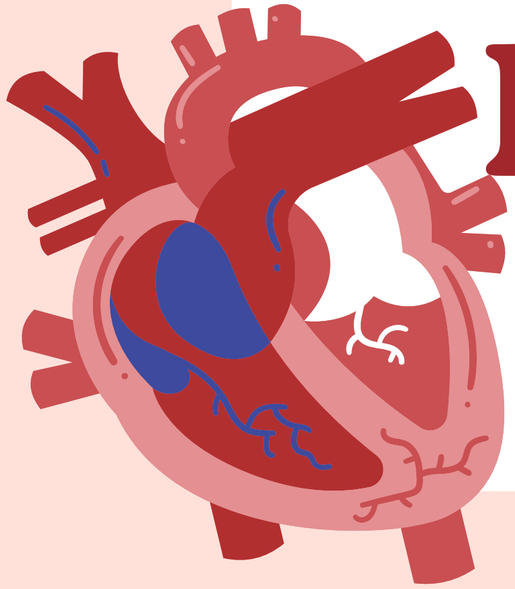
HEART DISEASE PREDICTION



introduction

Heart disease is a leading cause of death worldwide. This project uses machine learning to predict the risk of heart disease based on medical data such as age, blood pressure, and cholesterol levels. The goal is to support early diagnosis and improve treatment outcomes.

Do I have a heart disease?

YES

NO

Milestone 1:

1. Data Collection

The dataset used in this project is the Heart Disease Risk Prediction Dataset, a synthetic dataset developed by students at Vellore Institute of Technology (VIT-AP) as part of the EarlyMed initiative. It contains 70,000 samples, each representing a patient, and is specifically designed for binary classification tasks (no risk = 0, risk = 1).

Milestone 1:

1. Data Collection

2. Dataset Features (Column Descriptions)

Column Name	Description	Data Type
Heart_Risk	Heart disease risk (target variable).	Binary (0 or 1)
Cold_Sweats_Nausea	Presence of cold sweats or nausea.	Binary (0 or 1)
Pain_Arms_Jaw_Back	Pain in arms, jaw, or back.	Binary (0 or 1)
Swelling	Swelling in the body.	Binary (0 or 1)
Dizziness	Feeling of dizziness.	Binary (0 or 1)
Fatigue	Chronic fatigue.	Binary (0 or 1)
Shortness_of_Breath	Difficulty in breathing.	Binary (0 or 1)
Chest_Pain	Presence of chest pain.	Binary (0 or 1)
Palpitations	Experience of heart palpitations.	Binary (0 or 1)

Milestone 1:

1. Data Collection

Age	Age of the individual.	Integer (Numerical)
Sedentary_Lifestyle	Leading a sedentary (inactive) lifestyle.	Binary (0 or 1)
Obesity	Whether the person is obese.	Binary (0 or 1)
Chronic_Stress	Experience of long-term stress.	Binary (0 or 1)
Diabetes	Presence of diabetes.	Binary (0 or 1)
Family_History	Family history of heart disease.	Binary (0 or 1)
Smoking	Whether the person is a smoker.	Binary (0 or 1)
High_BP	High blood pressure (hypertension).	Binary (0 or 1)
High_Cholesterol	High cholesterol levels.	Binary (0 or 1)
Gender	Gender (1 for male, 0 for female).	Binary (0 or 1)

Milestone 1:

2. Data Exploration

handling duplicate rows

```
df.duplicated().sum()
```

✓ 0.1s

```
np.int64(5843)
```

handling missing values

```
df.isnull().sum()
```

[11] ✓ 0.0s

```
Chest_Pain                  0
Shortness_of_Breath         0
Fatigue                     0
Palpitations                0
Dizziness                   0
Swelling                    0
Pain_Arms_Jaw_Back          0
Cold_Sweats_Nausea          0
High_BP                  9942
High_Cholesterol        10200
Diabetes                    0
Smoking                     0
Obesity                     0
Sedentary_Lifestyle         0
Family_History              0
Chronic_Stress              0
Gender                      0
Age                      8110
Heart_Risk                  0
dtype: int64
```

Milestone 1:

2. Data Exploration

```
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFram
Index: 66257 entries, 0 to 72099
Data columns (total 19 columns):
```

```
df.describe()
```

✓ 0.0s

	# Chest_Pain	# Shortness_of_Breath	# Fatigue
count	63815.0	63815.0	63815.0
mean	0.4965760401159602	0.4981430698111729	0.4966387213037687
std	0.49999219388969207	0.500000469391647	0.49999261920962995
min	0.0	0.0	0.0
25%	0.0	0.0	0.0
50%	0.0	0.0	0.0
75%	1.0	1.0	1.0
max	1.0	1.0	1.0

Milestone 1:

2. Data Exploration

```
df['High_BP'].value_counts()
```

✓ 0.0s

```
High_BP
0.0    27124
1.0    26749
2.0     2442
Name: count, dtype: int64
```

```
df['High_Cholesterol'].value_counts
```

✓ 0.0s

```
High_Cholesterol
0.0    27199
1.0    26804
Name: count, dtype: int64
```

```
print(df['Smoking'].value_counts())
```

✓ 0.0s

```
Smoking
0     27708
1     27615
3      1801
99     1780
2      1694
Name: count, dtype: int64
```

Milestone 1:

3. Data Preprocessing

handling duplicate rows

```
df.duplicated().sum()
```
[7] ✓ 0.1s

```
np.int64(5843)
```

```
df.drop_duplicates(inplace=True)
```
[8] ✓ 0.0s

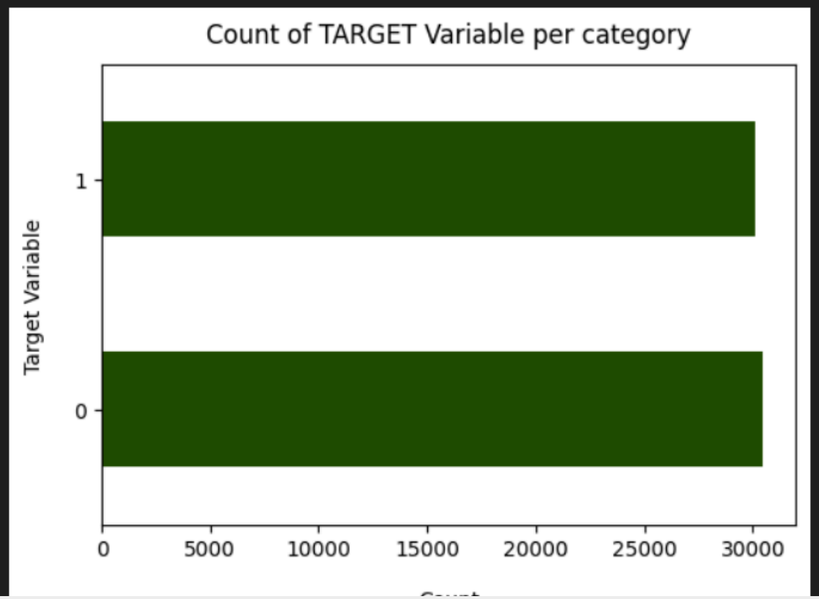
```
df.duplicated().sum()
```
[9] ✓ 0.0s

```
np.int64(0)
```

Checking Balanced or Imbalanced Dataset

```
df['Heart_Risk'].value_counts().plot(kind='barh', color="#204D00", figsize=(6, 4))
plt.xlabel("Count", labelpad=14)
plt.ylabel("Target Variable", labelpad=14)
plt.title("Count of TARGET Variable per category", y=1.02)
```
4] ✓ 0.2s

```
Text(0.5, 1.02, 'Count of TARGET Variable per category')
```



Milestone 1:

