In [1]:
```
import pandas as pd
import numpy as np
import re
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [13]:
```
df= pd.read_csv('fetal_health.csv')
```

In [15]:
```
df.head(5)
```

Out[15]:

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

5 rows × 22 columns

In [16]:
```
df.shape
```

Out[16]:
```
(2126, 22)
```

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
baseline value                                            0
accelerations                                             0
fetal_movement                                            0
uterine_contractions                                      0
light_decelerations                                       0
severe_decelerations                                      0
prolongued_decelerations                                  0
abnormal_short_term_variability                           0
mean_value_of_short_term_variability                      0
percentage_of_time_with_abnormal_long_term_variability    0
mean_value_of_long_term_variability                       0
histogram_width                                           0
histogram_min                                             0
histogram_max                                             0
histogram_number_of_peaks                                 0
histogram_number_of_zeroes                                0
histogram_mode                                            0
histogram_mean                                            0
histogram_median                                          0
histogram_variance                                        0
histogram_tendency                                        0
fetal_health                                              0
dtype: int64
```

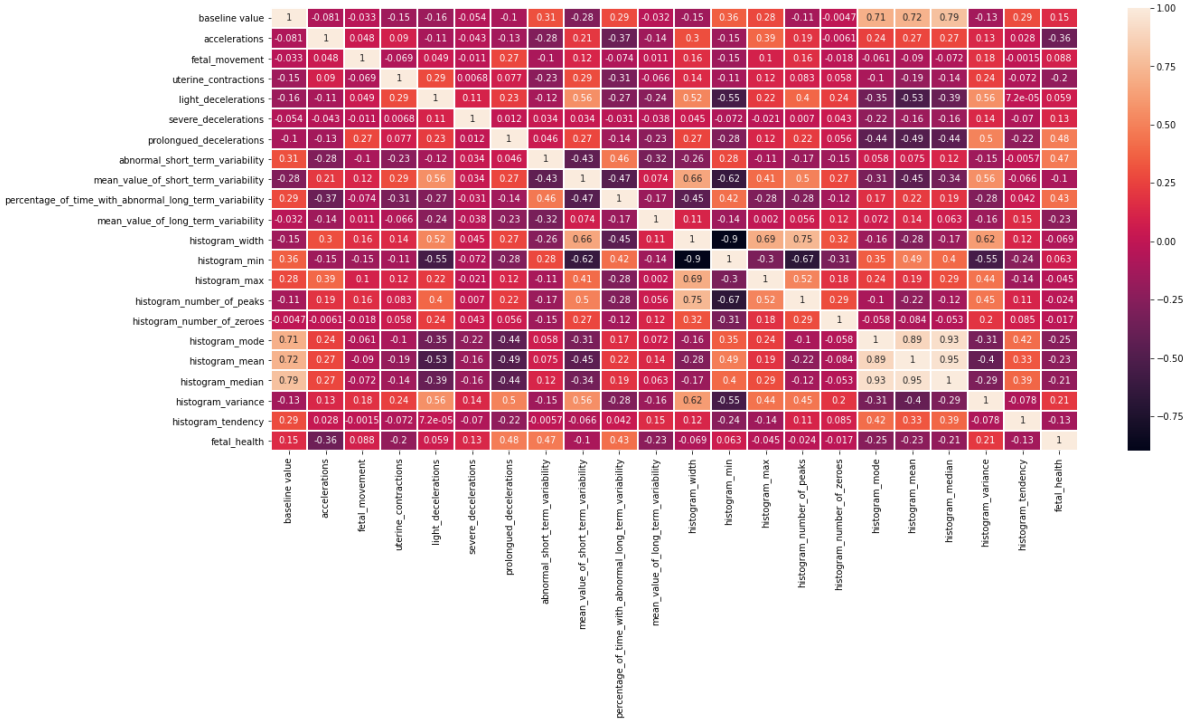
In [18]:
```
df.fetal_health.unique()
```

Out[18]:
```
array([2., 1., 3.])
```

In [19]:
```
corr_mat = df.corr()
```

In [20]:
```
plt.figure(figsize=(20,9))
sns.heatmap(data=corr_mat,annot=True,linewidth=1,linecolor='w')
```

