

Data **Mining** (COCSC16) **Lab Report**



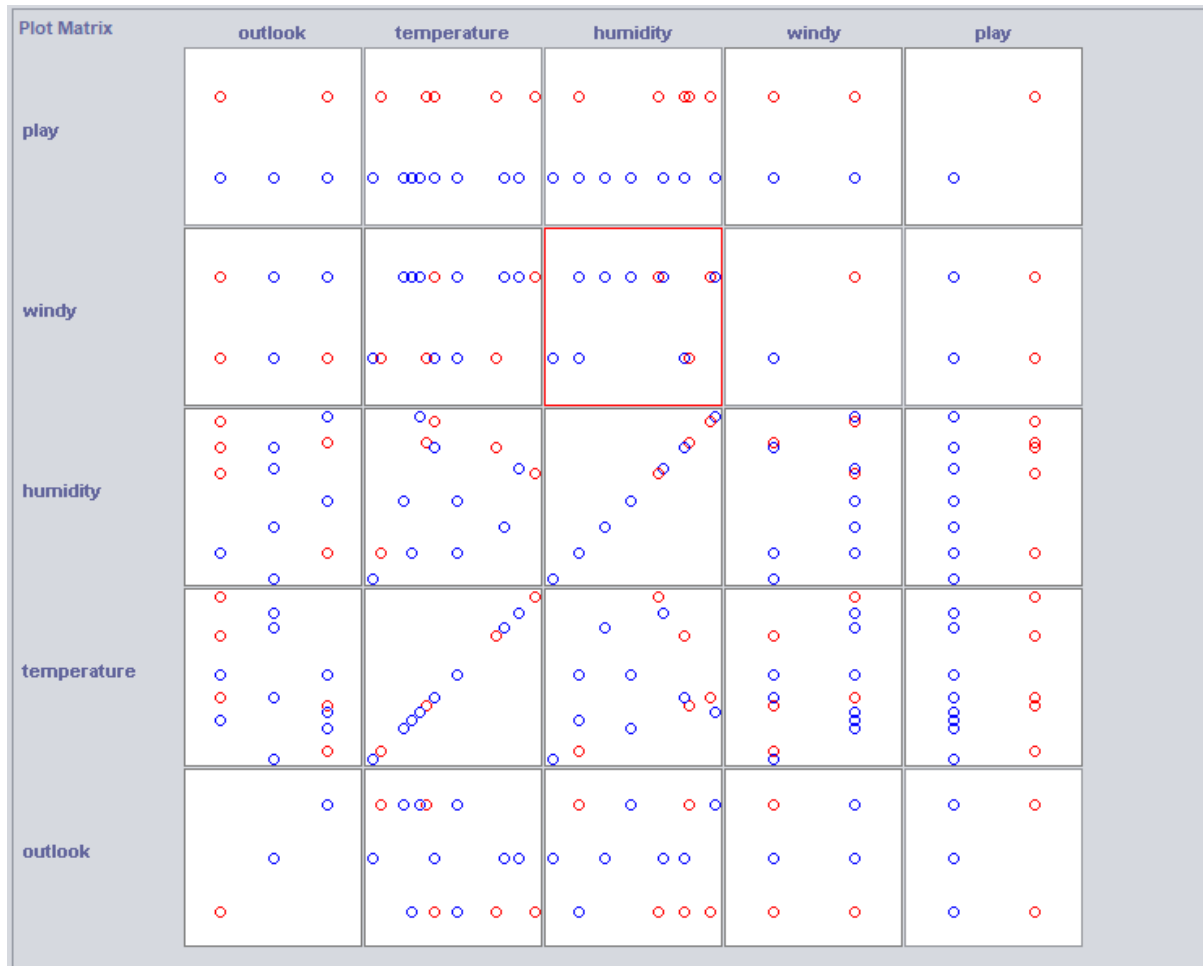
Submitted By: -
Ashish Kumar
2019UCO1518
COE – 1

Index

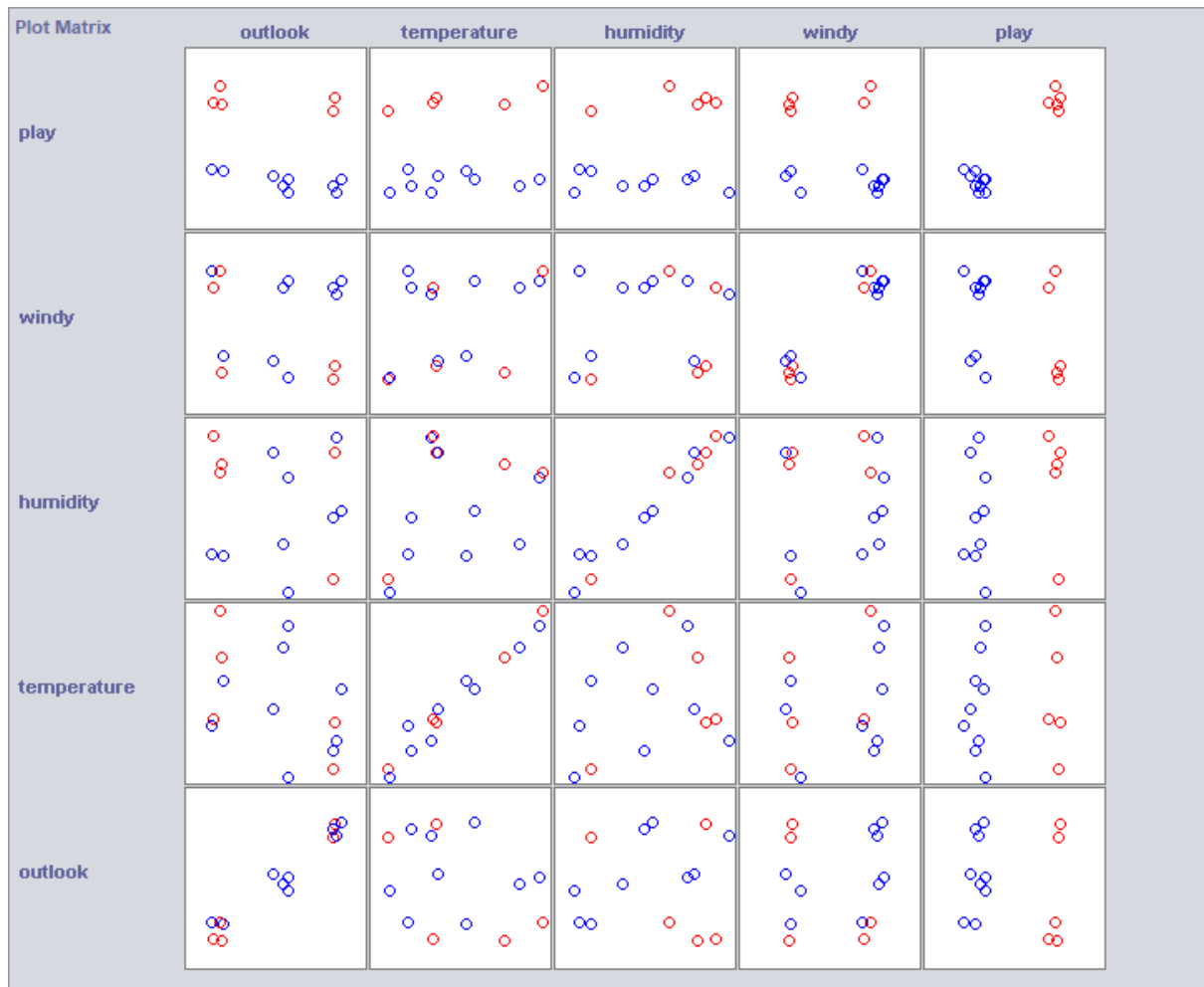
S No	Content	Page No.
1	Visualise your data set in Weka and learn about jitter	3
2	Calculate Co-relation among attributes	5
3	Calculate Information gain of different attributes	6
4	Perform PCA on dataset	9
5	Train a Decision Tree	12
6	Train a Naïve bayes classifier	14
7	Train a Bayesian belief network classifier	15
8	Train a linear regression model	16
9	Train a Logistic regression model	17
10	Train a KNN model	18
11	Train a K-means clustering model	19
12	Analysis of Data Know the types of data – ordinal, nominal, ratio, interval	20
13	Statistical Data Analysis Find the mean, median, variance and standard deviation of data.	21
14	Presentation	23

1.) Visualise your data set in Weka and learn about jitter

Visualising the dataset



After increasing jitter



Conclusion:

Increasing jitter adds a noise to the dataset therefore the points that were overlapping before can now be seen separately.

2.) Calculate Co-relation among attributes

```
outlook
temperature
humidity
windy
play
Evaluation mode:    evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (unsupervised):
    Principal Components Attribute Transformer

Correlation matrix
  1    -0.47  -0.56   0.31   0.03   0.04
-0.47   1    -0.47   0.14  -0.17  -0.09
-0.56  -0.47   1    -0.44   0.13   0.04
  0.31   0.14  -0.44   1     0.32   0.33
  0.03  -0.17   0.13   0.32   1     0.2
  0.04  -0.09   0.04   0.33   0.2   1
```

Correlation matrix tells us which pair of attributes are highly correlated

3.) Calculate Information gain of different attributes

Information gain wrt 'outlook'

```
        humidity
        windy
        play
Evaluation mode:    evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 1 outlook):
    Information Gain Ranking Filter

Ranked attributes:
1.29169  2 temperature
1.23777  3 humidity
0.24675  5 play
0.00598  4 windy

Selected attributes: 2,3,5,4 : 4
```

Information gain wrt temperature

```
        humidity
        windy
        play
Evaluation mode:    evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 2 temperature):
    Information Gain Ranking Filter

Ranked attributes:
2.896   3 humidity
1.292   1 outlook
0.797   5 play
0.7     4 windy

Selected attributes: 3,1,5,4 : 4
```

Information gain wrt humidity

```
        humidity
        windy
        play
Evaluation mode:    evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 3 humidity):
    Information Gain Ranking Filter

Ranked attributes:
2.896  2 temperature
1.238  1 outlook
0.788  4 windy
0.601  5 play

Selected attributes: 2,1,4,5 : 4
```

Information gain wrt windy

```
        humidity
        windy
        play
Evaluation mode:    evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 4 windy):
    Information Gain Ranking Filter

Ranked attributes:
0.78845  3 humidity
0.69951  2 temperature
0.04813  5 play
0.00598  1 outlook

Selected attributes: 3,2,5,1 : 4
```

Information gain wrt play

```
        humidity
        windy
        play
Evaluation mode:    evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 5 play):
    Information Gain Ranking Filter

Ranked attributes:
0.7974  2 temperature
0.6007  3 humidity
0.2467  1 outlook
0.0481  4 windy

Selected attributes: 2,3,1,4 : 4
```


4.) Perform PCA on dataset

Output:

=== Run information ===

Evaluator: weka.attributeSelection.PrincipalComponents -R 0.95 -A 5
 Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
 Relation: weather-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last
 Instances: 14
 Attributes: 5
 outlook
 temperature
 humidity
 windy
 play
 Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
 Attribute ranking.

