WQD7006

MACHINE LEARNING FOR DATA SCIENCE

2022/2023 SEMESTER 2

RICE TYPE CLASSIFICATION

GROUP ASSIGNMENT

ML4DS G1(RL) – Group 4		
No.	**MATRIC ID**	**NAME**
1	17186722	Chen Bao Gang
2	22056764	Devayani A/P Balkrishnan
3	S2195613	Li Xin Qi
4	S2155659	Lim Yu Xuan
5	S2192763	Navaneeta A/P P Shanmugam

Table of Content

1 Introduction

1.1 Project Background

Rice is rich in carbohydrates and starch that is essential for meeting our dietary needs, providing energy for people in carrying out daily activities. With rice consumption across half of the world population, it is deemed as a highly produced crop for global food security (Gurrala, 2021). The global rice annual supply and consumption amounted to over 700 million metric and 502 million metric tons of rice respectively in recent years (*Major Rice Exporting Countries Worldwide 2022/2023 | Statista*, 2023). Without rice production, it leads to poverty, hunger, social stability problems and adversely impacts globally (Rahman & Zhang, 2022).

Rice type classification is becoming crucial for guaranteeing a sustainable and consistent supply. It involves the process of classifying rice into distinct types depending on characteristics, including grain size, shape, color, quality, degree of milling, scent, and cooking quality. It ensures that consumers and traders can precisely market rice based on its quality, maintaining fair competition in the market while stabilizing economy. Farmers and producers can make informed decisions about which rice varieties to cultivate based on factors such as market demand, environmental adaptability, yield potential, quality characteristics, pests, and diseases resistance (*Variety Selection*, 11 B.C.E.).

Nowadays, the trend in rice classification is shifting towards advanced analytical techniques and machine learning (ML) algorithms to improve efficiency, accuracy and consistency while fulfilling consumer's needs. ML models are trained via rice samples to distinguish the feature types, then categorize new rice samples based on given features and properties. Recent studies applied image processing techniques to extract features from rice images and employed multivariate data analysis like Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) along with ML algorithms, i.e., Support Vector Machines (SVM), k-Nearest Neighbor (KNN), Artificial Neural Networks (ANN) to automate the classification process. Besides, researchers focused on applying molecular, atomic spectroscopy, chromatography, and mass spectrometry to analyze chemical components and contaminants of rice samples, then input as parameters for rice discrimination. However, most studies involved small datasets with limited samples for each rice variety, and only a few emphasized feature selections (Maione & Barbosa, 2019) and explored mobile application for rice type classification.

1.2 Problem Statement

According to Fitch Solutions, global rice cultivation in 2023 is predicted to hit the lowest since 2003, resulting in insufficient rice supply to satisfy expanding rice demand, which may lead to a worldwide rice crisis (Thornton, 2023). In this project, we proposed the study of rice type classification by implementing a ML framework, applying, and evaluating various classification models such as SVM, KNN, Decision Tree, Random Forest and Logistic Regression. The best classification model will be deployed in the mobile application named Ricense as per the prototypes demonstrated for future development.

1.3 Project Objectives

The main objectives of this project are outlined as below:
1. To propose a ML framework to classify rice types.
2. To evaluate the best ML algorithm for rice type classification.
3. To design the prototype of the mobile application for rice type classification.

2 Dataset description and pre-processing (Practical)

2.1 Dataset description

The dataset, available on Kaggle at https://www.kaggle.com/datasets/mssmartypants/rice-type-classification, consists of 12 variables that describes the rice information, which includes 1 unique identifier, 10 distinct features (Area to AspectRatio) and 1 label variable (Class).

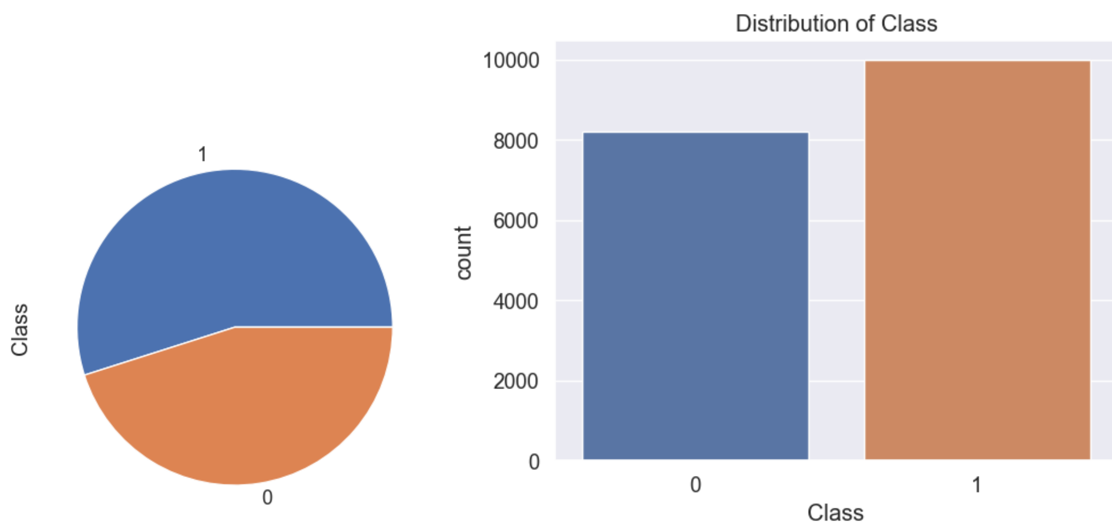
	Variable	**Description**
	Rice Information	
1	Id	The unique identifier for each rice
2	Area	The size or surface area of the rice grain.
3	MajorAxisLength	The length of the major axis of the rice grain.
4	MinorAxisLength	The length of the minor axis of the rice grain.
5	Eccentricity	The elongation or circularity of the rice grain shape.
6	ConvexArea	The area of the smallest convex polygon that can completely enclose the rice grain.
7	EquivDiameter	The diameter of a circle with the same area as the rice grain.
8	Extent	the ratio of how much the rice grain fills the bounding box
9	Perimeter	The total length of the boundary or contour of the rice grain.
10	Roundness	The similarity of the rice grain shape to a perfect circle.
11	AspectRation	The ratio of the major axis length to the minor axis length.
	Target Variable	
12	Class	The type of rice Jasmine - 1, Gonen - 0

Table 2.1: Variables' Description

1. The dataset contains **10 features** (Area to AspectRation) and **1 label** (Class)

2. There are **18,185 entries** of data which is quite perfect for ML
3. **Most** of the **features** are of **float64** datatype
4. There are **no missing elements** in any features, and the dataset is quite clean

2.2 Bivariate analysis: The Distribution of the Class Column



Figures 2.2.1 & 2.2.2: Distribution of Class

There is no significant imbalance or bias towards any specific class within the dataset. As a result, sampling techniques such as oversampling or undersampling are not required to address class imbalance.

2.3 Detect outliers: IQR

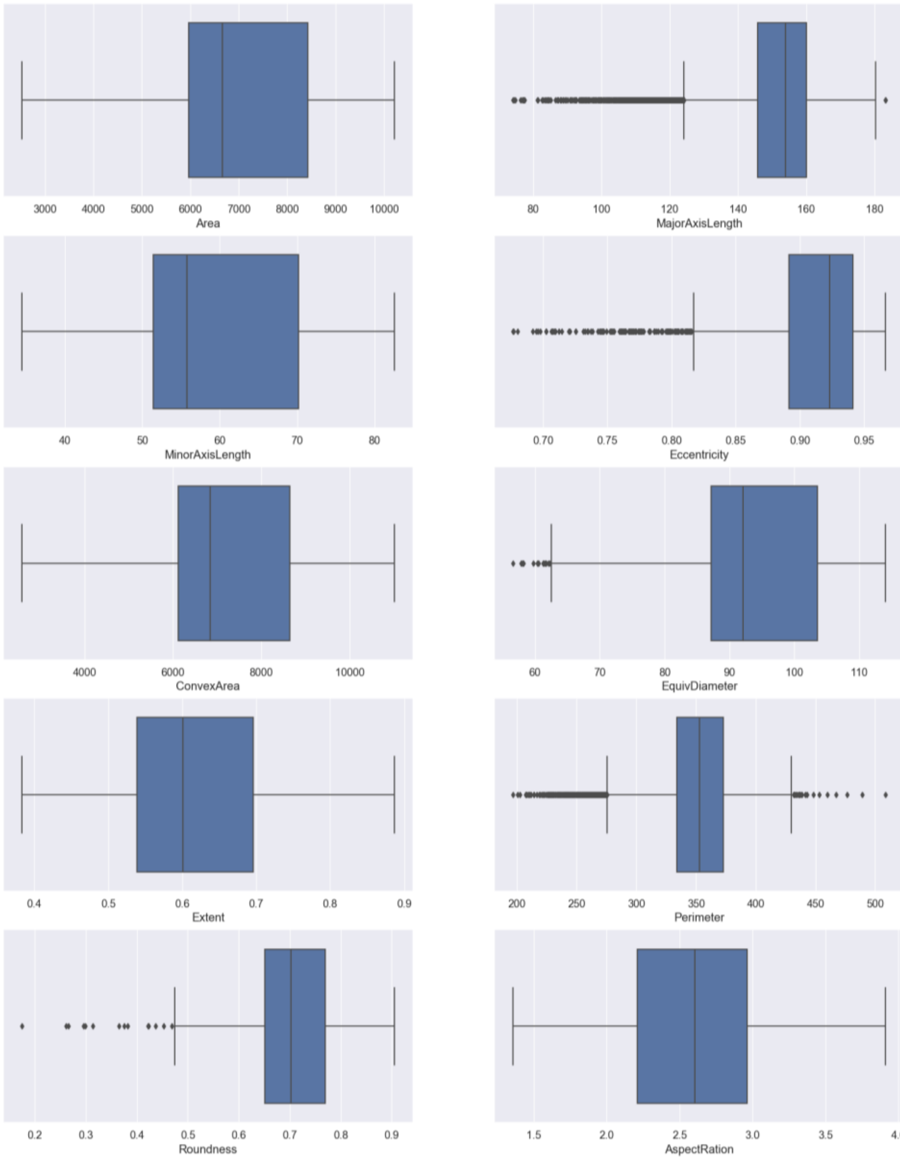


Figure 2.3.1: Box Plot of each feature before Outlier Detection

Outliers are detected in MajorAxisLength, Eccentricity, EquivDiameter, Perimeter, and Roundness with deviations from normal distribution. Only AspectRation exhibits a normal distribution pattern, while the remaining variables illustrate skewed distribution. Hence, removing outliers is essential to increase the performance of classification model.

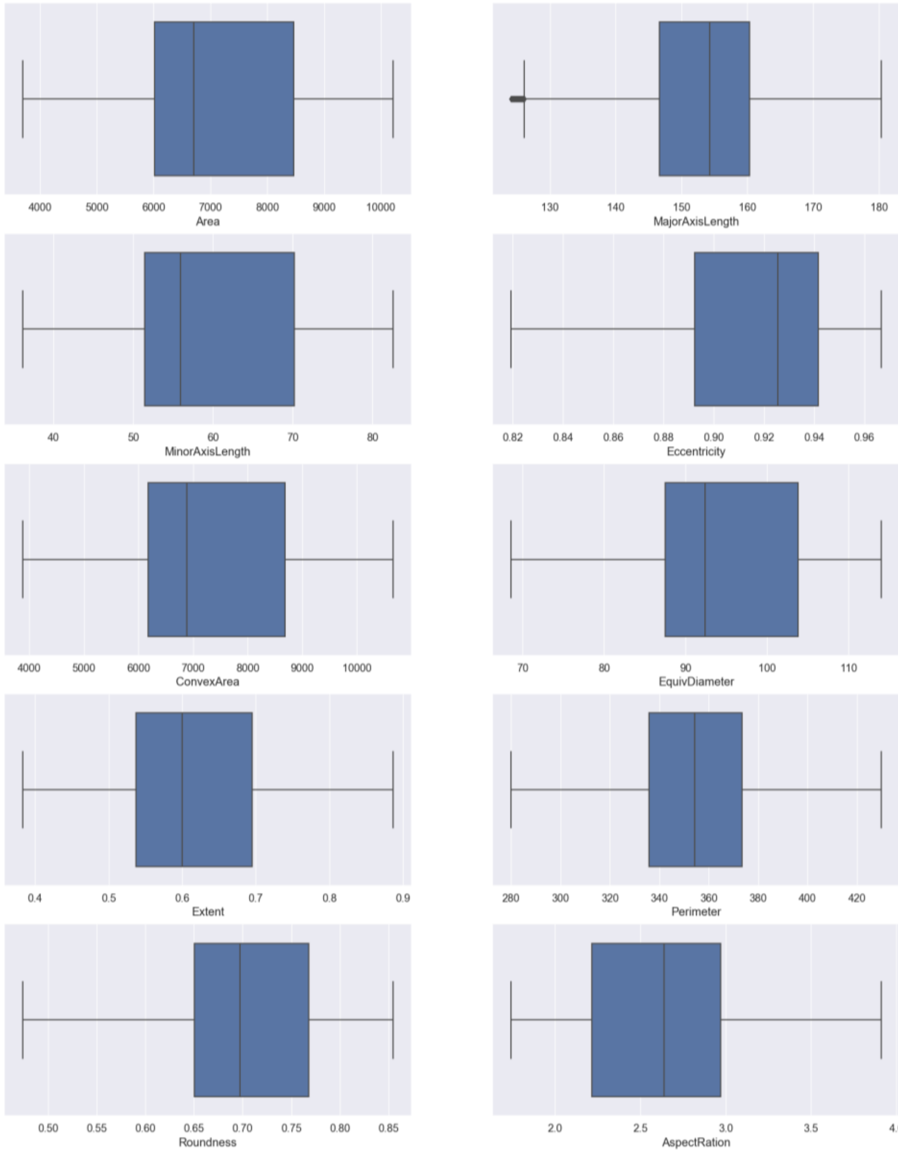


Figure 2.3.2: Box Plot of each feature after removing Outlier

After removing the outliers, the boxplots reveal a significantly cleaner representation of the data, and extreme values which skew the distribution are removed.

