Statistical Machine Learning 1RT700 - Mini-Project

Tim Leon Wywiol, Sudheer Sujay, Paarth Sanhotra, Akanksha Makkar

Contents

1 Problem

The goal of the project is to train a model to predict the gender of the lead role in Hollywood movies based on 14 different movies characteristics such as release year, number of female actors, number of male actors, profits, total number of words spoken in the film, etc. For this, a set of six different machine learning methods are evaluated based on various performing metrics. Hereby, the project focuses on features selection and model hyperparameter tuning. The report elaborates based on the selected models and proceeds to give the results of each of them. The solution recommended for this gender classification problem is a Quadratic Discriminant Analysis (QDA) model with an expected accuracy of 89,7 percent on validation data.

2 Descriptions of applied methods

2.1 Logistic Regression

Logistic regression is a machine learning probabilistic model for (binary) classification. When fitting the model, it learns from the given features x_i. Hereby, it calculates for every feature a weight also owns as coefficient θ by using the maximum-likelihood estimation. This step is called linear regression and is defined as $y = \theta_0 + \theta_1 x_1 + ...\theta_i x_i$. The goal is to predict the output y by minimizing the errors between the model and the given inputs points. While this is the same as in linear regression (Output is a numeric value), logistic regression goes one step further with a sigmoid (logistic) function defined as $\sigma(y) = \frac{1}{1+e^{-y}}$. The logistic function aka. S-shaped curve maps any numeric output into a value between 0 and 1. Any high positive number will be close to 1 and a negative number close to 0. These new outputs $\sigma(y)$ in a range of [0,1], can be represented as probabilities on how confident the model is of its class prediction $\hat{y}$. Hereby, a defined threshold t (often 0.5) divides the probabilities into two classes $\hat{y}$. [8] [7]

To sum up, the logistic regression model calculates the optimal coefficients to the inputs x_i by a loss function, squeezed the results into a sigmoid (logistic) function, which returns the probabilities of the prediction $\hat{y}$ between 0 and 1 based on a define threshold t. [9]

2.2 Linear Discriminant Analysis (LDA)

Linear discriminant analysis is a technique which is widely used for supervised classification issues . When employing this model, the data is split into two or more classes in order to project higher-dimensional features into lower-dimensional space. Data is always separated in a linear fashion. Each class is assumed to be regularly distributed.
The implementation of LDA is done using Sci-Kit Learn which uses a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix [6]

2.3 Quadratic Discriminant Analysis (QDA)

Quadratic discriminant analysis is closely related to LDA, however in QDA there is no assumption that covariance on each of the models is identical. QDA assumes that each class has its own covarience matrix. QDA classifiers uses several parameters to determine in which class should an observation be classified.
The implementation of QDA is done using Sci-Kit Learn which uses a classifier with a quadratic decision boundary based on fitted conditional densities as described by Bayes' Theorem. Each class is fitted with a Gaussian density. [11]

2.4 Tree-Based Methods

2.4.1 Classification Trees

This technique involves Recursive Binary Splitting where we create a partition amongst the graphical data representation so that it minimizes the average loss for the model. This is infeasible. Hence we make the splits without looking ahead at future splits and make efforts to minimize the loss by repeating this till most of the regions on the graph are classified.[10]

2.4.2 Bootstrap Aggregation (Bagging)

In bagging, we create a few datasets out of the dataset we already have (bootstrapping) by choosing records with replacement for it. We then train models on each of the dataset we have created and take an average over the ensemble of them. Here the bias is small but variance is reduced. [10]

2.4.3 Random Forest

Whenever we are splitting nodes, we do not consider all possible input variables but pick a random subset of it every time while training. These subsets are generated independently for each of the ensemble members so that we end up with different subsets for different trees. In Bagging, since the bootstrapped datasets are co-related, the variance reduction is diminished. This can be avoided by perturbing the depth of each tree. This element of randomness decreases the correlation between in the datasets (trees) and the variance is reduced compared to Bagging.[12][10]

2.5 k-Nearest Neighbors

The supervised machine learning algorithm k-nearest neighbors (KNN) is a simple, easy-to-implement technique that may be used to address both classification and regression tasks. The KNN algorithm believes that objects that are similar are close together. The approach relies on selecting an ideal k-value (the number of neighbors in close proximity to the data point). Furthermore, selecting the best mathematical approach for calculating the distance between the points is critical.
In our code, we utilized sklearn.neighbors.KNeighborsClassifier to implement the model.[5] The function is only approximated locally in k-NN classification, and all computation is postponed until the function is evaluated. Because this method relies on distance for classification, normalizing the training data can greatly increase its performance if the features represent various physical units or come in wildly different scales.

2.6 adaBoosting

An AdaBoost classifier is a meta-estimator that starts by fitting a classifier on the original dataset, then fits further copies of the classifier on the same dataset, but adjusts the weights of poorly classified instances so that future classifiers focus more on difficult cases. Boosting algorithms combine many low accuracy (weak) models to produce high accuracy (strong) models. It combines many classifiers to improve classifier accuracy.
Library used: sklearn.ensemble.AdaBoostClassifier.AdaBoost is a method for creating iterative ensembles. Its core principle is to establish the weights of classifiers and train the data sample in each iteration so that reliable predictions of uncommon observations may be made. Any machine learning method that accepts weights on the training set can be used as a basic classifier.
It must meet two conditions: The classifier should be interactively trained using a variety of weighed training examples.
It seeks to offer an excellent fit for these instances in each iteration by minimizing training error.

3 Implementation on the data

This chapter for implementation is divided into 3 steps, Data exploration, Data preprocessing and Model Tuning.
1. Data exploration: The goal here is to get familiar with the given data set about Hollywood movies. The first findings are that the data set has 14 different information about 1039 movies. 13 features are numeric and only the column *Lead* is categorical. Another finding is that features have different ranges. For instance, `Total words` have a range of [1351,67548] and the `number of female actors` has a range of [1, 16]. With the help of a scatter matrix, the distribution of each feature can be analyzed, which can be used to detect for example outliers. Furthermore, **Imbalanced classification** is present as seen in the number of male and female leads given in the training data. The number of movies with male leads is almost 3 times higher compared to female leads (Ratio 3:1). This slight imbalance will make it harder to predict the minority class in this case `female`, due to the

less amount of data points.[2] Based on those finding data preprocessing measures such as Scaling, normalizing, removing outliers or threshold-moving can be derived.

2. Data preprocessing: In the report different data preprocessing steps were evaluated. Two steps were implemented. First, the transformation of the `Lead` column from categorical to numeric [0,1]. Second, any model method, which is using distance measures to compare data points, requires features from the same range. Otherwise features with big numbers, will have a bigger impact. Therefore, normalization will be applied for in the project on kNN and Adaboosting [13].

Lastly, the split of training and validation data is done by the Cross-validation technique with 10-fold.

3. Model Tuning: Having the data set split into training and validation sets, all models are trained to predict the lead for unseen data points from the validation data. To measure the performances of the models, comparative metrics are calculated such as missclassification error, accuracy, RoC/AUC, confusion matrix, F1-Score, False Negative, False Positive. However, the key decision-making factor of this project is the highest accuracy of the model. As a starting point, figure 1 illustrates the performance of all applied models based on the validation error without any tuning technique used.

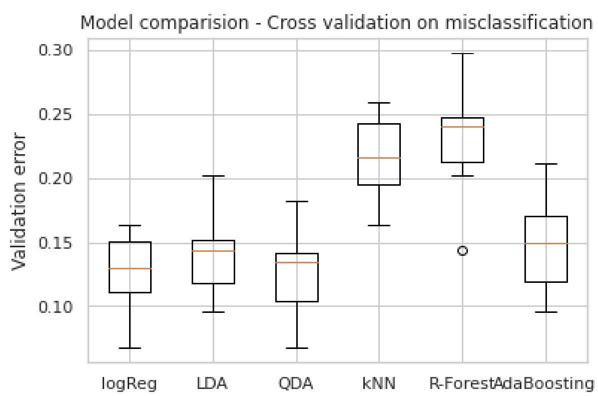


Figure 1: Model classification error comparison before tuning

Having a first understanding of every models performance, the task is now to tune every model by using tuning techniques such as (A) Hyperparameter tuning and (B) feature selection.

A) Hyperparameter tuning
GridSearchCV
The parameters that determine the model architecture are known as hyperparameters, and the process of finding the perfect model architecture is known as hyperparameter tuning. [14] These hyperparameters could be used to answer questions like: For my linear model, what degree of polynomial features should I use? What is the greatest depth that my decision tree can have? In my decision tree, what should be the minimal amount of samples required for a leaf node? To be specific: hyperparameters are not model parameters, and they cannot be learned directly from data. Implemented using: sklearn.modelselection.GridSearchCV

Logistic Regression: hyperparameter tuning
The hyperparameter `solvers`, `C-value`, `penalty` and `class_weight` of the logistic regression model are tuned. The best performing solver from the possibilities of [‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’] is **‘liblinear’** with a C value of 1, L2-penalty, and none for class_weight. These hyperparameters will be applied for further training.

LDA and QDA: Hyperparameter Tuning LDA and QDA are two classic classifiers, with, as their names suggest, a linear and a quadratic decision surface, respectively, and have no hyperparameters to tune.[4]

Random Forest: Hyperparameter Tuning
For Random Forest, we are using GridSearchCV. The parameters being tuned are n-estimators, max-depth and max-features.

KNN: hyperparameter tuning
The specific aspect of GridSearchCV applied to KNN included tuning the model to find the optimum

k-neighbours value, the optimal distance method - which was chosen an Manhattan distance and the best leaf size. [**Tune Hyperparameters for Classification Machine Learning Algorithms**]
Based on our observations, after applying tuning methods, the model was seen to have a much better accuracy and performance. The same is displayed in the boxplot in the following section.

AdaBoost: hyperparameter tuning
Using GridSearchCV the following hyperparameters when tuned helped improve the performance of the model: nEstimators, learningRate. nEstimators: The number of base estimators or weak learners we want to use in our dataset. learning rate: This parameter is provided to shrink the contribution of each classifier.

B) Feature selection
The goal of feature selection is to reduce overfitting, improve accuracy, and speed up training time. This is done by finding the most important features, which have a big impact on the `Lead` prediction between male and female. In other words, it removes irrelevant features, which makes the data set noisy. One method to applied this is by the Recursive Feature Elimination (RFE) technique.[3] Hereby, the accuracy is compared by excluding different input features such as Number words, Year and Gross. The results of the feature score for the Logistic Regression are:

Feature	Rang		Feature	Rang
Mean Age Male	1		Year	3
Age Lead	1		Difference in words lead and co-lead	4
Mean Age Female	1		Number of words lead	5
Number words female	1		Number words male	6
Mean Age Male	1		Gross	7
Number of male actors	2		Total words	8

Figure 2: Feature Selection with Recursive Feature Elimination

The table shows that `Total words` and `Gross` are the most irrelevant features and therefore could be removed. Additional to the RFE methods, which only take the accuracy into account, all applied models were compared with the false-negative rate, false-positive rate, and its AUC-value. Hereby, every model method is trained by excluding different feature combinations. The Figure 5 shows a subset of the results for the models Logistic Regression, LDA, and QDA. The most important finding is that excluding the number of actors has a big negative impact on the performance of all models. On the other hand, excluding the features “Total Words" and "Gross", the error rate and the AUC values improves, especially for QDA. This finding underlines the result from the previous RFE approach. To sum up, the features "Total Words" and "Gross" are not essential for the prediction of lead based on misclassification error and AUC and therefore will be excluded from fitting the models.

4 Evaluation

After having built a predictive classification model, one needs to assess its performance, or how well it predicts the outcome of new observations test data that were not used to train the model.

In other words, it needs to use a new test data set to assess the model’s prediction accuracy and errors. Because by knowing the actual outcome of the observations in the test data set, the predictive model’s performance can be evaluated by comparing the anticipated and known outcome values. For evaluating the models, the method **cross-validation** with a 10-fold is used. The results of the validation error of each model are plotted in a box-plot diagram. Fist evaluation was done in Figure 1 using all the features in the training data. The second evaluation is done after data preprocessing and model tuning (Figure 3. The first finding is that all the models have been improved after applying the describe tuning techniques.

As it can be seen in the Figure 3 and 4, QDA has the best performance with a mean validation error of around 10%, followed by AdaBoosting and Logistic Regression. Additionally, QDA range is one of best, assuming the prediction of unseen data is steady and reliable. Thus, expecting a low variance.

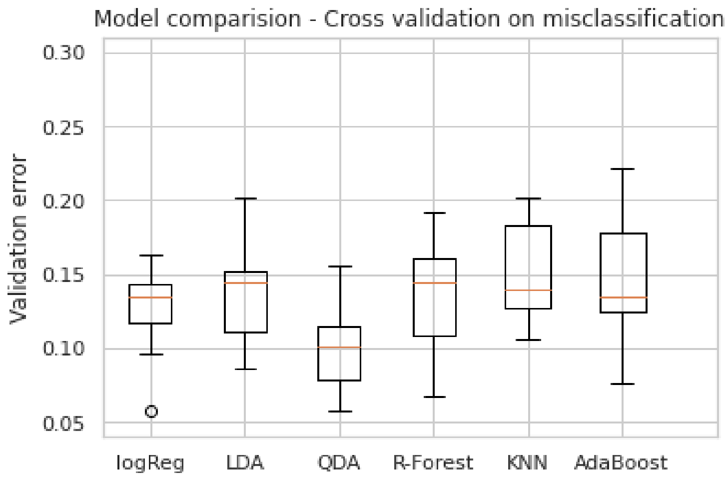


Figure 3: Model classification error comparison after tuning

5 Production

As mention in the evaluation, the QDA model will be the choice for production. The main reason is the lowest mean validation error and the highest F1 score, meaning it is the best model for prediction the lead overall. Additionally, it has one of the lowest ranges, which indicates a low model variance. Thus, it can be assume that QDA predicts reliable results. Furthermore, QDA proves with good AUC (False Negative False Positive) values that it can best handle the imbalance in the data.

