PCA & Classification of Water Quality

Darren Kang Wan Chee

1

Data Preview

Data Preview

Data Source

The data used for this analysis is a dataset about water quality which is obtained from Kaggle website. The variables of the dataset are:

Aluminum	Copper	Perchlorate
Ammonia	Fluoride	Radium
Arsenic	Bacteria	Selenium
Barium	Viruses	Silver
Cadmium	Lead	Uranium
Chloramine	Nitrates	Nitrites
Chromium	Mercury	Is_safe (0 – not safe, 1 – safe)

Data Structure

The dataset consists of 21 columns and 7999 rows. The variables used for analysis is is_safe as the response variable and the rest of the variables as the predictor variables.

Aluminum	Copper	Ammonia	...	Is safe
X1,1	X2,1	X3,1	...	Y1
X1,2	X2,2	X3,2	...	Y2
...	...	...	...	...
X1,n	X2,n	X3,n	...	Yn

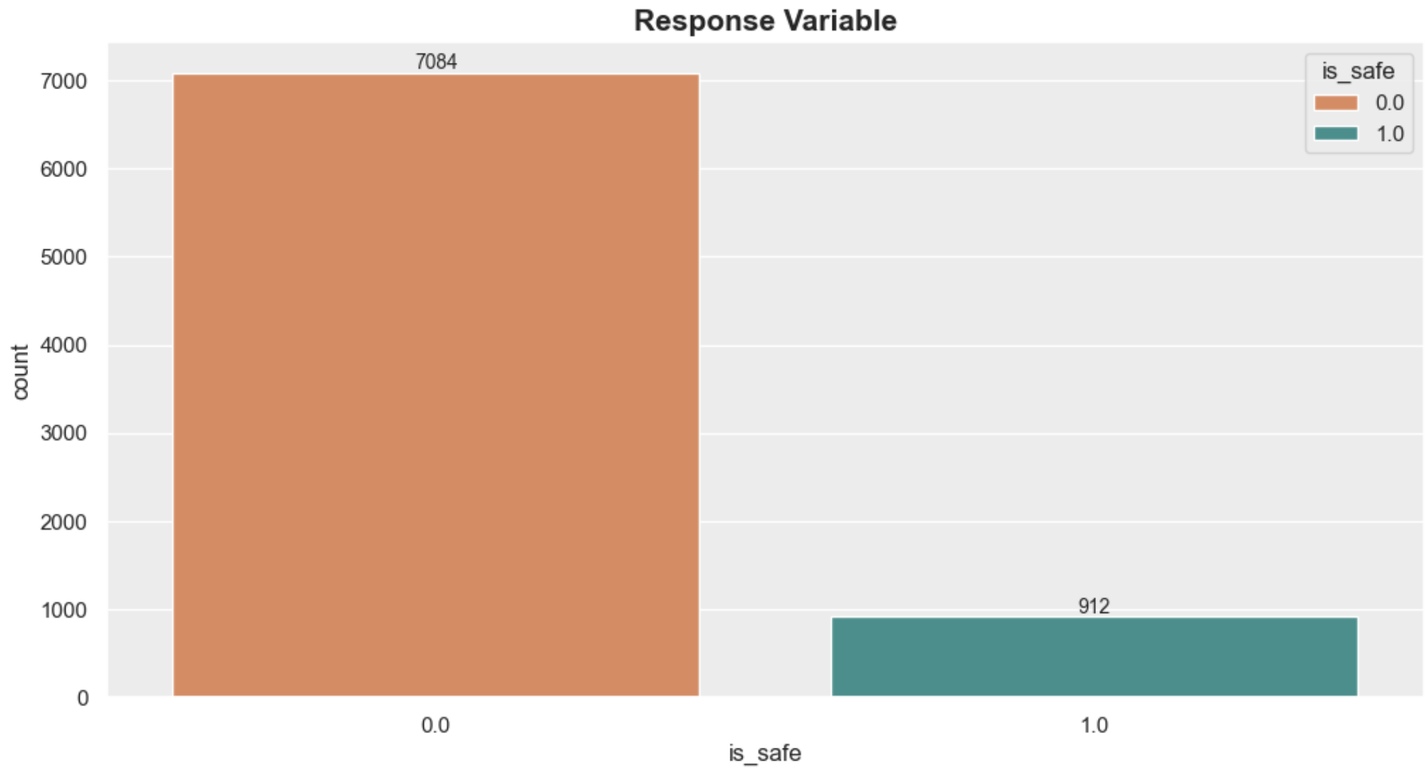
Data Preview

Descriptive Statistics

Variable	Min	Median	Mean	Std	Max
Aluminum	0	0.07	0.666	1.27	5.05
Ammoma	-0.08	14.1	14.3	8.88	29.8
Arsenic	0	0.05	0.161	0.253	1.05
Barium	0	1.19	1.57	1.22	4.94
Cadmium	0	0.04	0.0428	0.036	0.13
Chloramine	0	0.53	2.18	2.57	8.68
Chromium	0	0.09	0.247	0.271	0.9
Copper	0	0.75	0.806	0.654	2
Fluoride	0	0.77	0.772	0.435	1.5
Bacteria	0	0.22	0.32	0.329	1
Viruses	0	0.008	0.329	0.378	1
Lead	0	0.102	0.0994	0.0582	0.2
Nitrates	0	9.93	9.82	5.54	19.8
Nitrites	0	1.42	1.33	0.573	2.93
Mercury	0	0.005	0.00519	0.00297	0.01
Perchlorate	0	7.74	16.5	17.7	60
Radium	0	2.41	2.92	2.32	7.99
Selenium	0	0.05	0.0497	0.0288	0.1
Silver	0	0.08	0.148	0.144	0.5
Uranium	0	0.05	0.0447	0.0269	0.09
Is safe	0	0	0.114	0.318	1

Data Preview

Balance of Response Variable



2

Data Preprocessing

Variable Type

From the table, we can see that the variables ‘ammonia’ and ‘is_safe’ are object types.

We can convert them into numeric data type later on

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7999 entries, 0 to 7998
Data columns (total 21 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   aluminium    7999 non-null   float64
 1   ammonia      7999 non-null   object 
 2   arsenic      7999 non-null   float64
 3   barium       7999 non-null   float64