2.4 Multivariate Analysis: Heatmap

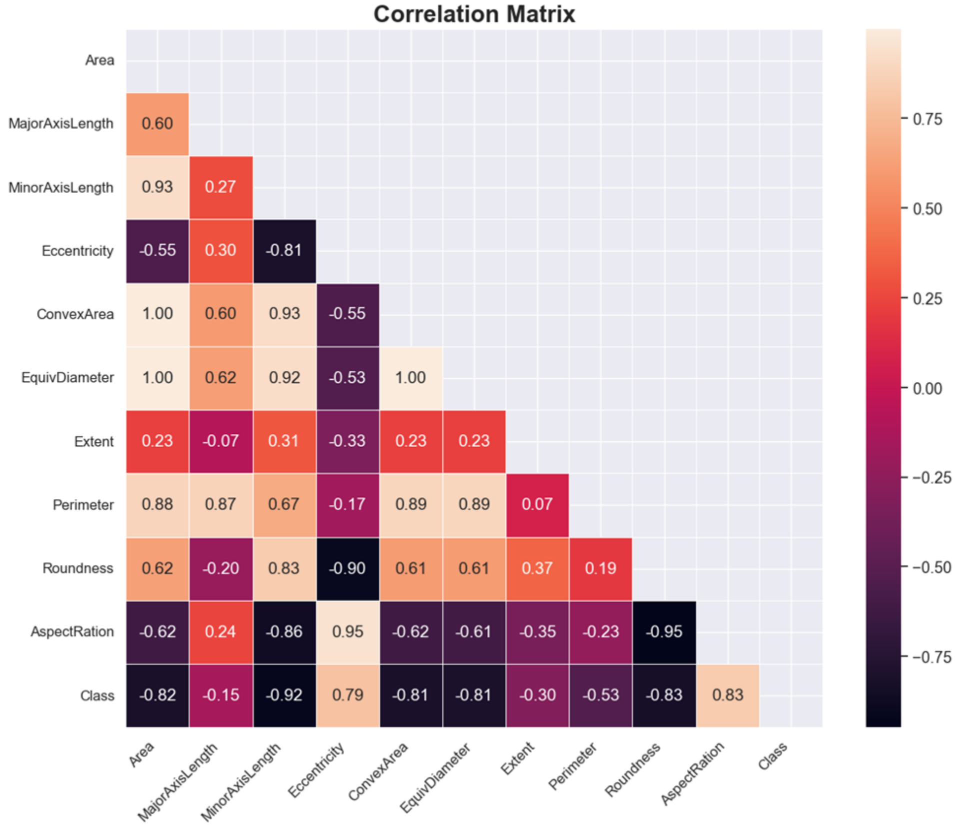


Figure 2.4: Correlation Matrix

The analysis reveals notable correlations within the dataset, where certain features exhibit significant correlation with the target variable. MinorAxisLength, AspectRatio, Roundness, Area, ConvexArea, EquivDiameter, and Eccentricity demonstrate particularly strong correlations with the target variable.

2.5 Check Multicollinearity: Variable Inflation Factors (VIF)

Unlike the correlation matrix, VIF measures the degree of correlation between a variable and other independent variable. VIF values typically begin at 1, and a value exceeding 10 is commonly considered indicative of substantial multicollinearity among the independent variables.

	feature	VIF
0	const	103954.790226
1	Area	2347.459870
2	MajorAxisLength	318.714833
3	MinorAxisLength	1296.490608
4	Eccentricity	51.971818
5	ConvexArea	1813.854237
6	EquivDiameter	2844.995049
7	Extent	1.157557
8	Perimeter	346.462939
9	Roundness	157.668241
10	AspectRation	118.998499

Figure 2.5.1: All Features and its VIF

VIF value of 10 or higher is often considered significant multicollinearity between the independent variables. From the figure, it is shown that there are some features with extremely high VIF values.

	feature	VIF
0	const	8707.834300
1	MajorAxisLength	1.123905
2	Roundness	5.753991
3	Eccentricity	5.876661
4	Extent	1.155467

Figure 2.5.2 4: Features and its VIF

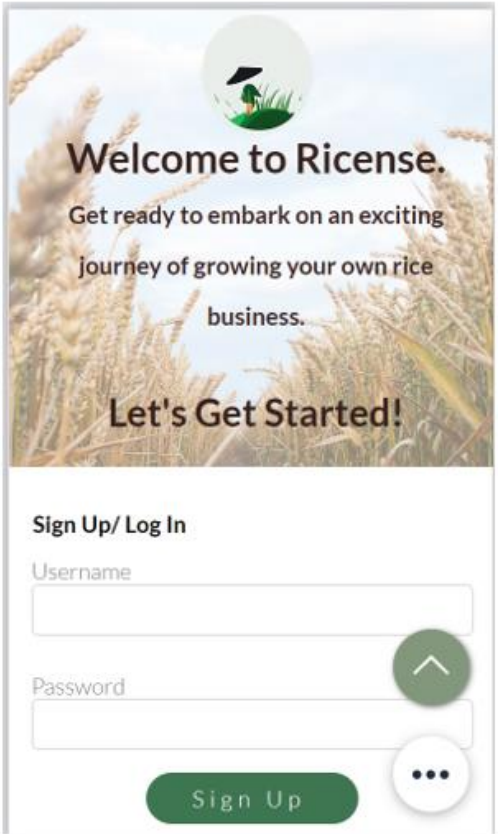
After selecting MajorAxisLength, Roundness, Eccentricity and Extent, all the independent variables have decreased to a reasonable extent (< 10).

3 Mobile application prototype design (5 Screens)

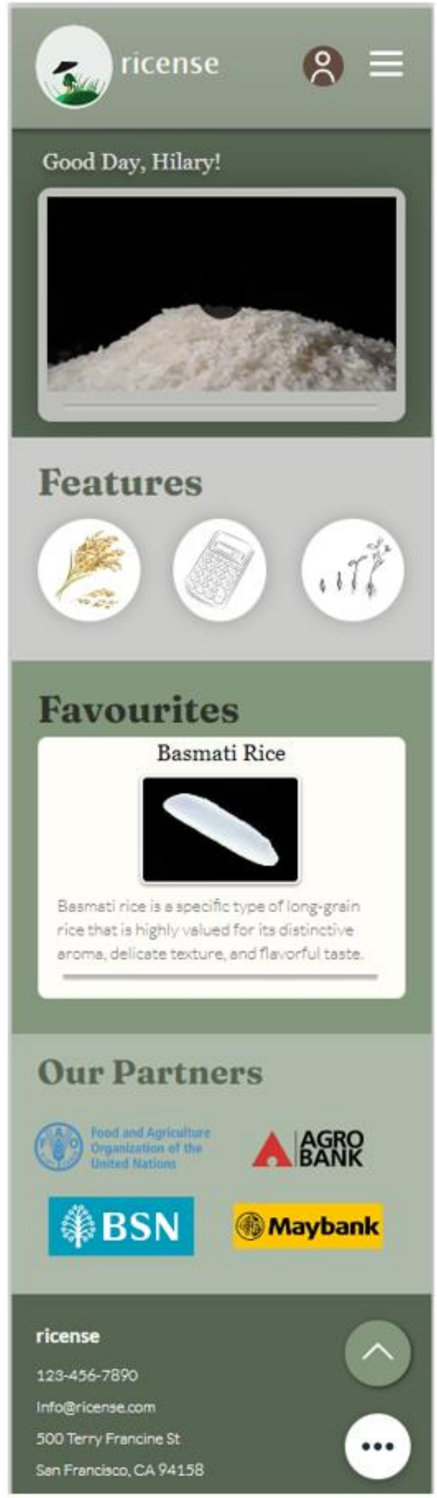
ricense

Website Link: https://wix.to/fBY2eL7

Page 1: Login Page



Page 2: Get Started



Page 3: Rice Type Classification

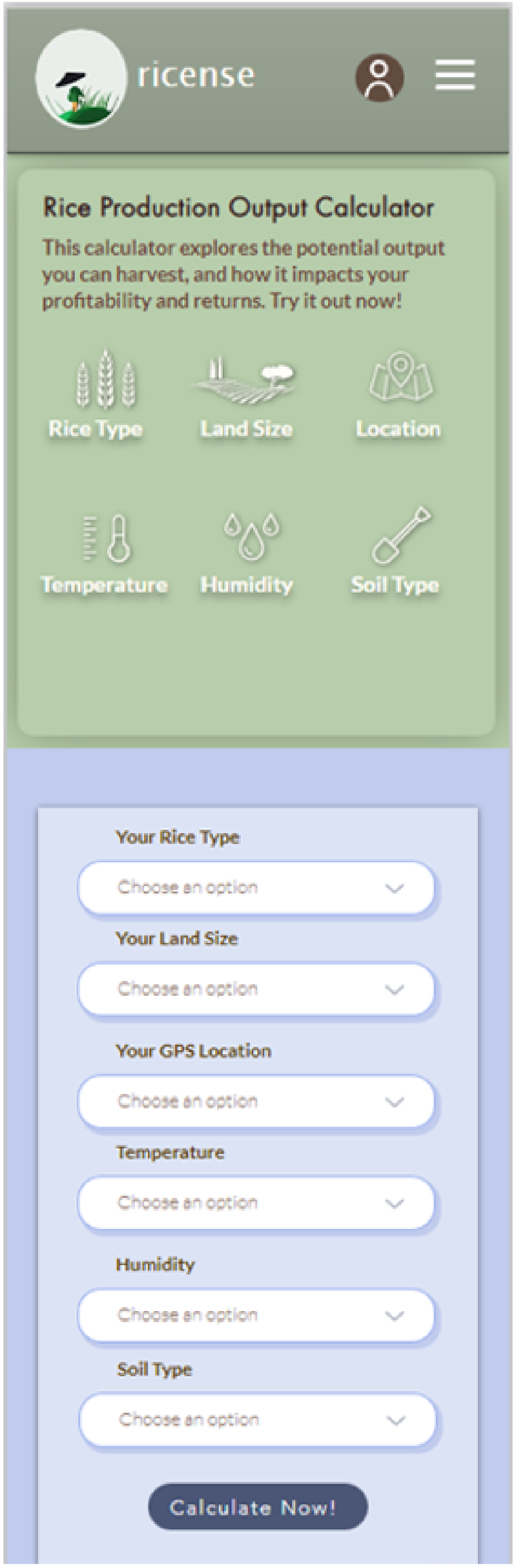
Step 1: Tap on the camera to capture.

Step 2: Tap 'Load' after captured rice picture.

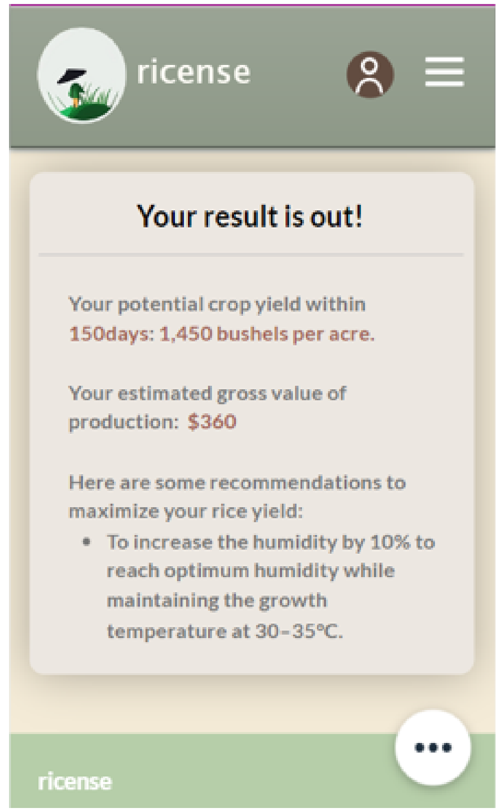
Step 3: Rice type result and description as shown below.

Page 4: Rice Production Output Calculator

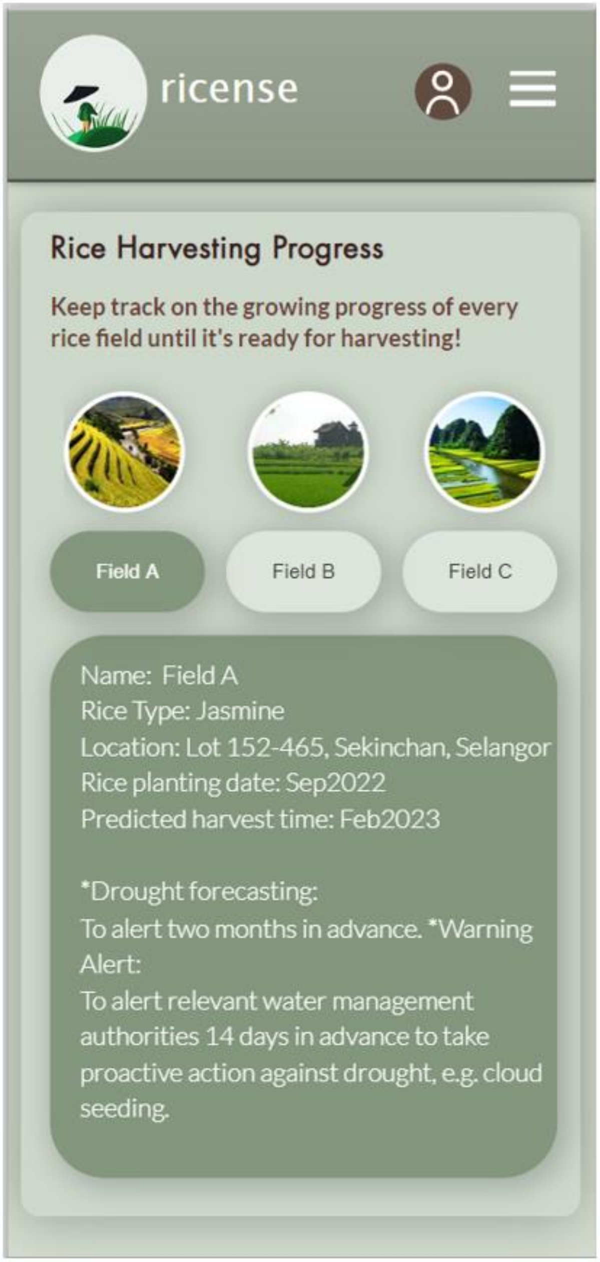
Step 1: Fill up rice production details and click 'Calculate now'.



Step 2: Rice profitability and return result as shown below.

