Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
# NN models
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.wrappers.scikit_learn import KerasClassifier
from keras.callbacks import EarlyStopping, ModelCheckpoint
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/milkquality/milknew.csv
```

Loading the dataset

In [2]:
```
df=pd.read_csv('/kaggle/input/milkquality/milknew.csv')
df
```

Out[2]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	high
1	6.6	36	0	1	0	1	253	high
2	8.5	70	1	1	1	1	246	low
3	9.5	34	1	1	0	1	255	low
4	6.6	37	0	0	0	0	255	medium
...	...	...	...	...	...	...	...	...
1054	6.7	45	1	1	0	0	247	medium
1055	6.7	38	1	0	1	0	255	high
1056	3.0	40	1	1	1	1	255	low
1057	6.8	43	1	0	1	0	250	high
1058	8.6	55	0	1	1	1	255	low

1059 rows × 8 columns

Exploratory Data Analysis

In [3]:
```
df.shape
```

Out[3]:
```
(1059, 8)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1059 entries, 0 to 1058
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   pH          1059 non-null   float64
 1   Temprature  1059 non-null   int64  
 2   Taste       1059 non-null   int64  
 3   Odor        1059 non-null   int64  
 4   Fat         1059 non-null   int64  
 5   Turbidity   1059 non-null   int64  
 6   Colour      1059 non-null   int64  
 7   Grade       1059 non-null   object 
dtypes: float64(1), int64(6), object(1)
memory usage: 66.3+ KB
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
976
```

In [7]:
```
df.nunique()
```

Out[7]:
```
pH            16
Temprature    17
Taste          2
Odor           2
Fat            2
Turbidity      2
Colour         9
Grade          3
dtype: int64
```

In [8]:
```
df['Grade'].value_counts()
```

Out[8]:
```
low       429
medium    374
high      256
Name: Grade, dtype: int64
```

In [9]:
```
#df['Grade'] =df['Grade'].map({'low': 0, 'medium': 1,'high':2})
```