 4   cadmium      7999 non-null   float64
 5   chloramine   7999 non-null   float64
 6   chromium     7999 non-null   float64
 7   copper       7999 non-null   float64
 8   flouride     7999 non-null   float64
 9   bacteria     7999 non-null   float64
 10  viruses      7999 non-null   float64
 11  lead         7999 non-null   float64
 12  nitrates     7999 non-null   float64
 13  nitrites     7999 non-null   float64
 14  mercury      7999 non-null   float64
 15  perchlorate  7999 non-null   float64
 16  radium       7999 non-null   float64
 17  selenium     7999 non-null   float64
 18  silver       7999 non-null   float64
 19  uranium      7999 non-null   float64
 20  is_safe      7999 non-null   object 
dtypes: float64(19), object(2)
memory usage: 1.3+ MB
```

Missing Value

```
# checking missing value
missing_value = ['#NUM!', np.nan]
data =pd.read_csv('waterQuality1.csv', na_values = missing_value)
data.isnull().sum()
```

✓ 0.0s

```
aluminium      0
ammonia        3
arsenic        0
barium         0
cadmium        0
chloramine     0
chromium       0
copper         0
flouride       0
bacteria       0
viruses        0
lead           0
nitrates       0
nitrites       0
mercury        0
perchlorate    0
radium         0
selenium       0
silver         0
uranium        0
is_safe        3
dtype: int64
```

Note that there are missing values in the datasets. We can proceed to remove the missing values

```
# remove missing value
data.dropna( subset=['ammonia', 'is_safe'], axis=0, inplace=True)
data.isnull().sum()
```

✓ 0.0s

```
aluminium      0
ammonia        0
arsenic        0
barium         0
cadmium        0
chloramine     0
chromium       0
copper         0
flouride       0
bacteria       0
viruses        0
lead           0
nitrates       0
nitrites       0
mercury        0
perchlorate    0
radium         0
selenium       0
silver         0
uranium        0
is_safe        0
dtype: int64
```

3

Assumption Checking

Bartlett Test

H_0 = Variance among groups is equal

H_1 = At least 1 group has a variance that is not equal