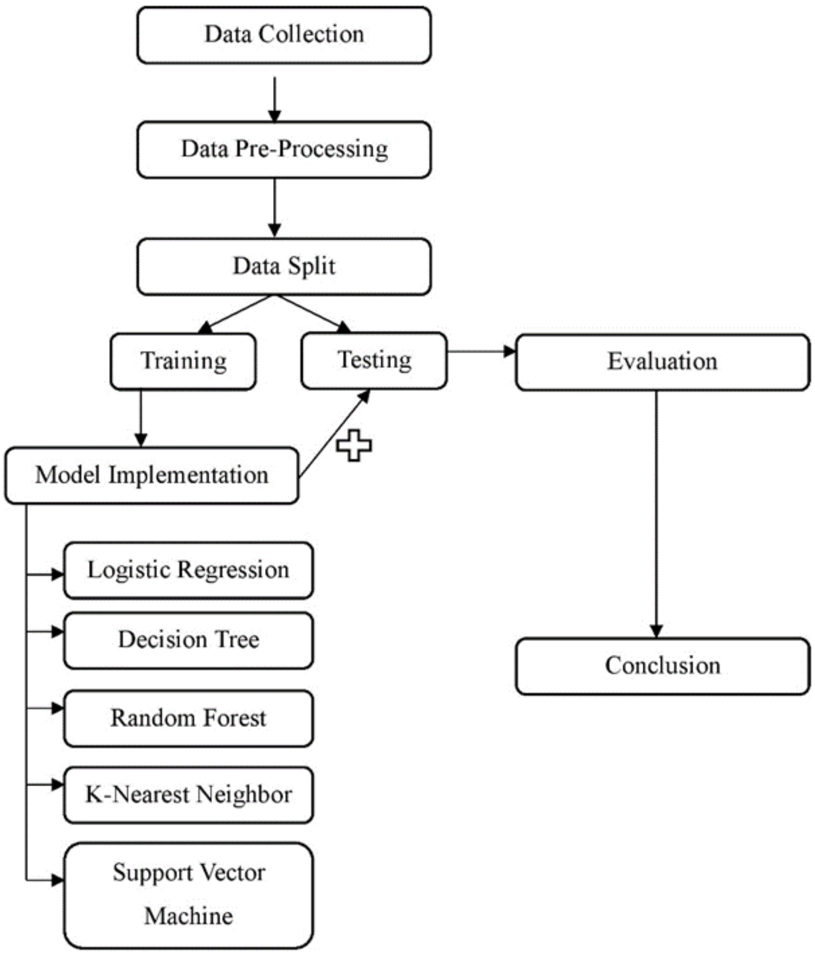


Page 5: Rice Harvesting Progress



4 Machine learning model diagram and explanation

4.1 Model Diagram



4.2 Explanation

The rice type classification dataset obtained from Kaggle consists of 12 features with 18,185 records. The data is then pre-processed by removing outliers in variables like MajorAxisLength and Eccentricity and reducing multicollinearity between correlated features via VIF. Thus, all features are dropped except MajorAxisLength, Roundness, Eccentricity, and Extent to reduce the redundancy in dataset. The dataset is now ready for data splitting, where 70% of the records are randomly chosen for training and the remaining 30% for testing.

Five ML algorithms, which are Logistic Regression, K-Nearest Neighbor, Decision Tree, Random Forest, and Support Vector Machine, will be applied to implement the classification model. These models' performance will be evaluated using the accuracy, confusion matrix, precision, recall, F1-score, K-Folds cross-validation, and ROC curve, and the best model will be identified based on these metrics.

4.2.1 Logistic Regression

Despite its name, logistic regression is a classification model rather than a regression model. Logistic regression is a straightforward and effective method for dealing with binary and linear classification problems (Subasi, 2020). It is a classification model that is simple to implement and delivers excellent results with linearly separable classes. It is a widely used classification method in industry. The logistic regression model is a binary classification statistical method that can be generalised to multiclass classification. Scikit-learn includes a highly optimised logistic regression implementation that can handle multiclass classification tasks (Raschka & Mirjalili, 2019).

4.2.2 K-NN

A k-Nearest Neighbour (K-NN) algorithm is a supervised classification model that allows users to change both the distance measure and the number of nearest neighbours. K-NNs work solely on feature similarities and make no assumptions about the underlying data distribution. The class of an unclassified point can be determined by counting the majority class from its k-nearest neighbour training points (Ghiasi et al., 2022).

4.2.3 Decision Tree

Decision trees are used to generate a training model that can be used to predict the class or value of a target variable. They start at the root and compare the values of the attribute on the record with those of the root attribute. Every node in the tree represents a test case for a specific property, and every edge descending from the node symbolizes numerous potential solutions to the test case (Jijo & Abdulazeez, 2021).

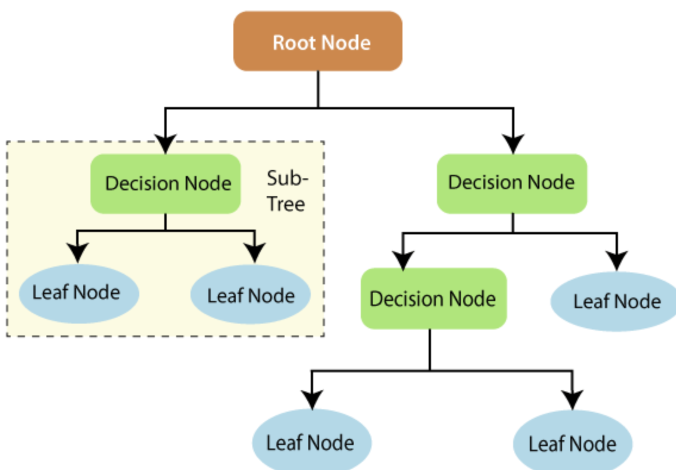


Figure 4.2.3.1: Types of nodes in a Decision Tree

4.2.4 Random Forest

Random forest (RF) is a hierarchical grouping of base classifiers with a tree-structure, using ensemble learning to solve complex problems (Jackins et al., 2020). It creates a forest using bagging or bootstrap aggregation and makes predictions by averaging the results from different trees.

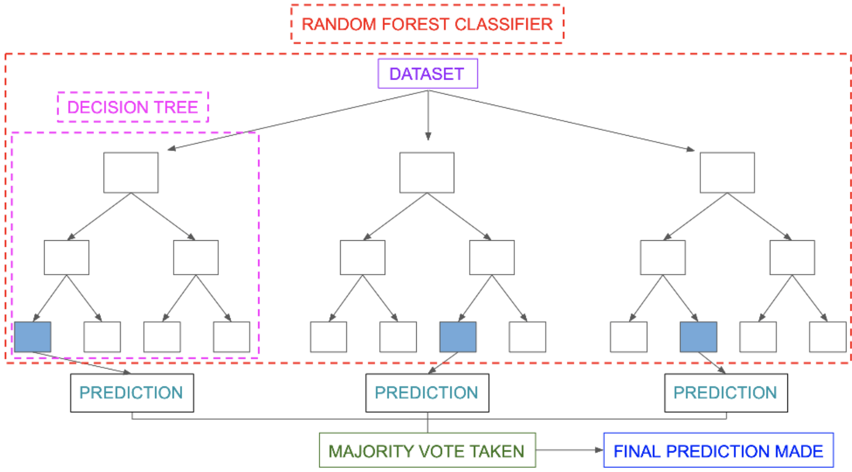


Figure 4.2.4.1: Simplified Working Principle of Random Forest Classifier

4.2.5 Support Vector Machine

The Support Vector Machine (SVM) algorithm is a mathematical model used for regression and classification of input data (Kader et al., 2019). It represents each data point as a point in n-dimensional space and uses a hyper-plane to distinguish between two classes. This algorithm offers elevated levels of classification accuracy and efficiency with large-scale datasets.

4.2.6 Evaluation Metrics

Accuracy, confusion matrix, precision, recall, F1-score, K-Folds cross-validation, and ROC curve are used in this project. The accuracy is the percentage of correctly classified data ranges between 0 and 1. Using a N x N matrix known as a confusion matrix, where N is the total number of target classes, one can assess the effectiveness of a classification model. High TP and TN rates and low FP and FN rates are indicators of a strong model. Positive predictions' accuracy is gauged by a statistic called precision. It is calculated by dividing true positive predictions with the sum of true positive predictions and false positive predictions. On the other hand, recall gauges how comprehensively accurate forecasts are. The recall is calculated by dividing the true positive predictions with the sum of true positive predictions and false negative predictions. The F1-score combines a model's precision and recall scores. How many times a model correctly predicted throughout the full dataset is determined by the accuracy statistic.

A statistical technique called cross-validation compares and evaluates learning algorithms by splitting the data into two groups. With K-Folds Cross-validation, users can split the training data into k-folds, each of which will be utilized as a validation set in training the other (k-1) folds together as training set.

Since the dataset utilized in this study has a balanced class distribution, ROC curves can be used (Saito & Rehmsmeier, 2015). ROC depicts the performance of a classification model over all classification thresholds. AUC is an abbreviation for "Area Under the ROC Curve." It is the two-dimensional region beneath the whole ROC curve from (0,0) to (1,1) and is a metric that aggregates performance across all classification thresholds.

5 Model practical implementation and comparisons (Practical)

5.1 Model Implementation

The dimensionality of the dataset was reduced into X data frame (containing MajorAxisLength, Roundness, Eccentricity, and Extent) and Y data frame (Class variable). These dataframes were split into 70% training data and 30% testing data. X_test and X_train were then standardized.

Scikit-Learn library was used to import all the models and the metrics. First, each of the five models were initialized. Then, each model is trained with the X_train and y_train datasets. The trained model is then used to predict the X_test dataset. The accuracy value of each model was obtained. Next, the confusion matrix and classification report were generated.

```
In [86]:
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Logistic Regression

```python
logreg = LogisticRegression()
log_model = logreg.fit(X_train, y_train)
```

```python
y_pred_log = logreg.predict(X_test)
y_proba_log = log_model.predict_proba(X_test)[:, 1]
acc_log = logreg.score(X_test, y_test)
print('Accuracy of Logistic Regression classifier on test set: {:.7f}'.format(acc_log))
```

```
Accuracy of Logistic Regression classifier on test set: 0.9903409
```

```python
cm_log = confusion_matrix(y_test, y_pred_log)
print(cm_log)
```

```
[[2371   38]
 [  13 2858]]
```

```python
print(classification_report(y_test, y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      2409
           1       0.99      1.00      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

Decision Tree

```python
dt = DecisionTreeClassifier()
dt_model = dt.fit(X_train, y_train)
```

```python
y_pred_dt = dt.predict(X_test)
y_proba_dt = dt_model.predict_proba(X_test)[:, 1]
acc_dt = dt.score(X_test, y_test)
print('Accuracy of Decision Tree classifier on test set: {:.7f}'.format(acc_dt))
```

```
Accuracy of Decision Tree classifier on test set: 0.9831439
```

```python
cm_dt = confusion_matrix(y_test, y_pred_dt)
print(cm_dt)
```

```
[[2367   42]
 [  47 2824]]
```

```python
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2409
           1       0.99      0.98      0.98      2871

    accuracy                           0.98      5280
   macro avg       0.98      0.98      0.98      5280
weighted avg       0.98      0.98      0.98      5280
```

K-NN

```python
knn = KNeighborsClassifier(n_neighbors=7)
knn_model = knn.fit(X_train, y_train)
```

```python
y_pred_knn = knn.predict(X_test)
y_proba_knn = knn_model.predict_proba(X_test)[:, 1]
acc_knn = knn.score(X_test, y_test)
print('Accuracy of K-NN classifier on test set: {:.7f}'.format(acc_knn))
```

```
Accuracy of K-NN classifier on test set: 0.9903409
```

```python
cm_knn = confusion_matrix(y_test, y_pred_knn)
print(cm_knn)
```

```
[[2369   40]
 [  11 2860]]
```

```python
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2409
           1       0.99      1.00      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

Random Forest

```python
rf = RandomForestClassifier()
rf_model = rf.fit(X_train,y_train)
```

```python
y_pred_rf = rf.predict(X_test)
y_proba_rf = rf_model.predict_proba(X_test)[:, 1]
acc_rf = rf.score(X_test, y_test)
print('Accuracy of Random Forest classifier on test set: {:.7f}'.format(acc_rf))
```

```
Accuracy of Random Forest classifier on test set: 0.9912879
```

```python
cm_rf = confusion_matrix(y_test, y_pred_rf)
print(cm_rf)
```

```
[[2378   31]
 [  15 2856]]
```

```python
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2409
           1       0.99      0.99      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

