Part 1: Maternal Health Risk Data Set

Attribute Information:

Age: Any ages in years when a women is pregnant.
SystolicBP: Upper value of Blood Pressure in mmHg, another significant attribute during pregnancy.
DiastolicBP: Lower value of Blood Pressure in mmHg, another significant attribute during pregnancy.
BS: Blood glucose levels is in terms of a molar concentration, mmol/L.
HeartRate: A normal resting heart rate in beats per minute.
Risk Level: Predicted Risk Intensity Level during pregnancy considering the previous attribute.

1.1. Load

Below are some basic information about the dataframe.

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk
...	...	...	...	...	...	...	...
1009	22	120	60	15.0	98.0	80	high risk
1010	55	120	90	18.0	98.0	60	high risk
1011	35	85	60	19.0	98.0	86	high risk
1012	43	120	90	18.0	98.0	70	high risk
1013	32	120	65	6.0	101.0	76	mid risk

1014 rows × 7 columns

Figure 1: Dataframe

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate
count	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000
mean	29.871795	113.198225	76.460552	8.725986	98.665089	74.301775
std	13.474386	18.403913	13.885796	3.293532	1.371384	8.088702
min	10.000000	70.000000	49.000000	6.000000	98.000000	7.000000
25%	19.000000	100.000000	65.000000	6.900000	98.000000	70.000000
50%	26.000000	120.000000	80.000000	7.500000	98.000000	76.000000
75%	39.000000	120.000000	90.000000	8.000000	98.000000	80.000000
max	70.000000	160.000000	100.000000	19.000000	103.000000	90.000000

Figure 2: Summary Table

```
RangeIndex: 1014 entries, 0 to 1013
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          1014 non-null   int64
 1   SystolicBP   1014 non-null   int64
 2   DiastolicBP  1014 non-null   int64
 3   BS           1014 non-null   float64
 4   BodyTemp     1014 non-null   float64
 5   HeartRate    1014 non-null   int64
 6   RiskLevel    1014 non-null   object
dtypes: float64(2), int64(4), object(1)
memory usage: 55.6+ KB
```

Figure 3: Information Table

1.2. Encoding on RiskLevel

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk
...	...	...	...	...	...	...	...
1009	22	120	60	15.0	98.0	80	high risk
1010	55	120	90	18.0	98.0	60	high risk
1011	35	85	60	19.0	98.0	86	high risk
1012	43	120	90	18.0	98.0	70	high risk
1013	32	120	65	6.0	101.0	76	mid risk

1014 rows × 7 columns

Figure 4: Before

```
Age              int64
SystolicBP       int64
DiastolicBP      int64
BS             float64
BodyTemp       float64
HeartRate        int64
RiskLevel       object
dtype: object
```

Figure 5: Before

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	2
1	35	140	90	13.0	98.0	70	2
2	29	90	70	8.0	100.0	80	2
3	30	140	85	7.0	98.0	70	2
4	35	120	60	6.1	98.0	76	0
...	...	...	...	...	...	...	...
1009	22	120	60	15.0	98.0	80	2
1010	55	120	90	18.0	98.0	60	2
1011	35	85	60	19.0	98.0	86	2
1012	43	120	90	18.0	98.0	70	2
1013	32	120	65	6.0	101.0	76	1

1014 rows × 7 columns

Figure 6: After

```
Age              int64
SystolicBP       int64
DiastolicBP      int64
BS             float64
BodyTemp       float64
HeartRate        int64
RiskLevel      float64
dtype: object
```

Figure 7: After

1.3. Data Preprocessing

1.3.1. Missing Values

```
Age            0
SystolicBP     0
DiastolicBP    0
BS             0
BodyTemp       0
HeartRate      0
RiskLevel      0
```

Figure 8: No missing values

1.3.2. Duplicated Values

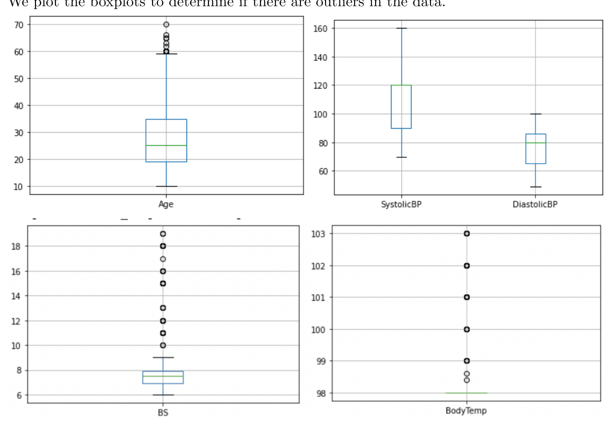
```
0       False
1       False
2       False
3       False
4       False
        ...
1009     True
1010     True
1011     True
1012     True
1013     True
Length: 1014, dtype: bool
```

Figure 9: We have duplicated values

If an element is duplicated in the training data, it is effectively the same as having its 'weight' doubled. That element becomes twice as important when the classifier is fitting our data, and the classifier becomes biased towards correctly classifying that particular scenario over others so we drop the duplicated values.