Attribute Evaluator (unsupervised):
 Principal Components Attribute Transformer

Correlation matrix

1	-0.47	-0.56	-0.21	-0.21	-0.21	0.37	-0.21	-0.21	0.12	0.12	0.37	-0.21	-0.21	0.37	-0.21	0.34	-0.21	-0.3	0.37	-0.21	0.12	-0.21	0.37	-0.21	0.04
-0.47	1	-0.47	0.44	-0.18	-0.18	-0.18	-0.18	-0.18	0.19	-0.26	-0.18	0.44	0.44	-0.18	0.44	-0.33	0.44	-0.26	-0.18	0.44	0.19	-0.18	-0.18	-0.18	-0.09
-0.56	-0.47	1	-0.21	0.37	0.37	-0.21	0.37	0.37	-0.3	0.12	-0.21	-0.21	-0.21	-0.21	-0.21	-0.03	-0.21	0.55	-0.21	-0.21	-0.3	0.37	-0.21	0.37	0.04
-0.21	0.44	-0.21	1	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	1	-0.14	-0.08	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.32	
-0.21	-0.18	0.37	-0.08	1	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	-0.08	0.53	-0.08	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.32	
-0.21	-0.18	0.37	-0.08	-0.08	1	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	-0.08	-0.14	-0.08	0.68	-0.08	-0.08	-0.11	-0.08	-0.08	0.24	
0.37	-0.18	-0.21	-0.08	-0.08	-0.08	1	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	-0.08	0.53	-0.08	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.11	
-0.21	-0.18	0.37	-0.08	-0.08	-0.08	1	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.14	-0.08	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.11	
-0.21	-0.18	0.37	-0.08	-0.08	-0.08	-0.08	1	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.14	-0.08	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.11	
1	-0.08	-0.08	-0.32	0.12	0.19	-0.3	-0.11	-0.11	-0.11	-0.11	-0.11	1	-0.17	-0.11	-0.11	-0.11	-0.11	-0.21	-0.11	-0.17	-0.11	-0.11	0.42	-0.11	
0.12	0.19	-0.3	-0.11	-0.11	-0.11	-0.11	-0.11	-0.11	1	-0.17	-0.11	-0.11	-0.11	-0.11	-0.11	-0.21	-0.11	-0.17	-0.11	-0.11	-0.11	0.42	-0.11	-0.17	
-0.11	-0.11	-0.11	-0.06	0.37	-0.18	-0.21	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	1	-0.08	-0.08	-0.08	-0.08	-0.14	-0.08	-0.11	-0.08	-0.08	0.68	
-0.21	0.44	-0.21	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	1	-0.08	-0.08	-0.08	-0.14	1	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.11	
-0.21	0.44	-0.21	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	1	-0.08	-0.08	-0.14	-0.08	-0.11	-0.08	1	-0.11	-0.08	-0.08	-0.11	
0.37	-0.18	-0.21	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	1	-0.08	-0.14	-0.08	-0.11	1	-0.08	-0.11	-0.08	-0.11	-0.08	
-0.21	0.44	-0.21	1	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	-0.08	1	-0.14	-0.08	-0.11	-0.08	-0.08	-0.11	-0.08	-0.11	-0.08	
0.34	-0.33	-0.03	-0.14	0.53	-0.14	0.53	-0.14	-0.14	-0.21	0.28	-0.14	-0.14	-0.14	-0.14	-0.14	1	-0.14	-0.21	-0.14	-0.14	-0.14	-0.21	-0.14	-0.21	
-0.21	0.44	-0.21	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	1	-0.08	-0.08	-0.08	-0.14	1	-0.11	-0.08	-0.08	-0.11	-0.08	-0.08	-0.11	
-0.3	-0.26	0.55	-0.11	-0.11	0.68	-0.11	-0.11	-0.11	-0.17	0.42	-0.11	-0.11	-0.11	-0.11	-0.11	-0.21	-0.11	1	-0.11	-0.11	-0.17	-0.11	-0.11	-0.17	
0.37	-0.18	-0.21	-0.08	-0.08	-0.08	-0.08	-0.08	-0.08	-0.11	-0.11	-0.08	-0.08	-0.08	1	-0.08	-0.14	-0.08	-0.11	1	-0.08	-0.11	-0.08	-0.11	-0.08	

```

-0.21 0.44 -0.21 -0.08 -0.08 -0.08 -0.08 -0.08 -0.08 -0.11 -0.11 -0.08 -0.08 1 -0.08 -0.08 -0.14 -0.08 -0.11 -0.08 1 -0.11 -
0.08 -0.08 -0.08 0.24
0.12 0.19 -0.3 -0.11 -0.11 -0.11 -0.11 -0.11 -0.11 0.42 -0.17 0.68 -0.11 -0.11 -0.11 -0.21 -0.11 -0.17 -0.11 -0.11 1 -
0.11 -0.11 -0.11 -0.47
-0.21 -0.18 0.37 -0.08 -0.08 -0.08 -0.08 -0.08 1 -0.11 -0.11 -0.08 -0.08 -0.08 -0.08 -0.14 -0.08 -0.11 -0.08 -0.08 -0.11
1 -0.08 -0.08 -0.32
0.37 -0.18 -0.21 -0.08 -0.08 -0.08 -0.08 -0.08 -0.08 0.68 -0.11 -0.08 -0.08 -0.08 -0.08 -0.14 -0.08 -0.11 -0.08 -0.08 -
0.11 -0.08 1 -0.08 0.24
-0.21 -0.18 0.37 -0.08 -0.08 -0.08 -0.08 1 -0.08 -0.11 -0.11 -0.08 -0.08 -0.08 -0.08 -0.14 -0.08 -0.11 -0.08 -0.08 -0.11
-0.08 -0.08 1 0.24
0.04 -0.09 0.04 -0.32 -0.32 0.24 0.24 0.24 -0.32 -0.06 -0.06 -0.32 0.24 0.24 0.24 -0.32 -0.25 0.24 0.35 0.24 0.24 -0.47
-0.32 0.24 0.24 1

```

eigenvalue proportion cumulative

```

3.54154 0.13621 0.13621 -0.468outlook=rainy+0.414outlook=overcast-
0.275humidity=80+0.206temperature=64+0.206humidity=65...
3.27247 0.12586 0.26208 -0.506outlook=sunny+0.312outlook=overcast-0.242temperature=85-0.242humidity=85-
0.225humidity=70...
2.88832 0.11109 0.37317 -0.543windy=FALSE+0.28 humidity=91+0.28
temperature=71+0.233temperature=64+0.233humidity=65...
2.25961 0.08691 0.46007 -
0.452humidity=70+0.306humidity=96+0.306temperature=70+0.288temperature=72+0.275humidity=90...
2.18654 0.0841 0.54417 -0.487temperature=85-0.487humidity=85+0.282temperature=72+0.269humidity=90-
0.252humidity=91...
2.15385 0.08284 0.62701 -0.415humidity=75-0.415temperature=81+0.327humidity=65+0.327temperature=64-
0.327humidity=91...
2.15385 0.08284 0.70985 -0.527humidity=86-0.527temperature=83+0.337humidity=75+0.337temperature=81-
0.19humidity=91...
2.07209 0.0797 0.78955 -0.481humidity=80-
0.41temperature=68+0.332humidity=70+0.319humidity=96+0.319temperature=70...
1.73819 0.06685 0.8564 -0.55humidity=95+0.477temperature=80-0.391temperature=72+0.365humidity=90-
0.145humidity=91...
1.3611 0.05235 0.90875 0.638temperature=65-0.425temperature=69+0.243temperature=72-
0.218outlook=sunny+0.206temperature=85...
1.16019 0.04462 0.95337 0.744temperature=75-0.441temperature=68-0.422temperature=69-0.163temperature=65-
0.116windy=FALSE...

```

Eigenvectors

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
0.0777	-0.506	-0.0615	-0.0065	-0.0191	-0	-0	0.0297	-0.0346	-0.218	0.0339
	outlook=sunny									
0.4145	0.3122	0.0008	-0.029	0.0097	0	0	-0.0005	0.0112	0.0634	0.0082
	outlook=overcast									
-0.4685	0.2117	0.0607	0.0338	0.01	0	0	-0.0293	0.024	0.1582	-0.0417
	outlook=rainy									
0.2064	0.1717	0.2327	-0.1464	-0.2469	0.3266	0.1903	-0.1443	-0.1279	-0.109	-0.0372
	temperature=64									
-0.1269	-0.0133	0.154	-0.2401	0.0467	-0	0	0.2209	0.0379	0.6379	-0.1631
	temperature=65									
-0.2003	0.1106	-0.1146	-0.0762	0.1032	0	0	-0.4095	0.0616	-0.0101	-0.4409
	temperature=68									
-0.0134	-0.1925	-0.053	-0.2509	0.0398	-0	0	0.2339	-0.1343	-0.4248	-0.4215
	temperature=69									
-0.1987	0.1296	-0.1346	0.3059	0.0291	0.3266	0.1903	0.319	0.0477	-0.0692	0.0578
	temperature=70									
-0.1724	0.1099	0.2798	0.207	-0.2518	-0.3266	-0.1903	0.0595	-0.1448	-0.1754	-0.0079
	temperature=71									
0.1705	-0.1304	0.0419	0.2877	0.2818	-0	0	-0.1396	-0.3908	0.243	0.0208
	temperature=72									
-0.1519	-0.0706	0.0003	-0.2675	0.1079	-0	0	-0.1248	0.0411	-0.149	0.7438
	temperature=75									
0.0906	-0.1839	0.1726	0.183	0.1695	-0	0	-0.0516	0.4767	-0.1771	-0.0969
	temperature=80									
0.1801	0.1914	-0.1817	-0.0475	0.034	-0.415	0.3369	0.1151	0.0645	-0.0028	0.0285
	temperature=81									
0.1801	0.1914	-0.1817	-0.0475	0.034	0.0884	-0.5272	0.1151	0.0645	-0.0028	0.0285
	temperature=83									
0.0292	-0.2418	-0.2308	0.0853	-0.4869	0	-0	-0.0988	0.1292	0.2055	0.0138
	temperature=85									
0.2064	0.1717	0.2327	-0.1464	-0.2469	0.3266	0.1903	-0.1443	-0.1279	-0.109	-0.0372 humidity=65

-0.1088	-0.2252	0.1087	-0.4524	0.0889	-0	0	0.3325	-0.0519	0.0625	-0.0316	humidity=70
0.1801	0.1914	-0.1817	-0.0475	0.034	-0.415	0.3369	0.1151	0.0645	-0.0028	0.0285	humidity=75
-0.275	0.1234	-0.1371	-0.1545	0.1433	0	0	-0.4813	0.0764	-0.0728	0.0262	humidity=80
0.0292	-0.2418	-0.2308	0.0853	-0.4869	0	-0	-0.0988	0.1292	0.2055	0.0138	humidity=85
0.1801	0.1914	-0.1817	-0.0475	0.034	0.0884	-0.5272	0.1151	0.0645	-0.0028	0.0285	humidity=86
0.1847	-0.1404	0.2243	0.2748	0.269	-0	0	-0.1017	0.3646	0.0359	-0.0752	humidity=90
-0.1724	0.1099	0.2798	0.207	-0.2518	-0.3266	-0.1903	0.0595	-0.1448	-0.1754	-0.0079	humidity=91
0.0712	-0.1703	-0.0752	0.2004	0.1869	-0	0	-0.1031	-0.5496	0.1043	0.0336	humidity=95
-0.1987	0.1296	-0.1346	0.3059	0.0291	0.3266	0.1903	0.319	0.0477	-0.0692	0.0578	humidity=96
-0.0651	0.0393	-0.5431	0.0187	0.0164	-0	0	-0.0378	-0.1427	-0.1502	-0.1165	windy=FALSE