Support Vector Machine

```python
svc = SVC(probability=True)
svc_model = svc.fit(X_train,y_train)
```

```python
y_pred_svc = svc.predict(X_test)
y_proba_svc = svc_model.predict_proba(X_test)[:, 1]
acc_svc = svc.score(X_test, y_test)
print('Accuracy of Support Vector classifier on test set: {:.7f}'.format(acc_svc))
```

```
Accuracy of Support Vector classifier on test set: 0.9899621
```

```python
cm_svc = confusion_matrix(y_test, y_pred_svc)
print(cm_svc)
```

```
[[2370   39]
 [  14 2857]]
```

```python
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      2409
           1       0.99      1.00      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

Figure 5.1.1: Code snippets for each model

For KNN, the value of k was chosen by plotting Error Rate versus K Value and Accuracy versus K Value. The optimal k value is 7 due to it having the minimum error and maximum accuracy.

Minimum error:- 0.00946969696969697 at K = 7

Figure 5.1.2: Graph of Error Rate vs. K Value

Maximum accuracy:- 0.990530303030303 at K = 7

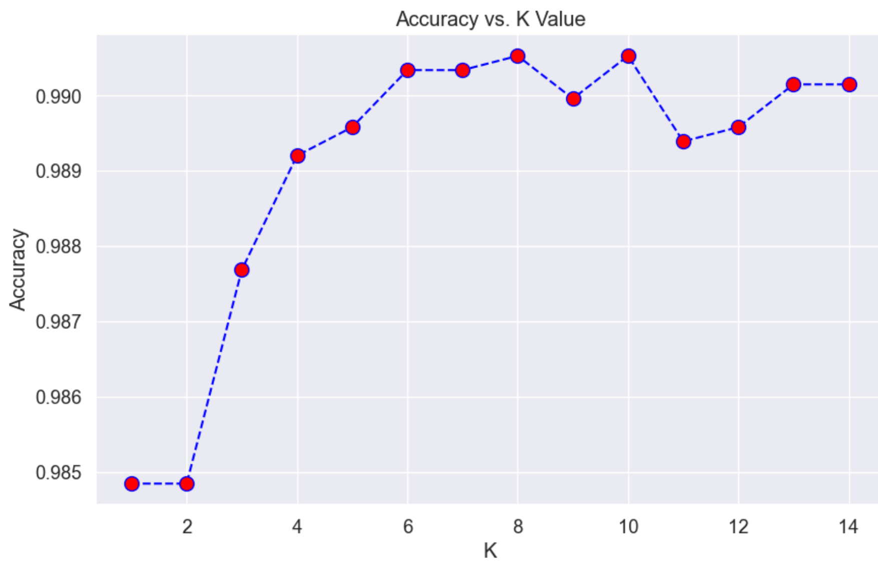


Figure 5.1.2: Graph of Accuracy vs. K Value

5.2 Model Evaluation

5.2.1 Accuracy

Measure	LR	KNN	DT	RF	SVM
Accuracy	0.990340	0.9903409	0.9831439	0.9912879	0.9899621
Mean Cross-validation	0.9890394	0.987902	0.9810018	0.9879027	0.9893642

Table 5.2.1.1: The Accuracy and Mean Cross-validation Scores of Models

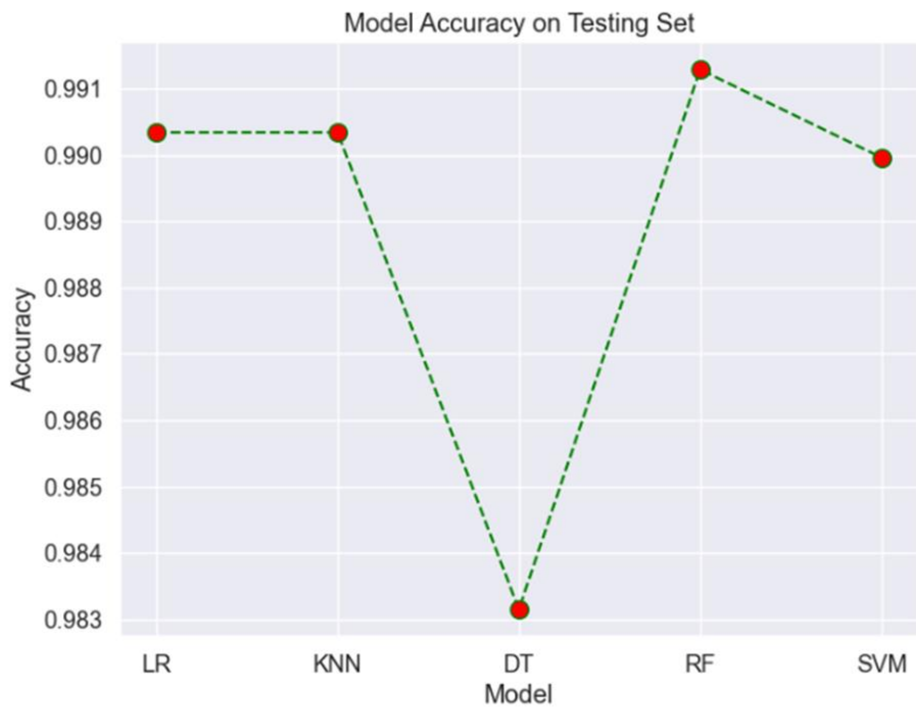


Figure 5.2.1.1: Model Accuracy on Testing Set

Of the five models, four models have an accuracy of 0.99 and Decision Tree has an accuracy of 0.98. These accuracy scores are calculated based on the values from the confusion matrix in Figure 5.2.1.2 below.

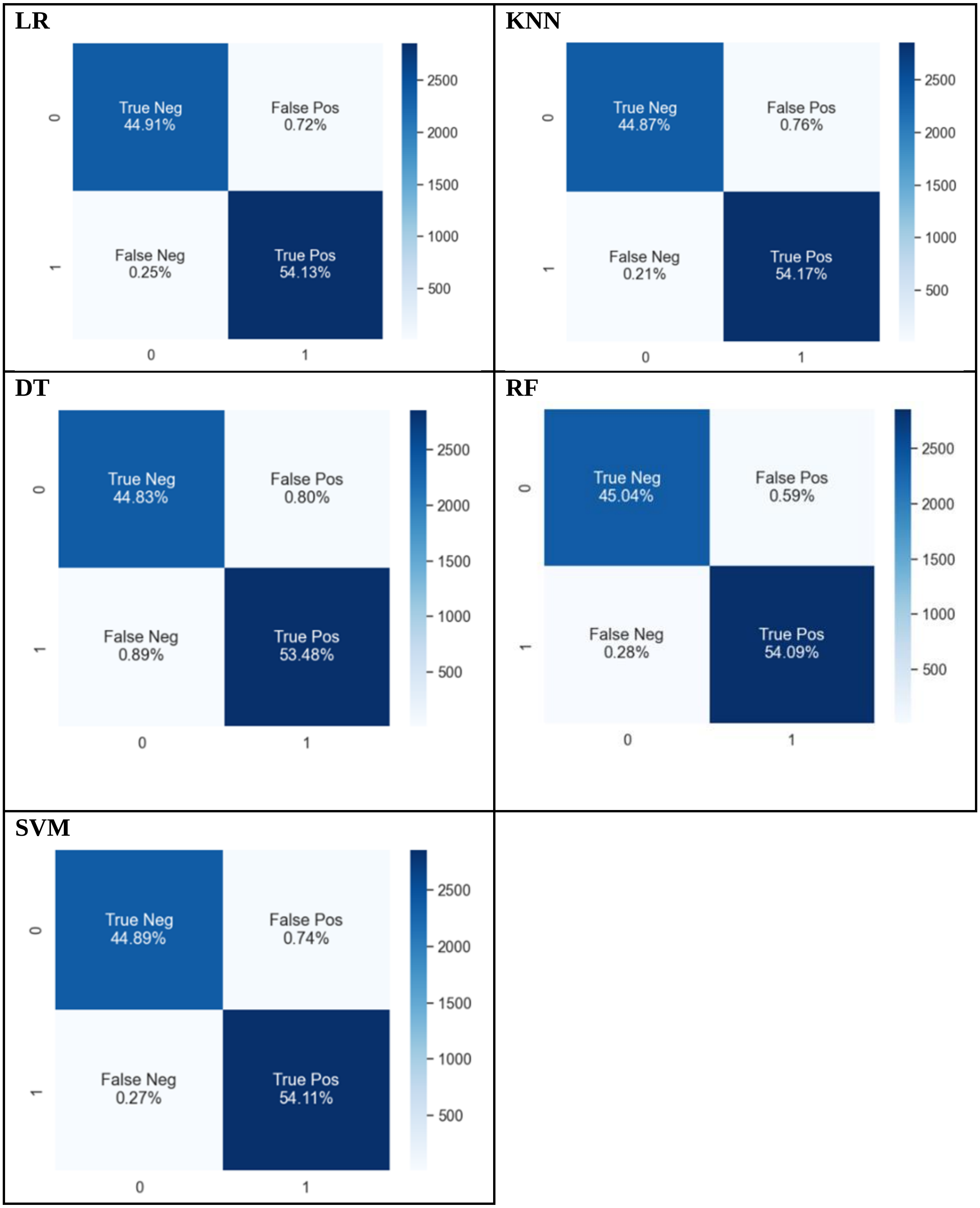


Figure 5.2.1.2: Confusion Matrix of Models

Measure	LR	KNN	DT	RF	SVM

Precision	0.99	0.99	0.98	0.99	0.99
Recall	0.99	0.99	0.98	0.99	0.99
F1 score	0.99	0.99	0.98	0.99	0.99

Table 5.2.1.2: Precision, Recall, and F1-score of Models

The precision, recall and F1 score values of the all the five models are more than 0.90 where four models achieved 0.99 and the Decision Tree model achieved 0.98.

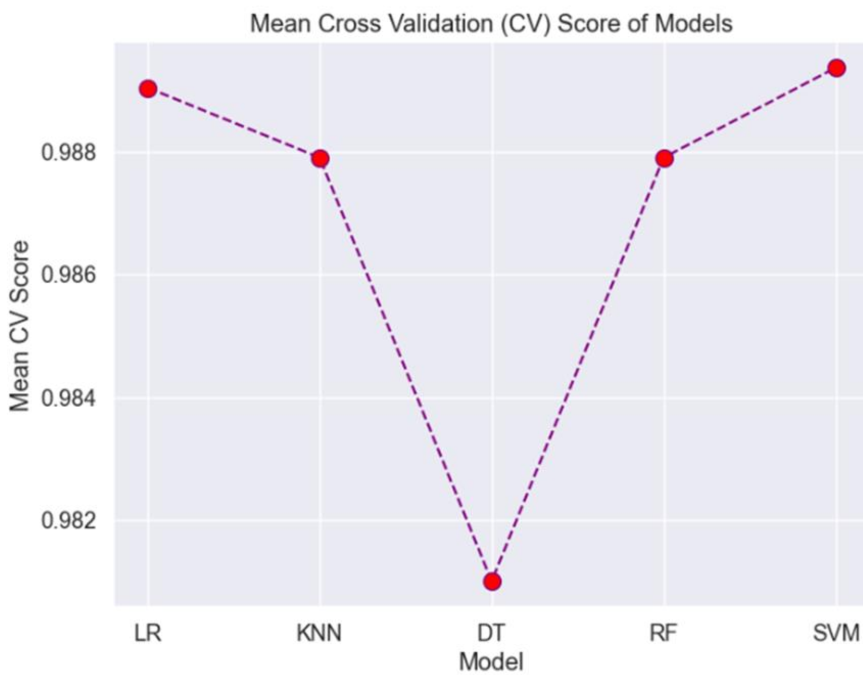


Figure 5.2.1.3: Mean Cross-validation Score of Models

The mean cross-validation scores of all the models are in between 0.98 to 0.99. Logistic Regression, K-Nearest Neighbor, Random Forest, and Support Vector Machine have the mean value of 0.99 and Decision Tree has a mean value of 0.98. Figure 5.2.1.4 below is the mean cross-validation plot of these models.

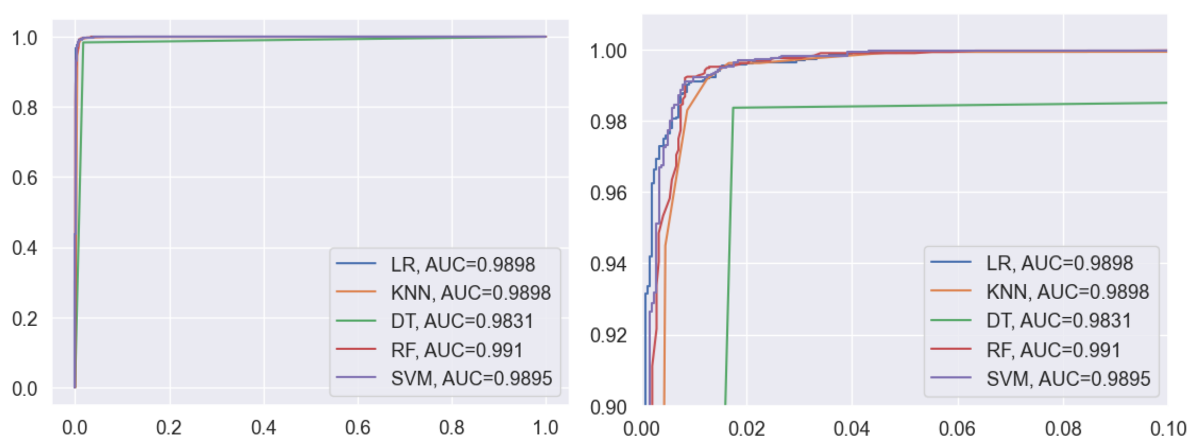


Figure 5.2.1.4: ROC curve of all models (on left) and closer view (on right)

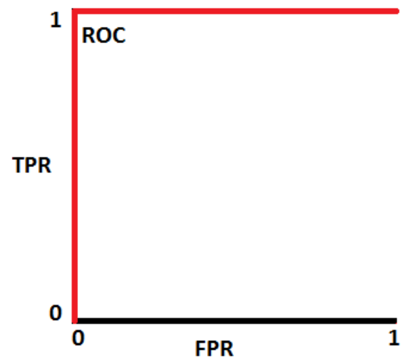


Figure 5.2.1.5: Ideal ROC curve

Figure 5.2.1.5 shows that Random Forest has the highest AUC (0.991) followed by Logistic Regression and K-Nearest Neighbor (0.9898), Support Vector Machine (0.9895), and Decision Tree with the lowest AUC (0.9831). An excellent model has an AUC close to one, indicating that it has a high level of separability. Overall, all the models have exceptionally good AUC.

5.2.6 Chosen Model

Based on all the evaluation metrics used, all five models performed very well. However, the chosen model for this project is Random Forest as it performed slightly better than the rest. As observed on Table 5.2.1.1, the accuracy score of the Random Forest model is 0.9912879 and the mean cross-validation score is 0.9879027. Even though the mean cross-validation score of the Random Forest model is slightly lower than the other models, the area under the curve value is still higher. The AUC of the Random Forest model is 0.991. The precision, recall and F1 score values of this Random Forest model is also higher, 0.99. Random Forest is advantageous because it has great default hyperparameters, can handle large datasets, and does not overfit the data. A major disadvantage of Random Forest is that many trees can make the algorithm too slow and ineffective for real-time predictions, which was not an issue faced during our implementation.

6 Conclusion and future work

Through this study, Gonen and Jasmine rice species are classified based on their unique characteristics using the best classification model identified, i.e., the Random Forest model with 0.99 accuracy and mean cross-validation score. With such outstanding performance, it is believed that the model can perform well in classifying these two rice types quickly and accurately without human intervention.

In future work, we aim to extend our study by collecting more diverse features and making feature inferences for other rice types, especially morphological features for further improvement on our rice type classification model. Apart from that, research will be performed on the rice image processing to achieve an automatic rice classification system, instead of relying on manual data collection of rice features. The rice image classification model will serve as the foundation of the mobile application designed named Ricense. It will integrate with other functional features such as the rice production calculator and rice harvesting progress tracker using sensors to achieve a smart farming solution for rice producers to improve their rice cultivation productivity and efficiency. The goal of the application is to identify different rice types by simply capturing the picture of the rice, provide information on the rice species, and suggestions on how to achieve better rice yield. Besides, the rice production calculator allows farmers to estimate their rice yield, as well as their return of investments using the data recorded by sensors. The real-time figures will be employed in tracking and monitoring of rice harvesting progress to alert farmers and offer actionable insights for immediate decision-making, thus improving the producer's potential yield. Certainly, more studies and effort are required to achieve the objectives of this one-stop smart farming mobile application.

7 Reference

Arora, B., Bhagat, N., Saritha, L. R., & Arcot, S. (2020). Rice grain classification using Image Processing & Machine Learning Techniques. *2020 International Conference on Inventive Computation Technologies (ICICT)*. https://doi.org/10.1109/icict48043.2020.9112418

Ghiasi, A., Ng, C.-T., & Sheikh, A. H. (2022). Damage detection of in-service steel railway bridges using a fine k-nearest neighbor machine learning classifier. *Structures*, *45*, 1920–1935. https://doi.org/10.1016/j.istruc.2022.10.019

Gurrala, K. R. (2021). Application of Big Data Tools for Seed Classification. *International Research Journal of Engineering and Technology (IRJET)*, *8*(1), 998. https://www.irjet.net/archives/V8/i1/IRJET-V8I1182.pdf

Jackins, V., Vimal, S., Kaliappan, M., & Lee, M. Y. (2020). AI-based smart prediction of clinical disease using random forest classifier and naive bayes. *The Journal of Supercomputing*, *77*(5), 5198–5219. https://doi.org/10.1007/s11227-020-03481-x

Jijo, B. T., & Abdulazeez, A. (2021). Classification based on Decision Tree Algorithm for Machine Learning. *Journal of Applied Science and Technology Trends*, *2*(01), 20–28. https://doi.org/10.38094/jastt20165

Kader, N. I. A., Yusof, U. K., & Naim, S. (2019). *Diabetic Retinopathy Classification Using Support Vector Machine with Hyperparameter Optimization*.

Maione, C., & Barbosa, R. M. (2019). Recent applications of multivariate data analysis methods in the authentication of rice and the most analyzed parameters: A review. *Critical Reviews in Food Science and Nutrition*, *59*(12), 1868–1879. https://doi.org/10.1080/10408398.2018.1431763

Major rice exporting countries worldwide 2022/2023 | Statista. (2023, February 16). Statista. https://www.statista.com/statistics/255947/top-rice-exporting-countries-worldwide-2011/

Rahman, A. N. M. R. B., & Zhang, J. (2022). Trends in rice research: 2030 and beyond. *Food and Energy Security*, *12*(2). https://doi.org/10.1002/fes3.390

Raschka, S., & Mirjalili, V. (2019). *Python machine learning: machine learning and deep learning with python, scikit-learn, and tensorflow 2*. Packt Publishing, Limited.

Saito, T., & Rehmsmeier, M. (2015). The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*, *10*(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

Subasi, A. (2020). Machine learning techniques. *Practical Machine Learning for Data Analysis Using Python*, 91–202. https://doi.org/10.1016/b978-0-12-821379-7.00003-5

Thornton, C. (2023, April 19). *Global rice shortage possible in 2023, prices are expected to remain high, analysts say*. USA Today. Retrieved May 2, 2023, from https://www.usatoday.com/story/money/food/2023/04/19/rice-shortage-2023-worldwide-outlook/11697581002/

Variety selection. (11 B.C.E.). Rice Knowledge Bank. Retrieved May 2, 2023, from http://www.knowledgebank.irri.org/training/fact-sheets/crop-establishment/item/variety-selection-fact-sheet

8 Appendix

Jupyter Notebook link: https://github.com/cbaogang/RICE-TYPE-CLASSIFICATION.git

File name: WQD7006 GROUP 4.ipynb

Rice Type Classification

Members:

ML4DS G1(RL) – Group 4		
No.	**MATRIC ID**	**NAME**
1	22056764	Devayani A/P Balkrishnan
2	S2195613	Li Xin Qi
3	S2155659	Lim Yu Xuan
4	S2192763	Navaneeta A/P P Shanmugam
5	17186722	Chen Bao Gang

Content

Import libraries

In [62]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Importing the dataset

In [63]:
```
df = pd.read_csv('riceClassification.csv')
```

1.0 Dataset Description

In [64]:
```
df.head()
```

Out[64]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	EquivDiameter	Extent	Perimeter	Roundness	AspectRation	Class
0	1	4537	92.229316	64.012769	0.719916	4677	76.004525	0.657536	273.085	0.764510	1.440796	1
1	2	2872	74.691881	51.400454	0.725553	3015	60.471018	0.713009	208.317	0.831658	1.453137	1
2	3	3048	76.293164	52.043491	0.731211	3132	62.296341	0.759153	210.012	0.868434	1.465950	1
3	4	3073	77.033628	51.928487	0.738639	3157	62.551300	0.783529	210.657	0.870203	1.483456	1
4	5	3693	85.124785	56.374021	0.749282	3802	68.571668	0.769375	230.332	0.874743	1.510000	1

In [65]:
```
print('Total number of rows are:', df.shape[0])
print('Total number of columns are:', df.shape[1])
```

