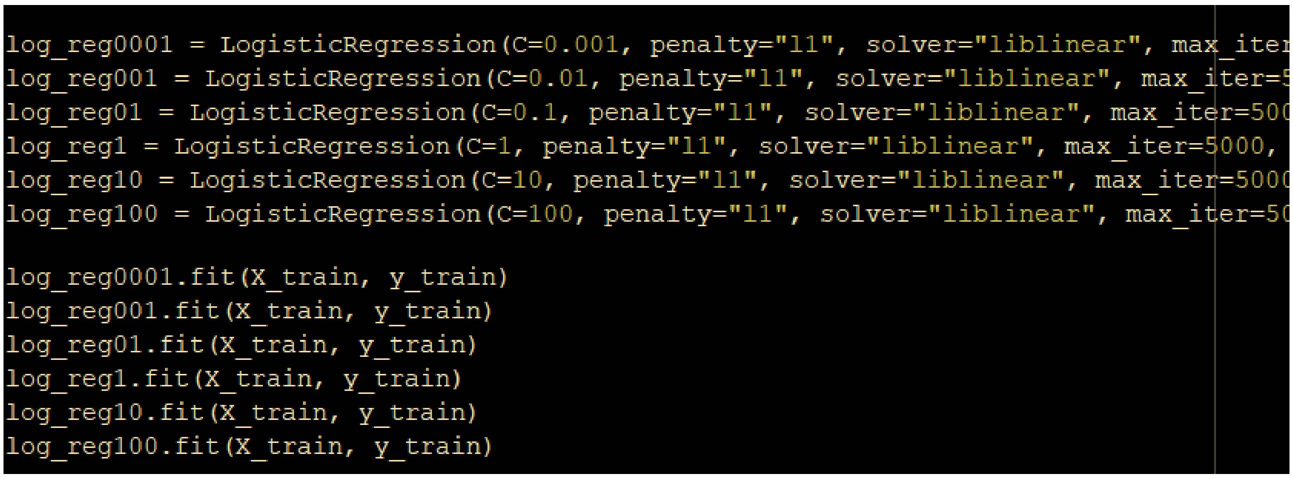
Running **k-means algorithm** on different number of principal components yields the following results:



With no apparent difference between cases

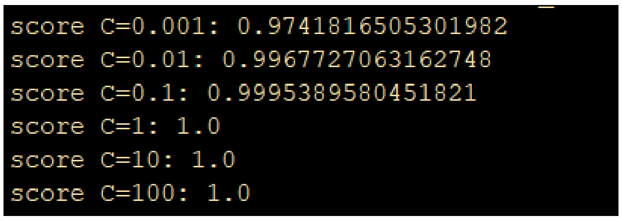
Magnitude of regularization and its relation to the feature selection

We train 6 logistic regression models with different regularization



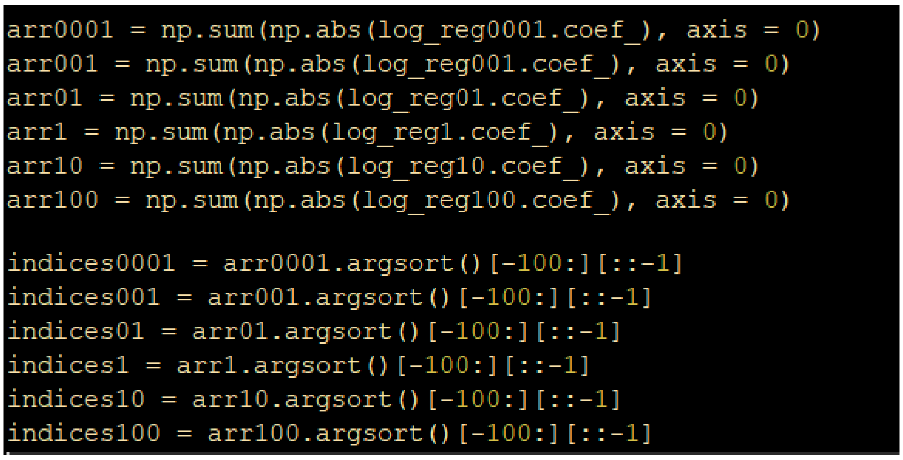
```
log_reg0001 = LogisticRegression(C=0.001, penalty="l1", solver="liblinear", max_iter
log_reg001 = LogisticRegression(C=0.01, penalty="l1", solver="liblinear", max_iter=5
log_reg01 = LogisticRegression(C=0.1, penalty="l1", solver="liblinear", max_iter=500
log_reg1 = LogisticRegression(C=1, penalty="l1", solver="liblinear", max_iter=5000,
log_reg10 = LogisticRegression(C=10, penalty="l1", solver="liblinear", max_iter=5000
log_reg100 = LogisticRegression(C=100, penalty="l1", solver="liblinear", max_iter=50

log_reg0001.fit(X_train, y_train)
log_reg001.fit(X_train, y_train)
log_reg01.fit(X_train, y_train)
log_reg1.fit(X_train, y_train)
log_reg10.fit(X_train, y_train)
log_reg100.fit(X_train, y_train)
```

All of them fit perfectly to this training set:



