Fetal Health Classification

Import libraries to be used

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import cm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV, StratifiedKFold
```

Exploratory Data Analysis and Pre-Processing

In [ ]:

```
data = pd.read_csv('fetal_health.csv')
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2126 entries, 0 to 2125
Data columns (total 22 columns):
 #   Column                                                  Non-Null Count  Dtype
---  ------                                                  --------------  -----
 0   baseline value                                          2126 non-null   float64
 1   accelerations                                           2126 non-null   float64
 2   fetal_movement                                          2126 non-null   float64
 3   uterine_contractions                                    2126 non-null   float64
 4   light_decelerations                                     2126 non-null   float64
 5   severe_decelerations                                    2126 non-null   float64
 6   prolongued_decelerations                                2126 non-null   float64
 7   abnormal_short_term_variability                         2126 non-null   float64
 8   mean_value_of_short_term_variability                    2126 non-null   float64
 9   percentage_of_time_with_abnormal_long_term_variability  2126 non-null   float64
 10  mean_value_of_long_term_variability                     2126 non-null   float64
 11  histogram_width                                         2126 non-null   float64
 12  histogram_min                                           2126 non-null   float64
 13  histogram_max                                           2126 non-null   float64
 14  histogram_number_of_peaks                               2126 non-null   float64
 15  histogram_number_of_zeroes                              2126 non-null   float64
 16  histogram_mode                                          2126 non-null   float64
 17  histogram_mean                                          2126 non-null   float64
 18  histogram_median                                        2126 non-null   float64
 19  histogram_variance                                      2126 non-null   float64
 20  histogram_tendency                                      2126 non-null   float64
 21  fetal_health                                            2126 non-null   float64
dtypes: float64(22)
memory usage: 365.5 KB
```

Out[ ]:

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations	prolongued_deceler
0	120.0	0.000	0.0	0.000	0.000	0.0	
1	132.0	0.006	0.0	0.006	0.003	0.0	

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations	prolongued_deceler
1	132.0	0.006	0.0	0.006	0.003	0.0	
2	132.0	0.003	0.0	0.008	0.003	0.0	
3	134.0	0.003	0.0	0.008	0.003	0.0	
4	132.0	0.007	0.0	0.008	0.000	0.0	

Note that there are no null data values and all column Dtypes are floats. There are 21 columns which translates to 21 features to analyze. Let's now remove duplicate data entries.

In [ ]:

```
unduplicated_data = data.copy()
unduplicated_data.drop_duplicates(inplace=True)
print("Removed " + str(data.shape[0] - unduplicated_data.shape[0]) + " duplicates")
data = unduplicated_data
```

```
Removed 13 duplicates
```

In [ ]:

```
print("Total: " + str(data.shape[0]) + " samples")
```

```
Total: 2113 samples
```

In [ ]:

```
data.describe().T
```

Out[ ]:

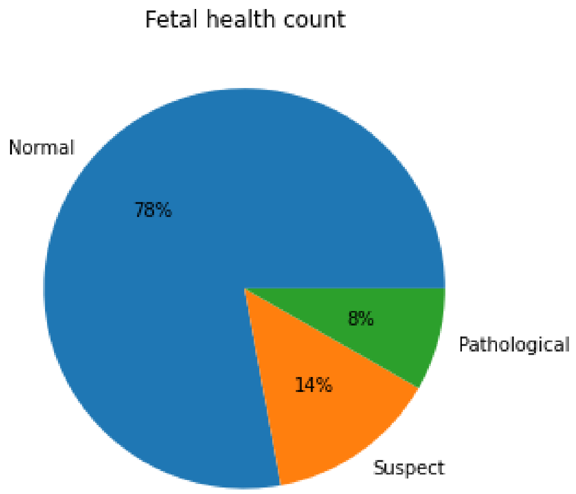
	count	mean	std	min	25%	50%	75%	m
baseline value	2113.0	133.304780	9.837451	106.0	126.000	133.000	140.000	160.0
accelerations	2113.0	0.003188	0.003871	0.0	0.000	0.002	0.006	0.0
fetal_movement	2113.0	0.009517	0.046804	0.0	0.000	0.000	0.003	0.4
uterine_contractions	2113.0	0.004387	0.002941	0.0	0.002	0.005	0.007	0.0
light_decelerations	2113.0	0.001901	0.002966	0.0	0.000	0.000	0.003	0.0
severe_decelerations	2113.0	0.000003	0.000057	0.0	0.000	0.000	0.000	0.0
prolongued_decelerations	2113.0	0.000159	0.000592	0.0	0.000	0.000	0.000	0.0
abnormal_short_term_variability	2113.0	46.993848	17.177782	12.0	32.000	49.000	61.000	87.0
mean_value_of_short_term_variability	2113.0	1.335021	0.884368	0.2	0.700	1.200	1.700	7.0
percentage_of_time_with_abnormal_long_term_variability	2113.0	9.795078	18.337073	0.0	0.000	0.000	11.000	91.0
mean_value_of_long_term_variability	2113.0	8.166635	5.632912	0.0	4.600	7.400	10.800	50.7
histogram_width	2113.0	70.535258	39.007706	3.0	37.000	68.000	100.000	180.0
histogram_min	2113.0	93.564600	29.562269	50.0	67.000	93.000	120.000	159.0
histogram_max	2113.0	164.099858	17.945175	122.0	152.000	162.000	174.000	238.0
histogram_number_of_peaks	2113.0	4.077142	2.951664	0.0	2.000	4.000	6.000	18.0
histogram_number_of_zeroes	2113.0	0.325603	0.707771	0.0	0.000	0.000	0.000	10.0
histogram_mode	2113.0	137.454330	16.402026	60.0	129.000	139.000	148.000	187.0
histogram_mean	2113.0	134.599621	15.610422	73.0	125.000	136.000	145.000	182.0
histogram_median	2113.0	138.089446	14.478957	77.0	129.000	139.000	148.000	186.0
histogram_variance	2113.0	18.907241	29.038766	0.0	2.000	7.000	24.000	269.0
histogram_tendency	2113.0	0.318504	0.611075	-1.0	0.000	0.000	1.000	1.0
fetal_health	2113.0	1.303833	0.614279	1.0	1.000	1.000	1.000	3.0

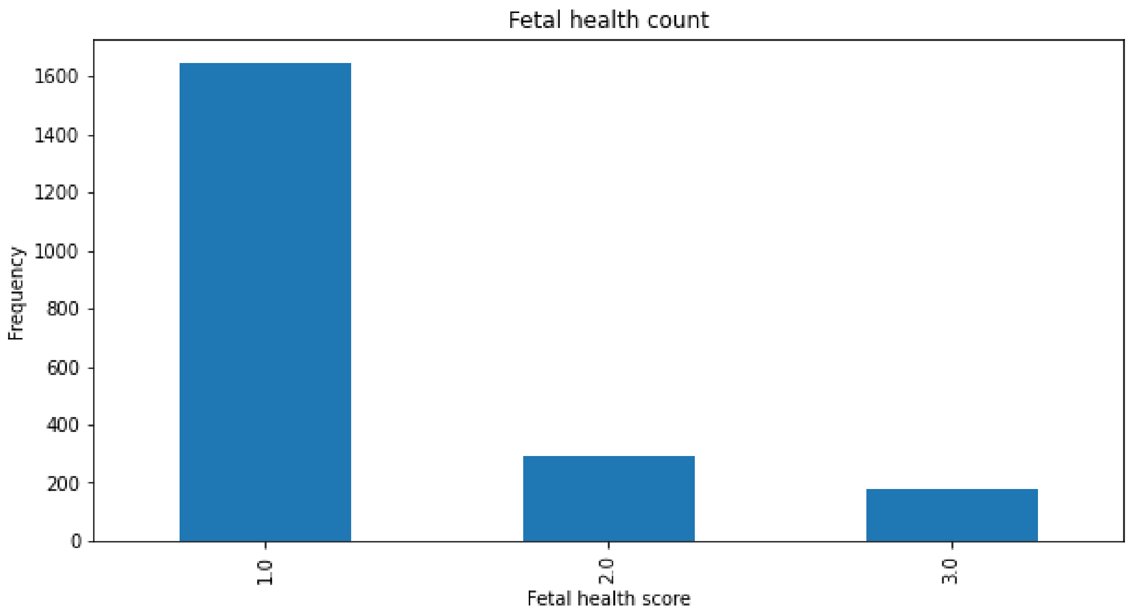
Visualizing Fetal Health Classification Raw Data

In [ ]:

```
normal = data.loc[data['fetal_health'] == 1].shape[0]
suspect = data.loc[data['fetal_health'] == 2].shape[0]
pathological = data.loc[data['fetal_health'] == 3].shape[0]
total = data.fetal_health.shape[0]

print("Total count: " + str(total))
print("Normal count: " + str(normal))
print("Suspect count: " + str(suspect))
print("Pathological count: " + str(pathological))

plt.figure(figsize = (10,5))
pie_fetal_health = plt.pie([normal, suspect, pathological], labels=["Normal", "Suspect",
"Pathological"], autopct="%1.0f%%")
plt.title("Fetal health count")

plt.figure(figsize = (10,5))
data['fetal_health'].value_counts().plot(figsize=(10, 5), kind="bar")
plt.title("Fetal health count")
plt.xlabel("Fetal health score")
plt.ylabel("Frequency")