```
Total number of rows are: 18185
Total number of columns are: 12
```

In [66]:
```
columns=df.columns.to_list()
print(columns)
```

```
['id', 'Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'ConvexArea', 'EquivDiameter', 'Extent', 'Perimeter', 'Rou
ndness', 'AspectRation', 'Class']
```

All attributes are numeric variables and they are listed as below:

- id
- Area
- MajorAxisLength
- MinorAxisLength
- Eccentricity
- ConvexArea
- EquivDiameter
- Extent
- Perimeter
- Roundness
- AspectRation
- Class

In [67]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18185 entries, 0 to 18184
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               18185 non-null  int64
 1   Area             18185 non-null  int64
 2   MajorAxisLength  18185 non-null  float64
 3   MinorAxisLength  18185 non-null  float64
 4   Eccentricity     18185 non-null  float64
 5   ConvexArea       18185 non-null  int64
 6   EquivDiameter    18185 non-null  float64
 7   Extent           18185 non-null  float64
 8   Perimeter        18185 non-null  float64
 9   Roundness        18185 non-null  float64
 10  AspectRation     18185 non-null  float64
 11  Class            18185 non-null  int64
dtypes: float64(8), int64(4)
memory usage: 1.7 MB
```

It is clear from the above output that there are no categorical columns present in the dataset.

In [68]:
```
df.isna().sum(axis=0)
```

Out[68]:
```
id                 0
Area               0
MajorAxisLength    0
MinorAxisLength    0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Perimeter          0
Roundness          0
AspectRation       0
Class              0
dtype: int64
```

The number of missing values are counted in each column. Based on the above output, there are no missing values in the columns.

In [69]:
```
df.describe()
```

Out[69]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	EquivDiameter	Extent	Perimeter	Roundness
count	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000
mean	9093.000000	7036.492989	151.680754	59.807851	0.915406	7225.817872	94.132952	0.616653	351.606949	0.707998
std	5249.701658	1467.197150	12.376402	10.061653	0.030575	1502.006571	9.906250	0.104389	29.500620	0.067310
min	1.000000	2522.000000	74.133114	34.409894	0.676647	2579.000000	56.666658	0.383239	197.015000	0.174590
25%	4547.000000	5962.000000	145.675910	51.393151	0.891617	6125.000000	87.126656	0.538530	333.990000	0.650962
50%	9093.000000	6660.000000	153.883750	55.724288	0.923259	6843.000000	92.085696	0.601194	353.088000	0.701941
75%	13639.000000	8423.000000	160.056214	70.156593	0.941372	8645.000000	103.559146	0.695664	373.003000	0.769280
max	18185.000000	10210.000000	183.211434	82.550762	0.966774	11008.000000	114.016559	0.886573	508.511000	0.904748

INFERENCE

- The dataset contains 10 features (Area to AspectRation) and 1 label (Class).
- There are 18,185 entries of data which is quite ideal for building Machine Learning model.
- There are no categorical columns present in the dataset. Most of the features are of float64 datatype.
- There are no missing value in any features or data i.e. the dataset is quite clean.

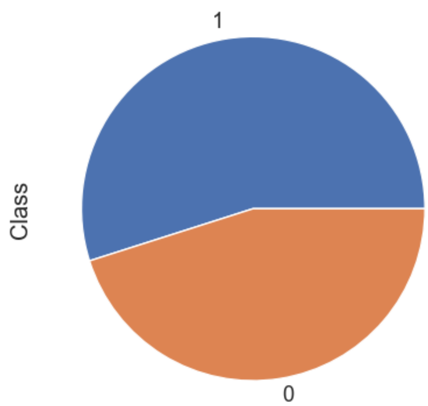
2.0 Data Preprocessing and EDA

Drop id column

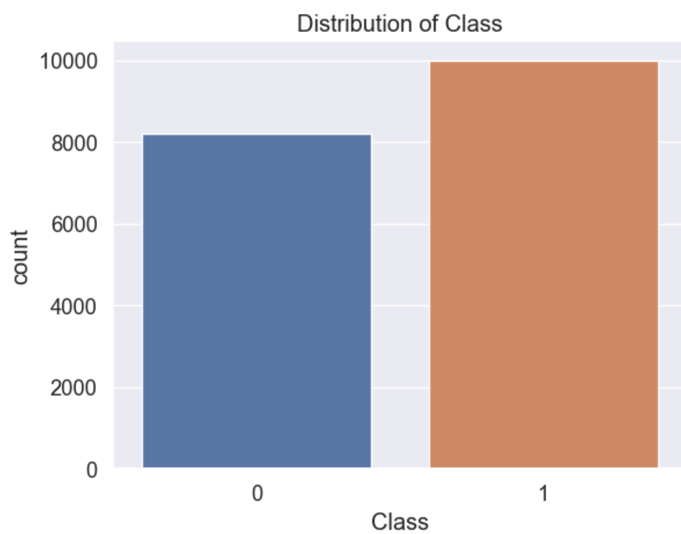
In [72]:
```
df = df.drop(columns = 'id', axis = 1)
```

The Distribution of the Class Column

In [70]:
```
# Checking whether dataset is balanced or not
piechart = df['Class'].value_counts().plot(kind='pie')
fig1 = piechart.get_figure()
```

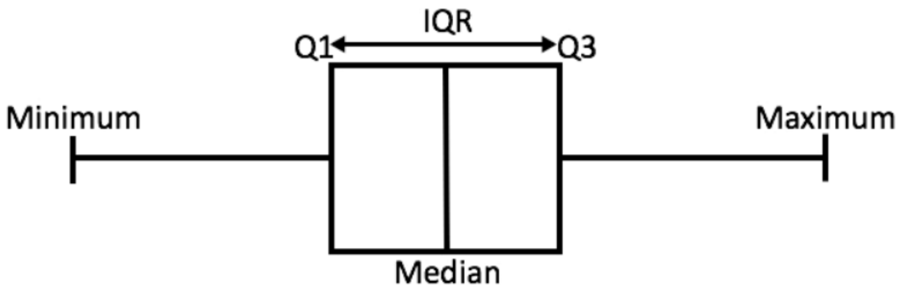


In [71]:
```
import seaborn as sns
# Check the distribution of Class
sns.countplot(x='Class', data=df)
plt.title('Distribution of Class')
plt.show()
```



From the above bar chart, the class output are relatively well-balanced and no sampling is required.

Inter Quartile Range to detect outliers



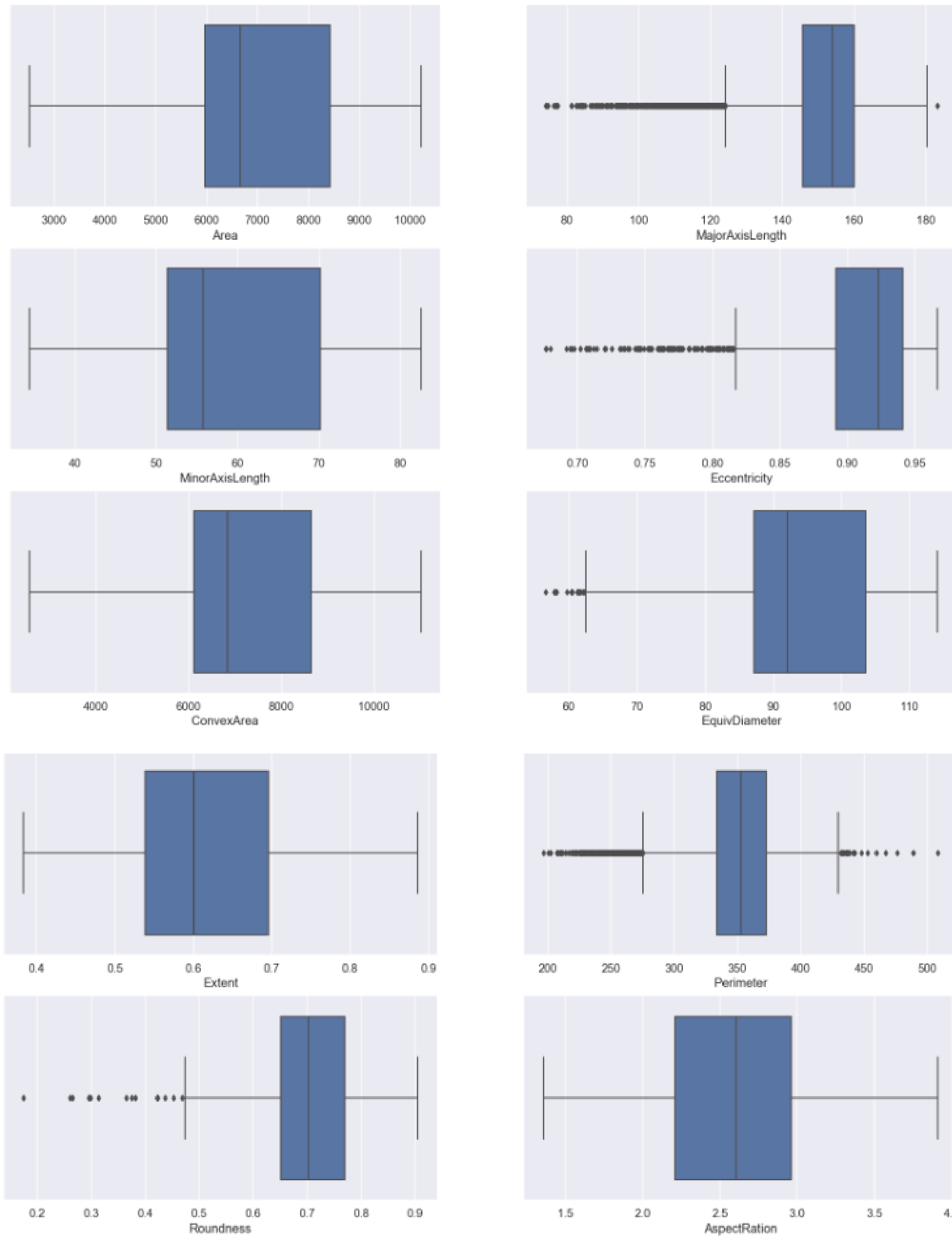
- Minimum indicates the minimum value in the dataset and maximum is the maximum value in the dataset. So the difference between the two tells us about the range of dataset.
- The median is the median (or centre point), also called second quartile of the data (resulting from the fact that the data is in an ordered manner).
- Q1 is the first quartile of the data, stating that 25% of the data lies between minimum and Q1.
- Q3 is the third quartile of the data, stating that 75% of the data lies between minimum and Q3.
- The difference between Q3 and Q1 is called the Inter-Quartile Range or IQR.

