VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI - 590 018, KARNATAKA

#### vtulogo

*A Report on*

“Leveraging Predictive Analytics for Patient Care Dashboard”

(BCS358D)

in

#### **INFORMATION SCIENCE AND ENGINEERING**

#### By

**Tanya Deep USN: 1BY23IS231**

**Dr. AK Saritha**

Assisstant Professor

Department of ISE, BMSIT&M.



**ಬಿ.ಎಂ.ಎಸ್. ತಾಂತ್ರಿಕ ಮತ್ತು ವ್ಯವಸ್ಥಾಪನಾ ಮಹಾವಿದ್ಯಾಲಯ**

**BMS Institute of Technology and Management**

**(An Autonomous Institution Affiliated to VTU, Belagavi)**

**Avalahalli, Doddaballapur Main Road, Bengaluru – 560119**

**2024-2025**

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

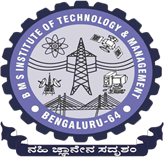
**BELAGAVI – 590 018, KARNATAKA**

**ಬಿ.ಎಂ.ಎಸ್. ತಾಂತ್ರಿಕ ಮತ್ತು ವ್ಯವಸ್ಥಾಪನಾ ಮಹಾವಿದ್ಯಾಲಯ**

**BMS Institute of Technology and Management**

**(An Autonomous Institution Affiliated to VTU, Belagavi)**

**Avalahalli, Doddaballapur Main Road, Bengaluru – 560119**

****

**CERTIFICATE**

This is to certify that the AAT **“Leveraging Predictive Analytics for Patient Care Dashboard”** is the work carried out by **Ms Tanya Deep (1BY23IS231)** of Data Visualization with Python (BCS358D) of the BMSIT&M during the year 2024-25. The report has been approved as it satisfies the academic requirements in respect AAT work for the B.EDegree.

**Signature of the Course Coordinator**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl No.** | **Topic** | **Page No.** |
| **1.** | **Abstract** | **4** |
| **2.** | **Methodology** | **5-6** |
| **3.** | **Code** | **7-9** |
| **4.** | **Output** | **10-11** |

**ABSTRACT**

This Python code develops an interactive data dashboard using **Dash** and **Plotly**, designed to visualize and analyze key aspects of a dataset related to insurance claims. The dashboard includes multiple visualizations that allow users to explore the distribution and relationships of different features within the dataset. The primary visualizations included are:

1. **Line Graph for a Single Feature**: A line graph is generated for the feature PREAUTH\_AMT (or any other chosen feature) plotted against the index, allowing for the identification of trends and patterns over the dataset. This is particularly useful for visualizing how a specific feature varies across different data points in a sequential manner.
2. **Pie Chart for Target Variable Distribution**: A pie chart is created to display the distribution of the target variable AGE, providing an overview of the proportions of different age categories within the dataset. This allows for quick insights into how the target variable is spread across the dataset.
3. **Scatter Plot for Two Features**: A scatter plot visualizes the relationship between two continuous features, PREAUTH\_AMT and CLAIM\_AMOUNT. The points are color-coded based on the target variable AGE, providing a clear representation of how these features are related and how the target variable might influence their distribution.
4. **Bar Graph for Age Distribution**: A bar graph visualizes the frequency distribution of the AGE feature, displaying the count of occurrences for each unique age category. This helps to quickly assess how many entries fall into each age group and provides a simple yet effective way to analyze categorical data.
5. **Histogram for Feature Distribution**: A histogram is generated to display the distribution of the AGE feature, with the dataset divided into bins. This visualization allows users to observe how AGE is distributed across the entire dataset, revealing insights about the frequency and spread of values within specific ranges.
6. **Random Forest Classifier for Prediction**: The dataset is split into training and testing sets, and a Random Forest Classifier model is trained to predict the target variable AGE. The model's performance is evaluated using accuracy and other classification metrics, providing an indication of how well the model can predict age categories based on other input features.
7. **Dash Application Layout**: All these visualizations are incorporated into a **Dash** app layout, making the entire analysis interactive. Users can explore different aspects of the dataset through the graphical representations, allowing them to better understand the relationships between variables, identify trends, and gain insights into the predictions made by the model. The interactivity and integration of machine learning make the dashboard an effective tool for data exploration and decision-making.