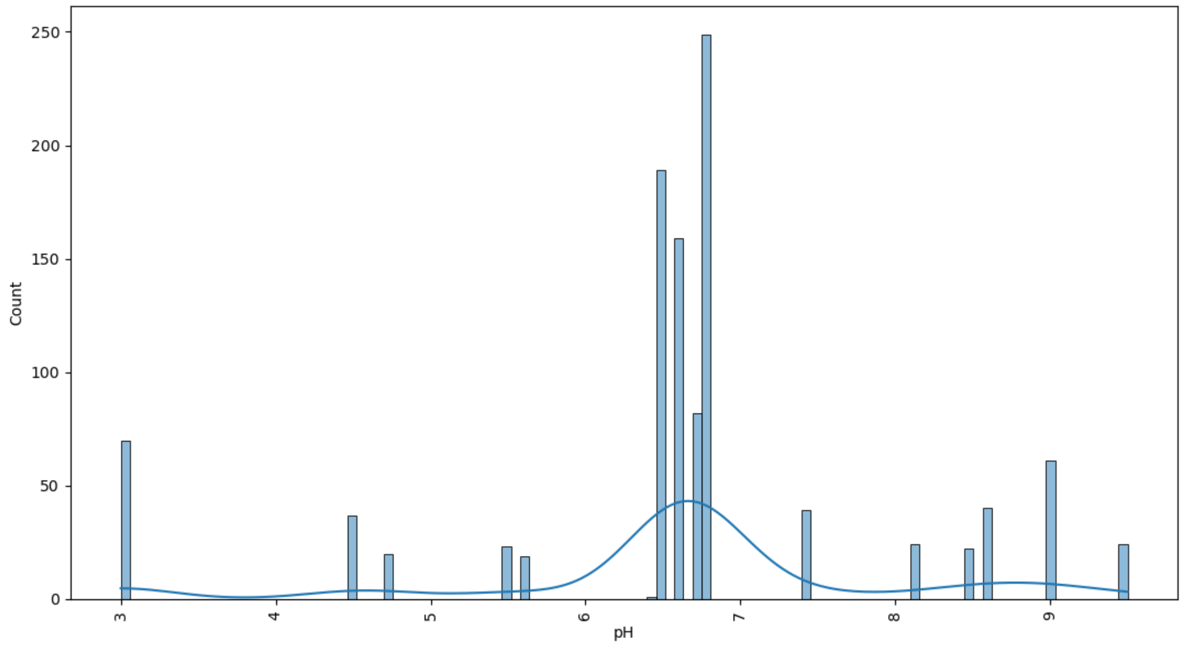
In [10]:
```
df.describe().T
```

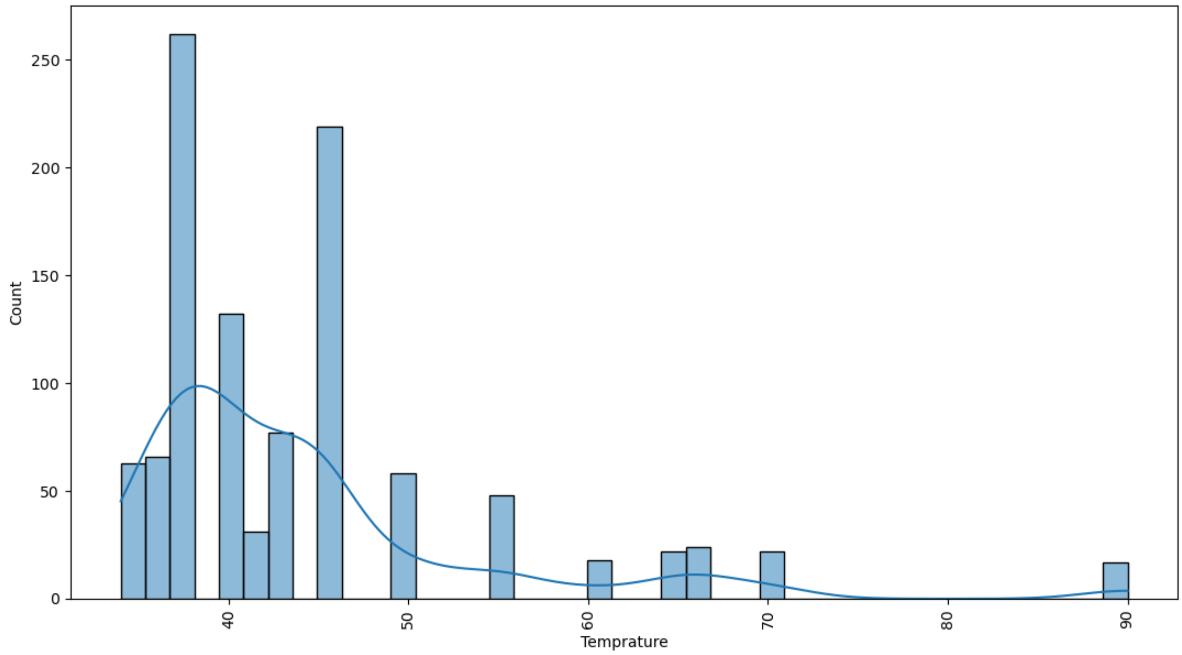
Out[10]:

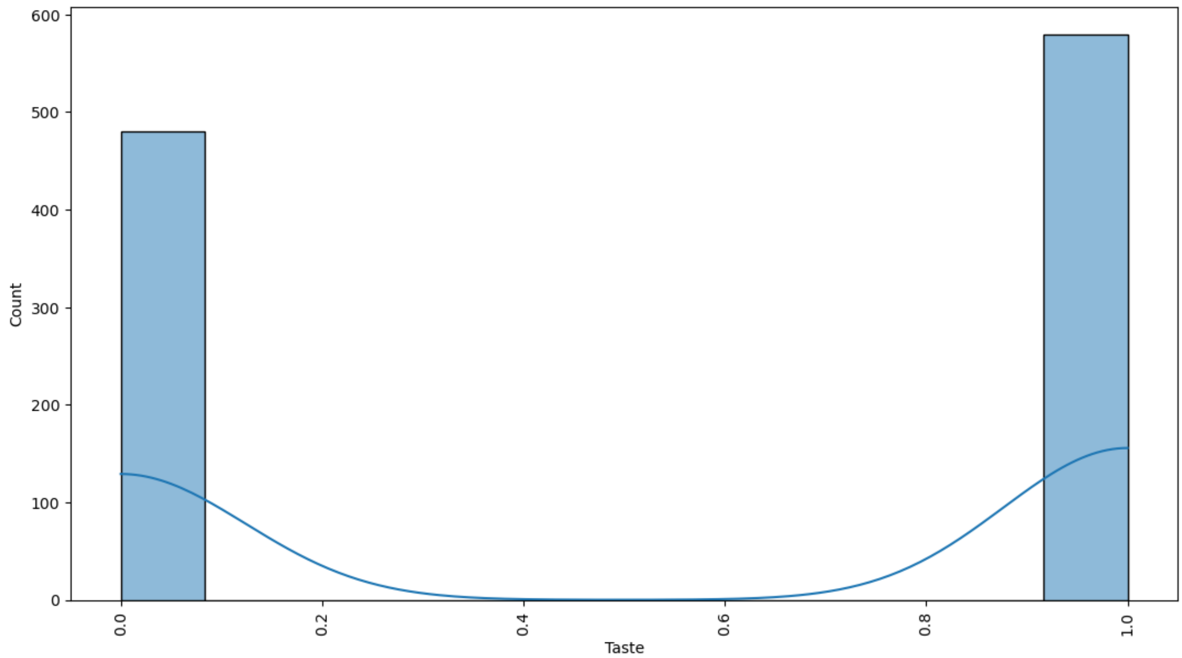
	count	mean	std	min	25%	50%	75%	max
pH	1059.0	6.630123	1.399679	3.0	6.5	6.7	6.8	9.5
Temprature	1059.0	44.226629	10.098364	34.0	38.0	41.0	45.0	90.0
Taste	1059.0	0.546742	0.498046	0.0	0.0	1.0	1.0	1.0
Odor	1059.0	0.432483	0.495655	0.0	0.0	0.0	1.0	1.0
Fat	1059.0	0.671388	0.469930	0.0	0.0	1.0	1.0	1.0
Turbidity	1059.0	0.491029	0.500156	0.0	0.0	0.0	1.0	1.0
Colour	1059.0	251.840415	4.307424	240.0	250.0	255.0	255.0	255.0

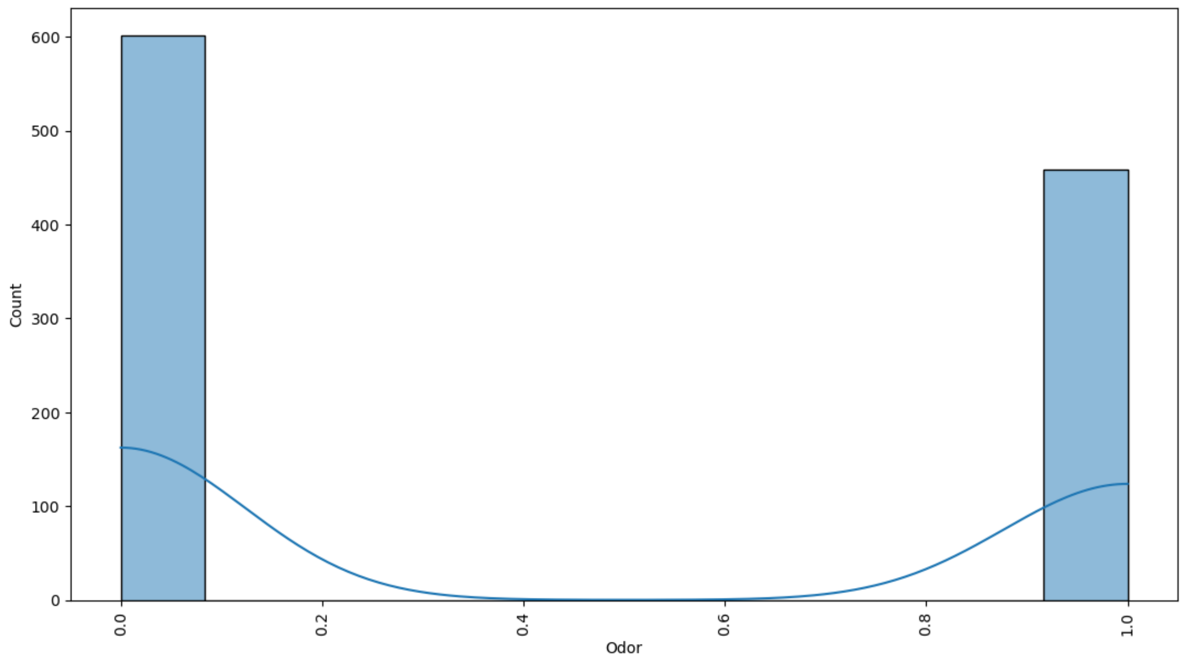
Data Visualization

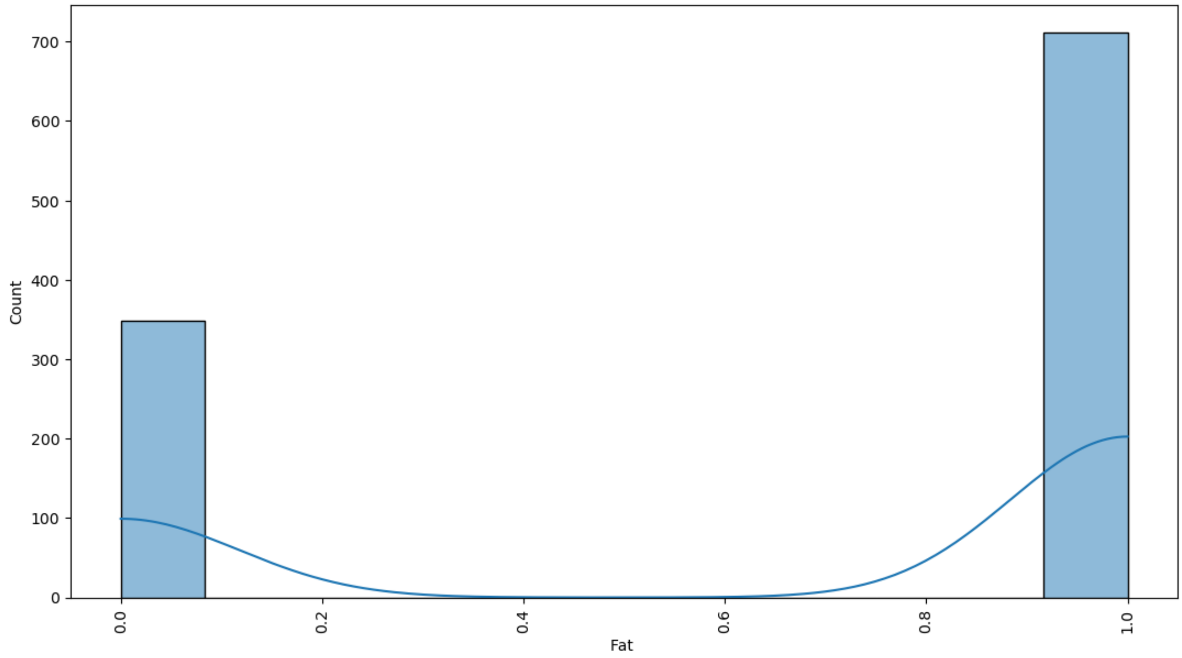
In [11]:
```
for i in df.columns:
    plt.figure(figsize=(13,7))
    sns.histplot(data = df[i], kde=True, multiple='stack')
    plt.xticks(rotation=90)
    plt.show()
```

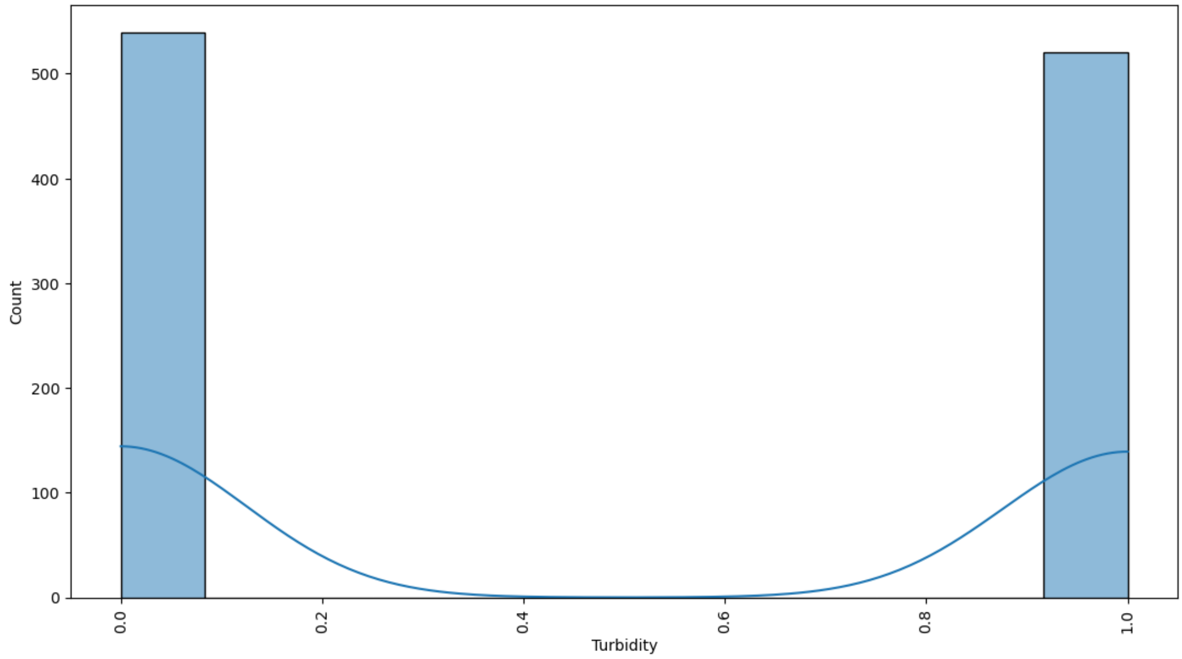


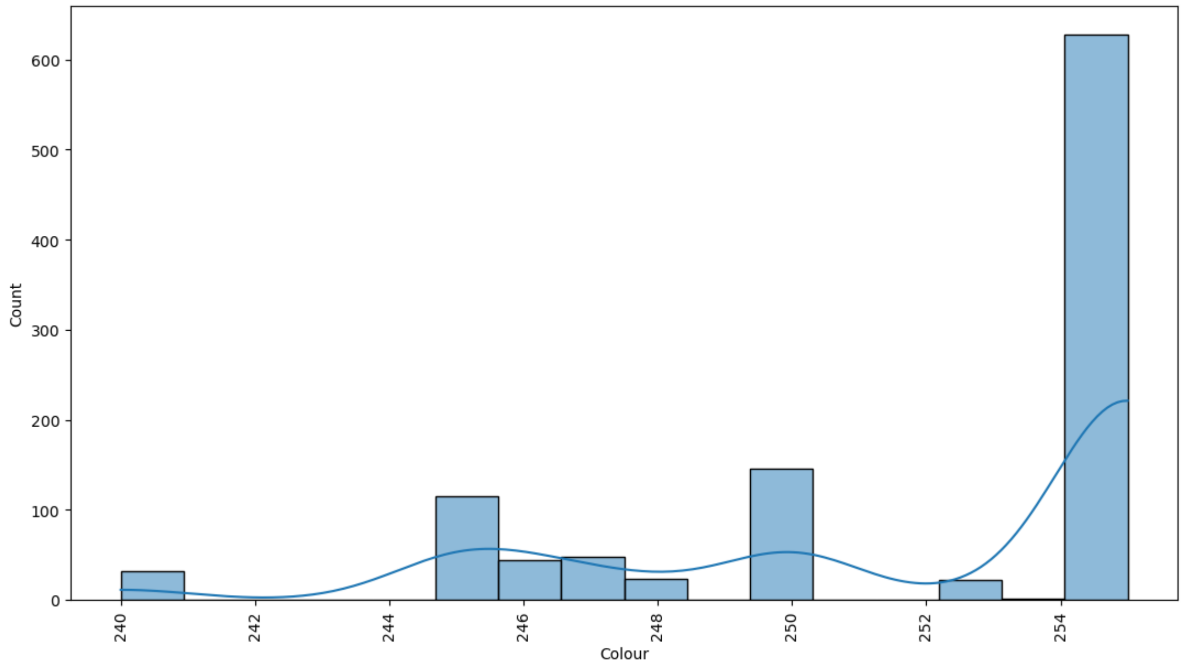

Count
Temprature

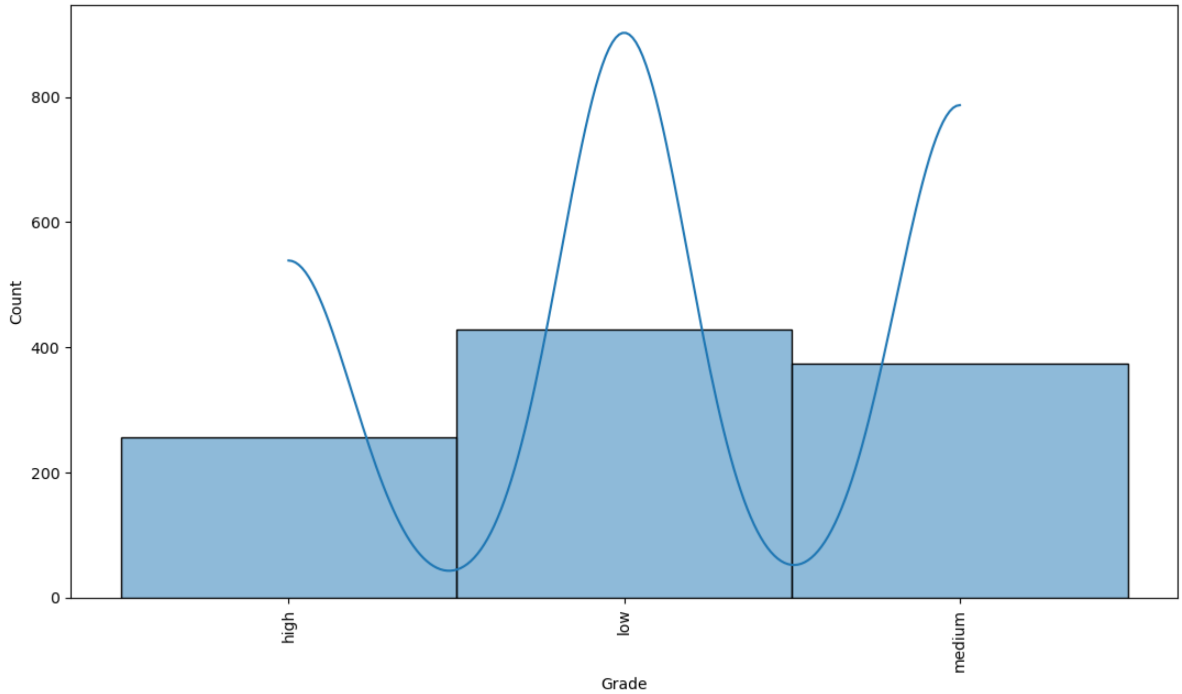

Count
Taste


Count
Odor


Count
Fat
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500
600
700

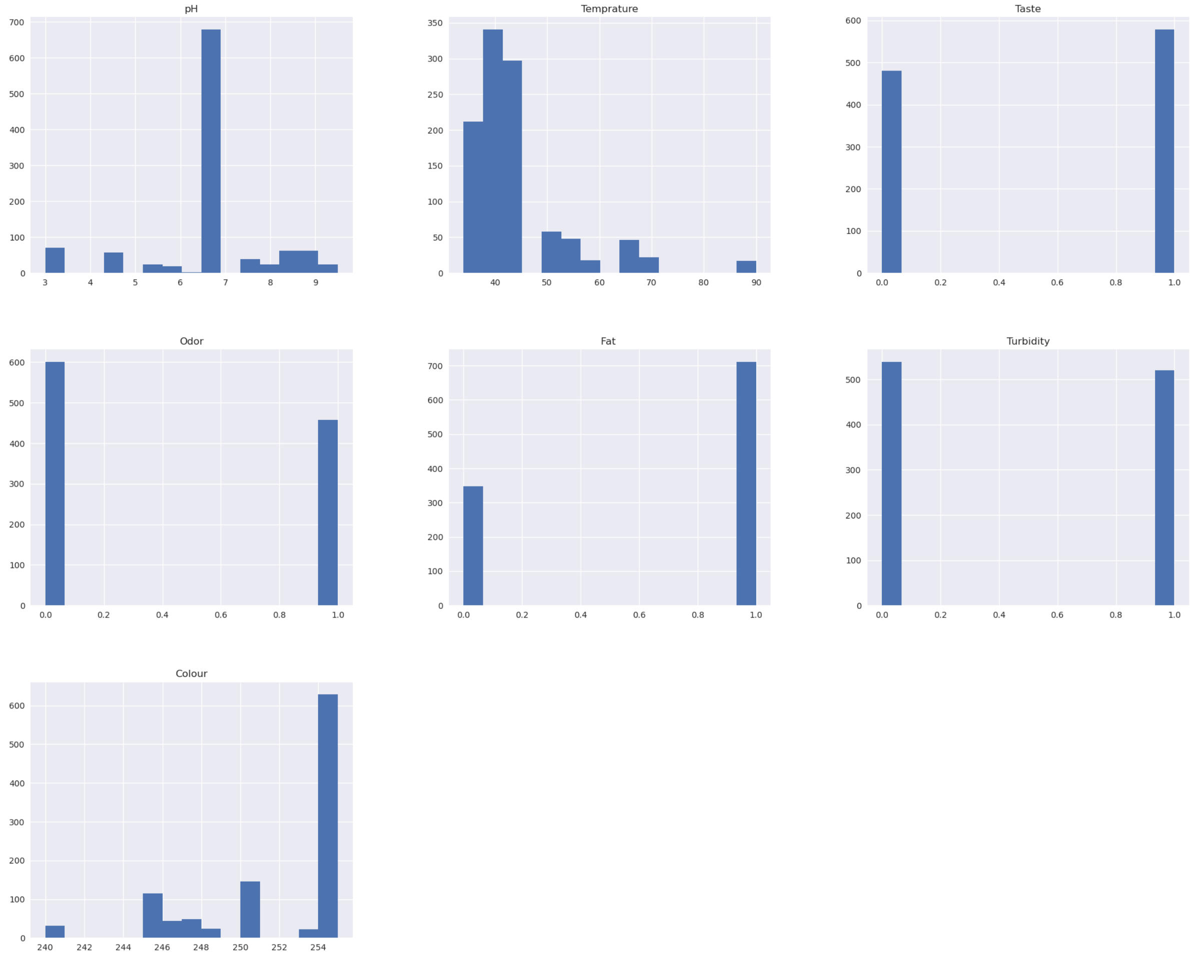

Count
Turbidity
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500


Count
Colour
240
242
244
246
248
250
252
254
0
100
200
300
400
500
600

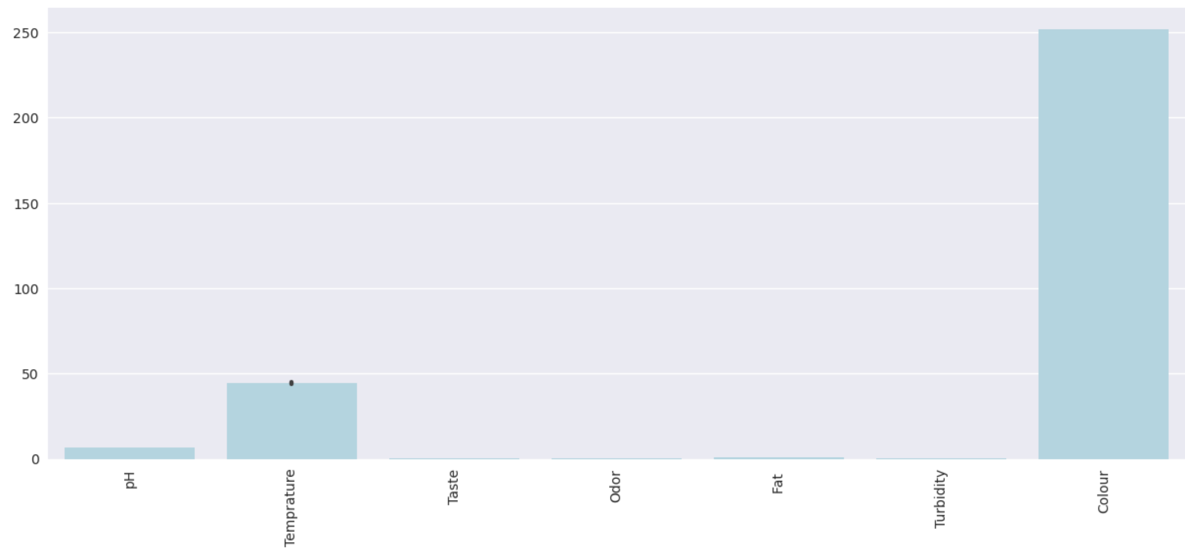


In [12]:
```
plt.style.use("seaborn")
df.hist(figsize=(25,20), bins=15)
```

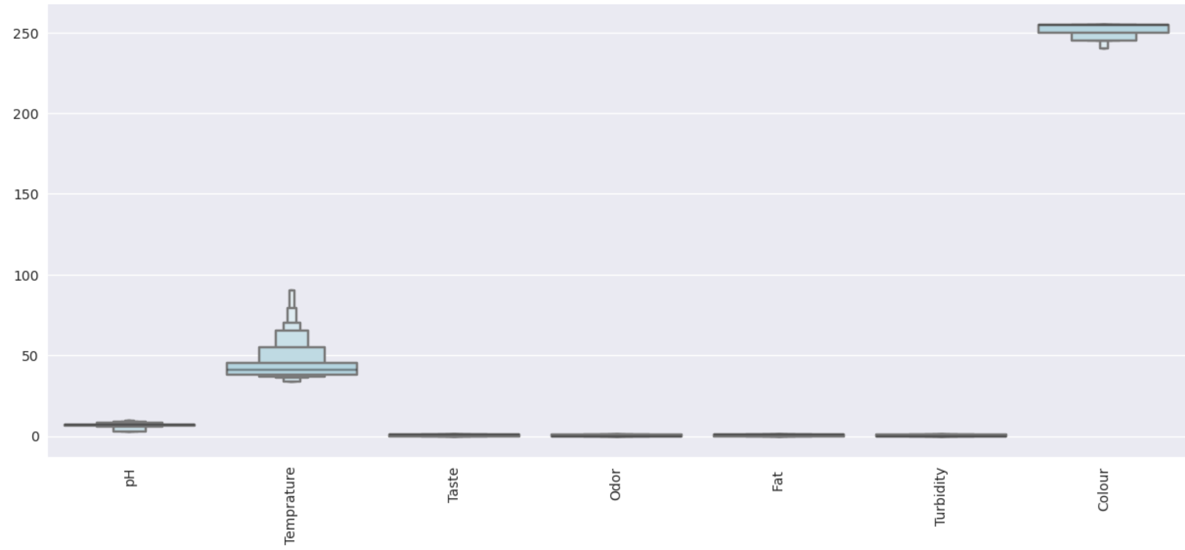
Out[12]:
```
array([[<AxesSubplot:title={'center':'pH'}>,
        <AxesSubplot:title={'center':'Temprature'}>,
        <AxesSubplot:title={'center':'Taste'}>],
       [<AxesSubplot:title={'center':'Odor'}>,
        <AxesSubplot:title={'center':'Fat '}>,
        <AxesSubplot:title={'center':'Turbidity'}>],
       [<AxesSubplot:title={'center':'Colour'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