```
# bartlett test for homogenity
st.bartlett(data['aluminium'],data['ammonia'], data['arsenic'], data['barium'],data['cadmium'],data['chloramine'],
            data['chromium'], data['copper'], data['flouride'], data['bacteria'], data['viruses'], data['lead'],
            data['nitrates'], data['nitrites'], data['mercury'], data['perchlorate'], data['radium'], data['selenium'],
            data['silver'], data['uranium'],data['is_safe'])
```

✓ 0.2s

```
BartlettResult(statistic=827447.8902262661, pvalue=0.0)
```

From the result we can see that the p-value of the Bartlett Test is 0, we can proceed to **reject the null hypothesis**. Concluding that there is **at least 1 group that has a different variance.**

Multivariate Normality

H_0 = Variables follow the multivariate normal distribution

H_1 = Variables don't follow the multivariate normal distribution

```
# multivariate normality test
multivariate_normality(data, alpha = 0.05)
```

✓ 19.7s

```
HZResults(hz=1.329326759171939, pval=0.0, normal=False)
```

From the result we can see that the p-value of the multivariate normality test is 0, we can proceed to **reject the null hypothesis**. Concluding that the **variables don't follow the multivariate normal distribution.**

KMO Test

```
np.reshape(kmo[1], (7,3))
```

✓ 0.0s

```
array([[0.78740317, 0.672512  , 0.75302935],
       [0.92988414, 0.57840761, 0.89895301],
       [0.91295702, 0.61523988, 0.40569316],
       [0.47810342, 0.42570968, 0.57693071],
       [0.53413116, 0.76885505, 0.46637601],
       [0.89790035, 0.93181318, 0.41386868],
       [0.91778618, 0.51125505, 0.58546328]])
```

KMO Function Source
https://github.com/Sarmentor/KMO-Bartlett-Tests-Python/blob/master/correlation.py

We conduct the KMO Test by using a function for testing KMO. It will return the KMO value and also the value for each variables.

We can proceed to remove the variables with the KMO value less than 0.5

The removed variables are ‘flouride’, ‘bacteria’, ‘viruses’, ‘ mercury’, ‘selenium’

4

PCA & FA

PCA & FA

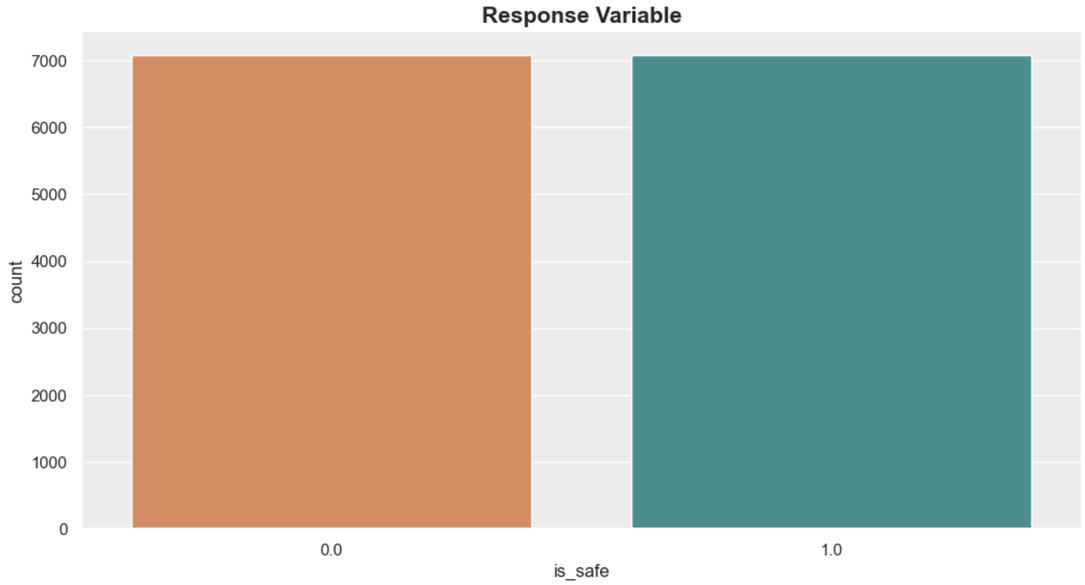
Balancing Response Variable

Since the response variable is not balanced, we proceed to balance the variable. The method used for balancing is Random Over Sampling with sampling strategy minotiry