	Accuracy	F1-score	AUC (+- 0.05)	False Negative(Male)	False Positive(Female)
Logistic Regression	87.5	70.9	88	0.04	0.377
LDA	86.4	67.9	88	0.045	0.413
QDA	89.7	78	95	0.054	0.251
KNN	82.5	55	76	0.56	0.051
AdaBoosting	88	70.5	91	0.089	0.358
Random Forest	85.7	64.8	90	0.04	0.46

Figure 4: Metrics of all models compared

6 Conclusion

Cross validation on misclassification confirms that QDA is an appropriate choice of a model for this problem. The mean error is 10%, with lower and higher bounds of 6% and 16%, respectively. We used hyper parameter adjustment and feature selection to try to improve the other models.

The most important metric in model selection, for this gender classification problem is accuracy! The drawback is that because of the unbalanced data, it has a high FalseNegative rate. It is therefore unsuitable for predicting the minority class `Female`. However, as we have shown, the gender difference will narrow in the future, and hence fresh data sets for 2015 and beyond will not feature such a male-biased data set. We recommend to use our model for determining good overall accuracy in both classes male and female.

7 Feature Importance

7.1 Feature Selection

The table of the RFE score and Figure 5 illustrate that `Words spoken by males and females` are important features for predicting the lead. Furthermore, the feature `year` is also a crucial factor. However, the feature `Gross` is one of the irrelevant information for the lead classification. More information for Feature Selection can be found in the tuning chapter above.

	LOG REG		LDA		QDA	
	Error rate	AUC	Error rate	AUC	Error rate	AUC
All features	**11,85897**	**0,89**	**12,179**	**0,89**	**16,15**	**0,8**
Exclude Age Lead and Age Co-lead	11,54	0,87	12,179	0,87	16,15	0,8
~~Exclude Total Words~~	12,5	0,89	12,179	0,89	15,77	0,9
~~Exclude Gross~~	11,85897	0,89	11,21795	0,89	17,69	0,87
Exclude Number words lead & Difference to Co-lead	13,46	0,88	15,7	0,87	22,31	0,77
Exclude Mean Ages [male/female]	13,46	0,89	12,17	0,89	16,15	0,8
Exclude Number of actors [male/female]	20,834	0,77	20,83	0,76	27,69	0,71
Exclude YEAR	13,14	0,83	12,5	0,89	26,92	0,75
Exclude Words of [male/female]	18,91	0,83	20,19231	0,83	25,77	0,74

Figure 5: Feature Selection for LogReg, LDA, QDA - Excluding Total Words and Gross has a positive impact on the performance for LDA and QDA

7.2 Three Questions about the training data broken down by gender

In this chapter the data from 1039 Hollywood movies from the years 1934 to 2015 was analysed to gain more insights about differences in movies between male and female main actors.

Do men or women dominate speaking roles in Hollywood movies?

The findings show that 76 percent of the main roles in Hollywood movies are male. In other words, only 1 out of 4 leads, which have the most speaking parts in movies, are female. This results in the conclusion that men dominate speaking roles in Hollywood movies between 1934 to 2015.

Has gender balance in speaking roles changed over time?

The figure 6 illustrate the gender gap (Difference between #Female_Lead and #Male_Lead) in speaking roles in Hollywood movies. The gender gap in terms of which gender is the main lead in Hollywood movies has increased over the time in favor of men. For instance the biggest disparity was in 1997, where out of 47 new Hollywood movies 7 women and 40 men were the lead, which is a total gap of 33. Additionally, the findings show that in a time range of 1995 to 2015 on average 18,1 more male were the main actors in released Hollywood movies. However, in the year 2015 for the first time more females are the lead in movies. Furthermore, looking at the rolling average, which is taking 5 years into consideration, it can be seen that the peak of male dominance in the late 90s has slightly dropped down to around 10 in the year 2015. Even though a forecast can not be derived from the graph, knowing the fact that gender equality has been discussed in society, gives space for predictions that the gender gap might further decrease in the future.

To sum up, in Hollywood movies there is a great chasm between men and women. It has increased constantly over the years until the peak in the late 90s and since then it still remains on a high level in favor of men. The year 2015 breaks this trend and represents a equal gender balance in speaking roles in the Hollywood industry.

Do films in which men do more speaking make a lot more money than films in which women speak more?

Assumptions: Number of speaking words male: Number of words spoken by all other male actors in the film plus lead words if lead is male) Number of speaking words female: Number of words spoken

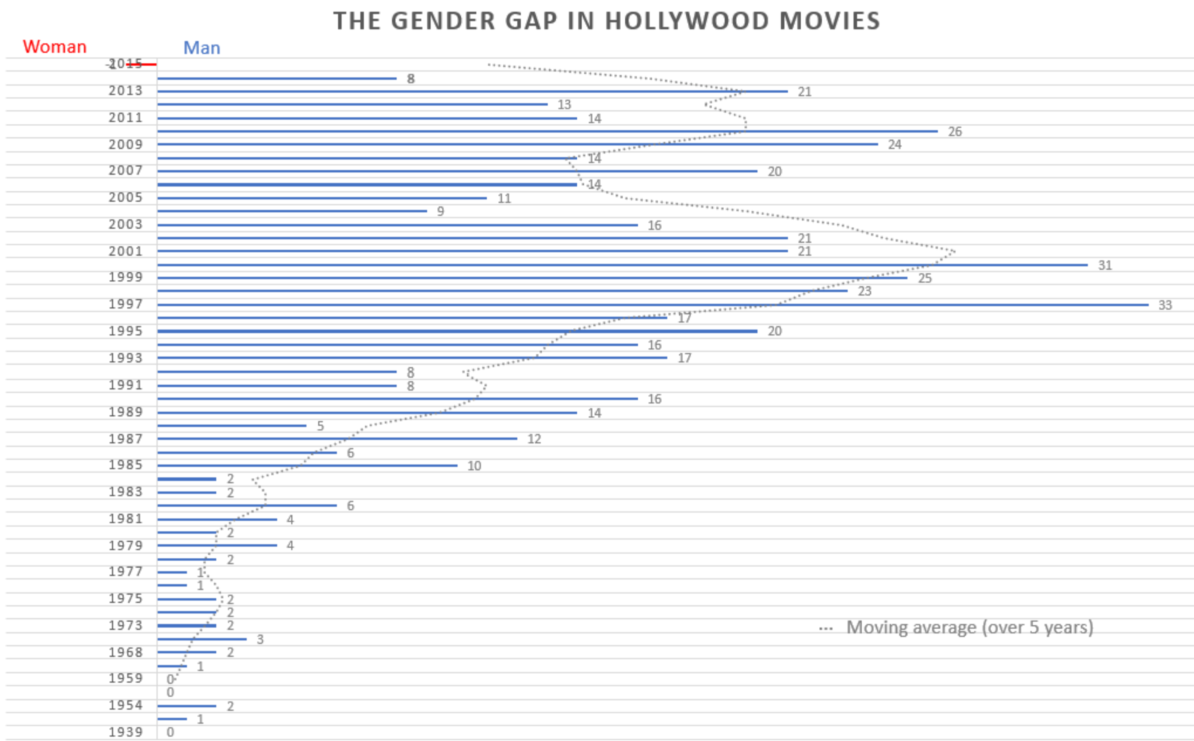


Figure 6: Gender Gap in Hollywood movies between 1939 and 2015

by all other female actors in the film plus lead words if lead is female) The findings show that movies in which males have a larger share (at least 20 % more) in the dialog make 115 Million Dollars profit on average. In comparison, movies where females dominate the dialog (at least 20 % more) make 99 Million Dollars in profit on average. However, based on the feature importance analysis, the Gross feature has a low impact on the prediction of the lead. Thus, other information has a higher impact on why movies make more or less profit. In conclusion, Hollywood movies, in which males speak more words, make 16 Millions dollars more profit on average.

Discussion about gender analysis in Hollywood movies

The gender analysis discovered interesting insights about the differences between males and females in Hollywood movies. The following discussion will touch on a few significant findings from the data study, particularly regarding a film's profit based on male versus female leads, and gender gap trend predictions for the future. Firstly, the significant finding is that men have been dominating the Hollywood industry for 80 years. From 1934 until 2015 the ratio of staffing of speaking roles was 3:1 in favor of men, with the peak being around the year 2000. However, since then the trend has dropped and in the year 2015 more females were in speaking roles than men for the first time. This begs the question: Has the male-dominated film industry run its course? Unfortunately, it is too soon to tell simply from the data we have now. More recent data is needed to make an informed prediction about the possible future state of the film industry. However, taking into consideration that society has become more focused on gender equality in the past 10 years, we can assume that this may be the case.

References

[1] *3.1. Cross-validation: evaluating estimator performance — scikit-learn 1.0.1 documentation*. https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-and-model-selection. (Accessed on 12/05/2021).

[2] *A Gentle Introduction to Imbalanced Classification*. https://machinelearningmastery.com/what-is-imbalanced-classification/. (Accessed on 12/06/2021).

[3] *Feature Selection For Machine Learning in Python*. https://machinelearningmastery.com/feature-selection-machine-learning-python/. (Accessed on 12/06/2021).

[4] *hyperparameter tuning lda and qda sklearn - Google Search*. https://www.google.com/search?q=hyperparameter+tuning+lda+and+qda+sklearn&rlz=1C1GCEA_enIN880IN880&sxsrf=AOaemvJIwblSiz5e0OfvDlcHSnbSyiyLSQ. (Accessed on 12/06/2021).

[5] *KNN Model*. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[6] *LDA Scikit Learn Documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html. (Accessed on 12/17/2021).

[7] *Logistic Regression — Detailed Overview | by Saishruthi Swaminathan | Towards Data Science*. https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc. (Accessed on 12/06/2021).

[8] *Logistic Regression for Machine Learning*. https://machinelearningmastery.com/logistic-regression-for-machine-learning/. (Accessed on 12/06/2021).

[9] *Logistic Regression Scikit Learn Documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. (Accessed on 12/17/2021).

[10] *Machine Learning - A First Course for Engineers and Scientists | sml-book-page*. http://smlbook.org/book/sml-book-draft-latest.pdf. (Accessed on 12/17/2021).

[11] *QDA Scikit Learn Documentation*. https://scikit-learn.org/0.15/modules/generated/sklearn.qda.QDA.html. (Accessed on 12/17/2021).

[12] *Random Forest Scikit Learn Documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. (Accessed on 12/17/2021).

[13] *sklearn standardscaler - Google Search*. https://www.google.com/search?q=sklearn+standardscaler&rlz=1C1GCEA_enIN880IN880&oq=sklear+standa&aqs=chrome.1.69i57j0i13l9.9300j1j7&sourceid=chrome&ie=UTF-8. (Accessed on 12/05/2021).

[14] *Tune Hyperparameters for Classification Machine Learning Algorithms*. https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/. (Accessed on 12/06/2021).

A Appendix

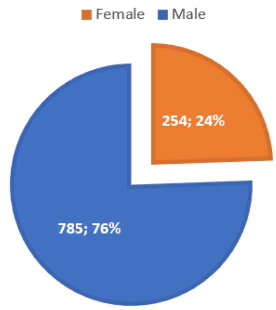


Figure 7: Speaking roles broken down by Gender

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
import sklearn.model_selection as skl_ms
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
import warnings
warnings.filterwarnings('ignore')
from matplotlib.rcsetup import validate_aspect
from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import RocCurveDisplay
from sklearn.model_selection import StratifiedKFold
import sklearn.linear_model as skl_lm
import sklearn.model_selection as skl_ms
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay


plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call driv
e.mount("/content/drive/", force_remount=True).
```

In [ ]:

```
data = pd.read_csv('/content/drive/My Drive/SML/train.csv')
```

data.describe()

In [ ]:

```
data.head(10)
data.describe()
```

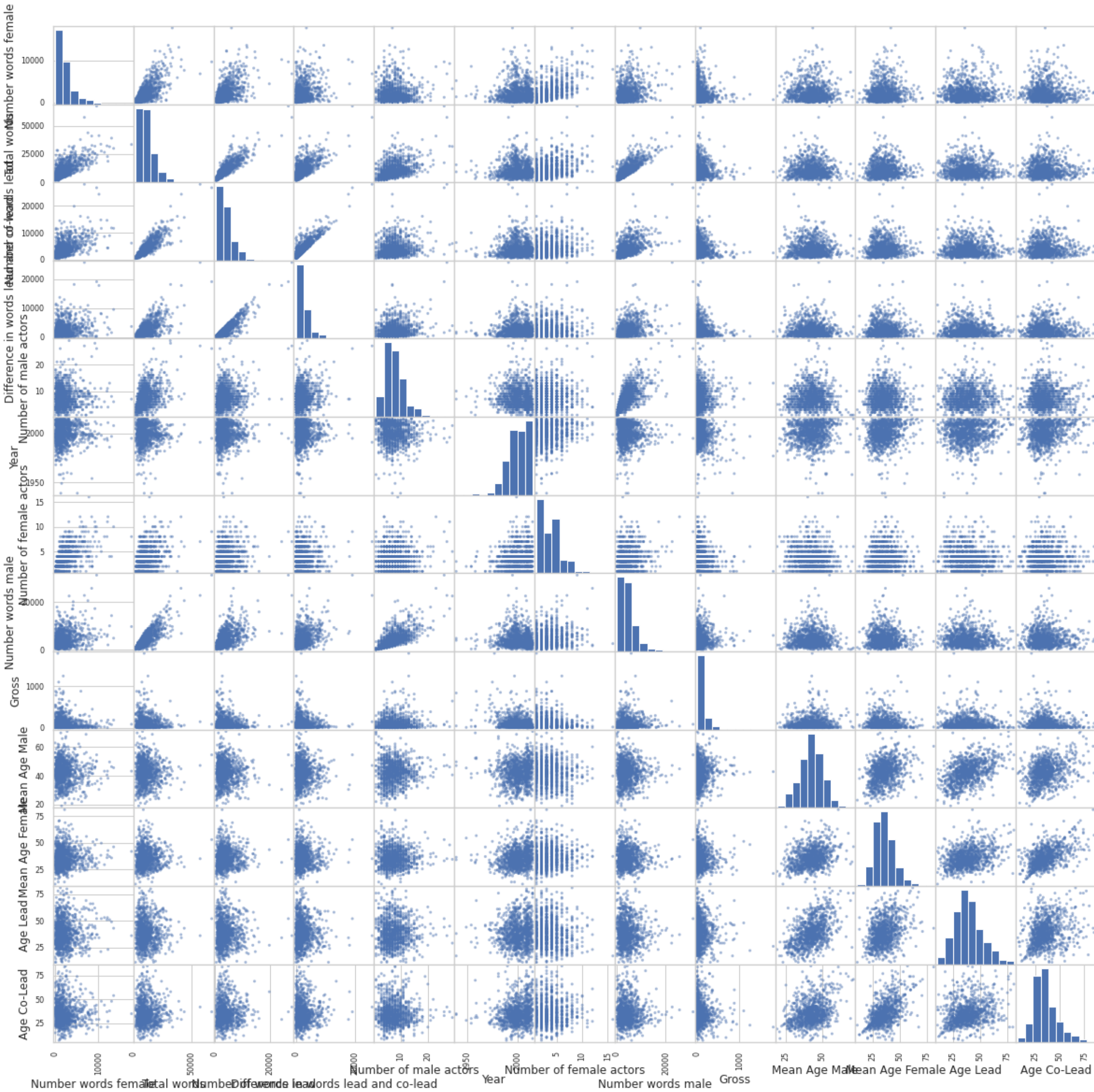
Out[ ]:

	Number words female	Total words	Number of words lead	Difference in words lead and co-lead	Number of male actors	Year	Number fema acto
count	1039.000000	1039.000000	1039.000000	1039.000000	1039.000000	1039.000000	1039.0000
mean	2334.256015	11004.368624	4108.256978	2525.024062	7.767084	1999.862368	3.5072
std	2157.216744	6817.397413	2981.251156	2498.747279	3.901439	10.406632	2.0885
min	0.000000	1351.000000	318.000000	1.000000	1.000000	1939.000000	1.0000
25%	904.000000	6353.500000	2077.000000	814.500000	5.000000	1994.000000	2.0000
50%	1711.000000	9147.000000	3297.000000	1834.000000	7.000000	2000.000000	3.0000
75%	3030.500000	13966.500000	5227.000000	3364.000000	10.000000	2009.000000	5.0000
max	17658.000000	67548.000000	28102.000000	25822.000000	29.000000	2015.000000	16.0000

Explore Data set

In [ ]:

```
pd.plotting.scatter_matrix(data.iloc[:, 0:14], figsize=(20,20))
plt.show()
# big outliers in a feature?
# Yes, there might be some, but they are relevent. Example: Number words female is o
# Meaning, we need a preprocessing methods, with accepts outliers but normalize the 
```

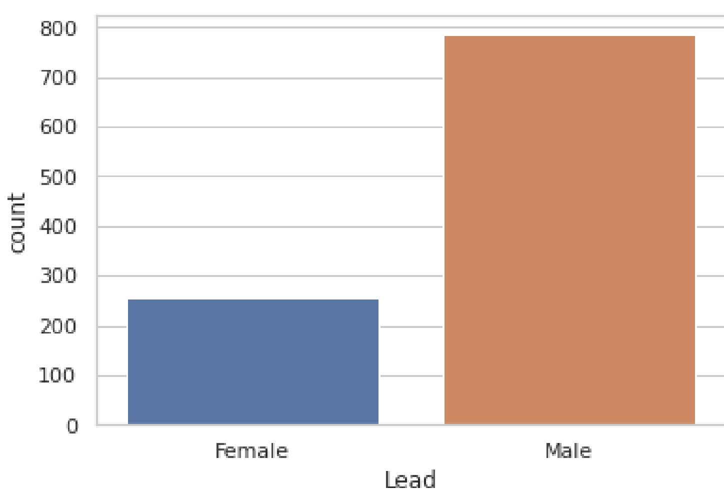


In [ ]:

```
sns.countplot(data['Lead'])
```

Out[ ]:

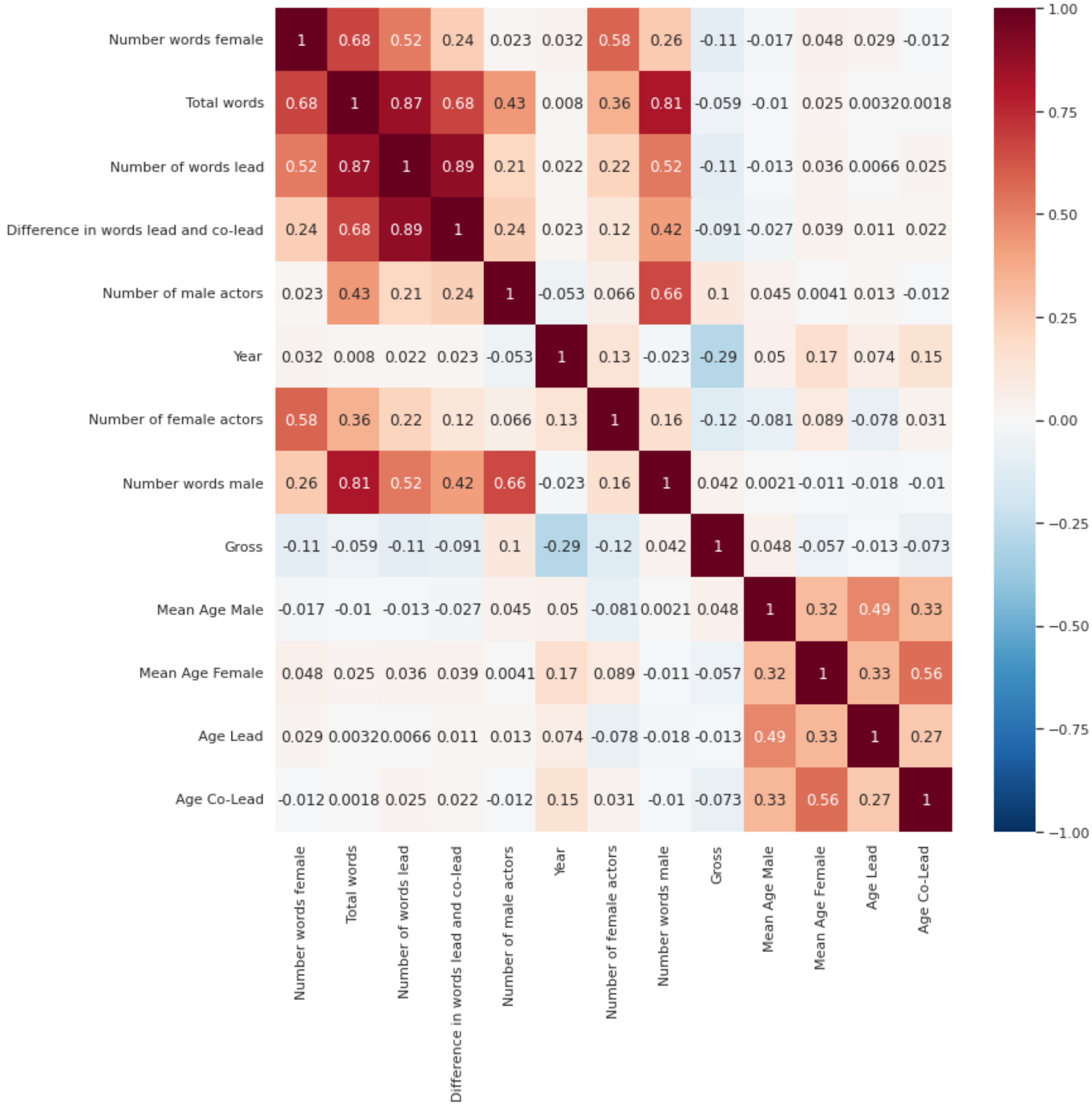
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5517b90610>
```



In [ ]:

```
# Correlation Heat map
corr = data.corr()
corr
```

```
plt.figure(figsize=(12,12))
sns.heatmap(corr, cmap='RdBu_r', annot=True, vmax=1, vmin=-1)
plt.show()
```



Data Preprocessing

In [ ]:

```
##########################################################################################
################################################## Data preparation ################
##########################################################################################
# from sklearn.preprocessing import StandardScaler
# sc = StandardScaler()
# data.iloc[:, :5] = sc.fit_transform(X=data.iloc[:, :5])
# data.iloc[:, 6:8] = sc.fit_transform(X=data.iloc[:, 6:8])

# print(data.head(10))
# print(data.describe())

#STep 0 FEATURE SELECTION X and y
#X = data.drop(columns=['Lead','Gross', 'Mean Age Male', 'Mean Age Female', 'Age Lea
#y = data['Lead']
```

Convert target feature to catergorical number values (0 and 1)

In [ ]:
```
#### Convert LEAD from labels into [0,1]

data = pd.get_dummies(data,columns=['Lead'])
data = data.rename(columns={'Lead_Female':'Lead'})
data = data.drop(columns='Lead_Male')
```

Convert target feature to catergorical number values (-1 and 1)

In [ ]:
```
# #### Convert Lead into -1 , 1
# data['Lead_bin'] = data['Lead'].apply({'Male': -1, 'Female':1}.get)

# data.head()
```

In [ ]:
```
#### Convert LEAD from labels into [0,1]
# data_test = pd.get_dummies(data_test,columns=['Lead'])
# data_test = data_test.rename(columns={'Lead_Female':'Lead'})
# data_test = data_test.drop(columns='Lead_Male')
# data_test.head()
```

Splitting Data into train, test ; X, y

In [107…
```
# define dataset
X = data.drop(columns=['Lead','Gross','Total words'])
y = data['Lead']
np.random.seed(1)
X_train, X_val, y_train, y_val = skl_ms.train_test_split(X,y,train_size=0.75, random
```

Logistic Regression

Data preprocession LogReg

In [108…
```
data = pd.read_csv('/content/drive/My Drive/SML/train.csv')
#### Convert LEAD from categorical to numberic [0,1]
data = pd.get_dummies(data,columns=['Lead'])
data = data.rename(columns={'Lead_Female':'Lead'})
data = data.drop(columns='Lead_Male')
```

In [109…
```
# define dataset
X = data.drop(columns=['Lead'])
y = data['Lead']
np.random.seed(1)
X_train, X_val, y_train, y_val = skl_ms.train_test_split(X,y,train_size=0.75, random
```