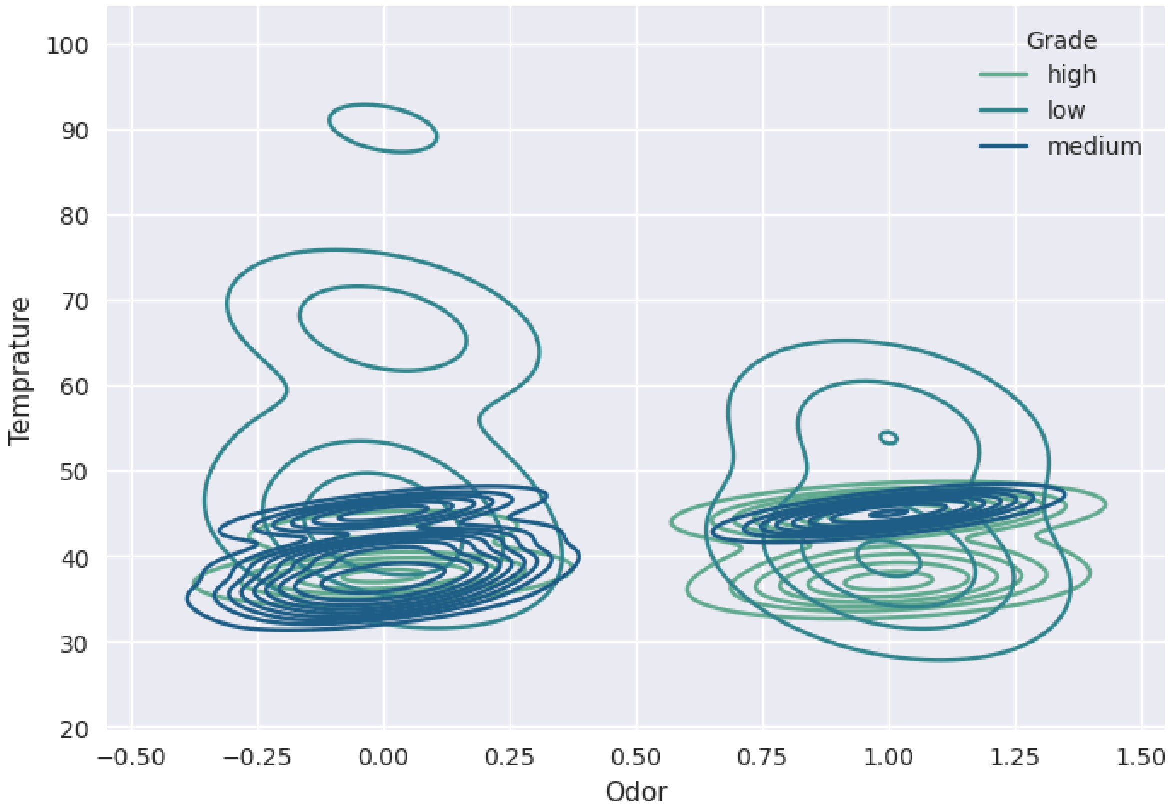
In [13]:
```
plt.figure(figsize=(15,6))
sns.barplot(data = df, color = 'lightblue')
plt.xticks(rotation=90, fontsize=10)
plt.show()
```