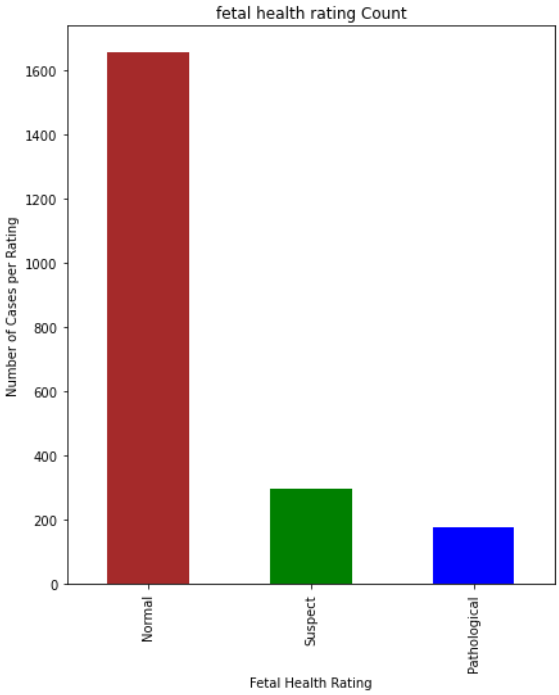
Out[20]:
```
<AxesSubplot:>
```

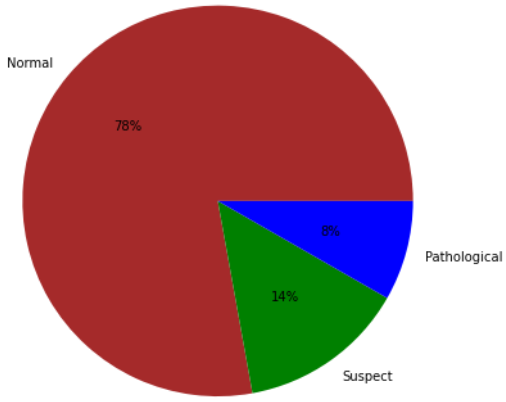


In [12]:
```
data1 = data.copy()
data1.fetal_health = data1.fetal_health.astype('int')
data1.fetal_health = data1.fetal_health.replace([1,2,3],['Normal','Suspect','Patho
```

In [13]:
```
plt.figure(figsize=(15,8))

plt.subplot(121)
data1.fetal_health.value_counts().plot(kind='bar',figsize=(15,8),color = ['brown',
plt.title('fetal health rating Count')
plt.xlabel('Fetal Health Rating')
plt.ylabel('Number of Cases per Rating')

plt.subplot(122)
plt.pie(data1.fetal_health.value_counts(),labels=[
    'Normal', 'Suspect', 'Pathological'], colors = ['brown', 'g', 'b'], autopct='%
```

Out[13]:
```
([<matplotlib.patches.Wedge at 0x229007b9460>,
  <matplotlib.patches.Wedge at 0x229007b9bb0>,
  <matplotlib.patches.Wedge at 0x229007c9280>],
 [Text(-0.8441562311892146, 0.7052660897451555, 'Normal'),
  Text(0.6344062353380878, -0.8986260226390926, 'Suspect'),
  Text(1.0630076905865578, -0.28286860863983, 'Pathological')],
 [Text(-0.4604488533759351, 0.38469059440644837, '78%'),
  Text(0.3460397647298661, -0.4901596487122323, '14%'),
  Text(0.5798223766835768, -0.15429196834899814, '8%')])
```





KNN

In [21]:
```
# Defining Features Matrix
X = df.drop(['fetal_health'], axis=1)
X.head()
```

Out[21]:

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

5 rows × 21 columns

In [22]:
```
# Define Target
y = data['fetal_health']
y.head()
```

Out[22]:
```
0    2.0
1    1.0
2    1.0
3    1.0
4    1.0
Name: fetal_health, dtype: float64
```

In [24]:
```
# Separate training and testing sets, stratifying by class
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
```

In [25]:
```
#Standardize variables
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

In [26]:
```
y_train.value_counts()
```

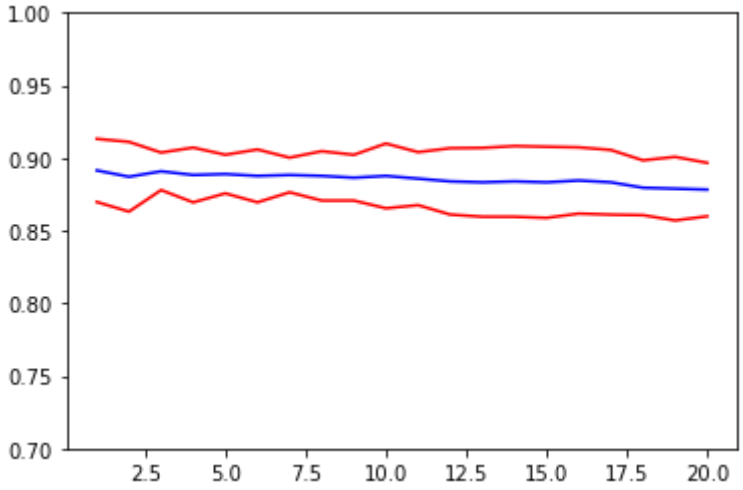
Out[26]:
```
1.0    1241
2.0     221
3.0     132
Name: fetal_health, dtype: int64
```

In [31]:
```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.neighbors import KNeighborsClassifier
```

In [60]:
```
#Creating Function for Cross Validation

