Voice classsification from numerical attributes/features

Problem statement:

Create a classification model to predict the gender (male or female) based on different acoustic parameters

Context:

This database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers. The voice samples are preprocessed by acoustic analysis in R using the seewave and tuneR packages, with an analyzed frequency range of 0hz-280hz (human vocal range).

Contents

Importing necessary libraries 📚

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.model_selection, sklearn.linear_model, sklearn.svm, sklear
n.metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.svm import SVC
```

Importing the dataset ⬇️

In [2]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/voicegender/voice.csv
```

In [3]:
```
df = pd.read_csv('/kaggle/input/voicegender/voice.csv')
```

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

Out[4]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.4029
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.6138
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713

5 rows × 21 columns

In [5]:

```
df.describe()
```

Out[5]:

	meanfreq	sd	median	Q25	Q75	IQR
count	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.00000
mean	0.180907	0.057126	0.185621	0.140456	0.224765	0.084309
std	0.029918	0.016652	0.036360	0.048680	0.023639	0.042783
min	0.039363	0.018363	0.010975	0.000229	0.042946	0.014558
25%	0.163662	0.041954	0.169593	0.111087	0.208747	0.042560
50%	0.184838	0.059155	0.190032	0.140286	0.225684	0.094280
75%	0.199146	0.067020	0.210618	0.175939	0.243660	0.114175
max	0.251124	0.115273	0.261224	0.247347	0.273469	0.252225

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object 
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

Data Pre-processing ⏳

Checking for Null values

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
meanfreq    0
sd          0
median      0
Q25         0
Q75         0
IQR         0
skew        0
kurt        0
sp.ent      0
sfm         0
mode        0
centroid    0
meanfun     0
minfun      0
maxfun      0
meandom     0
mindom      0
maxdom      0
dfrange     0
modindx     0
label       0
dtype: int64
```

Encoding labels from string to boolean value

In [8]:
```
df.replace(to_replace="male", value=1, inplace=True)
df.replace(to_replace="female", value=0, inplace=True)
df.label.unique()
```

Out[8]:
```
array([1, 0])
```

Splitting the data for training and testing

In [9]:
```
xData=df.iloc[:,:-1]
yData=df.iloc[:,-1]
xData.shape, yData.shape
```

Out[9]:
```
((3168, 20), (3168,))
```

In [10]:
```
TRAINSPLIT = 0.8

xTrain, xTest, yTrain, yTest = sklearn.model_selection.train_test_split(x
Data, yData, train_size=TRAINSPLIT)
xTrain.shape, yTrain.shape
```

Out[10]:
```
((2534, 20), (2534,))
```

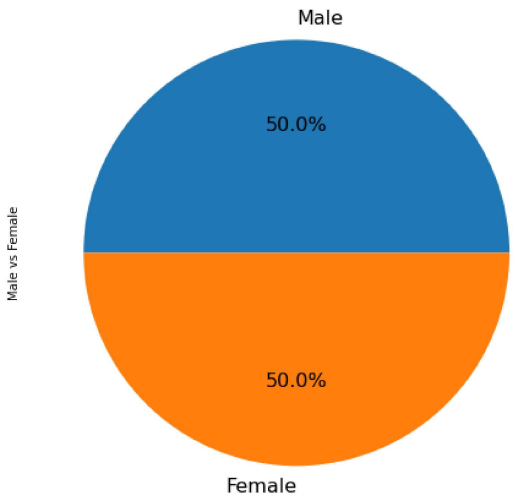
Exploratory Data Analysis 📊

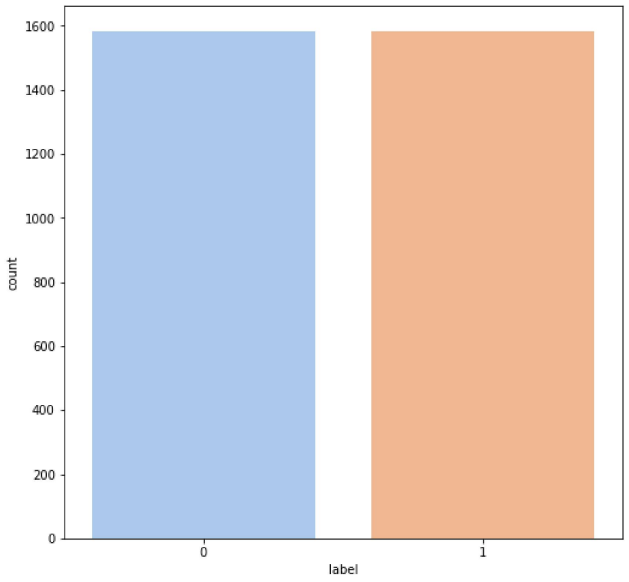
Distribution of labels

In [11]:

