logistic-regression

November 18, 2024

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, roc_auc_score, roc_curve
```

[ ]:
```
data=pd.read_csv("/content/framingham.csv")
```

[ ]:
```
data.head(20)
```

[ ]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	\
0	1	39	4.0	0	0.0	0.0	0	
1	0	46	2.0	0	0.0	0.0	0	
2	1	48	1.0	1	20.0	0.0	0	
3	0	61	3.0	1	30.0	0.0	0	
4	0	46	3.0	1	23.0	0.0	0	
5	0	43	2.0	0	0.0	0.0	0	
6	0	63	1.0	0	0.0	0.0	0	
7	0	45	2.0	1	20.0	0.0	0	
8	1	52	1.0	0	0.0	0.0	0	
9	1	43	1.0	1	30.0	0.0	0	
10	0	50	1.0	0	0.0	0.0	0	
11	0	43	2.0	0	0.0	0.0	0	
12	1	46	1.0	1	15.0	0.0	0	
13	0	41	3.0	0	0.0	1.0	0	
14	0	39	2.0	1	9.0	0.0	0	
15	0	38	2.0	1	20.0	0.0	0	
16	1	48	3.0	1	10.0	0.0	0	
17	0	46	2.0	1	20.0	0.0	0	
18	0	38	2.0	1	5.0	0.0	0	
19	1	41	2.0	0	0.0	0.0	0	

	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	\

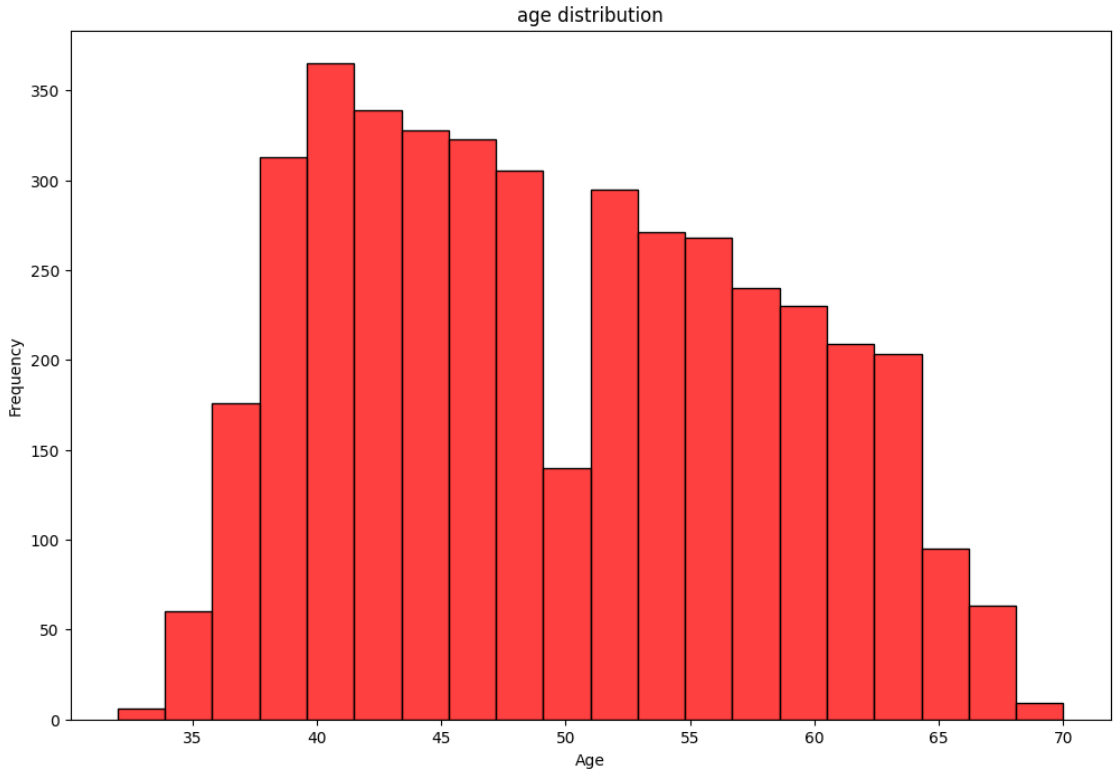
0	0	0	195.0	106.0	70.0	26.97	80.0	77.0
1	0	0	250.0	121.0	81.0	28.73	95.0	76.0
2	0	0	245.0	127.5	80.0	25.34	75.0	70.0
