MACHINE LEARNING CAPSTONE PROJECT

USING DENTAL METRICS TO PREDICT GENDER

MADE BY:

NIKET CHOUDHARY

DATA SCIENCE COURSE

06TH AUGUST 2024

TABLE OF CONTENTS

1 PROBLEM STATEMENT

1.1 ABOUT DATASET

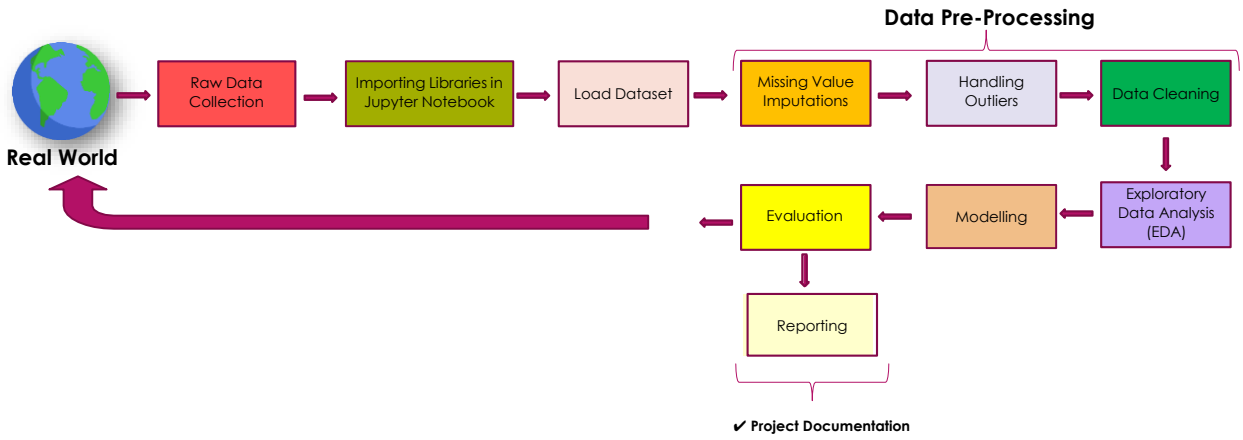
Project Title	Using Dental Metrics to Predict Gender
Tech Stack	Python, Machine Learning
Domain	Healthcare
Project Difficulty level	Rookie/Basic
Programming Language Used	Python
Tools Used	Jupyter Notebook, MS-Excel

The goal of this project is to analyze the data and predict, based on a combination of dental features that describes the gender of the person.

1.2 BACKGROUND & SCOPE

- Forensic medicine is an interesting area of study. Forensic dentistry is a branch of forensic medicine.
- During natural calamities or due to some other reasons, many times, it will not be possible to find out the gender of the deceased person.
- In such cases, certain measurements of the tooth will be taken (as bones and teeth do not decay easily) and gender will be determined.

1.3 ARCHITECTURE



1.4 DATASET INFORMATION

Age: The person's age in years

Gender: The person's sex (male, female) - Target Variable

SampleID and Sl No.: The sampleID & SL No. represents individuals unique ID

Inter-canine distance intraoral , inter-canine distance casts, right canine cast, left canine cast, etc. These features represent the measurement of the oral teeth - the independent variables

1.5 STEP-BY-STEP APPROACH TO FOLLOW

Step 1: Raw data collection : Click the hyperlink to download the dataset - Hyperlink

Step 2: Importing the necessary packages in JupyterNotebook/ Any IDE

Note: For JupyterNotebook Installation kindly follow the documentation

Packages involved:-

```
import pandas as pd                    #Used to load the dataset
import Numpy as np                     #Used to perform mathematical operations
import matplotlib.pyplot as plt        #Used to visualize the data
import seaborn as sns
```

Step 3: Import the dataset using pandas

```
Variable_name = pd.read_csv("Dentistry.csv")
```

Step 4: Data Preprocessing

i) Identify and handle missing values

ii) Encoding categorical data

i.e from sklearn.preprocessing import LabelEncoder

iii) Split independent and dependent variables i.e. X and Y

iv) Normalize the X variable

from sklearn.preprocessing import Normalizer #all the values will fall in the range

[0,1] or sometimes[-1 , +1]

Step 5: Exploratory Data Analysis

i) You need to check the correlation of the data using Heatmap between X-to-X features and X-to-Y features to understand the relationship and collinearity issues between the features. (seaborn library)

Step 6: Model Building

i) Drop the unwanted independent variables which you see not important for model building.

ii) Drop the independent features which are highly correlated to each other

iii) Split the Data into Train and Test set

from sklearn.preprocessing import train_test_split

iv) Use Logistic Regression, Decision Tree classifier, Random Forest classifier and XGBoost classifier.