3. Data Preprocessing

```
df.info()
✓ 0.0s
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 66257 entries, 0 to 72099
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Chest_Pain            66257 non-null  int64
 1   Shortness_of_Breath   66257 non-null  int64
 2   Fatigue               66257 non-null  int64
 3   Palpitations          66257 non-null  int64
 4   Dizziness             66257 non-null  int64
 5   Swelling              66257 non-null  int64
 6   Pain_Arms_Jaw_Back    66257 non-null  int64
 7   Cold_Sweats_Nausea    66257 non-null  int64
 8   High_BP               56315 non-null  float64
 9   High_Cholesterol      56057 non-null  float64
 10  Diabetes              66257 non-null  int64
 11  Smoking               66257 non-null  int64
 12  Obesity               66257 non-null  int64
 13  Sedentary_Lifestyle   66257 non-null  int64
 14  Family_History        66257 non-null  int64
 15  Chronic_Stress        66257 non-null  int64
 16  Gender                66257 non-null  int64
 17  Age                   58147 non-null  object
 18  Heart_Risk            66257 non-null  int64
dtypes: float64(2), int64(16), object(1)
memory usage: 10.1+ MB
```

handling data type

```
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')

✓ 0.0s
```

Milestone 1:

3. Data Preprocessing

```
# Checking which feature/column should be converted into Bins
for column in df.columns:
    print(f'{column} =>', df[column].value_counts().shape)
```

✓ 0.0s

```
Chest_Pain => (2,)
Shortness_of_Breath => (2,)
Fatigue => (2,)
Palpitations => (2,)
Dizziness => (2,)
Swelling => (2,)
Pain_Arms_Jaw_Back => (2,)
Cold_Sweats_Nausea => (2,)
High_BP => (2,)
High_Cholesterol => (2,)
Diabetes => (2,)
Smoking => (2,)
Obesity => (2,)
Sedentary_Lifestyle => (2,)
Family_History => (2,)
Chronic_Stress => (2,)
Gender => (2,)
Age => (65,)
Heart_Risk => (2,)
```

```
bins = [18, 25, 35, 45, 55, float('inf')]
labels = ['18-25', '26-35', '36-45', '46-55', '56+']

df['Age_Binned'] = pd.cut(df['Age'], bins=bins, labels=labels, right=True)

print(df['Age_Binned'].value_counts())
```

✓ 0.0s

```
Age_Binned
56+      35294
46-55    11454
36-45     5690
26-35     5128
18-25     3032
Name: count, dtype: int64
```

Milestone 1:

3. Data Preprocessing

```
most_frequent_value = df['Smoking'].mode()[0]
df['Smoking'] = df['Smoking'].apply(lambda x: most_frequent_value if x not in [0, 1] else x)
```

✓ 0.0s

```
df = df[df['High_BP'] != 2.0]
```

✓ 0.0s

```
df = df[(df['Age'] >= 18) & (df['Age'] <= 120)]
print(df['Age'].describe())
```

✓ 0.0s

```
count    60598.000000
mean        54.638734
std         15.004951
min         20.000000
25%         47.000000
50%         56.000000
75%         64.000000
max         84.000000
Name: Age, dtype: float64
```

milestone 2:

Data Analysis

Is there a relationship between Age and heart risk?

```
grouped_stats = df.groupby('Heart_Risk')['Age'].agg(['count', 'mean', 'std', 'min', 'max']).reset_index()
print(grouped_stats)
```

✓ 0.0s Open 'grouped_stats' in Data Wrangler

	Heart_Risk	count	mean	std	min	max
0	0	30452	46.387823	13.857814	20.0	69.0
1	1	30146	62.973396	10.970959	45.0	84.0

```
risk_0 = df[df['Heart_Risk'] == 0]['Age']
risk_1 = df[df['Heart_Risk'] == 1]['Age']
t_stat, p_val = ttest_ind(risk_0, risk_1)
print(f"\nT-test: t-statistic = {t_stat:.2f}, p-value = {p_val:.4f}")
```

✓ 0.0s

```
T-test: t-statistic = -163.24, p-value = 0.0000
```

People with a higher risk of heart disease tend to age higher, and the difference in age between the two groups is statistically significant, meaning it's not just a coincidence.

milestone 2:

Data Analysis

Are people suffering from Chronic Stress at greater risk?

```
count_table = pd.crosstab(df['Chronic_Stress'], df['Heart_Risk'], margins=True)
count_table.columns = ['Heart_Risk = 0', 'Heart_Risk = 1', 'Total']
count_table.index = ['No Stress', 'Stress', 'Total']
count_table
```

✓ 0.0s Open 'count_table' in Data Wrangler

	# Heart_Risk = 0	# Heart_Risk = 1	# Total
No Stress	21093	9260	30353
Stress	9359	20886	30245
Total	30452	30146	60598

milestone 2:

Data Analysis

Does a sedentary lifestyle clearly show with obesity and heart risks?

```
table = pd.crosstab([df['Sedentary_Lifestyle'], df['Obesity']], df['Heart_Risk'], margins=True)
table.columns = ['No Heart Risk', 'Heart Risk', 'Total']
table.index = ['Active & No Obesity', 'Active & Obesity', 'Sedentary & No Obesity', 'Sedentary & Obesity', 'Total']
table
```

Open 'table' in Data Wrangler

	# No Heart Risk	# Heart Risk	# Total
Missing:	0 (0%)	0 (0%)	0 (0%)
Distinct:	5 (100%)	5 (100%)	5 (100%)
	Min 2861 / Max 30452	Min 2754 / Max 30146	Min 12817 / Max 60598
Active & No Obesity	14552	2754	17306
Active & Obesity	6464	6353	12817
Sedentary & No Obesity	6575	6533	13108
Sedentary & Obesity	2861	14506	17367
Total	30452	30146	60598

milestone 2:

Data Analysis

Are those who suffer from chronic stress and obesity together at greater risk than those who suffer from only one of them?

```
df['Stress_Obesity_Group'] = df['Chronic_Stress'].astype(str) + df['Obesity'].astype(str)
df['Stress_Obesity_Group'] = df['Stress_Obesity_Group'].map({
    '00': 'None',
    '10': 'Chronic Stress Only',
    '01': 'Obesity Only',
    '11': 'Both'
})
grouped = df.groupby('Stress_Obesity_Group')['Heart_Risk'].agg(['count', 'sum', 'mean'])
grouped.columns = ['Total', 'Heart Cases', 'Heart Risk Rate']
grouped.sort_values('Heart Risk Rate', ascending=False)
```

[190] ✓ 0.0s

Stress_Obesity_Group	# Total	# Heart Cases	# Heart Risk Rate
Both	17343	14475	0.8346306867323993
Obesity Only	12841	6384	0.49715754224748854
Chronic Stress Only	12902	6411	0.49689970547201984
None	17512	2876	0.16423024211968937

milestone 2:

Data Analysis

What are the most common characteristics among those affected?