3	1	0	225.0	150.0	95.0	28.58	65.0	103.0
4	0	0	285.0	130.0	84.0	23.10	85.0	85.0
5	1	0	228.0	180.0	110.0	30.30	77.0	99.0
6	0	0	205.0	138.0	71.0	33.11	60.0	85.0
7	0	0	313.0	100.0	71.0	21.68	79.0	78.0
8	1	0	260.0	141.5	89.0	26.36	76.0	79.0
9	1	0	225.0	162.0	107.0	23.61	93.0	88.0
10	0	0	254.0	133.0	76.0	22.91	75.0	76.0
11	0	0	247.0	131.0	88.0	27.64	72.0	61.0
12	1	0	294.0	142.0	94.0	26.31	98.0	64.0
13	1	0	332.0	124.0	88.0	31.31	65.0	84.0
14	0	0	226.0	114.0	64.0	22.35	85.0	NaN
15	1	0	221.0	140.0	90.0	21.35	95.0	70.0
16	1	0	232.0	138.0	90.0	22.37	64.0	72.0
17	0	0	291.0	112.0	78.0	23.38	80.0	89.0
18	0	0	195.0	122.0	84.5	23.24	75.0	78.0
19	0	0	195.0	139.0	88.0	26.88	85.0	65.0

	TenYearCHD
0	0
1	0
2	0
3	1
4	0
5	0
6	1
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	1
18	0
19	0

[ ]:
```
data.fillna(0,inplace=True)
```

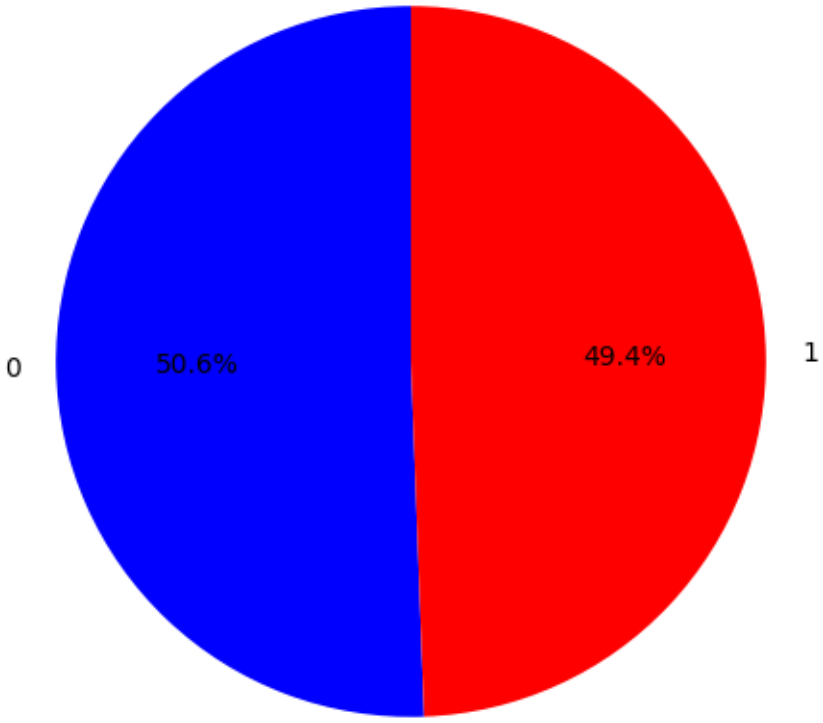
[ ]:
```
plt.figure(figsize=(12,8))
sns.histplot(data['age'],bins=20,color='red') #kde=True before bins
```