Step 7: Evaluation

You need to evaluate the model based on the models evaluation metrics i.e. Confusion matrix(Accuracy), ROC curve and AUC curve to check model accuracy and plot them

Step 8: Make a documentation file of the project for submission

2 CODE DESCRIPTION

2.1 PACKAGE IMPORT AND DATASET LOADING

```
import pandas as pd                    #Used to load the dataset

import numpy as np                     #Used to perform mathematical operations

import matplotlib.pyplot as plt        #Used to visualize the data

import seaborn as sns

#Save the dataset csv file in D: Drive

ds = pd.read_csv("D:/Dentistry Dataset.csv")
```

Description: The code imports essential libraries for data analysis and visualization in Python. It then loads a dataset from a CSV file located on the D: drive into a pandas DataFrame, which will be used for further analysis and visualization.

2.2 IDENTIFICAITON OF MISSING VALUES AND HANDLING THEM

```
#Identify and handle missing values

MissingValuesCount = ds.isna().sum()

print(MissingValuesCount)
```

Description: The code checks for missing values in the dataset and prints the number of missing values for each column.

2.3 DROP COLUMNS WITH NO VALUES

```
#drop column with no value

ds = ds.drop('Sample ID', axis=1)

ds
```

Description: This code removes the column named 'Sample ID' from the dataset because it contains no values. The updated dataset is then stored back in `ds`.

2.4 ENCODE COLUMNS

```
#Encoding categorical data

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

ds['Gender'] = le.fit_transform(ds['Gender'])

ds
```

Description: The code encodes categorical data by converting the 'Gender' column into numerical values using LabelEncoder from scikit-learn. This transformation allows the dataset to be used in machine learning models.

2.5 EXTRACTING INDEPENDENT AND DEPENDENT VARIABLES

```
#Extracting Independent and Dependent Variables

x = ds.iloc[:,3:13]

y = ds.iloc[:,2]


#In the above code, we have taken [3:12] for x because these are our independent variables.

#And we have taken 2 i.e. Gender for y variable because its our dependent or target variable.
```

Description: The code extracts independent and dependent variables from the dataset. The comments clarify that column 3 to 12 are chosen for x as they represent independent features, while column 2 is chosen for y as it represents the target variable.

2.6 EXTRACTING INDEPENDENT AND DEPENDENT VARIABLES

```
#Extracting Independent and Dependent Variables

x = ds.iloc[:,3:13]

y = ds.iloc[:,2]


#In the above code, we have taken [3:12] for x because these are our independent variables.

#And we have taken 2 i.e. Gender for y variable because its our dependent or target variable.
```

Description: The code extracts independent and dependent variables from the dataset. The comments clarify that column 3 to 12 are chosen for x as they represent independent features, while column 2 is chosen for y as it represents the target variable.

2.7 NORMALIZE THE INDEPENDENT VARIABLES

```
#Normalize the X variable

from sklearn.preprocessing import Normalizer

norm = Normalizer()

x_normalized = norm.fit_transform(x)

x_normalized
```

Description: The code normalizes the independent variables, scaling them so that each feature has a unit norm.

2.8 NORMALIZE THE INDEPENDENT VARIABLES

```
# Compute correlation matrix for X

CorrelationMatrixX = x_normalized_df.corr()

# Compute correlation of features with the target variable

CorrelationXY = x_normalized_df.copy()

CorrelationXY['Gender'] = y.values

CorrelationMatrixXY = CorrelationXY.corr()
```

Description: The code calculates correlation matrices. It computes the correlation matrix for the normalized independent variables to understand their relationships. It then creates a new DataFrame that includes the target variable and calculates the correlation between the independent variables and the target variable.

2.9 CREATE FEATURE-TO-FEATURE CORRELATION HEATMAP

```
#Create Feature-to-Feature Correlation Heatmap

plt.figure(figsize=(10,10))

sns.heatmap(CorrelationMatrixX,annot=True,fmt=".0%")
```

```
plt.title('Feature-to-Feature Correlation Heatmap')

plt.show()
```

Description: The code creates and displays a heatmap that visualizes the correlations between features in the dataset. The heatmap includes annotations to show the correlation values as percentages.

