**Detailed Explanation of the Code Block for interactive dashboard for Credit card Fraud detection prediction**

This code creates an **interactive dashboard** using **Dash** and **Plotly** to visualize fraud detection results, including model performance, confusion matrices, and data exploration.

**1. Importing Required Libraries**

import pandas as pd # Data manipulation and reading CSV files

import plotly.express as px # Creating interactive visualizations

import dash\_bootstrap\_components as dbc # Bootstrap components for better styling

from jupyter\_dash import JupyterDash # Dash framework for Jupyter Notebooks

from dash import dcc, html, Input, Output # Core Dash components and callbacks

* pandas: Reads CSV files (credit\_card\_fraud.csv, model\_metrics.csv).
* plotly.express: Creates interactive graphs like bar charts and scatter plots.
* dash\_bootstrap\_components: Adds **Bootstrap styling** for a professional look.
* jupyter\_dash: Runs the Dash app inside Jupyter Notebook.
* dash: Core Dash components:
  + dcc: Dash Core Components (e.g., graphs, dropdowns).
  + html: Defines the dashboard layout using HTML elements.
  + Input and Output: Used for **callbacks** (interactive updates).

**2. Initializing the Dashboard**

app = JupyterDash(\_\_name\_\_, external\_stylesheets=[dbc.themes.BOOTSTRAP])

* Creates a **Dash app** that runs inside **Jupyter Notebook**.
* Uses **Bootstrap CSS** to improve the design.

**3. Loading the Data**

data = pd.read\_csv("credit\_card\_fraud.csv") # Your main dataset

model\_metrics = pd.read\_csv("model\_metrics.csv") # Model metrics CSV

* credit\_card\_fraud.csv: Contains transaction data.
* model\_metrics.csv: Contains evaluation metrics for trained models.

**4. Defining the Dashboard Layout**

app.layout = html.Div([

html.H1("Fraud Detection Dashboard", style={"text-align": "center"}),

* html.Div([]): Wraps all dashboard components inside a container.
* html.H1(): Adds a title.

**4.1 Section 1: Model Performance Metrics**

html.H3("1. Model Performance Metrics"),

dcc.Graph(

id="model-metrics-bar-chart",

style={"height": "500px"}

),

* html.H3(): Adds a section heading.
* dcc.Graph(): Placeholder for the **bar chart** that will display **model performance** (accuracy, precision, recall, etc.).

**4.2 Section 2: Confusion Matrix Visualizations**

html.H3("2. Confusion Matrix Visualization"),

dcc.Dropdown(

id="confusion-matrix-selector",

options=[

{"label": "Logistic Regression", "value": "logistic"},

{"label": "Random Forest", "value": "rf"},

{"label": "XGBoost", "value": "xgb"}

],

value="logistic",

style={"width": "50%"}

),

dcc.Graph(id="confusion-matrix-plot"),

* **Dropdown (dcc.Dropdown)**: Allows users to **select a model** to view its confusion matrix.
* **Graph (dcc.Graph)**: Displays the **selected confusion matrix**.

**4.3 Section 3: Data Exploration**

python

CopyEdit

html.H3("3. Data Exploration"),

dcc.Dropdown(

id="data-exploration-selector",

options=[

{"label": "Histogram: Limit Balance", "value": "hist\_limit\_bal"},

{"label": "Scatter Plot: Limit Balance vs Age", "value": "scatter\_limit\_age"}

],

value="hist\_limit\_bal",

style={"width": "50%"}

),

dcc.Graph(id="data-exploration-plot")

])

* **Dropdown (dcc.Dropdown)**: Allows users to select a **data visualization**:
  + Histogram of LIMIT\_BAL (credit limit).
  + Scatter plot of LIMIT\_BAL vs. AGE (colored by fraud status).
* **Graph (dcc.Graph)**: Displays the selected plot.

**5. Callbacks (Interactivity)**

Dash **callbacks** automatically update graphs when the user interacts with dropdowns.

**5.1 Model Performance Metrics Bar Chart**

python

CopyEdit

@app.callback(

Output("model-metrics-bar-chart", "figure"),

Input("model-metrics-bar-chart", "id")

)

def update\_model\_metrics\_bar\_chart(\_):

fig = px.bar(

model\_metrics,

x="Model",

y=["Accuracy", "Precision", "Recall", "F1-Score", "ROC-AUC"],

barmode="group",

title="Model Performance Metrics",

labels={"value": "Score", "variable": "Metric"}

)

return fig

**How It Works**

* **Input**: Uses the component ID "model-metrics-bar-chart".
* **Output**: Updates the **bar chart** displaying model performance.
* **Plotly Bar Chart (px.bar)**:
  + X-axis: Model names (Model column).
  + Y-axis: Performance metrics (Accuracy, Precision, Recall, etc.).
  + barmode="group": Groups bars by metric.

**5.2 Confusion Matrix Visualization**

python

CopyEdit

@app.callback(

Output("confusion-matrix-plot", "figure"),

Input("confusion-matrix-selector", "value")

)

def update\_confusion\_matrix\_plot(model\_type):

if model\_type == "logistic":

cm = pd.read\_csv("confusion\_matrix\_logistic.csv")

elif model\_type == "rf":

cm = pd.read\_csv("confusion\_matrix\_rf.csv")

elif model\_type == "xgb":

cm = pd.read\_csv("confusion\_matrix\_xgb.csv")

fig = px.imshow(

cm.values,

labels=dict(x="Predicted", y="Actual"),

x=["Not Fraud", "Fraud"],

y=["Not Fraud", "Fraud"],

text\_auto=True,

title=f"Confusion Matrix: {model\_type.capitalize()}"

)

return fig

**How It Works**

* **Input**: Listens to the **confusion matrix dropdown** (confusion-matrix-selector).
* **Reads the correct CSV file** based on the selected model.
* **Uses px.imshow()** to create a **heatmap** visualization.

**5.3 Data Exploration Plots**

python

CopyEdit

@app.callback(

Output("data-exploration-plot", "figure"),

Input("data-exploration-selector", "value")

)

def update\_data\_exploration\_plot(plot\_type):

if plot\_type == "hist\_limit\_bal":

return px.histogram(data, x="LIMIT\_BAL", title="Histogram of Limit Balance")

elif plot\_type == "scatter\_limit\_age":

return px.scatter(

data, x="LIMIT\_BAL", y="AGE", color="default.payment.next.month",

title="Limit Balance vs Age (Fraud Highlighted)"

)

**How It Works**

* **Input**: Listens to the **data exploration dropdown** (data-exploration-selector).
* **Generates different plots** based on selection:
  + **Histogram (px.histogram)**: Shows credit limit distribution.
  + **Scatter Plot (px.scatter)**: Plots age vs. credit limit, coloring fraud cases.

**6. Running the App in Jupyter Notebook**

python

CopyEdit

app.run\_server(mode="inline", debug=True)

* **Runs the dashboard** inside Jupyter Notebook.
* mode="inline": Displays output inside the notebook.
* debug=True: Enables debugging.

**📌 Summary**

| **Section** | **Description** |
| --- | --- |
| **Imports** | Loads Dash, Plotly, Pandas for visualization & interactivity. |
| **App Initialization** | Creates a Dash app with Bootstrap styling. |
| **Data Loading** | Reads credit\_card\_fraud.csv and model\_metrics.csv. |
| **Layout** | Creates sections for Model Performance, Confusion Matrix, and Data Exploration. |
| **Callbacks** | Updates plots dynamically based on user input. |
| **Running the App** | Displays the dashboard inside Jupyter Notebook. |

**📝 Key Takeaways**

1. **Dash + Plotly** → Used to create an **interactive web app**.
2. **Callbacks** → Allow users to dynamically update plots using dropdowns.
3. **Confusion Matrices & Model Metrics** → Visualized interactively.
4. **Data Exploration** → Helps analyze credit card fraud data.

Would you like to add any additional features, such as **real-time updates** or **additional charts**? 🚀

4o

**You said:**

YES

**ChatGPT said:**

ChatGPT

Great! Here are a few **enhancements** we can add to your **Fraud Detection Dashboard**:

**🔹 Suggested Features for Improvement**

1. **Live Data Updates** – Refresh model performance metrics or fraud detection data in real-time.
2. **Feature Importance Visualization** – Show which features are most important for fraud detection.
3. **Threshold Adjustment for Fraud Detection** – Add a slider to adjust fraud classification thresholds dynamically.
4. **Correlation Heatmap** – Show a heatmap of feature correlations in the dataset.
5. **Time-Series Analysis** – Plot fraud cases over time to identify trends.

**📌 Let’s Implement Feature Importance Visualization**

This will help understand **which features contribute the most** to fraud detection.

**🟢 Steps to Implement**

1. Train a model (Random Forest or XGBoost) and extract feature importance.
2. Save feature importance as a CSV file (feature\_importance.csv).
3. Add a **bar chart in the dashboard** to visualize feature importance.

**✅ Step 1: Extract Feature Importance**

Run this **Python script** after training your models to save feature importance:

python

CopyEdit

import pandas as pd

# Assuming rf\_grid.best\_estimator\_ is your best Random Forest model

rf\_best = rf\_grid.best\_estimator\_

# Extract feature importance

feature\_importance = pd.DataFrame({

"Feature": X\_train.columns,

"Importance": rf\_best.feature\_importances\_

}).sort\_values(by="Importance", ascending=False)

# Save feature importance to CSV

feature\_importance.to\_csv("feature\_importance.csv", index=False)

print("Feature importance saved to feature\_importance.csv!")

**✅ Step 2: Add Feature Importance Section to the Dashboard**

Modify the layout to **add a new section for feature importance**:

python

CopyEdit

# Section 4: Feature Importance

html.H3("4. Feature Importance"),

dcc.Graph(id="feature-importance-bar-chart"),

**✅ Step 3: Create Callback to Update Feature Importance Plot**

Now, add the callback function to **read the feature importance CSV** and create a bar chart:

python

CopyEdit

@app.callback(

Output("feature-importance-bar-chart", "figure"),

Input("feature-importance-bar-chart", "id")

)

def update\_feature\_importance(\_):

# Load feature importance data

feature\_importance = pd.read\_csv("feature\_importance.csv")

# Create a bar chart

fig = px.bar(

feature\_importance,

x="Importance",

y="Feature",

orientation="h",

title="Feature Importance",

labels={"Importance": "Score", "Feature": "Feature"},

)

return fig

**🎯 What This Update Does**

✔ **Extracts feature importance** from the Random Forest model.  
✔ **Saves it to a CSV file** for easy access.  
✔ **Displays a horizontal bar chart** of important features in the dashboard.

Would you like to implement another **enhancement** (e.g., time-series analysis or threshold tuning)?