USER MANUAL

Analysing Diabetic Patient Readmissions:
Uncovering Patterns and Insights

Table of Contents

1. Data Cleaning and Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial phase in data analysis, focusing on understanding and summarizing key characteristics of a dataset. Through visualizations and statistical techniques, EDA helps identify patterns, outliers, and relationships within the data, providing insights that guide further analysis and inform data-driven decision-making processes.

1.1. Import Dataset

1. Import Required Library: Begin by importing the necessary Python library, typically pandas, using the command `import pandas as pd`.

2. Load CSV File: Use the pandas `read_csv` function, specifying the file path or URL of the CSV file as an argument, for example, `df = pd.read_csv('file.csv')`.

3. Explore the Data: Quickly inspect the loaded data using commands like `df.head()` to display the first few rows or `df.info()` for summary information.

1.2. Subsequent Cleaning and EDA steps

Run the EDA and cleaning steps consecutively:

1. Run Cells Sequentially: Execute the Jupyter Notebook cells in sequential order by clicking "Run" or pressing Shift + Enter.

2. Check Dependencies: Ensure necessary libraries are installed. If not, use `!pip install library_name` in a cell to install.

3. Review Comments: Read comments in each cell for insights into the purpose and actions performed.

4. Understand Markdown Cells: Markdown cells provide information on sections, assumptions, and overall process; review these for context.

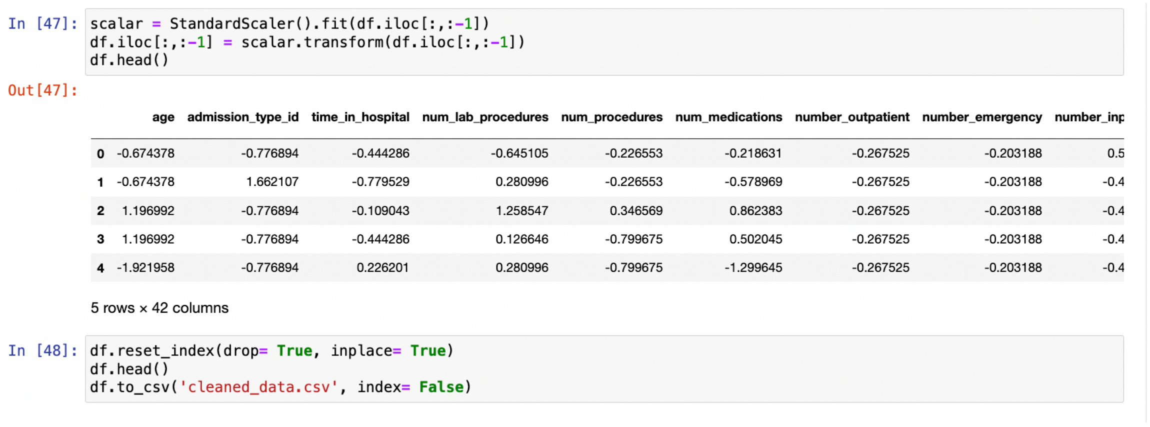
The above steps helps to effortlessly reproduce the entire EDA and cleaning process.

2. Data Pre-Processing

Data preprocessing is a crucial step in preparing raw datasets for analysis or modeling. It involves cleaning, transforming, and organizing data to enhance its quality and utility. Techniques include handling missing values, normalization, encoding categorical variables, and outlier detection. Effective preprocessing ensures accurate, reliable, and efficient downstream analyses.

2.1. Scaling The Data

Scaling data before modeling is essential for ensuring fair treatment of features with different scales. Scaling prevents dominant features from overshadowing others and helps algorithms converge faster. It enhances model performance, stability, and interpretability, particularly for distance-based methods like k-NN or gradient descent-based algorithms like SVMs and neural networks.



```
In [47]: scalar = StandardScaler().fit(df.iloc[:,:-1])
         df.iloc[:,:-1] = scalar.transform(df.iloc[:,:-1])
         df.head()
Out[47]:
```

	age	admission_type_id	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inp
0	-0.674378	-0.776894	-0.444286	-0.645105	-0.226553	-0.218631	-0.267525	-0.203188	0.5
1	-0.674378	1.662107	-0.779529	0.280996	-0.226553	-0.578969	-0.267525	-0.203188	-0.4
2	1.196992	-0.776894	-0.109043	1.258547	0.346569	0.862383	-0.267525	-0.203188	-0.4
3	1.196992	-0.776894	-0.444286	0.126646	-0.799675	0.502045	-0.267525	-0.203188	-0.4
4	-1.921958	-0.776894	0.226201	0.280996	-0.799675	-1.299645	-0.267525	-0.203188	-0.4