Ranked attributes:

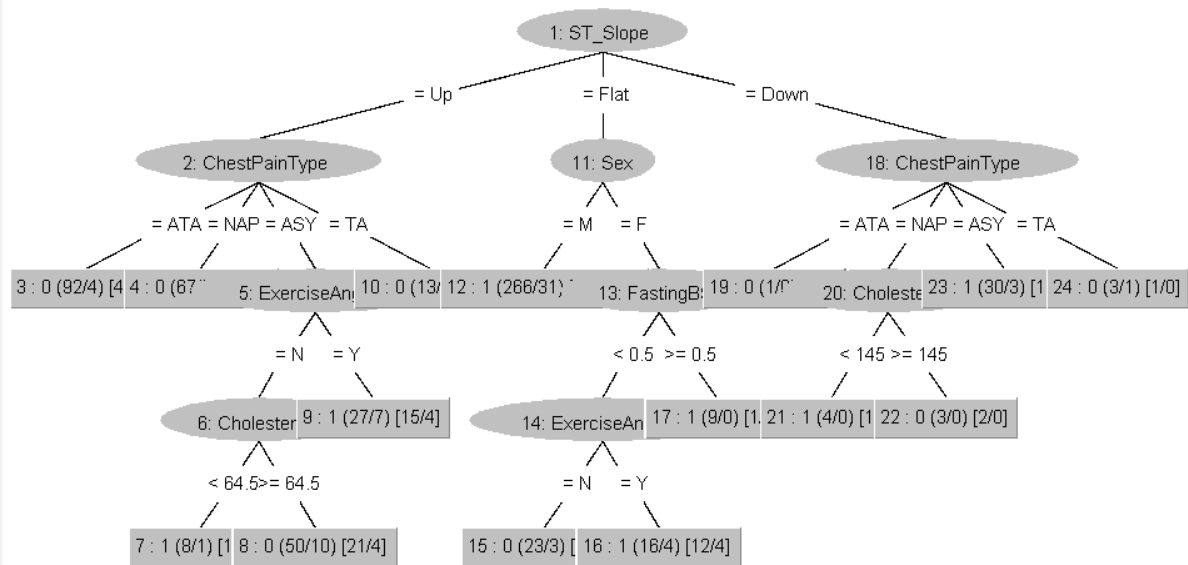
0.8638 1 -0.468outlook=rainy+0.414outlook=overcast-0.275humidity=80+0.206temperature=64+0.206humidity=65...
0.7379 2 -0.506outlook=sunny+0.312outlook=overcast-0.242temperature=85-0.242humidity=85-0.225humidity=70...
0.6268 3 -0.543windy=FALSE+0.28 humidity=91+0.28 temperature=71+0.233temperature=64+0.233humidity=65...
0.5399 4 -0.452humidity=70+0.306humidity=96+0.306temperature=70+0.288temperature=72+0.275humidity=90...
0.4558 5 -0.487temperature=85-0.487humidity=85+0.282temperature=72+0.269humidity=90-0.252humidity=91...
0.373 6 -0.415humidity=75-0.415temperature=81+0.327humidity=65+0.327temperature=64-0.327humidity=91...
0.2901 7 -0.527humidity=86-0.527temperature=83+0.337humidity=75+0.337temperature=81-0.19humidity=91...
0.2105 8 -0.481humidity=80-0.41temperature=68+0.332humidity=70+0.319humidity=96+0.319temperature=70...
0.1436 9 -0.55humidity=95+0.477temperature=80-0.391temperature=72+0.365humidity=90-0.145humidity=91...
0.0912 10 0.638temperature=65-0.425temperature=69+0.243temperature=72-0.218outlook=sunny+0.206temperature=85...
0.0466 11 0.744temperature=75-0.441temperature=68-0.422temperature=69-0.163temperature=65-0.116windy=FALSE...

Selected attributes: 1,2,3,4,5,6,7,8,9,10,11 : 11

5.) Train a Decision Tree

batchSize	100
debug	False
doNotCheckCapabilities	False
initialCount	0.0
maxDepth	-1
minNum	2.0
minVarianceProp	0.001
noPruning	False
numDecimalPlaces	2
numFolds	3
seed	1
spreadInitialCount	False

Tree View



=== Summary ===

Correctly Classified Instances	760	82.7887 %
Incorrectly Classified Instances	158	17.2113 %
Kappa statistic	0.6492	
Mean absolute error	0.2357	
Root mean squared error	0.3623	
Relative absolute error	47.684 %	
Root relative squared error	72.8761 %	
Total Number of Instances	918	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.768	0.124	0.833	0.768	0.799	0.651	0.876	0.843	0
	0.876	0.232	0.824	0.876	0.849	0.651	0.876	0.854	1
Weighted Avg.	0.828	0.184	0.828	0.828	0.827	0.651	0.876	0.849	

=== Confusion Matrix ===

```
a  b  <-- classified as
315 95 |  a = 0
63 445 |  b = 1
```

6.) Train a naïve bayes classifier

Time taken to build model: 0 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	5	35.7143 %
Incorrectly Classified Instances	9	64.2857 %
Kappa statistic	-0.4651	
Mean absolute error	0.4724	
Root mean squared error	0.5032	
Relative absolute error	99.2024 %	
Root relative squared error	101.9848 %	
Total Number of Instances	14	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.556	1.000	0.500	0.556	0.526	-0.471	0.511	0.780	yes
	0.000	0.444	0.000	0.000	0.000	-0.471	0.511	0.383	no
Weighted Avg.	0.357	0.802	0.321	0.357	0.338	-0.471	0.511	0.638	

=== Confusion Matrix ===

```
a b  <-- classified as
5 4 | a = yes
5 0 | b = no
```

7.) Train a Bayesian belief network classifier

Network structure (nodes followed by parents)

```

outlook(3): play
temperature(12): play
humidity(10): play
windy(2): play
play(2):
LogScore Bayes: -115.33073562481593
LogScore BDeu: -246.09337962191782
LogScore MDL: -212.8110485913026
LogScore ENTROPY: -150.79320134534402
LogScore AIC: -197.79320134534402

```

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
 === Summary ===

Correctly Classified Instances	6	42.8571 %
Incorrectly Classified Instances	8	57.1429 %
Kappa statistic	-0.2444	
Mean absolute error	0.457	
Root mean squared error	0.5026	
Relative absolute error	95.9602 %	
Root relative squared error	101.8771 %	
Total Number of Instances	14	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.556	0.800	0.556	0.556	0.556	-0.244	0.511	0.780	yes
	0.200	0.444	0.200	0.200	0.200	-0.244	0.511	0.383	no
Weighted Avg.	0.429	0.673	0.429	0.429	0.429	-0.244	0.511	0.638	

=== Confusion Matrix ===

```

a b  <-- classified as
5 4 | a = yes
4 1 | b = no

```

8.) Train a Linear Regression Model

```
Linear Regression Model
```

```
HeartDisease =
```

```
    0.003 * Age +  
    0.1618 * Sex=M +  
    0.2357 * ChestPainType=ASY +  
-0.0005 * Cholesterol +  
    0.1317 * FastingBS +  
    0.1396 * ExerciseAngina=Y +  
    0.0499 * Oldpeak +  
    0.2196 * ST_Slope=Down,Flat +  
    0.164 * ST_Slope=Flat +  
-0.0996
```

```
Time taken to build model: 0.09 seconds
```

```
=== Cross-validation ===
```

```
=== Summary ===
```

Correlation coefficient	0.7476
Mean absolute error	0.2444
Root mean squared error	0.3302
Relative absolute error	49.4224 %
Root relative squared error	66.3966 %
Total Number of Instances	918

9.) Train a Logistic regression model

```

time taken to build model: 0.13 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      7           50      %
Incorrectly Classified Instances    7           50      %
Kappa statistic                    -0.2564
Mean absolute error                 0.493
Root mean squared error             0.6688
Relative absolute error             103.5257 %
Root relative squared error         135.5607 %
Total Number of Instances          14

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0.778    1.000    0.583     0.778    0.667      -0.304   0.411    0.619    yes
                0.000    0.222    0.000     0.000    0.000      -0.304   0.400    0.349    no
Weighted Avg.    0.500    0.722    0.375     0.500    0.429      -0.304   0.407    0.522

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 5 0 | b = no

```

10.) Train a KNN model

```
=== Summary ===
```

Correctly Classified Instances	735	80.0654 %
Incorrectly Classified Instances	183	19.9346 %
Kappa statistic	0.5979	
Mean absolute error	0.2001	
Root mean squared error	0.4459	
Relative absolute error	40.4746 %	
Root relative squared error	89.7013 %	
Total Number of Instances	918	

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.793	0.193	0.768	0.793	0.780	0.598	0.798	0.702	0
	0.807	0.207	0.828	0.807	0.818	0.598	0.798	0.781	1
Weighted Avg.	0.801	0.201	0.802	0.801	0.801	0.598	0.798	0.746	

```
=== Confusion Matrix ===
```

```
  a   b  <-- classified as
325  85 |   a = 0
 98 410 |   b = 1
```

11.) Train a K-means clustering model

```

=== Clustering model (full training set) ===

kMeans
=====

Number of iterations: 2
Within cluster sum of squared errors: 35.0

Initial starting points (random):

Cluster 0: rainy,75,80,FALSE,yes
Cluster 1: overcast,64,65,TRUE,yes

Missing values globally replaced with mean/mode

Final cluster centroids:

```

		Cluster#	
Attribute	Full Data	0	1
	(14.0)	(11.0)	(3.0)
=====			
outlook	sunny	rainy	overcast
temperature	72	75	64
humidity	70	70	90
windy	FALSE	FALSE	TRUE
play	yes	yes	yes

```

Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      11 ( 79%)
1       3 ( 21%)

```

12.) Analysis of Data Know the types of data – ordinal, nominal, ratio, interval

- Nominal
 - Nominal scales are used for labeling variables, without any quantitative value. “Nominal” scales could simply be called “labels.”
 - all of the scales shown below are mutually exclusive (no overlap) and none of them have any numerical significance.
 - Properties: distinctness (= != are meaningful)
 - Ex. ID Numbers, eye color, Zip Codes
- Ordinal
 - With ordinal scales, the order of the values is what’s important and significant, but the differences between each one is not really known.
 - Ordinal scales are typically measures of non-numeric concepts like satisfaction, happiness, discomfort, etc.
 - distinctness and order, (= != < > are meaningful)
 - Ex. Grades, ranks etc.
- Interval
 - Interval scales are numeric scales in which we know both the order and the exact differences between the values.
 - For interval attributes, differences between values are meaningful.
 - Properties: distinctness, order and differences, (= != < > + - are meaningful)
 - Ex. Calendar dates, temperature in Celsius
- Ratio
 - For ratio attributes, both differences and ratio are meaningful.
 - They have an absolute 0.
 - Properties: distinctness, order, differences and ratios (= != < > + - * / are meaningful)
 - Ex. Temperature in Kelvin, age, mass.