Box Plot of each feature before Outlier Detection

In [78]:
```
fig, axes = plt.subplots(5,2, figsize=(20,25))
fig.suptitle('Box Plot Before Outlier Detection')
k = 0
for i in range(0,5):
    for j in range(0,2):
        sns.boxplot(ax=axes[i,j], data=df, x=df.columns[k])
        k=k+1
```

Box Plot Before Outlier Detection

Box Plot Before Outlier Detection



Removing the outliers

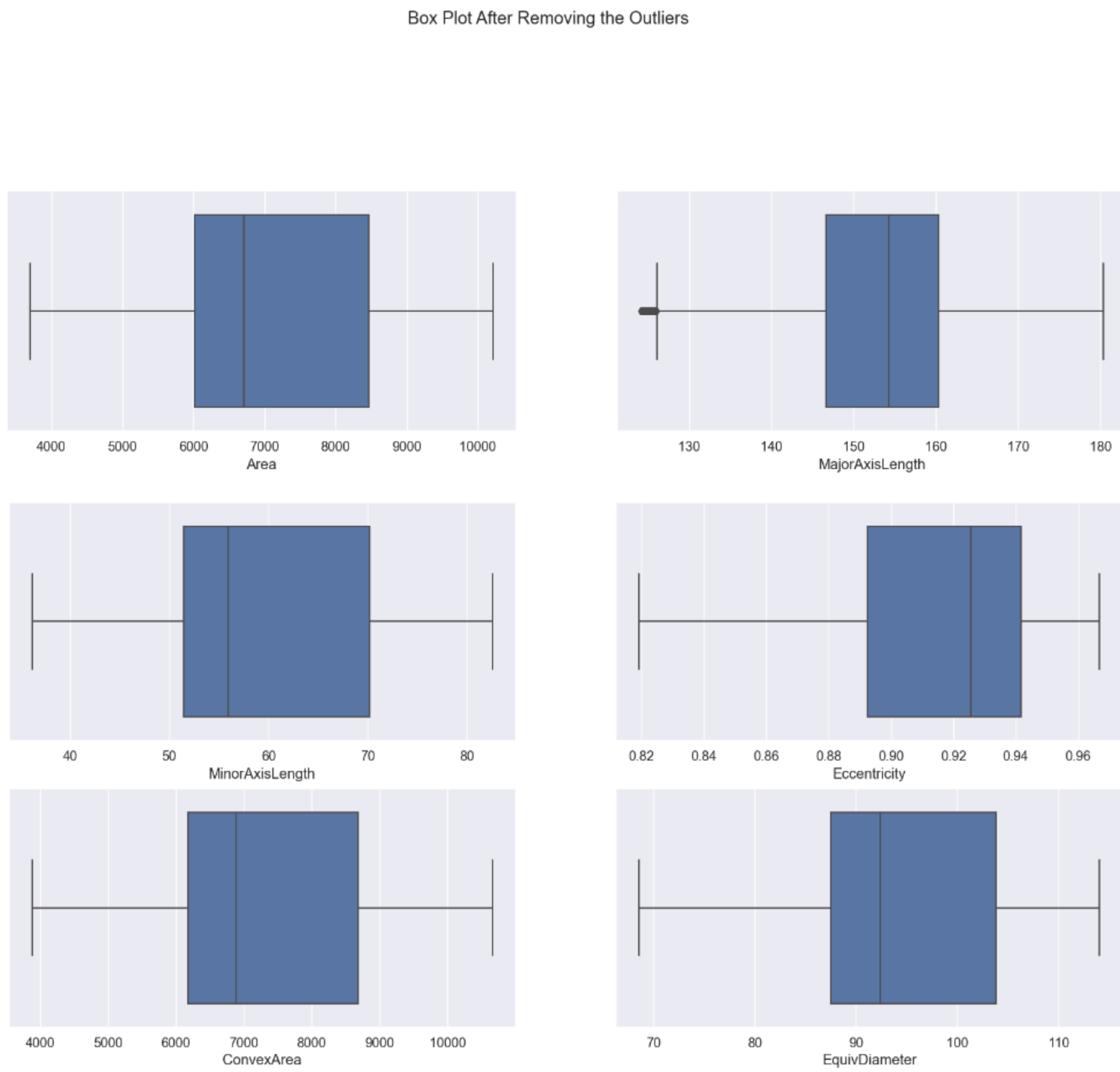
In [79]:
```
"""
    Calculates the Interquartile Range (IQR) for a given column in df
    Returns the upper and lower bounds of the IQR.
"""
def iqr_calculation(col):
    Q1 = np.percentile(col, 25)
    Q3 = np.percentile(col, 75)
    IQR = Q3 - Q1
    upper = np.where(col >= (Q3 + 1.5 * IQR))[0]
    lower = np.where(col <= (Q1 - 1.5 * IQR))[0]
    return upper, lower

def remove_outliers(df, col_name):
    upper, lower = iqr_calculation(df[col_name])
    if len(upper) == 0 and len(lower) == 0:
        print("No outliers are removed")
        return df
    else:
        df = df.drop(upper).drop(lower)
        print("New Shape: ", df.shape)
        return df.reset_index(drop=True)
```

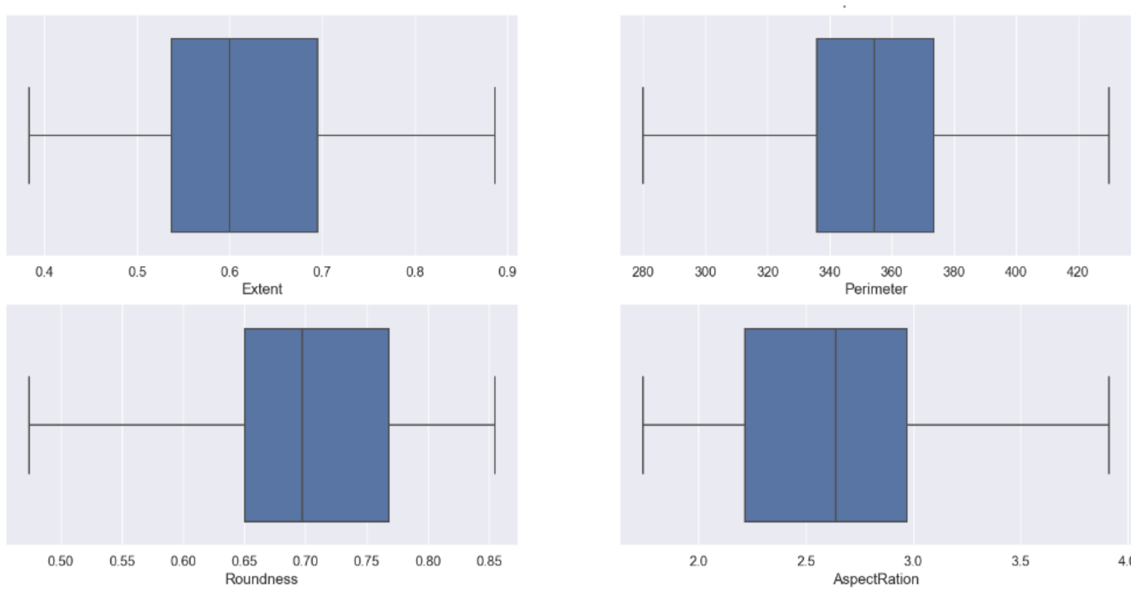
In [80]:
```
main_df = df.copy()

for col in main_df.columns[:-1]:
    print('For', col)
    main_df = remove_outliers(main_df, col)

df = main_df
```

```
For Area
No outliers are removed
For MajorAxisLength
New Shape:  (17647, 11)
For MinorAxisLength
No outliers are removed
For Eccentricity
New Shape:  (17631, 11)
For ConvexArea
No outliers are removed
For EquivDiameter
No outliers are removed
For Extent
No outliers are removed
For Perimeter
New Shape:  (17602, 11)
For Roundness
New Shape:  (17597, 11)
For AspectRation
No outliers are removed
```

Box Plot of each feature after Outlier Detection

In [81]:
```
fig, axes = plt.subplots(5,2, figsize=(20,25))
fig.suptitle('Box Plot After Removing the Outliers')
k = 0
for i in range(0,5):
    for j in range(0,2):
        col_name = df.columns[k]
        sns.boxplot(ax=axes[i,j], data=main_df,x=col_name)
        k=k+1
```





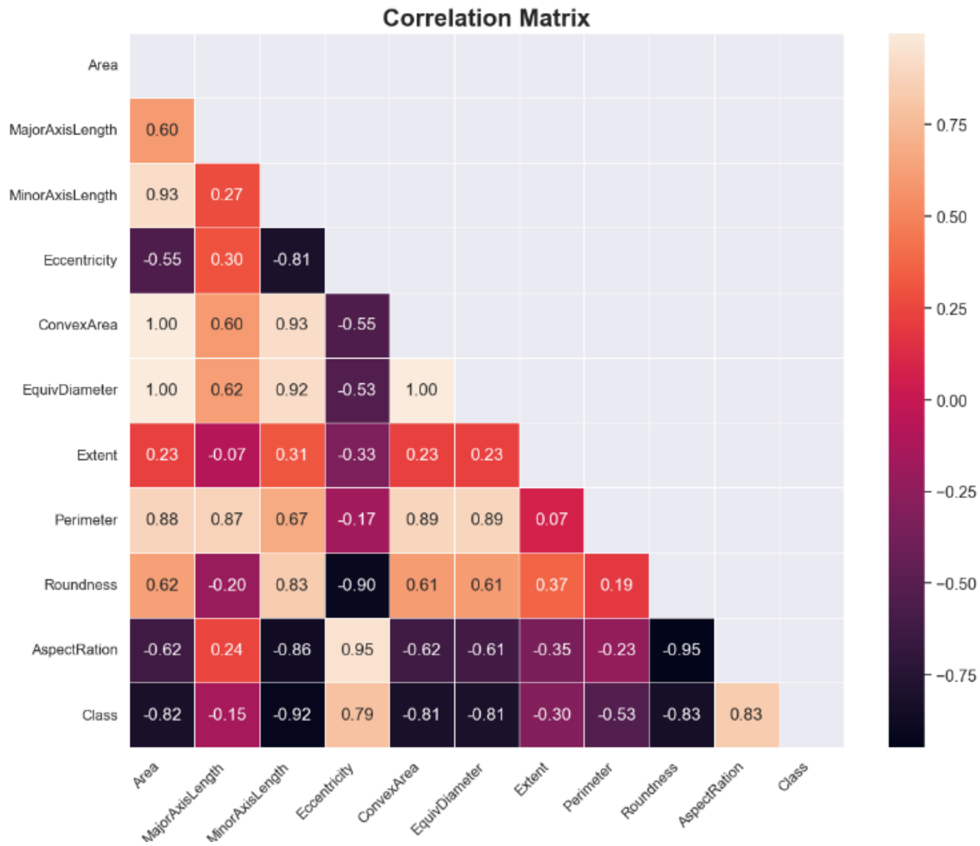
In [82]:
```
df.shape
```
Out[82]: (17597, 11)

Building correlation matrix

In [73]:
```
correlation_mat=df.corr()
correlation_mat
```
Out[73]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	EquivDiameter	Extent	Perimeter	Roundness	AspectRation	C
Area	1.000000	0.599939	0.930215	-0.550073	0.999362	0.998158	0.230541	0.881540	0.620490	-0.623979	-0.81
MajorAxisLength	0.599939	1.000000	0.273211	0.295717	0.602061	0.618002	-0.073549	0.870178	-0.202566	0.240471	-0.14
MinorAxisLength	0.930215	0.273211	1.000000	-0.808640	0.928992	0.923790	0.308541	0.674249	0.834398	-0.860516	-0.91
Eccentricity	-0.550073	0.295717	-0.808640	1.000000	-0.547896	-0.534688	-0.329954	-0.165915	-0.903657	0.950301	0.78
ConvexArea	0.999362	0.602061	0.928992	-0.547896	1.000000	0.997403	0.227359	0.886987	0.610236	-0.621472	-0.81
EquivDiameter	0.998158	0.618002	0.923790	-0.534688	0.997403	1.000000	0.225944	0.891567	0.607432	-0.609957	-0.80
Extent	0.230541	-0.073549	0.308541	-0.329954	0.227359	0.225944	1.000000	0.073227	0.366793	-0.350875	-0.30
Perimeter	0.881540	0.870178	0.674249	-0.165915	0.886987	0.891567	0.073227	1.000000	0.186063	-0.227256	-0.53
Roundness	0.620490	-0.202566	0.834398	-0.903657	0.610236	0.607432	0.366793	0.186063	1.000000	-0.947875	-0.83
AspectRation	-0.623979	0.240471	-0.860516	0.950301	-0.621472	-0.609957	-0.350875	-0.227256	-0.947875	1.000000	0.83
Class	-0.816589	-0.147741	-0.917766	0.788636	-0.814214	-0.809361	-0.303440	-0.533274	-0.831759	0.832563	1.00

In [74]:
```
corr_features = correlation_mat.index

plt.figure(figsize=(12,10))
sns.set(font_scale=1.2)

# Set mask to display only the lower triangular part of the heatmap
mask = np.zeros_like(df[corr_features].corr())
mask[np.triu_indices_from(mask)] = True

g = sns.heatmap(df[corr_features].corr(), annot=True, fmt='.2f',
                linewidths=.5, annot_kws={"size": 14}, mask=mask)

g.set_xticklabels(g.get_xticklabels(), rotation=45, ha='right', fontsize=12)
g.set_yticklabels(g.get_yticklabels(), rotation=0, ha='right', fontsize=12)
plt.title('Correlation Matrix', fontsize=20, fontweight='bold')
plt.tight_layout()
plt.show()
```



INFERENCE:

The highest negative correlations are between:

- feature-to-feature - Roundness and AspectRatio (-0.95), Roundness and Eccentricity (-0.90), MinorAxisLength and AxisRatio (-0.86), MinorAxisLength and Eccentricity (-0.81).
- feature-to-class - MinorAxisLength (-0.92), Roundness (-0.83), Area (-0.82), ConvexArea (-0.81), EquivDiameter (-0.81).

The highest positive correlations are between:

- feature-to-feature - AspectRatio and Eccentricity (0.95), Area and MinorAxisLength (0.93), MinorAxisLength and ConvexArea (0.93), MinorAxisLength and EquivDiameter (0.92), ConvexArea and Perimeter (0.89), EquivDiameter and Perimeter (0.89), Area and Perimeter (0.88), MajorAxisLength and Perimeter (0.87), MinorAxisLength and Roundness (0.83).
- feature-to-class - AspectRatio (0.83), Eccentricity (0.79).

We can see that MinorAxisLength, AspectRatio, Roundness, Area, ConvexArea, EquivDiameter and Eccentricity have a very high correlation with the target variable.

Check Multicollinearity by using VIF

Multicollinearity can be calculated using Variable Inflation Factors (VIF). Unlike Correlation matrix, VIF determines the strength of the correlation of a variable with several other independent variables in a dataset. VIF usually starts at 1 and anywhere exceeding 10 indicates high multicollinearity between the independent variables.