1.3.3. Checking for Outliers

We plot the boxplots to determine if there are outliers in the data.



As we can see in boxplots, there are no outliers to be dropped.

1.4. Correlation

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
Age	1.000000	0.375931	0.347846	0.376616	-0.188843	0.077407	0.183010
SystolicBP	0.375931	1.000000	0.790002	0.347534	-0.207267	-0.006088	0.327365
DiastolicBP	0.347846	0.790002	1.000000	0.300423	-0.201992	-0.016470	0.254239
BS	0.376616	0.347534	0.300423	1.000000	-0.042511	0.135605	0.548888
BodyTemp	-0.188843	-0.207267	-0.201992	-0.042511	1.000000	0.087262	0.259701
HeartRate	0.077407	-0.006088	-0.016470	0.135605	0.087262	1.000000	0.183289
RiskLevel	0.183010	0.327365	0.254239	0.548888	0.259701	0.183289	1.000000

Figure 10: Correlation Table

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
Age	1.000000	0.375931	0.347846	0.376616	-0.188843	0.077407	0.183010
SystolicBP	0.375931	1.000000	0.790002	0.347534	-0.207267	-0.006088	0.327365
DiastolicBP	0.347846	0.790002	1.000000	0.300423	-0.201992	-0.016470	0.254239
BS	0.376616	0.347534	0.300423	1.000000	-0.042511	0.135605	0.548888
BodyTemp	-0.188843	-0.207267	-0.201992	-0.042511	1.000000	0.087262	0.259701
HeartRate	0.077407	-0.006088	-0.016470	0.135605	0.087262	1.000000	0.183289
RiskLevel	0.183010	0.327365	0.254239	0.548888	0.259701	0.183289	1.000000

Figure 11: Correlation Heatmap 1

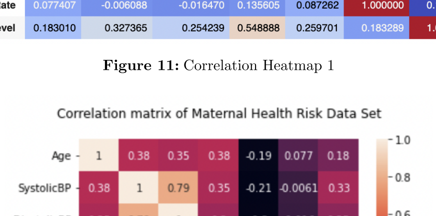


Figure 12: Correlation Heatmap 2

As we can see above our features are:

a. Highly correlated (0.7 - 1): DiastolicBP and SystolicBP
b. Moderately correlated (0.5 - 0.7): BS and RiskLevel

Others have a very low to no correlation.
Thus blood glucose levels may be a good predicate for risk level.

1.5. Shuffle data and split train/test data

I used *train_test_split* in *sklearn.model_selection* to shuffle and split data into 30/70 portions.
Also I used StandardScaler to help you standardize the dataset's features onto unit scale (mean = 0 and variance = 1) which is a requirement for the optimal performance of many machine learning algorithms.

1.6. Classification

In this section we have the

- Classification Model
  - Confusion Matrix
  - Train/Test Accuracy
  - Classification Report

For both before and after parameter tuning using cross validation.

1.6.1. Decision Tree

```
decision_tree confusion_matrix :
 [[58  7  5]
 [19 11 11]
 [ 2  5 18]]
Accuracy: 63.97 %
Train accuracy: 95.25 %
Test accuracy: 63.97 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.73      0.83      0.78        70
         1.0       0.48      0.27      0.34        41
         2.0       0.53      0.72      0.61        25

    accuracy                           0.64       136
   macro avg       0.58      0.61      0.58       136
weighted avg       0.62      0.64      0.62       136
```

Figure 13: Before Parameter Tuning

```
decision_tree confusion_matrix :
 [[70  0  0]
 [29  7  5]
 [ 5  5 15]]
Accuracy: 67.65 %
Train accuracy: 75.0 %
Test accuracy: 67.65 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      1.00      0.80        70
         1.0       0.58      0.17      0.26        41
         2.0       0.75      0.60      0.67        25

    accuracy                           0.68       136
   macro avg       0.67      0.59      0.58       136
weighted avg       0.66      0.68      0.62       136
```

Figure 14: After Parameter Tuning

```
{'ccp_alpha': 0.01,
 'criterion': 'entropy',
 'max_depth': 6,
 'max_features': 'auto'}
```

Figure 15: Best Parameters Using Cross Validation

1.6.2. KNN

```
knn confusion_matrix :
 [[63  5  2]
 [25  8  8]
 [ 7  3 15]]
Accuracy: 63.24 %
Train accuracy: 76.58 %
Test accuracy: 63.24 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.66      0.90      0.76        70
         1.0       0.50      0.20      0.28        41
         2.0       0.60      0.60      0.60        25

    accuracy                           0.63       136
   macro avg       0.59      0.57      0.55       136
weighted avg       0.60      0.63      0.59       136
```