13.) Statistical Data Analysis Find the mean, median, variance and standard deviation of data.

Code:

```
#include <bits/stdc++.h>
#include <fstream>

using namespace std;

int main() {

    ifstream ip("data.csv");

    if (!ip.is_open()) std::cout << "ERROR: File Open" << '\n';
    vector<vector<string>> mat;

    while (ip.good()) {
        string temp;
        getline(ip, temp, '\n');
        string str = temp;
        vector<string> v;

        stringstream ss(str);

        while (ss.good()) {
            string substr;
            getline(ss, substr, ',');
            v.push_back(substr);
        }
        mat.push_back(v);
    }
    ip.close();
    int n = mat.size();
    int m = mat[0].size();
    cout<<"There are "<<n<<" data entries"<<endl;
    cout<<"There are "<<m<<" features"<<endl;
    for(int j=0;j<m;j++){
        double mean = 0;
        for (int i = 0; i < n; i++) {
            mean += stoi(mat[i][j]);
        }
        mean /= (double)n;
        double variance = 0;
        for (int i = 0; i < n; i++) {
            double x = abs(mean - stoi(mat[i][j]));
            x = x * x;
            variance += x;
        }
        variance /= (double)n;
        double std_deviation = sqrt(variance);
        vector<int> vec;
        for (int i = 0; i < n; i++) {
```

```

        vec.push_back(stoi(mat[i][j]));
    }
    sort(vec.begin(), vec.end());
    double median = 0;
    if (n & 1) {
        int idx = (n + 1) / 2;
        idx--;
        median = vec[idx];
    }
    else {
        int idx = n / 2;
        median = vec[idx];
        median += vec[idx - 1];
        median /= (double)2;
    }
    cout<<"Measures for "<<j<<" feature : "<<endl;
    cout << "mean" << " : " << mean << endl;
    cout << "median" << " : " << median << endl;
    cout << "variance" << " : " << variance << endl;
    cout << "standard deviation" << " : " << std_deviation << endl;
    }
    return 0;
}

```

Output:

```

Measures for 0 feature :
mean : 6
median : 6
variance : 16.6667
standard deviation : 4.08248
Measures for 1 feature :
mean : 7
median : 7
variance : 16.6667
standard deviation : 4.08248
Measures for 2 feature :
mean : 8
median : 8
variance : 16.6667
standard deviation : 4.08248
Measures for 3 feature :
mean : 9
median : 9
variance : 16.6667
standard deviation : 4.08248
Measures for 4 feature :
mean : 9.66667
median : 10
variance : 20.2222
standard deviation : 4.49691
[Finished in 8.3s]

```

73 lines, 1485 characters selected

Presentation

Presentation attached:

DETERMINING THE FETAL HEALTH STATUS

A PRESENTATION BY -

ASHISH KUMAR 2019UCO1518

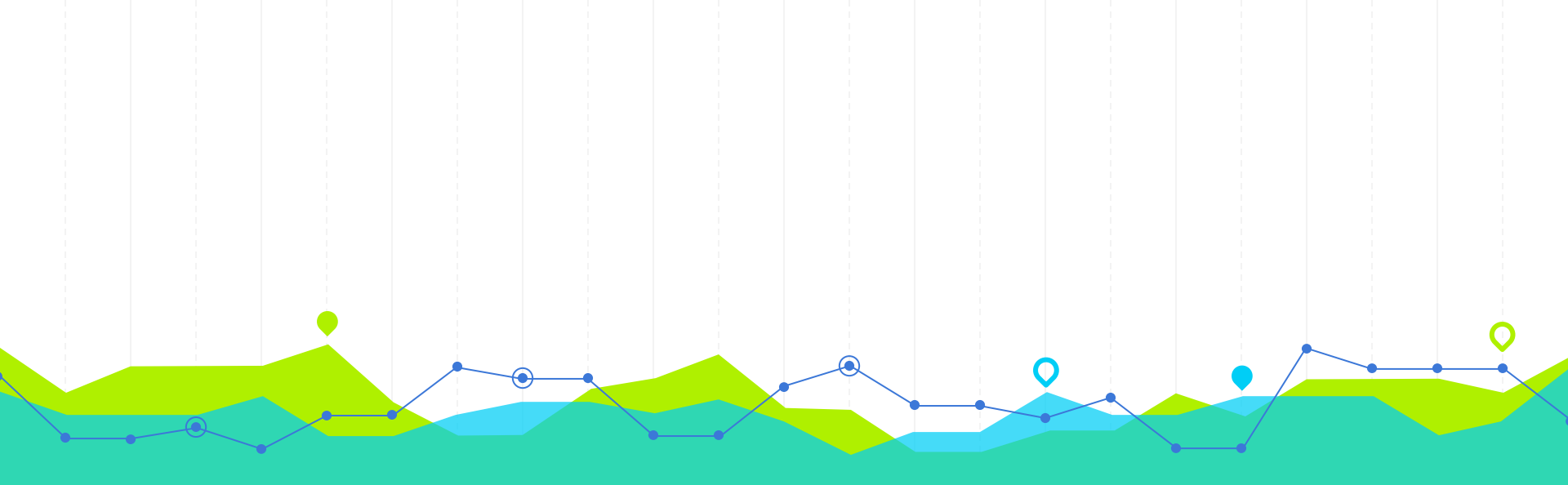
CHETAN RAGHAV 2019UCO1525

CHIRAG KANKHERIA 2019UCO1527

NISHANT GOEL 2019UCO1529

KAUSHAL AGGARWAL 2019UCO1535





Objective - Predicting Fetal Health and Performing Classification

In this work, we use machine learning for predicting fetal health to prevent child and maternal mortality.

1

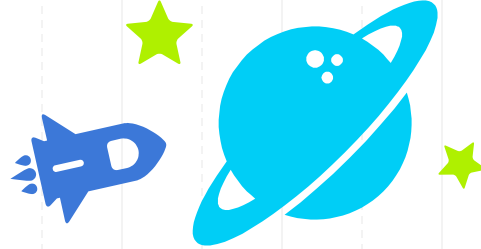
PROJECT CONTENT :

- Data Analysis / Visualization
- Data Preprocessing
- Correlation & Correlation Matrix
- Predictive Modeling
- Confusion Matrix
- Precision and Recall
- Knowledge Discovery

GOAL:

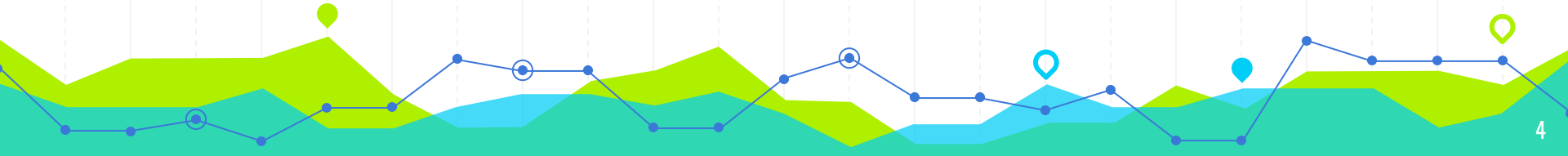
The goal of this project is to predict Fetal Health. We will use various Classification Algorithms to achieve the lowest prediction error.





Machine Learning Methods

We have applied consolidated methodologies to identify the most suitable machine learning model for the task from a pool of candidate methodologies. We have taken into consideration a pool of three state-of-the-art machine learning models, that are briefly reviewed in the following:



Logistic Regression (LR):
It is the baseline model in this Kernel.

The key advantages of LR are its simplicity, the scalability to very large datasets and the interpretation it provides in terms of how unitary changes in an input feature influence the log-odds of the associated linear parameter.

K-nearest neighbors (KNN): K-Nearest Neighbor is a memory-based model, where predictions are performed by the similarity of the current sample to k nearest elements in the training set, according to the given distance metric.

The key advantage of this method lies in its sheer simplicity, compensated by the difficulties in robustly determining the most appropriate similarity function as well as the choice of the k meta-parameter.

Random Forest (RF): It is a type of ensemble of methods in which multiple learning models are combined together to improve generalization.

The rationale behind this selection of candidate models was to provide reasonable coverage of different methodologies, to achieve the lowest prediction error.

The principle of this challenge is :

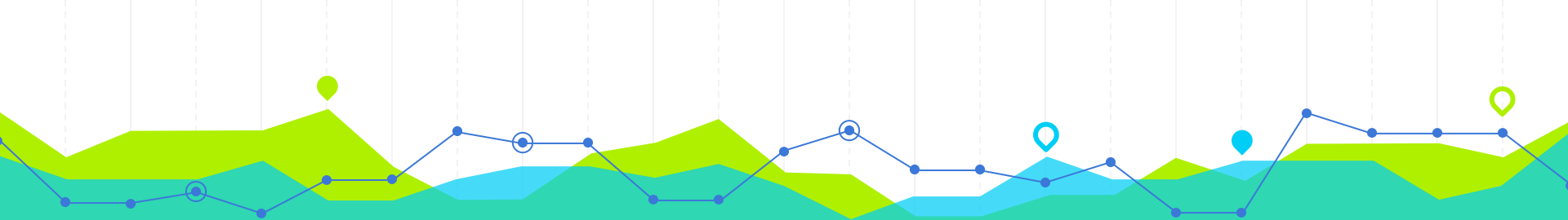
Classify fetal health in order to prevent child and maternal mortality. In order to do so we first get information about "fetal_health" which is classified by expert obstetricians into 3 classes as:

- Normal
- Suspect
- Pathological

To do so, we will analyze and visualize the target column (fetal_health).



Dataset Overview



22 PRIMARY
FEATURES

2000+
DATAPPOINTS

Important features (biologically) among these are :



Baseline value : The baseline FHR is the heart rate during a 10 min segment rounded to the nearest 5 beat per min increment excluding periods of marked FHR variability, periodic or episodic changes and segments of baseline that differ by more than 25 beats per min (bpm). The minimum baseline duration must be at least 2 min.



Accelerations: Accelerations are short-term rises in the heart rate of at least 15 beats per minute above the baseline, lasting at least 15 seconds. Accelerations are normal and healthy. They tell the doctor that the baby has an adequate oxygen supply, which is critical.



Uterine_contractions: Contractions are used to gauge the activity of labour. Too few contractions indicate labour is not progressing. Too many contractions can mean uterine hyperstimulation, which can lead to fetal compromise.



Fetal_movement: It's often called kick counting. It's done by counting the number of kicks you feel from your baby in the uterus in a certain time period.