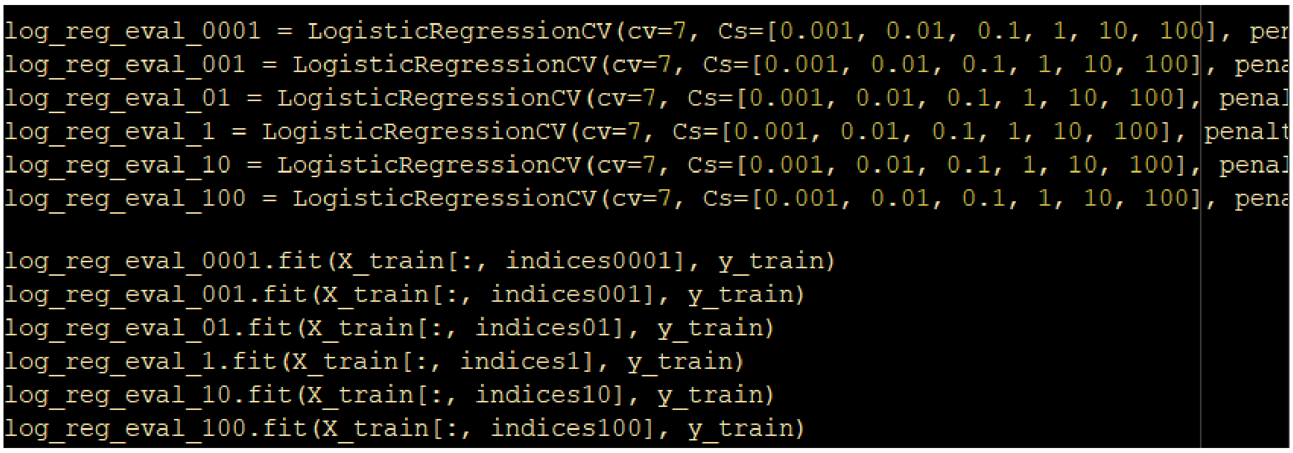
```
score C=0.001: 0.9741816505301982
score C=0.01: 0.9967727063162748
score C=0.1: 0.9995389580451821
score C=1: 1.0
score C=10: 1.0
score C=100: 1.0
```

Then we select top 100 features with the highest coefficients from LR:



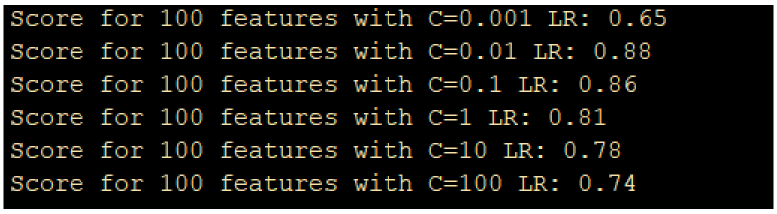
```
arr0001 = np.sum(np.abs(log_reg0001.coef_), axis = 0)
arr001 = np.sum(np.abs(log_reg001.coef_), axis = 0)
arr01 = np.sum(np.abs(log_reg01.coef_), axis = 0)
arr1 = np.sum(np.abs(log_reg1.coef_), axis = 0)
arr10 = np.sum(np.abs(log_reg10.coef_), axis = 0)
arr100 = np.sum(np.abs(log_reg100.coef_), axis = 0)

indices0001 = arr0001.argsort()[-100:][::-1]
indices001 = arr001.argsort()[-100:][::-1]
indices01 = arr01.argsort()[-100:][::-1]
indices1 = arr1.argsort()[-100:][::-1]
indices10 = arr10.argsort()[-100:][::-1]
indices100 = arr100.argsort()[-100:][::-1]
```

Then we use a different – evaluation – dataset and train LR on selected 100 features for all cases



```
log_reg_eval_0001 = LogisticRegressionCV(cv=7, Cs=[0.001, 0.01, 0.1, 1, 10, 100], per
log_reg_eval_001 = LogisticRegressionCV(cv=7, Cs=[0.001, 0.01, 0.1, 1, 10, 100], pena
log_reg_eval_01 = LogisticRegressionCV(cv=7, Cs=[0.001, 0.01, 0.1, 1, 10, 100], penal
log_reg_eval_1 = LogisticRegressionCV(cv=7, Cs=[0.001, 0.01, 0.1, 1, 10, 100], penalt
log_reg_eval_10 = LogisticRegressionCV(cv=7, Cs=[0.001, 0.01, 0.1, 1, 10, 100], penal
log_reg_eval_100 = LogisticRegressionCV(cv=7, Cs=[0.001, 0.01, 0.1, 1, 10, 100], pena

log_reg_eval_0001.fit(X_train[:, indices0001], y_train)
log_reg_eval_001.fit(X_train[:, indices001], y_train)
log_reg_eval_01.fit(X_train[:, indices01], y_train)
log_reg_eval_1.fit(X_train[:, indices1], y_train)
log_reg_eval_10.fit(X_train[:, indices10], y_train)
log_reg_eval_100.fit(X_train[:, indices100], y_train)
```

The results are presented below:



```
Score for 100 features with C=0.001 LR: 0.65
Score for 100 features with C=0.01 LR: 0.88
Score for 100 features with C=0.1 LR: 0.86
Score for 100 features with C=1 LR: 0.81
Score for 100 features with C=10 LR: 0.78
Score for 100 features with C=100 LR: 0.74
```

We see that the performance of logistic regression is highly dependent on the features selection – in case of “bad” regularization we indeed end up having “bad” features.