5 rows × 42 columns

```
In [48]: df.reset_index(drop= True, inplace= True)
         df.head()
         df.to_csv('cleaned_data.csv', index= False)
```

Output: ‘cleaned_data.csv’

The scaled data is saved into the same directory as the notebook directory. This file can be further used whenever scaled data is needed.

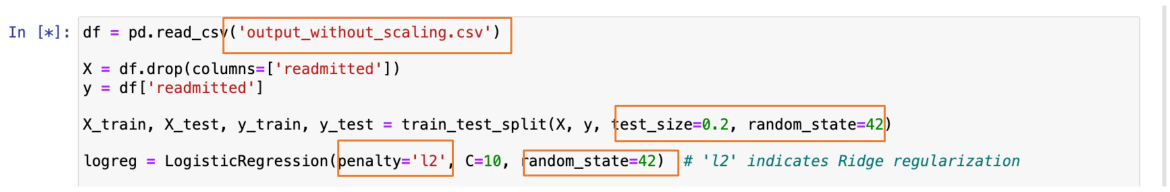
3. Machine Learning Algorithms

To address our problem statement of predicting patient readmission, we will explore a range of machine learning models and techniques. The goal is to determine the best-performing model for this specific task. Our selection includes the following algorithms

1. Logistic Regression (LASSO, Ridge, Elastic Net)
2. Decision Tree with XG Boost
3. Support Vector Machine (SVM)
4. Random Forest
5. Naïve Bayes
6. K-Nearest Neighbours (KNN)

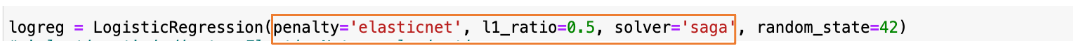
3.1. Logistic Regression

Input parameters:
Train vs Test split: We have used 80% – 20%
Input file = ‘csv_file_data_without_scaling.csv’ (csv file with data that is needed)
test_size = 0.2 (This can be adjusted as per the use case)
random_state =42 (This is used to seed the randomness)
penalty = l2 (This is for the ridge regularizartion we need)
penalty = l1 (This is for the LASSO regularizartion we need)



```
In [*]: df = pd.read_csv('output_without_scaling.csv')

        X = df.drop(columns=['readmitted'])
        y = df['readmitted']

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        logreg = LogisticRegression(penalty='l2', C=10, random_state=42)  # 'l2' indicates Ridge regularization
```

Penalty = ‘elasticnet’ (This is for elasticnet)
L1_ratio = 0.5 (This balances the L1 (Lasso) and L2 (Ridge) penalties equally)
Solver = ‘saga’ (This is is used for large datasets and supports both L1 and L2 penalties)



```
logreg = LogisticRegression(penalty='elasticnet', l1_ratio=0.5, solver='saga', random_state=42)
```

3.2. Decision Tree with XG Boost

Input parameters:

Train vs Test split: We have used 80% – 20%

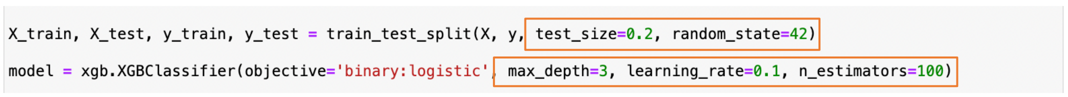
test_size = 0.2 (This can be adjusted as per the use case)

random_state =42 (This is used to seed the randomness)

max_depth = 3. (This parameter determines the maximum depth of each tree in the boosting process)

learning_rate=0.1. (It controls the step size at each iteration while moving toward a minimum of the loss function)

n_estimators=100. (This parameter defines the number of boosting rounds (trees) to be run. It represents the total number of trees in the ensemble)



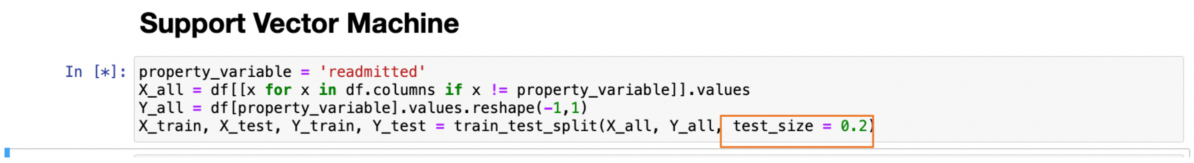
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = xgb.XGBClassifier(objective='binary:logistic', max_depth=3, learning_rate=0.1, n_estimators=100)
```

3.3. Support Vector Machine (SVM)

Input parameters:

Train vs Test split: We have used 80% – 20%

test_size = 0.2 (This can be adjusted as per the use case)



Support Vector Machine

```
In [*]: property_variable = 'readmitted'
        X_all = df[[x for x in df.columns if x != property_variable]].values
        Y_all = df[property_variable].values.reshape(-1,1)
        X_train, X_test, Y_train, Y_test = train_test_split(X_all, Y_all, test_size = 0.2)
```

Hyper tuning parameters:

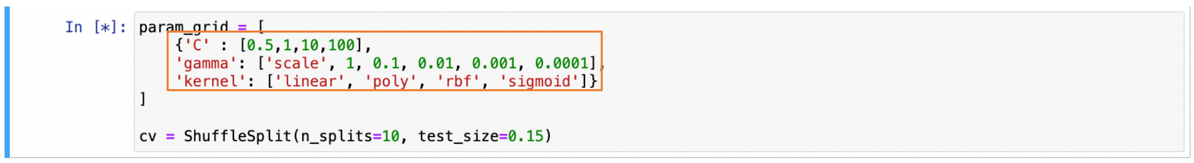
'C' : [0.5,1,10,100] (The regularization parameter in a support vector machine (SVM) model. It controls the trade-off between achieving a low training error and a low testing error)

'gamma': ['scale', 1, 0.1, 0.01, 0.001, 0.0001] (The kernel coefficient for 'rbf', 'poly', and 'sigmoid'. It defines how far the influence of a single training example reaches in a given kernel function)

'kernel': ['linear', 'poly', 'rbf', 'sigmoid'] (The kernel type to be used in the algorithm. It can be 'linear', 'poly', 'rbf' (Radial basis function), or 'sigmoid')

n_splits=10 (The number of re-shuffling & splitting iterations in the cross-validation strategy)

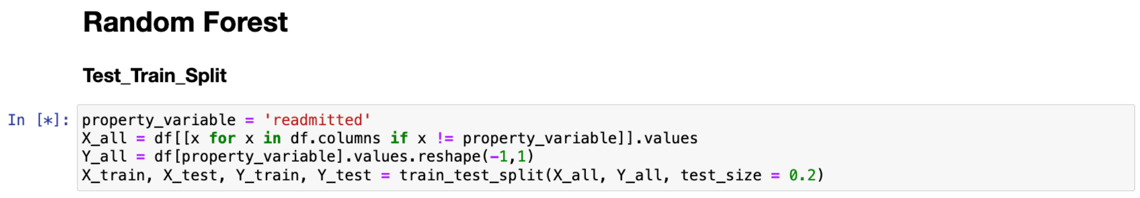
test_size=0.15 (The proportion of the dataset to include in the test split)



```
In [*]: param_grid = [
            {'C' : [0.5,1,10,100],
            'gamma': ['scale', 1, 0.1, 0.01, 0.001, 0.0001],
            'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
        ]

        cv = ShuffleSplit(n_splits=10, test_size=0.15)
```

3.4. Random Forest

Input parameters:
Train vs Test split: We have used 80% – 20%
test_size = 0.2 (This can be adjusted as per the use case)

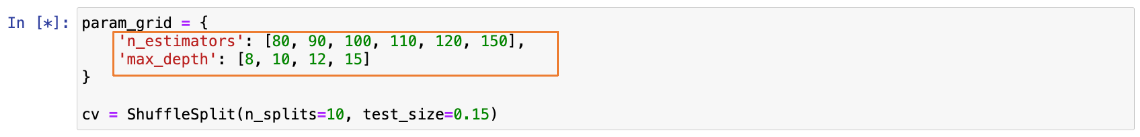


Random Forest

Test_Train_Split

```
In [*]: property_variable = 'readmitted'
        X_all = df[[x for x in df.columns if x != property_variable]].values
        Y_all = df[property_variable].values.reshape(-1,1)
        X_train, X_test, Y_train, Y_test = train_test_split(X_all, Y_all, test_size = 0.2)
```

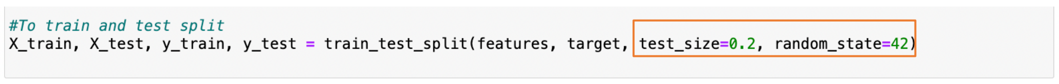
Hyper tuning parameters:
'n_estimators': [80, 90, 100, 110, 120, 150]. (This hyperparameter represents the number of trees in the forest. In the context of ensemble methods like RandomForest)
'max_depth': [8, 10, 12, 15] (This hyperparameter represents the maximum depth of the individual trees in the forest)



```
In [*]: param_grid = {
            'n_estimators': [80, 90, 100, 110, 120, 150],
            'max_depth': [8, 10, 12, 15]
        }

        cv = ShuffleSplit(n_splits=10, test_size=0.15)
```

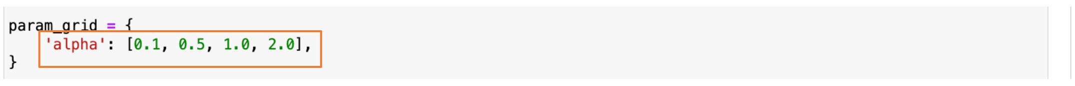
3.5. Naïve Bayes

Input parameters:
Train vs Test split: We have used 80% – 20%
test_size = 0.2 (This can be adjusted as per the use case)
random_state =42 (This is used to seed the randomness)



```
#To train and test split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

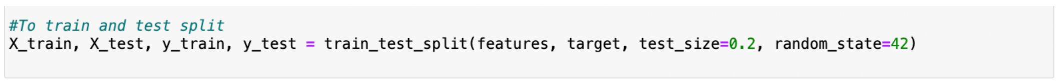
Hyper tuning parameters:
'alpha': [0.1, 0.5, 1.0, 2.0] (This is a smoothing parameter (also known as Laplace smoothing or additive smoothing). It is used to prevent zero probabilities for features not present in the learning samples and to handle unseen features in new data)



```
param_grid = {
    'alpha': [0.1, 0.5, 1.0, 2.0],
}
```

3.6. K-Nearest Neighbours (KNN)

Input parameters:
Train vs Test split: We have used 80% – 20%
test_size = 0.2 (This can be adjusted as per the use case)
random_state =42 (This is used to seed the randomness)



```
#To train and test split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

4. Final Product

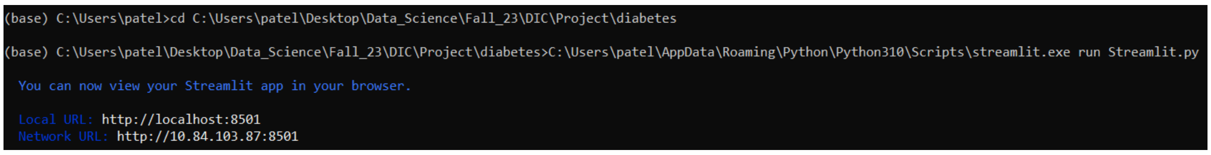
4.1. Introduction

This application uses a machine learning model that has been trained on a large dataset of patient records to predict the risk of hospital readmission. By providing information about a patient's current health conditions, the website will generate a prediction on if the patient will have hospital readmission.

4.2. Accessing the Website

To access the website and use its functionality, you'll need to run the application code first. Make sure the depend file and the code file are placed in the same folder. You can achieve this by following these steps:

- Open a terminal window or command prompt.
- Navigate to the directory where the application code is located.
- Run as given in the following sample command:



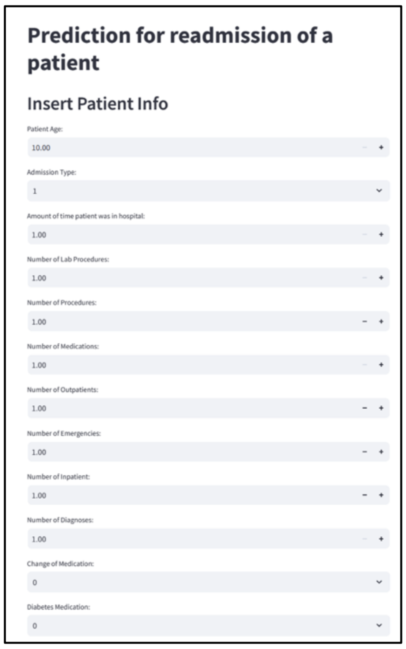
```
(base) C:\Users\patel>cd C:\Users\patel\Desktop\Data_Science\Fall_23\DIC\Project\diabetes

(base) C:\Users\patel\Desktop\Data_Science\Fall_23\DIC\Project\diabetes>C:\Users\patel\AppData\Roaming\Python\Python310\Scripts\streamlit.exe run Streamlit.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.84.103.87:8501
```

This command will start the application and launch the website.

4.3. Patient Information

On the application’s homepage, you will be prompted to enter information about the patient's current health conditions. This information may include:

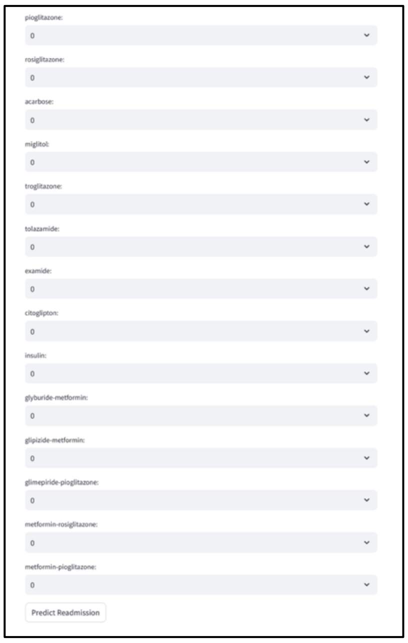
- Demographic information (age, gender, etc.)
- Medical history (Procedures done, admission type .etc)



Prediction for readmission of a patient

Insert Patient Info

Patient Age:	10.00
Admission Type:	1
Amount of time patient was in hospital:	1.00
Number of Lab Procedures:	1.00
Number of Procedures:	1.00
Number of Medications:	1.00
Number of Outpatients:	1.00
Number of Emergencies:	1.00
Number of Inpatient:	1.00
Number of Diagnoses:	1.00
Change of Medication:	0
Diabetes Medication:	0

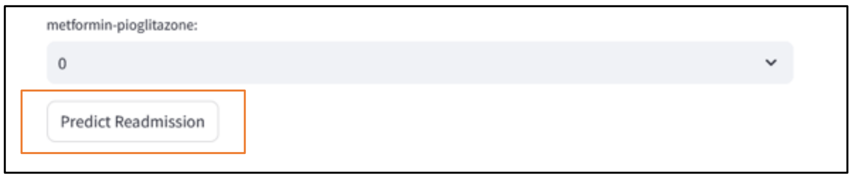


pioglitazone:	0
rosiglitazone:	0
acarbose:	0
miglitol:	0
troglitazone:	0
tolazamide:	0
examide:	0
citoglipton:	0
insulin:	0
glyburide-metformin:	0
glipizide-metformin:	0
glimepiride-pioglitazone:	0
metformin-rosiglitazone:	0
metformin-pioglitazone:	0

- You can enter this information directly into the application's form fields, like shown in the images below

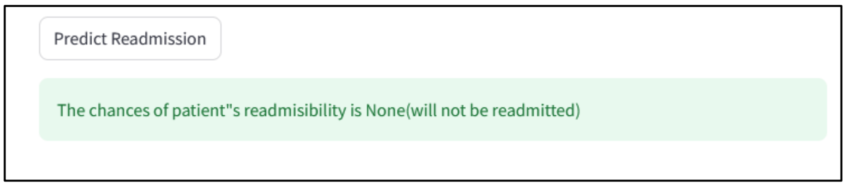
4.4. Prediction Generation:

- Once you have entered all the necessary information, click the "Predict Readmission" button.



- The application will then process the information and generate a prediction of the patient's hospital readmission.

  Sample output:



The chances of patient"s readmisibility is None(will not be readmitted)

4.5. Important Notes

- The predictions generated by the application are based on a statistical model and should not be considered definitive.
- The application is intended as a tool to assist healthcare professionals in making informed decisions about patient care.
- It is important to consider all patient information and consult with a qualified healthcare professional before making any decisions about treatment or care.