```
#balance the target variable
columns = data.columns
columns = [c for c in columns if c not in ['is_safe']]
y = data['is_safe'] #prior target variable
X = data[columns]  #prior features

ros = RandomOverSampler(sampling_strategy='minority')
X, y = ros.fit_resample(X, y) #y_train as balanced target variable
print(f"Imbalanced target class: {(y)}\n\nBalanced target class: {Counter(y)}\n")
print(X.shape[0] - data.shape[0], 'new random picked points')
```

✓ 0.0s

```
Imbalanced target class: 0        1.0
1        1.0
2        0.0
3        1.0
4        1.0
        ...
14163    1.0
14164    1.0
14165    1.0
14166    1.0
14167    1.0
Name: is_safe, Length: 14168, dtype: float64

Balanced target class: Counter({1.0: 7084, 0.0: 7084})
```



PCA & FA

Choosing Number of Factors

```
sc = StandardScaler()
X_std = sc.fit_transform(X)
```
✓ 0.0s

Standardized the data

```
fa = FactorAnalyzer()
fa.set_params(n_factors = 16, rotation = None)
fa.fit(X_std)
# Check Eigenvalues
ev, v = fa.get_eigenvalues()
ev
```
✓ 0.0s

```
array([3.51942973, 1.69589176, 1.13088084, 1.02910948, 1.00388945,
       0.9425932 , 0.86540629, 0.77525217, 0.70696263, 0.67458414,
       0.62935166, 0.59616887, 0.53785154, 0.52164639, 0.37098187])
```

Obtain the eigenvalues of all variables and select those with eigenvalues > 1

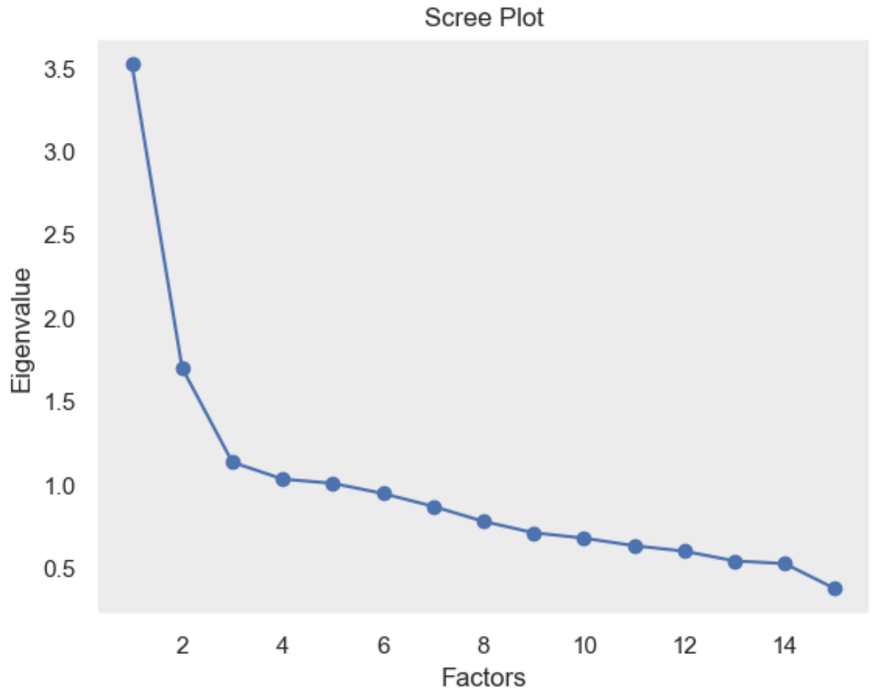
```
pca=decomposition.PCA(n_components=5)