```
plt.figure(figsize=(18, 8))
plt.subplot(1, 2, 1)
df.label.value_counts().plot(kind="pie",
                                                fontsize=16,
                                                labels=["Male", "Female"],
                                                ylabel="Male vs Female",
                                                autopct='%1.1f%%');

plt.subplot(1, 2, 2)
sns.countplot(x="label",data=df, palette="pastel")
plt.show()
```





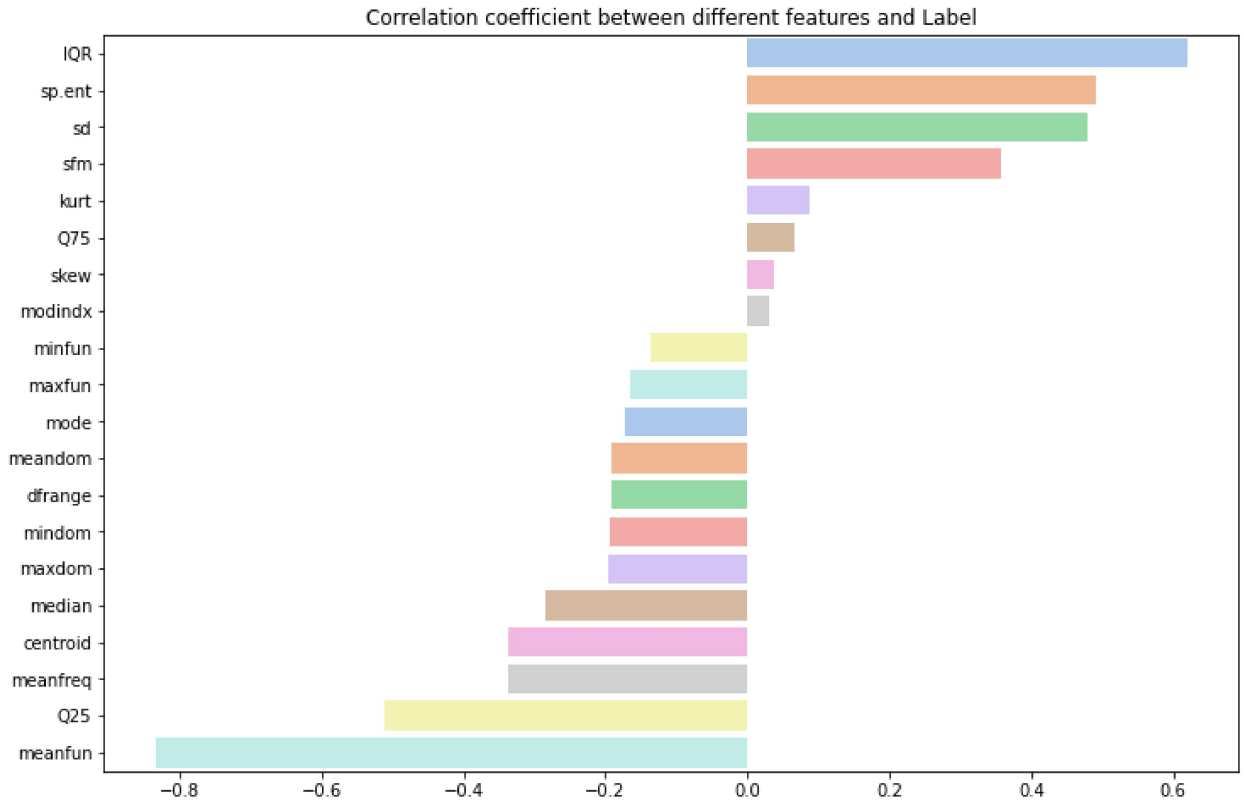
Correlation between different features and our target variable

Correlation coefficient

The correlation coefficient is a statistical measure of the strength of the relationship between the relative movements of two variables. The values range between -1.0 and 1.0. A calculated number greater than 1.0 or less than -1.0 means that there was an error in the correlation measurement. A correlation of -1.0 shows a perfect negative correlation, while a correlation of 1.0 shows a perfect positive correlation. A correlation of 0.0 shows no linear relationship between the movement of the two variables.

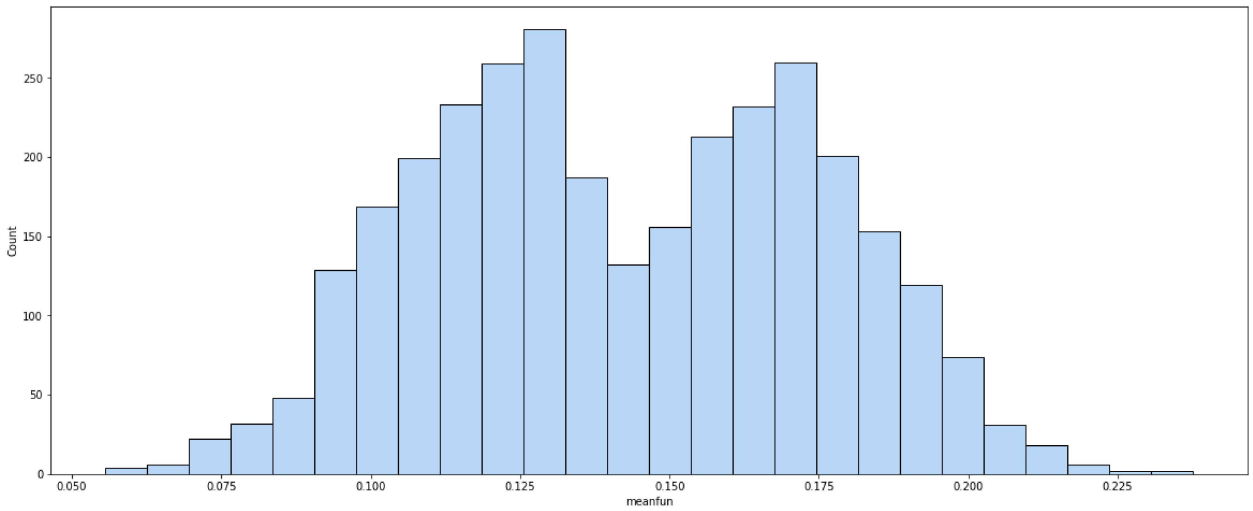
In [12]:

```
plt.figure(figsize=(12,8))
data = df.corr()["label"].sort_values(ascending=False)
indices = data.index
labels = []
corr = []
for i in range(1, len(indices)):
    labels.append(indices[i])
    corr.append(data[i])
sns.barplot(x=corr, y=labels, palette='pastel')
plt.title('Correlation coefficient between different features and Label')
plt.show()
```



Distribution of meanfun in our data

In [13]:
```
plt.figure(figsize=(20,8))
sns.histplot(df.meanfun, color=sns.color_palette('pastel')[0])
plt.show()
```



Classifying 🔃

Building and fitting the models ⚗️

Logistic Regression

Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1.

In [14]:
```
regressionModel = LogisticRegression(solver='liblinear')
regressionModel.fit(xTrain,yTrain)
regressionModel.score(xTrain,yTrain)
```

Out[14]:
```
0.9112075769534334
```

The K-nearest neighbours

The K-nearest neighbours (KNN) classifier uses proximity to make classifications or predictions about independent data points. This technique may be used for both classification and regression scenarios and the output will vary. In classification instances, a decision is made based on majority vote, i.e., the class assigned to the new data point is taken to be the one that is most frequently seen in the vicinity of the point. KNN is also known as a lazy learner technique since a model is not learned. Instead, the raw data is stored and used everytime a prediction must be made.

In [15]:
```
KNNModel = KNeighborsClassifier(n_neighbors=3)
KNNModel.fit(xTrain,yTrain)
KNNModel.score(xTrain,yTrain)
```

Out[15]:
```
0.8567482241515391
```

Support Vector Classifier

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane.

In [16]:
```
svmLinearModel=sklearn.svm.SVC(kernel='linear',C=10)
svmLinearModel.fit(xTrain,yTrain)
svmLinearModel.score(xTrain,yTrain)
```

Out[16]:
```
0.9743488555643252
```

In [17]:
```
svmRbfModel=sklearn.svm.SVC(kernel='rbf',C=10)
svmRbfModel.fit(xTrain,yTrain)
svmRbfModel.score(xTrain,yTrain)
```

Out[17]:
```
0.6740331491712708
```

In [18]:
```
svmPolyModel=sklearn.svm.SVC(kernel='poly',C=10000)
svmPolyModel.fit(xTrain,yTrain)
svmPolyModel.score(xTrain,yTrain)
```

Out[18]:
```
0.6440410418310971
```

Random forest classifier

The random forest classifier is an improvement over decision tree classifiers. Based on ensemble learning, a random forest classifier contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. In general, a greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

In [19]:
```
randomFModel = RandomForestClassifier()
randomFModel.fit(xTrain, yTrain)
randomFModel.score(xTrain,yTrain)
```

Out[19]:
```
1.0
```

Decision tree classifier

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In [20]:
```
dTreeModel = DecisionTreeClassifier()
dTreeModel.fit(xTrain, yTrain)
dTreeModel.score(xTrain,yTrain)
```

Out[20]:
```
1.0
```

Gaussian Process Classifier

The GaussianProcessClassifier implements Gaussian processes (GP) for classification purposes, more specifically for probabilistic classification, where test predictions take the form of class probabilities. GaussianProcessClassifier places a GP prior on a latent function , which is then squashed through a link function to obtain the probabilistic classification. The latent function is a so-called nuisance function, whose values are not observed and are not relevant by themselves. Its purpose is to allow a convenient formulation of the model, and is removed (integrated out) during prediction. GaussianProcessClassifier implements the logistic link function, for which the integral cannot be computed analytically but is easily approximated in the binary case.

In [21]:
```
gpcModel = GaussianProcessClassifier()
gpcModel.fit(xTrain, yTrain)
gpcModel.score(xTrain, yTrain)
```

Out[21]:
```
0.8145224940805051
```

Performance Analysis of the different models 📈

In [22]:

```
trainScores = [regressionModel.score(xTrain, yTrain), KNNModel.score(xTra
in, yTrain), svmLinearModel.score(xTrain, yTrain), svmRbfModel.score(xTra
in, yTrain),svmPolyModel.score(xTrain, yTrain), randomFModel.score(xTrai
n,yTrain), dTreeModel.score(xTrain,yTrain), gpcModel.score(xTrain, yTrai
n)]
testScores = [regressionModel.score(xTest, yTest), KNNModel.score(xTest,
yTest), svmLinearModel.score(xTest, yTest), svmRbfModel.score(xTest, yTes
t),svmPolyModel.score(xTest, yTest), randomFModel.score(xTest,yTest), dTr
eeModel.score(xTest,yTest), gpcModel.score(xTest, yTest)]
indices = ['Logistic Regression', 'KNN', 'SVM-Linear', 'SVM-RBF','SVM-Pol
y', 'RandomForest', 'DecisionTree', 'GPC']
scores = pd.DataFrame({'Training Score': trainScores,'Testing Score': tes
tScores}, index=indices)
plot = scores.plot.bar(figsize=(16, 8), rot=0, color=['#df6589ff', '#3c10
53ff'])
plt.title('Training and Testing Scores')
plt.show()
```

In [23]:
```
scores
```

Out[23]:

