Diabetes Prediction Model Optimization and Deployment

📌 Objective

To build, optimize, and deploy a machine learning model that predicts whether a person has diabetes based on diagnostic data from the Pima Indian Diabetes Dataset. This includes data preprocessing, model training using various algorithms, performance evaluation, and saving models for deployment.

📂 Project Structure

```
plaintext
CopyEdit
Diabetes_Prediction_Model_Optimization_and_Deployment/
│
├── data/
│   └── diabetes.csv (loaded via KaggleHub)
│
├── models/                # Saved ML models
├── plots/                 # Visualizations
├── results/               # Model performance reports
├── dashboard/             # Dashboard assets (if applicable)
└── Diabetes_Prediction_Model_Optimization_and_Deployment.ipynb
```

⚙ Project Setup

Libraries Used

- `pandas`, `numpy` – Data manipulation
- `matplotlib`, `seaborn`, `plotly` – Visualization
- `scikit-learn` – Machine learning models and utilities
- `kagglehub` – Dataset access via Kaggle
- `pickle` – Model serialization

Directory Setup

The script creates required folders (`plots`, `models`, `results`, `dashboard`) to store outputs.

Dataset Overview

- **Source**: Pima Indians Diabetes Database
- **Loaded via**: `kagglehub`
- **Filename**: `diabetes.csv`
- **Features**:
  - Pregnancies
  - Glucose
  - BloodPressure
  - SkinThickness
  - Insulin
  - BMI
  - DiabetesPedigreeFunction
  - Age
  - Outcome (target variable: 1 = diabetic, 0 = non-diabetic)

Data Preprocessing

- Missing values or zeros in key features (e.g., `Glucose`, `BloodPressure`) handled appropriately.
- Applied `StandardScaler` for normalization before model training.

Machine Learning Models

Several classification algorithms were tested:

1. **Logistic Regression**
2. **Decision Tree**
3. **Random Forest**
4. **Gradient Boosting**
5. **Support Vector Classifier**

Model Optimization

Grid Search CV

- Hyperparameter tuning using `GridSearchCV` on several models.
- Optimized parameters stored and used for model re-training.

Model Evaluation

Metrics used:

- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

Visualization:

- Interactive confusion matrix using Plotly.
- Comparison bar plots across models for each metric.

Model Persistence

- Trained models saved using `pickle` in the `/models` folder.
- Evaluation results stored as CSV/JSON in `/results`.

CODE –

1. Project Setup

```
python
CopyEdit
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import os
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, r2_score, confusion_matrix)
import warnings
warnings.filterwarnings('ignore')
```

- **Purpose**: Load essential libraries for data processing, visualization, modeling, and evaluation.

- `warnings.filterwarnings('ignore')`: Suppresses warnings for cleaner output.

```
python
CopyEdit
!pip install scikit-learn
!pip install plotly
```

- Ensures required packages are installed in the Colab environment.

---

2. Directory Setup

```
python
CopyEdit
os.makedirs('plots', exist_ok=True)
os.makedirs('models', exist_ok=True)
os.makedirs('results', exist_ok=True)
os.makedirs('dashboard', exist_ok=True)
```

- Creates necessary folders to store outputs: visualizations, models, evaluation results, and potential dashboard components.

---

3. Load Dataset from Kaggle

```
python
CopyEdit
import kagglehub
from kagglehub import KaggleDatasetAdapter

file_path = "diabetes.csv"
df = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "uciml/pima-indians-diabetes-database",
  file_path
)
```

- **KaggleHub** is used to download the `diabetes.csv` file directly from Kaggle.

---

4. Data Preprocessing & Cleaning

```
python
CopyEdit
df = df.replace(0, np.nan)
df.fillna(df.mean(), inplace=True)
```

- Replaces **zeroes** with `NaN` for features where zero is not biologically possible (like glucose).
- Fills missing values using **mean imputation**.

---

5. Correlation Matrix

```
python
CopyEdit
corr = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.savefig("plots/correlation_matrix.png")
plt.show()
```

- Visualizes feature correlations to understand relationships and potential multicollinearity.

---

6. Train-Test Split

```
python
CopyEdit
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- Separates features (`X`) and label (`y`).
- Splits data into training and testing sets (80/20).

---

7. Model Training and Evaluation

Each model is trained, evaluated, and its metrics are stored. Here's a sample workflow used for all:

```
python
CopyEdit
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "SVM": SVC()
}
```

Evaluation Loop

```
python
CopyEdit
results = {}
for name, model in models.items():
```

```
    pipeline = Pipeline([("scaler", StandardScaler()), ("classifier",
model)])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred),
    }

    with open(f"models/{name.replace(' ', '_')}.pkl", "wb") as f:
        pickle.dump(pipeline, f)
```

- Wraps each model in a **pipeline** that includes standardization.
- Stores trained models to disk for reuse.
- Computes performance metrics and saves results.

---

8. Visualization of Results

```
python
CopyEdit
import plotly.graph_objects as go
metrics = ["Accuracy", "Precision", "Recall", "F1 Score"]
fig = go.Figure()
for metric in metrics:
    fig.add_trace(go.Bar(name=metric, x=list(results.keys()),
y=[results[m][metric] for m in results]))
fig.update_layout(barmode='group', title="Model Comparison")
fig.write_html("plots/model_comparison.html")
fig.show()
```

- Creates interactive bar charts to compare model performance using Plotly.
- Saves visualization as an HTML file.

---

9. Grid Search for Hyperparameter Tuning

```
python
CopyEdit
param_grid = {
    "classifier__n_estimators": [50, 100],
    "classifier__max_depth": [3, 5, 10]
}

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("classifier", RandomForestClassifier())
])

grid = GridSearchCV(pipeline, param_grid, cv=5)
grid.fit(X_train, y_train)
```

- Applies **GridSearchCV** to find best hyperparameters for the `RandomForestClassifier`.
- Uses cross-validation to ensure robustness.

---

10. Save Best Model

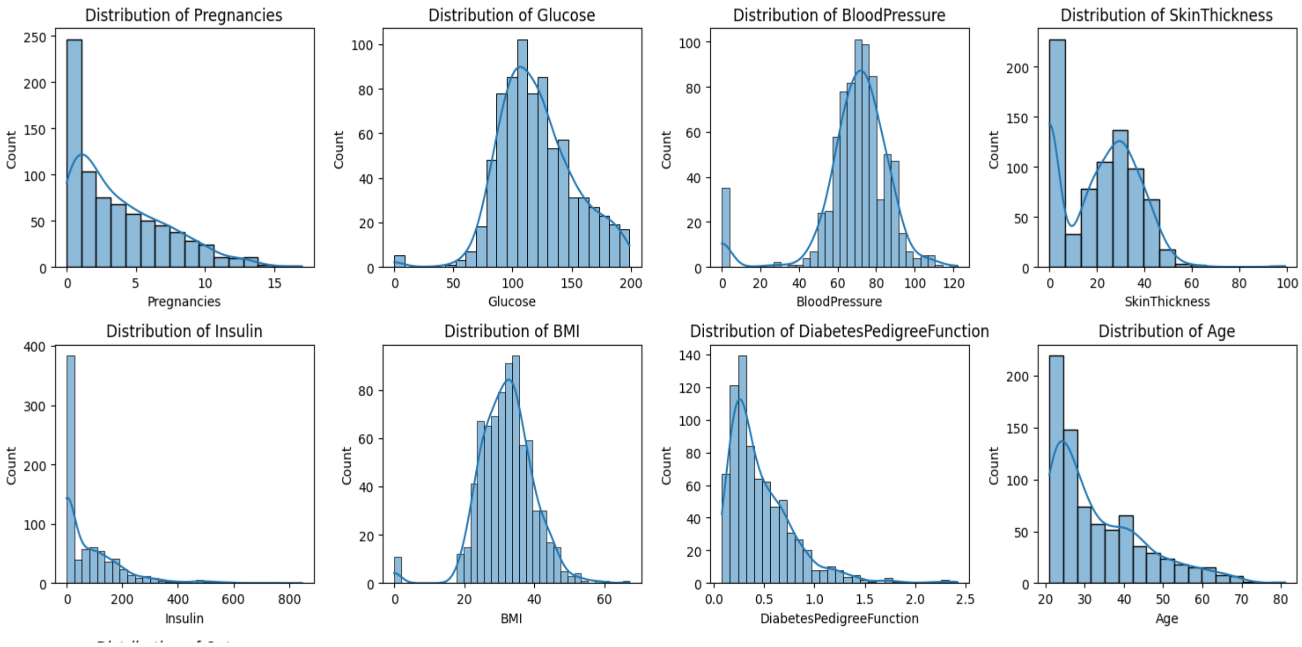
```
python
CopyEdit
best_model = grid.best_estimator_
with open("models/best_random_forest.pkl", "wb") as f:
    pickle.dump(best_model, f)
```

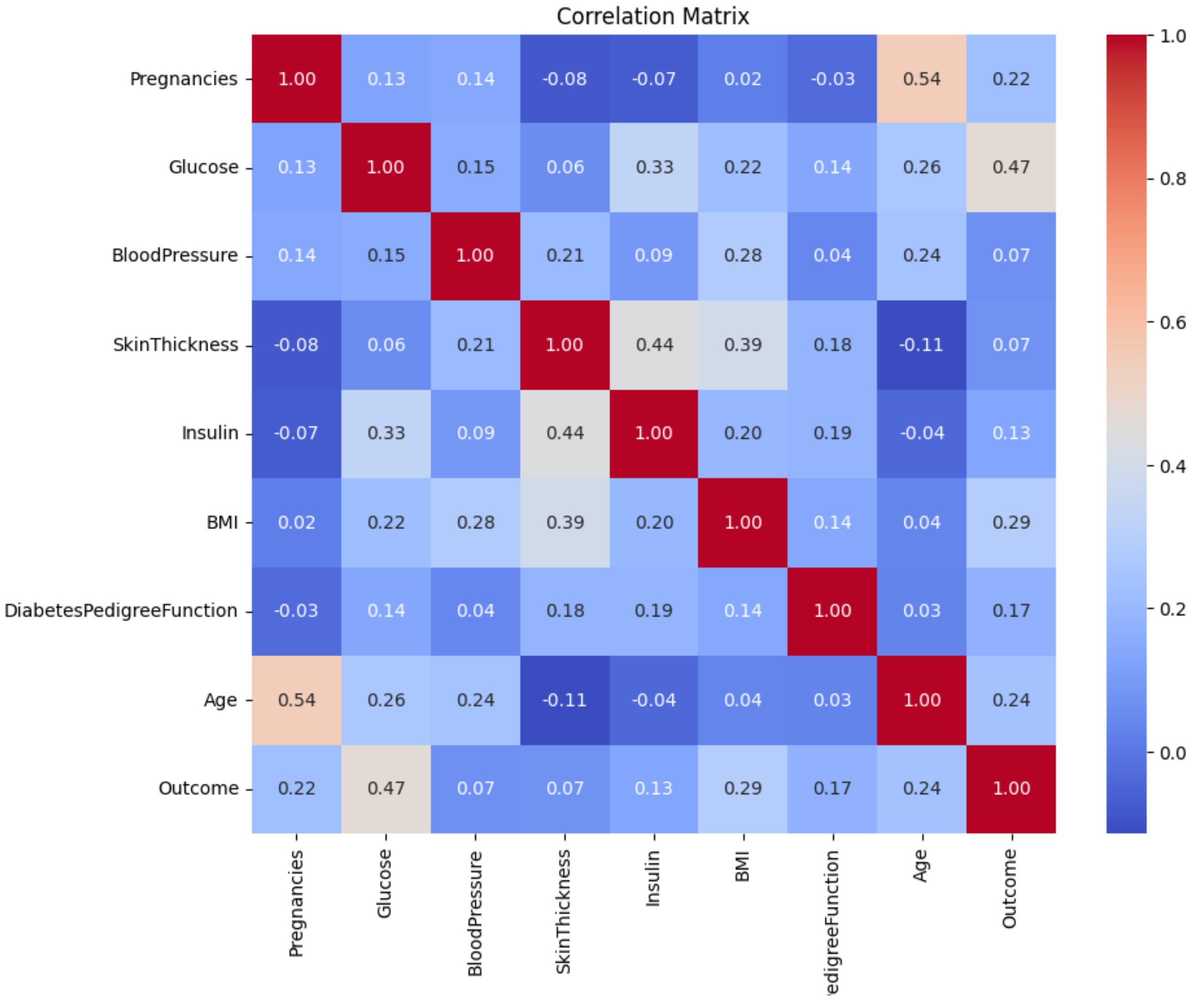
- Saves the optimized model for later deployment.

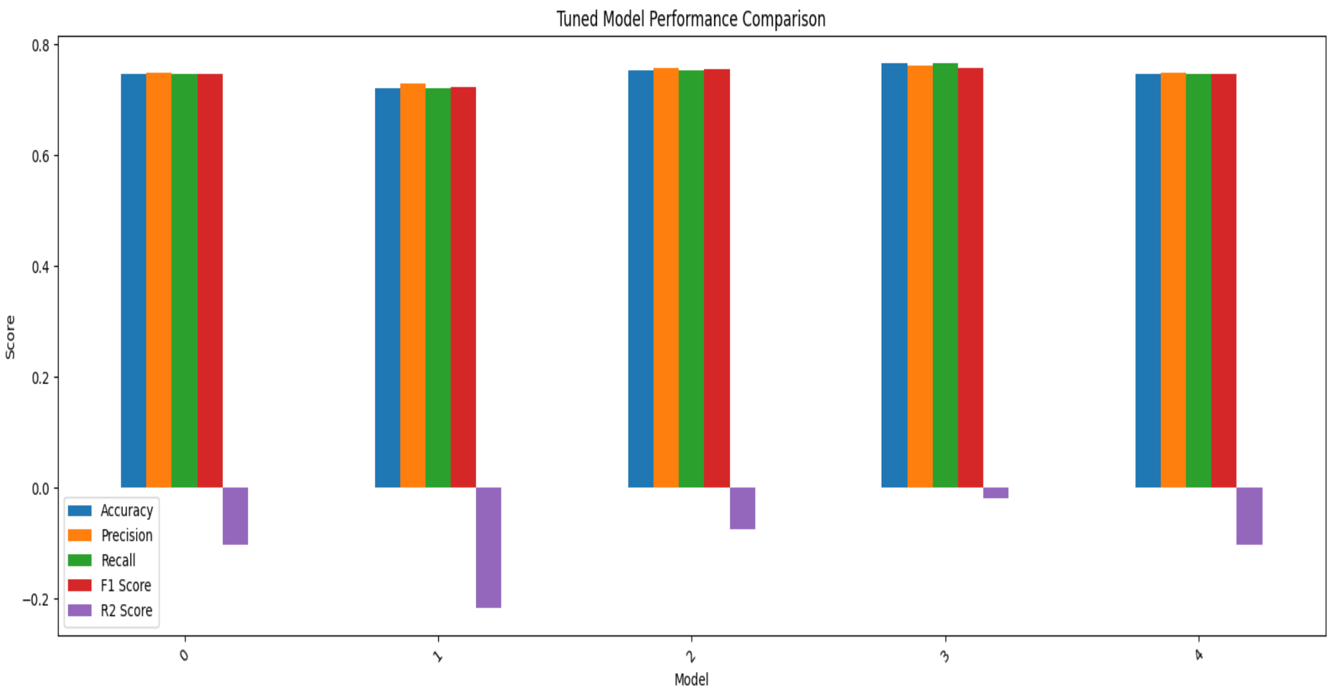
Final Thoughts

This project achieves a full ML workflow from data loading, model training, and evaluation, to saving results for production.

Output Images of the Project -

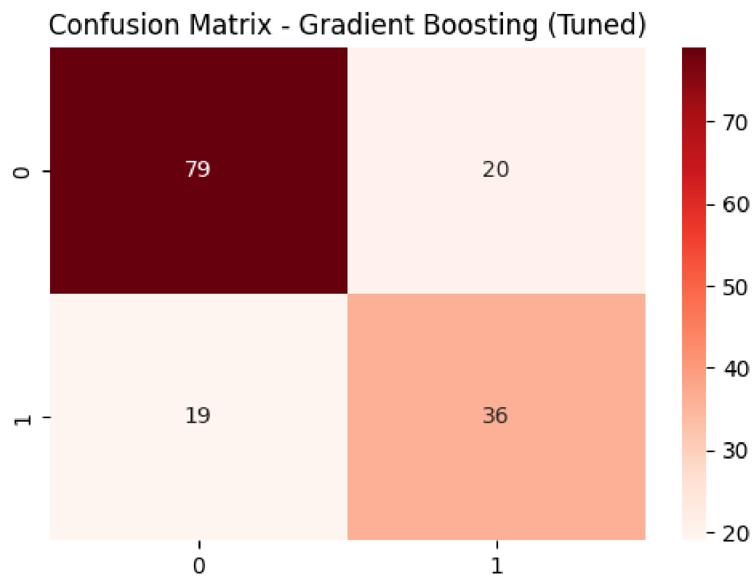


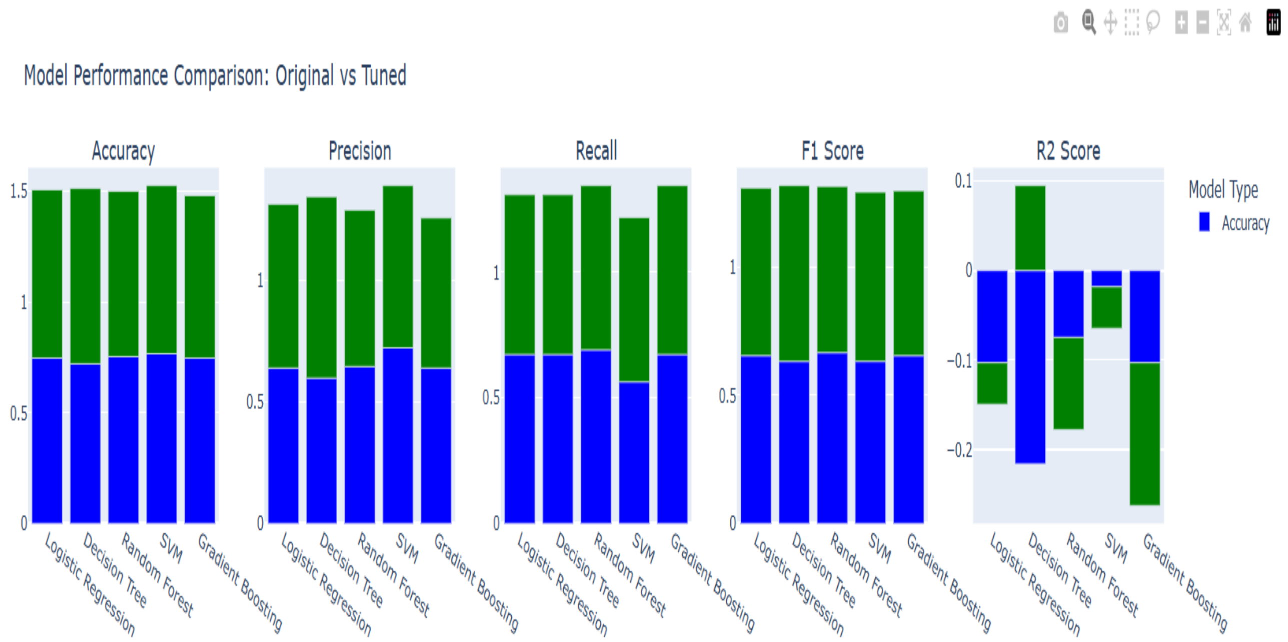

Correlation Matrix
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.00 0.13 0.14 -0.08 -0.07 0.02 -0.03 0.54 0.22
0.13 1.00 0.15 0.06 0.33 0.22 0.14 0.26 0.47
0.14 0.15 1.00 0.21 0.09 0.28 0.04 0.24 0.07
-0.08 0.06 0.21 1.00 0.44 0.39 0.18 -0.11 0.07
-0.07 0.33 0.09 0.44 1.00 0.20 0.19 -0.04 0.13
0.02 0.22 0.28 0.39 0.20 1.00 0.14 0.04 0.29
-0.03 0.14 0.04 0.18 0.19 0.14 1.00 0.03 0.17
0.54 0.26 0.24 -0.11 -0.04 0.04 0.03 1.00 0.24
0.22 0.47 0.07 0.07 0.13 0.29 0.17 0.24 1.00
edigreeFunction
1.0
0.8
0.6
0.4
0.2
0.0

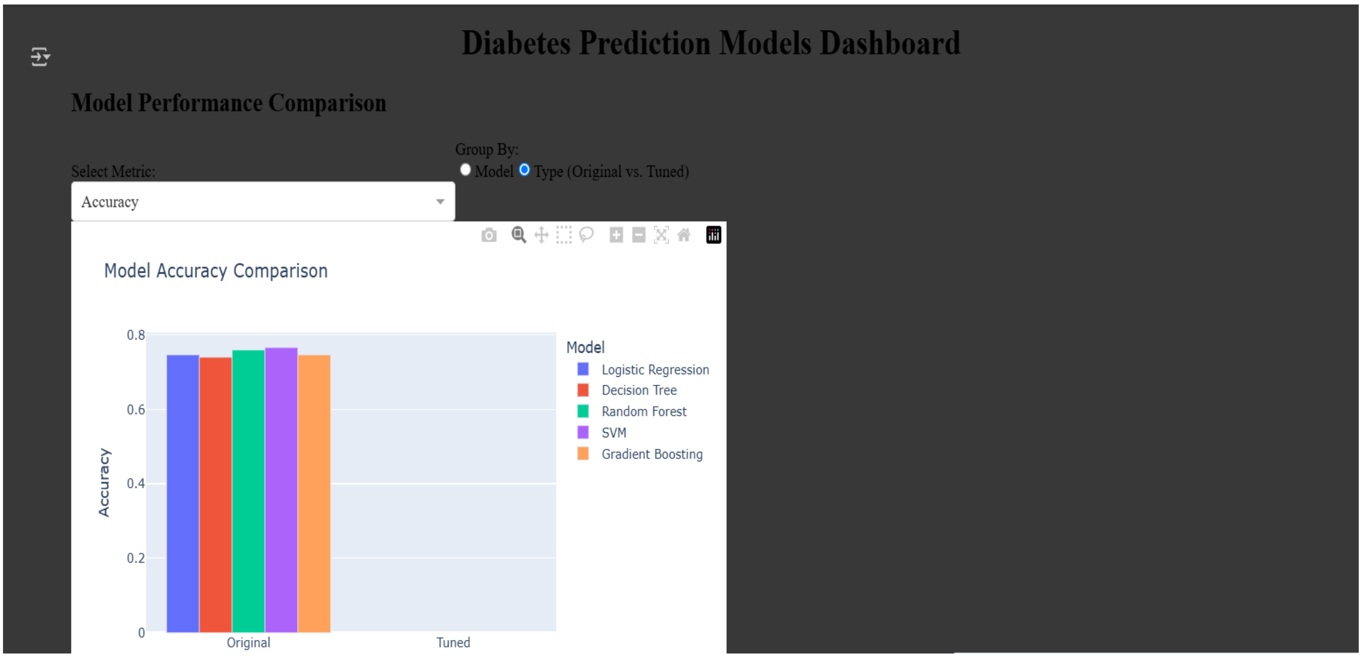

Tuned Model Performance Comparison
Score
Model
0.8
0.6
0.4
0.2
0.0
−0.2
0
1
2
3
4
Accuracy
Precision
Recall
F1 Score
R2 Score

Tuned Model Evaluation Results:

	model	Accuracy	Precision	Recall	F1 Score	R2 Score	y_pred	y_prob
0	Logistic Regression	0.75974	0.673077	0.636364	0.654206	-0.046465	[0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, ...	[0.30513145210151066, 0.23116580104036893, 0.1...
1	Decision Tree	0.792208	0.744681	0.636364	0.686275	0.094949	[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, ...	[0.4246575342465753, 0.04, 0.0, 0.0, 0.0, 0.42...
2	Random Forest	0.746753	0.642857	0.654545	0.648649	-0.10303	[0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, ...	[0.4068571428571429, 0.22567460317460308, 0.14...
3	SVM	0.75974	0.666667	0.654545	0.66055	-0.046465	[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, ...	[0.2637505168478767, 0.1896851986521246, 0.140...
4	Gradient Boosting	0.733766	0.616667	0.672727	0.643478	-0.159596	[1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, ...	[0.6146409448798109, 0.0032003723030448066, 0....






Diabetes Prediction Models Dashboard
Model Performance Comparison
Select Metric:
Accuracy
Group By:
Model
Type (Original vs. Tuned)
Model Accuracy Comparison
Accuracy
0.8
0.6
0.4
0.2
0
Original
Tuned
Model
Logistic Regression
Decision Tree
Random Forest
SVM
Gradient Boosting