#persamaan PC baru
X_std_pca=pca.fit_transform(X_std)
X_std_pca
```
✓ 0.1s

```
array([[ 1.76352305e+00, -1.08715992e-01, -6.55272265e-01,
        -4.63479370e-01,  4.25123491e-01],
       [ 2.48525620e+00, -5.56754783e-01, -1.15436210e+00,
        -1.81908227e-01, -5.32463759e-01],
       [ 1.73541382e+00, -1.98200375e-01, -7.61078394e-01,
        -5.97509723e-01,  4.98654073e-02],
       ...,
       [-1.49086312e-01, -6.45704218e-01,  1.03090730e-03,
         9.61283960e-01, -1.30317670e+00],
       [ 3.58076833e+00, -1.22235102e+00,  2.43338687e-01,
        -1.21702127e+00, -8.40851878e-01],
       [ 1.57878763e+00, -2.19691373e+00, -2.96893992e-01,
         3.11034906e-01,  3.71423576e-01]])
```

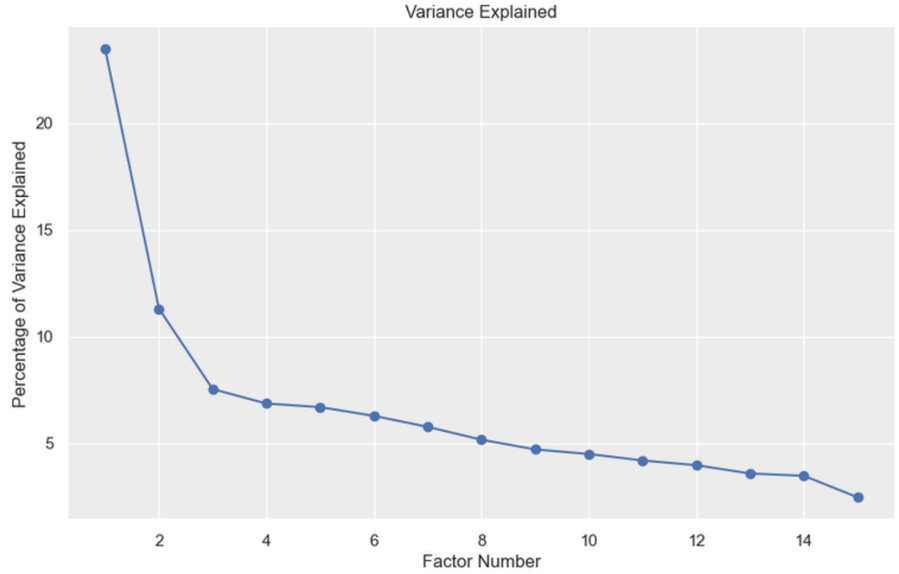
Resulting in 5 components and proceed to obtain the PC values

PCA & FA

Scree Plot



Scree plot of eigenvalue



Scree plot of Variance Explained

5

Classification Analysis

Classification Analysis

Train Test Split

```
#preparation of train and test data for K-NN
np.random.seed(13)
X_train, X_test, y_train, y_test = train_test_split(X_std_pca, y, test_size=0.2, random_state=1)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.0s

```
((11334, 5), (2834, 5), (11334,), (2834,))
```

Splitting the data where y is the response variable and X is the PCs

The splitting is 80% training and 20% testing with randomization

Classification Analysis

K-Nearest Neighbors

```
#implement K-NN algorithm
knnc = KNeighborsClassifier()
knnc.fit(X_train, y_train)
y_pred_knnc = knnc.predict(X_test)

#plotting Confusion Matrix
cf_matrix_knnc = confusion_matrix(y_test, y_pred_knnc)
print(cf_matrix_knnc)

ax = sns.heatmap(cf_matrix_knnc/np.sum(cf_matrix_knnc), annot=True, fmt='.2%', cmap='binary')
ax.set_title('K-NN Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

✓ 0.4s

	New 0	New 1
0	1075	315
1	16	1428

Model Evaluating Metrics

	Accuracy	Precision	Recall	FI Score
New 0	0.883	0.99	0.77	0.87
New 1	0.883	0.82	0.99	0.90

Classification Analysis

K-NN Hyperparameter Tuning

```
#hyperparameter tuning
param_grid = {'n_neighbors':np.arange(1,40), 'metric':['euclidean', 'manhattan', 'minkowski']}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X_train,y_train)

y_pred_knn_cv = knn_cv.predict(X_test)

print("Best Score:" + str(knn_cv.best_score_))
print("Best Parameters: " + str(knn_cv.best_params_))

cf_matrix_knn_cv = confusion_matrix(y_test, y_pred_knn_cv)
print(cf_matrix_knn_cv)

ax = sns.heatmap(cf_matrix_knn_cv/np.sum(cf_matrix_knn_cv), annot=True, fmt='.2%', cmap='binary')
ax.set_title('K-NN Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