def scores_knn(X, y, start,stop,step):

    # We are going to graph the different values of the cross validation score base
    # For this we are going to generate a list of dictionaries that can then be eas

    # List of dictionaries - we initialize it empty and outside the for loop to fee
    scores_for_df = []


    for i in range(start,stop,step):

        # At each iteration, we instantiate the model with a different hyperparamet
        model = KNeighborsClassifier(n_neighbors=i)

        # cross_val_scores returns an array of 5 results, one for each partition th
        kf = KFold(n_splits=10, shuffle=True, random_state=10)
        cv_scores = cross_val_score(model, X, y, cv=kf)

        # For each value of n_neighbours, we create a dictionary with the value of
        dict_row_score = {'medium_score':np.mean(cv_scores),'score_std':np.std(cv_s

        # We save each one in the list of dictionaries
        scores_for_df.append(dict_row_score)

    # We create the DF from the list of results
    df_scores = pd.DataFrame(scores_for_df)

    # We incorporate the lower and upper limits, subtracting and adding the value o
    df_scores['lower_limit'] = df_scores['medium_score'] - df_scores['score_std']
    df_scores['upper_limit'] = df_scores['medium_score'] + df_scores['score_std']

    # Return DF
    return df_scores
```

In [61]:
```
# CV for 1 to 20 neighbors
df_scores= scores_knn(X_train, y_train, 1, 21, 1)
```

In [62]:
```
# viewing the results

plt.plot(df_scores['n_neighbours'], df_scores['lower_limit'], color='r')
plt.plot(df_scores['n_neighbours'], df_scores['medium_score'], color='b')
plt.plot(df_scores['n_neighbours'], df_scores['upper_limit'], color='r')
plt.ylim(0.7, 1);
```



In [63]:
```
df_scores
```

Out[63]:

	medium_score	score_std	n_neighbours	lower_limit	upper_limit
0	0.891458	0.021735	1	0.869723	0.913193
1	0.887099	0.024012	2	0.863087	0.911111
2	0.890841	0.012905	3	0.877936	0.903746
3	0.888318	0.018853	4	0.869465	0.907171
4	0.888947	0.013282	5	0.875665	0.902228
5	0.887685	0.018183	6	0.869502	0.905867
6	0.888322	0.011924	7	0.876398	0.900245
7	0.887697	0.016989	8	0.870708	0.904685
8	0.886447	0.015748	9	0.870699	0.902194
9	0.887677	0.022292	10	0.865385	0.909969
10	0.885802	0.018258	11	0.867544	0.904060
11	0.883907	0.022847	12	0.861060	0.906754
12	0.883278	0.023687	13	0.859591	0.906965
13	0.883903	0.024342	14	0.859562	0.908245
14	0.883282	0.024487	15	0.858795	0.907770
15	0.884536	0.022810	16	0.861727	0.907346
16	0.883286	0.022264	17	0.861022	0.905550
17	0.879532	0.018877	18	0.860655	0.898410
18	0.878895	0.021886	19	0.857009	0.900782
19	0.878270	0.018392	20	0.859879	0.896662

In [64]:
```
#Balancing Target Data with SMOTE
from imblearn.over_sampling import SMOTE