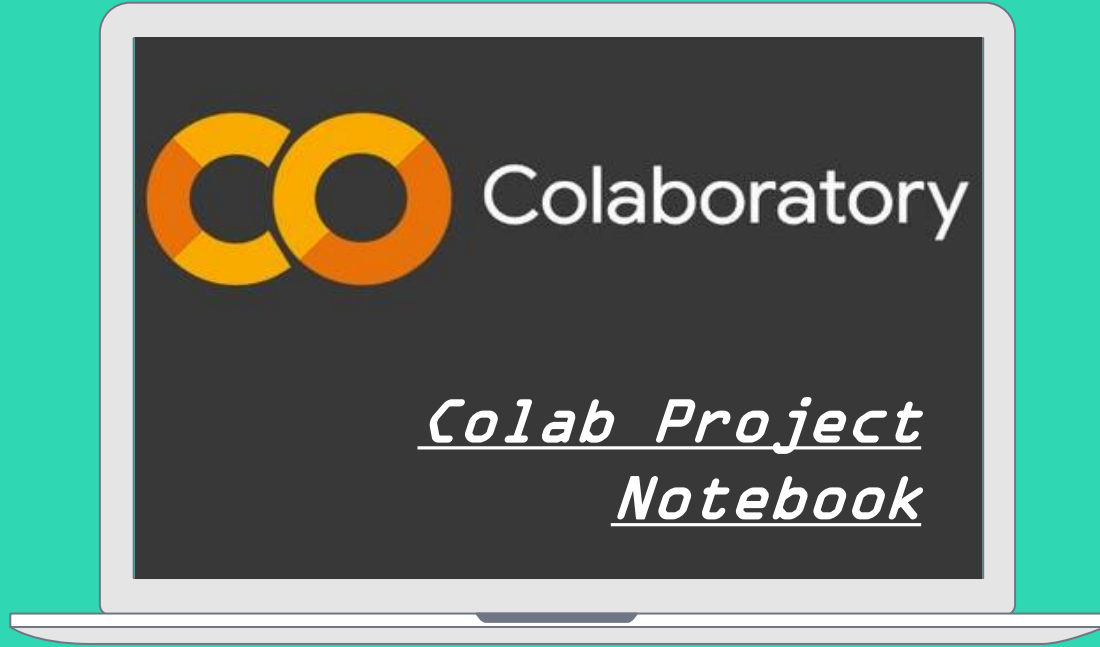


Decelerations: Decelerations are transient episodes of decrease of FHR below the baseline of more than 15 bpm lasting at least 15 seconds. It can be of type:

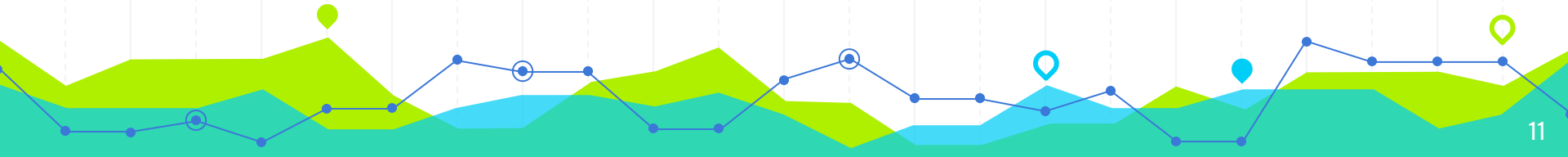
- Light decelerations
- Severe decelerations
- Prolonged decelerations: Prolonged decelerations are defined as a decrease of FHR below the baseline of more than 15 bpm for longer than 90 seconds but less than 5 minutes.



Lets explore
the notebook :



Colab Notebook attached on
next page



DATA MINING

- ASHISH KUMAR 2019UCO1518
- CHETAN RAGHAV 2019UCO1525
- CHIRAG KANKHERIA 2019UCO1527
- NISHANT GOEL 2019UCO1529
- KAUSHAL AGGRAWAL 2019UCO1535

SUBMITTED TO MS. ANSHIKA ARORA

▼ Predicting Fetal Health Classification:

In this work, we use machine learning for the prediction of fetal health to prevent child and maternal mortality.

PROJECT CONTENT:

- Data Analysis / Visualization
- Data Preprocessing
- Correlation & Correlation Matrix
- Predictive Modeling
- Confusion Matrix
- Precision and Recall
- Knowledge Discovery

Goal:

The goal of this project is to predict Fetal Health. We will use various Classification Algorithms to achieve the lowest prediction error.

The **principle** of this challenge is:

Classify fetal health in order to prevent child and maternal mortality. In order to do so we get information about "fetal_health" which is classified by expert obstetricians into 3 classes as:

- Normal (class 1)
- Suspect (class 2)

- Pathological (class 3)

To do so, we will **analyze and visualize** the target column (fetal_health).

Machine learning methods:

We have applied consolidated methodologies to identify the most suitable machine learning model for the task from a pool of candidate methodologies. We have taken into consideration a pool of three state-of-the-art machine learning models, that are briefly reviewed in the following:

- **Logistic Regression (LR)**: It is the baseline model in this Kernel.

**NOTE: The key advantages of LR are its simplicity, the scalability to very large datasets and the interpretation it provides in terms of how unitary changes in an input feature influence the log-odds of the associated linear parameter. **

- **K-nearest neighbors (KNN)**: K-Nearest Neighbor is a memory-based model, where predictions are performed by the similarity of the current sample to k nearest elements in the training set, according to the given distance metric.

**NOTE: The key advantage of this method lies in its sheer simplicity, compensated by the difficulties in robustly determining the most appropriate similarity function as well as the choice of the k meta-parameter. **

- **Random Forest (RF)**: It is a type of ensemble methods in which multiple learning models are combined together to improve generalization.

The **rationale** behind this selection of candidate models was to provide reasonable coverage of different methodologies, to achieve the lowest prediction error.

▼ Import Necessary Libraries and Data Sets.

```
from subprocess import check_output

import warnings
warnings.simplefilter(action="ignore")

from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# Import the necessary packages
import numpy as np
import pandas as pd

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
```

```
# Algorithms
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import learning_curve

# Load Dataset
data = pd.read_csv('fetal_health.csv')
```

▼ Exploratory Data Analysis (EDA) and Analysis

In this step we want to get basic information about the data types, columns, null value counts, memory usage, etc. EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

```
data.head()
```

	baseline value	accelerations	fetal_movement	uterine_contractions	light_deceleration
0	120.0	0.000	0.0	0.000	0.00
1	132.0	0.006	0.0	0.006	0.00
2	133.0	0.003	0.0	0.008	0.00
3	134.0	0.003	0.0	0.008	0.00
4	132.0	0.007	0.0	0.008	0.00

```
#shape of dataset
print(f"The dataset size: {data.shape}")
```

```
The dataset size: (2126, 22)
```

Observation:

- there are 2126 data points with 22 features

nominal attributes

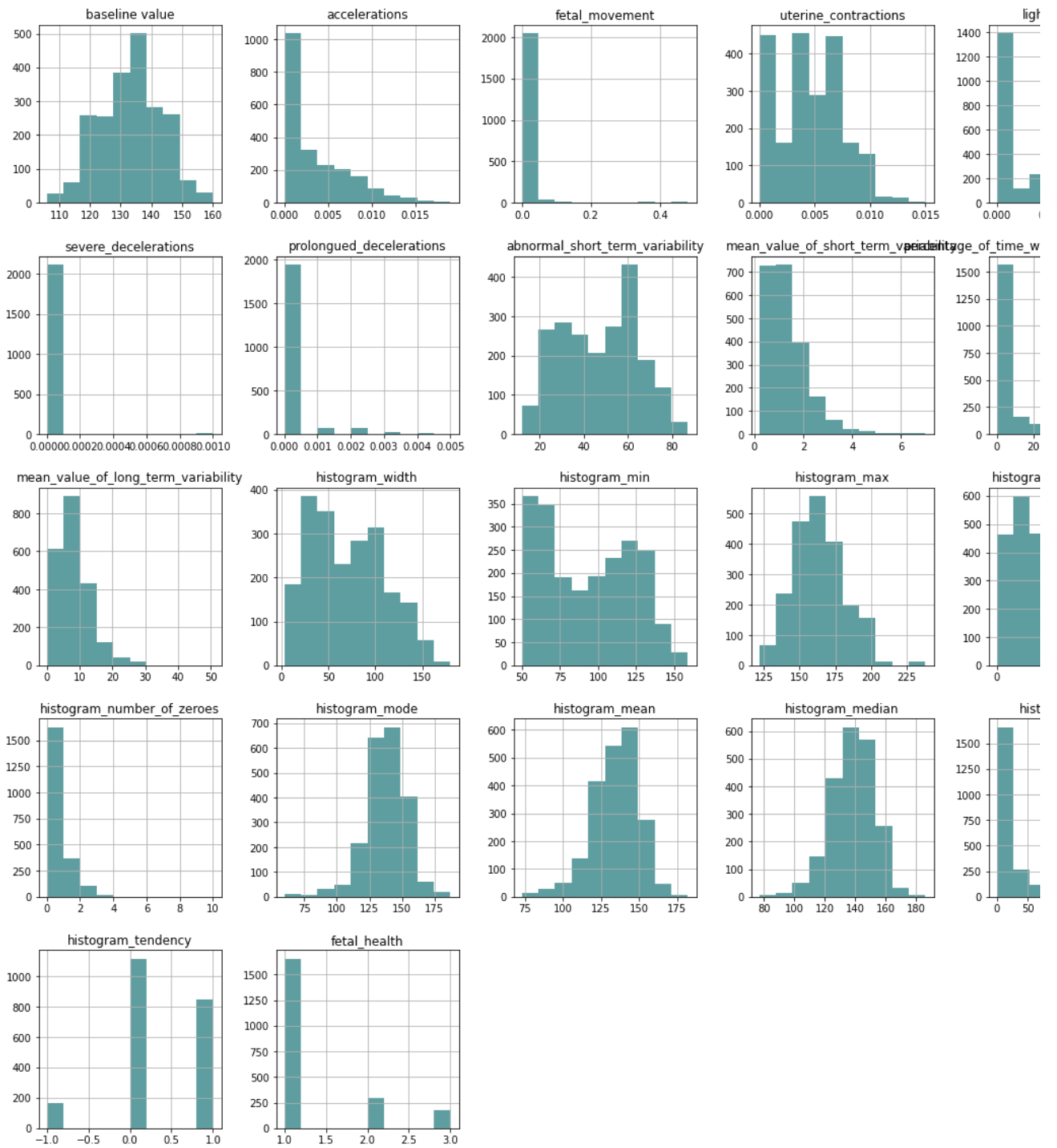
```
# Analyse statically insight of data
data.describe().T
```

	count	mean	std	min
baseline value	2126.0	133.303857	9.840844	106.0
accelerations	2126.0	0.003178	0.003866	0.0
fetal_movement	2126.0	0.009481	0.046666	0.0
uterine_contractions	2126.0	0.004366	0.002946	0.0
light_decelerations	2126.0	0.001889	0.002960	0.0
severe_decelerations	2126.0	0.000003	0.000057	0.0
prolongued_decelerations	2126.0	0.000159	0.000590	0.0
abnormal_short_term_variability	2126.0	46.990122	17.192814	12.0
mean_value_of_short_term_variability	2126.0	1.332785	0.883241	0.2
percentage_of_time_with_abnormal_long_term_variability	2126.0	9.846660	18.396880	0.0
mean_value_of_long_term_variability	2126.0	8.187629	5.628247	0.0
histogram_width	2126.0	70.445908	38.955693	3.0
histogram_min	2126.0	93.579492	29.560212	50.0
histogram_max	2126.0	164.025400	17.944183	122.0
histogram_number_of_peaks	2126.0	4.068203	2.949386	0.0
histogram_number_of_zeroes	2126.0	0.323612	0.706059	0.0
histogram_mode	2126.0	137.452023	16.381289	60.0
histogram_mean	2126.0	134.610536	15.593596	73.0
histogram_median	2126.0	138.090310	14.466589	77.0
histogram_variance	2126.0	18.808090	28.977636	0.0
histogram_tendency	2126.0	0.320320	0.610829	-1.0
fetal_health	2126.0	1.304327	0.614377	1.0

Data visualizations of "fetal_health" column shows us the percentage of fetal health state

so , now we check the distribution of feature data and check if there are outliers or not.

```
data_hist_plot = data.hist(figsize = (20,20), color = "#5F9EA0")
```



Observation

- target feature is nominal
- class 1 is majority class
- feature values do not belong to same range so it needs standardisation/normalisation
- there are no outliers

▼ Count the missing and null values

Here is good to count the **missing** and **null** values. In the case of a real-world dataset, it is very common that some values in the dataset are missing. We represent these missing values as NaN (Not a Number) values. But to build a good machine learning model our dataset should be complete. That's why we use some imputation techniques to replace the NaN values with some probable values.

```
# Count the missing and null values for dataset fetal health.
miss_values = data.columns[data.isnull().any()]
print(f"Missing values:\n{data[miss_values].isnull().sum()}")

null_values = data.columns[data.isna().any()]
print(f"Null values:\n{data[null_values].isna().sum()}")
```

```
Missing values:
Series([], dtype: float64)
Null values:
Series([], dtype: float64)
```

Observation

- In this case, there is *neither null values nor missing values* in the dataset.