2.10 CREATE FEATURE-TO-TARGET CORRELATION HEATMAP

```
#Create Feature-to-Target Correlation Heatmap

plt.figure(figsize=(10,10))

sns.heatmap(CorrelationMatrixXY,annot=True,fmt=".0%")

plt.title('Feature-to-Target Correlation Heatmap')

plt.show()
```

Description: The code creates and displays a heatmap that visualizes the correlations between features and the target variable. Annotations show the correlation values as percentages.

2.11 REMOVING UNWANTED FEATURES BASED ON THE FEATURES-TO-TARGET CORRELATION HEATMP

```
#Removing unwanted features based on the Features-to-Target Correlation Heatmp

#Values closer to 0 show less correlation with the Target variable. Hence removing them.

x_new = x.drop(x.columns[[0,1,6,7,8,9]],axis=1)

x_new
```

Description: The code removes features with low correlation to the target variable based on the heatmap. Features with values closer to 0 (indicating weak correlation) are dropped from the dataset, resulting in a new set of independent variables, "x_new".

2.12 REMOVING FEATURES BASED ON THE FEATURES-TO-FEATURES CORRELATION HEATMP

```
#Removing features based on the Features-to-Features Correlation Heatmp

#Since, by observing the previous feature dataset and Feature-to-Feature Correlation Heatmap, columns 0 & 1,and
```

#2 & 3 are highly correlated. Hence, keeping columns 0 and 2 only for model building.

x_final = x_new.drop(x_new.columns[[1,3]],axis=1)

x_final

Description: The code removes highly correlated features based on the feature-to-feature correlation heatmap. Columns with high correlation are dropped, resulting in a refined set of independent variables, "x_final", by keeping only the most relevant columns for model building.

2.13 REMOVING FEATURES BASED ON THE FEATURES-TO-FEATURES CORRELATION HEATMP

#Removing features based on the Features-to-Features Correlation Heatmp

#Since, by observing the previous feature dataset and Feature-to-Feature Correlation Heatmap, columns 0 & 1,and

#2 & 3 are highly correlated. Hence, keeping columns 0 and 2 only for model building.

x_final = x_new.drop(x_new.columns[[1,3]],axis=1)

x_final

Description: The code removes highly correlated features based on the feature-to-feature correlation heatmap. Columns with high correlation are dropped, resulting in a refined set of independent variables, "x_final", by keeping only the most relevant columns for model building.

2.14 SPLIT THE DATA INTO TRAIN AND TEST SET

#Split the Data into Train and Test set

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x_final,y,test_size=0.2,random_state=0)

Description: The code splits the data into training and testing sets, using 80% of the data for training and 20% for testing.

2.15 BUILD MODEL USING LOGISTIC REGRESSION

#Build model using Logistic Regression

from sklearn.linear_model import LogisticRegression

```
reg = LogisticRegression()


#Train the model using the training data

reg.fit(x_train,y_train)

print("Logistic Regression Accuracy: {:.2f}%".format(reg.score(x_test,y_test)*100))
```

Description: The code builds and trains a Logistic Regression model, then evaluates its accuracy on the test data and prints the result as a percentage.

2.16 MAKE PREDICTIONS USING LOGISTIC REGRESSION

```
#Make predictions using LogisticRegression

y_pred = reg.predict(x_test)
```

Description: The code uses the trained Logistic Regression model to make predictions on the test data.

2.17 CREATE CONFUSION MATRIX

```
#Create Confusion Matrix

from sklearn import metrics


cm = metrics.confusion_matrix(y_test,y_pred)


# Display the confusion matrix

print("Confusion Matrix:")

print(cm)
```

Description: The code creates and displays a confusion matrix to evaluate the performance of the Logistic Regression model.

2.18 FIND ACCURACY OF LOGISTIC REGRESSION

```
#Accuracy = (TP+TN)/(TP+TN+FP+FN). Library function is available in sklearn.metrics

print("Accuracy : {:.2f}%".format(metrics.accuracy_score(y_test,y_pred)*100))
```

Description: The code calculates and prints the accuracy of the Logistic Regression model using the accuracy_score function from scikit-learn.