This approach not only provides powerful visual insights into the data but also combines the use of machine learning to predict outcomes, giving users a comprehensive understanding of the dataset. By integrating statistical visualizations and predictive modeling, this code serves as an example of how to analyze and visualize data in a meaningful and interactive way.

**METHODOLOGY**

The methodology for the provided code follows a systematic approach that integrates data preprocessing, machine learning model training, and interactive data visualization using Dash and Plotly. The steps are outlined as follows:

1. **Data Loading and Preprocessing:**

* Dataset Loading: The dataset (ntrarogyaseva.csv) is loaded into the Python environment using the pandas library.
* Handling Missing Data: To ensure that the dataset is clean and ready for analysis, any rows containing missing values are dropped. This step ensures that the machine learning model and visualizations are based on complete data.
* Encoding Categorical Variables: Since many machine learning algorithms require numerical input, categorical features are encoded into numerical values using pd.get\_dummies(). This step converts categorical variables into binary columns, making them suitable for machine learning models.

2. **Data Splitting:**

* Defining the Target and Features: The target variable, AGE, is identified, and the rest of the columns are considered as features for model training. The dataset is divided into X (features) and y (target).
* Train-Test Split: The dataset is split into training and testing sets using the train\_test\_split() function from sklearn.model\_selection. 80% of the data is allocated for training, while 20% is reserved for testing. The split ensures that the model can be evaluated on unseen data.

3. **Random Forest Classifier Model:**

* Model Initialization: A Random Forest Classifier is initialized with 100 estimators, and the model is trained using the training data (X\_train, y\_train).
* Model Prediction: Once trained, the model is used to predict the AGE category for the testing data (X\_test), and the predictions are stored in y\_pred.
* Model Evaluation: The accuracy and other classification metrics are computed using the accuracy\_score() and classification\_report() functions from sklearn.metrics. These metrics provide an evaluation of how well the model can predict AGE.

4. **Data Visualization:**

* Line Graph: A line graph is created to visualize trends in a single feature (PREAUTH\_AMT or other specified features). This is useful for analyzing how the feature varies with respect to the index (or sequence) of the dataset.
* Pie Chart: A pie chart is used to display the distribution of the target variable (AGE). It shows the proportion of different age categories in the dataset.
* Scatter Plot: A scatter plot visualizes the relationship between two selected features (PREAUTH\_AMT and CLAIM\_AMOUNT), color-coded by the target variable (AGE). This allows for understanding the correlation between the features and the target.
* Bar Graph: A bar graph is created to represent the frequency distribution of the AGE feature. It shows the count of different AGE categories within the dataset.
* Histogram: A histogram is plotted for the AGE feature to visualize its distribution across different ranges or bins. This helps to assess the spread of values in the dataset.
* Violin Plot: A violin plot is created to visualize the distribution of a selected feature (AGE or another specified feature). The plot shows the density distribution and allows for identifying patterns such as skewness, multimodality, etc.

5. **Building the Dash Application:**

* Dash Layout: The individual visualizations are incorporated into the layout of a Dash web application. This allows users to interact with the data and view the plots in a dynamic, web-based interface.
* Interactive Components: The visualizations are embedded within HTML and dcc.Graph components in Dash. The layout is structured with headings and visualizations arranged sequentially. Each visualization is placed in a separate html.Div to ensure proper formatting and organization on the webpage.
* App Execution: The Dash application is launched using app.run\_server(debug=True), which hosts the app locally and makes it accessible via a web browser.

6. **Model Integration with Visualizations:**

* Predictions and Insights: While the model's predictions are not explicitly displayed in the visualizations, the overall goal is to provide users with an interactive platform to explore and analyze the data. The Dash app's ability to present interactive plots complements the machine learning predictions by providing insight into the relationships between features and the target variable.