def SMOTE_f(X_train,y_train):
    sm = SMOTE(random_state=42)
    X_train_smo, y_train_smo = sm.fit_resample(X_train,y_train)
    return X_train_smo, y_train_smo
```

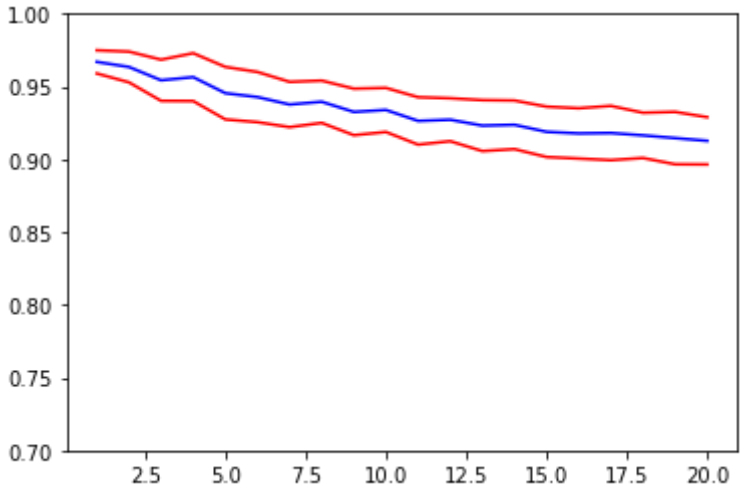
In [65]:
```
X_train_smo, y_train_smo = SMOTE_f(X_train,y_train)
```

In [66]:
```
#checking that the target data is balanced
y_train_smo.value_counts()
```

Out[66]:
```
2.0    1241
3.0    1241
1.0    1241
Name: fetal_health, dtype: int64
```

In [67]:
```
# PCV for 1 to 20 neighbors with SMOTE
df_scores= scores_knn(X_train_smo, y_train_smo, 1, 21, 1)
```

In [68]:
```
# viewing the results

plt.plot(df_scores['n_neighbours'], df_scores['lower_limit'], color='r')
plt.plot(df_scores['n_neighbours'], df_scores['medium_score'], color='b')
plt.plot(df_scores['n_neighbours'], df_scores['upper_limit'], color='r')
plt.ylim(0.7, 1);
```



In [69]:
```
df_scores
```

Out[69]:

	medium_score	score_std	n_neighbours	lower_limit	upper_limit
0	0.966964	0.007965	1	0.958999	0.974928
1	0.963470	0.010553	2	0.952918	0.974023
2	0.954340	0.014151	3	0.940189	0.968491
3	0.956490	0.016447	4	0.940043	0.972936
4	0.945476	0.018047	5	0.927429	0.963522
5	0.942787	0.017235	6	0.925553	0.960022
6	0.937685	0.015557	7	0.922128	0.953242
7	0.939564	0.014526	8	0.925037	0.954090
8	0.932584	0.015985	9	0.916598	0.948569
9	0.933925	0.015143	10	0.918782	0.949068
10	0.926406	0.016292	11	0.910114	0.942698
11	0.927213	0.014757	12	0.912457	0.941970
12	0.923186	0.017506	13	0.905680	0.940692
13	0.923722	0.016706	14	0.907016	0.940428
14	0.918891	0.017306	15	0.901586	0.936197
15	0.917815	0.017289	16	0.900526	0.935105
16	0.918086	0.018646	17	0.899440	0.936733
17	0.916472	0.015554	18	0.900918	0.932027
18	0.914595	0.018001	19	0.896593	0.932596
19	0.912716	0.016245	20	0.896471	0.928961

In [70]:
```
#Identifying the maximum score
df_scores.loc[df_scores.medium_score == df_scores.medium_score.max()]
```

Out[70]:

	medium_score	score_std	n_neighbours	lower_limit	upper_limit
0	0.966964	0.007965	1	0.958999	0.974928

In [71]:
```
# Assigning the value of optimal k to a variable
best_k = df_scores.loc[df_scores.medium_score == df_scores.medium_score.max(),'n_n
best_k
```

Out[71]:
```
1
```

In [72]:
```
# Choosing the optimal model that cross validation had indicated
model = KNeighborsClassifier(n_neighbors=best_k)

# Fitting on training data
model.fit(X_train_smo, y_train_smo)
```

Out[72]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [75]:
```
from sklearn.metrics import accuracy_score