Hyperparameter tuning

In [ ]:
```
# https://machinelearningmastery.com/hyperparameters-for-classification-machine-lear
from sklearn.metrics import classification_report
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

#List Hyperparameters that we want to tune.
```

```
model = LogisticRegression()
solvers = ['liblinear','newton-cg', 'lbfgs', 'sag', 'saga']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
class_weight = [None, {1:2,0:1}]

#Convert to dictionary
hyperparameters =  dict(solver=solvers,penalty=penalty,C=c_values, class_weight=clas

#Create new KNN object
model_logReg = skl_lm.LogisticRegression()
#Use GridSearch

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model_logReg, param_grid=hyperparameters, n_job

#Fit the model
grid_result = grid_search.fit(X_train,y_train)


#Print The value of best Hyperparameters
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.870385 using {'C': 0.01, 'class_weight': None, 'penalty': 'l2', 'solver': 'l
iblinear'}
0.868676 (0.032423) with: {'C': 100, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'liblinear'}
0.866117 (0.036385) with: {'C': 100, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'newton-cg'}
0.799806 (0.044025) with: {'C': 100, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'lbfgs'}
0.756943 (0.016660) with: {'C': 100, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'sag'}
0.753108 (0.012973) with: {'C': 100, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'saga'}
0.831441 (0.038044) with: {'C': 100, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'liblinear'}
0.831013 (0.040328) with: {'C': 100, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'newton-cg'}
0.784826 (0.045628) with: {'C': 100, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'lbfgs'}
0.750566 (0.040467) with: {'C': 100, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'sag'}
0.752270 (0.042680) with: {'C': 100, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'saga'}
0.866972 (0.034502) with: {'C': 10, 'class_weight': None, 'penalty': 'l2', 'solver':
'liblinear'}
0.866117 (0.036385) with: {'C': 10, 'class_weight': None, 'penalty': 'l2', 'solver':
'newton-cg'}
0.797686 (0.045358) with: {'C': 10, 'class_weight': None, 'penalty': 'l2', 'solver':
'lbfgs'}
0.756943 (0.016660) with: {'C': 10, 'class_weight': None, 'penalty': 'l2', 'solver':
'sag'}
0.753108 (0.012973) with: {'C': 10, 'class_weight': None, 'penalty': 'l2', 'solver':
'saga'}
0.831868 (0.037409) with: {'C': 10, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'liblinear'}
0.830586 (0.040369) with: {'C': 10, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
```

```
'solver': 'newton-cg'}
0.786092 (0.045896) with: {'C': 10, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'lbfgs'}
0.750993 (0.040459) with: {'C': 10, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'sag'}
0.752270 (0.042680) with: {'C': 10, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'saga'}
0.867405 (0.035570) with: {'C': 1.0, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'liblinear'}
0.865690 (0.036542) with: {'C': 1.0, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'newton-cg'}
0.801082 (0.038057) with: {'C': 1.0, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'lbfgs'}
0.756943 (0.016660) with: {'C': 1.0, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'sag'}
0.753108 (0.012973) with: {'C': 1.0, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'saga'}
0.832301 (0.037176) with: {'C': 1.0, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'liblinear'}
0.830159 (0.040133) with: {'C': 1.0, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'newton-cg'}
0.788667 (0.051129) with: {'C': 1.0, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'lbfgs'}
0.750566 (0.040467) with: {'C': 1.0, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'sag'}
0.752270 (0.042936) with: {'C': 1.0, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'saga'}
0.867399 (0.033349) with: {'C': 0.1, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'liblinear'}
0.866117 (0.034212) with: {'C': 0.1, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'newton-cg'}
0.801521 (0.043413) with: {'C': 0.1, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'lbfgs'}
0.756516 (0.017154) with: {'C': 0.1, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'sag'}
0.753108 (0.012973) with: {'C': 0.1, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'saga'}
0.832301 (0.036880) with: {'C': 0.1, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'liblinear'}
0.829731 (0.039180) with: {'C': 0.1, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'newton-cg'}
0.788656 (0.046913) with: {'C': 0.1, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'lbfgs'}
0.750993 (0.040459) with: {'C': 0.1, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'sag'}
0.752697 (0.042268) with: {'C': 0.1, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'saga'}
0.870385 (0.032382) with: {'C': 0.01, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'liblinear'}
0.868670 (0.032774) with: {'C': 0.01, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'newton-cg'}
0.799795 (0.042156) with: {'C': 0.01, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'lbfgs'}
0.756943 (0.016660) with: {'C': 0.01, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'sag'}
0.753108 (0.012973) with: {'C': 0.01, 'class_weight': None, 'penalty': 'l2', 'solve
r': 'saga'}
0.825036 (0.043463) with: {'C': 0.01, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'liblinear'}
0.828882 (0.040994) with: {'C': 0.01, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'newton-cg'}
0.785670 (0.047735) with: {'C': 0.01, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'lbfgs'}
0.750566 (0.040467) with: {'C': 0.01, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
```

```
'solver': 'sag'}
0.752697 (0.042268) with: {'C': 0.01, 'class_weight': {1: 2, 0: 1}, 'penalty': 'l2',
'solver': 'saga'}
```

Feature Selection

In [ ]:

```
# Feature Extraction with RFE
from sklearn.feature_selection import RFE

# define dataset
X = data.drop(columns=['Lead'])
y = data['Lead']

# feature extraction
model = LogisticRegression(solver='liblinear')
# model = skl_da.LinearDiscriminantAnalysis()
rfe = RFE(model)
rfe.fit(X,y)
print("Num Features: %d" % rfe.n_features_)
print("Selected Features: %s" % rfe.support_)
print("Feature Ranking: %s" % rfe.ranking_)
data.head()
```

```
Num Features: 6
Selected Features: [False False False False  True False  True False False  True  Tru
e  True
  True]
Feature Ranking: [3 8 5 4 1 2 1 6 7 1 1 1 1]
```

Out[ ]:

	Number words female	Total words	Number of words lead	Difference in words lead and co-lead	Number of male actors	Year	Number of female actors	Number words male	Gross	Mean Age Male	Fe
0	1512	6394	2251.0	343	2	1995	5	2631	142.0	51.500000	42.3
1	1524	8780	2020.0	1219	9	2001	4	5236	37.0	39.125000	29.3
2	155	4176	942.0	787	7	1968	1	3079	376.0	42.500000	37.0
3	1073	9855	3440.0	2623	12	2002	2	5342	19.0	35.222222	21.5
4	1317	7688	3835.0	3149	8	1988	4	2536	40.0	45.250000	45.0

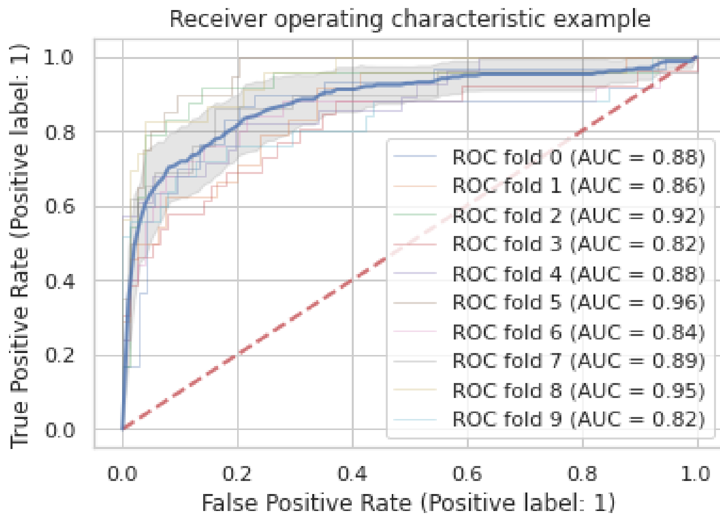
Logistic regression with cross validation

In [ ]:

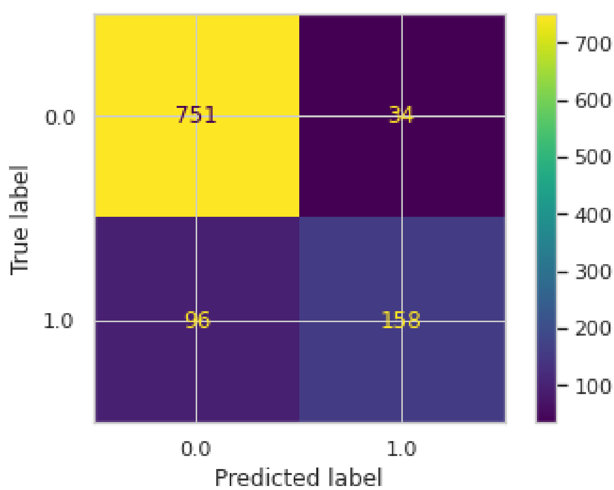
```
#####################################################################################
################################################ Logistic regression with cross va
#####################################################################################

# ##############################################################################
# Classification and ROC analysis
n_folds = 10
cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
model_LogReg = skl_lm.LogisticRegression(solver='liblinear', C=1.0, penalty='l1' )

tprs = []
aucs = []
prediction_all = np.zeros(1)
val_all = np.zeros(1)
```

```
misclassification = []
val_all = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, val) in enumerate(cv.split(X)):
    model_LogReg.fit(X.iloc[train], y.iloc[train])
    viz = RocCurveDisplay.from_estimator(
        model_LogReg,
        X.iloc[val],
        y.loc[val],
        name="ROC fold {}".format(i),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    prediction = model_LogReg.predict(X.iloc[val])
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
    misclassification.append(np.mean(prediction != y.iloc[val]))
    prediction_all = np.concatenate((prediction_all, prediction))
    val_all = np.concatenate((val_all, y.iloc[val]))

ax.plot([0, 1], [0, 1], linestyle="--", lw=2, color="r", label="Chance", alpha=0.8)

prediction_all = prediction_all[1:]
vall_all = val_all[1:]
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color="b",
    label=r"Mean ROC (AUC = %0.2f $\pm$ %0.2f)" % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label=r"$\pm$ 1 std. dev.",
)

ax.set(
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="Receiver operating characteristic example",
)
# ax.legend(loc="lower right")
plt.show()
print('\n ########### Metrics about Logistic Regression ##########')
print('Missclassification Rate: %0.6f ' % np.mean(misclassification))
print('Accuracy: %.3f\n' % accuracy_score(val_all, prediction_all))
```

```
ConfusionMatrixDisplay.from_predictions(val_all , prediction_all)
false_positive_rate = (np.sum((prediction_all==0)&(val_all==1)))/(np.sum(val_all ==
false_negative_rate = (np.sum((prediction_all==1)&(val_all==0)))/(np.sum(val_all ==
print("False Positive (Female): ", false_positive_rate)
print("False Negative (Male): ", false_negative_rate)
print('F1 Score Female: %.3f' % f1_score(val_all, prediction_all))
print('Precision Female: %.3f' % precision_score(val_all, prediction_all))
# print('Recall Female: %.3f' % recall_score(val_all, prediction_all))
```




```
 ########### Metrics about Logistic Regression ##########
Missclassification Rate: 0.125112
Accuracy: 0.875

False Positive (Female):  0.3779527559055118
False Negative (Male):  0.04331210191082802
F1 Score Female: 0.709
Precision Female: 0.823
```



LDA

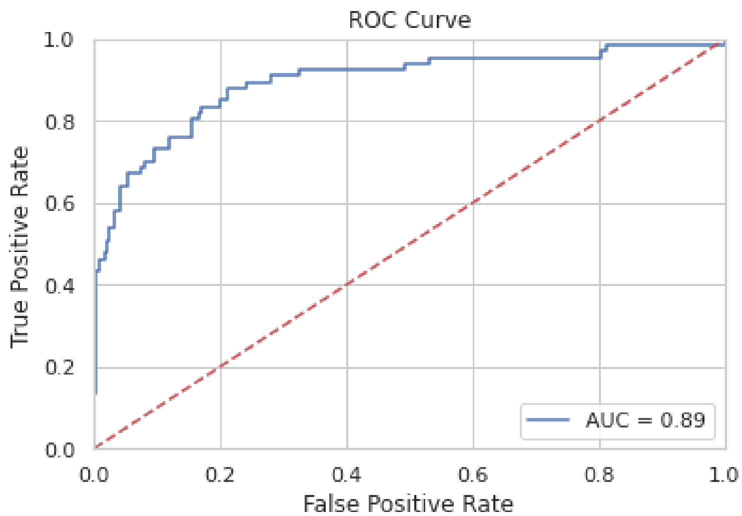
In [ ]:

```
np.random.seed(1)
X = data.drop(columns=['Lead','Gross','Total words'])
y = data['Lead']
X_train, X_val, y_train, y_val = skl_ms.train_test_split(X,y,train_size=0.7, random_
model = skl_da.LinearDiscriminantAnalysis()
model.fit(X_train, y_train)
```

```
prediction = model.predict(X_val)
err = np.mean(prediction != y_val)
print('Error rate [Validation Error] for LDA '+ str(err))
predict_prob = model.predict_proba(X_val)
print('The class order in the model: ')
print(model.classes_)
print('Examples of predicted probabilities for the above classes')
print('prediction: ')
print(prediction[0:5])
print('y_val: ')
print(y_val)
print('prediction_prob')
print(predict_prob[0:5])

print('Confusion matrix:\n')
print(pd.crosstab(prediction, y_val), '\n')
print(f"Accuracy: {np.mean(prediction == y_val):.3f}")
false_positive_rate = (np.sum((prediction==0)&(y_val==1)))/(np.sum(y_val == 1))
false_negative_rate = (np.sum((prediction==1)&(y_val==0)))/(np.sum(y_val == 0))
print(false_positive_rate)
print(false_negative_rate)

##ROC
print('F1 Score: %.3f' % f1_score(y_val, prediction))
print('Precision: %.3f' % precision_score(y_val, prediction))
print('Recall: %.3f' % recall_score(y_val, prediction))
print('Accuracy: %.3f' % accuracy_score(y_val, prediction))

fpr, tpr, threshold = roc_curve(y_val, predict_prob[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve')
plt.show()
```

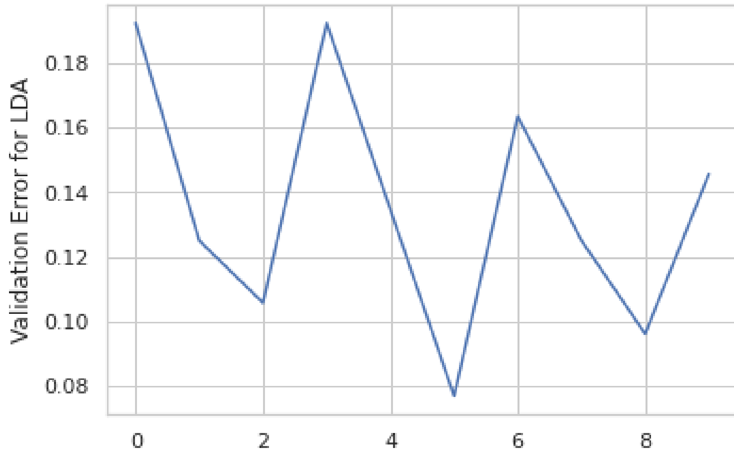
```
Error rate [Validation Error] for LDA 0.11217948717948718
The class order in the model:
[0 1]
Examples of predicted probabilities for the above classes
prediction:
[1 1 0 0 0]
y_val:
724     0
581     0
705     0
783     1
80      0
       ..
512     0
276     0
449     1
538     1
1010    0
Name: Lead, Length: 312, dtype: uint8
prediction_prob
[[0.44323422 0.55676578]
 [0.47854959 0.52145041]
```

```
 [0.93659667 0.06340333]
 [0.86883622 0.13116378]
 [0.79014576 0.20985424]]
Confusion matrix:

Lead      0   1
row_0
0       232  22
1        13  45

Accuracy: 0.888
0.3283582089552239
0.053061224489795916
F1 Score: 0.720
Precision: 0.776
Recall: 0.672
Accuracy: 0.888
```