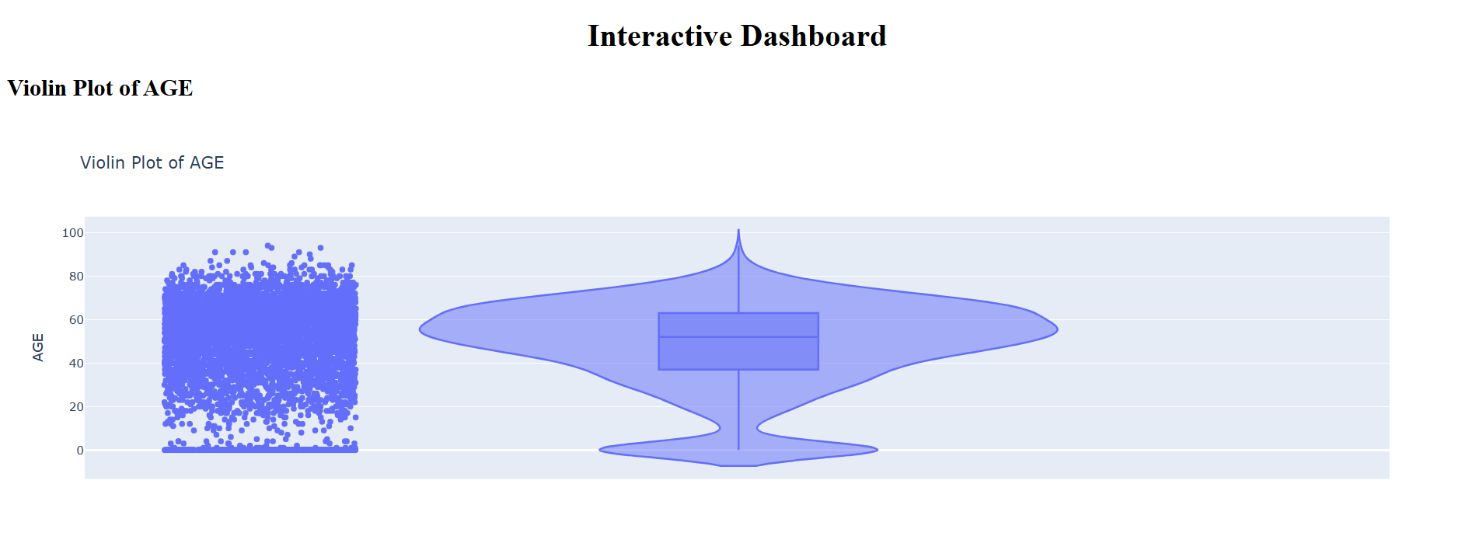
7. **Interactive Data Exploration:**

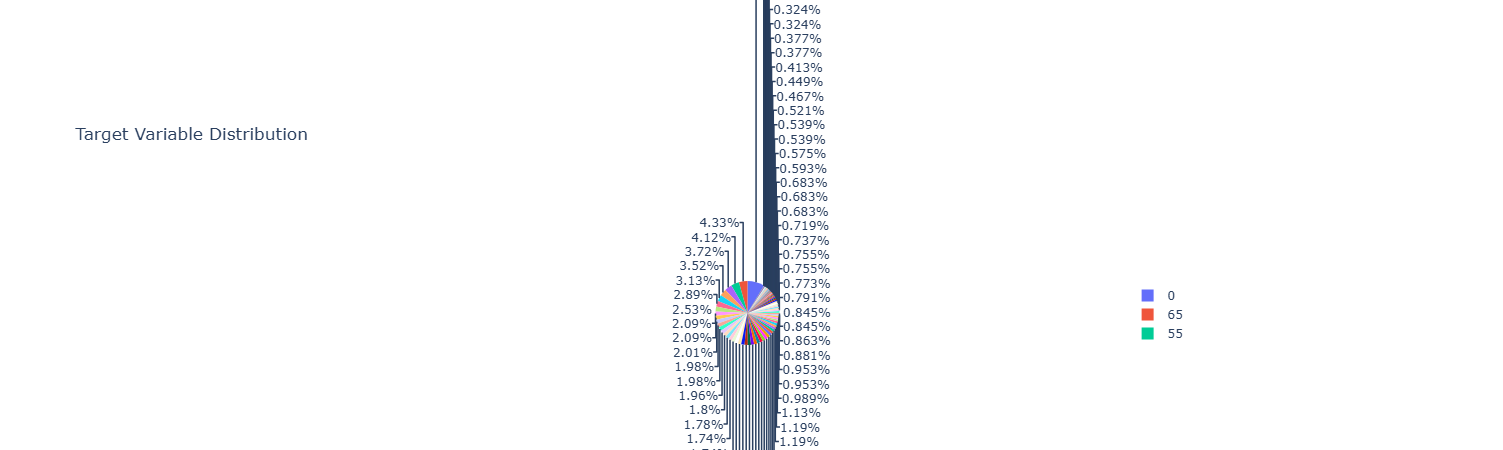
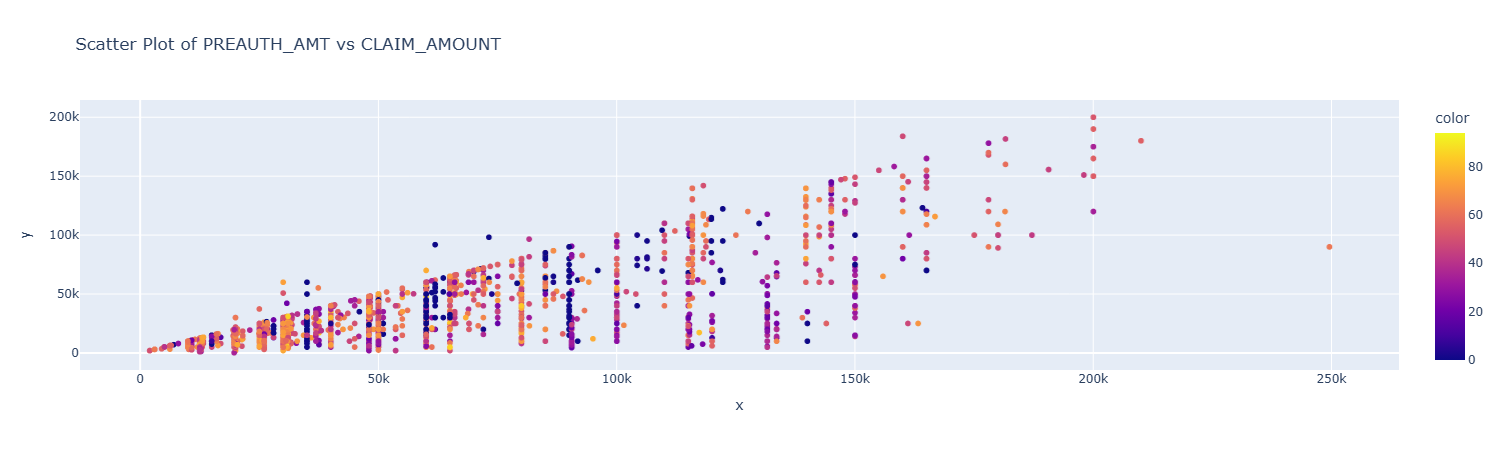
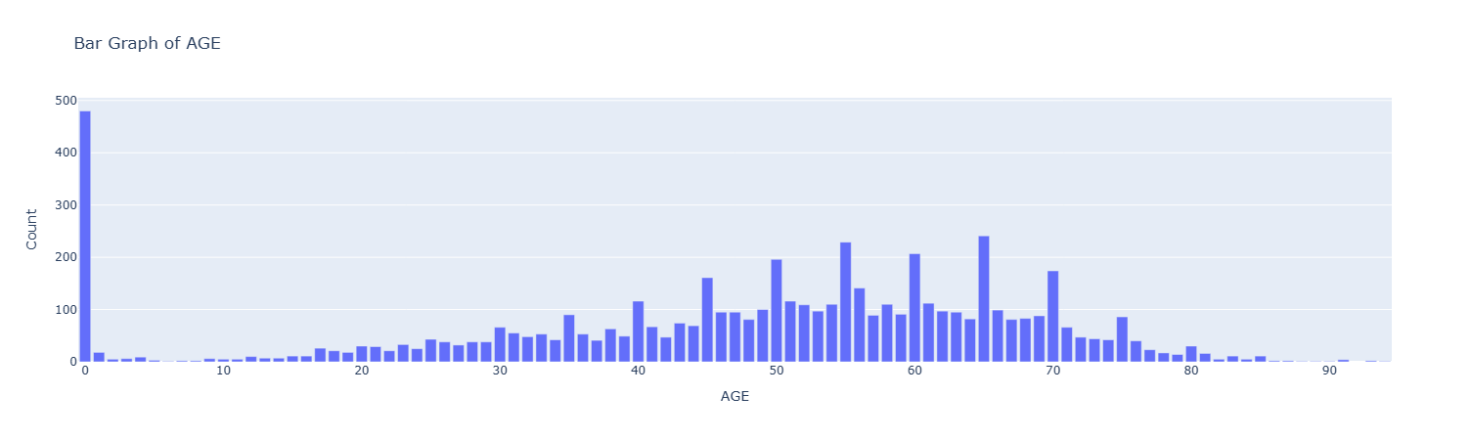