plt.show()
```

```
Total count: 2113
Normal count: 1646
Suspect count: 292
Pathological count: 175
```



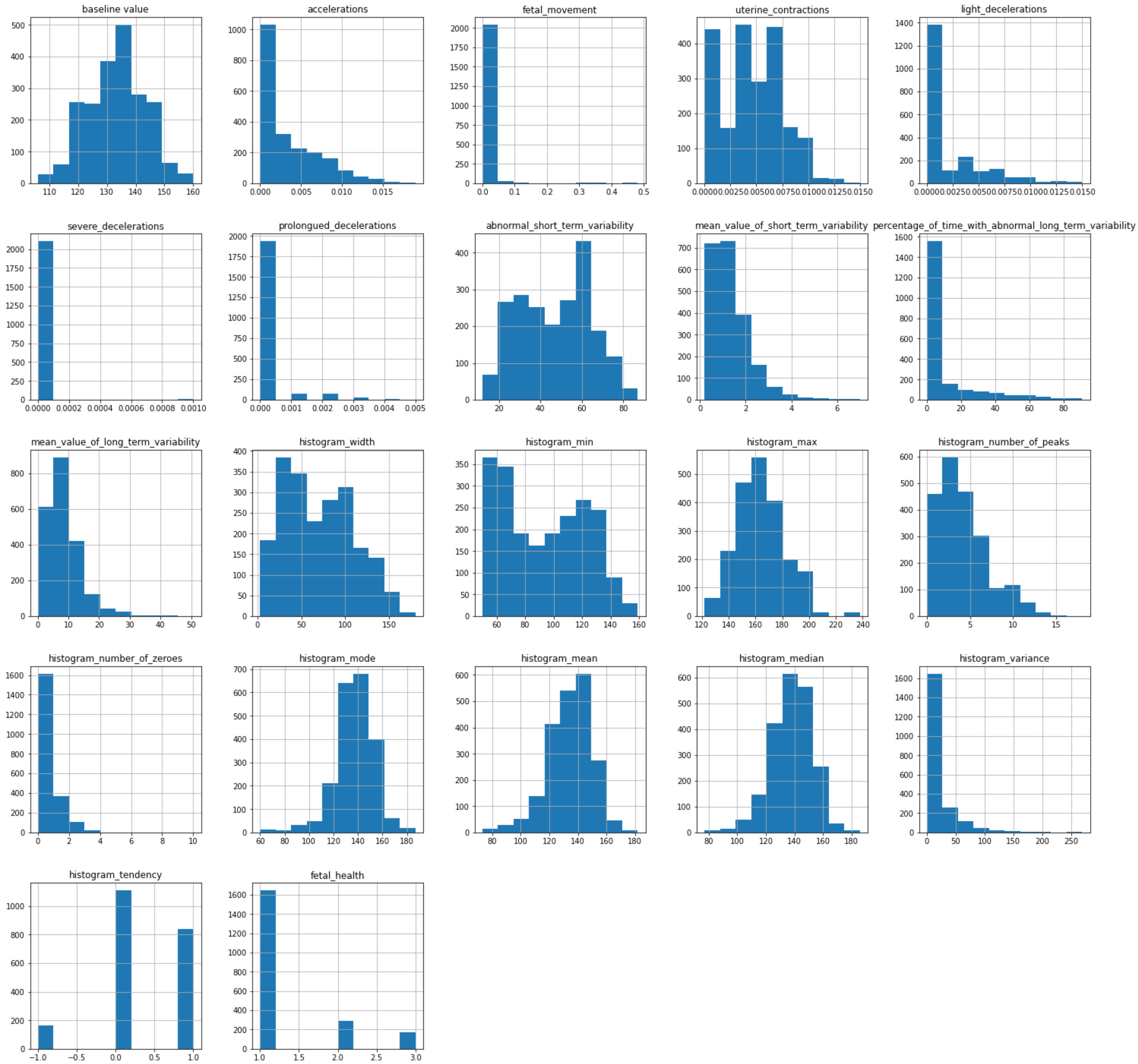


Note the imbalanced dataset in terms of fetal health scoring, leading to classification inaccuracy. To better observe feature importance and correlation, a confusion matrix will be assembled to observe correlation coefficients, giving us a better idea of what the most important features are.

Feature Analysis and Selection

In [ ]:

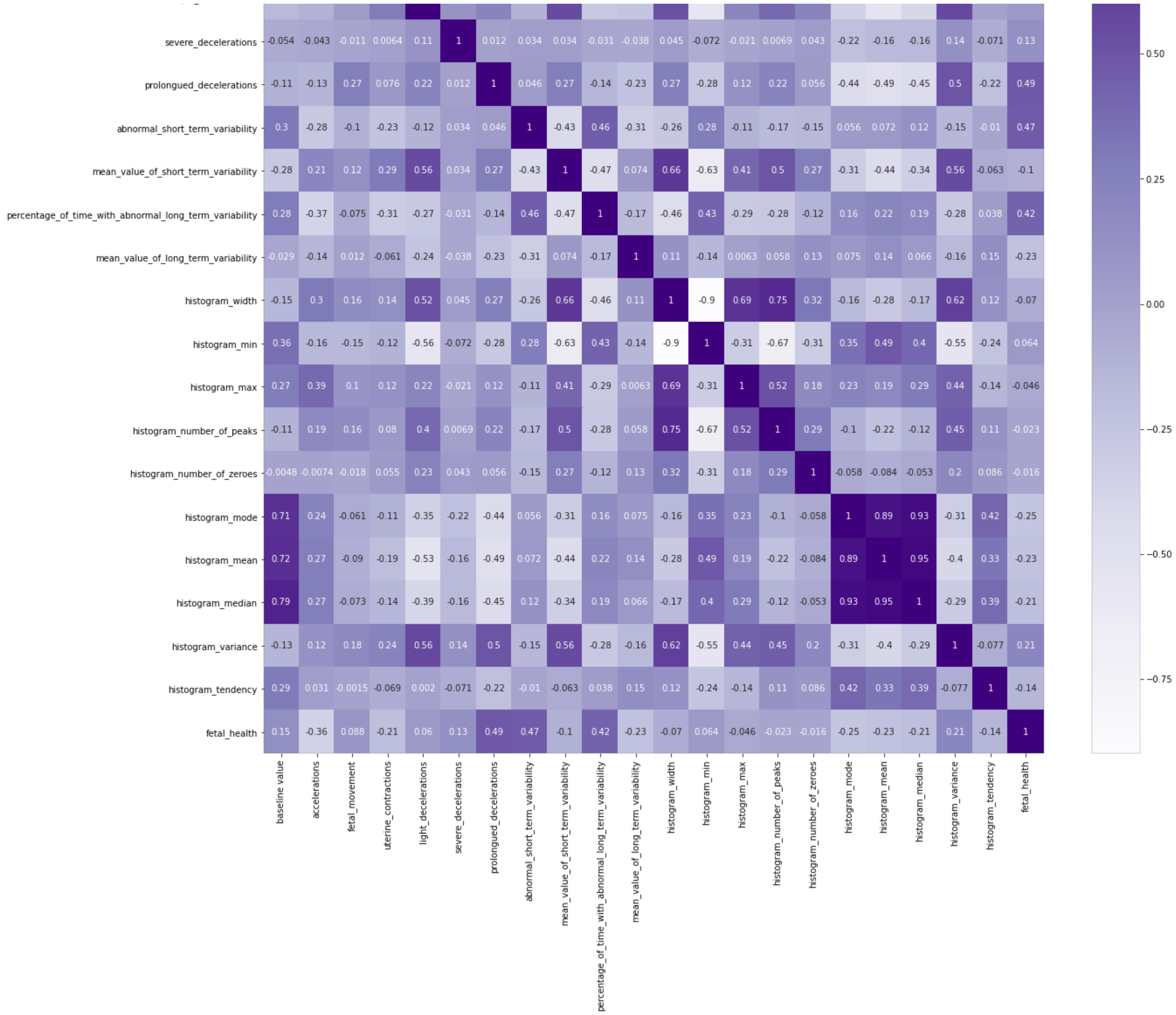
```
feature_hist_plot = data.hist(figsize = (25,25))
```



Confusion Matrix

In [ ]:

```
plt.figure(figsize=(20, 20))
correlation_matrix = sns.heatmap(data.corr(), annot=True, cmap="Purples")
```



In [ ]:

```
feature_correlation = data.corr()["fetal_health"].sort_values(ascending=False).to_frame()
feature_correlation.style.background_gradient(cmap=cm.Blues)
```

Out[ ]:

	fetal_health
fetal_health	1.000000
prolongued_decelerations	0.486752
abnormal_short_term_variability	0.469671
percentage_of_time_with_abnormal_long_term_variability	0.421634
histogram_variance	0.208171
baseline value	0.146077
severe_decelerations	0.132408
fetal_movement	0.088057
histogram_min	0.063529
light_decelerations	0.059651
histogram_number_of_zeroes	-0.016376
histogram_number_of_peaks	-0.022856
histogram_max	-0.046480
histogram_width	-0.069529

	fetal_health
mean_value_of_short_term_variability	-0.101089
histogram_tendency	-0.135573
uterine_contractions	-0.205117
histogram_median	-0.208334
mean_value_of_long_term_variability	-0.225685
histogram_mean	-0.230243
histogram_mode	-0.253612
accelerations	-0.363947

Note that the row we are focused on is that of "fetal_health", where we can clearly see which features are most correlated with fetal_health. Of the 21 features, the features with highest correlation is "prolongued_decelerations, "abnormal_short_term_variability", and "percentage_of_time_with_abnormal_long_term_variability".

Now that we have our correlation coefficients, let's select the most important features with the KBest algorithm.

In [ ]:

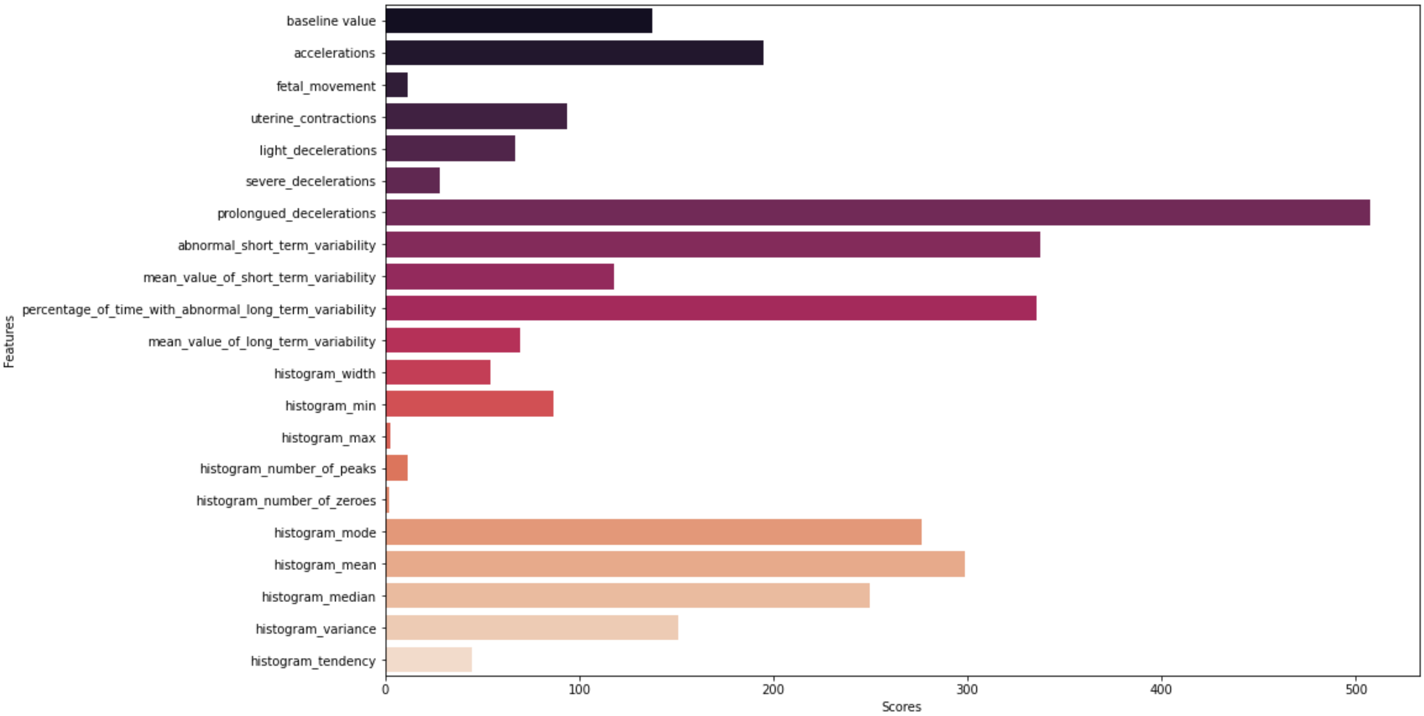
```
X = data.drop(["fetal_health"], axis=1)
Y = data["fetal_health"]
selected_features = SelectKBest(score_func=f_classif, k='all')