```
plt.title('age distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



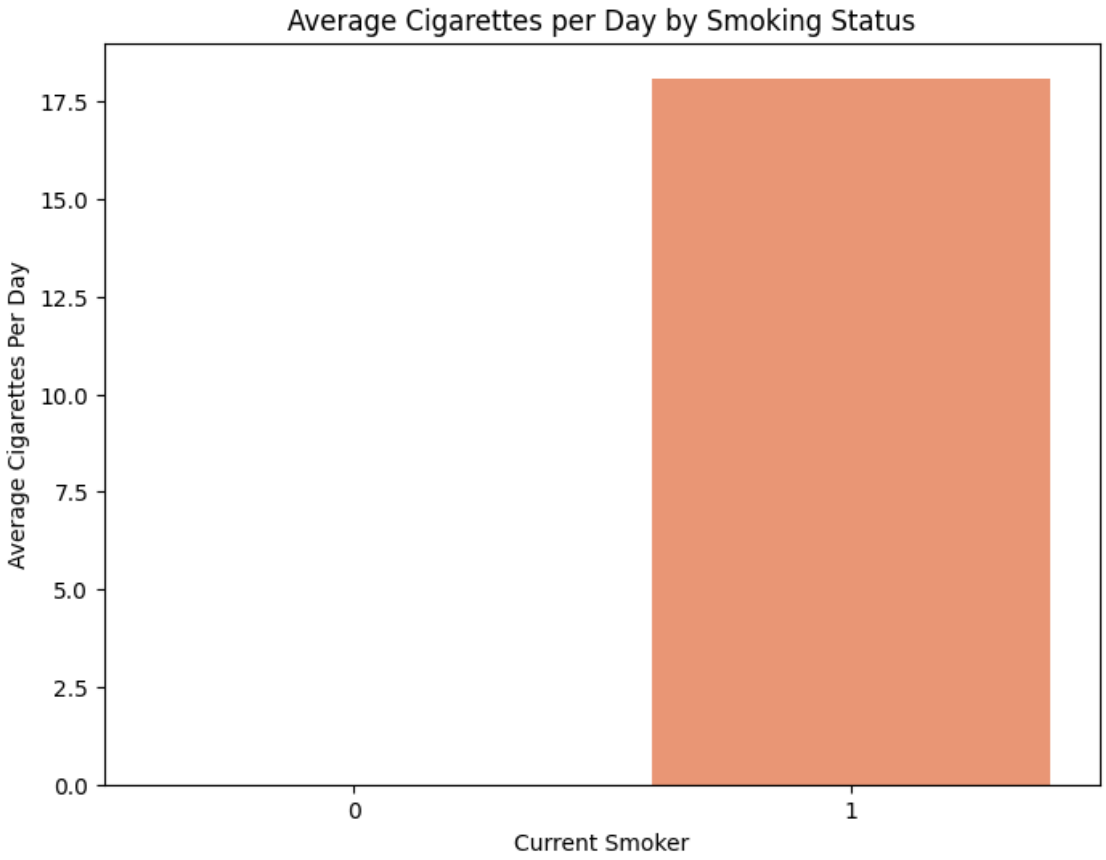
[ ]:
```
smoker_counts=data['currentSmoker'].value_counts()
plt.figure(figsize=(6,6))
plt.pie(smoker_counts,labels=smoker_counts.index,autopct='%1.
↪1f%%',colors=['blue','red'],startangle=90)
```

[ ]:
```
([<matplotlib.patches.Wedge at 0x79454b4ebe20>,
  <matplotlib.patches.Wedge at 0x79454b4eaf50>],
 [Text(-1.099811110169255, -0.020384355478431218, '0'),
  Text(1.099811110169255, 0.020384355478430597, '1')],
 [Text(-0.5998969691832299, -0.011118739351871573, '50.6%'),
  Text(0.5998969691832299, 0.011118739351871233, '49.4%')])
```