In [ ]:

```
#Step1 cross validation
n_folds = 10

cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
misclassification = np.zeros(n_folds)
counter = 0

for train_index, val_index in cv.split(X):

  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  # Train model for every n_folds
  model = skl_da.LinearDiscriminantAnalysis()
  model.fit(X_train, y_train)
  prediction = model.predict(X_val)
  misclassification[counter] = np.mean(prediction != y_val)
  counter = counter + 1


plt.plot(misclassification)
plt.ylabel('Validation Error for LDA')
plt.show()
misclassification_mean = np.mean(misclassification)
print('Error rate [Validation Error] for LDA '+ str(misclassification_mean))
```

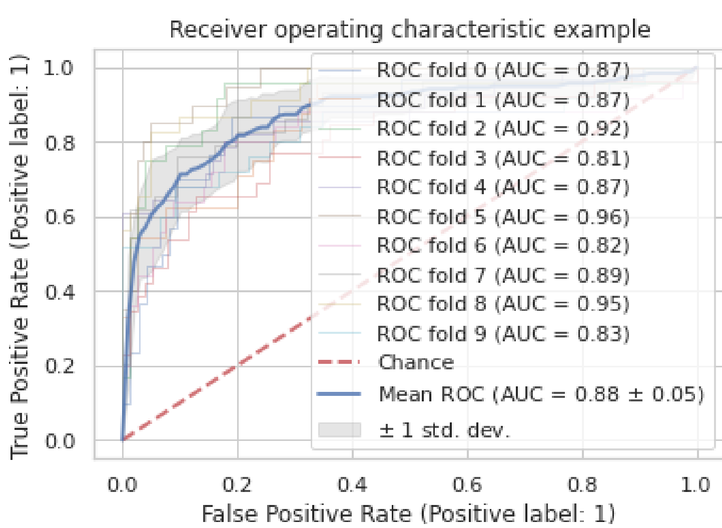



```
Error rate [Validation Error] for LDA 0.13571695294996267
```

In [ ]:

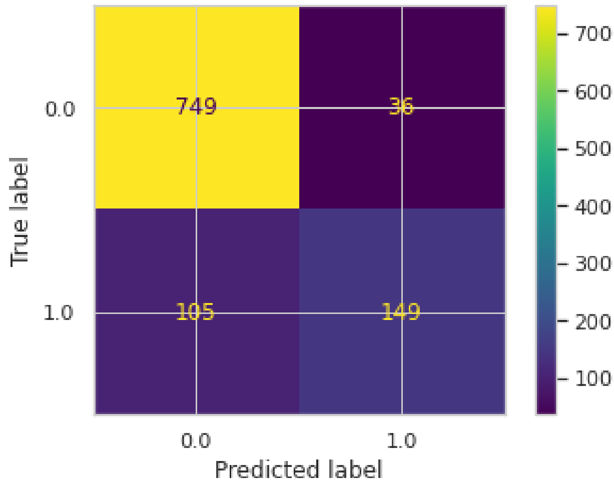
```python
from matplotlib.rcsetup import validate_aspect
# #############################################################################
# Classification and ROC analysis


# Run classifier with cross-validation and plot ROC curves
n_folds = 10
cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
model_lda = skl_da.LinearDiscriminantAnalysis()

tprs = []
aucs = []
prediction_all = np.zeros(1)
val_all = np.zeros(1)
misclassification = []
val_all = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, val) in enumerate(cv.split(X)):
    model_lda.fit(X.iloc[train], y.iloc[train])
    viz = RocCurveDisplay.from_estimator(
        model_lda,
        X.iloc[val],
        y.loc[val],
        name="ROC fold {}".format(i),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    prediction = model_lda.predict(X.iloc[val])
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
    misclassification.append(np.mean(prediction != y.iloc[val]))
    prediction_all = np.concatenate((prediction_all, prediction))
    val_all = np.concatenate((val_all, y.iloc[val]))

ax.plot([0, 1], [0, 1], linestyle="--", lw=2, color="r", label="Chance", alpha=0.8)

prediction_all = prediction_all[1:]
vall_all = val_all[1:]
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
```

```
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color="b",
    label=r"Mean ROC (AUC = %0.2f $\pm$ %0.2f)" % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label=r"$\pm$ 1 std. dev.",
)

ax.set(
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="Receiver operating characteristic example",
)
ax.legend(loc="lower right")
plt.show()
print('\n ########### Metrics about Logistic Regression ##########')
print('Missclassification Rate: %0.6f ' % np.mean(misclassification))
print('Accuracy: %.3f\n' % accuracy_score(val_all, prediction_all))

ConfusionMatrixDisplay.from_predictions(val_all , prediction_all)
false_positive_rate = (np.sum((prediction_all==0)&(val_all==1)))/(np.sum(val_all ==
false_negative_rate = (np.sum((prediction_all==1)&(val_all==0)))/(np.sum(val_all ==
print("False Positive (Female): ", false_positive_rate)
print("False Negative (Male): ", false_negative_rate)
print('F1 Score Female: %.3f' % f1_score(val_all, prediction_all))
print('Precision Female: %.3f' % precision_score(val_all, prediction_all))
# print('Recall Female: %.3f' % recall_score(val_all, prediction_all))
```




```
 ########### Metrics about Logistic Regression ##########
Missclassification Rate: 0.135717
Accuracy: 0.864
```

```
False Positive (Female):  0.41338582677165353
False Negative (Male):  0.045859872611464965
F1 Score Female: 0.679
Precision Female: 0.805
```



QDA

Data preprocession LogReg

In [ ]:
```
data = pd.read_csv('/content/drive/My Drive/SML/train.csv')
#### Convert LEAD from categorical to numberic [0,1]
data = pd.get_dummies(data,columns=['Lead'])
data = data.rename(columns={'Lead_Female':'Lead'})
data = data.drop(columns='Lead_Male')
```

In [ ]:
```
# define dataset
X = data.drop(columns=['Lead'])
y = data['Lead']
np.random.seed(1)
X_train, X_val, y_train, y_val = skl_ms.train_test_split(X,y,train_size=0.75, random
```

Hypertuning

In [ ]:
```
from sklearn.metrics import classification_report
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

#List Hyperparameters that we want to tune.
model = skl_da.QuadraticDiscriminantAnalysis()

reg_param =[0,0.1, 0.2, 0.3, 0.4, 2,5]
#Convert to dictionary
hyperparameters =  dict(reg_param=reg_param)

#Create new model object
model_qda_hy = skl_da.QuadraticDiscriminantAnalysis()
#Use GridSearch

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model_qda_hy, param_grid=hyperparameters, n_job
```

```
#Fit the model
grid_result = grid_search.fit(X_train,y_train)


#Print The value of best Hyperparameters
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

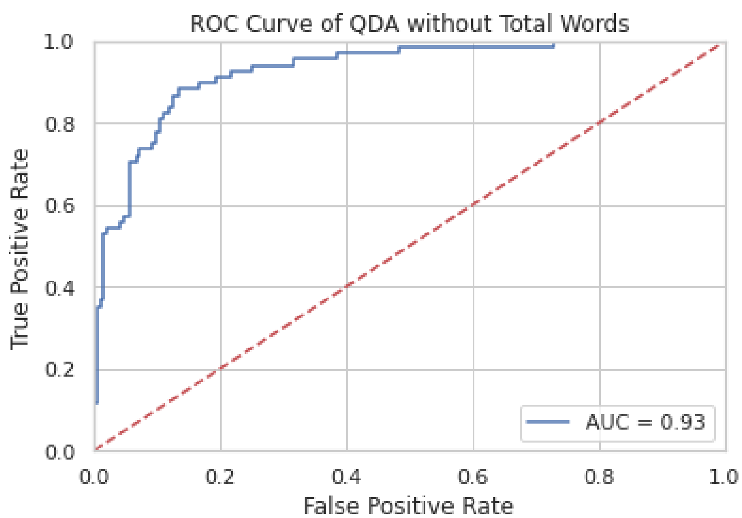
```
Best: 0.887529 using {'reg_param': 0.2}
0.856738 (0.052048) with: {'reg_param': 0}
0.887096 (0.032753) with: {'reg_param': 0.1}
0.887529 (0.033450) with: {'reg_param': 0.2}
0.885814 (0.033396) with: {'reg_param': 0.3}
0.881108 (0.035225) with: {'reg_param': 0.4}
0.745837 (0.004551) with: {'reg_param': 2}
0.745837 (0.004551) with: {'reg_param': 5}
```

In [ ]:

```
np.random.seed(1)

# Normalizing the dataset
newdata = data.copy().drop(columns=['Lead','Total words', 'Gross'])#, 'Age Lead',  '
newdata = skl_pre.normalize(newdata, axis = 0)#.iloc[:, 0:8]

normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mean
leadcolumn = data['Lead']
normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

# Dividing the dataset into train and test
trainIndex = np.random.choice(normalized_data.shape[0], size = int(len(normalized_da
train = normalized_data.iloc[trainIndex]
test = normalized_data.iloc[~normalized_data.index.isin(trainIndex)]

X_train = train.copy().drop(columns=['Lead'])
y_train = train['Lead']
X_val = test.copy().drop(columns=['Lead'])
y_val = test['Lead']

#Step2 QDA
model = skl_da.QuadraticDiscriminantAnalysis()
model.fit(X_train, y_train)
prediction = model.predict(X_val)
predict_prob = model.predict_proba(X_val)
err = np.mean(prediction != y_val)
print('Error rate when gross considered for QDA '+ str(err))

print('Confusion matrix:\n')
print(pd.crosstab(prediction, y_val), '\n')
print(f"Accuracy: {np.mean(prediction == y_val):.3f}")
false_positive_rate = (np.sum((prediction==0)&(y_val==1)))/(np.sum(y_val == 1))
false_negative_rate = (np.sum((prediction==1)&(y_val==0)))/(np.sum(y_val == 0))
print("False Positive: ", false_positive_rate)
print("False Negative: ", false_negative_rate)

##ROC
print('F1 Score: %.3f' % f1_score(y_val, prediction))
print('Precision: %.3f' % precision_score(y_val, prediction))
print('Recall: %.3f' % recall_score(y_val, prediction))
```

```
print('Accuracy: %.3f' % accuracy_score(y_val, prediction))
print(y_val.describe())
fpr, tpr, threshold = roc_curve(y_val, predict_prob[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of QDA without Total Words')
plt.show()
```

```
Error rate when gross considered for QDA 0.12307692307692308
Confusion matrix:

Lead     0   1
row_0
0      181  21
1       11  47

Accuracy: 0.877
False Positive:  0.3088235294117647
False Negative:  0.057291666666666664
F1 Score: 0.746
Precision: 0.810
Recall: 0.691
Accuracy: 0.877
count    260.000000
mean       0.261538
std        0.440320
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: Lead, dtype: float64
```

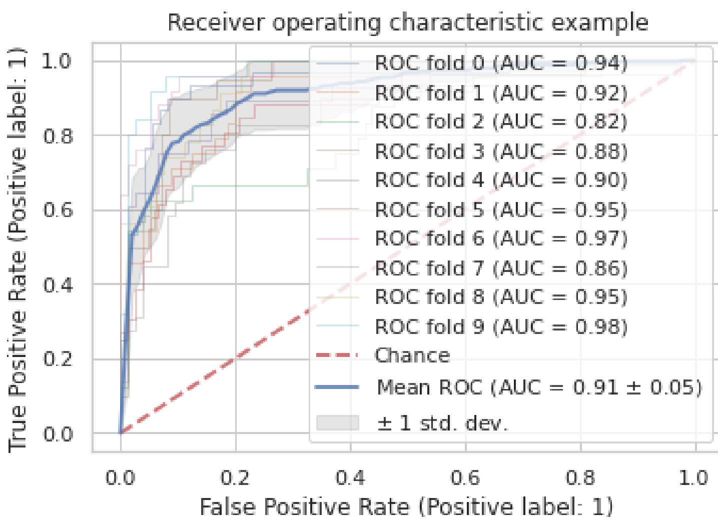


In [ ]:

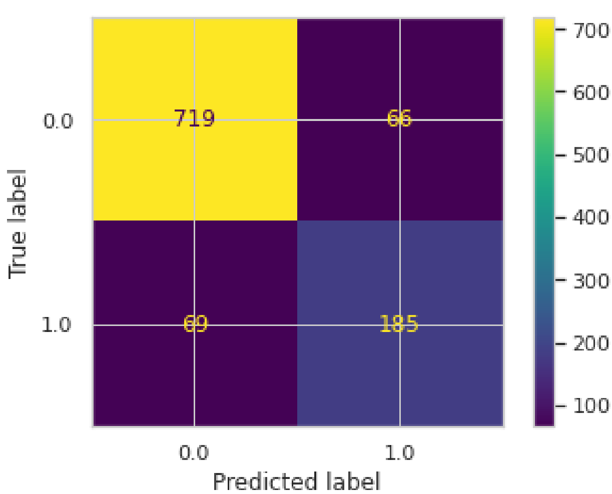
```
from matplotlib.rcsetup import validate_aspect
# ##############################################################################
# Classification and ROC analysis
```