2.19 MAKE PREDICTIONS AND COMPUTE AUC AND ROC CURVE

```
#Make Predictions and Compute AUC and ROC Curve

#Predict probabilities for the test set

y_prob = reg.predict_proba(x_test)[:, 1]  # Get probabilities for the positive class


#Compute ROC curve

fpr,tpr,thresholds = metrics.roc_curve(y_test,y_prob)


#Compute AUC

auc = metrics.roc_auc_score(y_test,y_prob)

print("AUC : {:.2f}%".format((auc)*100))


#Plot ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr,tpr,color='blue',label=f'ROC curve (area = {auc:.2f})')

plt.plot([0,1],[0,1],color='grey',linestyle='--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate (Recall)')

plt.title('Receiver Operating Characteristic (ROC)')
```

```
plt.legend(loc='lower right')

plt.show()
```

Description: The code computes and visualizes the ROC curve and AUC score for the Logistic Regression model to assess its performance in classifying the positive class.

2.20 BUILD MODEL USING DECISION TREE CLASSIFIER

```
#Build model using Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier ()


#Train the model using the training data

clf.fit(x_train,y_train)

print("Decision Tree Classifier Accuracy: {:.2f}%".format(clf.score(x_test,y_test)*100))
```

Description: The code builds and trains a Decision Tree Classifier model, then evaluates its accuracy on the test data and prints the result as a percentage.

2.21 MAKE PREDICTIONS USING DECISION TREE CLASSIFIER

```
#Make predictions using Decision Tree Classifier

y_pred = clf.predict(x_test)
```

Description: The code uses the trained Decision Tree Classifier model to make predictions on the test data.

2.22 CREATE CONFUSION MATRIX

```
#Create Confusion Matrix

from sklearn import metrics


cm = metrics.confusion_matrix(y_test,y_pred)
```

```
# Display the confusion matrix

print("Confusion Matrix:")

print(cm)
```

Description: The code creates and displays a confusion matrix to evaluate the performance of the Decision Tree Classifier model.

2.23 FIND ACCURACY OF DECISION TREE CLASSIFIER

```
#Accuracy = (TP+TN)/(TP+TN+FP+FN). Library function is available in sklearn.metrics

print("Accuracy : {:.2f}%".format(metrics.accuracy_score(y_test,y_pred)*100))
```

Description: The code calculates and prints the accuracy of the Decision Tree Classifier model using the accuracy_score function from scikit-learn.

2.24 MAKE PREDICTIONS AND COMPUTE AUC AND ROC CURVE

```
#Make Predictions and Compute AUC and ROC Curve

#Predict probabilities for the test set

y_prob = clf.predict_proba(x_test)[:, 1]  # Get probabilities for the positive class


#Compute ROC curve

fpr,tpr,thresholds = metrics.roc_curve(y_test,y_prob)


#Compute AUC

auc = metrics.roc_auc_score(y_test,y_prob)

print("AUC : {:.2f}%".format((auc)*100))


#Plot ROC curve

plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr,tpr,color='blue',label=f'ROC curve (area = {auc:.2f})')

plt.plot([0,1],[0,1],color='grey',linestyle='--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate (Recall)')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()
```

Description: The code computes and visualizes the ROC curve and AUC score for the Decision Tree Classifier model to assess its performance in classifying the positive class.

2.25 BUILD MODEL USING RANDOM FOREST CLASSIFIER

```
#Build model using Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()


#Train the model using the training data

rf.fit(x_train,y_train)

print("Random Forest Classifier Accuracy: {:.2f}%".format(rf.score(x_test,y_test)*100))
```

Description: The code builds and trains a Random Forest Classifier model, then evaluates its accuracy on the test data and prints the result as a percentage.

2.26 MAKE PREDICTIONS USING RANDOM FOREST CLASSIFIER

```
#Make predictions using Random Forest Classifier

y_pred = rf.predict(x_test)
```

Description: The code uses the trained Random Forest Classifier model to make predictions on the test data.

2.27 CREATE CONFUSION MATRIX

```
#Create Confusion Matrix

from sklearn import metrics


cm = metrics.confusion_matrix(y_test,y_pred)


# Display the confusion matrix

print("Confusion Matrix:")

print(cm)
```

Description: The code creates and displays a confusion matrix to evaluate the performance of the Random Forest Classifier model.

2.28 FIND ACCURACY OF RANDOM FOREST CLASSIFIER

```
#Accuracy = (TP+TN)/(TP+TN+FP+FN). Library function is available in sklearn.metrics

print("Accuracy : {:.2f}%".format(metrics.accuracy_score(y_test,y_pred)*100))
```

Description: The code calculates and prints the accuracy of the Random Forest Classifier model using the accuracy_score function from scikit-learn.