feature_fit = selected_features.fit(X, Y)
feature_scores = pd.DataFrame({"Features": X.columns,
                               "Scores": feature_fit.scores_})

feature_scores
```

Out[ ]:

	Features	Scores
0	baseline value	137.833999
1	accelerations	194.618345
2	fetal_movement	11.700712
3	uterine_contractions	93.647474
4	light_decelerations	66.750344
5	severe_decelerations	28.438837
6	prolongued_decelerations	507.304309
7	abnormal_short_term_variability	337.703020
8	mean_value_of_short_term_variability	118.050463
9	percentage_of_time_with_abnormal_long_term_var...	335.386156
10	mean_value_of_long_term_variability	69.418940
11	histogram_width	54.215605
12	histogram_min	86.468440
13	histogram_max	2.523350
14	histogram_number_of_peaks	11.726828
15	histogram_number_of_zeroes	2.134901
16	histogram_mode	276.382795
17	histogram_mean	298.759569
18	histogram_median	249.699523
19	histogram_variance	150.955827
20	histogram_tendency	44.854186

In [ ]:

```python
plt.figure(figsize = (15,10))
plot = sns.barplot(data=feature_scores, x='Scores', y='Features', palette="rocket")
```



Let's now select the features with scores above the 100 threshold as our most important features and the ones that we will train our models on.

In [ ]:

```python
best_features = feature_scores[feature_scores['Scores'] > 100]
best_features
```

Out[ ]:

	Features	Scores
0	baseline value	137.833999
1	accelerations	194.618345
6	prolongued_decelerations	507.304309
7	abnormal_short_term_variability	337.703020
8	mean_value_of_short_term_variability	118.050463
9	percentage_of_time_with_abnormal_long_term_var...	335.386156
16	histogram_mode	276.382795
17	histogram_mean	298.759569
18	histogram_median	249.699523
19	histogram_variance	150.955827

In [ ]:

```python
best_features_array = list(best_features['Features'])
best_features_array.append('fetal_health')
data = data[best_features_array]
data.head()
```

Out[ ]:

	baseline value	accelerations	prolongued_decelerations	abnormal_short_term_variability	mean_value_of_short_term_variability	per

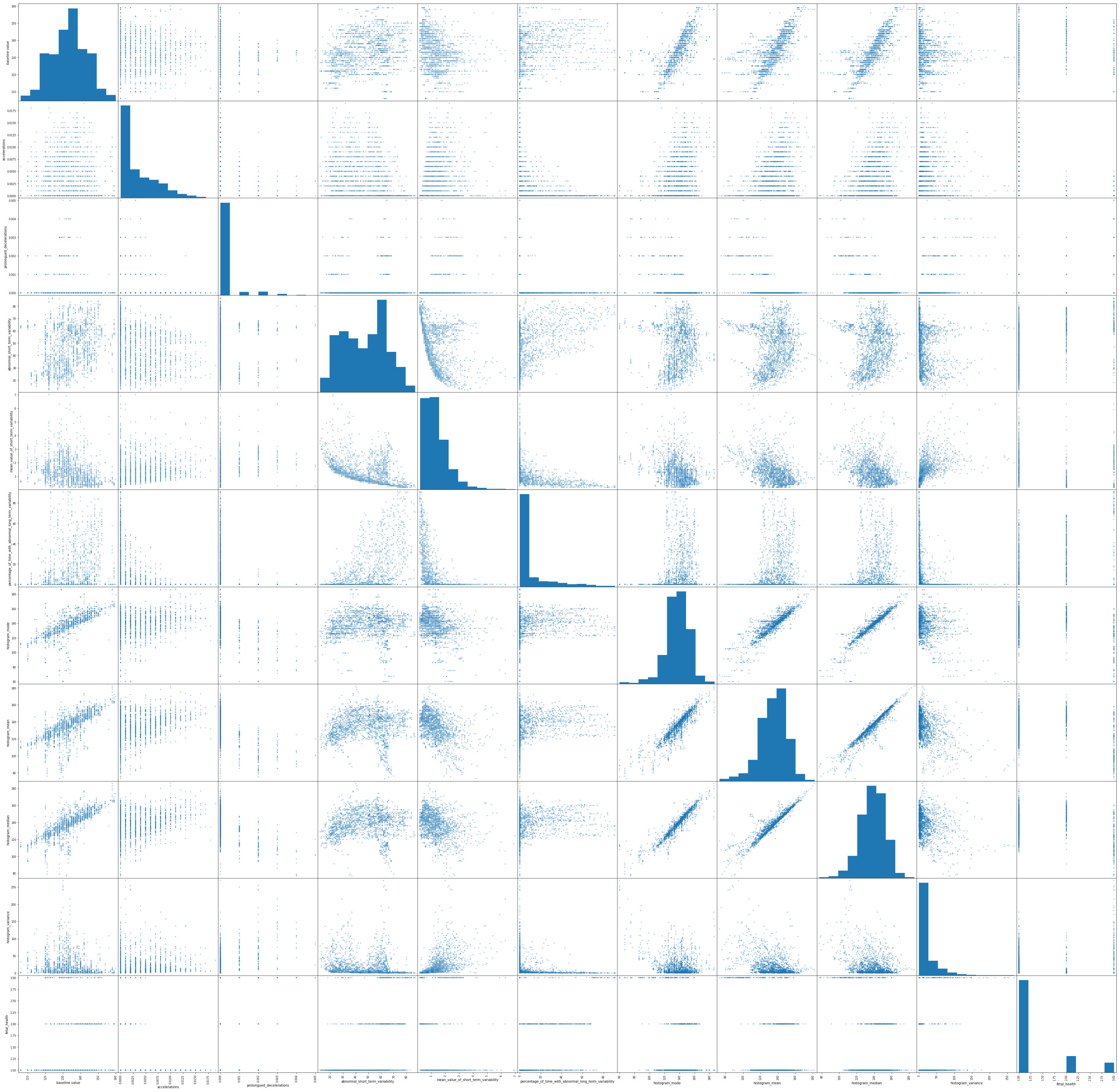
	baseline value	accelerations	prolongued_decelerations	abnormal_short_term_variability	mean_value_of_short_term_variability	per
0	120.0	0.000	0.0	73.0	0.5	
1	132.0	0.006	0.0	17.0	2.1	
2	133.0	0.003	0.0	16.0	2.1	
3	134.0	0.003	0.0	16.0	2.4	
4	132.0	0.007	0.0	16.0	2.4	

Scatter Matrix

Visualizes the relationships between the most important features

In [ ]:

```
from pandas.plotting import scatter_matrix
matrix = scatter_matrix(data, figsize=(60, 60))
```



Dataset split (train/test)

In order to use a model, we must first scale the array of features so that they are in the same range. We will also be using the standard 70/30 train/test split.

In [ ]:

```
scaler = StandardScaler()
X=data.drop(["fetal_health"], axis=1)
X_df = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2113 entries, 0 to 2112
Data columns (total 10 columns):
 #   Column                                                   Non-Null Count  Dtype  
---  ------                                                   --------------  -----  
 0   baseline value                                           2113 non-null   float64
 1   accelerations                                            2113 non-null   float64
 2   prolongued_decelerations                                 2113 non-null   float64
 3   abnormal_short_term_variability                          2113 non-null   float64
 4   mean_value_of_short_term_variability                     2113 non-null   float64
 5   percentage_of_time_with_abnormal_long_term_variability  2113 non-null   float64
 6   histogram_mode                                           2113 non-null   float64
 7   histogram_mean                                           2113 non-null   float64
 8   histogram_median                                         2113 non-null   float64
 9   histogram_variance                                       2113 non-null   float64
dtypes: float64(10)
memory usage: 165.2 KB
```

In [ ]:

```
X_df.head()
```

Out[ ]:

	baseline value	accelerations	prolongued_decelerations	abnormal_short_term_variability	mean_value_of_short_term_variability	pe
0	-1.352782	-0.823776	-0.26964	1.514300	-0.944425	
1	-0.132665	0.726444	-0.26964	-1.746497	0.865205	
2	-0.030989	-0.048666	-0.26964	-1.804726	0.865205	
3	0.070687	-0.048666	-0.26964	-1.804726	1.204511	
4	-0.132665	0.984814	-0.26964	-1.804726	1.204511	

In [ ]:

```
X_df.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%
baseline value	2113.0	-5.880031e-16	1.000237	-2.776252	-0.742724	-0.030989	0.680746
accelerations	2113.0	9.678328e-17	1.000237	-0.823776	-0.823776	-0.307036	0.726444
prolongued_decelerations	2113.0	2.528818e-15	1.000237	-0.269640	-0.269640	-0.269640	-0.269640
abnormal_short_term_variability	2113.0	6.567812e-17	1.000237	-2.037640	-0.873069	0.116815	0.815557
mean_value_of_short_term_variability	2113.0	-3.913365e-	1.000237	-	-	-	0.412798

	count	mean	std	min	25%	50%	75%
mean_value_of_short_term_variability	[illegible]	[illegible]e-16	[illegible]	1.283731	0.718221	0.152712	[illegible]
percentage_of_time_with_abnormal_long_term_variability	2113.0	2.212985e-15	1.000237	-0.534294	-0.534294	-0.534294	0.065725
histogram_mode	2113.0	-3.792255e-16	1.000237	-4.723360	-0.515566	0.094259	0.643101
histogram_mean	2113.0	7.585035e-16	1.000237	-3.946992	-0.615095	0.089729	0.666404
histogram_median	2113.0	9.018920e-16	1.000237	-4.220187	-0.627918	0.062903	0.684642
histogram_variance	2113.0	-3.865026e-16	1.000237	-0.651258	-0.582368	-0.410143	0.175419

In [ ]:

```
y=data["fetal_health"]
X_train, X_test, y_train, y_test = train_test_split(X_df, y, test_size=0.3, random_state
=42)
print("X_train samples: " + str(X_train.shape[0]) + " with " + str(X_train.shape[1]) + "
features")
print("Y_train samples: " + str(y_train.shape[0]))
print("X_test samples: " + str(X_test.shape[0]) + " with " + str(X_test.shape[1]) + " fe
atures")
print("Y_test samples: " + str(y_test.shape[0]))
```

```
X_train samples: 1479 with 10 features
Y_train samples: 1479
X_test samples: 634 with 10 features
Y_test samples: 634
```

Machine Learning Classification

Here we will be using several classifiers to compare accuracy scores - we will be settling and fine-tuning the best classifier after comparing effectiveness.

Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression
```

In [ ]:

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
log_reg_score = log_reg.score(X_test, y_test)
print("Baseline LR Score: " + str(log_reg_score))
```

```
Baseline LR Score: 0.8958990536277602
```

In [ ]:

```
cv_score_lr = cross_val_score(log_reg, X_train, y_train)

print("Baseline LR CV Score: " + str(cv_score_lr.mean()))
```

```
Baseline LR CV Score: 0.8823751717819514
```

In [ ]:

```
lr_predictions = log_reg.predict(X_test)
print(classification_report(y_test, lr_predictions))
```

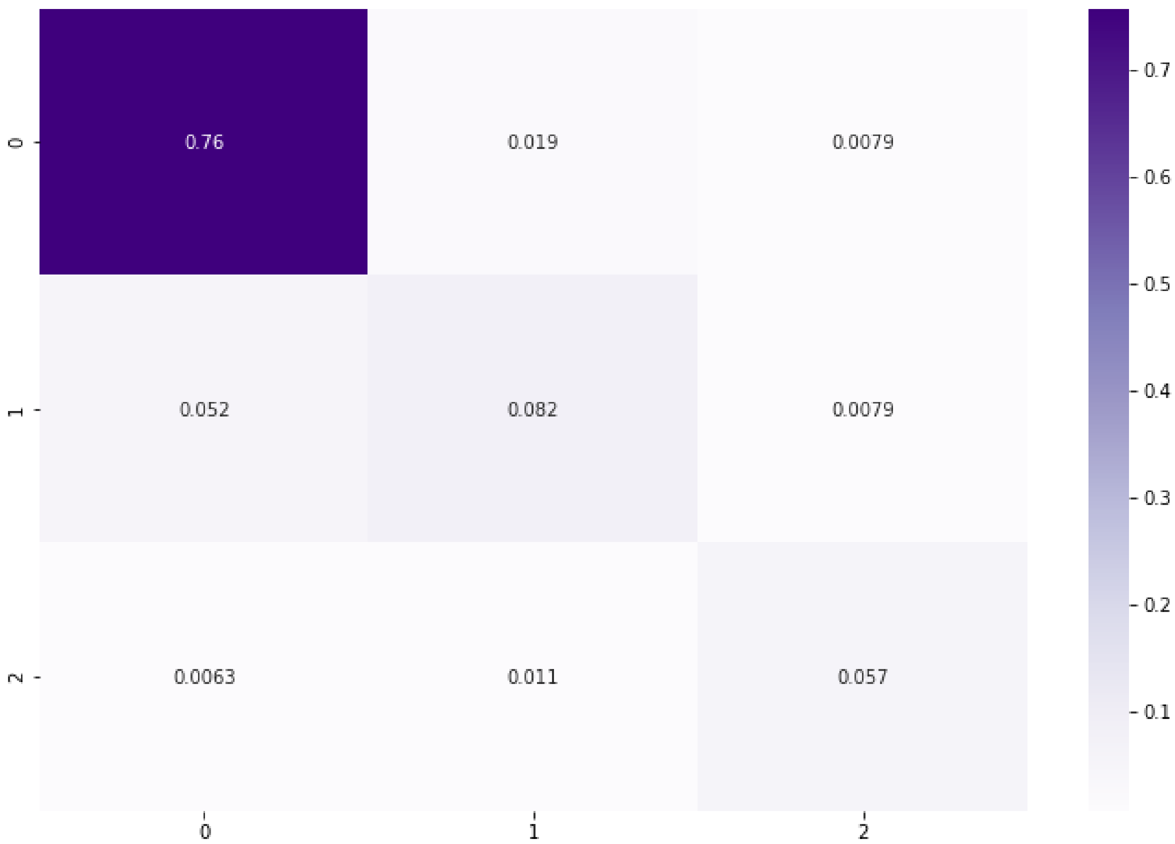
```
              precision    recall  f1-score   support

         1.0       0.93      0.97      0.95       497
         2.0       0.73      0.58      0.65        90
         3.0       0.78      0.77      0.77        47

    accuracy                           0.90       634
   macro avg       0.81      0.77      0.79       634
weighted avg       0.89      0.90      0.89       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
confusion_matrix_lr = confusion_matrix(y_test, lr_predictions)
sns.heatmap(confusion_matrix_lr/np.sum(confusion_matrix_lr),annot = True, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4af10abad0>
```



Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
random_forest_score = random_forest.score(X_test, y_test)
print("Baseline Random Forest Score: " + str(random_forest_score))
```

```
Baseline Random Forest Score: 0.9495268138801262
```

In [ ]:

```
cv_score_rf = cross_val_score(random_forest, X_train, y_train)

print("Baseline RF CV Score: " + str(cv_score_rf.mean()))
```

```
Baseline RF CV Score: 0.9323797526339899
```

In [ ]:

```python
rf_predictions = random_forest.predict(X_test)
print(classification_report(y_test, rf_predictions))
```

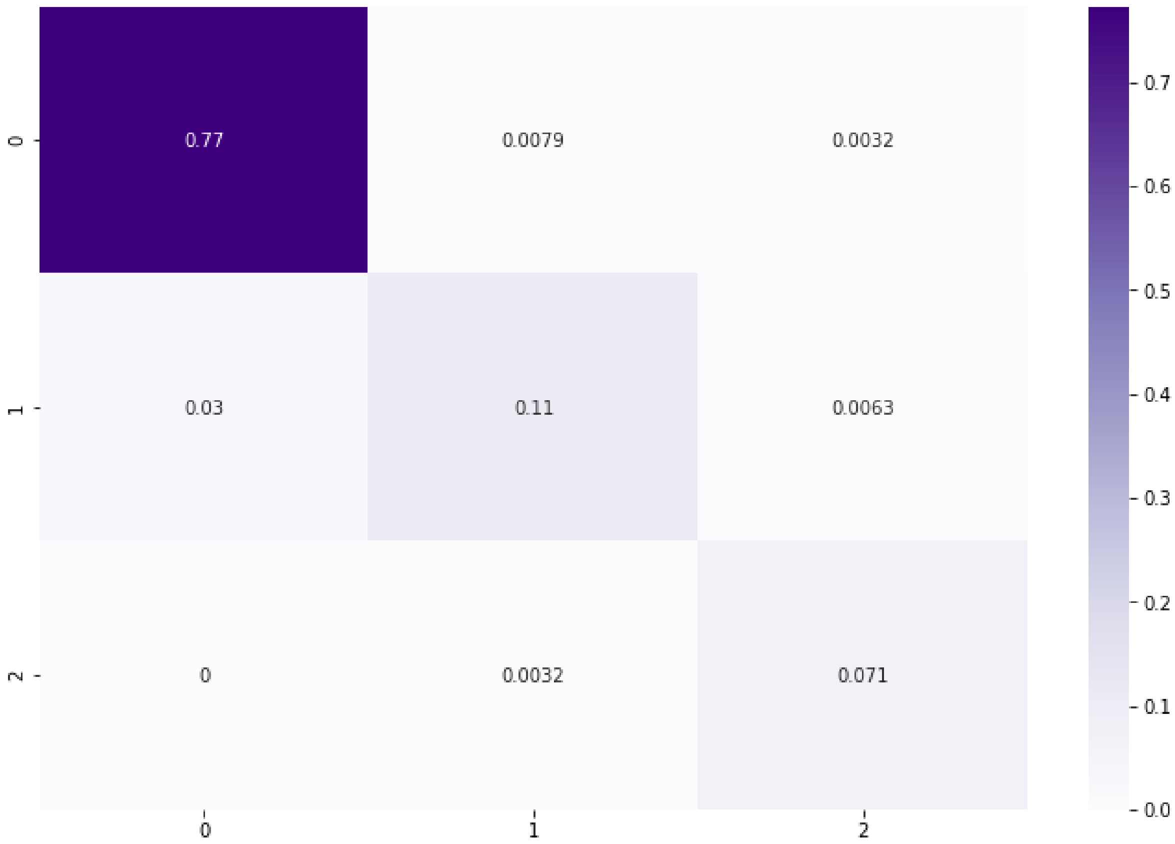
```
              precision    recall  f1-score   support

         1.0       0.96      0.99      0.97       497
         2.0       0.91      0.74      0.82        90
         3.0       0.88      0.96      0.92        47

    accuracy                           0.95       634
   macro avg       0.92      0.90      0.90       634
weighted avg       0.95      0.95      0.95       634
```

In [ ]:

```python
plt.subplots(figsize=(12,8))
confusion_matrix_rf = confusion_matrix(y_test, rf_predictions)
sns.heatmap(confusion_matrix_rf/np.sum(confusion_matrix_rf),annot = True, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae0b95f90>
```



K-Nearest Neighbors

In [ ]:

```python
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:

```python
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_score = knn.score(X_test, y_test)
print("Baseline KNN Score: " + str(knn_score))
```

```
Baseline KNN Score: 0.9400630914826499
```