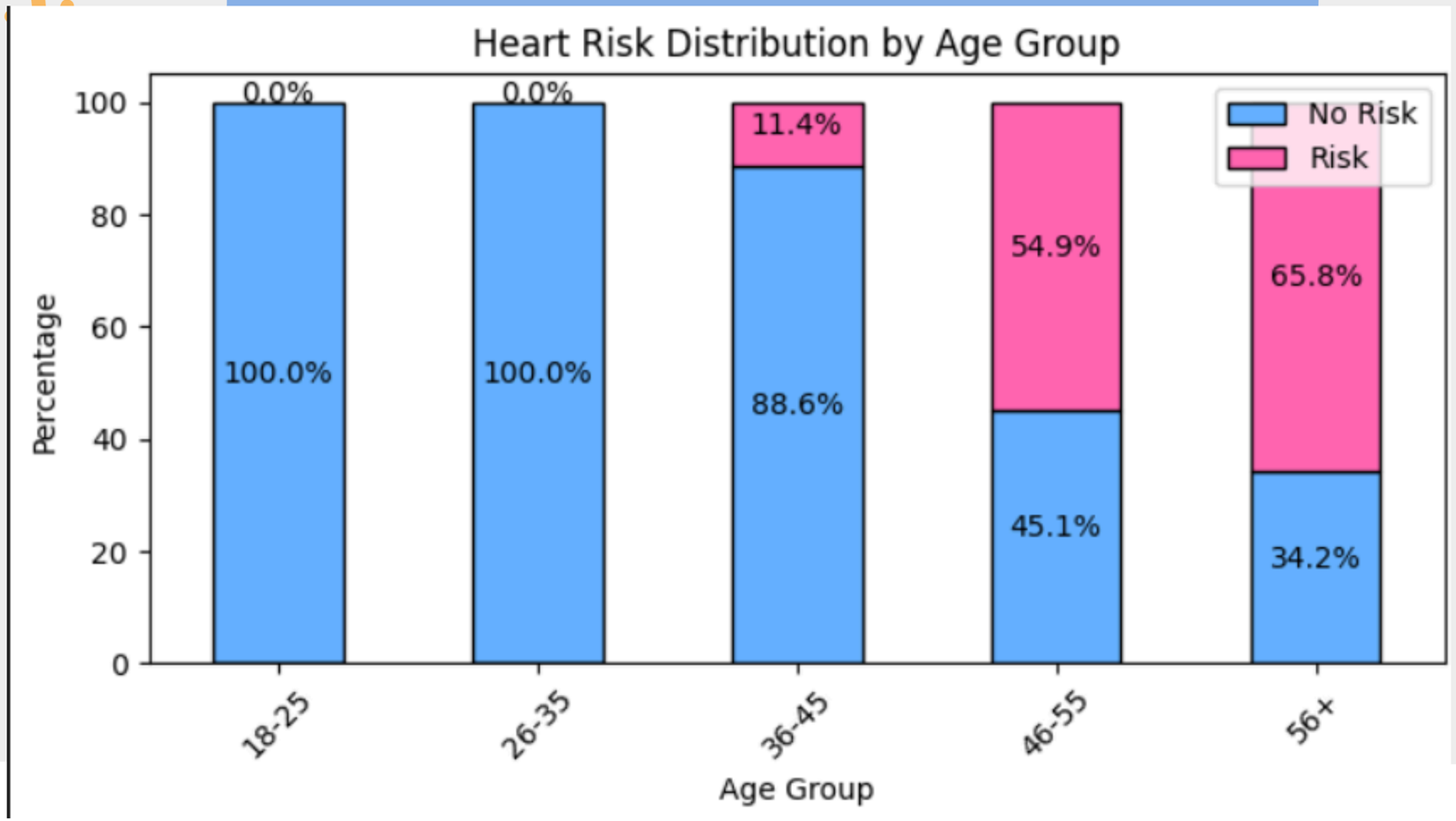
```
affected = df[df['Heart_Risk'] == 1]

columns = [
    'Chest_Pain', 'Shortness_of_Breath', 'Fatigue', 'Palpitations', 'Dizziness',
    'Swelling', 'Pain_Arms_Jaw_Back', 'Cold_Sweats_Nausea', 'High_BP', 'High_Cholesterol',
    'Diabetes', 'Smoking', 'Obesity', 'Sedentary_Lifestyle', 'Family_History', 'Chronic_Stress'
]

common_symptoms = affected[columns].mean().sort_values(ascending=False) * 100
print(common_symptoms)
```

✓ 0.0s Open 'common_symptoms' in Data Wrangler

```
Cold_Sweats_Nausea      79.463279
Pain_Arms_Jaw_Back      79.363763
Dizziness               79.333908
Shortness_of_Breath     79.105022
Swelling                79.028727
Chest_Pain              78.959066
Fatigue                 78.955749
Palpitations            78.776620
Sedentary_Lifestyle     69.790354
Diabetes                69.335899
Chronic_Stress          69.282824
Obesity                 69.193259
Family_History          68.854906
Smoking                 63.318517
High_Cholesterol        58.465468
High_BP                 58.389173
```

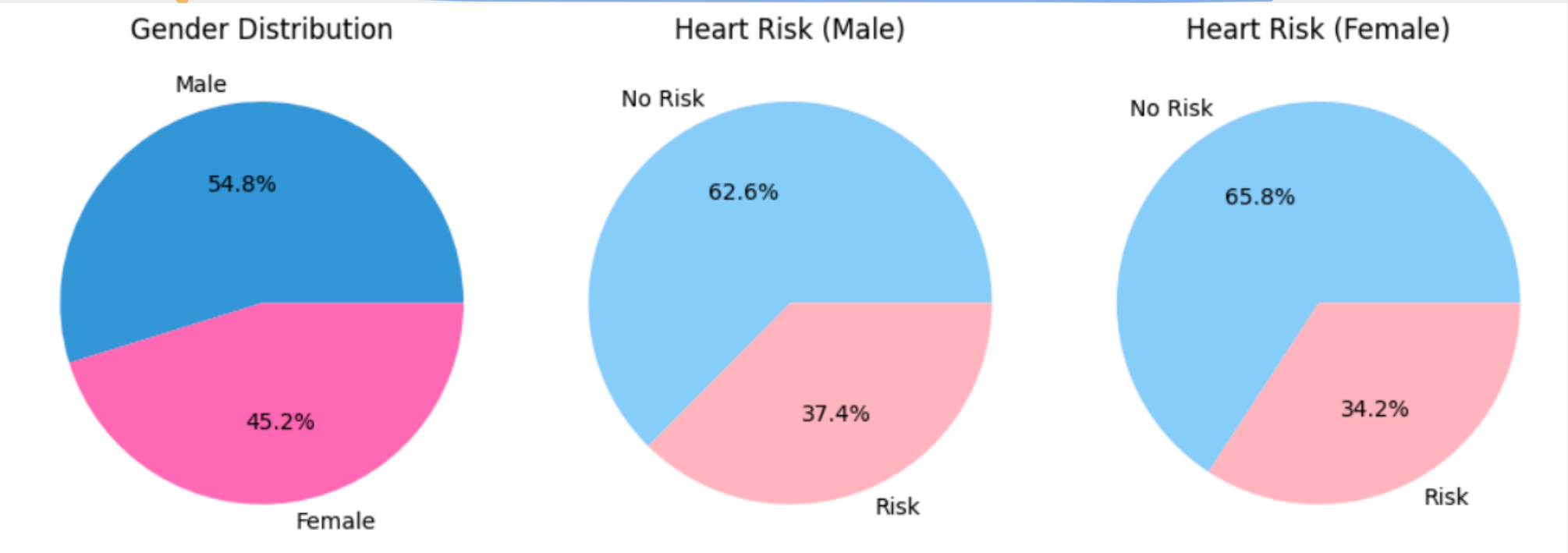
milestone 2:

Data Visualization:



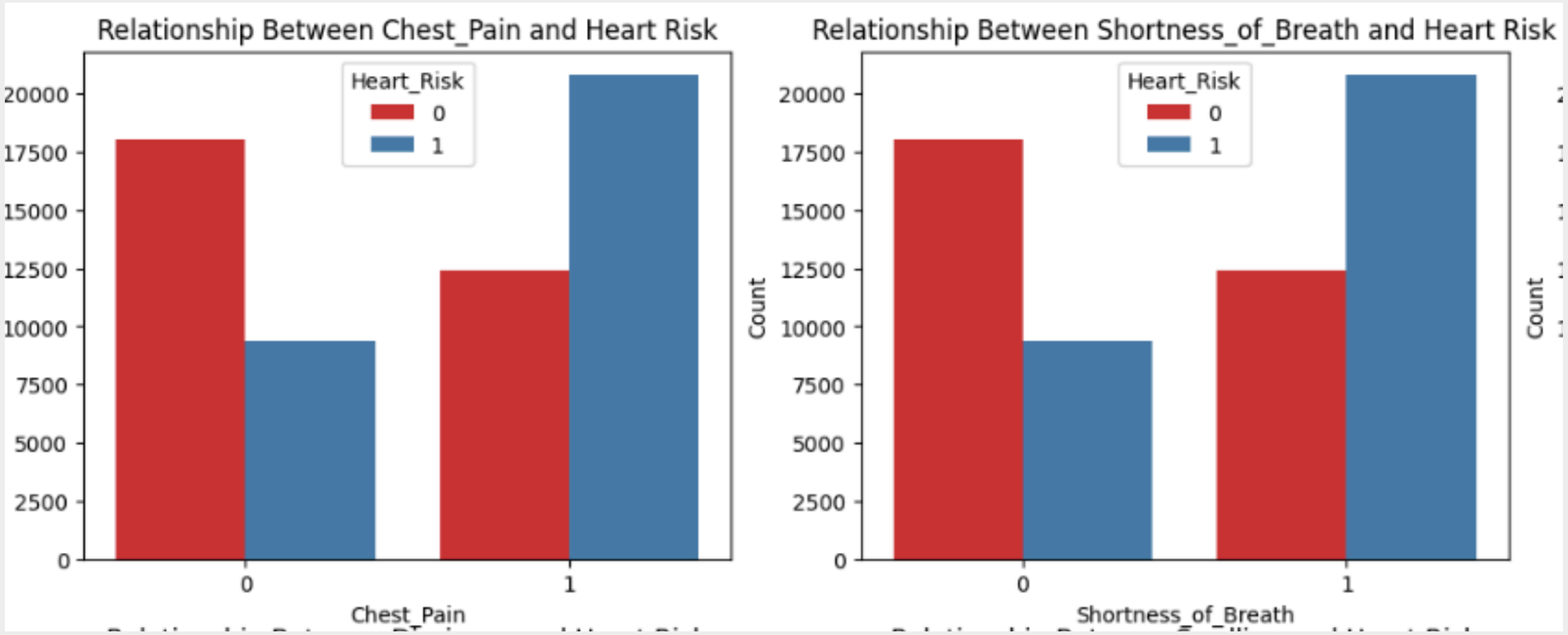
milestone 2:

Data Visualization:



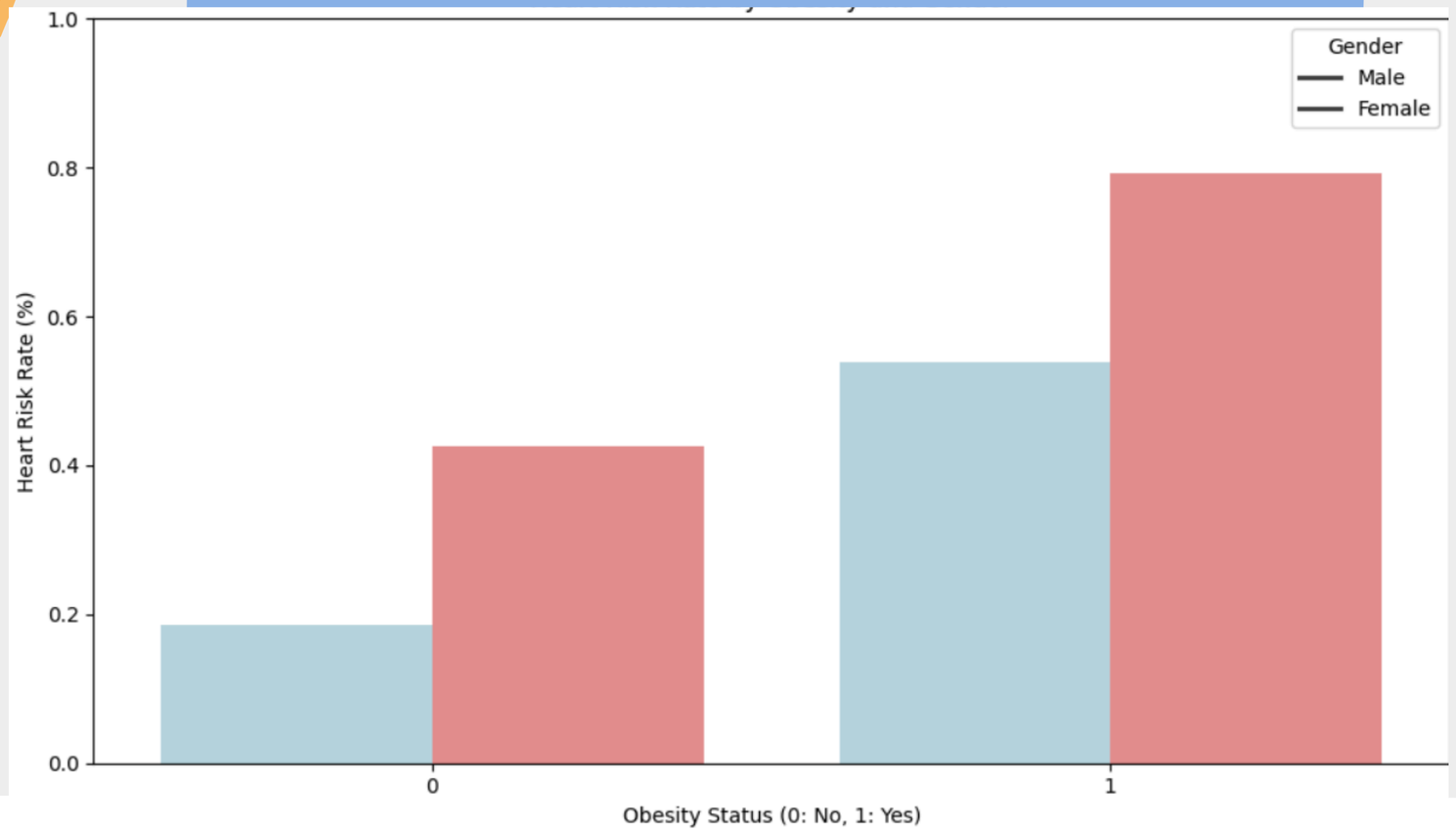
milestone 2:

Data Visualization:



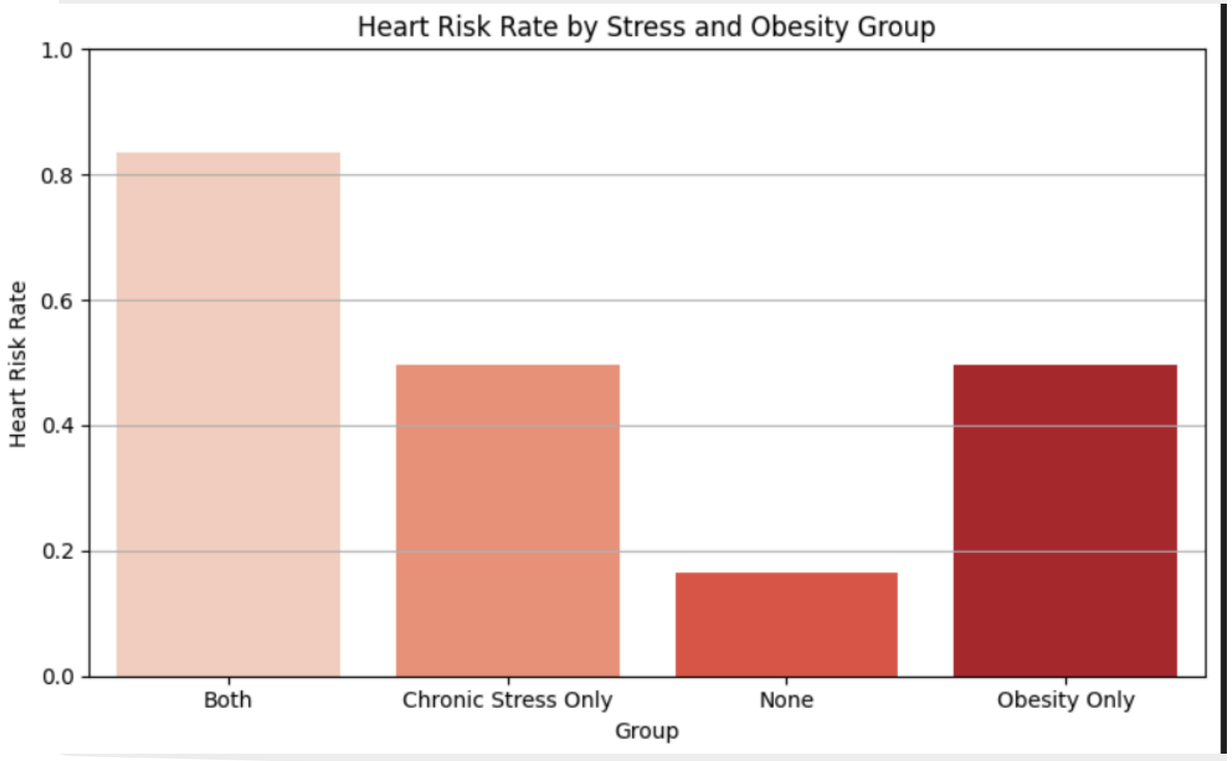
milestone 2:

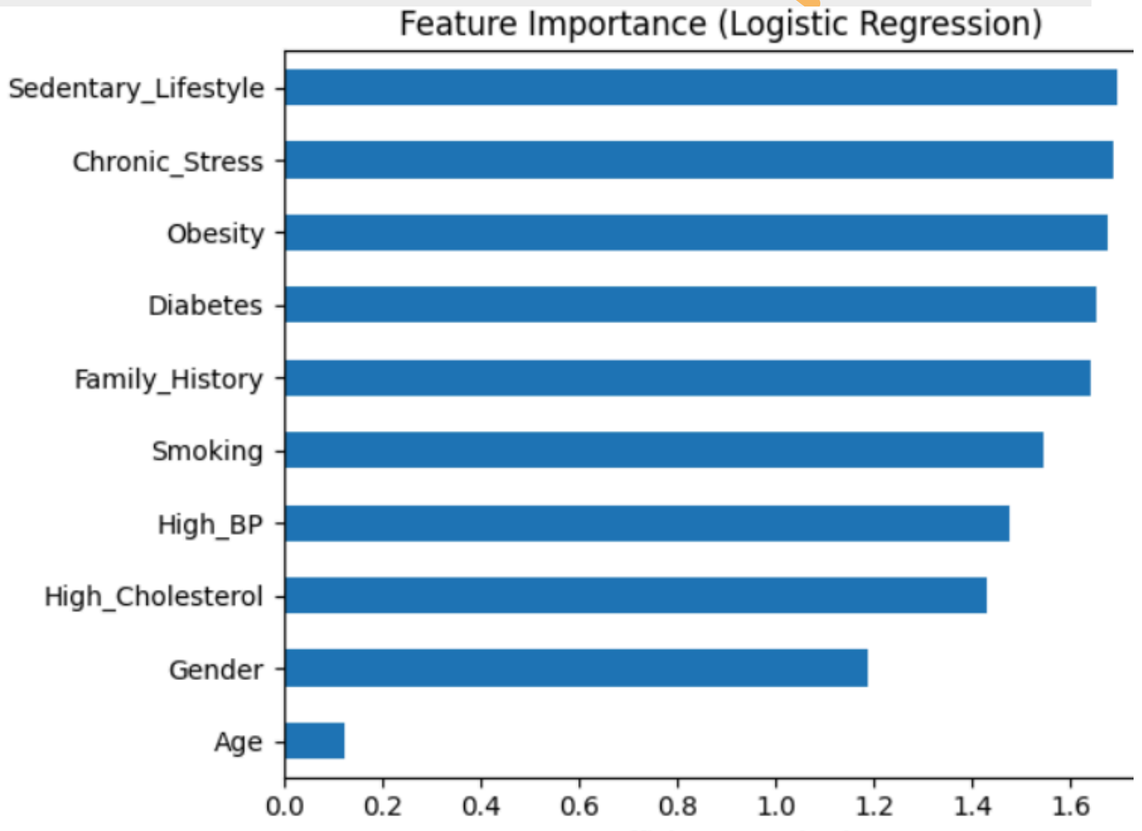
Data Visualization:



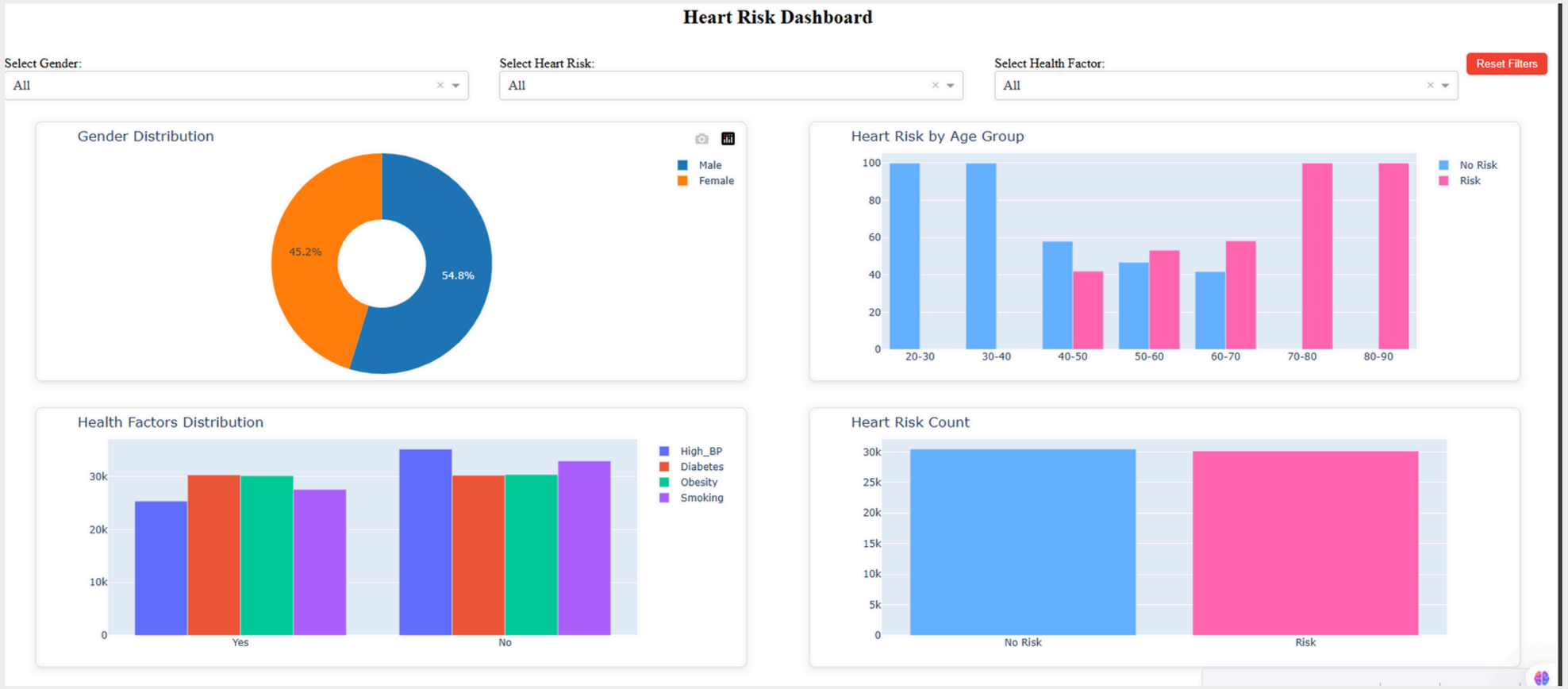
milestone 2:

Data Visualization:





milestone 2:


Heart Risk Dashboard
Select Gender:
All
Select Heart Risk:
All
Select Health Factor:
All
Reset Filters
Gender Distribution
Male
Female
45.2%
54.8%
Heart Risk by Age Group
No Risk
Risk
100
80
60
40
20
0
20-30
30-40
40-50
50-60
60-70
70-80
80-90
Health Factors Distribution
High_BP
Diabetes
Obesity
Smoking
30k
20k
10k
0
Yes
No
Heart Risk Count
30k
25k
20k
15k
10k
5k
0
No Risk
Risk

Milestone 3:

1. Model Selection

prepare data
before training

```python
from sklearn.model_selection import train_test_split

# Splitting the data into X (features) and y (target)
X = df.drop('Heart_Risk', axis=1)
y = df['Heart_Risk']

# Splitting the data into 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Milestone 3:

2. Model Training

```
# Defining the models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier(),
}

# Training the models and evaluating them
for name, model in models.items():
    model.fit(X_train, y_train)  # Train the model
    y_pred = model.predict(X_test)  # Make predictions on the test set

    print(f"Model: {name}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")  # Print accuracy
    print(f"Classification Report:\n {classification_report(y_test, y_pred)}")  # Print classification report
    print(f"Confusion Matrix:\n {confusion_matrix(y_test, y_pred)}")  # Print confusion matrix
    print("="*50)  # Print a separator line
```

Milestone 3:

3. Model Evaluation