Correlation Numeric features with output variable(fetal_health)

Correlation & Correlation Matrix

Here, we want to show the correlation between numerical features and the target "fetal_health", in order to have a first idea of the connections between features. Just by looking at the heatmap below we can see some features have the dark colors, Those features have high correlation with the target.

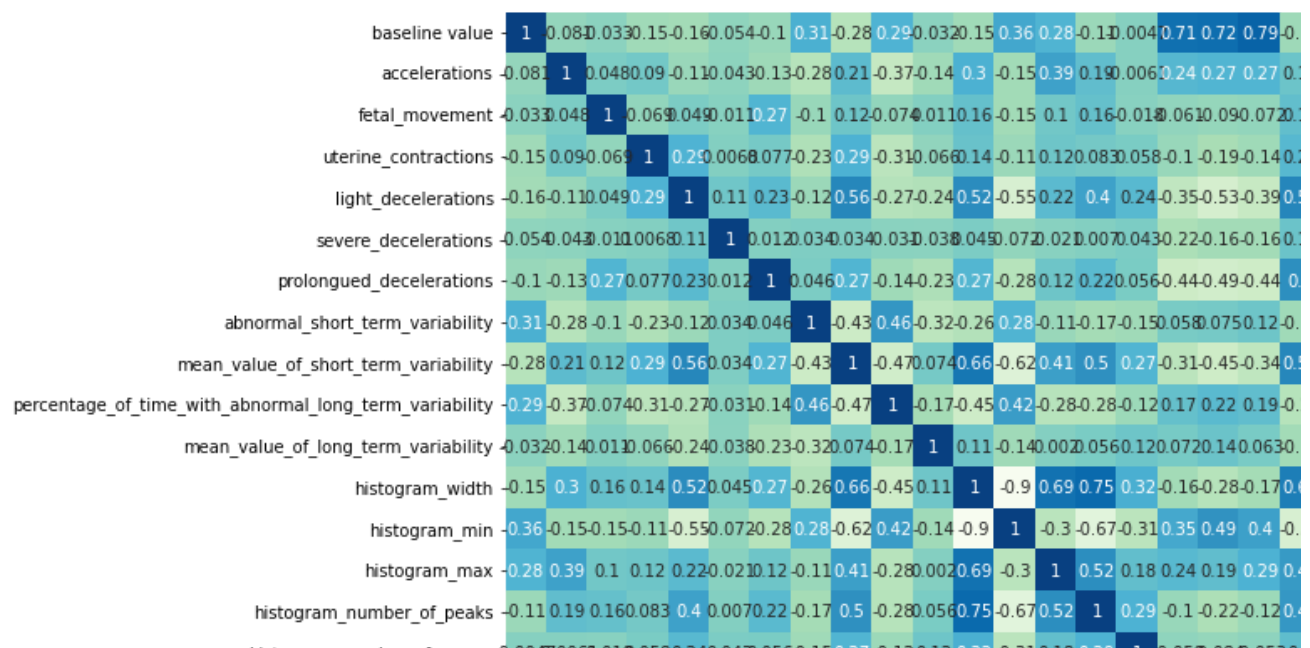
▼ Heatmap

A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information. Correlation heatmaps are ideal for comparing the measurement for each pair of dimension values.

```
# Set the size of figure to 12 by 10.  
plt.figure(figsize=(12,10))  
  
# Seaborn has very simple solution for heatmap  
p=sns.heatmap(data.corr(), annot=True, cmap = "GnBu")
```

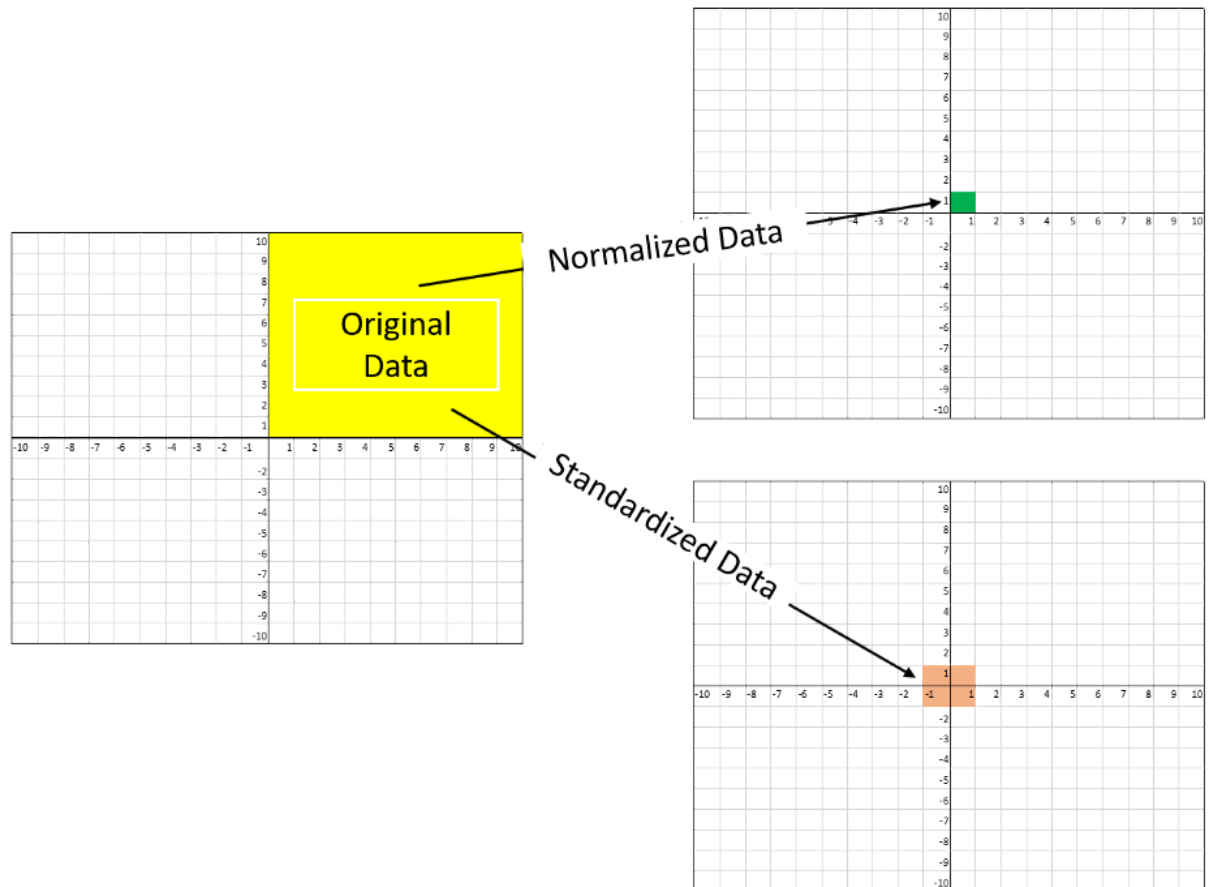
Observations :

- We can see three features: "prolongued_decelerations", "abnormal_short_term_variability", "percentage_of_time_with_abnormal_long_term_variability" have high correlation with the target column (fetal_health)
- although histogram_median, histogram_mode and histogram_mean are highly correlated removing any of them didn't give us the good results.



Scaling the data

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model. Scaling can make a difference between a weak machine learning model and a better one. The most common techniques of feature scaling are Normalization and Standardization. Normalization is used when we want to bound our values between two numbers, typically, between [0,1] or [-1,1]. While Standardization transforms the data to have zero mean and a variance of 1, they make our data unitless. Refer to the below diagram, which shows how data looks after scaling in the X-Y plane.



we are using `StandardScaler()` , which by default standardize features such that the mean is 0 and variance is 1 .

```
columns = ['baseline value', 'accelerations', 'fetal_movement',
           'uterine_contractions', 'light_decelerations', 'severe_decelerations',
           'prolongued_decelerations', 'abnormal_short_term_variability',
           'mean_value_of_short_term_variability',
           'percentage_of_time_with_abnormal_long_term_variability',
           'mean_value_of_long_term_variability', 'histogram_width',
           'histogram_min', 'histogram_max', 'histogram_number_of_peaks',
           'histogram_number_of_zeroes', 'histogram_mode', 'histogram_mean',
           'histogram_median', 'histogram_variance', 'histogram_tendency']
scale_X = StandardScaler()
X = pd.DataFrame(scale_X.fit_transform(data.drop(["fetal_health"],axis = 1)), columns =
X.head()
```

	baseline value	accelerations	fetal_movement	uterine_contractions	light_deceleration
0	-1.352220	-0.822388	-0.20321	-1.482465	-0.63843
1	-0.132526	0.730133	-0.20321	0.554627	0.37524

```
y = data["fetal_health"]
```

```
y
```

```
0      2.0
1      1.0
2      1.0
3      1.0
4      1.0
```

```
...
2121   2.0
2122   2.0
2123   2.0
2124   2.0
2125   1.0
```

```
Name: fetal_health, Length: 2126, dtype: float64
```

▼ Test Train Split and Cross Validation methods

- **Train Test Split** : To have unknown datapoints to test the data rather than testing with the same points with which the model was trained. This helps capture the model performance much better.
- **Cross-validation**, sometimes called rotation estimation or out-of-sample testing, is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set.

```
# Importing train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1488, 21), (638, 21), (1488,), (638,))
```

Building Machine Learning Models:

1. Logistic Regression (LR)
2. K-nearest neighbors (KNN)
3. Random Forest (RF)

NOTE: To improve all scores for each ML model, we want to search the set of "hyperparameters" by using the common approach "Grid search" for three models above. **Hyper-parameters** are a set of additional, model-dependent parameters that are not inferred automatically by the

learning algorithm but need to be specified before the learning phase: a common example of **hyper-parameter** is the value of k in k -Nearest Neighbor or the number of hidden units in a Neural Network. Hence, finding sub-optimal values of the hyper-parameters is crucial to ensure proper generalization. *The hyper-parameter optimization procedure*, which was repeated separately for each candidate learning methodology, encompassed the following steps:

- Firstly, a set of suitable hyper-parameters to optimize was identified; for each of them, a range of candidate values was specified. These choices are dependent both on our expertise and on the computational cost needed to train the models.
- Secondly, a predictor was learned for all the possible combinations of hyper-parameters and its out-of-sample performance was estimated using 3-fold Cross Validation (CV), i.e. We trained the model with 70% of the total training set size and validated its performance in the remaining 30%.