In [ ]:

```
cv_score_knn = cross_val_score(knn, X_train, y_train)

print("Baseline KNN CV Score: " + str(cv_score_knn.mean()))
```

```
Baseline KNN CV Score: 0.9060215300045809
```

In [ ]:

```
knn_predictions = knn.predict(X_test)
print(classification_report(y_test, knn_predictions))
```

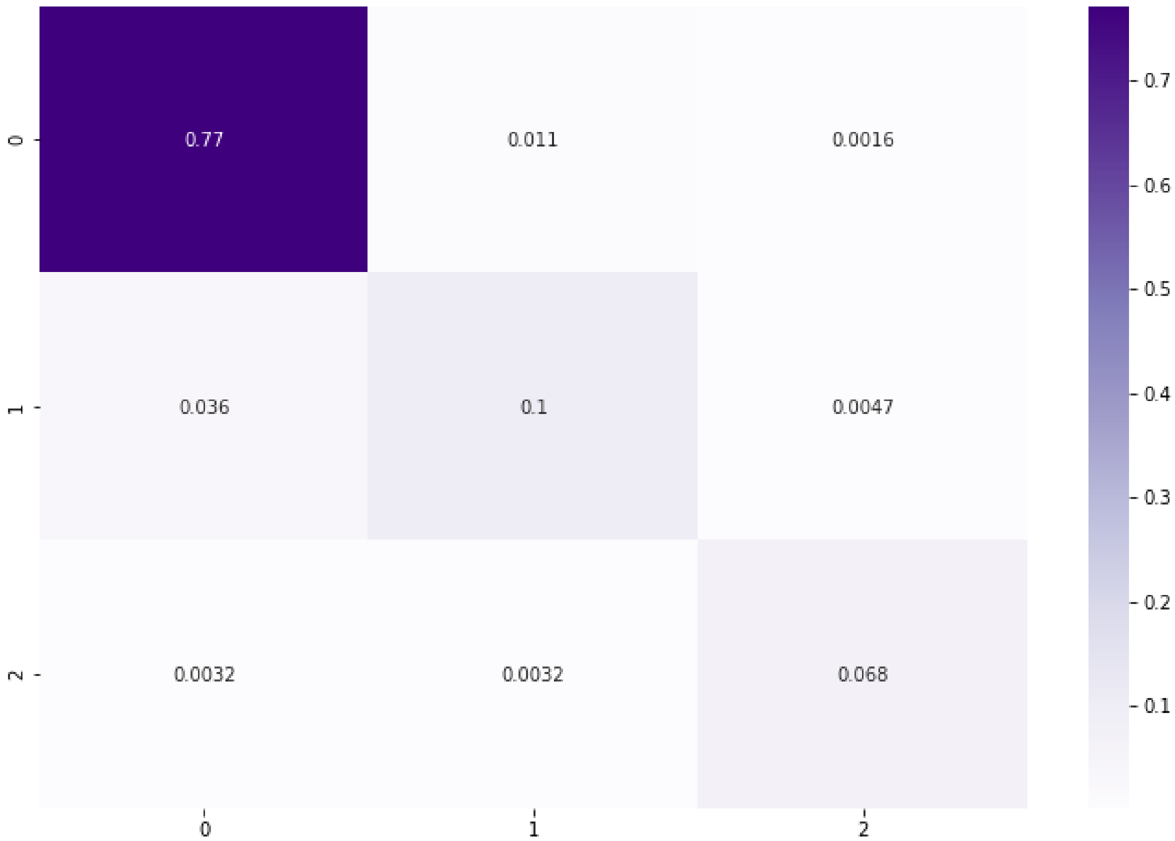
```
              precision    recall  f1-score   support

         1.0       0.95      0.98      0.97       497
         2.0       0.88      0.71      0.79        90
         3.0       0.91      0.91      0.91        47

    accuracy                           0.94       634
   macro avg       0.91      0.87      0.89       634
weighted avg       0.94      0.94      0.94       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
confusion_matrix_knn = confusion_matrix(y_test, knn_predictions)
sns.heatmap(confusion_matrix_knn/np.sum(confusion_matrix_knn),annot = True, cmap="Purples
")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae0983690>
```



Support Vector Classifier

In [ ]:

```
from sklearn.svm import SVC
```

In [ ]:

```
svc = SVC()
svc.fit(X_train, y_train)
svc_score = svc.score(X_test, y_test)
print("Baseline SVC Score: " + str(svc_score))
```

```
Baseline SVC Score: 0.9211356466876972
```

In [ ]:

```
vc_score_svc = cross_val_score(svc, X_train, y_train)

print("Baseline SVC CV Score: " + str(vc_score_svc.mean()))
```

```
Baseline SVC CV Score: 0.8924965643609711
```

In [ ]:

```
svc_predictions = svc.predict(X_test)
print(classification_report(y_test, svc_predictions))
```

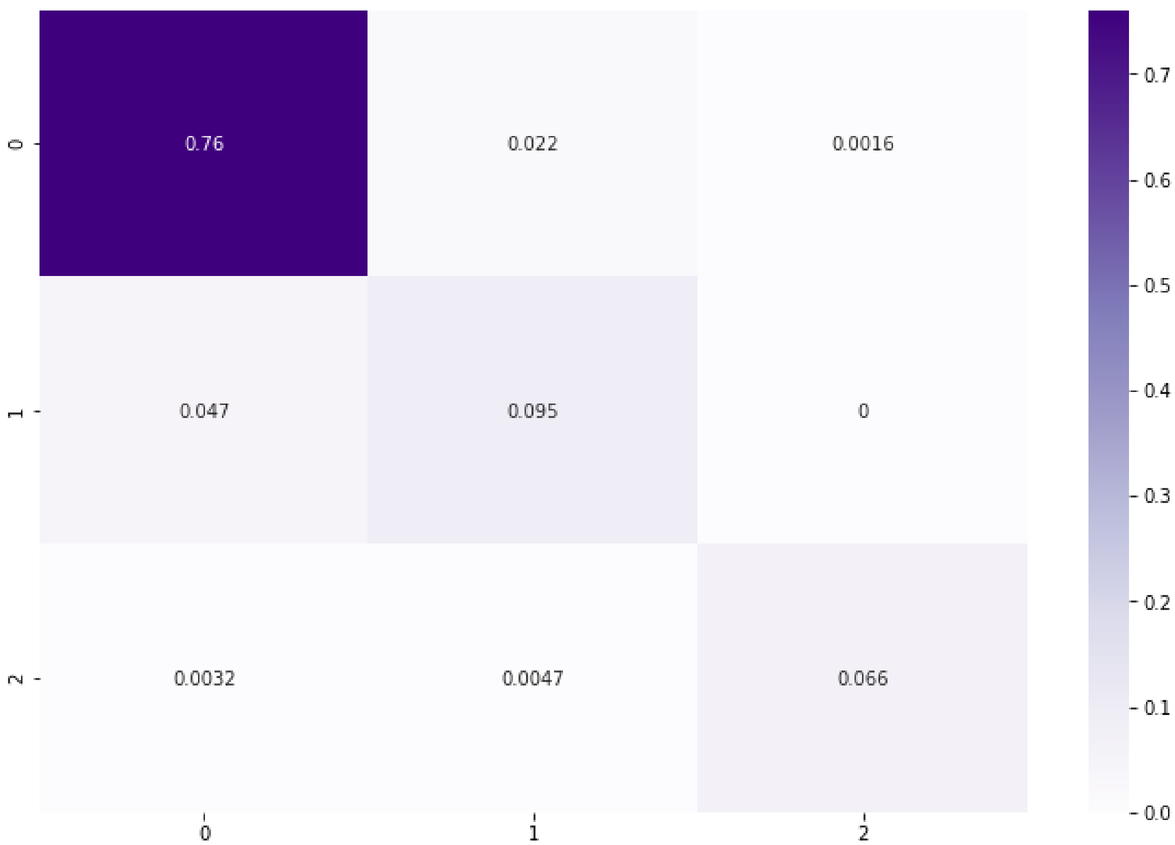
```
              precision    recall  f1-score   support

         1.0       0.94      0.97      0.95       497
         2.0       0.78      0.67      0.72        90
         3.0       0.98      0.89      0.93        47

    accuracy                           0.92       634
   macro avg       0.90      0.84      0.87       634
weighted avg       0.92      0.92      0.92       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
confusion_matrix_svc = confusion_matrix(y_test, svc_predictions)
sns.heatmap(confusion_matrix_svc/np.sum(confusion_matrix_svc),annot = True, cmap="Purples
")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae08af050>
```



Linear Support Vector Classifier

In [ ]:

```
from sklearn.svm import LinearSVC
```

In [ ]:

```
linear_svc = LinearSVC()
linear_svc.fit(X_train, y_train)
linear_svc_score = linear_svc.score(X_test, y_test)
print("Baseline Linear SVC Score: " + str(linear_svc_score))
```

```
Baseline Linear SVC Score: 0.8927444794952681
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [ ]:

```
cv_score_linear_svc = cross_val_score(linear_svc, X_train, y_train)

print("Baseline Linear SVC CV Score: " + str(cv_score_linear_svc.mean()))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Baseline Linear SVC CV Score: 0.8843999083829592
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [ ]:

```
linear_svc_predictions = linear_svc.predict(X_test)
print(classification_report(y_test, linear_svc_predictions))
```