```
# Run classifier with cross-validation and plot ROC curves
n_folds = 10
cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
model_qda = skl_da.QuadraticDiscriminantAnalysis()

tprs = []
aucs = []
prediction_all = np.zeros(1)
val_all = np.zeros(1)
misclassification = []
val_all = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, val) in enumerate(cv.split(X)):
    model_qda.fit(X.iloc[train], y.iloc[train])
    viz = RocCurveDisplay.from_estimator(
        model_qda,
        X.iloc[val],
        y.loc[val],
        name="ROC fold {}".format(i),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    prediction = model_qda.predict(X.iloc[val])
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
    misclassification.append(np.mean(prediction != y.iloc[val]))
    prediction_all = np.concatenate((prediction_all, prediction))
    val_all = np.concatenate((val_all, y.iloc[val]))

ax.plot([0, 1], [0, 1], linestyle="--", lw=2, color="r", label="Chance", alpha=0.8)

prediction_all = prediction_all[1:]
vall_all = val_all[1:]
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color="b",
    label=r"Mean ROC (AUC = %0.2f $\pm$ %0.2f)" % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label=r"$\pm$ 1 std. dev.",
)

ax.set(
```

```
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="Receiver operating characteristic example",
)
ax.legend(loc="lower right")
plt.show()
print('\n ########### Metrics about Logistic Regression ##########')
print('Missclassification Rate: %0.6f ' % np.mean(misclassification))
print('Accuracy: %.3f\n' % accuracy_score(val_all, prediction_all))

ConfusionMatrixDisplay.from_predictions(val_all , prediction_all)
false_positive_rate = (np.sum((prediction_all==0)&(val_all==1)))/(np.sum(val_all ==
false_negative_rate = (np.sum((prediction_all==1)&(val_all==0)))/(np.sum(val_all ==
print("False Positive (Female): ", false_positive_rate)
print("False Negative (Male): ", false_negative_rate)
print('F1 Score Female: %.3f' % f1_score(val_all, prediction_all))
print('Precision Female: %.3f' % precision_score(val_all, prediction_all))
# print('Recall Female: %.3f' % recall_score(val_all, prediction_all))
```




```
 ########### Metrics about Logistic Regression ##########
Missclassification Rate: 0.129873
Accuracy: 0.870

False Positive (Female):  0.27165354330708663
False Negative (Male):  0.08407643312101191
F1 Score Female: 0.733
Precision Female: 0.737
```



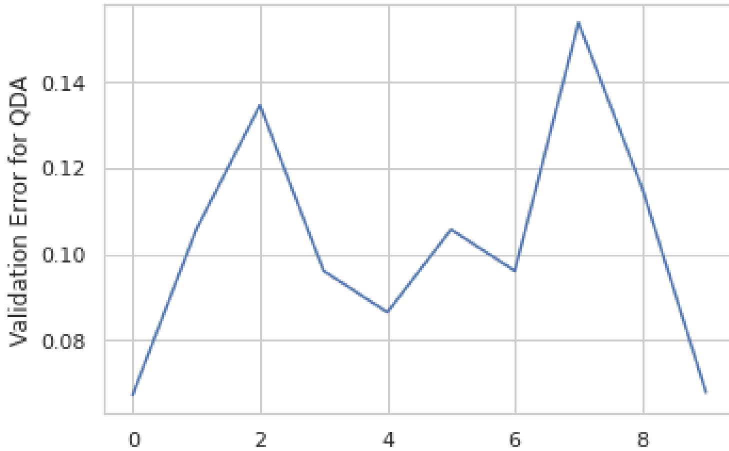
In [ ]:

```
################################################################################
```

```
############################################## Cross Validation for QDA ###########
###################################################################################
#STep 0 FEATURE SELECTION X and y
#np.random.seed(1)

# Normalizing the dataset
newdata = data.copy().drop(columns=['Lead', 'Total words', 'Gross'])#, 'Age Lead',

newdata = skl_pre.normalize(newdata, axis = 0) # .iloc[:, 0:8]

normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mean

# leadcolumn = data['Lead']
# normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

#X = normalized_data#.drop(columns=['Lead'])
X = data.drop(columns=['Lead', 'Gross', 'Total words'])
y = data['Lead']



n_folds = 10

cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
misclassification = np.zeros(n_folds)
prediction_all = np.zeros(1)
y_val_all =  np.zeros(1)
counter = 0
for train_index, val_index in cv.split(X):
  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]
  y_val_all = np.concatenate((y_val_all, y_val))
  # Train model for every n_folds
  model_LogReg = skl_da.QuadraticDiscriminantAnalysis()
  model_LogReg.fit(X_train, y_train)
  prediction = model_LogReg.predict(X_val)
  prediction_all = np.concatenate((prediction_all, prediction))
  misclassification[counter] = np.mean(prediction != y_val)
  counter = counter + 1


prediction_all = prediction_all[1:]
y_val_all = y_val_all[1:]
print(len(prediction_all))
print(prediction_all[:20])
print(misclassification)
plt.plot(misclassification)
plt.ylabel('Validation Error for QDA')
plt.show()
misclassification_mean = np.mean(misclassification)
print('Error rate [Validation Error] for QDA '+ str(misclassification_mean))

#Confussion matrix
predict_prob = model_LogReg.predict_proba(X_val)
print('The class order in the model: ')
print(model_LogReg.classes_)
print('Examples of predicted probabilities for the above classes')
print(predict_prob[0:5])

prediction = np.empty(len(X_val), dtype=object)
prediction = np.where(predict_prob[:, 0]>=0.5, 'Female','Male')
print(prediction[0:5])

print('Confusion matrix:\n')
```

```
print(pd.crosstab(prediction_all, y_val_all), '\n')
print(f"Accuracy: {np.mean(prediction_all == y_val_all):.3f}")
```

```
1039
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1.]
[0.06730769 0.10576923 0.13461538 0.09615385 0.08653846 0.10576923
 0.09615385 0.15384615 0.11538462 0.06796117]
```




```
Error rate [Validation Error] for QDA 0.10294996265870053
The class order in the model:
[0 1]
Examples of predicted probabilities for the above classes
[[0.91916168 0.08083832]
 [0.96831482 0.03168518]
 [0.98618482 0.01381518]
 [0.8961218  0.1038782 ]
 [0.82921302 0.17078698]]
['Female' 'Female' 'Female' 'Female' 'Female']
Confusion matrix:

col_0  0.0  1.0
row_0
0.0    742   64
1.0     43  190

Accuracy: 0.897
```

In [ ]:

```
normalized_data.describe()
normalized_data.head()
# data.head()
```

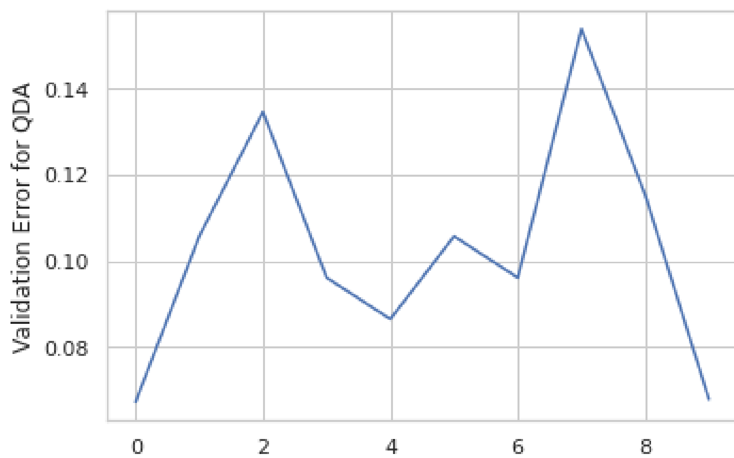
Out[ ]:

	0	1	2	3	4	5	6	7	8	9
0	0.014761	0.013760	0.002996	0.007139	0.030948	0.038006	0.014322	0.037097	0.035468	0.03513
1	0.014879	0.012348	0.010648	0.032126	0.031041	0.030404	0.028502	0.028183	0.024577	0.044301
2	0.001513	0.005758	0.006875	0.024987	0.030529	0.007601	0.016761	0.030614	0.031000	0.03513
3	0.010476	0.021028	0.022913	0.042835	0.031056	0.015202	0.029079	0.025372	0.018013	0.02520
4	0.012858	0.023443	0.027507	0.028557	0.030839	0.030404	0.013805	0.032595	0.037703	0.027497

In [ ]:

```
#Step1 cross validation
n_folds = 10
```

```
cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
misclassification = np.zeros(n_folds)
counter = 0
for train_index, val_index in cv.split(X):

  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  # Train model for every n_folds
  model = skl_da.QuadraticDiscriminantAnalysis()
  model.fit(X_train, y_train)
  prediction = model.predict(X_val)
  misclassification[counter] = np.mean(prediction != y_val)
  counter = counter + 1


plt.plot(misclassification)
plt.ylabel('Validation Error for QDA')
plt.show()
misclassification_mean = np.mean(misclassification)
print('Error rate [Validation Error] for QDA '+ str(misclassification_mean))
```




```
Error rate [Validation Error] for QDA 0.10294996265870053
```

KNN Model

Normalizing

In [ ]:

```
################################################################################
##################################  NORMALIZING FOR KNN   ##########################

np.random.seed(1)
# Normalizing the dataset
newdata = data.copy().drop(columns=['Lead','Total words', 'Gross'])
# pd.get_dummies(newdata, columns=['Lead'])
newdata = skl_pre.normalize(newdata.iloc[:, 0:8], axis = 0)

normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mean
leadcolumn = data['Lead']
normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

# Dividing the dataset into train and test
trainIndex = np.random.choice(normalized_data.shape[0], size = int(len(normalized_da
train = normalized_data.iloc[trainIndex]
test = normalized_data.iloc[~normalized_data.index.isin(trainIndex)]
```

Hyperparameter tuning

In [ ]:

```
##############################################################################
###################################K-NN Classifier##########################
##############################################################################

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]

#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)

#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
#Fit the model
best_model = clf.fit(X_train,y_train)

#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 2
Best n_neighbors: 5
```

KNN Model

In [ ]:

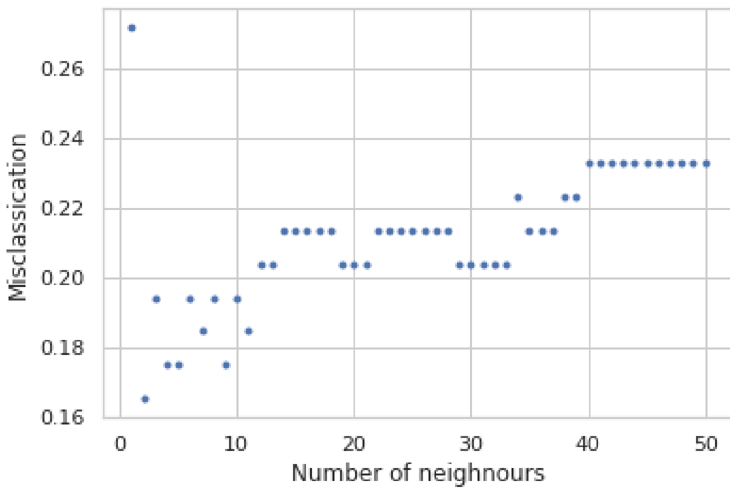
```
model_knn = skl_nb.KNeighborsClassifier(n_neighbors=5,leaf_size=1, p=2)
model_knn.fit(X_train,y_train)
prediction = model_knn.predict(X_val)

predict_prob = model_knn.predict_proba(X_val)
err = np.mean(prediction != y_val)
print('Error rate when gross considered for KNN '+ str(err))

print('Confusion matrix:\n')
print(pd.crosstab(prediction, y_val), '\n')
print(f"Accuracy: {np.mean(prediction == y_val):.3f}")
false_positive_rate = (np.sum((prediction==0)&(y_val==1)))/(np.sum(y_val == 1))
false_negative_rate = (np.sum((prediction==1)&(y_val==0)))/(np.sum(y_val == 0))
print("False Positive: ", false_positive_rate)
print("False Negative: ", false_negative_rate)
```

```
Error rate when gross considered for KNN 0.17475728155339806
Confusion matrix:

Lead     0   1
row_0
0       74  14
1        4  11
```

```
Accuracy: 0.825
False Positive:  0.56
False Negative:  0.05128205128205128
```

In [ ]:

```
misclassification = []
for k in range(50):
  model_knn = skl_nb.KNeighborsClassifier(n_neighbors=k+1)
  model_knn.fit(X_train,y_train)
  prediction = model_knn.predict(X_val)
  misclassification.append(np.mean(prediction != y_val))

K = np.linspace(1,50,50)
plt.plot(K, misclassification, '.')
plt.ylabel('Misclassication')
plt.xlabel('Number of neighnours')
plt.show()
```

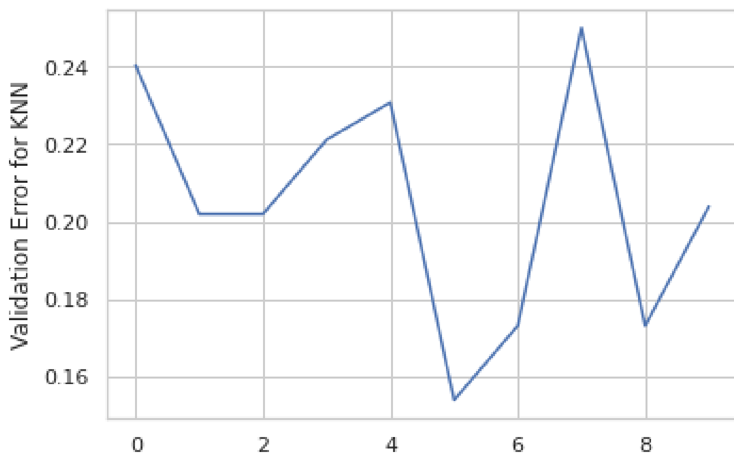


Cross Validation

In [ ]:

```
###############Cross-validation for Knn######################################
n_folds = 10

cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
misclassification = np.zeros(n_folds)
prediction_all = np.zeros(1)
counter = 0
y_val_all =  np.zeros(1)
for train_index, val_index in cv.split(X):

  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]
  y_val_all = np.concatenate((y_val_all, y_val))

  # Train model for every n_folds
  model_knnCross = skl_nb.KNeighborsClassifier(n_neighbors=12,leaf_size=1, p=2)
  model_knnCross.fit(X_train, y_train)
  prediction = model_knnCross.predict(X_val)

  prediction_all = np.concatenate((prediction_all, prediction))
  misclassification[counter] = np.mean(prediction != y_val)
  counter = counter + 1

prediction_all = prediction_all[1:]
y_val_all = y_val_all[1:]
```

```
plt.plot(misclassification)
plt.ylabel('Validation Error for KNN')
plt.show()
misclassification_mean = np.mean(misclassification)
print('Error rate [Validation Error] for KNN '+ str(misclassification_mean))

#Confussion matrix
predict_prob = model_knnCross.predict_proba(X_val)
print('The class order in the model: ')
print(model_knnCross.classes_)
print('Examples of predicted probabilities for the above classes')
print(predict_prob[0:5])

prediction = np.empty(len(X_val), dtype=object)
prediction = np.where(predict_prob[:, 0]>=0.5, 'Female','Male')
print(prediction[0:5])

print('Confusion matrix:\n')
print(pd.crosstab(prediction_all, y_val_all), '\n')
print(f"Accuracy: {np.mean(prediction_all == y_val_all):.3f}")
```




```
Error rate [Validation Error] for KNN 0.2050037341299477
The class order in the model:
[0 1]
Examples of predicted probabilities for the above classes
[[0.91666667 0.08333333]
 [0.83333333 0.16666667]
 [1.         0.        ]
 [0.83333333 0.16666667]
 [0.66666667 0.33333333]]
['Female' 'Female' 'Female' 'Female' 'Female']
Confusion matrix:

col_0  0.0  1.0
row_0
0.0    781  209
1.0      4   45

Accuracy: 0.795
```

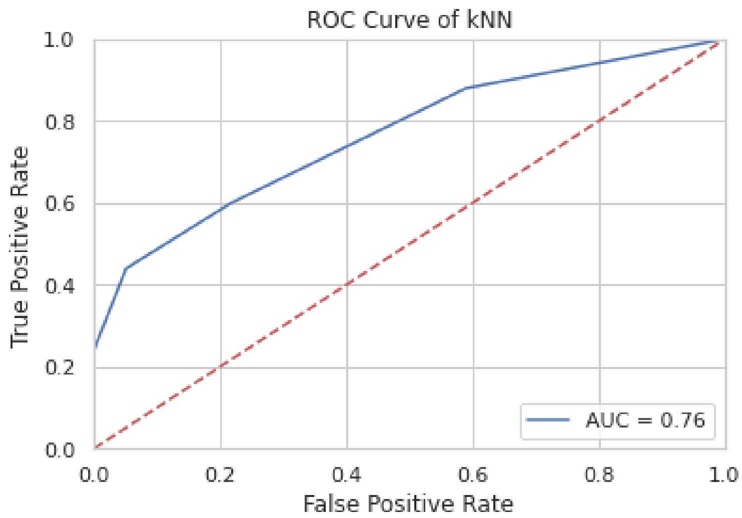
Error rate, Confusion Matrix, Accuracy, ROC

In [ ]:

```
# Defining the x and y axis of test and train
knn_model = skl_nb.KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X=X_train, y=y_train)
y_predict = knn_model.predict(X_val)
print("Test error rate is: ", np.mean(y_predict != y_val))
```

```
# Confusion Matrix
pd.crosstab(y_predict, y_val)
#Confussion matrix
predict_prob = model_knnCross.predict_proba(X_val)
print('The class order in the model: ')
print(model_knnCross.classes_)
print('Examples of predicted probabilities for the above classes')
print(predict_prob[0:5])

prediction = np.empty(len(X_val), dtype=object)
prediction = np.where(predict_prob[:, 0]>=0.5, 'Female','Male')
print(prediction[0:5])

print('Confusion matrix:\n')
print(pd.crosstab(prediction_all, y_val_all), '\n')
print(f"Accuracy: {np.mean(prediction_all == y_val_all):.3f}")

print('F1 Score: %.3f' % f1_score(y_val, y_predict))
print('Precision: %.3f' % precision_score(y_val, y_predict))
print('Recall: %.3f' % recall_score(y_val, y_predict))
print('Accuracy: %.3f' % accuracy_score(y_val, y_predict))

y_scores = knn_model.predict_proba(X_val)
fpr, tpr, threshold = roc_curve(y_val, y_scores[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of kNN')
plt.show()
```

```
Test error rate is:  0.17475728155339806
The class order in the model:
[0 1]
Examples of predicted probabilities for the above classes
[[0.91666667 0.08333333]
 [0.83333333 0.16666667]
 [1.         0.        ]
 [0.83333333 0.16666667]
 [0.66666667 0.33333333]]
['Female' 'Female' 'Female' 'Female' 'Female']
Confusion matrix:

col_0  0.0  1.0
row_0
0.0    781  209
1.0      4   45

Accuracy: 0.795
F1 Score: 0.550
Precision: 0.733
Recall: 0.440
Accuracy: 0.825
```



Random Forest

In [ ]:

```
###############################################################################
################################################# Random Forest ###################
###############################################################################

np.random.seed(1)

# Normalizing the dataset
newdata = data.copy().drop(columns=['Gross', 'Mean Age Male', 'Mean Age Female', 'Ag
# pd.get_dummies(newdata, columns=['Lead'])
newdata = skl_pre.normalize(newdata.iloc[:, 0:8], axis = 0)

normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mean
leadcolumn = data['Lead']
normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

# Dividing the dataset into train and test
trainIndex = np.random.choice(normalized_data.shape[0], size = int(len(normalized_da
train = normalized_data.iloc[trainIndex]
test = normalized_data.iloc[~normalized_data.index.isin(trainIndex)]
# print(train.head())
# print(test.head())
# print(train.describe())
# print(test.describe())
```

In [ ]:

```
# Defining the x and y axis of test and train
x_train = train.copy().drop(columns=['Lead'])
y_train = train['Lead']
x_test = test.copy().drop(columns=['Lead'])
y_test = test['Lead']

# model = tree.DecisionTreeClassifier(max_leaf_nodes=5)
model_randomforest = RandomForestClassifier(max_leaf_nodes=5, max_features='auto')
model_randomforest.fit(X=x_train, y=y_train)
y_predict = model_randomforest.predict(x_test)
print("Test error rate is: ", np.mean(y_predict != y_test))

# Confusion Matrix
pd.crosstab(y_predict, y_test)
```

```
# print('F1 Score: %.3f' % f1_score(y_test, y_predict))
# print('Precision: %.3f' % precision_score(y_test, y_predict))
# print('Recall: %.3f' % recall_score(y_test, y_predict))
# print('Accuracy: %.3f' % accuracy_score(y_test, y_predict))

# y_scores = model_randomforest.predict_proba(x_test)
# fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])
# roc_auc = auc(fpr, tpr)

# plt.title('Receiver Operating Characteristic')
# plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
# plt.legend(loc = 'lower right')
# plt.plot([0, 1], [0, 1],'r--')
# plt.xlim([0, 1])
# plt.ylim([0, 1])
# plt.ylabel('True Positive Rate')
# plt.xlabel('False Positive Rate')
# plt.title('ROC Curve of Random Forest')
# plt.show()
```

```
Test error rate is:  0.2423076923076923
```

Out[ ]:

Lead	0	1
row_0		
0	192	63
1	0	5

In [ ]:

```
# Bagging
baggingmodel = BaggingClassifier()
baggingmodel.fit(x_train, y_train)
error = np.mean(baggingmodel.predict(x_test) != y_test)
print(error)
```

```
0.17307692307692307
```

AdaBoost

Normalizing and diving the data

In [110…

```
np.random.seed(1)
# Normalizing the dataset
newdata = data.copy().drop(columns=['Lead','Total words', 'Gross'])
# pd.get_dummies(newdata, columns=['Lead'])
newdata = skl_pre.normalize(newdata.iloc[:, 0:8], axis = 0)

normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mean
leadcolumn = data['Lead']
normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

# Dividing the dataset into train and test
trainIndex = np.random.choice(normalized_data.shape[0], size = int(len(normalized_da
train = normalized_data.iloc[trainIndex]
test = normalized_data.iloc[~normalized_data.index.isin(trainIndex)]
```

Hyperparameter Tuning

In [111…

```
from sklearn.datasets import make_classification
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score

model = AdaBoostClassifier()
#X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redun
# define the grid of values to search
grid = dict()
grid['n_estimators'] = [10, 50, 100, 500]
grid['learning_rate'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
# define the evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define the grid search procedure
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scori
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_
# execute the grid search
grid_result = grid_search.fit(X_train, y_train)
# summarize the best score and configuration
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
# summarize all scores that were evaluated
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.847708 using {'learning_rate': 0.1, 'n_estimators': 500}
0.743284 (0.032246) with: {'learning_rate': 0.0001, 'n_estimators': 10}
0.743284 (0.032246) with: {'learning_rate': 0.0001, 'n_estimators': 50}
0.743284 (0.032246) with: {'learning_rate': 0.0001, 'n_estimators': 100}
0.750549 (0.025637) with: {'learning_rate': 0.0001, 'n_estimators': 500}
0.744994 (0.031180) with: {'learning_rate': 0.001, 'n_estimators': 10}
0.750549 (0.025637) with: {'learning_rate': 0.001, 'n_estimators': 50}
0.752259 (0.025757) with: {'learning_rate': 0.001, 'n_estimators': 100}
0.746265 (0.004908) with: {'learning_rate': 0.001, 'n_estimators': 500}
0.760390 (0.012702) with: {'learning_rate': 0.01, 'n_estimators': 10}
0.748829 (0.006055) with: {'learning_rate': 0.01, 'n_estimators': 50}
0.746692 (0.005205) with: {'learning_rate': 0.01, 'n_estimators': 100}
0.774958 (0.018873) with: {'learning_rate': 0.01, 'n_estimators': 500}
0.757826 (0.013236) with: {'learning_rate': 0.1, 'n_estimators': 10}
0.777950 (0.020689) with: {'learning_rate': 0.1, 'n_estimators': 50}
0.813886 (0.027903) with: {'learning_rate': 0.1, 'n_estimators': 100}
0.847708 (0.038581) with: {'learning_rate': 0.1, 'n_estimators': 500}
0.812171 (0.031388) with: {'learning_rate': 1.0, 'n_estimators': 10}
0.838739 (0.043094) with: {'learning_rate': 1.0, 'n_estimators': 50}
0.834449 (0.039767) with: {'learning_rate': 1.0, 'n_estimators': 100}
0.826757 (0.046519) with: {'learning_rate': 1.0, 'n_estimators': 500}
```

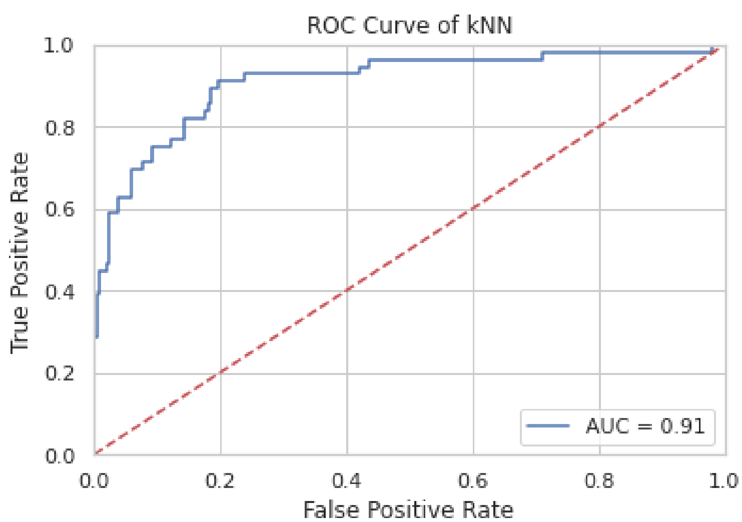
Training the model

In [112…

```
#data['lead'] = data['lead'].apply({-1:'Male', 1:'Female'}.get)


model = AdaBoostClassifier(n_estimators= 500, learning_rate=1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_val)
print("Test error = ")
print(np.mean(y_pred != y_val))
```

```
# Confusion Matrix
pd.crosstab(y_pred, y_val)

print('F1 Score: %.3f' % f1_score(y_val, y_pred))
print('Precision: %.3f' % precision_score(y_val, y_pred))
print('Recall: %.3f' % recall_score(y_val, y_pred))
print('Accuracy: %.3f' % accuracy_score(y_val, y_pred))

y_scores = model.predict_proba(X_val)
fpr, tpr, threshold = roc_curve(y_val, y_scores[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of kNN')
plt.show()
```