In [14]:
```
plt.figure(figsize=(15,6))
sns.boxenplot(data = df, color = 'lightblue')
plt.xticks(rotation=90, fontsize=10)
plt.show()
```



In [15]:
```
sns.kdeplot(x=df["Odor"], y=df["Temprature"], hue =df["Grade"], palette="crest");
plt.show()
```

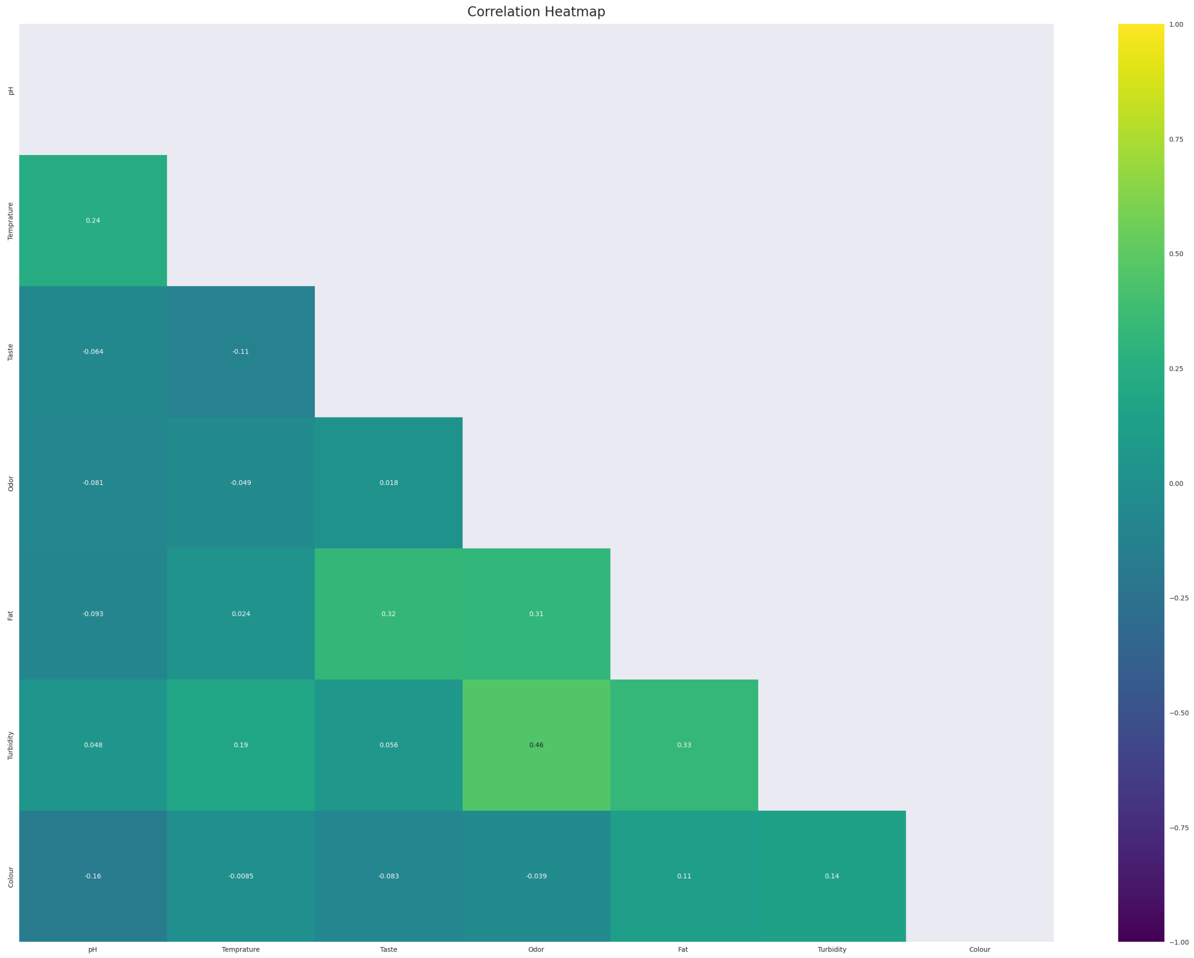


In [16]:
```
sns.pairplot(df,hue='Grade')
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x71c837028550>

In [17]:
```python
plt.figure(figsize=(35, 25))
corr = df.corr()
mask = np.triu(np.ones_like(df.corr(), dtype=np.bool))
heatmap = sns.heatmap(corr, mask = mask, vmin=-1, vmax=1, annot=True, cmap = 'viric
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':20}, pad=12)
```

Out[17]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



In [18]:
```
df['Grade'] =df['Grade'].map({'low': 0, 'medium': 1,'high':2})
```

Data Preparation

In [19]:
```
x=df.drop('Grade',axis=1)
y=df['Grade']
```

Model Building

In [20]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.30, random_state
```

In [21]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

x_train_std = ss.fit_transform(x_train)
x_test_std = ss.transform(x_test)
```

SVM

In [22]:
```
from sklearn import metrics
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.svm import SVC
```

```
svc=SVC()
svc.fit(x_train,y_train)
```

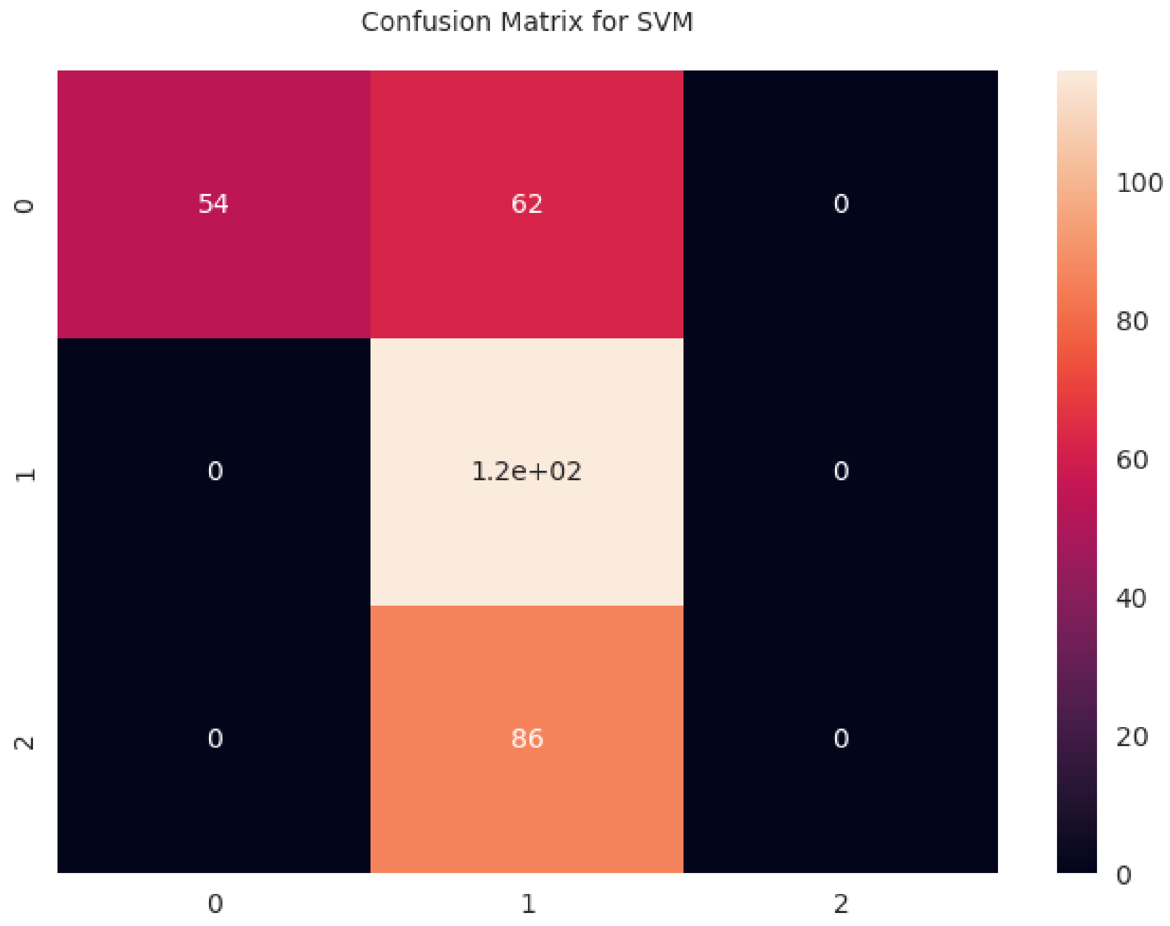
Out[22]: SVC()

In [23]:
```
print("Training Accuracy :",svc.score(x_train,y_train))
print("Testing Accuracy :",svc.score(x_test,y_test))
```

```
Training Accuracy : 0.5479082321187584
Testing Accuracy : 0.5345911949685535
```

In [24]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_svc = svc.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_svc)
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for SVM", fontsize=10, y=1.03)
```

Out[24]: Text(0.5, 1.03, 'Confusion Matrix for SVM')



In [25]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       1.00      0.47      0.64       116
           1       0.44      1.00      0.61       116
           2       0.00      0.00      0.00        86

    accuracy                           0.53       318
   macro avg       0.48      0.49      0.42       318
weighted avg       0.53      0.53      0.45       318
```

Decision Tree

In [26]:
```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(criterion='entropy',random_state=42)
dtc.fit(x_train,y_train)
```

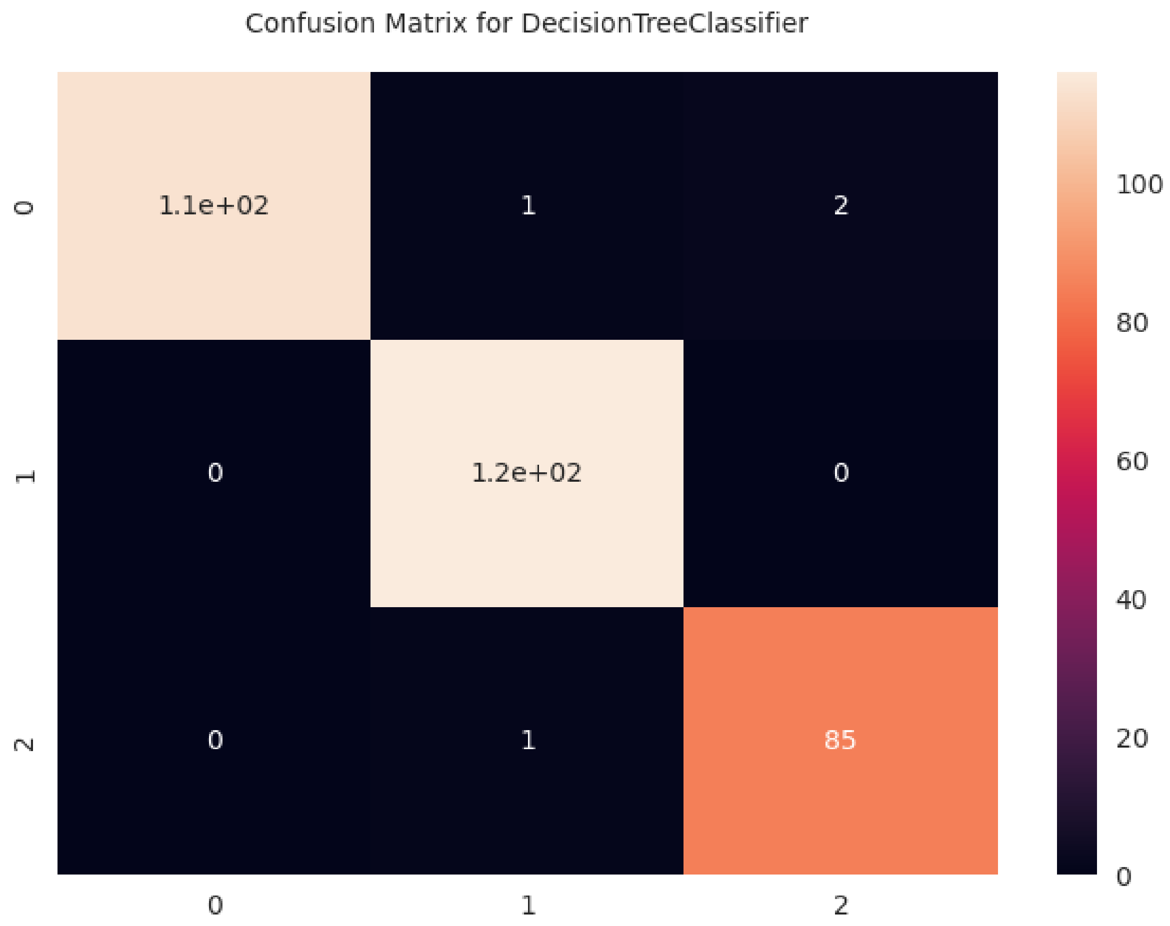
Out[26]:
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

In [27]:
```
print("Training Accuracy :",dtc.score(x_train,y_train))
print("Testing Accuracy :",dtc.score(x_test,y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.9874213836477987
```

In [28]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_dtc = dtc.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_dtc)
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for DecisionTreeClassifier", fontsize=10, y=1.03)
```

Out[28]:
```
Text(0.5, 1.03, 'Confusion Matrix for DecisionTreeClassifier')
```



In [29]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_dtc))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       116
           1       0.98      1.00      0.99       116
           2       0.98      0.99      0.98        86