In [75]:
```
def VIF(dataframe,chosen_cols):
    from statsmodels.stats.outliers_influence import variance_inflation_factor as VIF
    from statsmodels.tools.tools import add_constant
    X=dataframe[chosen_cols]
    X=add_constant(X)
    vif_data=pd.DataFrame()
    vif_data["feature"]=X.columns
    vif_data["VIF"]=[VIF(X.values, i) for i in range(len(X.columns))]
    return vif_data
```

```
In [76]: chosen_cols=['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'ConvexArea', 'EquivDiameter', 'Extent', 'Perimeter'

         VIF(df,chosen_cols)
```

Out[76]:

	feature	VIF
0	const	103954.790226
1	Area	2347.459870
2	MajorAxisLength	318.714833
3	MinorAxisLength	1296.490608
4	Eccentricity	51.971818
5	ConvexArea	1813.854237
6	EquivDiameter	2844.995049
7	Extent	1.157557
8	Perimeter	346.462939
9	Roundness	157.668241
10	AspectRation	118.998499

In the context of calculating VIF, a VIF value of 10 or higher is often considered significant multicollinearity between the independent variables. From the result, it shows that there are some features with extremely high VIF values. Let's try to drop some of the correlated features to see if it helps us in bringing down the multicollinearity between correlated features.

```
In [77]: new_chosen_cols=["MajorAxisLength","Roundness","Eccentricity","Extent"]
         VIF(df,new_chosen_cols)
```

Out[77]:

	feature	VIF
0	const	8707.834300
1	MajorAxisLength	1.123905
2	Roundness	5.753991
3	Eccentricity	5.876661
4	Extent	1.155467

VIF values for all the independent variables have decreased to a reasonable extent.

3.0 Splitting the Dataset

```
In [83]: X=df[["MajorAxisLength","Roundness","Eccentricity","Extent"]]
         Y=df["Class"]

         X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3,random_state=42)
```

Standardization

```
In [84]: scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_train

Out[84]: array([[-0.22947603, -0.79852573,  0.73387877,  0.50650271],
                [ 0.37240698,  0.76111001, -0.96377629, -0.11797718],
                [ 0.59292898, -1.66834014,  1.56394508, -1.88201494],
                ...,
                [ 1.08939692, -0.82063083,  0.91627689, -1.31857217],
                [-0.10765373, -0.9658156 ,  0.32824037, -1.03881585],
                [ 0.40863635,  0.80903888, -0.66096941, -0.4926156 ]])
```

```
In [85]: X_test = scaler.transform(X_test)
         X_test
```

```
Out[85]: array([[-1.17932721, -0.29294766,  0.44123002, -0.43392902],
                [ 1.35059936, -0.21465292,  0.65135606, -1.26869849],
                [ 1.0511613 ,  1.14707079, -1.48612111, -0.17214309],
                ...,
                [-1.29257639,  0.76827926, -1.07253233,  0.12099944],
                [ 0.06774699,  0.56091168, -0.43711659,  1.4088493 ],
                [ 0.5792953 , -0.98653376,  1.12399191,  1.10935002]])
```

4.0 Modelling

```
In [86]: from sklearn.linear_model import LogisticRegression
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.svm import SVC

         from sklearn import metrics
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import classification_report
```

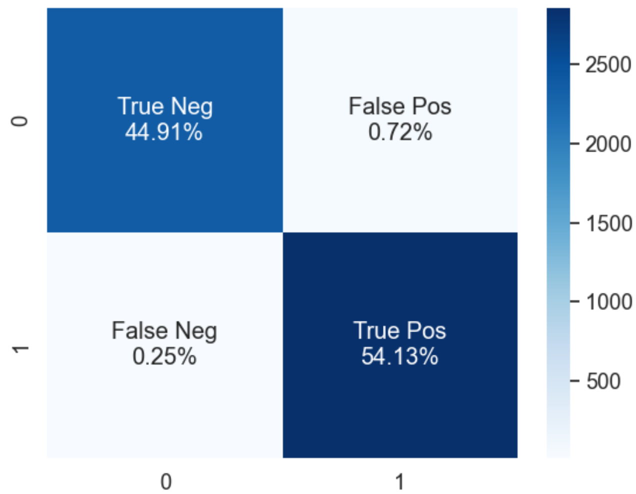
Logistic Regression

```
In [87]: logreg = LogisticRegression()
         log_model = logreg.fit(X_train, y_train)
```

```
In [88]: y_pred_log = logreg.predict(X_test)
         y_proba_log = log_model.predict_proba(X_test)[:, 1]
         acc_log = logreg.score(X_test, y_test)
         print('Accuracy of Logistic Regression classifier on test set: {:.7f}'.format(acc_log))

         Accuracy of Logistic Regression classifier on test set: 0.9903409
```

```
In [89]: cm_log = confusion_matrix(y_test, y_pred_log)
         print(cm_log)

         [[2371   38]
          [  13 2858]]
```

```
In [90]: plt.figure(figsize=(6.5,5))

         group_names = ['True Neg','False Pos','False Neg','True Pos']
         group_percentages = ['{0:.2%}'.format(value) for value in cm_log.flatten()/np.sum(cm_log)]
         labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
         labels = np.asarray(labels).reshape(2,2)
         sns.heatmap(cm_log, annot=labels, fmt='', cmap='Blues')

Out[90]: <AxesSubplot:>
```



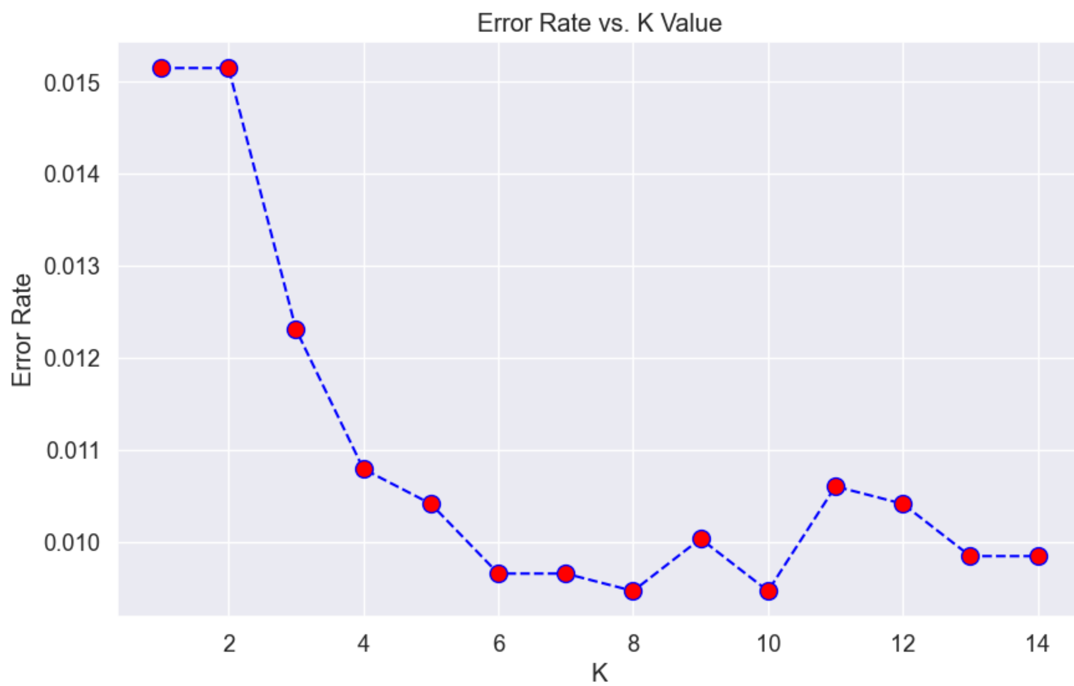
In [91]:
```
print(classification_report(y_test, y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      2409
           1       0.99      1.00      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

K-NN

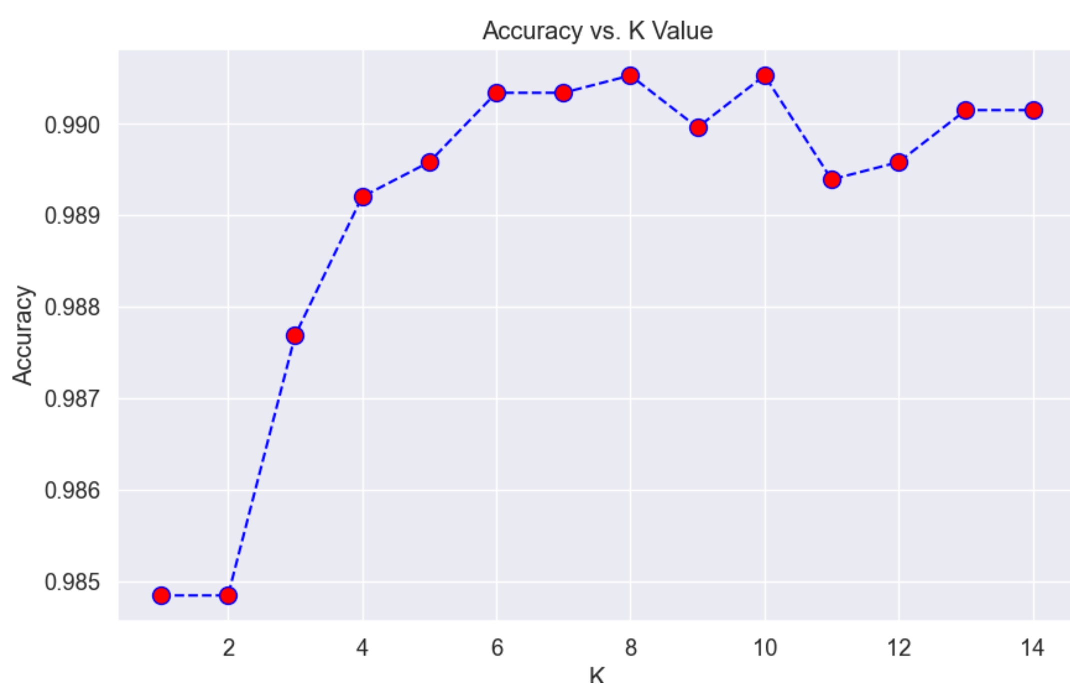
In [92]:
```
error_rate = []
for i in range(1,15):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,15),error_rate,color='blue', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))
```

```
Minimum error:- 0.009469696969696969697 at K = 7
```



In [93]:
```
acc = []
for i in range(1,15):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,15),acc,color = 'blue',linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))
```

```
Maximum accuracy:- 0.990530303030303 at K = 7
```



In [94]:

```
knn = KNeighborsClassifier(n_neighbors=7)
knn_model = knn.fit(X_train, y_train)
```

In [95]:

```
y_pred_knn = knn.predict(X_test)
y_proba_knn = knn_model.predict_proba(X_test)[:, 1]
acc_knn = knn.score(X_test, y_test)
print('Accuracy of K-NN classifier on test set: {:.7f}'.format(acc_knn))
```

```
Accuracy of K-NN classifier on test set: 0.9903409
```

In [96]:

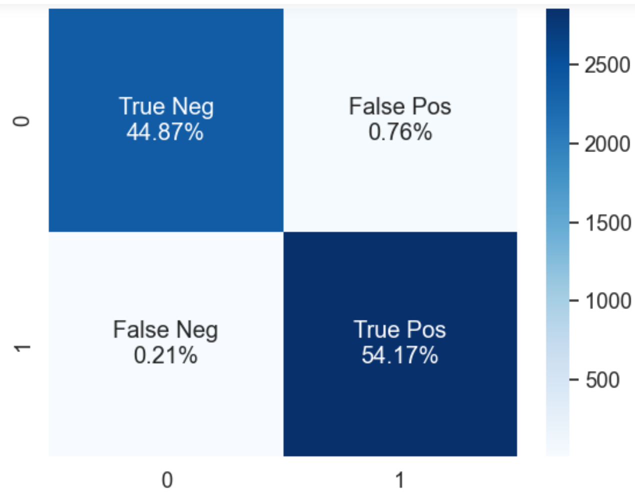
```
cm_knn = confusion_matrix(y_test, y_pred_knn)
print(cm_knn)
```

```
[[2369   40]
 [  11 2860]]
```

In [97]:

```
plt.figure(figsize=(6.5,5))

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in cm_knn.flatten()/np.sum(cm_knn)]
labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm_log, annot=labels, fmt='', cmap='Blues')
```

Out[97]: <AxesSubplot:>



In [98]:

```
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2409
           1       0.99      1.00      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

Decision Tree

In [99]:
```
dt = DecisionTreeClassifier()
dt_model = dt.fit(X_train, y_train)
```

In [100]:
```
y_pred_dt = dt.predict(X_test)
y_proba_dt = dt_model.predict_proba(X_test)[:, 1]
acc_dt = dt.score(X_test, y_test)
print('Accuracy of Decision Tree classifier on test set: {:.7f}'.format(acc_dt))
```

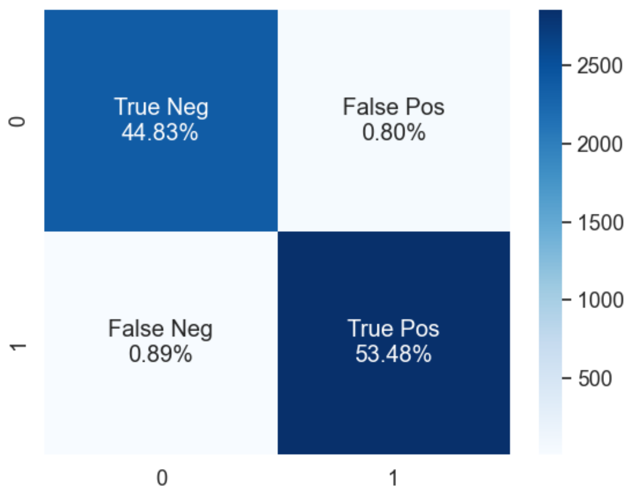
```
Accuracy of Decision Tree classifier on test set: 0.9831439
```

In [101]:
```
cm_dt = confusion_matrix(y_test, y_pred_dt)
print(cm_dt)
```

```
[[2367   42]
 [  47 2824]]
```

In [102]:
```
plt.figure(figsize=(6.5,5))

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in cm_dt.flatten()/np.sum(cm_dt)]
labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm_log, annot=labels, fmt='', cmap='Blues')
```

Out[102]: <AxesSubplot:>



In [103]:
```
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2409
           1       0.99      0.98      0.98      2871

    accuracy                           0.98      5280
   macro avg       0.98      0.98      0.98      5280
weighted avg       0.98      0.98      0.98      5280
```

Random Forest

In [104]:
```
rf = RandomForestClassifier()
rf_model = rf.fit(X_train,y_train)
```

In [105]:
```
y_pred_rf = rf.predict(X_test)
y_proba_rf = rf_model.predict_proba(X_test)[:, 1]
acc_rf = rf.score(X_test, y_test)
print('Accuracy of Random Forest classifier on test set: {:.7f}'.format(acc_rf))
```