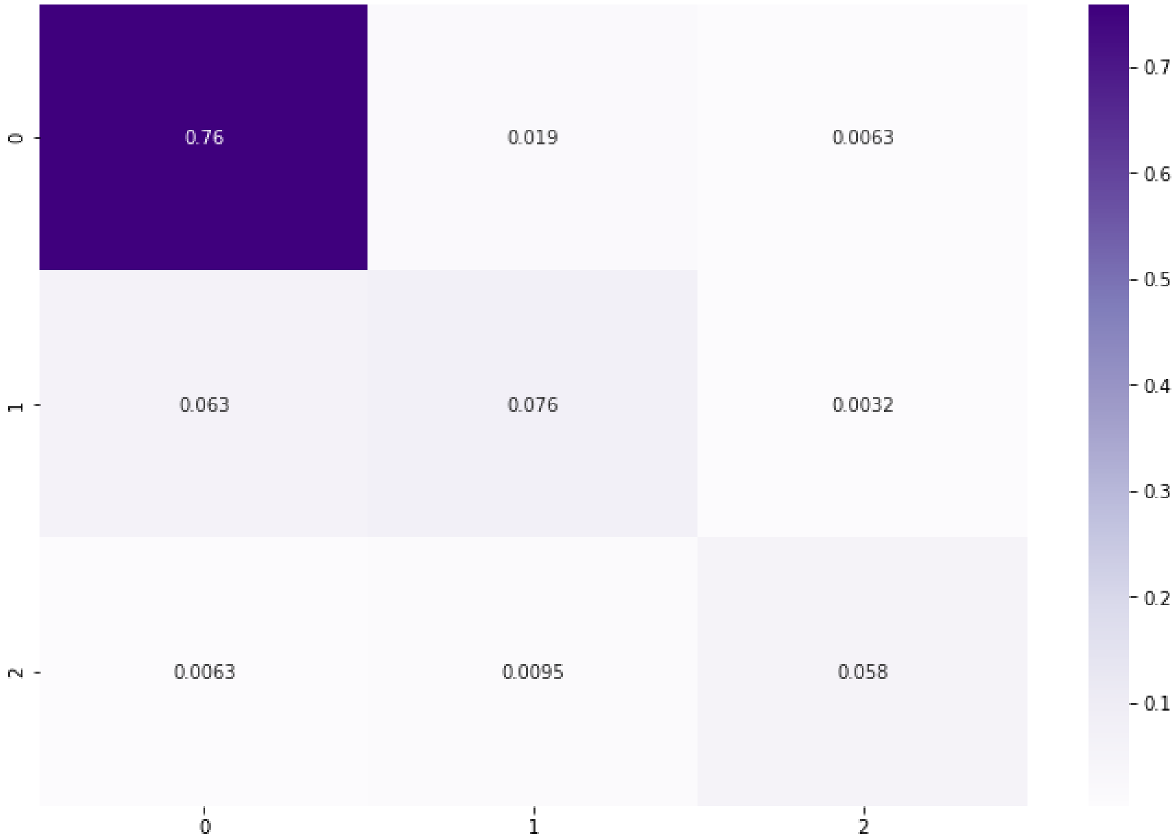
```
              precision    recall  f1-score   support

         1.0       0.92      0.97      0.94       497
         2.0       0.73      0.53      0.62        90
         3.0       0.86      0.79      0.82        47

    accuracy                           0.89       634
   macro avg       0.83      0.76      0.79       634
weighted avg       0.89      0.89      0.89       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
confusion_matrix_linear_svc = confusion_matrix(y_test, linear_svc_predictions)
sns.heatmap(confusion_matrix_linear_svc/np.sum(confusion_matrix_linear_svc),annot = True
, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae07ec2d0>
```



Decision Tree

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
decision_tree_score = decision_tree.score(X_test, y_test)
print("Baseline Decision Tree Score: " + str(decision_tree_score))
```

```
Baseline Decision Tree Score: 0.9337539432176656
```

In [ ]:

```
cv_score_decision_tree = cross_val_score(decision_tree, X_train, y_train)

print("Baseline Decision Tree CV Score: " + str(cv_score_decision_tree.mean()))
```

```
Baseline Decision Tree CV Score: 0.9073522675217591
```

In [ ]:

```
decision_tree_predictions = decision_tree.predict(X_test)
print(classification_report(y_test, decision_tree_predictions))
```

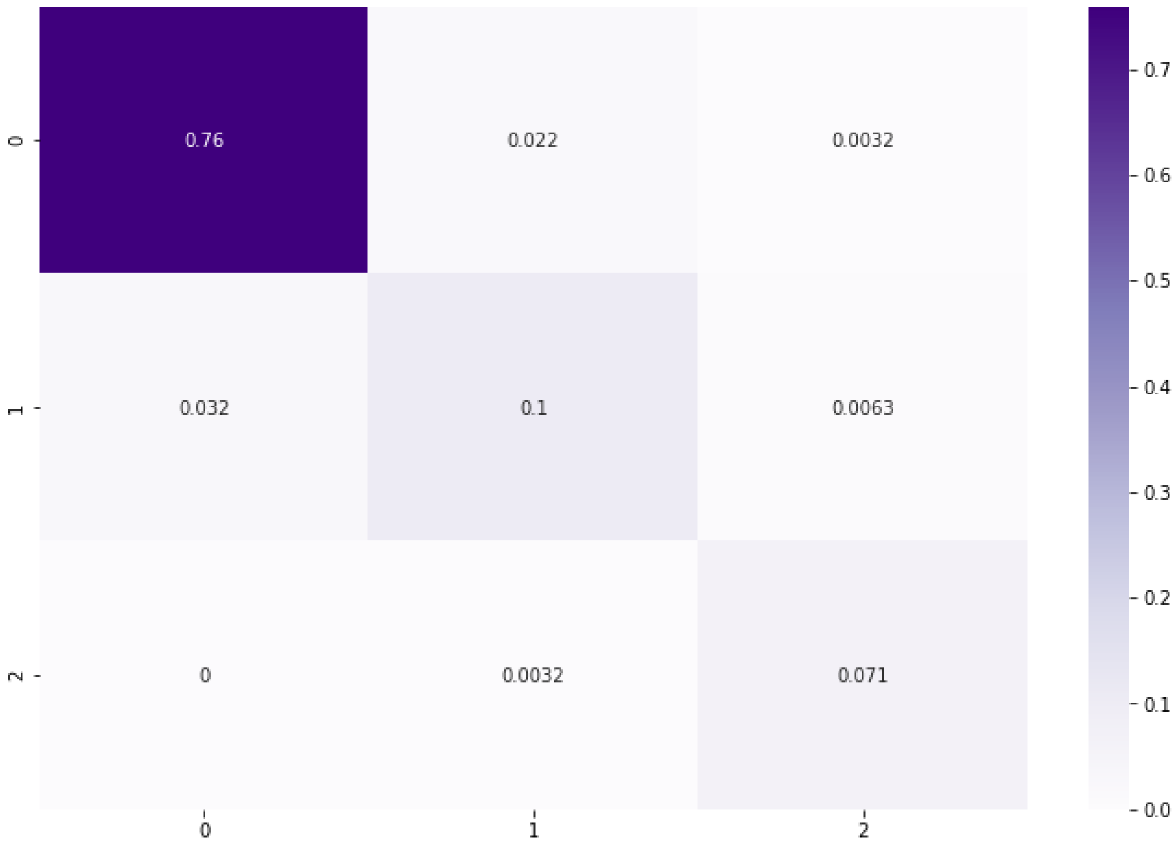
```
              precision    recall  f1-score   support

         1.0       0.96      0.97      0.96       497
         2.0       0.80      0.73      0.77        90
         3.0       0.88      0.96      0.92        47

    accuracy                           0.93       634
   macro avg       0.88      0.89      0.88       634
weighted avg       0.93      0.93      0.93       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
confusion_matrix_decision_tree = confusion_matrix(y_test, decision_tree_predictions)
sns.heatmap(confusion_matrix_decision_tree/np.sum(confusion_matrix_decision_tree),annot =
True, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae0719b90>
```



Multi-layer Perceptron (Neural Network)

In [ ]:

```
from sklearn.neural_network import MLPClassifier
```

In [ ]:

```
neural_network = MLPClassifier()
neural_network.fit(X_train, y_train)
neural_network_score = neural_network.score(X_test, y_test)
print("Baseline Multi-layer Perceptron Score: " + str(neural_network_score))
```

```
Baseline Multi-layer Perceptron Score: 0.9353312302839116
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [ ]:

```
cv_score_neural_network = cross_val_score(neural_network, X_train, y_train)

print("Baseline Neural Network CV Score: " + str(cv_score_neural_network.mean()))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
```

```
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Baseline Neural Network CV Score: 0.9107627118644068
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [ ]:

```
neural_network_predictions = neural_network.predict(X_test)
print(classification_report(y_test, neural_network_predictions))
```

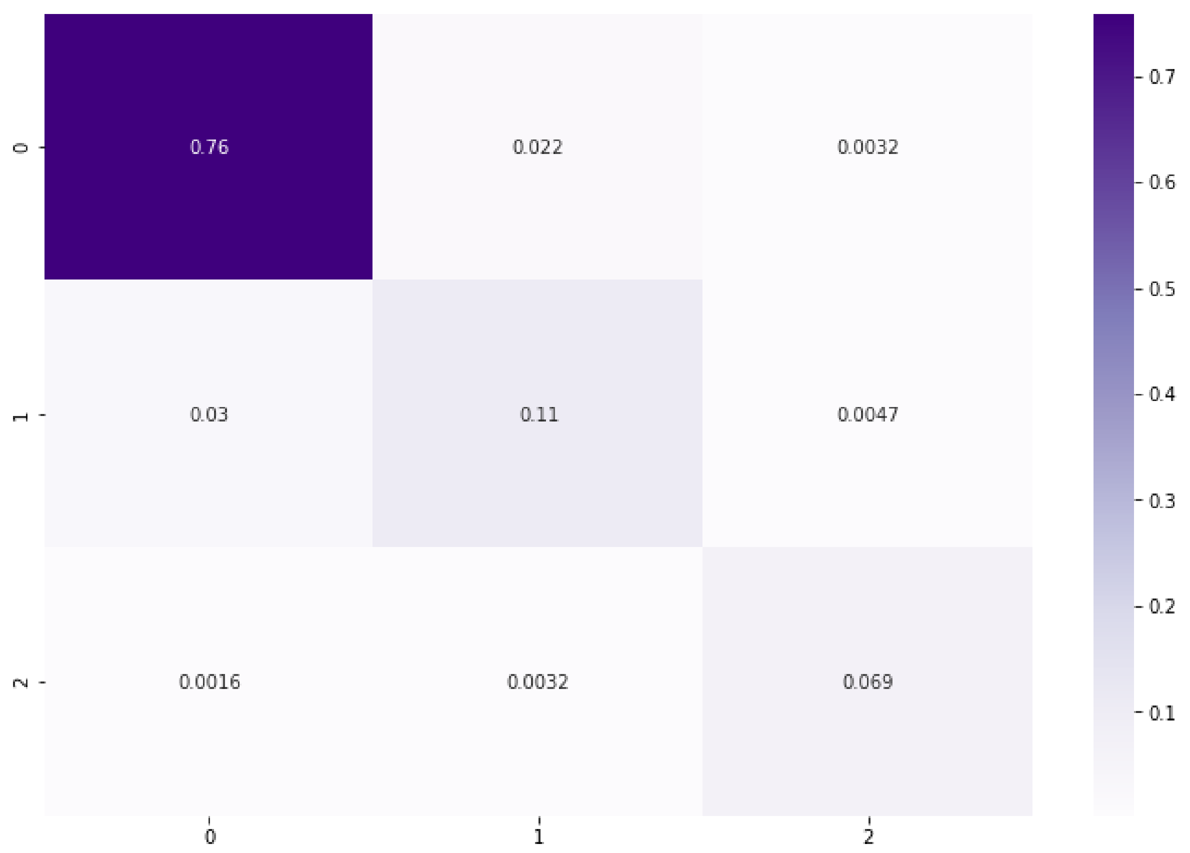
```
              precision    recall  f1-score   support