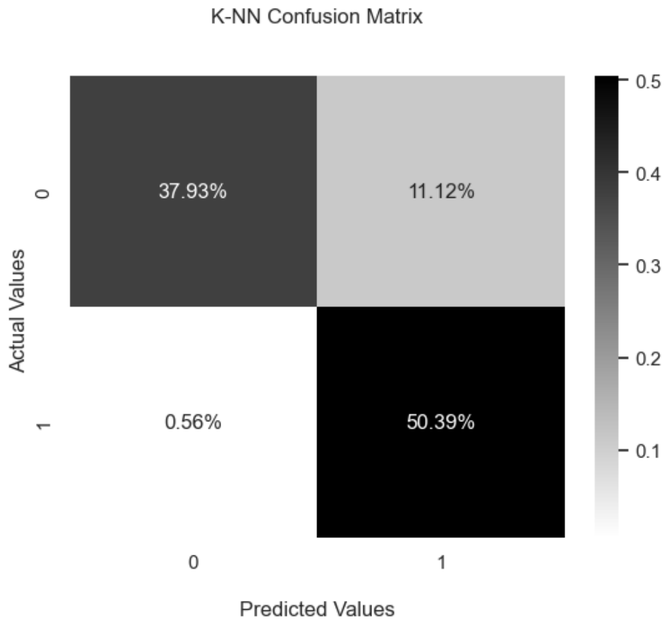
	New 0	New 1
0	1234	156
1	3	1441

Model Evaluating Metrics

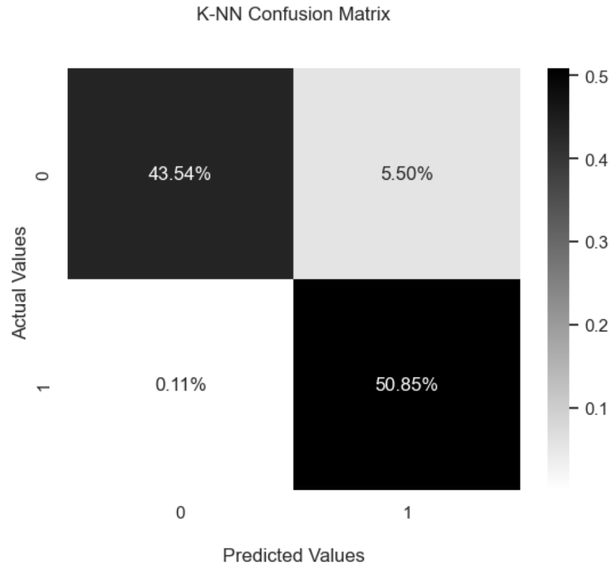
	Accuracy	Precision	Recall	FI Score
New 0	0.944	1.00	0.89	0.94
New 1	0.944	0.90	1.00	0.95

Classification Analysis

K-NN Confusion Matrix



K-NN Confusion Matrix



K-NN Hyperparameter Tuning Confusion Matrix

Classification Analysis

K-NN AUC Score

AUC Score of K-NN Classifier

	AUC Score
K-NN Classifier	0.881
K-NN after Hyperparameter Tuning	0.943

Classification Analysis

Support Vector Machine

```
# implementing SVM
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)
y_pred_svm = clf.predict(X_test)

#plotting Confusion Matrix
cf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
print(cf_matrix_svm)

ax = sns.heatmap(cf_matrix_svm/np.sum(cf_matrix_svm), annot=True, fmt='.2%', cmap='binary')
ax.set_title('SVM Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

✓ 12.1s

	New 0	New 1
0	1044	346
1	337	1107

Model Evaluating Metrics

	Accuracy	Precision	Recall	FI Score
New 0	0.759	0.76	0.75	0.75
New 1	0.759	0.7	0.77	0.76

Classification Analysis

SVM Hyperparameter Tuning