GridSearch exhaustively searches through all possible combinations of hyperparameters during training the phase. Before we proceed further, we shall cover another cross-validation (CV) methods since tuning hyperparameters via grid search is usually cross-validated to avoid overfitting. Hence, For accelerating the running GridSearchCV we set: $n_splits=3$, $n_jobs=2$.

▼ Logistic Regression (LR)

```
# Baseline model of Logistic Regression with default parameters:

logistic_regression = linear_model.LogisticRegression()
logistic_regression_mod = logistic_regression.fit(X_train, y_train)
print(f"Baseline Logistic Regression: {round(logistic_regression_mod.score(X_test, y_test)

pred_logistic_regression = logistic_regression_mod.predict(X_test)

    Baseline Logistic Regression: 0.893

cv_method = StratifiedKFold(n_splits=3 )

# Cross validate Logistic Regression model
scores_Logistic = cross_val_score(logistic_regression, X_train, y_train, cv =cv_method, n_

print(f"Scores(Cross validate) for Logistic Regression model:\n{scores_Logistic}")
print(f"CrossValMeans: {round(scores_Logistic.mean(), 3)}")
print(f"CrossValStandard Deviation: {round(scores_Logistic.std(), 3)}")

    Scores(Cross validate) for Logistic Regression model:
    [0.90927419 0.89717742 0.88306452]
    CrossValMeans: 0.897
    CrossValStandard Deviation: 0.011

params_LR = {"tol": [0.0001,0.0002,0.0003],
```

```
"C": [0.01, 0.1, 1, 10, 100],
"intercept_scaling": [1, 2, 3, 4]
}
```

- tol : Tolerance for stopping criteria.
- C : Inverse of regularization strength .smaller values specify stronger regularization.

```
GridSearchCV_LR = GridSearchCV(estimator=linear_model.LogisticRegression(),
                                param_grid=params_LR,
                                cv=cv_method,
                                verbose=1,
                                n_jobs=2,
                                scoring="accuracy",
                                return_train_score=True
                                )
```

```
# Fit model with train data
GridSearchCV_LR.fit(X_train, y_train);
best_estimator_LR = GridSearchCV_LR.best_estimator_
print(f"Best estimator for LR model:\n{best_estimator_LR}")
```

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
Best estimator for LR model:
LogisticRegression(C=10)
```

```
best_params_LR = GridSearchCV_LR.best_params_
print(f"Best parameter values for LR model:\n{best_params_LR}")
```

```
Best parameter values for LR model:
{'C': 10, 'intercept_scaling': 1, 'tol': 0.0001}
```

```
print(f"Best score for LR model: {round(GridSearchCV_LR.best_score_, 3)}")
```

```
Best score for LR model: 0.899
```

```
# The grid search returns the following as the best parameter set
logistic_regression = linear_model.LogisticRegression(C=10, intercept_scaling=1, tol=0.000
logistic_regression_mod = logistic_regression.fit(X_train, y_train)
pred_logistic_regression = logistic_regression_mod.predict(X_test)
```

```
mse_logistic_regression = mean_squared_error(y_test, pred_logistic_regression)
rmse_logistic_regression = np.sqrt(mean_squared_error(y_test, pred_logistic_regression))
score_logistic_regression_train = logistic_regression_mod.score(X_train, y_train)
score_logistic_regression_test = logistic_regression_mod.score(X_test, y_test)
```

```
print(f"Mean Square Error for Logistic Regression = {round(mse_logistic_regression, 3)}")
print(f"Root Mean Square Error for Logistic Regression = {round(rmse_logistic_regression,
print(f"R^2(coefficient of determination) on training set = {round(score_logistic_regressi
print(f"R^2(coefficient of determination) on testing set = {round(score_logistic_regressio
```

Mean Square Error for Logistic Regression = 0.138
 Root Mean Square Error for Logistic Regression = 0.371
 R²(coefficient of determination) on training set = 0.907
 R²(coefficient of determination) on testing set = 0.886

Based on the result above, after tuning our model (LR), We could boost the model just a little bit.
 So we keep going with other models.

▼ Model Performance Analysis

- **Classification Report:** Report which includes Precision, Recall and F1-Score.

1. **Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

2. **Recall (Sensitivity)** - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

3. **F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

$$\text{F1 Score} = 2(\text{Recall Precision}) / (\text{Recall} + \text{Precision})$$

```
print("Classification Report")
print(classification_report(y_test, pred_logistic_regression))
```

Classification Report				
	precision	recall	f1-score	support
1.0	0.93	0.95	0.94	497
2.0	0.60	0.66	0.63	88
3.0	0.95	0.70	0.80	53
accuracy			0.89	638
macro avg	0.83	0.77	0.79	638
weighted avg	0.89	0.89	0.89	638

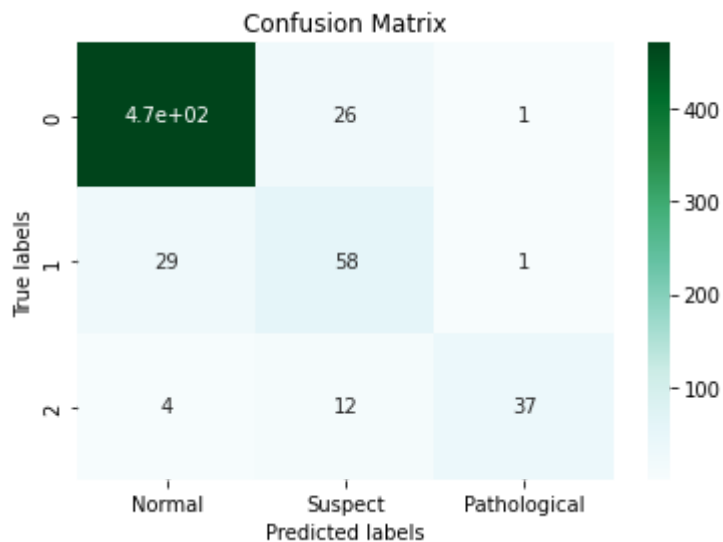
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred_logistic_regression))
```

Confusion Matrix:

```
[[470  26   1]
 [ 29  58   1]
 [   4  12  37]]
```

```
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, pred_logistic_regression), annot=True, ax = ax, cmap
```

```
# labels, title and ticks
ax.set_xlabel("Predicted labels");
ax.set_ylabel("True labels");
ax.set_title("Confusion Matrix");
ax.xaxis.set_ticklabels(["Normal", "Suspect", "Pathological"]);
```



▼ K-Nearest Neighbors (KNN)

Baseline model of K-Nearest Neighbors with default parameters:

```
knn = KNeighborsClassifier()
knn_mod = knn.fit(X_train, y_train)
print(f"Baseline K-Nearest Neighbors: {round(knn_mod.score(X_test, y_test), 3)}")

pred_knn = knn_mod.predict(X_test)
```

Baseline K-Nearest Neighbors: 0.878

▼ Here, we are going to tune the baseline model to boost the model.

```
# Cross validate K-Nearest Neighbors model
cv_method = StratifiedKFold(n_splits=3)

scores_knn = cross_val_score(knn, X_train, y_train, cv = cv_method, n_jobs = 2, scoring =
```



```
print(f"Scores(Cross validate) for K-Nearest Neighbors model:\n{scores_knn}")
print(f"CrossValMeans: {round(scores_knn.mean(), 3)}")
print(f"CrossValStandard Deviation: {round(scores_knn.std(), 3)}")
```

```
Scores(Cross validate) for K-Nearest Neighbors model:
[0.90927419 0.89717742 0.89717742]
CrossValMeans: 0.901
CrossValStandard Deviation: 0.006
```

```
params_knn = {"leaf_size": list(range(1,30)),
              "n_neighbors": list(range(1,21)),
              "p": [1,2]}
```

- leaf_size : maximum number of points a leaf can hold.
- n_neighbors : neighbors to be considered
- p = 1 means manhattan distance
- p = 2 means euclidean distance

```
GridSearchCV_knn = GridSearchCV(estimator=KNeighborsClassifier(),
                                param_grid=params_knn,
                                cv=cv_method,
                                verbose=1,
                                n_jobs=-1,
                                scoring="accuracy",
                                return_train_score=True
                                )
```

```
# Fit model with train data
GridSearchCV_knn.fit(X_train, y_train);
```

Fitting 3 folds for each of 1160 candidates, totalling 3480 fits

```
best_estimator_knn = GridSearchCV_knn.best_estimator_
print(f"Best estimator for KNN model:\n{best_estimator_knn}")
```

```
Best estimator for KNN model:
KNeighborsClassifier(leaf_size=1, n_neighbors=3, p=1)
```

```
best_params_knn = GridSearchCV_knn.best_params_
print(f"Best parameter values:\n{best_params_knn}")
```

```
Best parameter values:
{'leaf_size': 1, 'n_neighbors': 3, 'p': 1}
```

```
# Test with new parameter for KNN model
knn = KNeighborsClassifier(leaf_size=1, n_neighbors=3 , p=1)
knn_mod = knn.fit(X_train, y_train)
pred_knn = knn_mod.predict(X_test)
```

```

pred_knn = knn_mod.predict(X_test)

mse_knn = mean_squared_error(y_test, pred_knn)
rmse_knn = np.sqrt(mean_squared_error(y_test, pred_knn))
score_knn_train = knn_mod.score(X_train, y_train)
score_knn_test = knn_mod.score(X_test, y_test)

print(f"Mean Square Error for K_Nearest Neighbor = {round(mse_knn, 3)}")
print(f"Root Mean Square Error for K_Nearest Neighbor = {round(rmse_knn, 3)}")
print(f"R^2(coefficient of determination) on training set = {round(score_knn_train, 3)}")
print(f"R^2(coefficient of determination) on testing set = {round(score_knn_test, 3)}")

    Mean Square Error for K_Nearest Neighbor = 0.132
    Root Mean Square Error for K_Nearest Neighbor = 0.363
    R^2(coefficient of determination) on training set = 0.956
    R^2(coefficient of determination) on testing set = 0.897

print("Classification Report")
print(classification_report(y_test, pred_knn))

```

Classification Report					
	precision	recall	f1-score	support	
1.0	0.94	0.96	0.95	497	
2.0	0.66	0.66	0.66	88	
3.0	0.88	0.68	0.77	53	
accuracy			0.90	638	
macro avg	0.83	0.77	0.79	638	
weighted avg	0.90	0.90	0.89	638	

```

print("Confusion Matrix:")
print(confusion_matrix(y_test, pred_knn))