```
Test error =
0.11923076923076924
F1 Score: 0.705
Precision: 0.755
Recall: 0.661
Accuracy: 0.881
```



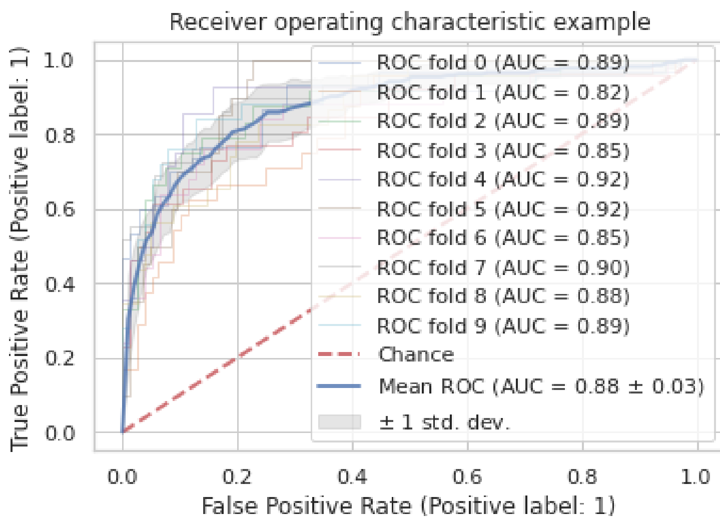
Cross Validation

In [113…

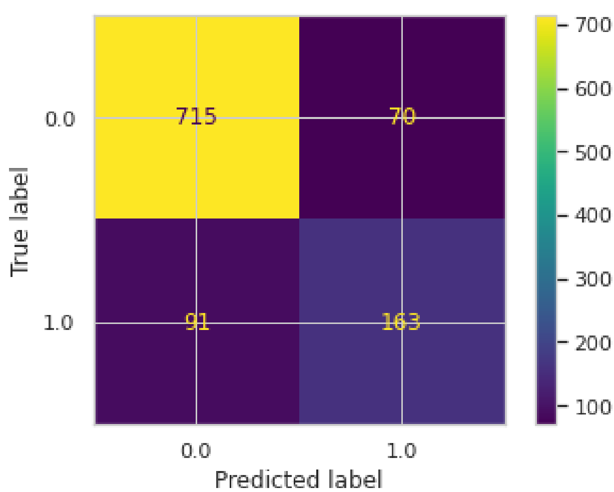
```
from matplotlib.rcsetup import validate_aspect
# #############################################################################
# Classification and ROC analysis


# Run classifier with cross-validation and plot ROC curves
n_folds = 10
cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=True)
model_adaCross = AdaBoostClassifier(n_estimators= 500, learning_rate=1.0)

tprs = []
aucs = []
prediction_all = np.zeros(1)
```

```
val_all = np.zeros(1)
misclassification = []
val_all = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, val) in enumerate(cv.split(X)):
    model_adaCross.fit(X.iloc[train], y.iloc[train])
    viz = RocCurveDisplay.from_estimator(
        model_adaCross,
        X.iloc[val],
        y.loc[val],
        name="ROC fold {}".format(i),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    prediction = model_adaCross.predict(X.iloc[val])
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
    misclassification.append(np.mean(prediction != y.iloc[val]))
    prediction_all = np.concatenate((prediction_all, prediction))
    val_all = np.concatenate((val_all, y.iloc[val]))

ax.plot([0, 1], [0, 1], linestyle="--", lw=2, color="r", label="Chance", alpha=0.8)

prediction_all = prediction_all[1:]
vall_all = val_all[1:]
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color="b",
    label=r"Mean ROC (AUC = %0.2f $\pm$ %0.2f)" % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label=r"$\pm$ 1 std. dev.",
)

ax.set(
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="Receiver operating characteristic example",
)
ax.legend(loc="lower right")
plt.show()
print('\n ########### Metrics about AdaBoost ##########')
print('Missclassification Rate: %0.6f ' % np.mean(misclassification))
```

```
print('Accuracy: %.3f\n' % accuracy_score(val_all, prediction_all))

ConfusionMatrixDisplay.from_predictions(val_all , prediction_all)
false_positive_rate = (np.sum((prediction_all==0)&(val_all==1)))/(np.sum(val_all ==
false_negative_rate = (np.sum((prediction_all==1)&(val_all==0)))/(np.sum(val_all ==
print("False Positive (Female): ", false_positive_rate)
print("False Negative (Male): ", false_negative_rate)
print('F1 Score Female: %.3f' % f1_score(val_all, prediction_all))
print('Precision Female: %.3f' % precision_score(val_all, prediction_all))
# print('Recall Female: %.3f' % recall_score(val_all, prediction_all))
```




```
 ########### Metrics about AdaBoost ##########
Missclassification Rate: 0.154920
Accuracy: 0.845

False Positive (Female):  0.35826771653543305
False Negative (Male):  0.08917197452229299
F1 Score Female: 0.669
Precision Female: 0.700
```

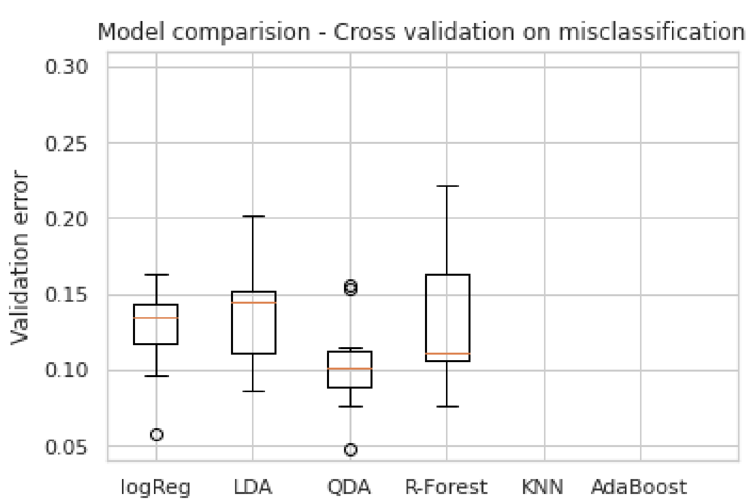


Model Selection

In [114…

```
#STep 0 FEATURE SELECTION X and y
X = data.drop(columns=['Lead', 'Gross', 'Total words'])#'Gross', 'Mean Age Male', 'M
y = data['Lead']

# # # Normalizing the dataset
```

```
# newdata = data.copy().drop(columns=['Lead', 'Gross', 'Total words'])
# # pd.get_dummies(newdata, columns=['Lead'])
# newdata = skl_pre.normalize(newdata.iloc[:, 0:8], axis = 0)

# normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mea
# leadcolumn = data['Lead']
# normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

# X =normalized_data.copy().drop(columns=['Lead'])
# y= data['Lead']
# # Dividing the dataset into train and test
# trainIndex = np.random.choice(normalized_data.shape[0], size = int(len(normalized_
# X = normalized_data.iloc[trainIndex]
# y = normalized_data.iloc[~normalized_data.index.isin(trainIndex)]

n_fold = 10

models=[]
models.append(skl_lm.LogisticRegression(solver='liblinear',C=1, penalty='l2'))
models.append(skl_da.LinearDiscriminantAnalysis())
models.append(skl_da.QuadraticDiscriminantAnalysis(reg_param= 0.1))
# models.append(skl_nb.KNeighborsClassifier(n_neighbors=12,leaf_size=1, p=2))
models.append(RandomForestClassifier(n_estimators=200, max_depth=10, max_features=7)
# models.append(AdaBoostClassifier())

cv = skl_ms.KFold(n_splits=n_fold, random_state=1, shuffle=True)
misclassification = np.zeros((n_fold, len(models)))

for i, (train_index, val_index) in enumerate(cv.split(X)):
  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  for m in range(np.shape(models)[0]):
    model = models[m]
    model.fit(X_train, y_train)
    prediction = model.predict(X_val)
    misclassification[i,m] = np.mean(prediction != y_val)

plt.ylim(0.04, 0.31)
plt.boxplot(misclassification)
plt.title('Model comparision - Cross validation on misclassification')
plt.xticks(np.arange(7)+1, ('logReg','LDA','QDA','R-Forest', 'KNN', 'AdaBoost'))
plt.ylabel('Validation error')
plt.show()
print(misclassification)
```



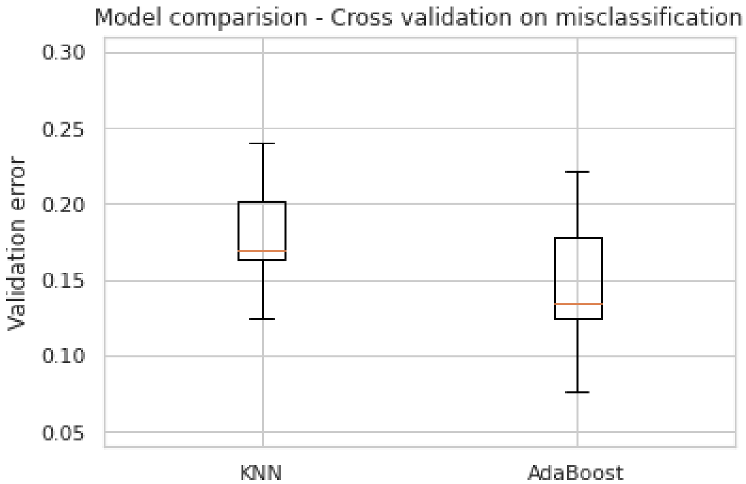

```
[[0.125      0.10576923 0.09615385 0.16346154]
```

```
[0.09615385 0.10576923 0.11538462 0.07692308]
[0.13461538 0.14423077 0.08653846 0.11538462]
[0.11538462 0.125      0.07692308 0.10576923]
[0.16346154 0.16346154 0.10576923 0.10576923]
[0.15384615 0.20192308 0.15384615 0.19230769]
[0.05769231 0.08653846 0.04807692 0.10576923]
[0.13461538 0.14423077 0.09615385 0.16346154]
[0.13461538 0.15384615 0.10576923 0.22115385]
[0.14563107 0.14563107 0.15533981 0.09708738]]
```

In [115…

```
#STep 0 FEATURE SELECTION X and y
# X = data.drop(columns=['Lead', 'Gross', 'Total words'])#'Gross', 'Mean Age Male',
# y = data['Lead']

# # Normalizing the dataset
newdata = data.copy().drop(columns=['Lead', 'Gross', 'Total words'])

newdata = skl_pre.normalize(newdata.iloc[:, 0:8], axis = 0)

normalized_data = pd.DataFrame(newdata) # columns=data.columns.drop(['Gross', 'Mean
leadcolumn = data['Lead']
normalized_data = pd.concat([normalized_data, leadcolumn], axis=1)

X =normalized_data.copy().drop(columns=['Lead'])
y= data['Lead']


n_fold = 10

models=[]
# models.append(skl_lm.LogisticRegression(solver='liblinear'))
# models.append(skl_da.LinearDiscriminantAnalysis())
# models.append(skl_da.QuadraticDiscriminantAnalysis())
models.append(skl_nb.KNeighborsClassifier(n_neighbors=12,leaf_size=1, p=2))
# models.append(RandomForestClassifier(max_leaf_nodes=5, max_features='auto'))#,oob_
models.append(AdaBoostClassifier())

cv = skl_ms.KFold(n_splits=n_fold, random_state=1, shuffle=True)
misclassification = np.zeros((n_fold, len(models)))

for i, (train_index, val_index) in enumerate(cv.split(X)):
  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  for m in range(np.shape(models)[0]):
    model = models[m]
    model.fit(X_train, y_train)
    prediction = model.predict(X_val)
    misclassification[i,m] = np.mean(prediction != y_val)

plt.ylim(0.04, 0.31)
plt.boxplot(misclassification)
plt.title('Model comparision - Cross validation on misclassification')
plt.xticks(np.arange(2)+1, ('KNN', 'AdaBoost'))
plt.ylabel('Validation error')
plt.show()
print(misclassification)

# plt.subplot(1, 2, 1) # row 1, col 2 index 1
# plt.boxplot(misclassification)
# plt.title('Model comparision - Cross validation on misclassification')
# plt.xticks(np.arange(4)+1, ('logReg','LDA','QDA','R-Forest'))
# plt.ylabel('Validation error')
```

```
# plt.figure(figsize=(200,200))
# plt.show()
# plt.subplot(1, 2, 2) # index 2
# plt.boxplot(misclassification)
# plt.title('Model comparision - Cross validation on misclassification')
# plt.xticks(np.arange(2)+1, ('KNN', 'AdaBoost'))
# plt.ylabel('Validation error')
# plt.figure(figsize=(200,200))

# plt.show()
```




```
[[0.14423077 0.19230769]
 [0.125      0.13461538]
 [0.17307692 0.13461538]
 [0.16346154 0.125     ]
 [0.24038462 0.22115385]
 [0.21153846 0.19230769]
 [0.16346154 0.07692308]
 [0.17307692 0.125     ]
 [0.21153846 0.13461538]
 [0.16504854 0.0776699 ]]
```

In [ ]:

```
########          MODEL ON TEST data         ##########

X = data.copy().drop(columns=['Lead','Total words', 'Gross'])
y = data['Lead']
# DATA preprocessing
# Train model with whole train data
model = skl_da.QuadraticDiscriminantAnalysis()
model.fit(X, y)
# Prediction of female male
data_test = data_test.copy().drop(columns=['Total words', 'Gross'])
y_hat_prediction = model.predict(data_test);

# add new colum prediction
data_test['Lead_prediction'] = y_hat_prediction
# store it
data_test.to_csv('/content/drive/My Drive/SML/data_test_prediction_QDA.csv')
```