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318
```

Random Forest

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

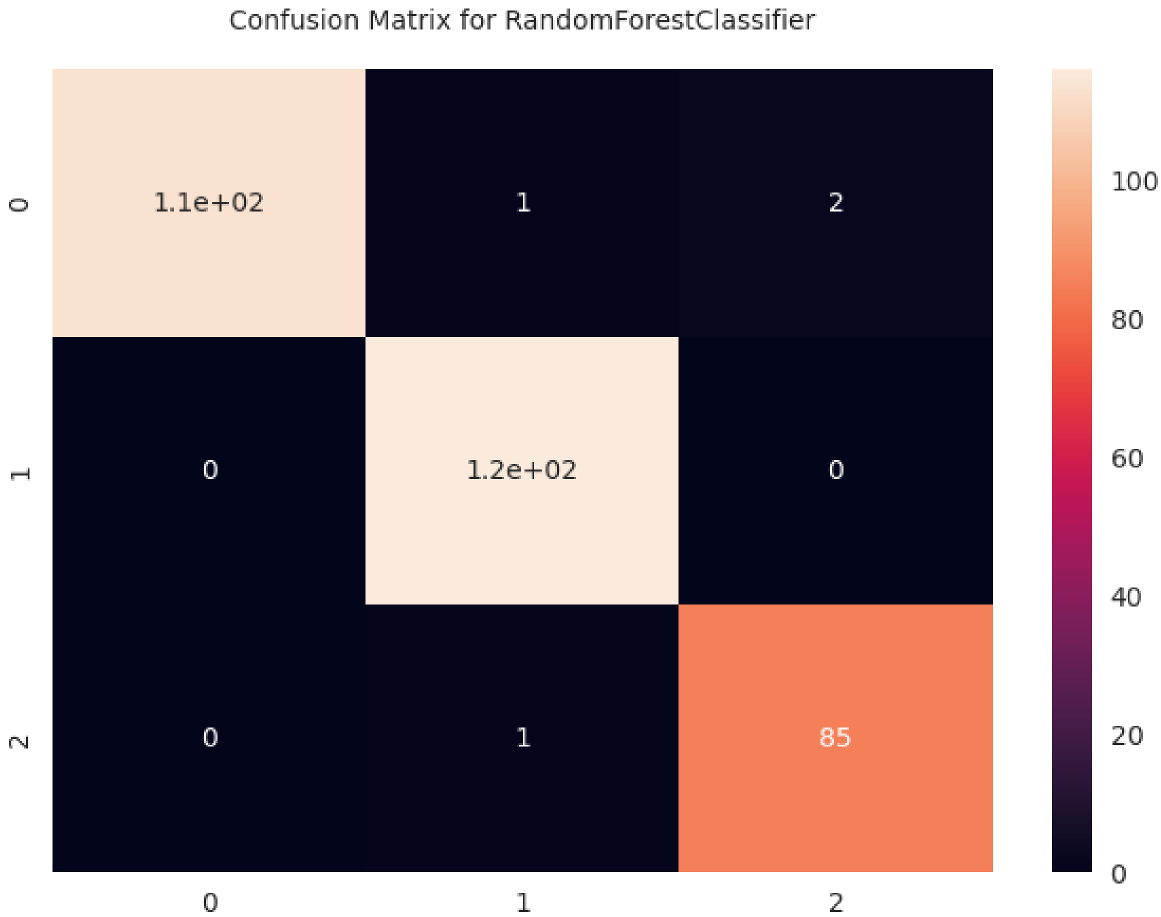
Out[30]:
```
RandomForestClassifier()
```

In [31]:
```
print("Training Accuracy :",rfc.score(x_train,y_train))
print("Testing Accuracy :",rfc.score(x_test,y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.9874213836477987
```

In [32]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_rfc = rfc.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_rfc)
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for RandomForestClassifier", fontsize=10, y=1.03)
```

Out[32]:
```
Text(0.5, 1.03, 'Confusion Matrix for RandomForestClassifier')
```



In [33]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       116
           1       0.98      1.00      0.99       116
           2       0.98      0.99      0.98        86

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318
```

Gaussian Naive Bias

In [34]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
```

Out[34]:
```
GaussianNB()
```

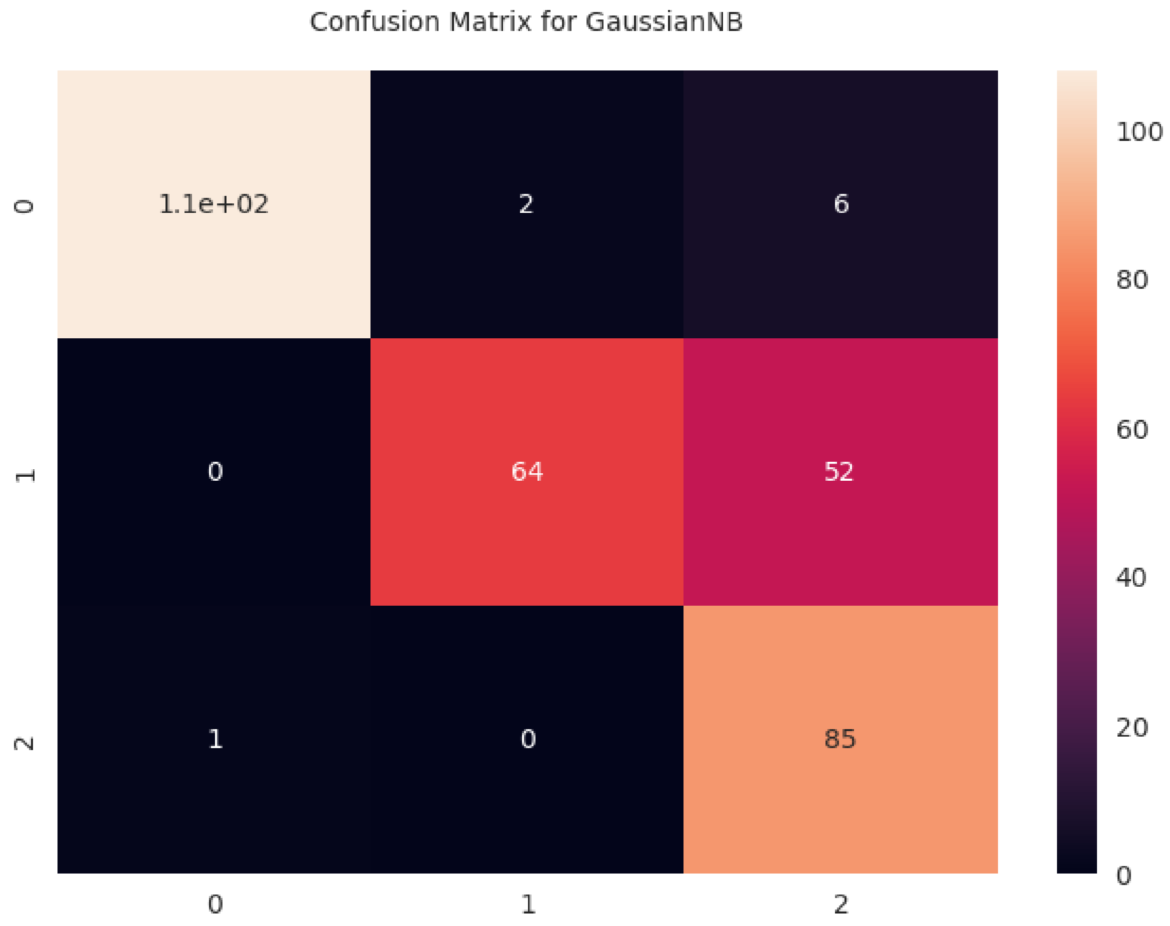
In [35]:
```
print("Training Accuracy :",nb.score(x_train,y_train))
print("Testing Accuracy :",nb.score(x_test,y_test))
```

```
Training Accuracy : 0.8421052631578947
Testing Accuracy : 0.8081761006289309
```

In [36]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_nb = nb.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_nb)
```

```
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for GaussianNB", fontsize=10, y=1.03)
```

Out[36]:
```
Text(0.5, 1.03, 'Confusion Matrix for GaussianNB')
```



In [37]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.99      0.93      0.96       116
           1       0.97      0.55      0.70       116
           2       0.59      0.99      0.74        86

    accuracy                           0.81       318
   macro avg       0.85      0.82      0.80       318
weighted avg       0.88      0.81      0.81       318
```

Linear Regression

In [38]:
```
from sklearn.linear_model import LinearRegression
lnr = LinearRegression()
lnr.fit(x_train, y_train)
```

Out[38]:
```
LinearRegression()
```

In [39]:
```
print("Training Accuracy :",lnr.score(x_train,y_train))
print("Testing Accuracy :",lnr.score(x_test,y_test))
```

```
Training Accuracy : 0.2797660354192826
Testing Accuracy : 0.2543546562708897
```

Logistic Regression

In [40]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
```

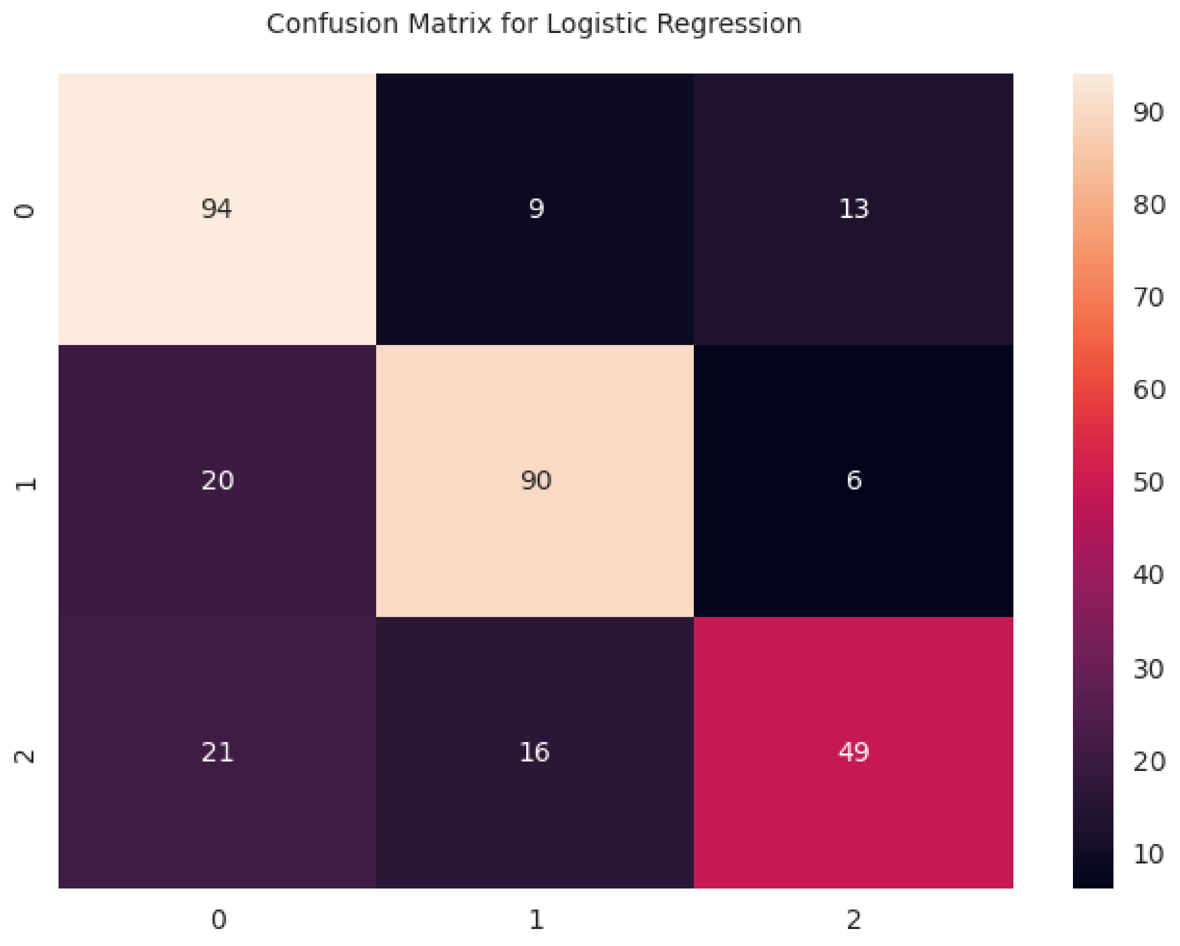
Out[40]:
```
LogisticRegression()
```

In [41]:
```
print("Training Accuracy :",lr.score(x_train,y_train))
print("Testing Accuracy :",lr.score(x_test,y_test))
```

```
Training Accuracy : 0.7462887989203779
Testing Accuracy : 0.7327044025157232
```

In [42]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_lr = lr.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for Logistic Regression", fontsize=10, y=1.03)
```

Out[42]:
```
Text(0.5, 1.03, 'Confusion Matrix for Logistic Regression')
```



In [43]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.70      0.81      0.75       116
           1       0.78      0.78      0.78       116
           2       0.72      0.57      0.64        86

    accuracy                           0.73       318
   macro avg       0.73      0.72      0.72       318
weighted avg       0.73      0.73      0.73       318
```

XgBoost

In [44]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(x_train, y_train)
```