2.29 MAKE PREDICTIONS AND COMPUTE AUC AND ROC CURVE

```
#Make Predictions and Compute AUC and ROC Curve

#Predict probabilities for the test set

y_prob = rf.predict_proba(x_test)[:, 1]  # Get probabilities for the positive class


#Compute ROC curve

fpr,tpr,thresholds = metrics.roc_curve(y_test,y_prob)


#Compute AUC

auc = metrics.roc_auc_score(y_test,y_prob)

print("AUC : {:.2f}%".format((auc)*100))


#Plot ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr,tpr,color='blue',label=f'ROC curve (area = {auc:.2f})')

plt.plot([0,1],[0,1],color='grey',linestyle='--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate (Recall)')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()
```

Description: The code computes and visualizes the ROC curve and AUC score for the Random Forest Classifier model to assess its performance in classifying the positive class.

2.30 BUILD MODEL USING XGB CLASSIFIER

```
#Build model using XGB Classifier

from xgboost import XGBClassifier

xgb = XGBClassifier()


#Train the model using the training data

xgb.fit(x_train,y_train)

print("XGB Classifier Accuracy: {:.2f}%".format(xgb.score(x_test,y_test)*100))
```

Description: The code builds and trains a XGB Classifier model, then evaluates its accuracy on the test data and prints the result as a percentage.

2.31 MAKE PREDICTIONS USING XGB CLASSIFIER

```
#Make predictions using XGB Classifier

y_pred = xgb.predict(x_test)
```

Description: The code uses the trained XGB Classifier model to make predictions on the test data.

2.32 CREATE CONFUSION MATRIX

```
#Create Confusion Matrix

from sklearn import metrics


cm = metrics.confusion_matrix(y_test,y_pred)


# Display the confusion matrix

print("Confusion Matrix:")

print(cm)
```

Description: The code creates and displays a confusion matrix to evaluate the performance of the XGB Classifier model.

2.33 FIND ACCURACY OF XGB CLASSIFIER

```
#Accuracy = (TP+TN)/(TP+TN+FP+FN). Library function is available in sklearn.metrics

print("Accuracy : {:.2f}%".format(metrics.accuracy_score(y_test,y_pred)*100))
```

Description: The code calculates and prints the accuracy of the XGB Classifier model using the accuracy_score function from scikit-learn.

2.34 MAKE PREDICTIONS AND COMPUTE AUC AND ROC CURVE

```
#Make Predictions and Compute AUC and ROC Curve

#Predict probabilities for the test set

y_prob = xgb.predict_proba(x_test)[:, 1]  # Get probabilities for the positive class


#Compute ROC curve

fpr,tpr,thresholds = metrics.roc_curve(y_test,y_prob)


#Compute AUC

auc = metrics.roc_auc_score(y_test,y_prob)

print("AUC : {:.2f}%".format((auc)*100))


#Plot ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr,tpr,color='blue',label=f'ROC curve (area = {auc:.2f})')

plt.plot([0,1],[0,1],color='grey',linestyle='--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate (Recall)')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()
```

Description: The code computes and visualizes the ROC curve and AUC score for the XGB Classifier model to assess its performance in classifying the positive class.

3 CONCLUSION

- **Data Preparation:** The dataset was loaded, and missing values and irrelevant features were handled to clean and prepare the data.
- **Feature Selection:** Features were encoded and normalized, with unnecessary or highly correlated features removed to refine the dataset.
- **Model Building and Training:** Machine learning models, namely, Logistic Regression, Decision Tree classifier, Random Forest classifier and XGBoost Classifier were built and trained using the cleaned and prepared dataset.
- **Model Evaluation:** The models' accuracies were assessed using the test data, and confusion matrices were generated to evaluate classification performances.
- **Performance Metrics:** The ROC curve and AUC score were computed to further assess the models' ability to distinguish between classes.
- **Most Accurate Model:** Below table shows the accuracy levels of different ML models used in the project:

Sr. No.	Model	Accuracy	AUC Score
1	Logistic Regression	76.36%	85.08%
2	Decision Tree Classifier	72.73%	75.97%
3	Random Forest Classifier	74.09%	86.41%
4	**XGBoost Classifier**	**78.64%**	**86.96%**