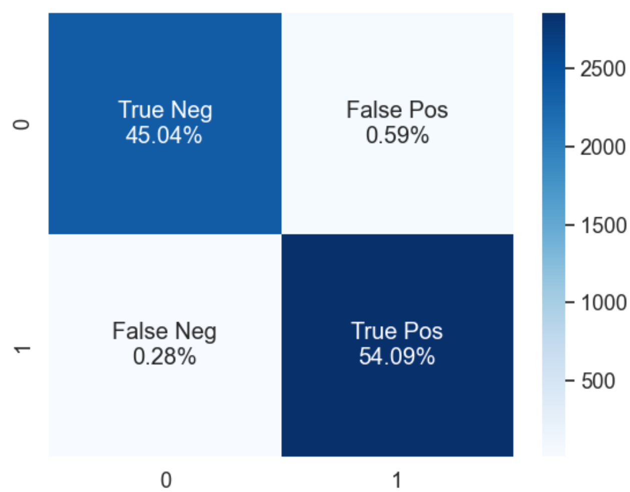
```
Accuracy of Random Forest classifier on test set: 0.9912879
```

In [106]:
```
cm_rf = confusion_matrix(y_test, y_pred_rf)
print(cm_rf)
```

```
[[2378   31]
 [  15 2856]]
```

In [107]:
```
plt.figure(figsize=(6.5,5))

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in cm_rf.flatten()/np.sum(cm_rf)]
labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm_log, annot=labels, fmt='', cmap='Blues')
```

Out[107]: <AxesSubplot:>



In [108]:
```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2409
           1       0.99      0.99      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

Support Vector Machine

In [109]:
```
svc = SVC(probability=True)
svc_model = svc.fit(X_train,y_train)
```

In [110]:
```
y_pred_svc = svc.predict(X_test)
y_proba_svc = svc_model.predict_proba(X_test)[:, 1]
acc_svc = svc.score(X_test, y_test)
print('Accuracy of Support Vector classifier on test set: {:.7f}'.format(acc_svc))
```

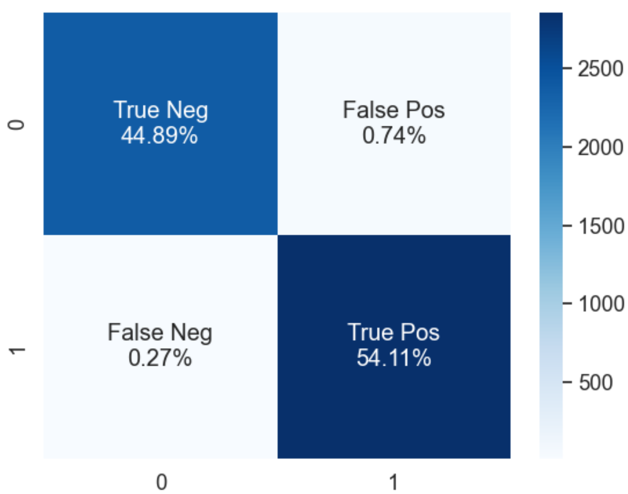
```
Accuracy of Support Vector classifier on test set: 0.9899621
```

In [111]:
```
cm_svc = confusion_matrix(y_test, y_pred_svc)
print(cm_svc)
```

```
[[2370   39]
 [  14 2857]]
```

In [112]:
```
plt.figure(figsize=(6.5,5))

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in cm_svc.flatten()/np.sum(cm_svc)]
labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm_log, annot=labels, fmt='', cmap='Blues')
```

Out[112]: <AxesSubplot:>



In [113]:
```
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      2409
           1       0.99      1.00      0.99      2871

    accuracy                           0.99      5280
   macro avg       0.99      0.99      0.99      5280
weighted avg       0.99      0.99      0.99      5280
```

5.0 Evaluation

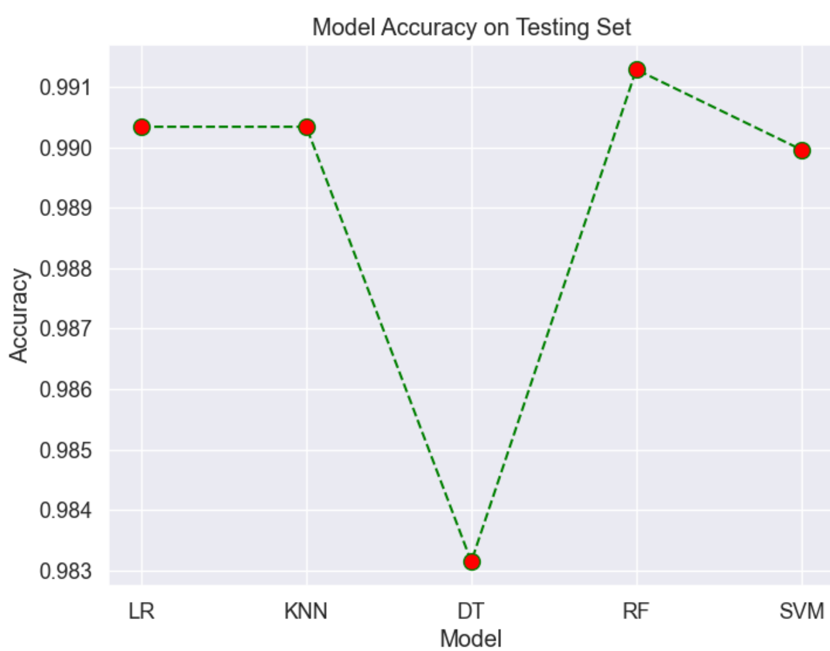
In [114]:
```
from sklearn.model_selection import cross_val_score
from sklearn.calibration import calibration_curve
```

Model Accuracy Comparison

Models	Accuracy on Testing Set
Logistic Regression	0.9903409
K Nearest Neighbor	0.9903409
Decision Tree	0.9831439
Random Forest	0.9912879
Support Vector Machine	0.9899621

In [122]:
```
model_acc = [acc_log, acc_knn, acc_dt, acc_rf, acc_svc]
models = ['LR', 'KNN', 'DT', 'RF', 'SVM']

plt.figure(figsize=(8,6))
plt.plot(models, model_acc, color = 'green', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Model Accuracy on Testing Set')
plt.xlabel('Model')
plt.ylabel('Accuracy')
```

Out[122]:
```
Text(0, 0.5, 'Accuracy')
```



K-Fold Cross Validation (w/ training data)

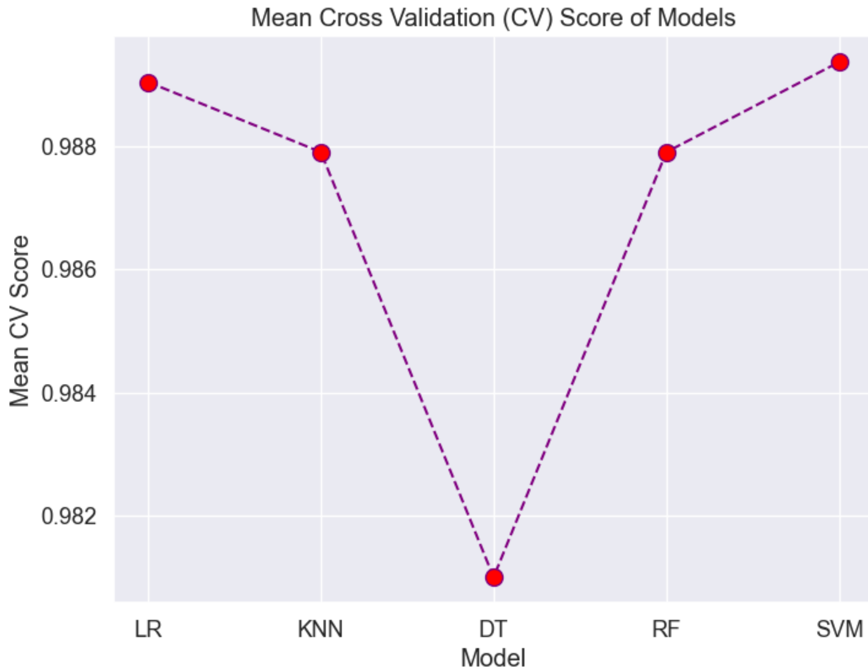
In [116]:
```
k = 5

"""LR"""
cv_scores_log = cross_val_score(log_model, X_train, y_train, cv=k)
cv_mean_log = cv_scores_log.mean()
print("Cross validation scores for LR:", cv_scores_log)
print("Mean cross validation score for LR: {:.7f}".format(cv_mean_log))
print()

"""KNN"""
cv_scores_knn = cross_val_score(knn_model, X_train, y_train, cv=k)
cv_mean_knn = cv_scores_knn.mean()
print("Cross validation scores for KNN:", cv_scores_knn)
print("Mean cross validation score for KNN: {:.7f}".format(cv_mean_knn))
print()

"""DT"""
cv_scores_dt = cross_val_score(dt_model, X_train, y_train, cv=k)
cv_mean_dt = cv_scores_dt.mean()
print("Cross validation scores for DT:", cv_scores_dt)
print("Mean cross validation score for DT: {:.7f}".format(cv_mean_dt))
print()

"""RF"""
cv_scores_rf = cross_val_score(rf_model, X_train, y_train, cv=k)
cv_mean_rf = cv_scores_rf.mean()
print("Cross validation scores for RF:", cv_scores_rf)
print("Mean cross validation score for RF: {:.7f}".format(cv_mean_rf))
print()

"""SVM"""
cv_scores_svc = cross_val_score(svc_model, X_train, y_train, cv=k)
cv_mean_svc = cv_scores_svc.mean()
print("Cross validation scores for SVM:", cv_scores_svc)
print("Mean cross validation score for SVM: {:.7f}".format(cv_mean_svc))
```

```
Cross validation scores for LR: [0.98782468 0.99147727 0.9910678  0.98660171 0.98822574]
Mean cross validation score for LR: 0.9890394
Cross validation scores for KNN: [0.98579545 0.99107143 0.98781973 0.98781973 0.98700771]
Mean cross validation score for KNN: 0.9879028

Cross validation scores for DT: [0.98376623 0.9788961  0.97969955 0.97929354 0.98335363]
Mean cross validation score for DT: 0.9810018

Cross validation scores for RF: [0.98701299 0.99066558 0.98822574 0.98538368 0.98822574]
Mean cross validation score for RF: 0.9879027

Cross validation scores for SVM: [0.98904221 0.99066558 0.99025579 0.98822574 0.98863175]
Mean cross validation score for SVM: 0.9893642
```

Models	Mean Cross Validation Score
Logistic Regression	0.9890394
K Nearest Neighbor	0.9879028
Decision Tree	0.9809206
Random Forest	0.9879838
Support Vector Machine	0.9893642

In [117]:
```
cv_mean = [cv_mean_log, cv_mean_knn, cv_mean_dt, cv_mean_rf, cv_mean_svc]
models = ['LR', 'KNN', 'DT', 'RF', 'SVM']

plt.figure(figsize=(8,6))
plt.plot(models, cv_mean, color = 'purple', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Mean Cross Validation (CV) Score of Models')
plt.xlabel('Model')
plt.ylabel('Mean CV Score')
```

Out[117]:
```
Text(0, 0.5, 'Mean CV Score')
```



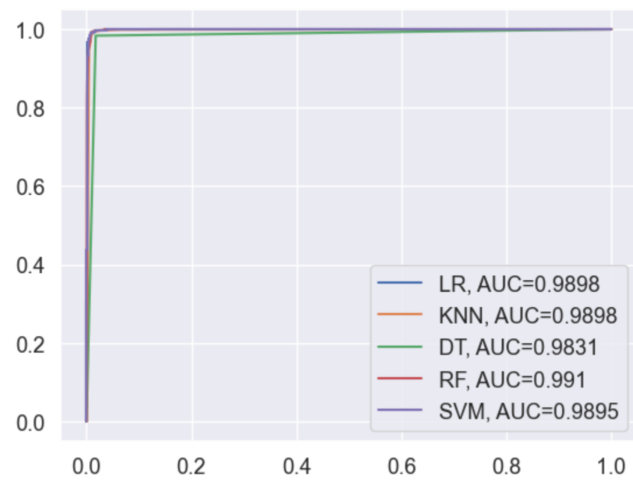
ROC Curve

In [118]:

```
fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_log)
auc = round(metrics.roc_auc_score(y_test, y_pred_log), 4)
plt.plot(fpr,tpr,label="LR, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_knn)
auc = round(metrics.roc_auc_score(y_test, y_pred_knn), 4)
plt.plot(fpr,tpr,label="KNN, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_dt)
auc = round(metrics.roc_auc_score(y_test, y_pred_dt), 4)
plt.plot(fpr,tpr,label="DT, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_rf)
auc = round(metrics.roc_auc_score(y_test, y_pred_rf), 4)
plt.plot(fpr,tpr,label="RF, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_svc)
auc = round(metrics.roc_auc_score(y_test, y_pred_svc), 4)
plt.plot(fpr,tpr,label="SVM, AUC="+str(auc))

plt.legend()
```

Out[118]:

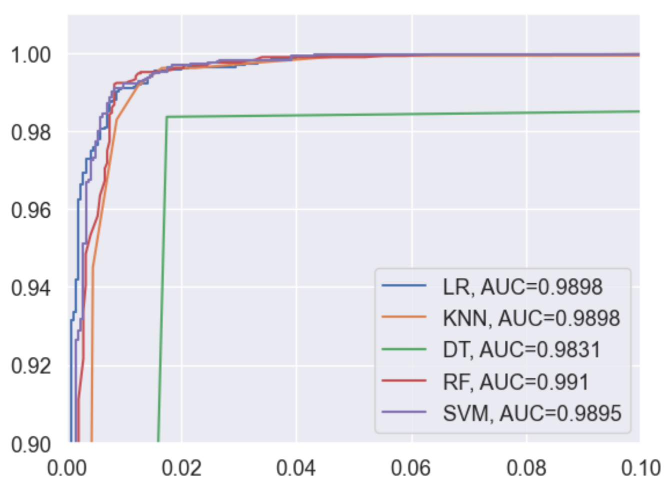
```
<matplotlib.legend.Legend at 0x25e88808670>
```



In [119]:

```
plt.xlim(0.0, 0.1)
plt.ylim(0.9, 1.01)

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_log)
auc = round(metrics.roc_auc_score(y_test, y_pred_log), 4)
plt.plot(fpr,tpr,label="LR, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_knn)
auc = round(metrics.roc_auc_score(y_test, y_pred_knn), 4)
plt.plot(fpr,tpr,label="KNN, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_dt)
auc = round(metrics.roc_auc_score(y_test, y_pred_dt), 4)
plt.plot(fpr,tpr,label="DT, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_rf)
auc = round(metrics.roc_auc_score(y_test, y_pred_rf), 4)
plt.plot(fpr,tpr,label="RF, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test, y_proba_svc)
auc = round(metrics.roc_auc_score(y_test, y_pred_svc), 4)
plt.plot(fpr,tpr,label="SVM, AUC="+str(auc))

plt.legend()
```

Out[119]: <matplotlib.legend.Legend at 0x25e8f1299d0>



--END--