```

```

Confusion Matrix:
[[478  18   1]
 [ 26  58   4]
 [   5  12  36]]

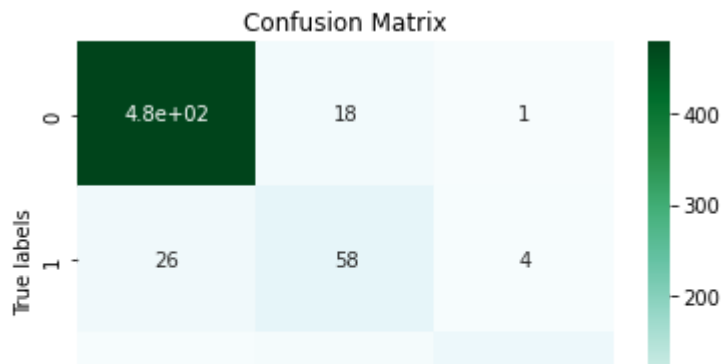
```

```

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, pred_knn), annot=True, ax = ax, cmap = "BuGn");

# labels, title and ticks
ax.set_xlabel("Predicted labels");
ax.set_ylabel("True labels");
ax.set_title("Confusion Matrix");
ax.xaxis.set_ticklabels(["Normal", "Suspect", "Pathological"]);

```



▼ Random Forest (RF)

Normal Suspect Pathological

Baseline model of RF with default parameters:

```
random_forest = RandomForestClassifier()
random_forest_mod = random_forest.fit(X_train, y_train)
print(f"Baseline Random Forest: {round(random_forest_mod.score(X_test, y_test), 3)}")
```

```
pred_random_forest = random_forest_mod.predict(X_test)
```

Baseline Random Forest: 0.92

Cross validate Random forest model

```
scores_RF = cross_val_score(random_forest, X_train, y_train, cv = cv_method, n_jobs = 2, s
```

```
print(f"Scores(Cross validate) for Random forest model:\n{scores_RF}")
```

```
print(f"CrossValMeans: {round(scores_RF.mean(), 3)}")
```

```
print(f"CrossValStandard Deviation: {round(scores_RF.std(), 3)}")
```

Scores(Cross validate) for Random forest model:

[0.94959677 0.9375 0.93548387]

CrossValMeans: 0.941

CrossValStandard Deviation: 0.006

```
params_RF = {"min_samples_split": [2, 6, 20],
             "min_samples_leaf": [1, 4, 16],
             "n_estimators": [100, 200, 300, 400],
             "criterion": ["gini"]}
}
```

- `min_samples_split`: The minimum number of samples required to split an internal node
- `min_sample_leaf`: The minimum number of samples required to be at a leaf node
- `n_estimators`: The number of trees in the forest.

```
GridSearchCV_RF = GridSearchCV(estimator=RandomForestClassifier(),
                                param_grid=params_RF,
                                cv=cv_method,
                                verbose=1,
                                n_jobs=2,
```

```

        scoring="accuracy",
        return_train_score=True
    )

```

```

# Fit model with train data
GridSearchCV_RF.fit(X_train, y_train);

```

Fitting 3 folds for each of 36 candidates, totalling 108 fits

```

best_estimator_RF = GridSearchCV_RF.best_estimator_
print(f"Best estimator for RF model:\n{best_estimator_RF}")

```

```

Best estimator for RF model:
RandomForestClassifier(n_estimators=300)

```

```

best_params_RF = GridSearchCV_RF.best_params_
print(f"Best parameter values for RF model:\n{best_params_RF}")

```

```

Best parameter values for RF model:
{'criterion': 'gini', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':

```



```

best_score_RF = GridSearchCV_RF.best_score_
print(f"Best score for RF model: {round(best_score_RF, 3)}")

```

Best score for RF model: 0.942

```

random_forest = RandomForestClassifier(criterion="gini", n_estimators=100, min_samples_lea
random_forest_mod = random_forest.fit(X_train, y_train)
pred_random_forest = random_forest_mod.predict(X_test)

```

```

mse_random_forest = mean_squared_error(y_test, pred_random_forest)
rmse_random_forest = np.sqrt(mean_squared_error(y_test, pred_random_forest))
score_random_forest_train = random_forest_mod.score(X_train, y_train)
score_random_forest_test = random_forest_mod.score(X_test, y_test)

```

```

print(f"Mean Square Error for Random Forest = {round(mse_random_forest, 3)}")
print(f"Root Mean Square Error for Random Forest = {round(rmse_random_forest, 3)}")
print(f"R^2(coefficient of determination) on training set = {round(score_random_forest_tra
print(f"R^2(coefficient of determination) on testing set = {round(score_random_forest_test

```

```

Mean Square Error for Random Forest = 0.1
Root Mean Square Error for Random Forest = 0.317
R^2(coefficient of determination) on training set = 0.999
R^2(coefficient of determination) on testing set = 0.928

```

```

print("Classification Report")
print(classification_report(y_test, pred_random_forest))

```

```

Classification Report
precision    recall  f1-score   support

```

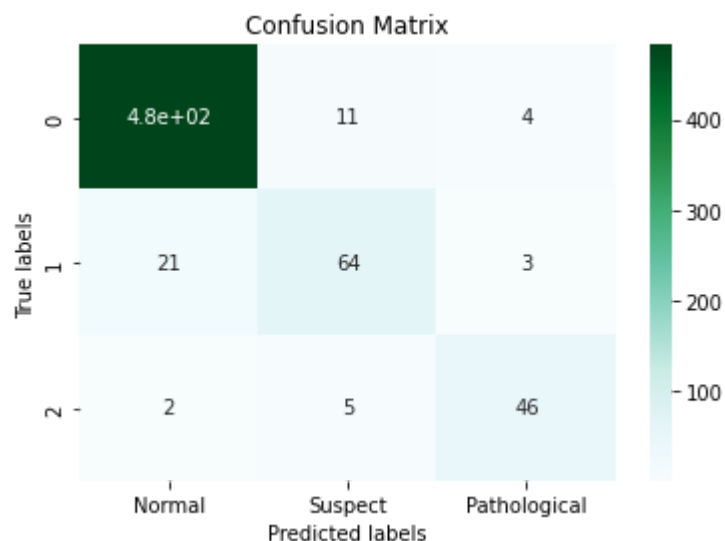
	1.0	0.95	0.97	0.96	497
	2.0	0.80	0.73	0.76	88
	3.0	0.87	0.87	0.87	53
accuracy				0.93	638
macro avg		0.87	0.86	0.86	638
weighted avg		0.93	0.93	0.93	638

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred_random_forest))
```

```
Confusion Matrix:
[[482  11   4]
 [ 21  64   3]
 [   2   5  46]]
```

```
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, pred_random_forest), annot=True, ax = ax, cmap = "BuG")

# labels, title and ticks
ax.set_xlabel("Predicted labels");
ax.set_ylabel("True labels");
ax.set_title("Confusion Matrix");
ax.xaxis.set_ticklabels(["Normal", "Suspect", "Pathological"]);
```



▼ Plotting the learning curve

- Learning curves are plots that show changes in learning performance over time in terms of experience.
- Learning curves of model performance on the train and validation datasets can be used to diagnose an underfit, overfit, or well-fit model.
- Learning curves of model performance can be used to diagnose whether the train or validation datasets are not relatively representative of the problem domain.

```

# Plot learning curve
def plot_learning_curve(estimator, title, x, y, ylim=None, cv=None,
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):

    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)

    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, x, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="#80CBC4",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="#00897B",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

```

▼ Result Visualisation of the learning curve

```

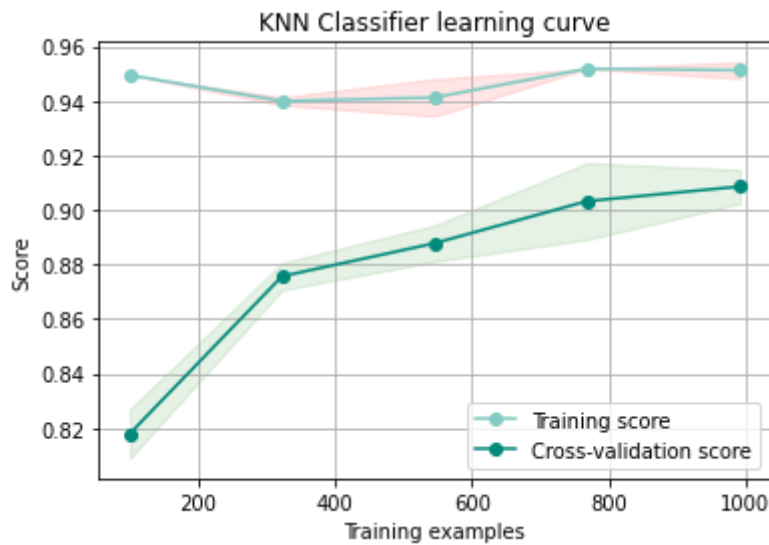
# Logistic Regression
plot_learning_curve(GridSearchCV_LR.best_estimator_, title = "Logistic Regression learning

```

Logistic Regression learning curve

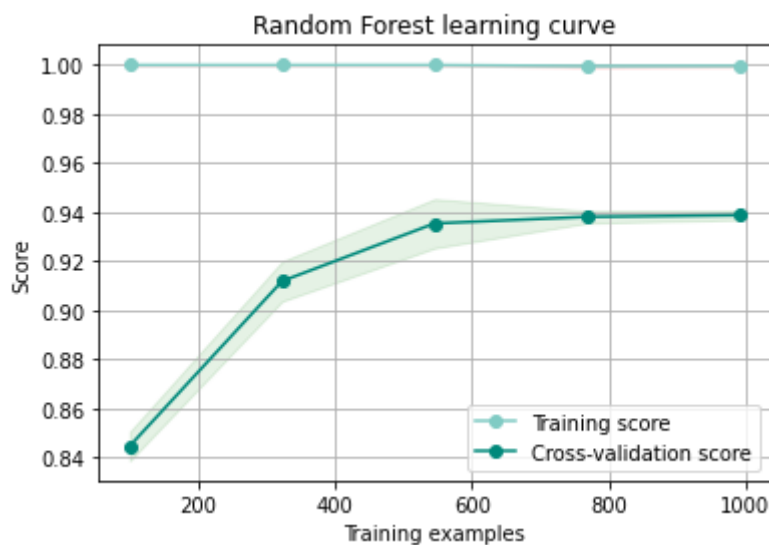
```
# knn
```

```
plot_learning_curve(GridSearchCV_knn.best_estimator_,title = "KNN Classifier learning curve"
```



```
# Random forest
```

```
plot_learning_curve(GridSearchCV_RF.best_estimator_,title = "Random Forest learning curve"
```



```
results = pd.DataFrame({
    "Model": ["Logistic Regression",
              "KNN",
              "Random Forest"],
    "Score": [logistic_regression_mod.score(X_train, y_train),
              knn_mod.score(X_train, y_train),
              random_forest_mod.score(X_train, y_train)]
})

result_df = results.sort_values(by="Score", ascending=False)
result_df = result_df.set_index("Score")
result_df.head(5)
```

Model	
Score	
0.999328	Random Forest
0.956317	KNN
0.906586	Logistic Regression

The results of the model selection phase are summarized in Table above. The Random Forest with 0.99 score has high percentage among models. Logistic Regression has lowest score (0.90).