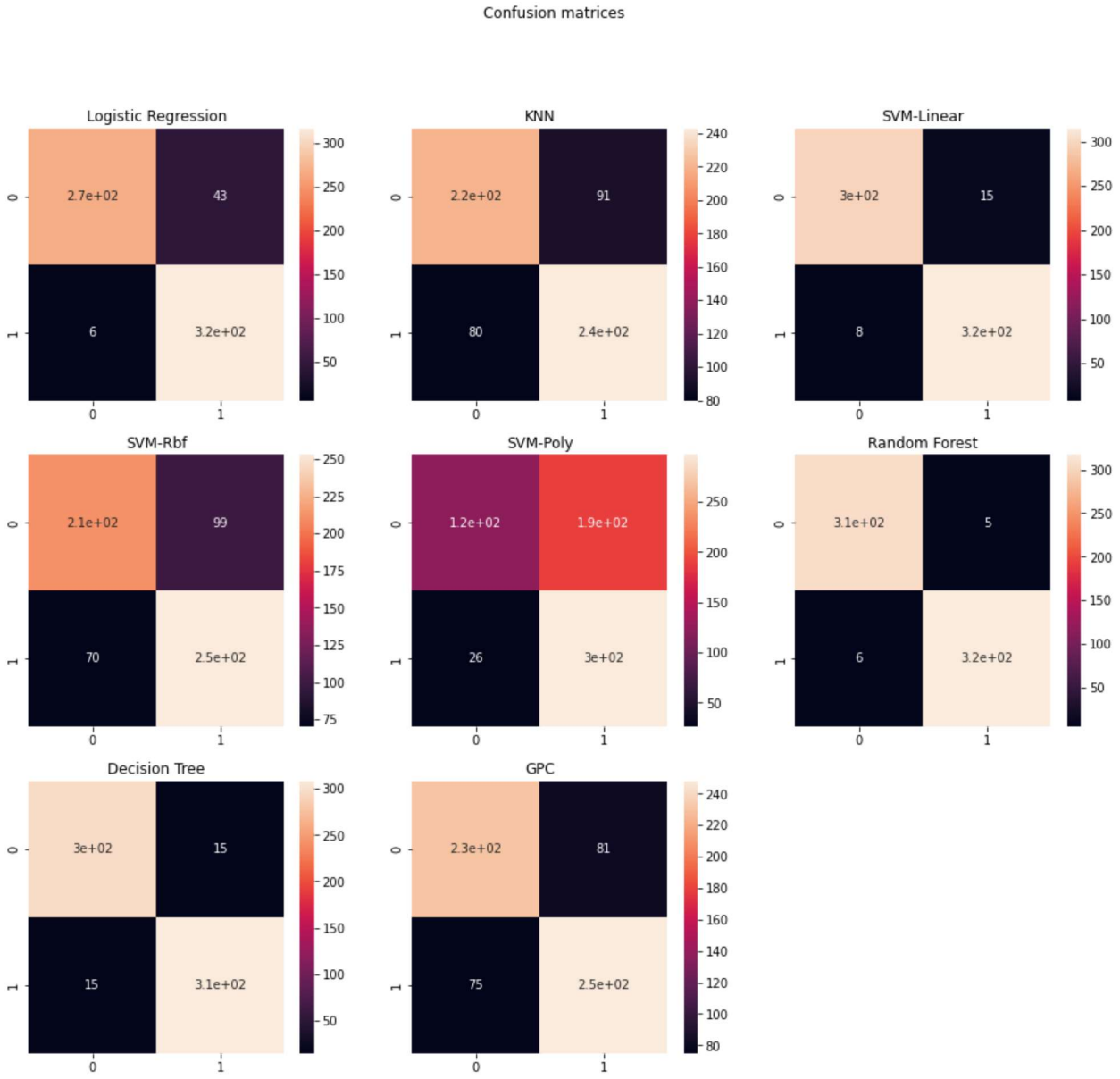
	Training Score	Testing Score
Logistic Regression	**0.911208**	**0.922713**
KNN	**0.856748**	**0.730284**
SVM-Linear	**0.974349**	**0.963722**
SVM-RBF	**0.674033**	**0.733438**
SVM-Poly	**0.644041**	**0.662461**
RandomForest	**1.000000**	**0.982650**
DecisionTree	**1.000000**	**0.952681**
GPC	**0.814522**	**0.753943**

Confusion Matrices 📈

In [24]:
```
predRegression = regressionModel.predict(xTest)
predSVMLinear = svmLinearModel.predict(xTest)
predSVMRbf = svmRbfModel.predict(xTest)
predKNN = KNNModel.predict(xTest)
predSVMPoly = svmPolyModel.predict(xTest)
predRandomF = randomFModel.predict(xTest)
predDTree = dTreeModel.predict(xTest)
predGPC = gpcModel.predict(xTest)
predVals = pd.DataFrame(data={'truth': yTest, 'regression': predRegressio
n, 'knn': predKNN, 'svm-linear': predSVMLinear, 'svm-rbf': predSVMRbf, 's
vm-poly': predSVMPoly, 'random-forest': predRandomF, 'decision-tree': pre
dDTree, 'GPC': predGPC})
```

In [25]:

```
plt.figure(figsize=(16, 14))
plt.subplot(3, 3, 1)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predRegression), anno
t=True).set(title='Logistic Regression')
plt.subplot(3, 3, 2)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predKNN), annot=Tru
e).set(title='KNN')
plt.subplot(3, 3, 3)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predSVMLinear), annot
=True).set(title='SVM-Linear')
plt.subplot(3, 3, 4)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predSVMRbf), annot=Tr
ue).set(title='SVM-Rbf')
plt.subplot(3, 3, 5)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predSVMPoly), annot=T
rue).set(title='SVM-Poly')
plt.subplot(3, 3, 6)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predRandomF), annot=T
rue).set(title='Random Forest')
plt.subplot(3, 3, 7)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predDTree), annot=Tru
e).set(title='Decision Tree')
plt.subplot(3, 3, 8)
sns.heatmap(sklearn.metrics.confusion_matrix(yTest, predGPC), annot=Tru
e).set(title='GPC')
plt.suptitle('Confusion matrices')
plt.show()
```



Classification reports

In [26]:

```
print("Logistic Regression:\n\n", sklearn.metrics.classification_report(y
Test, predRegression))
```

```
Logistic Regression:

              precision    recall  f1-score   support

           0       0.98      0.86      0.92       311
           1       0.88      0.98      0.93       323

    accuracy                           0.92       634
   macro avg       0.93      0.92      0.92       634
weighted avg       0.93      0.92      0.92       634
```

In [27]:

```
print("KNN:\n\n", sklearn.metrics.classification_report(yTest, predKNN))
```

```
KNN:

              precision    recall  f1-score   support

           0       0.73      0.71      0.72       311
           1       0.73      0.75      0.74       323

    accuracy                           0.73       634
   macro avg       0.73      0.73      0.73       634
weighted avg       0.73      0.73      0.73       634
```

In [28]:

```
print("SVM with linear kernel:\n\n", sklearn.metrics.classification_repor
t(yTest, predSVMLinear))
```

```
SVM with linear kernel:

              precision    recall  f1-score   support

           0       0.97      0.95      0.96       311
           1       0.95      0.98      0.96       323

    accuracy                           0.96       634
   macro avg       0.96      0.96      0.96       634
weighted avg       0.96      0.96      0.96       634
```

In [29]:

```
print("SVM with RBF kernel:\n\n", sklearn.metrics.classification_report(y
Test, predSVMRbf))
```

```
SVM with RBF kernel:

              precision    recall  f1-score   support

           0       0.75      0.68      0.72       311
           1       0.72      0.78      0.75       323

    accuracy                           0.73       634
   macro avg       0.74      0.73      0.73       634
weighted avg       0.73      0.73      0.73       634
```

In [30]:

```
print("SVM with poly kernel:\n\n", sklearn.metrics.classification_report
(yTest, predSVMPoly))
```

```
SVM with poly kernel:

              precision    recall  f1-score   support

           0       0.83      0.40      0.53       311
           1       0.61      0.92      0.74       323

    accuracy                           0.66       634
   macro avg       0.72      0.66      0.63       634
weighted avg       0.72      0.66      0.64       634
```

In [31]:

```
print("Random Forest:\n\n", sklearn.metrics.classification_report(yTest,
predRandomF))
```

```
Random Forest:

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       311
           1       0.98      0.98      0.98       323

    accuracy                           0.98       634
   macro avg       0.98      0.98      0.98       634
weighted avg       0.98      0.98      0.98       634
```

In [32]:

```
print("Decision Tree:\n\n", sklearn.metrics.classification_report(yTest,
predDTree))
```

```
Decision Tree:

              precision    recall  f1-score   support

           0       0.95      0.95      0.95       311
           1       0.95      0.95      0.95       323

    accuracy                           0.95       634
   macro avg       0.95      0.95      0.95       634
weighted avg       0.95      0.95      0.95       634
```

In [33]:

```
print("Decision Tree:\n\n", sklearn.metrics.classification_report(yTest,
predGPC))
```

```
Decision Tree:

              precision    recall  f1-score   support

           0       0.75      0.74      0.75       311
           1       0.75      0.77      0.76       323

    accuracy                           0.75       634
   macro avg       0.75      0.75      0.75       634
weighted avg       0.75      0.75      0.75       634
```

Overview of predictions

In [34]:
```
predVals.head()
```

Out[34]:

	truth	regression	knn	svm-linear	svm-rbf	svm-poly	random-forest	decision-tree	GPC
2494	0	0	0	0	0	0	0	0	0
2691	0	0	0	0	0	0	0	0	0
1844	0	0	1	0	0	1	0	0	1
2580	0	0	1	0	0	0	0	0	0
806	1	1	1	1	1	1	1	1	1

Conclusion ✅

Based on our analysis, we can conclude that the Random forest classifier is best-suited for this work.