```
#hyperparameter tuning
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
svm = SVC()
svm_cv = GridSearchCV(svm,param_grid,cv=5)
svm_cv.fit(X_train,y_train)

y_pred_svm_cv = svm_cv.predict(X_test)

print("Best Score:" + str(svm_cv.best_score_))
print("Best Parameters: " + str(svm_cv.best_params_))

cf_matrix_svm_cv = confusion_matrix(y_test, y_pred_svm_cv)
print(cf_matrix_svm_cv)

ax = sns.heatmap(cf_matrix_svm_cv/np.sum(cf_matrix_svm_cv), annot=True, fmt='.2%', cmap='binary')
ax.set_title('SVM Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

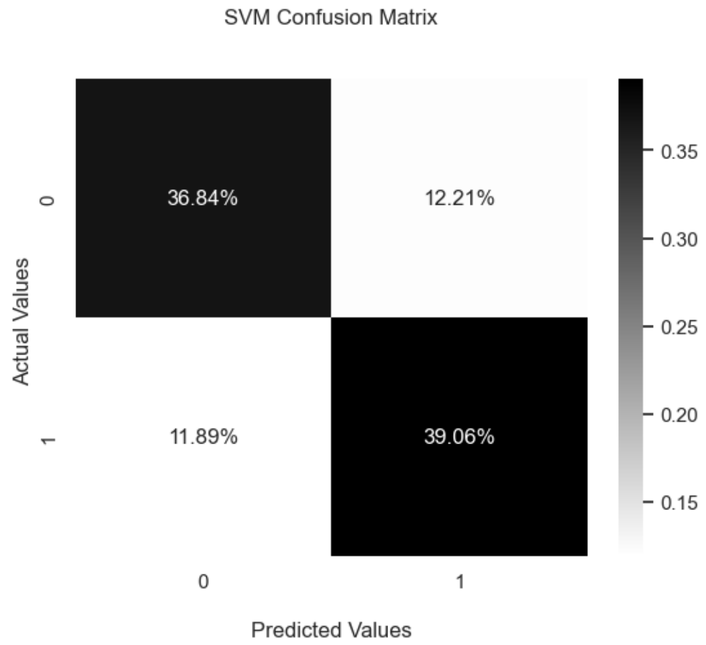
	New 0	New 1
0	1228	162
1	4	1440

Model Evaluating Metrics

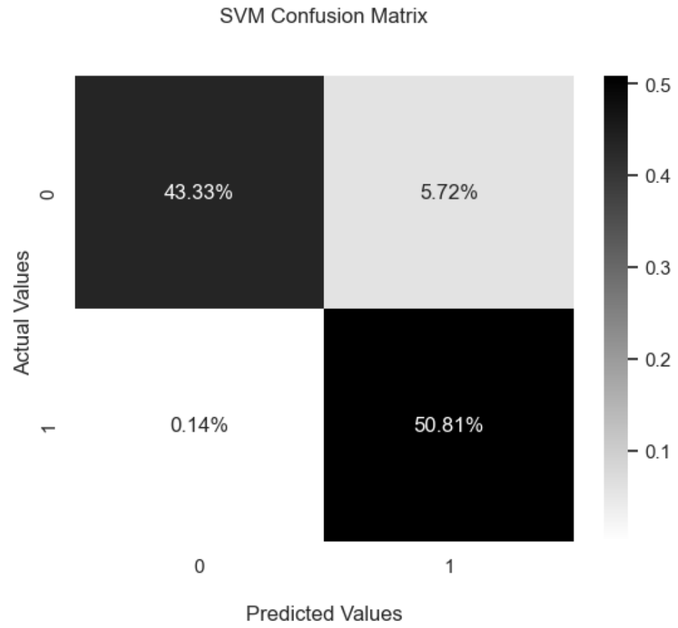
	Accuracy	Precision	Recall	FI Score
New 0	0.941	1.00	0.88	0.94
New 1	0.944	0.90	1.00	0.95

Classification Analysis

SVM Confusion Matrix



SVM Confusion Matrix



SVM Hyperparameter Tuning Confusion Matrix

Classification Analysis

SVM AUC Score

AUC Score of SVM Classifier

	AUC Score
SVM Classifier	0.759
SVM after Hyperparameter Tuning	0.94