```
Model: Logistic Regression
Accuracy: 0.9906765676567657
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6123
           1       0.99      0.99      0.99      5997

    accuracy                           0.99     12120
   macro avg       0.99      0.99      0.99     12120
weighted avg       0.99      0.99      0.99     12120

Confusion Matrix:
 [[6066   57]
 [  56 5941]]
```

```
Model: Random Forest
Accuracy: 0.9891089108910891
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6123
           1       0.99      0.99      0.99      5997

    accuracy                           0.99     12120
   macro avg       0.99      0.99      0.99     12120
weighted avg       0.99      0.99      0.99     12120

Confusion Matrix:
 [[6058   65]
 [  67 5930]]
```

Milestone 3:

3. Model Evaluation

```
Model: SVM
Accuracy: 0.9901815181518152
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6123
           1       0.99      0.99      0.99      5997

    accuracy                           0.99     12120
   macro avg       0.99      0.99      0.99     12120
weighted avg       0.99      0.99      0.99     12120

Confusion Matrix:
 [[6062   61]
 [  58 5939]]
```

```
Model: KNN
Accuracy: 0.9887788778877887
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6123
           1       0.99      0.99      0.99      5997

    accuracy                           0.99     12120
   macro avg       0.99      0.99      0.99     12120
weighted avg       0.99      0.99      0.99     12120

Confusion Matrix:
 [[6056   67]
 [  69 5928]]
```

Milestone 3:

4.Model Optimization

```python
# Define the Logistic Regression model
log_reg = LogisticRegression()

# Specify the parameters to test
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularization parameter (C)
    'solver': ['liblinear', 'saga'],  # Solver selection
    'max_iter': [100, 200, 300]  # Maximum number of iterations
}

# Apply GridSearchCV
grid_search = GridSearchCV(estimator=log_reg, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

# Print the best parameters
print(f"Best Parameters: {grid_search.best_params_}")

# Evaluate the model using the best parameters
best_log_reg = grid_search.best_estimator_
y_pred = best_log_reg.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy after hyperparameter tuning: {accuracy}")
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'C': 10, 'max_iter': 100, 'solver': 'saga'}
Accuracy after hyperparameter tuning: 0.9905940594059406
```

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(best_log_reg, X, y, cv=5)
print(f"Cross-Validation Accuracy: {scores.mean():.4f} ± {scores.std():.4f}")
```

```
Cross-Validation Accuracy: 0.9902 ± 0.0011
```

Milestone 4:

1.Model Deployment

1.Model Saving

```
import joblib

# Save the trained model
joblib.dump(best_log_reg, 'final_logistic_regression_model.pkl')
```

✓ 0.0s

2.Model Loading

```
try:
    model = joblib.load('final_logistic_regression_model.pkl')
except FileNotFoundError:
    st.error("❌ Model file not found. Please make sure 'final_logistic_regression_model.pkl' exists.")
    st.stop()
```

Milestone 4:

1.Model Deployment

Heart Disease Risk Prediction ❤️

This app uses **Heart Disease Prediction** based on several symptoms and factors.
Select the symptoms, and we will tell you if you're at risk of heart disease.

Do you have Chest pain?

- Yes
- No

Do you have Fatigue?

- Yes
- No

Do you have Dizziness?

- Yes
- No

Do you have Pain in arms/jaw/back?

- Yes
- No

Do you have High cholesterol?

- Yes
- No

Do you have Shortness of breath?

- Yes
- No

Do you have Palpitations?

- Yes
- No

Do you have Swelling (legs or ankles)?

- Yes
- No

Do you have Cold sweats or nausea?

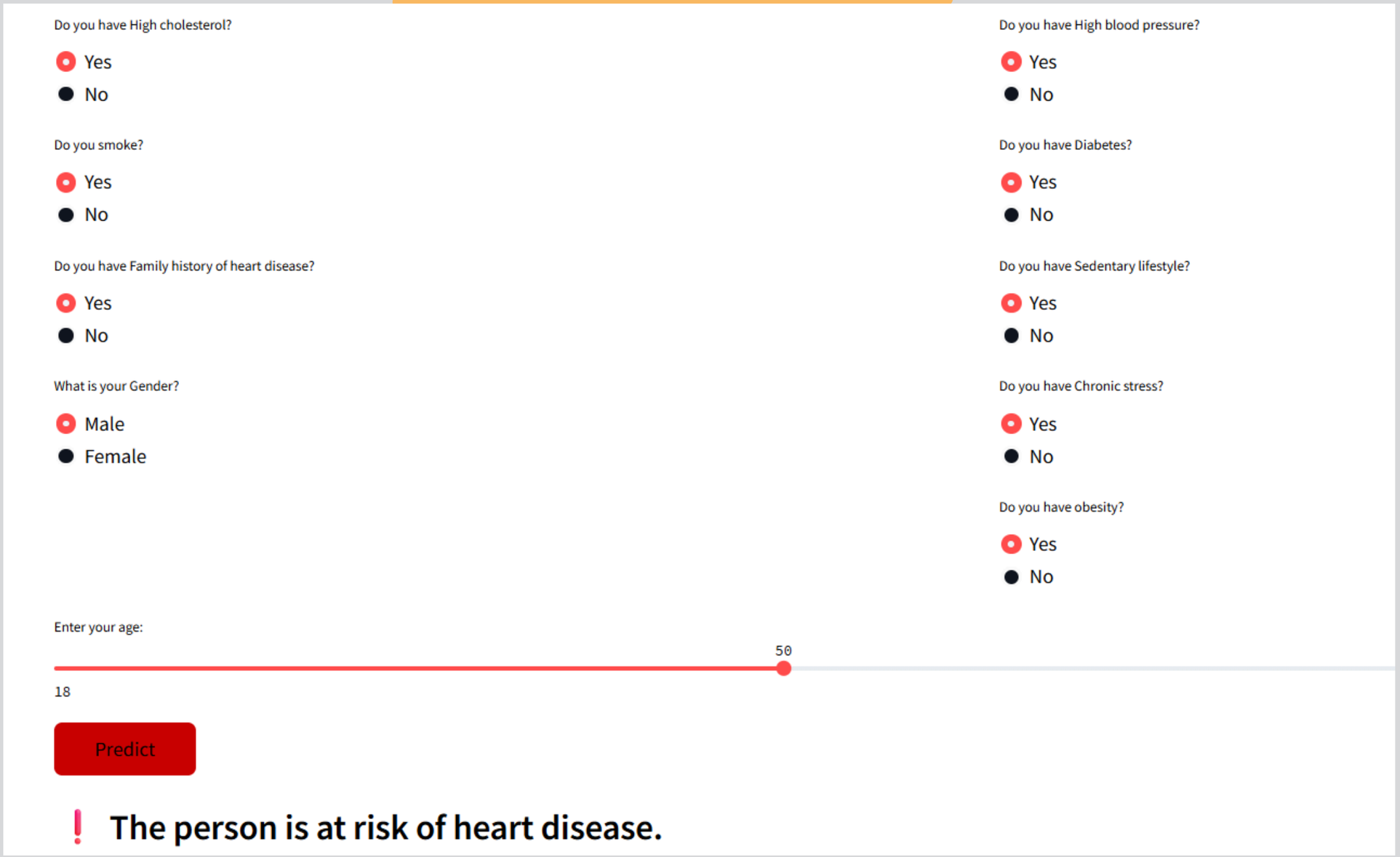
- Yes
- No

Do you have High blood pressure?