#  Evaluatingn accuracy on train
accuracy_score(y_train, model.predict(X_train))
```

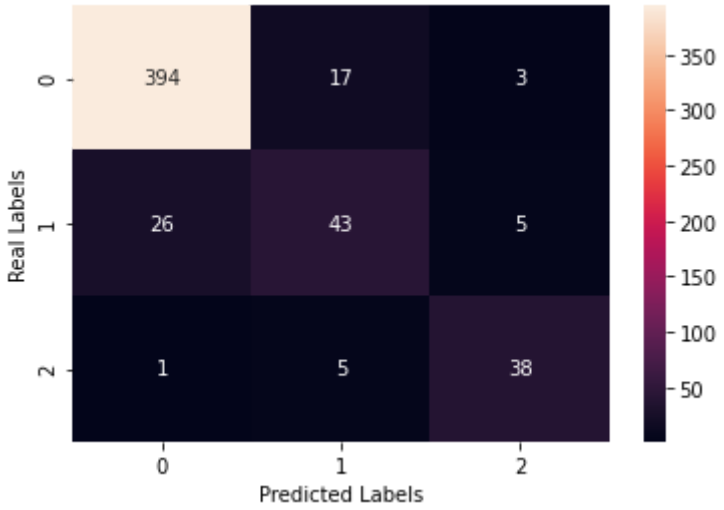
Out[75]:
```
0.9993726474278545
```

In [76]:
```
# Predicting on test
y_pred = model.predict(X_test)
```

In [77]:
```
# Evaluatingn accuracy on test
accuracy_score(y_test, y_pred)
```

Out[77]:
```
0.8928571428571429
```

In [78]:
```
from sklearn.metrics import confusion_matrix

# Plot Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='.0f')
plt.ylabel('Real Labels')
plt.xlabel('Predicted Labels');
```



In [87]:
```
#Clasification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.94      0.95      0.94       414
         2.0       0.66      0.58      0.62        74
         3.0       0.83      0.86      0.84        44

    accuracy                           0.89       532
   macro avg       0.81      0.80      0.80       532
weighted avg       0.89      0.89      0.89       532
```

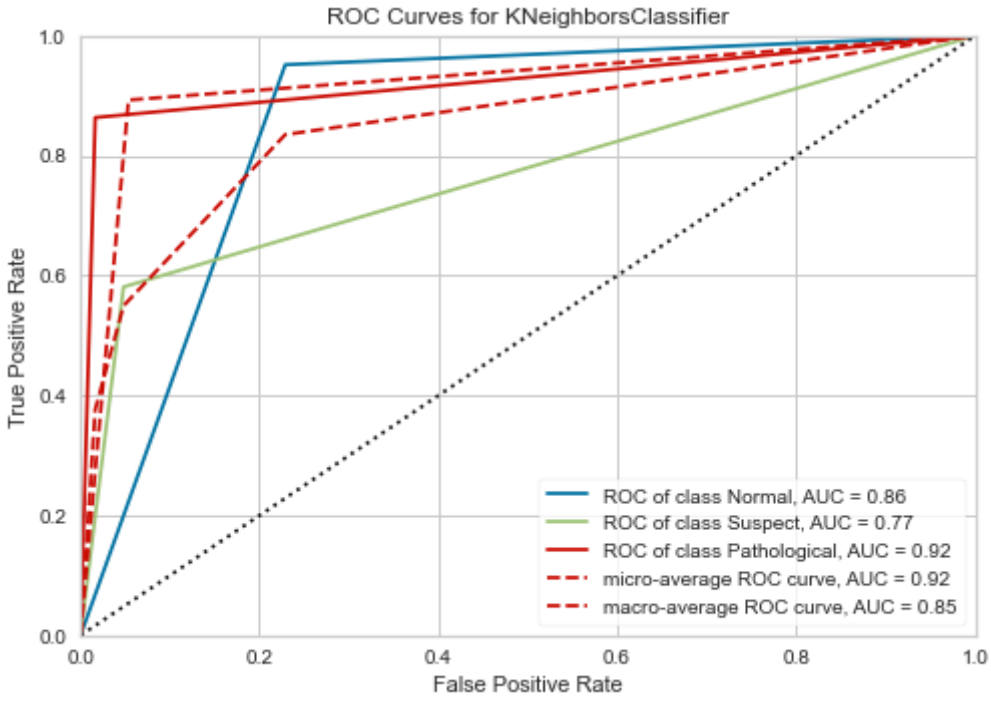
In [47]:
```
from sklearn.metrics import roc_curve
```

In [89]:
```
#Installing YellowBrick for multi class ROCAUC implementation
pip install yellowbrick
```

In [90]:
```
from yellowbrick.classifier import ROCAUC
```

In [91]:
```
model = model
```

```
visualizer = ROCAUC(model, classes=["Normal", "Suspect", "Pathological"])

visualizer.fit(X_train_smo, y_train_smo)        # Fit the training data to the visu
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```



Out[91]: <AxesSubplot:title={'center':'ROC Curves for KNeighborsClassifier'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>