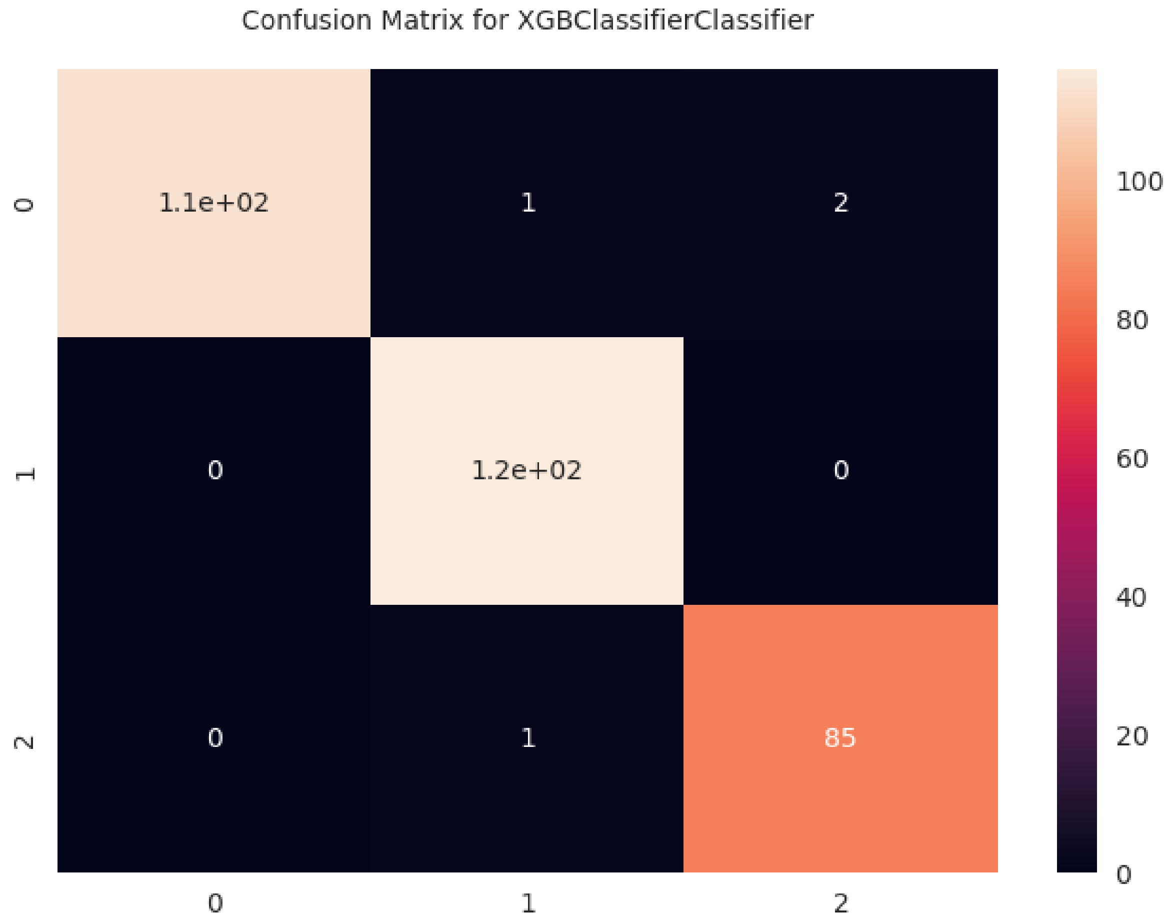
Out[44]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
```

In [45]:
```
print("Training Accuracy :",xgb.score(x_train,y_train))
print("Testing Accuracy :",xgb.score(x_test,y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.9874213836477987
```

In [46]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_xgb = xgb.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_xgb)
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for XGBClassifierClassifier", fontsize=10, y=1.03)
```

Out[46]:
```
Text(0.5, 1.03, 'Confusion Matrix for XGBClassifierClassifier')
```



In [47]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       116
           1       0.98      1.00      0.99       116
           2       0.98      0.99      0.98        86

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318
```

KNeighbors

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
```

Out[48]:
```
KNeighborsClassifier()
```

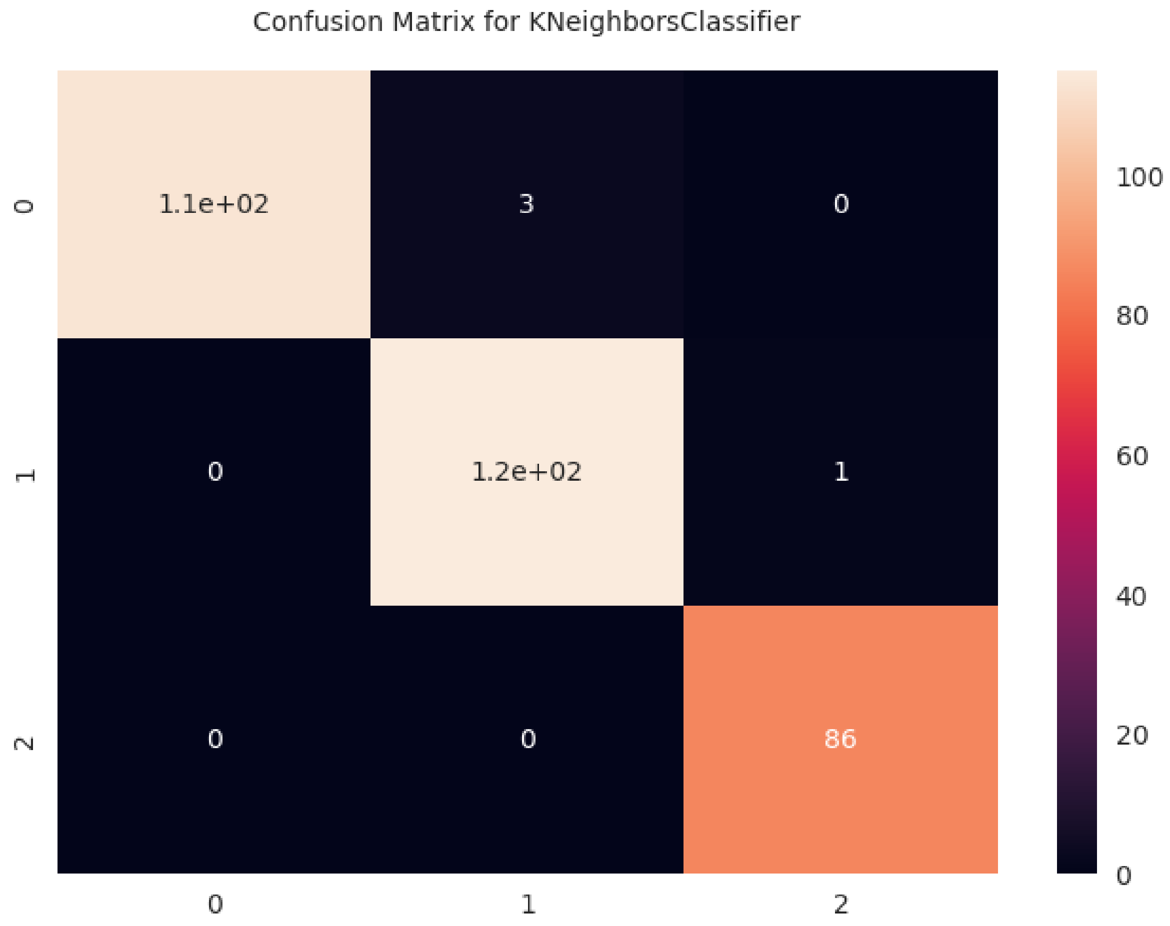
In [49]:
```
print("Training Accuracy :",knn.score(x_train,y_train))
print("Testing Accuracy :",knn.score(x_test,y_test))
```

```
Training Accuracy : 0.9919028340080972
Testing Accuracy : 0.9874213836477987
```

In [50]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_knn = knn.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_knn)
```

```
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for KNeighborsClassifier", fontsize=10, y=1.03)
```

Out[50]:
```
Text(0.5, 1.03, 'Confusion Matrix for KNeighborsClassifier')
```



In [51]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       116
           1       0.97      0.99      0.98       116
           2       0.99      1.00      0.99        86

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318
```

GradientBoosting

In [52]:
```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(x_train, y_train)
```

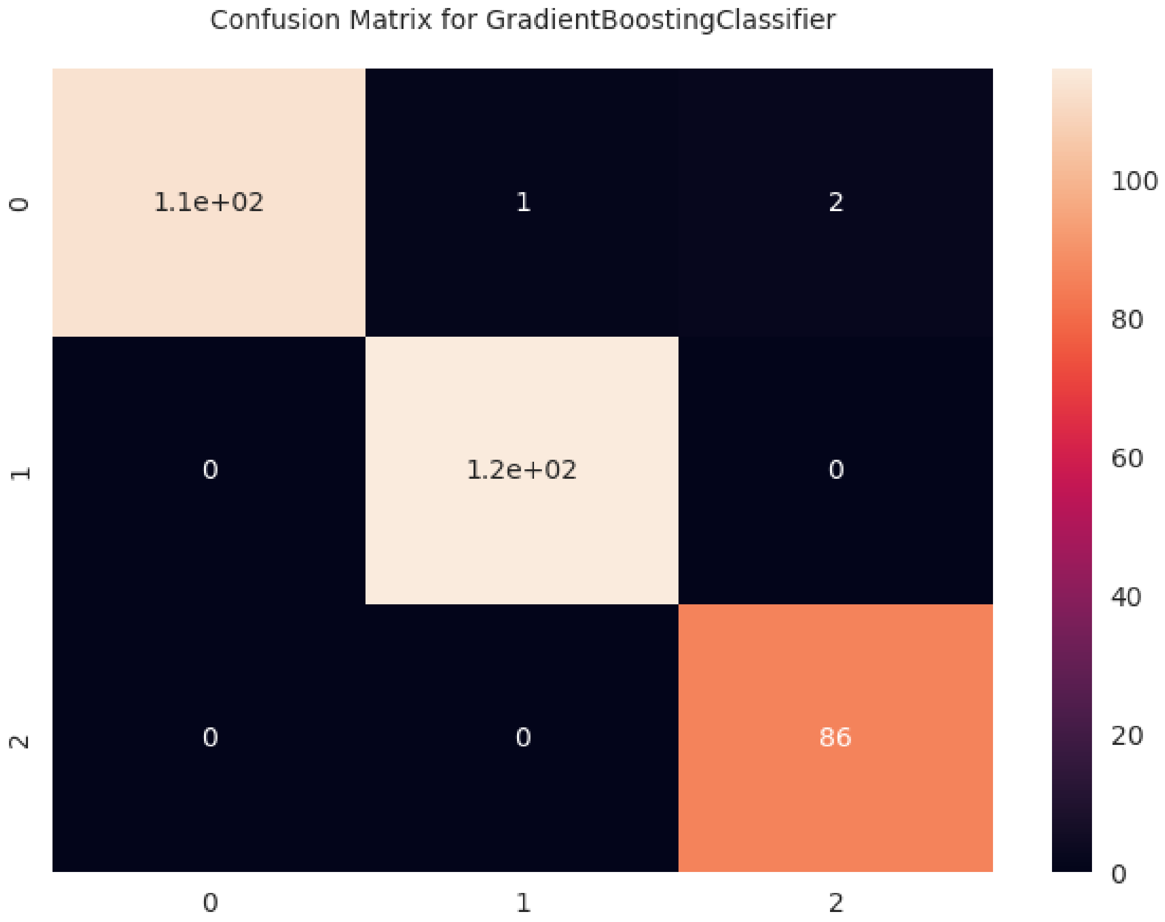
Out[52]:
```
GradientBoostingClassifier()
```

In [53]:
```
print("Training Accuracy :",gb.score(x_train,y_train))
print("Testing Accuracy :",gb.score(x_test,y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.9905660377358491
```

In [54]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_gb = gb.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred_gb)
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for GradientBoostingClassifier", fontsize=10, y=1.03)
```

Out[54]:
```
Text(0.5, 1.03, 'Confusion Matrix for GradientBoostingClassifier')
```



In [55]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_gb))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       116
           1       0.99      1.00      1.00       116
           2       0.98      1.00      0.99        86

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318
```

Artificial neural network

In [56]:
```
# ANN Model Layers
from tensorflow.keras.layers import BatchNormalization
ann_model =Sequential()
```

```
ann_model.add(Dense(units = 32,activation = 'relu'))
ann_model.add(BatchNormalization())
ann_model.add(Dropout(0.5))

ann_model.add(Dense(units = 64,activation = 'relu'))
ann_model.add(BatchNormalization())
ann_model.add(Dropout(0.5))

ann_model.add(Dense(units = 1,activation = 'sigmoid'))

# Model Optimizer
ann_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['acc
```

In [57]:

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)
```

In [58]:

```
# Training the ANN
history = ann_model.fit(x_train, y_train, batch_size=16, epochs=100, validation_dat
```

```
Epoch 1/100
47/47 [==============================] - 2s 8ms/step - loss: 0.9364 - accuracy: 0.
4521 - val_loss: 6.3271 - val_accuracy: 0.3648
Epoch 2/100
47/47 [==============================] - 0s 3ms/step - loss: 0.4443 - accuracy: 0.
5412 - val_loss: 1.0002 - val_accuracy: 0.4528
Epoch 3/100
47/47 [==============================] - 0s 3ms/step - loss: 0.0975 - accuracy: 0.
5776 - val_loss: 0.7874 - val_accuracy: 0.4686
Epoch 4/100
47/47 [==============================] - 0s 4ms/step - loss: 0.0673 - accuracy: 0.
5344 - val_loss: -0.0601 - val_accuracy: 0.5252
Epoch 5/100
47/47 [==============================] - 0s 3ms/step - loss: -0.0162 - accuracy:
0.5574 - val_loss: 1.2288 - val_accuracy: 0.4088
Epoch 6/100
47/47 [==============================] - 0s 4ms/step - loss: -0.1415 - accuracy:
0.5762 - val_loss: 4.8013 - val_accuracy: 0.3648
Epoch 7/100
47/47 [==============================] - 0s 5ms/step - loss: -0.3351 - accuracy:
0.5803 - val_loss: 1.4062 - val_accuracy: 0.3742
Epoch 8/100
47/47 [==============================] - 0s 3ms/step - loss: -0.4995 - accuracy:
0.5830 - val_loss: 1.0389 - val_accuracy: 0.4119
Epoch 9/100
47/47 [==============================] - 0s 3ms/step - loss: -0.6944 - accuracy:
0.5843 - val_loss: -0.2170 - val_accuracy: 0.5629
Epoch 10/100
47/47 [==============================] - 0s 3ms/step - loss: -0.8488 - accuracy:
0.5870 - val_loss: -0.0185 - val_accuracy: 0.5597
Epoch 11/100
47/47 [==============================] - 0s 3ms/step - loss: -0.8800 - accuracy:
0.5695 - val_loss: -0.8343 - val_accuracy: 0.4874
Epoch 12/100
47/47 [==============================] - 0s 4ms/step - loss: -1.2688 - accuracy:
0.5843 - val_loss: -0.7517 - val_accuracy: 0.5126
Epoch 13/100
47/47 [==============================] - 0s 4ms/step - loss: -1.3348 - accuracy:
0.5884 - val_loss: 4.9570 - val_accuracy: 0.2799
Epoch 14/100
47/47 [==============================] - 0s 3ms/step - loss: -1.7121 - accuracy:
0.5735 - val_loss: 7.0937 - val_accuracy: 0.3648
Epoch 15/100
47/47 [==============================] - 0s 4ms/step - loss: -2.0444 - accuracy:
0.5816 - val_loss: 6.9729 - val_accuracy: 0.3648
Epoch 16/100
47/47 [==============================] - 0s 4ms/step - loss: -2.3847 - accuracy:
0.5951 - val_loss: 12.9911 - val_accuracy: 0.3648
Epoch 17/100
47/47 [==============================] - 0s 4ms/step - loss: -2.7639 - accuracy:
0.5789 - val_loss: 2.9185 - val_accuracy: 0.4057
Epoch 18/100
47/47 [==============================] - 0s 4ms/step - loss: -2.8672 - accuracy:
0.5695 - val_loss: -1.0519 - val_accuracy: 0.6635
Epoch 19/100
47/47 [==============================] - 0s 4ms/step - loss: -3.5181 - accuracy:
0.5830 - val_loss: 0.0758 - val_accuracy: 0.4654
Epoch 20/100
47/47 [==============================] - 0s 4ms/step - loss: -3.7408 - accuracy:
0.5682 - val_loss: -1.5211 - val_accuracy: 0.6006
Epoch 21/100
```

```
47/47 [==============================] - 0s 3ms/step - loss: -4.0925 - accuracy: 
0.5655 - val_loss: 0.6326 - val_accuracy: 0.4434
Epoch 22/100
47/47 [==============================] - 0s 3ms/step - loss: -5.1065 - accuracy: 
0.5506 - val_loss: -1.1901 - val_accuracy: 0.5597
Epoch 23/100
47/47 [==============================] - 0s 4ms/step - loss: -5.2625 - accuracy: 
0.5628 - val_loss: -2.6385 - val_accuracy: 0.6069
Epoch 24/100
47/47 [==============================] - 0s 3ms/step - loss: -5.7392 - accuracy: 
0.5776 - val_loss: -3.2064 - val_accuracy: 0.5629
Epoch 25/100
47/47 [==============================] - 0s 4ms/step - loss: -5.7975 - accuracy: 
0.5493 - val_loss: -6.1050 - val_accuracy: 0.5157
Epoch 26/100
47/47 [==============================] - 0s 3ms/step - loss: -6.4957 - accuracy: 
0.5601 - val_loss: -6.2278 - val_accuracy: 0.4874
Epoch 27/100
47/47 [==============================] - 0s 4ms/step - loss: -7.2969 - accuracy: 
0.5628 - val_loss: -6.4301 - val_accuracy: 0.5283
Epoch 28/100
47/47 [==============================] - 0s 4ms/step - loss: -7.9232 - accuracy: 
0.5560 - val_loss: -5.6027 - val_accuracy: 0.4308
Epoch 29/100
47/47 [==============================] - 0s 4ms/step - loss: -8.0299 - accuracy: 
0.5466 - val_loss: -4.8958 - val_accuracy: 0.3994
Epoch 30/100
47/47 [==============================] - 0s 4ms/step - loss: -7.6518 - accuracy: 
0.5371 - val_loss: -5.8275 - val_accuracy: 0.4528
Epoch 31/100
47/47 [==============================] - 0s 4ms/step - loss: -8.2209 - accuracy: 
0.5452 - val_loss: -5.5315 - val_accuracy: 0.3648
Epoch 32/100
47/47 [==============================] - 0s 3ms/step - loss: -9.4250 - accuracy: 
0.5439 - val_loss: -9.9199 - val_accuracy: 0.4528
Epoch 33/100
47/47 [==============================] - 0s 4ms/step - loss: -9.1802 - accuracy: 
0.5520 - val_loss: -10.9591 - val_accuracy: 0.4465
Epoch 34/100
47/47 [==============================] - 0s 4ms/step - loss: -11.6490 - accuracy: 
0.5439 - val_loss: -15.8427 - val_accuracy: 0.4623
Epoch 35/100
47/47 [==============================] - 0s 4ms/step - loss: -11.1387 - accuracy: 
0.5385 - val_loss: -9.5735 - val_accuracy: 0.6384
Epoch 36/100
47/47 [==============================] - 0s 4ms/step - loss: -13.5299 - accuracy: 
0.5587 - val_loss: 10.5220 - val_accuracy: 0.3553
Epoch 37/100
47/47 [==============================] - 0s 3ms/step - loss: -15.1275 - accuracy: 
0.5479 - val_loss: -18.1456 - val_accuracy: 0.4497
Epoch 38/100
47/47 [==============================] - 0s 3ms/step - loss: -14.3272 - accuracy: 
0.5479 - val_loss: -20.7327 - val_accuracy: 0.5912
Epoch 39/100
47/47 [==============================] - 0s 4ms/step - loss: -14.6484 - accuracy: 
0.5520 - val_loss: 0.2331 - val_accuracy: 0.3994
Epoch 40/100
47/47 [==============================] - 0s 4ms/step - loss: -15.2954 - accuracy: 
0.5506 - val_loss: 9.7962 - val_accuracy: 0.3931
Epoch 41/100
47/47 [==============================] - 0s 4ms/step - loss: -17.3441 - accuracy: 
```

```
0.5452 - val_loss: 4.0273 - val_accuracy: 0.4057
Epoch 42/100
47/47 [==============================] - 0s 3ms/step - loss: -17.4842 - accuracy: 
0.5385 - val_loss: -12.3480 - val_accuracy: 0.5975
Epoch 43/100
47/47 [==============================] - 0s 4ms/step - loss: -17.4094 - accuracy: 
0.5709 - val_loss: -9.5445 - val_accuracy: 0.5220
Epoch 44/100
47/47 [==============================] - 0s 3ms/step - loss: -20.2418 - accuracy: 
0.5560 - val_loss: -34.2692 - val_accuracy: 0.4874
Epoch 45/100
47/47 [==============================] - 0s 4ms/step - loss: -20.9230 - accuracy: 
0.5601 - val_loss: -22.2479 - val_accuracy: 0.5975
Epoch 46/100
47/47 [==============================] - 0s 3ms/step - loss: -22.7544 - accuracy: 
0.5668 - val_loss: -29.6942 - val_accuracy: 0.5440
Epoch 47/100
47/47 [==============================] - 0s 3ms/step - loss: -24.4336 - accuracy: 
0.5506 - val_loss: -33.8989 - val_accuracy: 0.4277
Epoch 48/100
47/47 [==============================] - 0s 3ms/step - loss: -22.6367 - accuracy: 
0.5425 - val_loss: -35.0027 - val_accuracy: 0.3774
Epoch 49/100
47/47 [==============================] - 0s 3ms/step - loss: -25.2419 - accuracy: 
0.5425 - val_loss: -23.7054 - val_accuracy: 0.5220
Epoch 50/100
47/47 [==============================] - 0s 4ms/step - loss: -29.5448 - accuracy: 
0.5614 - val_loss: -36.6991 - val_accuracy: 0.4497
Epoch 51/100
47/47 [==============================] - 0s 3ms/step - loss: -28.5572 - accuracy: 
0.5479 - val_loss: -23.5705 - val_accuracy: 0.3774
Epoch 52/100
47/47 [==============================] - 0s 3ms/step - loss: -29.2854 - accuracy: 
0.5479 - val_loss: -36.7628 - val_accuracy: 0.6132
Epoch 53/100
47/47 [==============================] - 0s 3ms/step - loss: -30.0023 - accuracy: 
0.5358 - val_loss: 37.9504 - val_accuracy: 0.3113
Epoch 54/100
47/47 [==============================] - 0s 4ms/step - loss: -31.3725 - accuracy: 
0.5398 - val_loss: -16.2135 - val_accuracy: 0.5031
Epoch 55/100
47/47 [==============================] - 0s 3ms/step - loss: -33.4582 - accuracy: 
0.5439 - val_loss: -29.0869 - val_accuracy: 0.6069
Epoch 56/100
47/47 [==============================] - 0s 3ms/step - loss: -36.7839 - accuracy: 
0.5668 - val_loss: -58.7504 - val_accuracy: 0.4277
Epoch 57/100
47/47 [==============================] - 0s 4ms/step - loss: -32.0817 - accuracy: 
0.5344 - val_loss: -46.9754 - val_accuracy: 0.6069
Epoch 58/100
47/47 [==============================] - 0s 3ms/step - loss: -36.9227 - accuracy: 
0.5574 - val_loss: -39.8727 - val_accuracy: 0.4906
Epoch 59/100
47/47 [==============================] - 0s 3ms/step - loss: -41.1633 - accuracy: 
0.5479 - val_loss: -43.5720 - val_accuracy: 0.4780
Epoch 60/100
47/47 [==============================] - 0s 4ms/step - loss: -38.3367 - accuracy: 
0.5466 - val_loss: -54.9898 - val_accuracy: 0.4497
Epoch 61/100
47/47 [==============================] - 0s 3ms/step - loss: -43.9839 - accuracy: 
0.5398 - val_loss: -11.2951 - val_accuracy: 0.5566
```