Figure 16: Before Parameter Tuning

```
knn confusion_matrix :
 [[65  2  3]
 [26  7  8]
 [ 7  2 16]]
Accuracy: 64.71 %
Train accuracy: 73.42 %
Test accuracy: 64.71 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.66      0.93      0.77        70
         1.0       0.64      0.17      0.27        41
         2.0       0.59      0.64      0.62        25

    accuracy                           0.65       136
   macro avg       0.63      0.58      0.55       136
weighted avg       0.64      0.65      0.59       136
```

Figure 17: After Parameter Tuning

```
{'leaf_size': 20,
 'metric': 'minkowski',
 'n_neighbors': 10,
 'p': 1,
 'weights': 'uniform'}
```

Figure 18: Best Parameters Using Cross Validation

1.6.3. SVM

```
svm confusion_matrix :
 [[65  0  5]
 [27  5  9]
 [ 5  1 19]]
Accuracy: 65.44 %
Train accuracy: 74.68 %
Test accuracy: 65.44 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      0.93      0.78        70
         1.0       0.83      0.12      0.21        41
         2.0       0.58      0.76      0.66        25

    accuracy                           0.65       136
   macro avg       0.69      0.60      0.55       136
weighted avg       0.70      0.65      0.59       136
```

Figure 19: Before Parameter Tuning

```
svm confusion_matrix :
 [[65  0  5]
 [27  5  9]
 [ 5  1 19]]
Accuracy: 65.44 %
Train accuracy: 74.68 %
Test accuracy: 65.44 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      0.93      0.78        70
         1.0       0.83      0.12      0.21        41
         2.0       0.58      0.76      0.66        25

    accuracy                           0.65       136
   macro avg       0.69      0.60      0.55       136
weighted avg       0.70      0.65      0.59       136
```

Figure 20: After Parameter Tuning

```
{'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

Figure 21: Best Parameters Using Cross Validation

1.6.4. Random Forest

```
random_forest confusion_matrix :
 [[61  4  5]
 [20 12  9]
 [ 2  6 17]]
Accuracy: 66.18 %
Train accuracy: 95.25 %
Test accuracy: 66.18 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.73      0.87      0.80        70
         1.0       0.55      0.29      0.38        41
         2.0       0.55      0.68      0.61        25

    accuracy                           0.66       136
   macro avg       0.61      0.61      0.60       136
weighted avg       0.64      0.66      0.64       136
```

Figure 22: Before Parameter Tuning

```
random_forest confusion_matrix :
 [[66  0  4]
 [26  4 11]
 [ 1  1 23]]
Accuracy: 68.38 %
Train accuracy: 81.65 %
Test accuracy: 68.38 %
classification_report:
              precision    recall  f1-score   support

         0.0       0.71      0.94      0.81        70
         1.0       0.80      0.10      0.17        41
         2.0       0.61      0.92      0.73        25

    accuracy                           0.68       136
   macro avg       0.70      0.65      0.57       136
weighted avg       0.72      0.68      0.60       136
```

Figure 23: After Parameter Tuning

```
{'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 2,
 'min_samples_split': 10,
 'n_estimators': 500}
```

Figure 24: Best Parameters Using Cross Validation

1.7. Analysis

Gathering the model accuracy and test/train accuracy for both before and after parameter tuning into one dataframe and using barplot to visually compare them.

	Classification	ParameterTuning	Accuracy	Train accuracy	Test accuracy
7	Random Forest	after	71.32	82.28	71.32
1	Decision Tree	after	67.65	75.00	67.65
6	Random Forest	before	66.18	95.25	66.18
4	SVM	before	65.44	74.68	65.44
5	SVM	after	65.44	74.68	65.44
3	KNN	after	64.71	73.42	64.71
0	Decision Tree	before	63.97	95.25	63.97
2	KNN	before	63.24	76.58	63.24

Figure 25: Dataframe of Model Accuracy Before and After Parameter Tuning

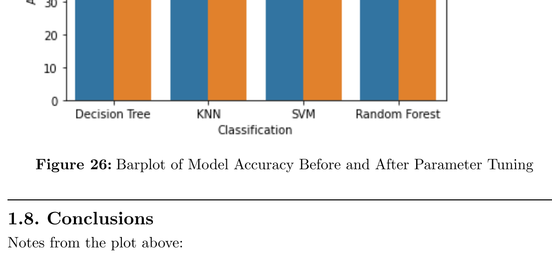


Figure 26: Barplot of Model Accuracy Before and After Parameter Tuning

1.8. Conclusions

Notes from the plot above:

- From **barplots**, we can see that **Random forest** has the **highest accuracy** both before and after parameter tuning.
- In order of accuracy we have from high to low:
  - Random Forest
  - Decision Tree
  - SVM
  - KNN
- **Parameter tuning** using cross validation has **improved the accuracy** for all models except SVM (because the best parameters were already the default parameters of the algorithm). It had the **most impact** on **Decision tree**.
- It's worth to mention that we could improve the accuracy of algorithms even more with cross validation method, if we consider a wider grid search and do the search on more parameters.