         1.0       0.96      0.97      0.96       497
         2.0       0.81      0.76      0.78        90
         3.0       0.90      0.94      0.92        47

    accuracy                           0.94       634
   macro avg       0.89      0.89      0.89       634
weighted avg       0.93      0.94      0.93       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
neural_network_decision_tree = confusion_matrix(y_test, neural_network_predictions)
sns.heatmap(neural_network_decision_tree/np.sum(neural_network_decision_tree),annot = Tru
e, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae065c310>
```



Gradient Boosting

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:

```
gradient_boosting = GradientBoostingClassifier()
gradient_boosting.fit(X_train, y_train)
gradient_boosting_score = gradient_boosting.score(X_test, y_test)
print("Baseline Gradient Boosting Classifier Score: " + str(gradient_boosting_score))
```

```
Baseline Gradient Boosting Classifier Score: 0.9558359621451105
```

In [ ]:

```
cv_score_gb = cross_val_score(gradient_boosting, X_train, y_train)

print("Baseline GB CV Score: " + str(cv_score_gb.mean()))
```

```
Baseline GB CV Score: 0.9371117727897389
```

In [ ]:

```
gb_predictions = gradient_boosting.predict(X_test)
print(classification_report(y_test, gb_predictions))
```

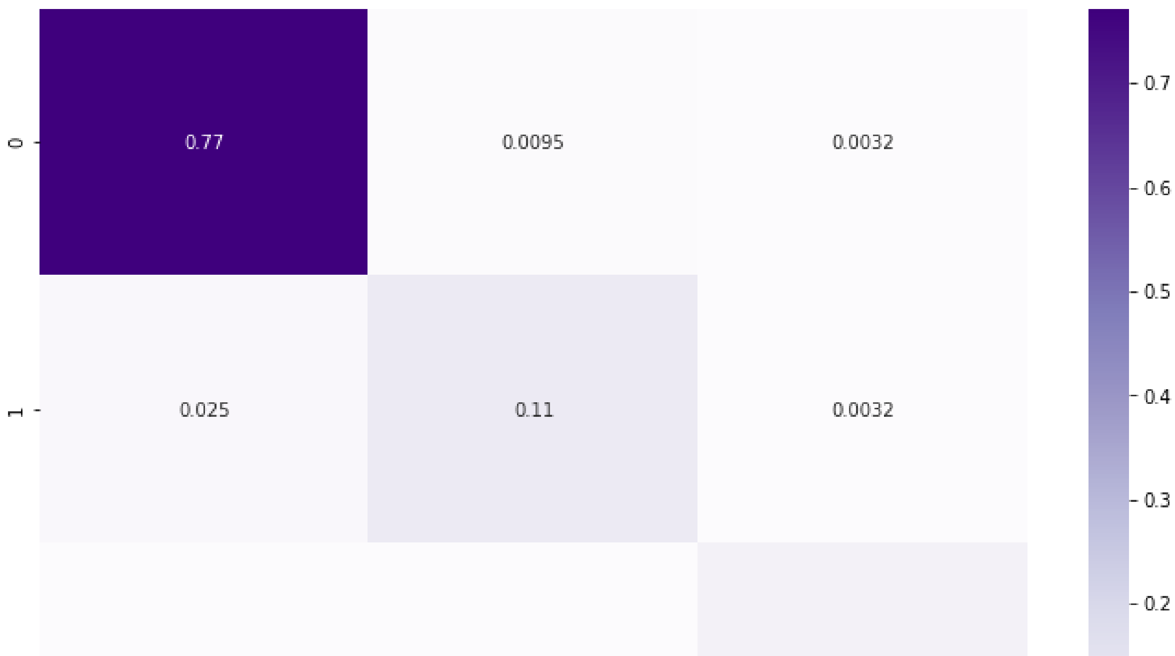
```
              precision    recall  f1-score   support

         1.0       0.97      0.98      0.98       497
         2.0       0.90      0.80      0.85        90
         3.0       0.92      0.96      0.94        47

    accuracy                           0.96       634
   macro avg       0.93      0.91      0.92       634
weighted avg       0.95      0.96      0.95       634
```

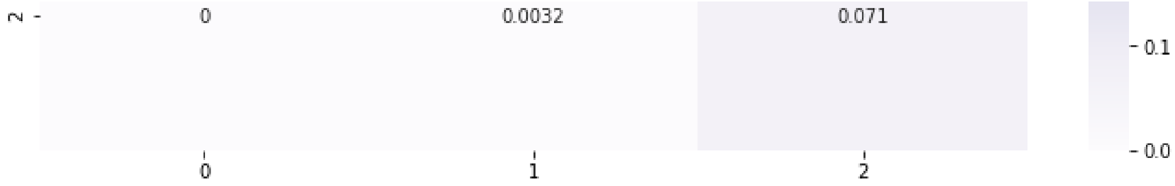
In [ ]:

```
plt.subplots(figsize=(12,8))
gb_decision_tree = confusion_matrix(y_test, gb_predictions)
sns.heatmap(gb_decision_tree/np.sum(gb_decision_tree),annot = True, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae05e90d0>
```





LightGBM Classifier

In [ ]:

```
from lightgbm import LGBMClassifier
```

In [ ]:

```
light_gbm = LGBMClassifier()
light_gbm.fit(X_train, y_train)
light_gbm_score = light_gbm.score(X_test, y_test)
print("Baseline LGBM Classifer Score: " + str(light_gbm_score))
```

```
Baseline LGBM Classifer Score: 0.9558359621451105
```

In [ ]:

```
cv_score_light_gbm = cross_val_score(light_gbm, X_train, y_train)

print("Baseline LightGBM CV Score: " + str(cv_score_light_gbm.mean()))
```

```
Baseline LightGBM CV Score: 0.9384722858451671
```

In [ ]:

```
light_gbm_predictions = light_gbm.predict(X_test)
print(classification_report(y_test, light_gbm_predictions))
```

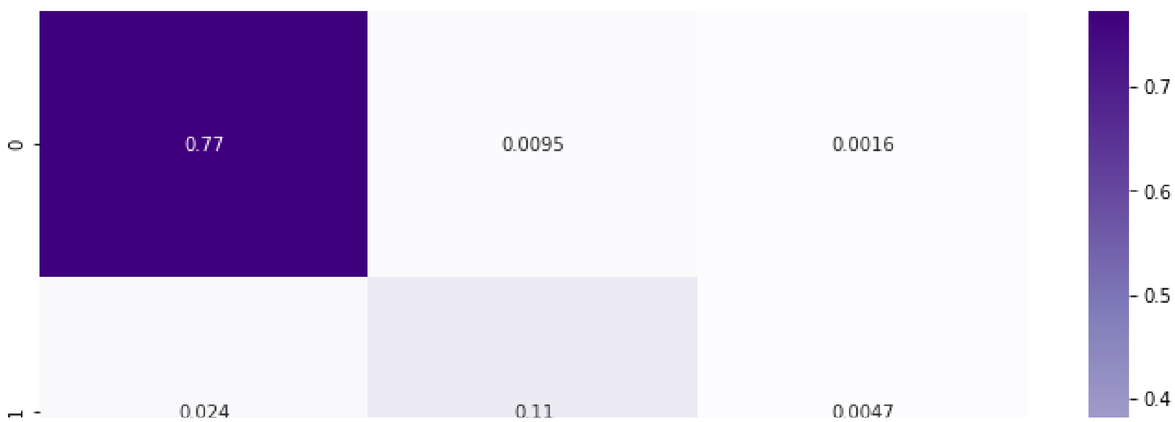
```
              precision    recall  f1-score   support

         1.0       0.97      0.99      0.98       497
         2.0       0.90      0.80      0.85        90
         3.0       0.92      0.94      0.93        47

    accuracy                           0.96       634
   macro avg       0.93      0.91      0.92       634
weighted avg       0.95      0.96      0.95       634
```

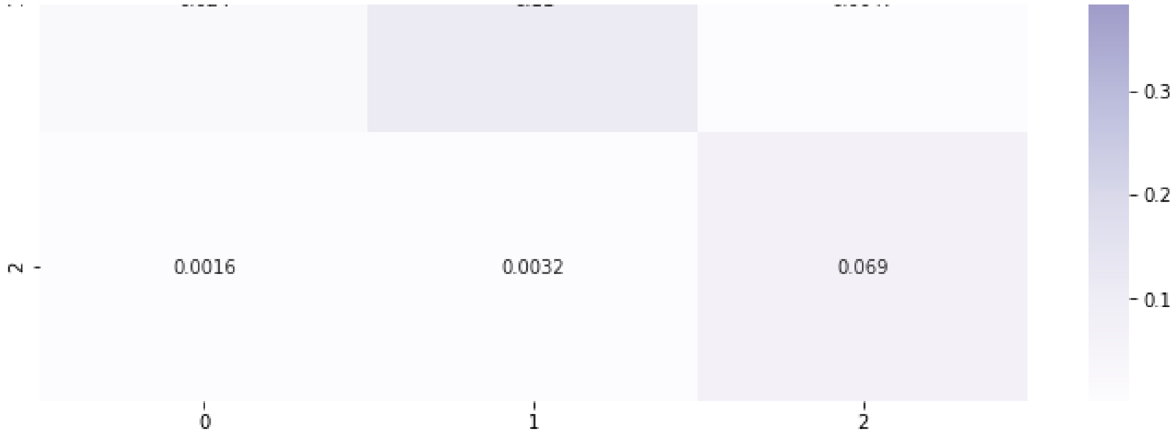
In [ ]:

```
plt.subplots(figsize=(12,8))
light_gbm_decision_tree = confusion_matrix(y_test, light_gbm_predictions)
sns.heatmap(light_gbm_decision_tree/np.sum(light_gbm_decision_tree),annot = True, cmap="Purples")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae053fad0>
```





XGBoost Classifier

In [ ]:

```
from xgboost import XGBClassifier
import xgboost
```

In [ ]:

```
xgboost = XGBClassifier()
xgboost.fit(X_train, y_train)
xgboost_score = xgboost.score(X_test, y_test)
print("Baseline XGBoost Classifer Score: " + str(xgboost_score))
```

```
Baseline XGBoost Classifer Score: 0.9495268138801262
```

In [ ]:

```
cv_score_xgboost = cross_val_score(xgboost, X_train, y_train)

print("Baseline XGBoost CV Score: " + str(cv_score_xgboost.mean()))
```

```
Baseline XGBoost CV Score: 0.9357581310123683
```

In [ ]:

```
xgboost_predictions = xgboost.predict(X_test)
print(classification_report(y_test, xgboost_predictions))
```

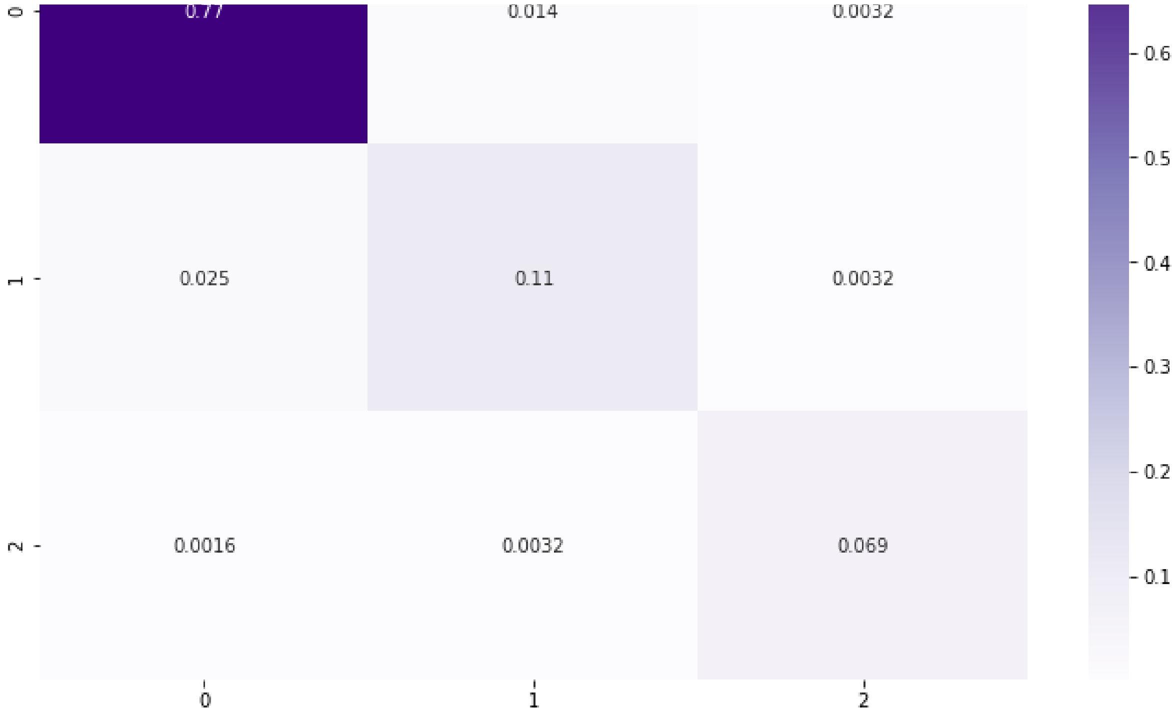
```
              precision    recall  f1-score   support

         1.0       0.97      0.98      0.97       497
         2.0       0.87      0.80      0.83        90
         3.0       0.92      0.94      0.93        47

    accuracy                           0.95       634
   macro avg       0.92      0.90      0.91       634
weighted avg       0.95      0.95      0.95       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
xgboost_decision_tree = confusion_matrix(y_test, xgboost_predictions)
sns.heatmap(xgboost_decision_tree/np.sum(xgboost_decision_tree),annot = True, cmap="Purp
les")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ae0478550>
```



Model Scoring and Selection

In [ ]:

```
model_scores = pd.DataFrame({"Model": ["Logistic Regression", "Random Forest",
                                        "K-Nearest Neighbors", "Support Vector", "Linear S
upport Vector",
                                        "Decision Tree", "Multi-layer Perceptron", "Gradie
nt Boosting",
                                        "LightGBM Classifer", "XGBoost Classifier"],
                             "Scores": [log_reg_score, random_forest_score, knn_score, sv
c_score, linear_svc_score, decision_tree_score,
                                         neural_network_score, gradient_boosting_score, li
ght_gbm_score, xgboost_score]})
model_scores.sort_values(by="Scores", ascending=False)
```

Out[ ]:

	Model	Scores
7	Gradient Boosting	0.955836
8	LightGBM Classifer	0.955836
1	Random Forest	0.949527
9	XGBoost Classifier	0.949527
2	K-Nearest Neighbors	0.940063
6	Multi-layer Perceptron	0.935331
5	Decision Tree	0.933754
3	Support Vector	0.921136
0	Logistic Regression	0.895899
4	Linear Support Vector	0.892744

Model Tuning and Optimization

LightGBM Classifier Tuning/Optimization

In [ ]:

```
param_grid = {
  'learning_rate': [0.001,0.01],
  'n_estimators': [ 1000],
  'num_leaves': [12, 30,80],
  'boosting_type' : ['gbdt'],
  'objective' : ['binary'],
  'random_state' : [1],
  'colsample_bytree' : [ 0.8, 1],
  'subsample' : [0.5,0.7,0.75],
  'reg_alpha' : [0.1, 1.2],
  'reg_lambda' : [0.1, 1.2],
  'subsample_freq' : [500,1000],
  'max_depth' : [15, 30, 80]
}
```

In [ ]:

```
cv_method = StratifiedKFold(n_splits=3)
```

In [ ]:

```
GridSearchCV_GBM = GridSearchCV(estimator=LGBMClassifier(),
                                param_grid=param_grid,
                                cv=cv_method,
                                verbose=1,
                                n_jobs=4,
                                scoring="accuracy",
                                return_train_score=True
                                )
```

In [ ]:

```
GridSearchCV_GBM.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 864 candidates, totalling 2592 fits
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:   36.7s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:  2.8min
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:  6.5min
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed: 11.8min
[Parallel(n_jobs=4)]: Done 1242 tasks      | elapsed: 19.4min
[Parallel(n_jobs=4)]: Done 1792 tasks      | elapsed: 27.9min
[Parallel(n_jobs=4)]: Done 2442 tasks      | elapsed: 38.9min
[Parallel(n_jobs=4)]: Done 2592 out of 2592 | elapsed: 41.6min finished
```

Out[ ]:

```
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=False),
             error_score=nan,
             estimator=LGBMClassifier(boosting_type='gbdt', class_weight=None,
                                      colsample_bytree=1.0,
                                      importance_type='split',
                                      learning_rate=0.1, max_depth=-1,
                                      min_child_samples=20,
                                      min_child_weight=0.001,
                                      min_split_gain=0.0, n_estimators=100,
                                      n_jobs=-1, num_leaves=31, objective=None...
                         'colsample_bytree': [0.8, 1],
                         'learning_rate': [0.001, 0.01],
                         'max_depth': [15, 30, 80], 'n_estimators': [1000],
                         'num_leaves': [12, 30, 80], 'objective': ['binary'],
                         'random_state': [1], 'reg_alpha': [0.1, 1.2],
                         'reg_lambda': [0.1, 1.2],
                         'subsample': [0.5, 0.7, 0.75],
                         'subsample_freq': [500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=1)
```

In [ ]:

```
best_params_GBM = GridSearchCV_GBM.best_params_
```

```
best_params_GBM = GridSearchCV_GBM.best_params_
best_params_GBM
```

Out[ ]:

```
{'boosting_type': 'gbdt',
 'colsample_bytree': 0.8,
 'learning_rate': 0.01,
 'max_depth': 15,
 'n_estimators': 1000,
 'num_leaves': 12,
 'objective': 'binary',
 'random_state': 1,
 'reg_alpha': 0.1,
 'reg_lambda': 1.2,
 'subsample': 0.7,
 'subsample_freq': 500}
```

In [ ]:

```
best_score_GBM = GridSearchCV_GBM.best_score_
print("Optimized Grid Search GBM best score: " + str(best_score_GBM))
```

```
Optimized Grid Search GBM best score: 0.9263015551048005
```

In [ ]:

```
optimized_LGBM = LGBMClassifier(**GridSearchCV_GBM.best_params_)
optimized_LGBM.fit(X_train, y_train)
```

Out[ ]:

```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=0.8,
               importance_type='split', learning_rate=0.01, max_depth=15,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=1000, n_jobs=-1, num_leaves=12, objective='binary',
               random_state=1, reg_alpha=0.1, reg_lambda=1.2, silent=True,
               subsample=0.7, subsample_for_bin=200000, subsample_freq=500)
```

In [ ]:

```
optimized_LGBM_score = optimized_LGBM.score(X_test, y_test)
print("Optimized LGBM Classifer Score: " + str(optimized_LGBM_score))
```

```
Optimized LGBM Classifer Score: 0.9479495268138801
```

In [ ]:

```
cv_score_optimized_LGBM = cross_val_score(optimized_LGBM, X_train, y_train)

print("Optimized LightGBM CV Score: " + str(cv_score_optimized_LGBM.mean()))
```

```
Optimized LightGBM CV Score: 0.9222446174988548
```

In [ ]:

```
optimized_LGBM_predictions = optimized_LGBM.predict(X_test)
print(classification_report(y_test, optimized_LGBM_predictions))
```

```
              precision    recall  f1-score   support

         1.0       0.96      0.98      0.97       497
         2.0       0.89      0.76      0.82        90
         3.0       0.88      0.98      0.93        47

    accuracy                           0.95       634
   macro avg       0.91      0.90      0.91       634
weighted avg       0.95      0.95      0.95       634
```

In [ ]:

```
plt.subplots(figsize=(12,8))
```

```
optimized_light_gbm_decision_tree = confusion_matrix(y_test, optimized_LGBM_predictions)
sns.heatmap(optimized_light_gbm_decision_tree/np.sum(optimized_light_gbm_decision_tree),a
nnot = True, cmap="Purples")
```

Out[ ]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f4adf2c1a50>