- Yes
- No

Milestone 4:

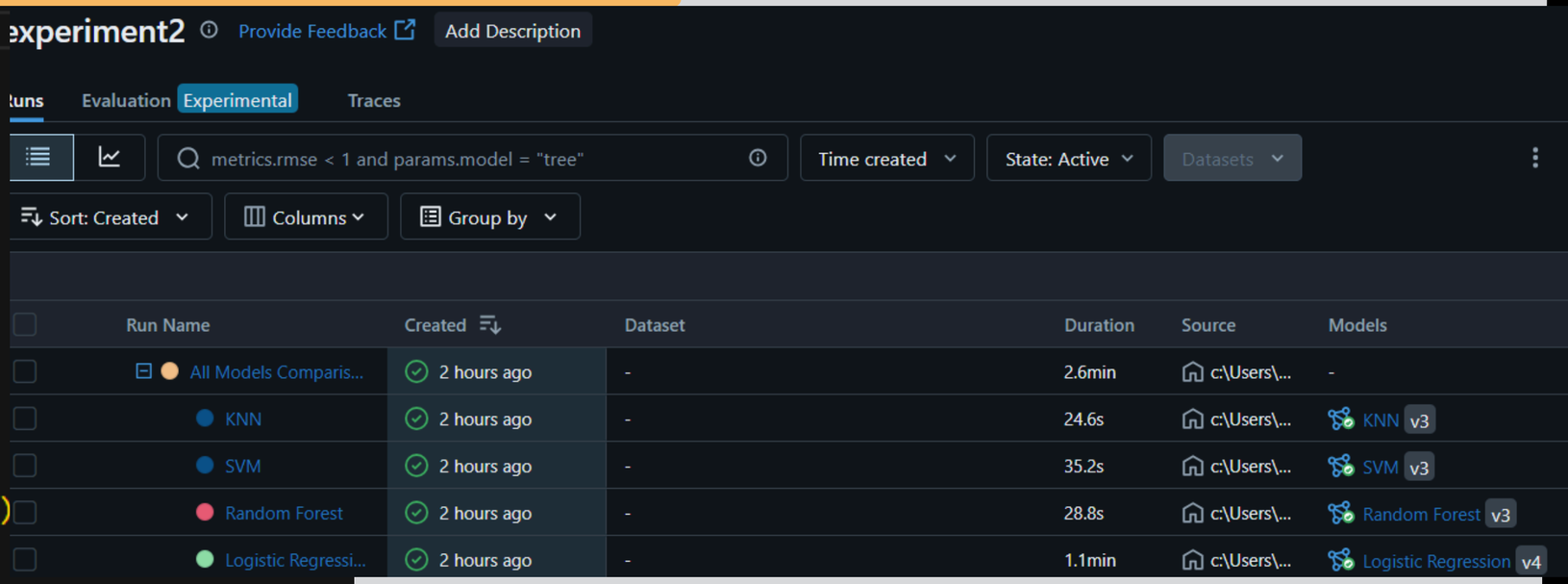
1.Model Deployment



Milestone 4:

2.MLFlow Models Tracking

```python
mlflow.set_tracking_uri("http://127.0.0.1:5000")
mlflow.set_experiment("experiment2")
client = MlflowClient()
with mlflow.start_run():
    for name, model in models.items():
        with mlflow.start_run(nested=True):
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            version=1
            mlflow.log_param("model_name", name)
            mlflow.log_param("algorithm", name)

            accuracy = accuracy_score(y_test, y_pred)
            mlflow.log_metric("accuracy", accuracy)

            mlflow.sklearn.log_model(model, name)
            mlflow.sklearn.log_model(model, "model")

            # Build model URI from the run ID
            model_uri = f"runs:/{mlflow.active_run().info.run_id}/model"

            # Register the model to the MLflow Model Registry
            result = mlflow.register_model(model_uri=model_uri, name=name)
            version = result.version  # Automatically assigned version

mlflow.end_run()
```



Milestone 4:

comparing the models

Default ›

Comparing 4 Runs from 1 Experiment

Visualizations

Parallel Coordinates Plot | Scatter Plot | Box Plot | Contour Plot

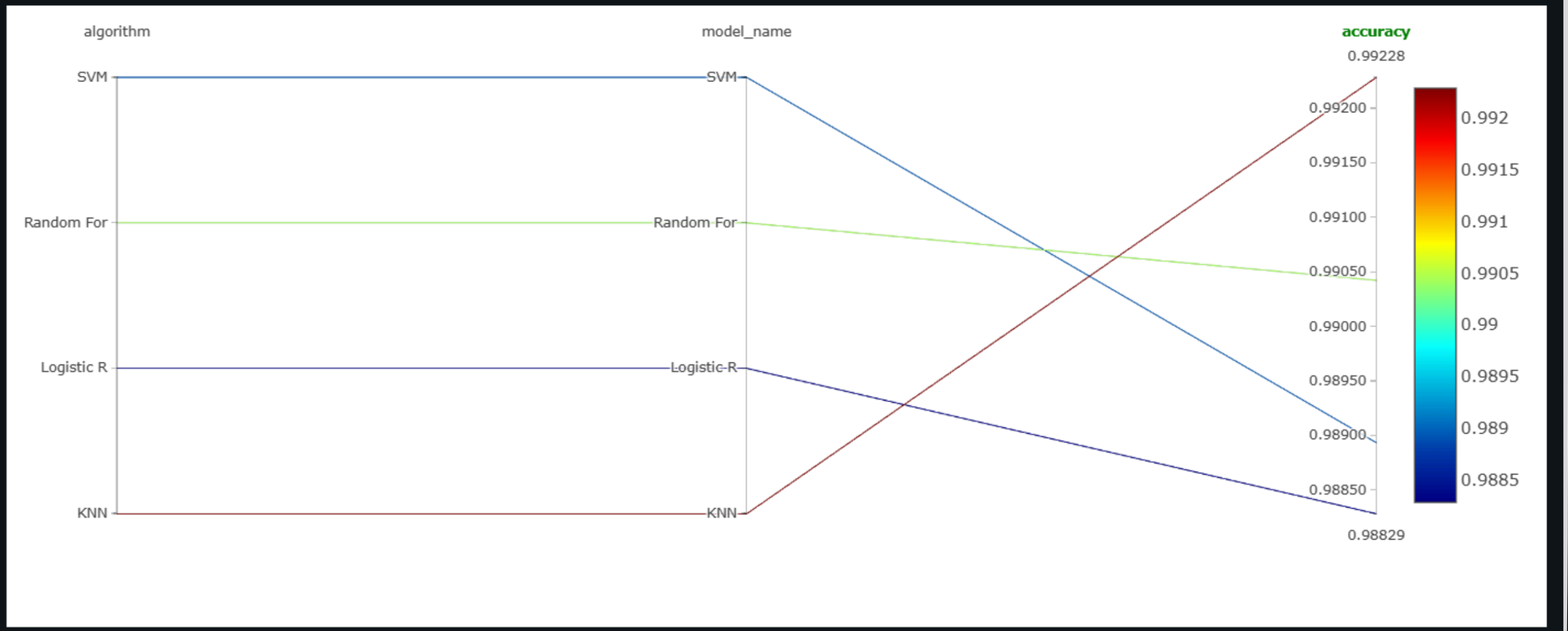
Parameters:

algorithm × model_name ×

Metrics:

accuracy ×

Clear All



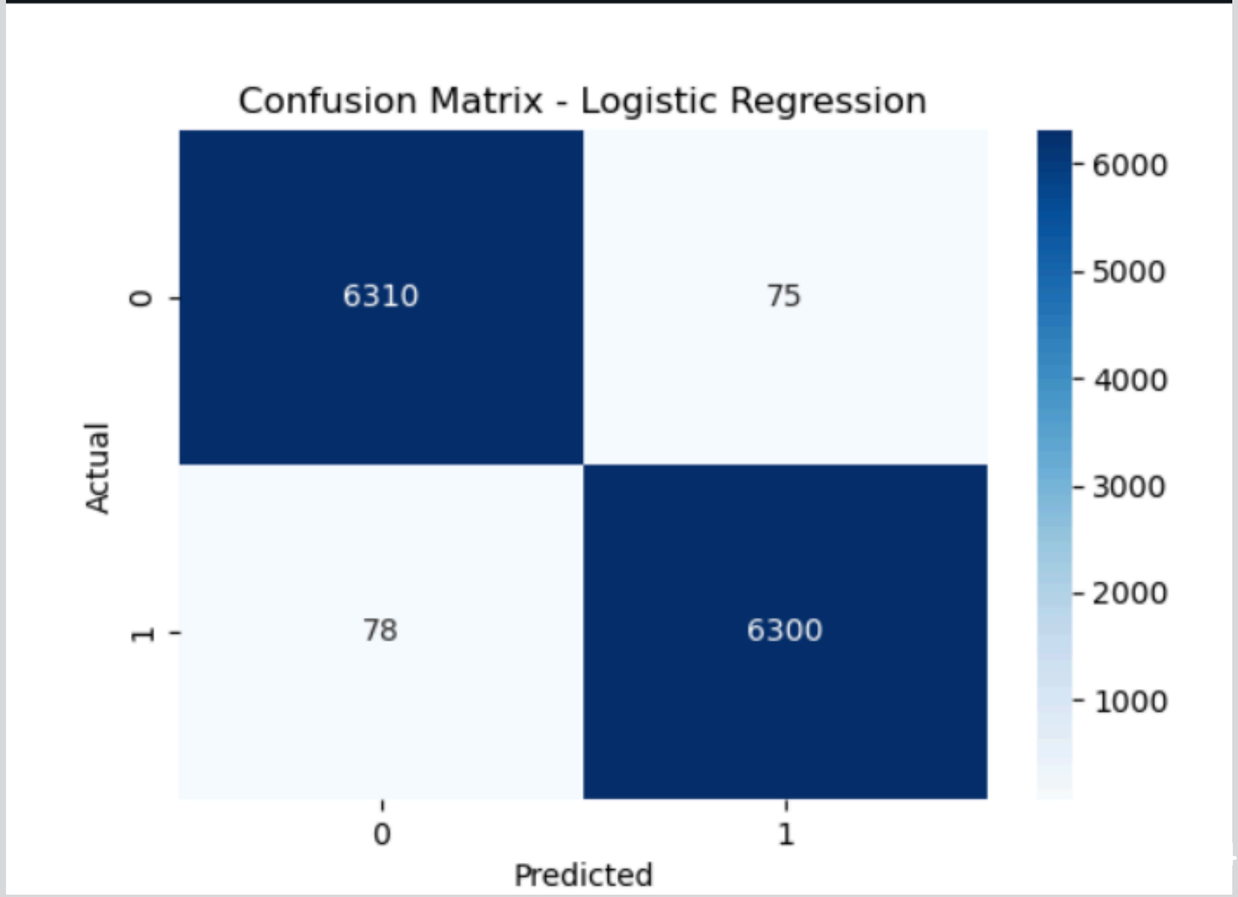
Milestone 4:

Performance Monitoring & Reporting

```python
# Log classification report
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_path = f"classification_report_{name}.csv"
report_df.to_csv(report_path)
mlflow.log_artifact(report_path)

# Log confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title(f"Confusion Matrix - {name}")
plt.xlabel("Predicted")
plt.ylabel("Actual")
cm_path = f"confusion_matrix_{name}.png"
plt.savefig(cm_path)
mlflow.log_artifact(cm_path)
plt.close()
```

confusion_matrix_Logistic Regression.png 19.56KB

Path: mlflow-artifacts:/149914103756672466/1bb2e59387bb4584a02197efcba967ef/artifa

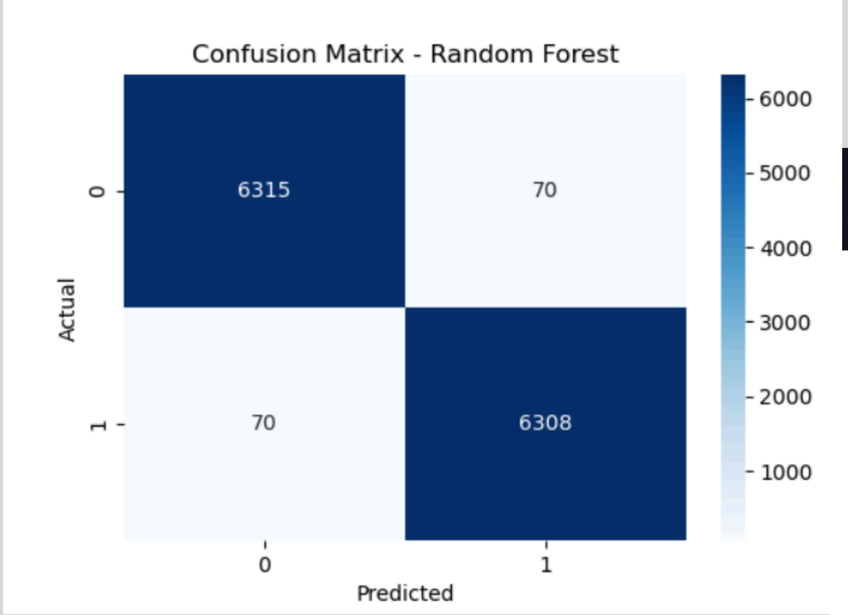


Milestone 4:

Performance Monitoring & Reporting

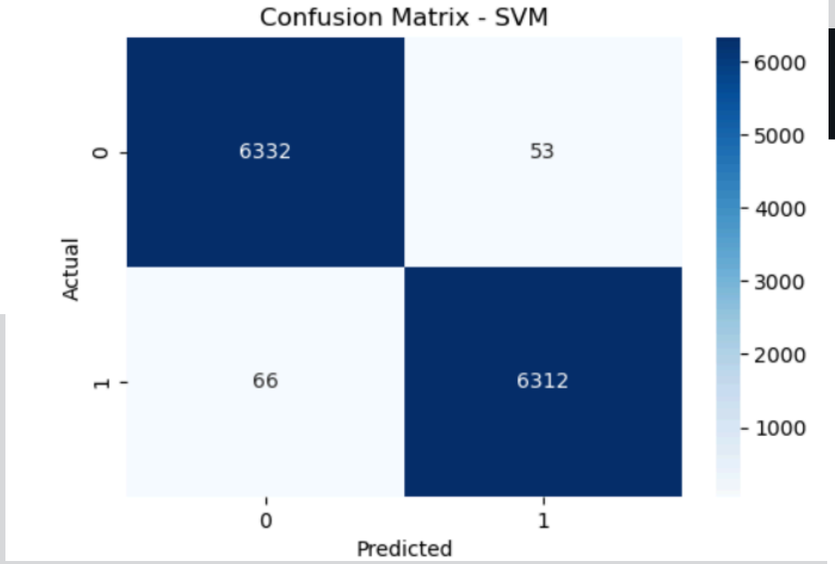
confusion_matrix_Random Forest.png 19KB

Path: mlflow-artifacts:/149914103756672466/e1e255dedcb349768654ffe89c5a6798/artif



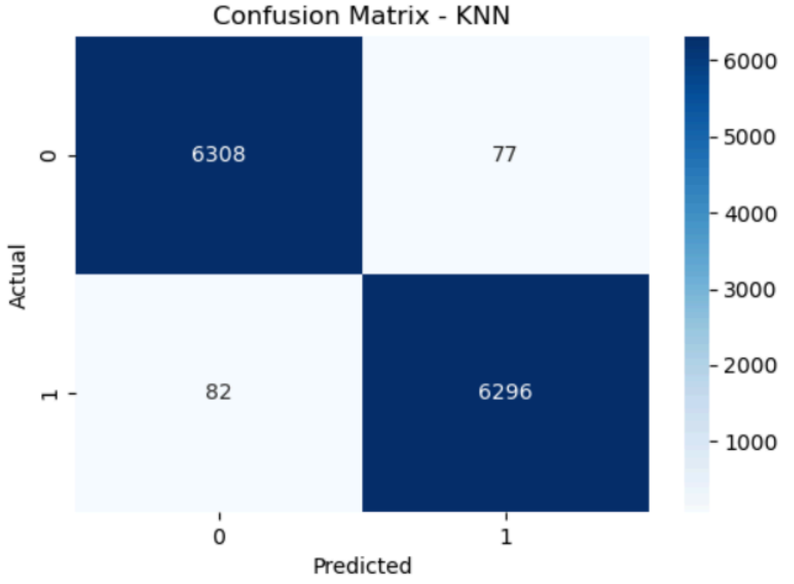
confusion_matrix_SVM.png 17.73KB

Path: mlflow-artifacts:/14991410375667246 6/b9fa721685f242f8bfb26149f077e823/artif



confusion_matrix_KNN.png 17.6KB

Path: mlflow-artifacts:/149914103756672466/d430b400c9b84d529eac091f397994a3/ar



Our Team

1. Reham Khaled
2. Alaa Ramadan
3. Aya Gamal
4. Nada Eslam

Thank you