```
Epoch 62/100
47/47 [==============================] - 0s 3ms/step - loss: -43.1137 - accuracy:
0.5385 - val_loss: 49.5336 - val_accuracy: 0.3648
Epoch 63/100
47/47 [==============================] - 0s 3ms/step - loss: -44.0907 - accuracy:
0.5412 - val_loss: -66.1592 - val_accuracy: 0.4874
Epoch 64/100
47/47 [==============================] - 0s 3ms/step - loss: -42.1352 - accuracy:
0.5439 - val_loss: -49.3381 - val_accuracy: 0.5535
Epoch 65/100
47/47 [==============================] - 0s 3ms/step - loss: -45.9768 - accuracy:
0.5547 - val_loss: -6.1029 - val_accuracy: 0.5377
Epoch 66/100
47/47 [==============================] - 0s 3ms/step - loss: -51.6089 - accuracy:
0.5601 - val_loss: -78.2757 - val_accuracy: 0.4937
Epoch 67/100
47/47 [==============================] - 0s 4ms/step - loss: -49.7343 - accuracy:
0.5506 - val_loss: -69.9387 - val_accuracy: 0.5472
Epoch 68/100
47/47 [==============================] - 0s 3ms/step - loss: -43.6494 - accuracy:
0.5439 - val_loss: -51.1160 - val_accuracy: 0.6069
Epoch 69/100
47/47 [==============================] - 0s 4ms/step - loss: -50.6234 - accuracy:
0.5250 - val_loss: -67.5859 - val_accuracy: 0.4245
Epoch 70/100
47/47 [==============================] - 0s 3ms/step - loss: -49.0533 - accuracy:
0.5371 - val_loss: -77.0359 - val_accuracy: 0.4151
Epoch 71/100
47/47 [==============================] - 0s 3ms/step - loss: -54.8261 - accuracy:
0.5425 - val_loss: -81.6451 - val_accuracy: 0.4119
Epoch 72/100
47/47 [==============================] - 0s 3ms/step - loss: -54.1311 - accuracy:
0.5439 - val_loss: -45.4728 - val_accuracy: 0.5409
Epoch 73/100
47/47 [==============================] - 0s 4ms/step - loss: -58.9882 - accuracy:
0.5560 - val_loss: 32.9783 - val_accuracy: 0.3302
Epoch 74/100
47/47 [==============================] - 0s 3ms/step - loss: -56.2245 - accuracy:
0.5547 - val_loss: -106.2492 - val_accuracy: 0.4717
Epoch 75/100
47/47 [==============================] - 0s 4ms/step - loss: -53.6323 - accuracy:
0.5479 - val_loss: -67.5443 - val_accuracy: 0.3648
Epoch 76/100
47/47 [==============================] - 0s 3ms/step - loss: -54.8622 - accuracy:
0.5398 - val_loss: -54.1232 - val_accuracy: 0.6164
Epoch 77/100
47/47 [==============================] - 0s 3ms/step - loss: -59.2817 - accuracy:
0.5277 - val_loss: -72.1177 - val_accuracy: 0.3648
Epoch 78/100
47/47 [==============================] - 0s 4ms/step - loss: -60.6974 - accuracy:
0.5290 - val_loss: -95.8444 - val_accuracy: 0.4780
Epoch 79/100
47/47 [==============================] - 0s 4ms/step - loss: -58.1961 - accuracy:
0.5290 - val_loss: -115.4964 - val_accuracy: 0.4811
Epoch 80/100
47/47 [==============================] - 0s 4ms/step - loss: -65.8021 - accuracy:
0.5425 - val_loss: -108.7465 - val_accuracy: 0.4025
Epoch 81/100
47/47 [==============================] - 0s 3ms/step - loss: -69.1130 - accuracy:
0.5398 - val_loss: -96.1030 - val_accuracy: 0.4025
Epoch 82/100
```

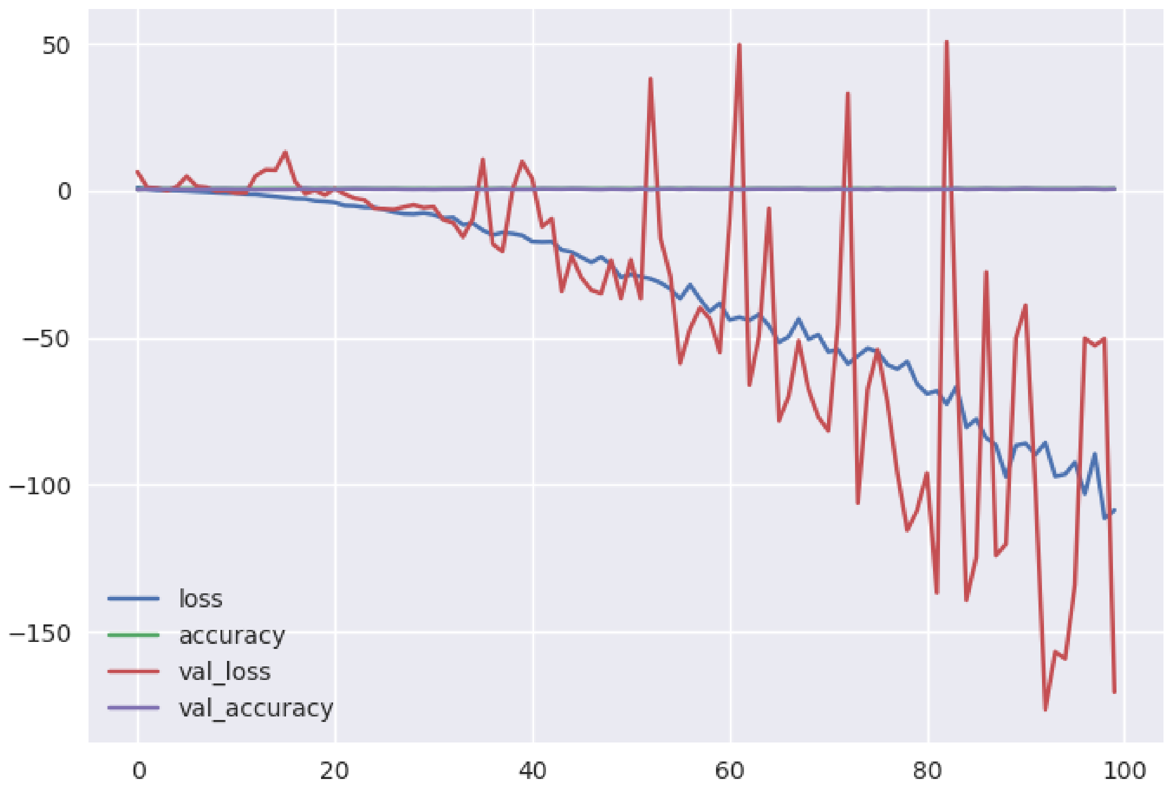
```
47/47 [==============================] - 0s 3ms/step - loss: -68.1573 - accuracy:
0.5398 - val_loss: -136.7848 - val_accuracy: 0.4686
Epoch 83/100
47/47 [==============================] - 0s 4ms/step - loss: -72.5733 - accuracy:
0.5520 - val_loss: 50.5330 - val_accuracy: 0.3994
Epoch 84/100
47/47 [==============================] - 0s 3ms/step - loss: -66.6355 - accuracy:
0.5385 - val_loss: -54.6375 - val_accuracy: 0.6541
Epoch 85/100
47/47 [==============================] - 0s 3ms/step - loss: -80.4892 - accuracy:
0.5385 - val_loss: -139.3182 - val_accuracy: 0.4371
Epoch 86/100
47/47 [==============================] - 0s 3ms/step - loss: -77.7431 - accuracy:
0.5344 - val_loss: -124.6676 - val_accuracy: 0.4874
Epoch 87/100
47/47 [==============================] - 0s 4ms/step - loss: -84.1267 - accuracy:
0.5506 - val_loss: -27.7219 - val_accuracy: 0.5943
Epoch 88/100
47/47 [==============================] - 0s 4ms/step - loss: -86.5639 - accuracy:
0.5506 - val_loss: -124.0542 - val_accuracy: 0.4906
Epoch 89/100
47/47 [==============================] - 0s 4ms/step - loss: -97.3375 - accuracy:
0.5533 - val_loss: -120.2535 - val_accuracy: 0.4403
Epoch 90/100
47/47 [==============================] - 0s 3ms/step - loss: -86.7052 - accuracy:
0.5344 - val_loss: -50.2680 - val_accuracy: 0.5786
Epoch 91/100
47/47 [==============================] - 0s 3ms/step - loss: -85.9945 - accuracy:
0.5385 - val_loss: -39.0791 - val_accuracy: 0.6792
Epoch 92/100
47/47 [==============================] - 0s 4ms/step - loss: -89.6942 - accuracy:
0.5466 - val_loss: -102.0823 - val_accuracy: 0.5346
Epoch 93/100
47/47 [==============================] - 0s 4ms/step - loss: -85.7191 - accuracy:
0.5547 - val_loss: -176.4807 - val_accuracy: 0.4591
Epoch 94/100
47/47 [==============================] - 0s 3ms/step - loss: -97.1916 - accuracy:
0.5439 - val_loss: -156.8604 - val_accuracy: 0.4811
Epoch 95/100
47/47 [==============================] - 0s 4ms/step - loss: -96.4436 - accuracy:
0.5452 - val_loss: -159.1734 - val_accuracy: 0.4717
Epoch 96/100
47/47 [==============================] - 0s 4ms/step - loss: -92.4171 - accuracy:
0.5331 - val_loss: -134.0214 - val_accuracy: 0.4811
Epoch 97/100
47/47 [==============================] - 0s 4ms/step - loss: -103.1512 - accuracy:
0.5547 - val_loss: -50.2631 - val_accuracy: 0.6132
Epoch 98/100
47/47 [==============================] - 0s 3ms/step - loss: -89.5648 - accuracy:
0.5304 - val_loss: -52.7098 - val_accuracy: 0.5597
Epoch 99/100
47/47 [==============================] - 0s 3ms/step - loss: -111.4318 - accuracy:
0.5466 - val_loss: -50.3987 - val_accuracy: 0.4088
Epoch 100/100
47/47 [==============================] - 0s 3ms/step - loss: -108.6408 - accuracy:
0.5506 - val_loss: -170.4726 - val_accuracy: 0.5377
```

In [59]:
```
ann_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                256

 batch_normalization (BatchN  (None, 32)               128
 ormalization)

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 64)                2112

 batch_normalization_1 (Batc  (None, 64)               256
 hNormalization)

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,817
Trainable params: 2,625
Non-trainable params: 192
_________________________________________________________________
```

In [60]:
```
loss_plot = pd.DataFrame(ann_model.history.history)
loss_plot.plot()
```

Out[60]: <AxesSubplot:>



In [61]:
```
#now testing for Test data
y_pred = ann_model.predict(x_test)
y_pred = (y_pred>0.5)
acc_test_ann1 = round(metrics.accuracy_score(y_test,y_pred) * 100, 2)
acc_test_ann1
```

```
10/10 [==============================] - 0s 1ms/step
```

Out[61]:

```
53.77
```

In [62]:

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
cm = confusion_matrix(y_test,y_pred)
acc_test_ann1 = accuracy_score(y_test,y_pred)
print(cm)
print('score is:',acc_test_ann1)
```

```
[[ 65  51   0]
 [ 10 106   0]
 [  0  86   0]]
score is: 0.5377358490566038
```

In [63]:

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

```
              precision    recall  f1-score   support

           0       0.87      0.56      0.68       116
           1       0.44      0.91      0.59       116
           2       0.00      0.00      0.00        86

    accuracy                           0.54       318
   macro avg       0.43      0.49      0.42       318
weighted avg       0.48      0.54      0.46       318
```

Prediction

In [64]:

```python
print(xgb.predict(x_test))
```

```
[2 0 1 0 2 1 0 2 2 1 0 2 1 1 2 1 2 2 2 0 2 0 1 2 2 0 2 2 0 1 2 0 0 1 2 2 0
 1 2 0 2 1 2 0 1 0 2 0 1 0 2 1 1 1 0 1 1 0 1 0 1 0 1 2 0 1 0 1 1 2 2 1 1 2
 1 1 1 0 2 1 0 1 1 0 1 1 2 0 2 0 1 2 1 0 2 1 0 2 1 1 2 1 0 1 2 2 2 2 0 0 2
 2 2 2 0 1 0 2 1 2 1 1 0 2 2 0 0 0 1 2 0 1 0 1 2 1 1 1 1 0 1 2 0 0 1 2 0 0
 0 2 0 1 1 0 2 2 1 1 1 1 1 1 0 1 1 2 2 0 2 1 0 0 1 2 2 0 2 2 2 1 1 0 0 0 0
 2 1 2 0 0 2 0 0 0 1 0 1 1 2 1 0 2 1 0 1 1 0 1 1 0 2 1 1 1 1 1 2 1 2 0 2 2
 1 2 1 0 0 1 2 2 2 0 0 2 0 1 0 0 0 2 0 2 0 1 2 0 1 1 0 0 1 0 2 0 0 0 2 0 2
 0 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 2 1 0 0 1 1 0 2 2 0 1 2 1 1 1 0 1 0 0 0 0
 0 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 0 2 0 2 0]
```

In [ ]: