HEART DISEASE PREDICTION PROJECT-01

Milestone 1: Data Collection, Exploration, and Preprocessing

1. Data Understanding:

column	Feature	type
age	Objective	int (days)
height	Objective	int (cm)
weight	Objective	float (kg)
gender	Objective	categorical code
ap_hi	Examination	int
ap_lo	Examination	int
cholesterol	Examination	1: normal, 2: above normal, 3: well above normal
gluc	Examination	1: normal, 2: above normal, 3: well above normal
smoke	Subjective	binary
alco	Subjective	binary
active	Subjective	binary
cardio	Target	binary

2. Data Cleaning & Preprocessing:

```
df=pd.read_csv("Cardiovasculardataset.csv")
df.head()
```

[2]

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62	110	80	1	1	0	0	1	0
1	1	20228	1	156	85	140	90	3	1	0	0	1	1
2	2	18857	1	165	64	130	70	3	1	0	0	0	1
3	3	17623	2	169	82	150	100	1	1	0	0	1	1
4	4	17474	1	156	56	100	60	1	1	0	0	0	0

```
df.info()
```

[3]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           70000 non-null  int64
 1   age          70000 non-null  int64
 2   gender       70000 non-null  int64
 3   height       70000 non-null  int64
 4   weight       70000 non-null  int64
 5   ap_hi        70000 non-null  int64
 6   ap_lo        70000 non-null  int64
 7   cholesterol  70000 non-null  int64
 8   gluc         70000 non-null  int64
 9   smoke        70000 non-null  int64
 10  alco         70000 non-null  int64
 11  active       70000 non-null  int64
 12  cardio       70000 non-null  int64
dtypes: int64(13)
memory usage: 6.9 MB
```

```
df.describe()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000
mean	49972.419900	19468.865814	1.349571	164.359229	74.205543	128.817714	96.630414	1.366871	1.226457	0.088129	0.053771	0.803729	0.499700
std	28851.302323	2467.251667	0.476838	8.210126	14.395829	154.011381	188.472530	0.680250	0.572270	0.283484	0.225568	0.397179	0.500003
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.000000	1.000000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000

```
df.shape
```

```
(70000, 13)
```

Handling Nulls & Duplicates:

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

```
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

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

```
0
```

no nulls or duplicates

converting age into years:

```
df['age']=df['age']/365
df
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	50.391781	2	168	62	110	80	1	1	0	0	1	0
1	55.419178	1	156	85	140	90	3	1	0	0	1	1
2	51.663014	1	165	64	130	70	3	1	0	0	0	1
3	48.282192	2	169	82	150	100	1	1	0	0	1	1
4	47.873973	1	156	56	100	60	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
69995	52.712329	2	168	76	120	80	1	1	1	0	1	0
69996	61.920548	1	158	126	140	90	2	2	0	0	1	1
69997	52.235616	2	183	105	180	90	3	1	0	1	0	1
69998	61.454795	1	163	72	135	80	1	2	0	0	0	1
69999	56.273973	1	170	72	120	80	2	1	0	0	1	0

70000 rows × 12 columns

```
df['age']=df['age'].round().astype(int)
df.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	50	2	168	62	110	80	1	1	0	0	1	0
1	55	1	156	85	140	90	3	1	0	0	1	1
2	52	1	165	64	130	70	3	1	0	0	0	1
3	48	2	169	82	150	100	1	1	0	0	1	1
4	48	1	156	56	100	60	1	1	0	0	0	0

Insert BMI Column:

```
[12]
df['BMI'] = df['weight'] / ((df['height'] / 100) ** 2)
df.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	BMI
0	50	2	168	62	110	80	1	1	0	0	1	0	21.967120
1	55	1	156	85	140	90	3	1	0	0	1	1	34.927679
2	52	1	165	64	130	70	3	1	0	0	0	1	23.507805
3	48	2	169	82	150	100	1	1	0	0	1	1	28.710479
4	48	1	156	56	100	60	1	1	0	0	0	0	23.011177

Define Blood pressure status:

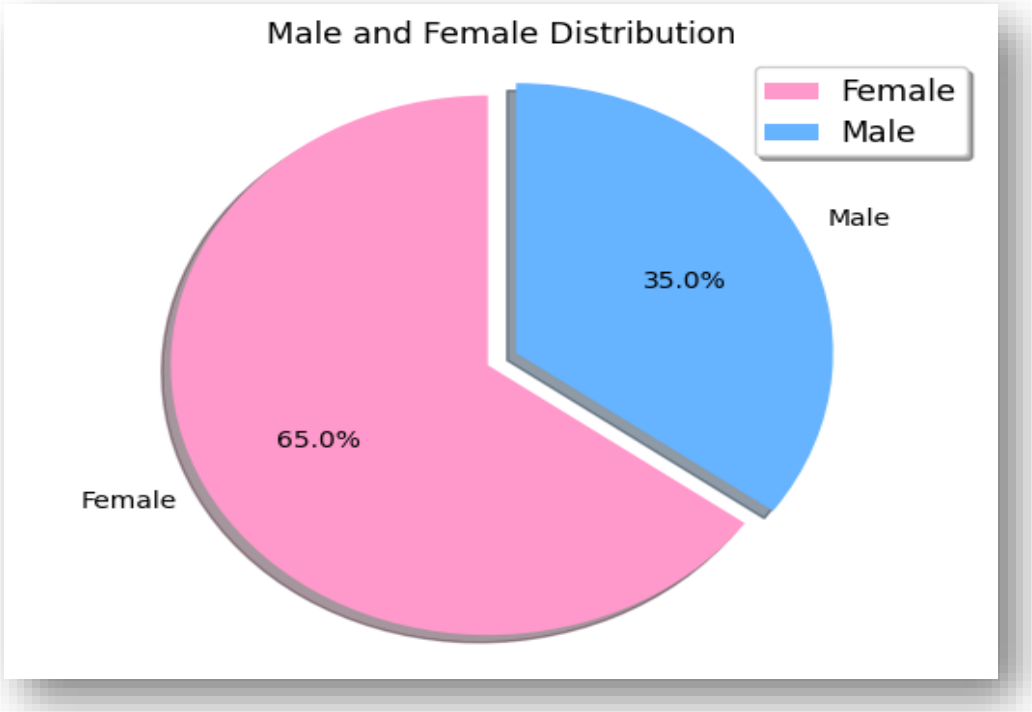
```
def bp_status(row):
    if row['ap_hi'] >= 140 or row['ap_lo'] >= 90:
        return 'High'
    elif (row['ap_hi'] >= 130) or (row['ap_lo'] >= 80):
        return 'Elevated'
    else:
        return 'Normal'

# Apply the function to create a new column
df['bp_status'] = df.apply(bp_status, axis=1)
```

```
df.head()
```

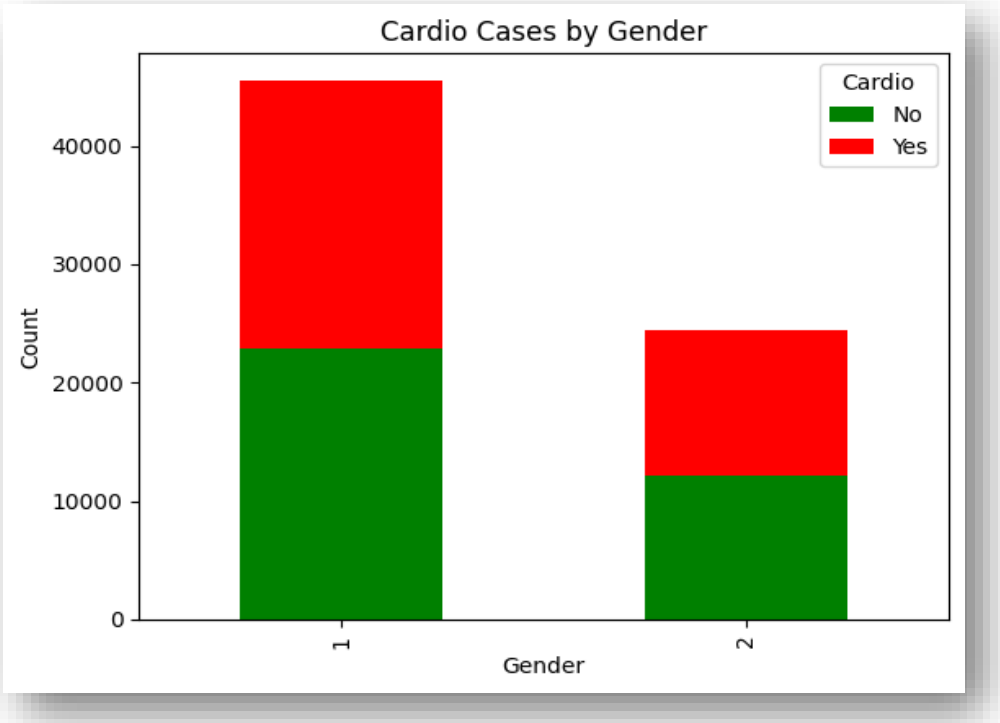
	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	BMI	bp_status
0	50	2	168	62	110	80	1	1	0	0	1	0	21.967120	Elevated
1	55	1	156	85	140	90	3	1	0	0	1	1	34.927679	High
2	52	1	165	64	130	70	3	1	0	0	0	1	23.507805	Elevated
3	48	2	169	82	150	100	1	1	0	0	1	1	28.710479	High
4	48	1	156	56	100	60	1	1	0	0	0	0	23.011177	Normal

Milestone 2: Data Analysis and Visualization

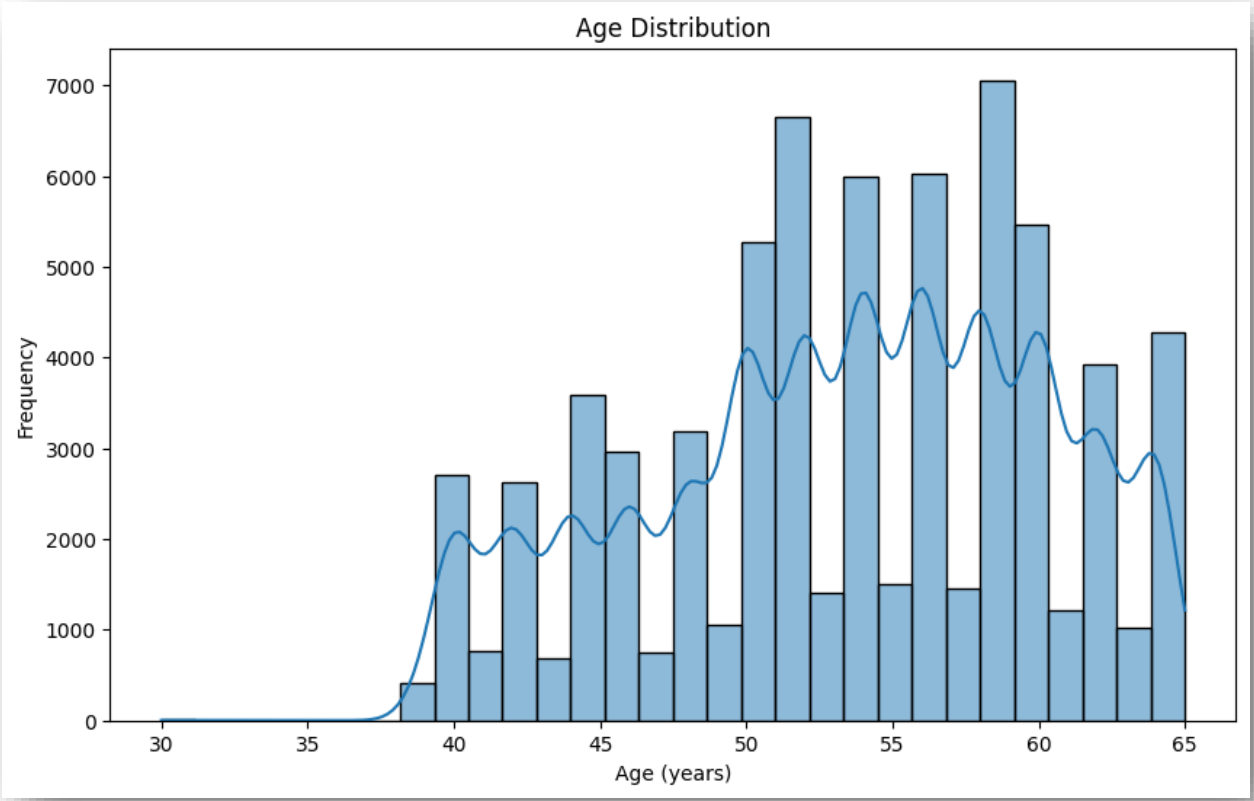
Gender Distribution in dataset:



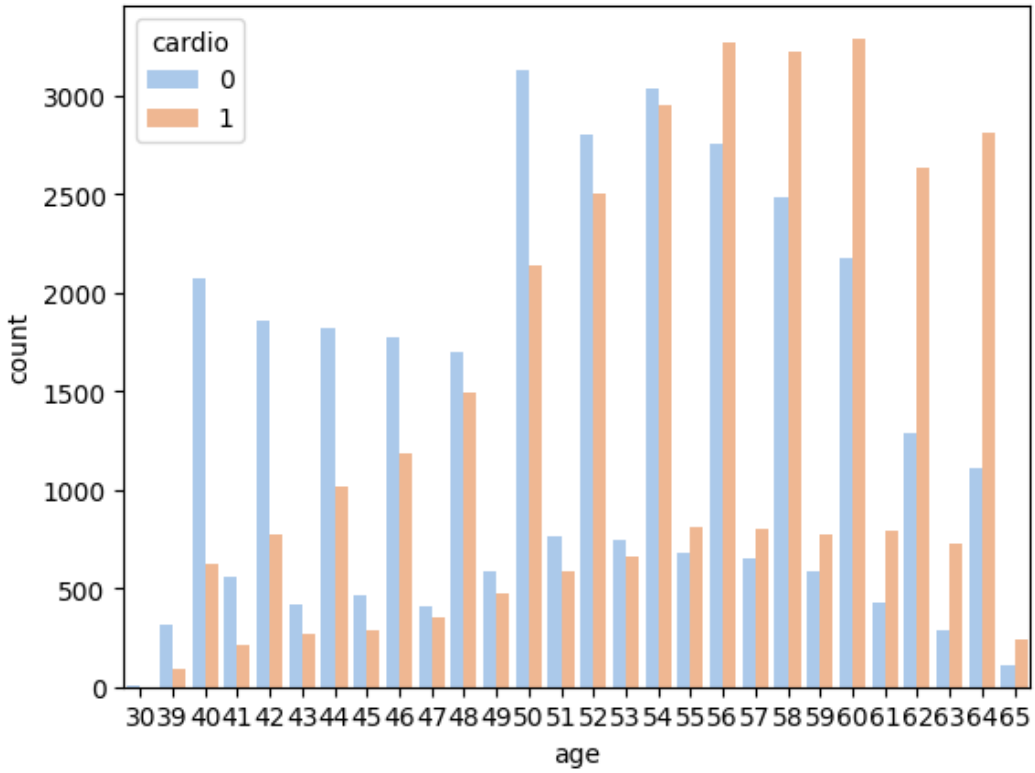
Cardio cases according to gender



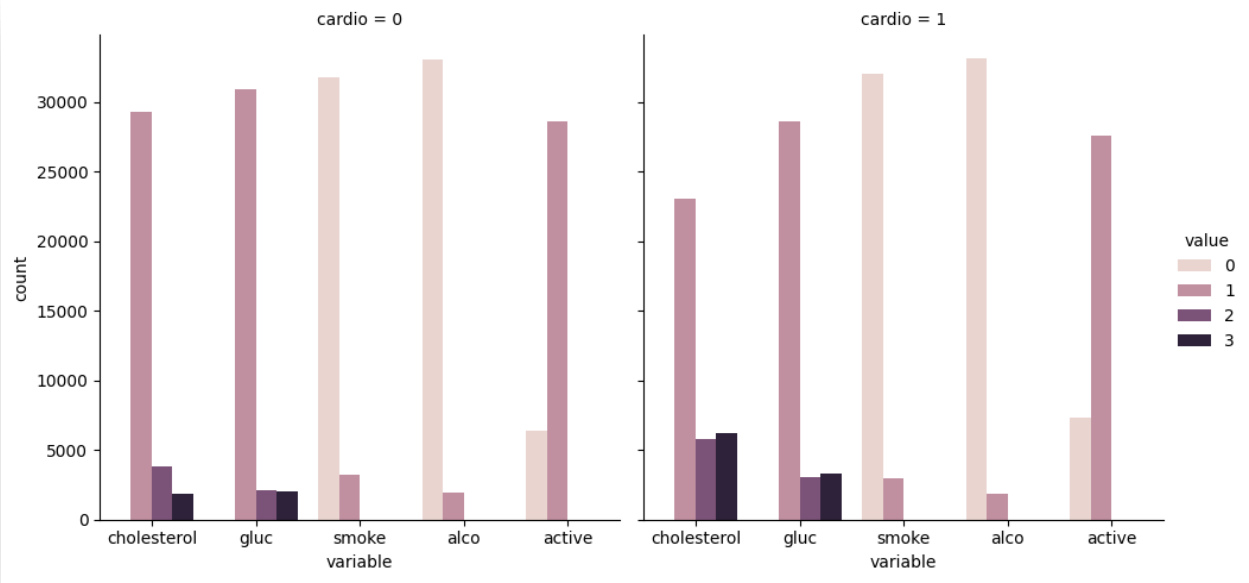
Age distribution in Dataset:



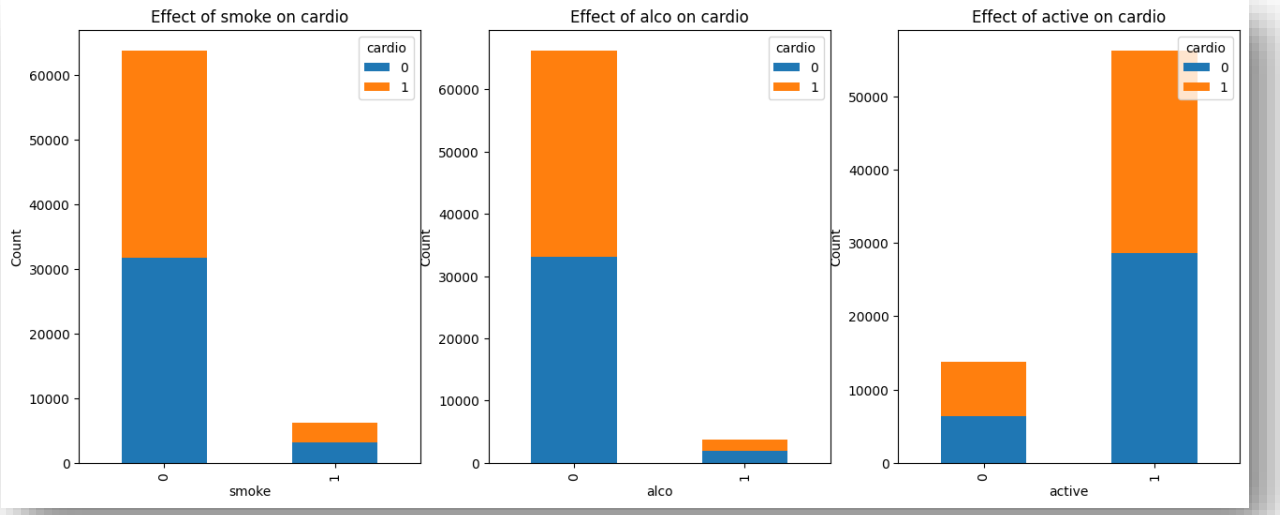
Age distribution & cardio:



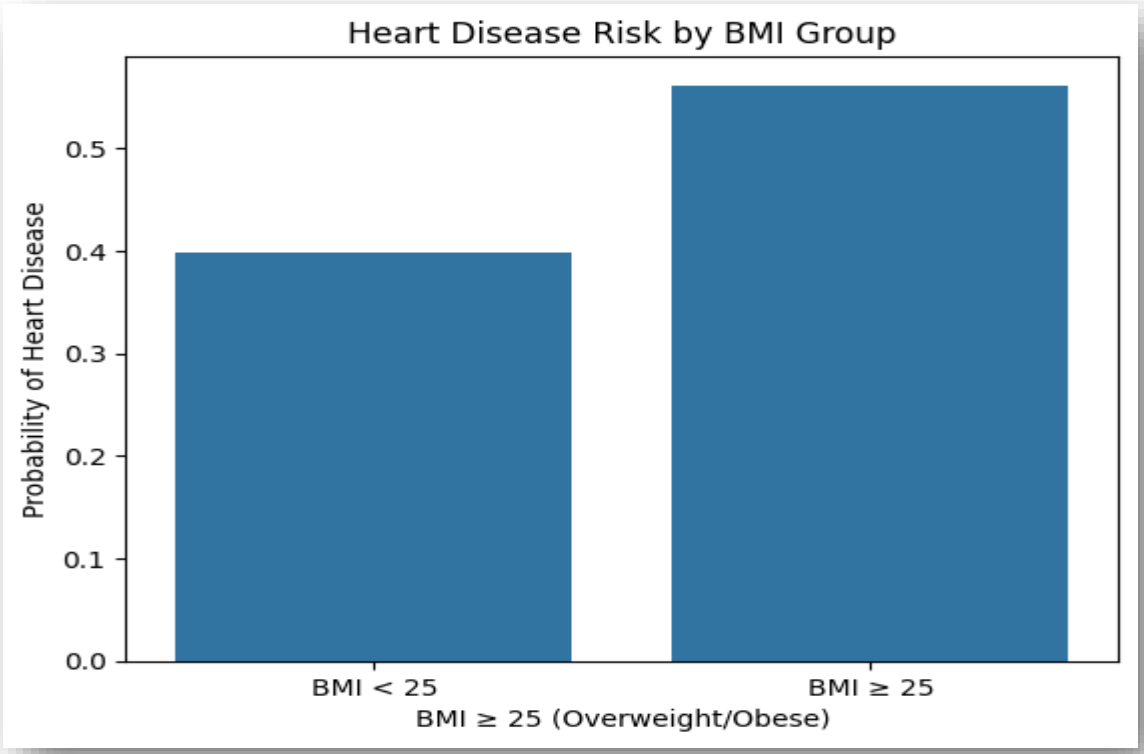
The difference in distribution between cardiac disease people and non-cardiac



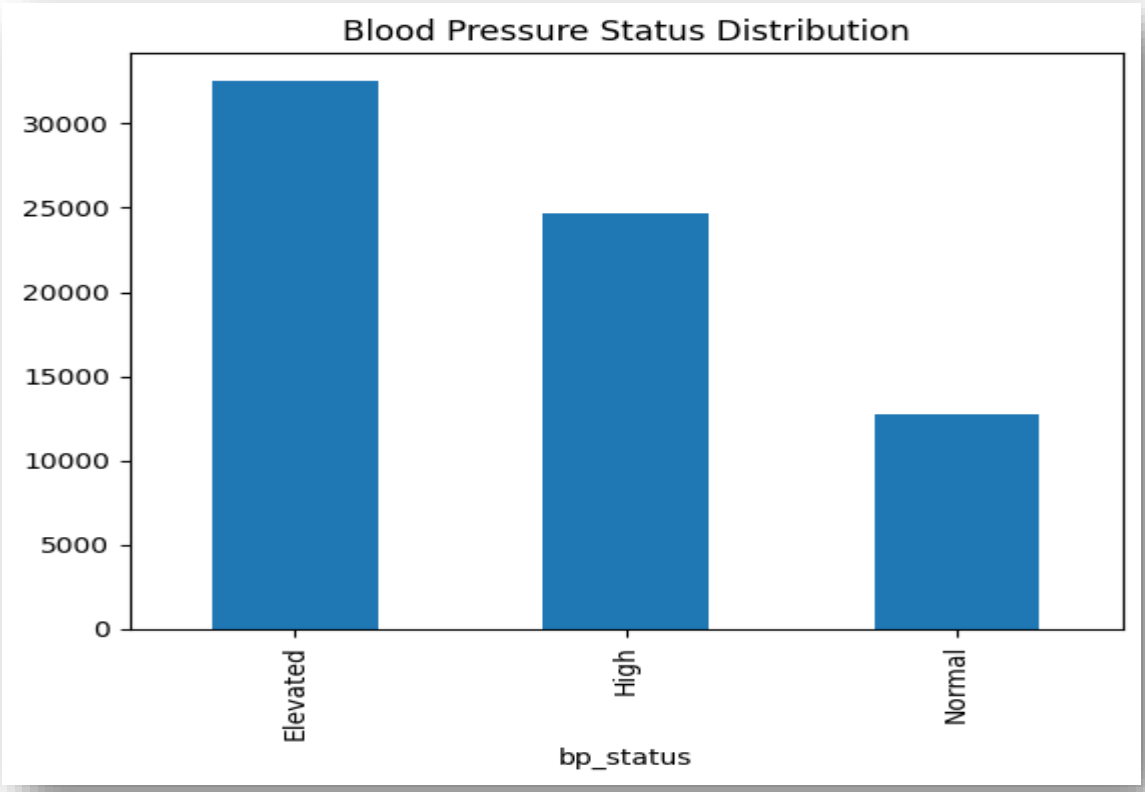
The Effect of subjective features on Cardiac health



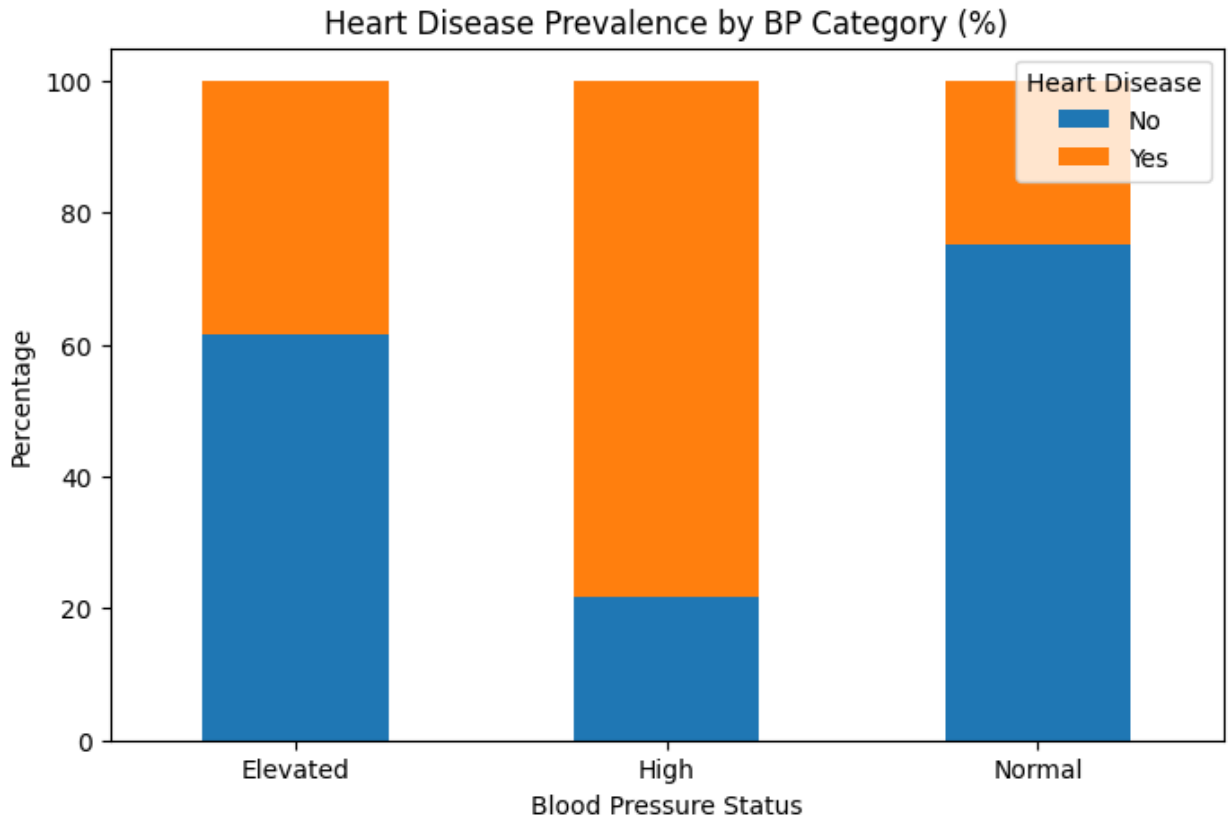
Heart Risk according to BMI:



Blood pressure distribution



heart disease distribution according to blood pressure:



Revealed Insights:

- Data set has more Female distribution so model may be biased toward Females.
- The heart dieses is even according to genders which is inaccurate in real life as Males are more prone to having heart diseases.
- When age increases above 50 probability of heart dieses increase
- Subjective features like smoking, Physical activity & Alcohol has no effect on heart diseases according to the dataset which is inaccurate in real life.
- When BMI increases above 25 probability of heart dieses increase.
- At High Blood pressure the probability of heart diseases is higher.

Its expected that the model accuracy is not the Best as the dataset insights are different from real life

Milestone 3: Predictive Model Development and Optimization

Scaling the data for better Model Performance:

```
from sklearn.preprocessing import StandardScaler

# Scale continuous features
scaler = StandardScaler()
df[continuous_features] = scaler.fit_transform(df[continuous_features])

# Display the first few rows of the scaled data
df.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	BMI	bp_status	BMI_High
0	-0.505507	2	0.477190	-0.911126	-1.149154	-0.221435	1	1	0	0	1	0	21.967120	Elevated	0
1	0.236325	1	-1.116157	0.963069	0.950292	1.081779	3	1	0	0	1	1	34.927679	High	1
2	-0.208775	1	0.078853	-0.748152	0.250477	-1.524649	3	1	0	0	0	1	23.507805	Elevated	0
3	-0.802240	2	0.609969	0.718609	1.650107	2.384993	1	1	0	0	1	1	28.710479	High	1
5	0.978157	1	-1.780052	-0.503692	-0.449339	-0.221435	2	2	0	0	0	0	29.384676	Elevated	1

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
```

```
# Perform one-hot encoding for categorical variables
df_encoded = pd.get_dummies(df, columns=['gender', 'cholesterol', 'gluc','bp_status'])


# Split into dependent and independent features
X = df_encoded.drop('cardio', axis=1)
y = df_encoded['cardio']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
```

Metrics after Logical Regression:

```
[151]  lr.score(X_train,y_train)

...    0.7242004586585128

[152]  lr.score(X_test,y_test)

...    0.7283634618461047
```

Metrics after Random Forrest Classifier:

```
from sklearn.ensemble import RandomForestClassifier

rand_forest = RandomForestClassifier()
rand_forest.fit(X_train,y_train)

print('Train Accuracy: ', rand_forest.score(X_train,y_train))
print('Test Accuracy: ', rand_forest.score(X_test,y_test))
```

53]

```
Train Accuracy:  0.9720893850775986
Test Accuracy:  0.7022876339785634
```

Overfitting very apparent

Applying Grid search to choose Best Parameters & revaluate the Model:

```
#choosing best parameters
grid_search = GridSearchCV(rand_forest, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print('Best Parameters: ', best_params)
```

[155]

```
Best Parameters:  {'max_depth': 10, 'n_estimators': 200}
```

```
best_rand_forest = RandomForestClassifier(**best_params)
best_rand_forest.fit(X_train, y_train)
print('Train Accuracy: ', best_rand_forest.score(X_train,y_train))
print('Test Accuracy: ', best_rand_forest.score(X_test,y_test))
```

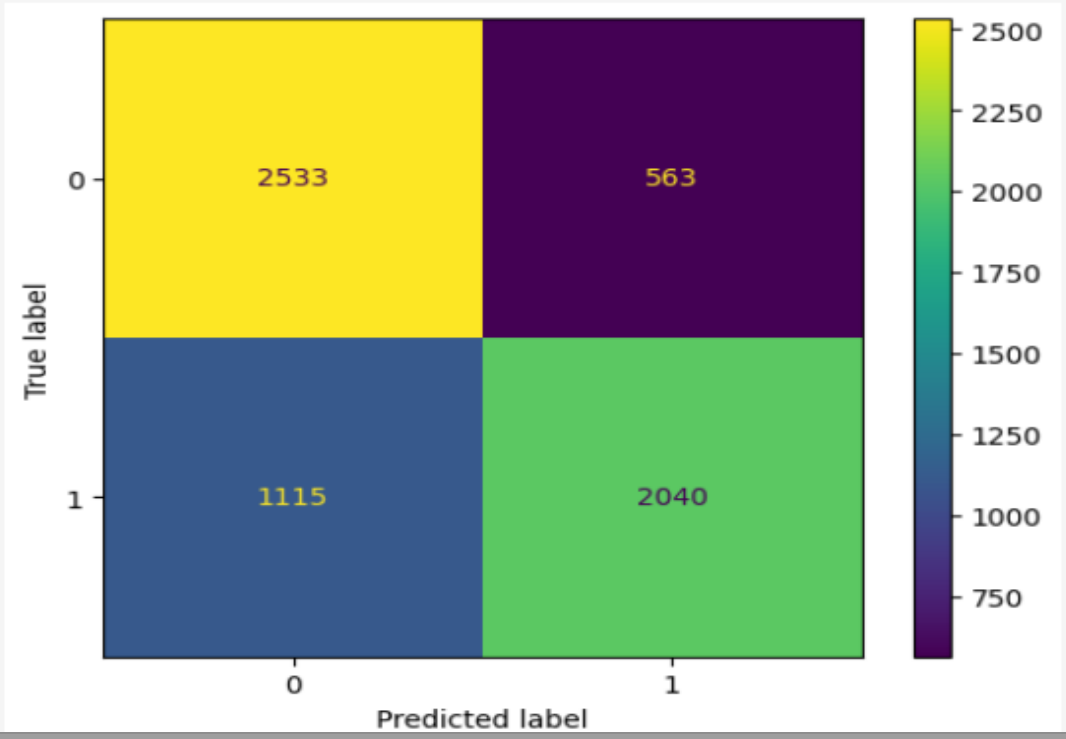
[156]

```
Train Accuracy:  0.745408970507191
Test Accuracy:  0.7315629499280115
```

```
[158]
print(classification_report(y_test, y_predictions))
```

```
...                precision    recall  f1-score   support

           0       0.69      0.82      0.75      3096
           1       0.78      0.65      0.71      3155

    accuracy                           0.73      6251
   macro avg       0.74      0.73      0.73      6251
weighted avg       0.74      0.73      0.73      6251
```

```
[159]
ConfusionMatrixDisplay.from_predictions(y_test, y_predictions)
plt.show()
```



Better accuracy than Logical regression So more suitable For Model deployment

Milestone 3: Deployment

Using pickle Module to load the model and scaler.

```
62]
    with open('model.pkl', 'wb') as file:
        pickle.dump(model, file)

    with open('scaler.pkl', 'wb') as file:
        pickle.dump(scaler, file)
```

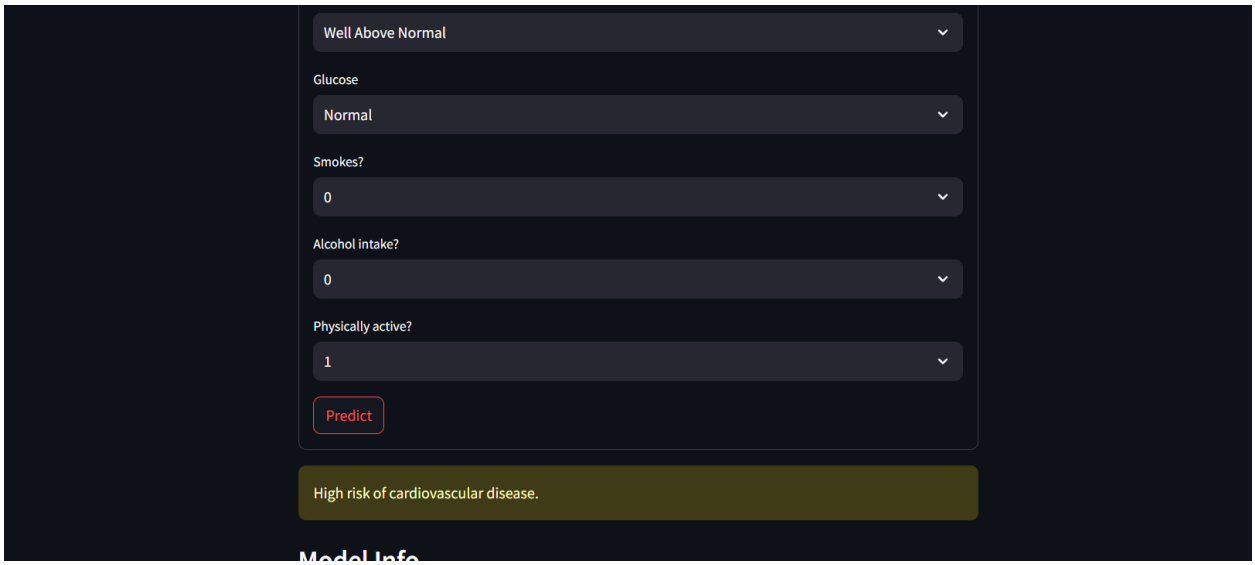
Deployment app Using streamlit:

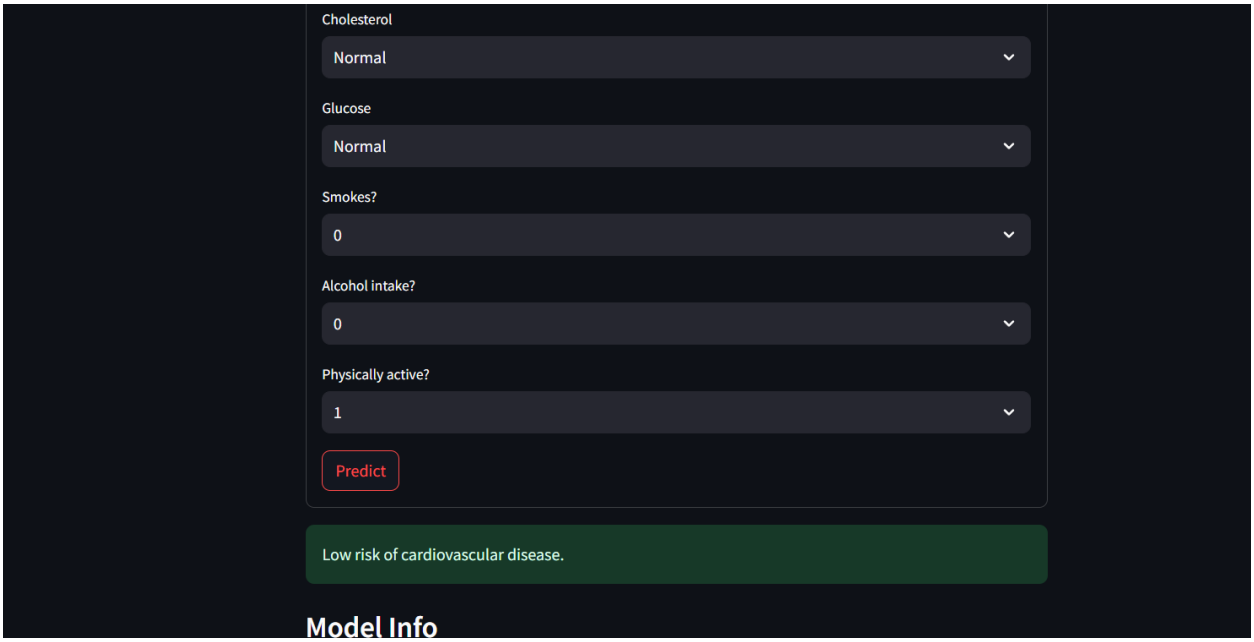
```
import streamlit as st
import pandas as pd
import numpy as np
import pickle

# Load the trained model and scaler
try:
    with open("model.pkl", "rb") as f:
        model = pickle.load(f)
    with open("scaler.pkl", "rb") as f:
        scaler = pickle.load(f)
except FileNotFoundError as e:
    st.error(f"Error: {e}. Please make sure model.pkl and scaler.pkl are in the
app directory.")
    st.stop()

def bp_status(row):
    if row['ap_hi'] >= 140 or row['ap_lo'] >= 90:
        return 'High'
    elif (row['ap_hi'] >= 130) or (row['ap_lo'] >= 80):
        return 'Elevated'
    else:
        return 'Normal'

# Streamlit app
st.title("Cardiovascular Disease Prediction")
st.write("Fill in the patient data to predict the risk of cardiovascular
disease.")
```

```
input_data = {}

with st.form("prediction_form"):
    input_data['age'] = st.number_input("Age (in years)", min_value=1,
max_value=120, value=50)*365
    input_data['gender'] = st.selectbox("Gender", options=[1, 2],
format_func=lambda x: "Female" if x == 1 else "Male")
    input_data['height'] = st.number_input("Height (in cm)", min_value=100,
max_value=250, value=165)
    input_data['weight'] = st.number_input("Weight (in kg)", min_value=30.0,
max_value=200.0, value=70.0)
    input_data['ap_hi'] = st.number_input("Systolic BP (ap_hi)", value=120)
    input_data['ap_lo'] = st.number_input("Diastolic BP (ap_lo)", value=80)
    input_data['cholesterol'] = st.selectbox("Cholesterol", options=[1, 2, 3],
format_func=lambda x: ["Normal", "Above Normal", "Well Above Normal"][x - 1])
    input_data['gluc'] = st.selectbox("Glucose", options=[1, 2, 3],
format_func=lambda x: ["Normal", "Above Normal", "Well Above Normal"][x - 1])
    input_data['smoke'] = st.selectbox("Smokes?", options=[0, 1])
    input_data['alco'] = st.selectbox("Alcohol intake?", options=[0, 1])
    input_data['active'] = st.selectbox("Physically active?", options=[0, 1])

    submitted = st.form_submit_button("Predict")

if submitted:
    try:
        # Create DataFrame from input
        input_df = pd.DataFrame([input_data])

        # Calculate BMI
        height_m = input_data['height'] / 100
        input_df['BMI'] = input_df['weight'] / (height_m ** 2)

        # Determine BP status
        input_df['bp_status'] = input_df.apply(bp_status, axis=1)

        # Create BMI_High feature
        input_df['BMI_High'] = (input_df['BMI'] >= 25).astype(int)

        # One-hot encode categorical features
        categorical_cols = ['bp_status', 'cholesterol', 'gluc', 'gender']
        input_df = pd.get_dummies(input_df, columns=categorical_cols)


        model_features = model.feature_names_in_
```

```
        # Add missing columns with 0 and reorder to match model
        for feature in model_features:
            if feature not in input_df.columns:
                input_df[feature] = 0

        input_df = input_df[model_features]

        # Scale continuous features
        continuous_features = ['age', 'height', 'weight', 'ap_hi', 'ap_lo']
        input_df[continuous_features] =
scaler.transform(input_df[continuous_features])

        # Make prediction
        prediction = model.predict(input_df)[0]

        if prediction == 1:
            st.warning("High risk of cardiovascular disease.")
        else:
            st.success("Low risk of cardiovascular disease.")

    except Exception as e:
        st.error(f"Prediction error: {str(e)}")
        st.error("Please check that all input values are valid.")

# Info section
st.markdown("""
### Model Info
This model uses a Random Forest Classifier trained on a cardiovascular dataset.
Inputs include age, BMI, blood pressure, cholesterol, lifestyle factors, etc.
""")
```

Output:


Well Above Normal
Glucose
Normal
Smokes?
0
Alcohol intake?
0
Physically active?
1
Predict
High risk of cardiovascular disease.


Cholesterol
Normal
Glucose
Normal
Smokes?
0
Alcohol intake?
0
Physically active?
1
Predict
Low risk of cardiovascular disease.
Model Info