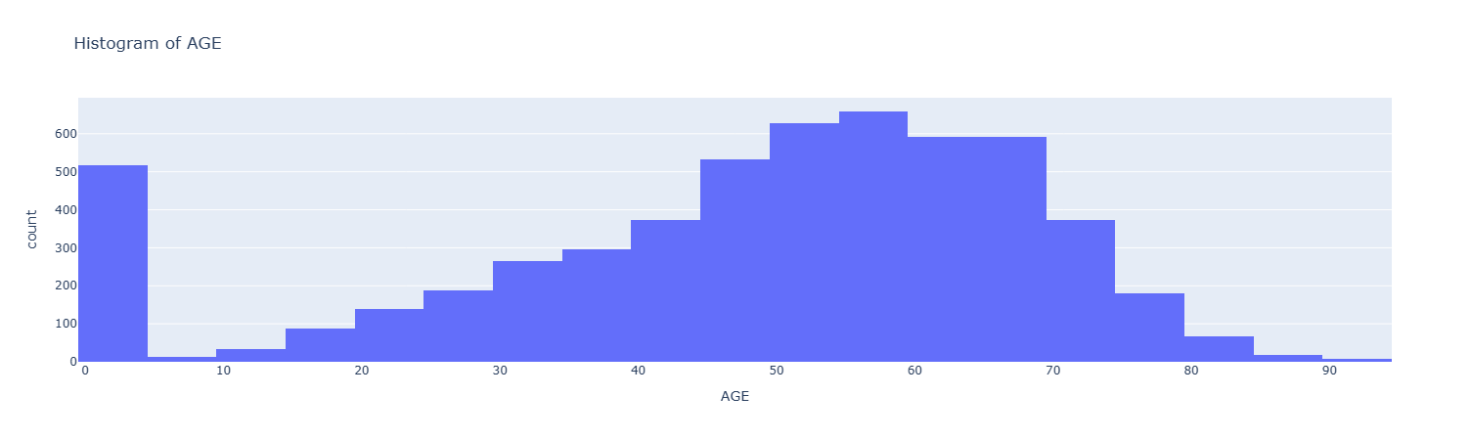
* User Interaction: The dashboard allows users to explore different aspects of the data interactively. Users can visually explore feature distributions, correlations between features, and the overall distribution of the target variable. The integration of the machine learning model's predictions allows for deeper understanding and further analysis.

**CODE**

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
import plotly.express as px  
from dash import Dash, dcc, html  
  
# Load the dataset  
data = pd.read\_csv('ntrarogyaseva.csv')  
  
# Drop rows with missing values  
data = data.dropna()  
  
# Encode categorical variables  
data = pd.get\_dummies(data, drop\_first=True)  
  
# Define the target variable and features  
target\_column = 'AGE'  
if target\_column not in data.columns:  
 raise ValueError(f"Target column '{target\_column}' not found in the dataset.")  
  
X = data.drop(target\_column, axis=1)  
y = data[target\_column]  
  
# Split the dataset into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Train Random Forest Classifier  
model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
  
# Violin Plot for Target Variable (AGE)  
violin\_fig = px.violin(data, y='AGE', box=True, points='all', title="Violin Plot of AGE")  
  
# Line Graph for a Single Feature  
feature\_line = 'PREAUTH\_AMT' # Replace with the feature you want to plot  
if feature\_line in data.columns:  
 line\_fig = px.line(data, x=data.index, y=feature\_line,  
 title=f"Line Graph of {feature\_line}")  
else:  
 line\_fig = None  
  
# Pie Chart for Target Variable Distribution  
pie\_chart\_fig = px.pie(names=y.value\_counts().index,  
 values=y.value\_counts().values,  
 title="Target Variable Distribution")  
  
# Scatter Plot for Two Features  
scatter\_feature\_1, scatter\_feature\_2 = 'PREAUTH\_AMT', 'CLAIM\_AMOUNT'  
if scatter\_feature\_1 in X.columns and scatter\_feature\_2 in X.columns:  
 scatter\_plot\_fig = px.scatter(x=X[scatter\_feature\_1],  
 y=X[scatter\_feature\_2],  
 color=y,  
 title=f"Scatter Plot of {scatter\_feature\_1} vs {scatter\_feature\_2}",  
 labels={scatter\_feature\_1: scatter\_feature\_1, scatter\_feature\_2: scatter\_feature\_2})  
else:  
 scatter\_plot\_fig = None  
  
# Bar Graph  
feature\_bar = 'AGE'  
if feature\_bar in data.columns:  
 bar\_fig = px.bar(data[feature\_bar].value\_counts(),  
 x=data[feature\_bar].value\_counts().index,  
 y=data[feature\_bar].value\_counts().values,  
 labels={'x': feature\_bar, 'y': 'Count'},  
 title=f"Bar Graph of {feature\_bar}")  
else:  
 bar\_fig = None  
  
# Histogram  
feature\_hist = 'AGE'  
if feature\_hist in data.columns:  
 hist\_fig = px.histogram(data, x=feature\_hist, nbins=20, title=f"Histogram of {feature\_hist}")  
else:  
 hist\_fig = None  
  
# Dash Application  
app = Dash(\_\_name\_\_)  
  
app.layout = html.Div([  
 html.H1("Interactive Dashboard", style={'textAlign': 'center'}),  
  
 # Violin Plot  
 html.Div([  
 html.H2("Violin Plot of AGE"),  
 dcc.Graph(figure=violin\_fig)  
 ]),  
  
 # Line Graph for Single Feature  
 html.Div([  
 html.H2(f"Line Graph of {feature\_line}"),  
 dcc.Graph(figure=line\_fig) if line\_fig else html.Div("Line graph feature not found.")  
 ]),  
  
 # Pie Chart  
 html.Div([  
 html.H2("Target Variable Distribution"),  
 dcc.Graph(figure=pie\_chart\_fig)  
 ]),  
  
 # Scatter Plot  
 html.Div([  
 html.H2(f"Scatter Plot of {scatter\_feature\_1} vs {scatter\_feature\_2}"),  
 dcc.Graph(figure=scatter\_plot\_fig) if scatter\_plot\_fig else html.Div("Scatter plot features not found.")  
 ]),  
  
 # Bar Graph  
 html.Div([  
 html.H2(f"Bar Graph of {feature\_bar}"),  
 dcc.Graph(figure=bar\_fig) if bar\_fig else html.Div("Bar graph feature not found.")  
 ]),  
  
 # Histogram  
 html.Div([  
 html.H2(f"Histogram of {feature\_hist}"),  
 dcc.Graph(figure=hist\_fig) if hist\_fig else html.Div("Histogram feature not found.")  
 ])  
])  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server(debug=True)

**OUTPUT**

****

****

****