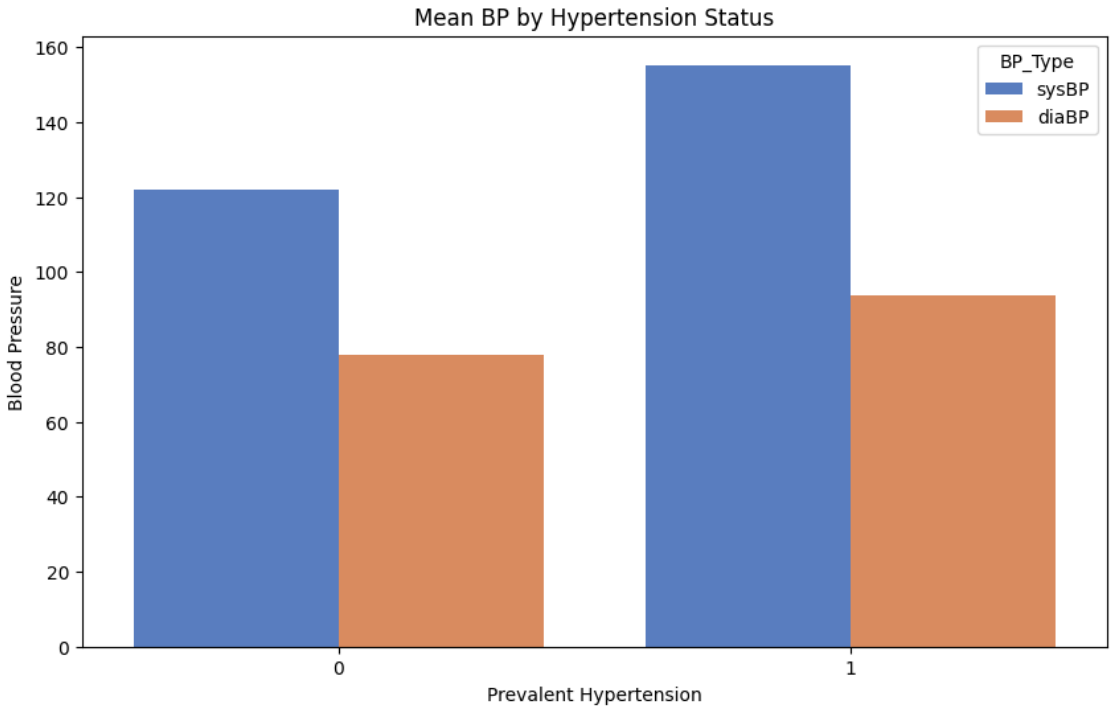


[ ]:
```
avg_cigs = data.groupby('currentSmoker')['cigsPerDay'].mean().reset_index()
plt.figure(figsize=(8, 6))
sns.barplot(data=avg_cigs, x='currentSmoker', y='cigsPerDay', palette='Set2')
plt.title('Average Cigarettes per Day by Smoking Status')
plt.xlabel('Current Smoker')
plt.ylabel('Average Cigarettes Per Day')
plt.show()
```

```
<ipython-input-69-f9f38b46d559>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=avg_cigs, x='currentSmoker', y='cigsPerDay', palette='Set2')
```

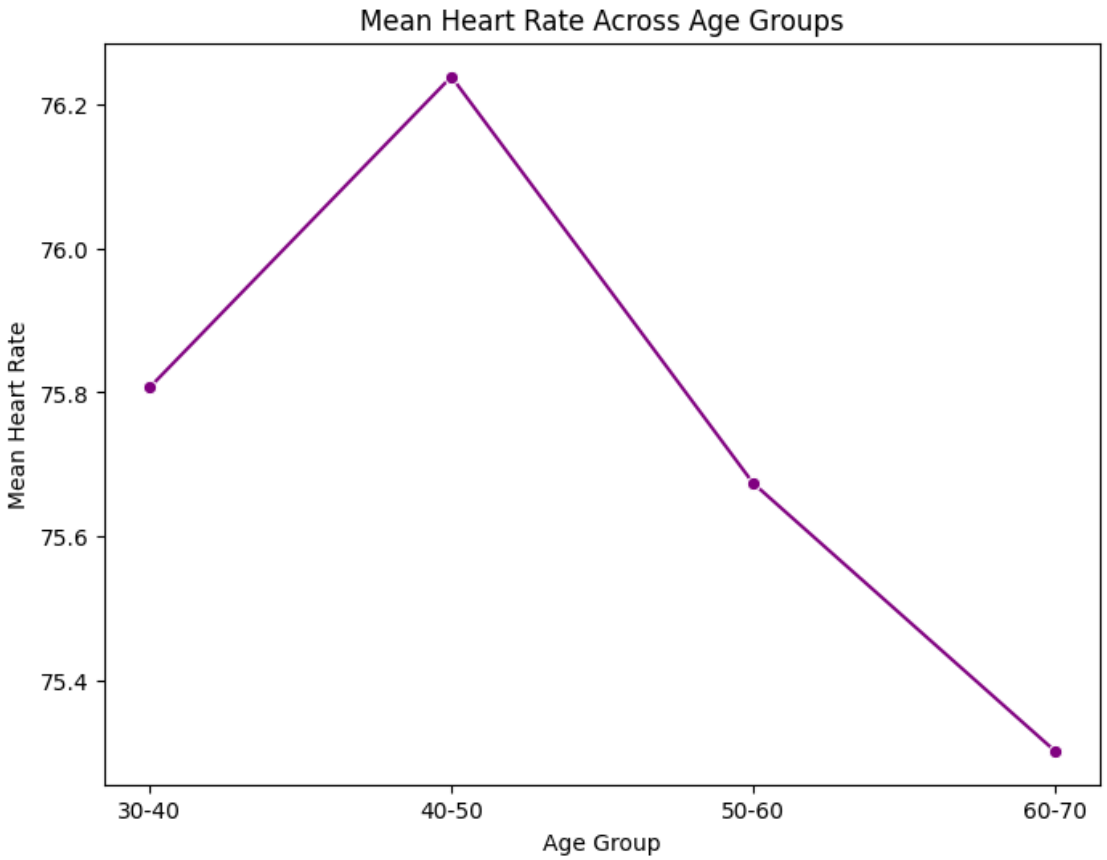


[ ]:
```
bp_means = data.groupby('prevalentHyp')[['sysBP', 'diaBP']].mean().reset_index()
bp_means_melted = bp_means.melt(id_vars='prevalentHyp', var_name='BP_Type',␣
 ↪value_name='BP_Value')
plt.figure(figsize=(10, 6))
sns.barplot(data=bp_means_melted, x='prevalentHyp', y='BP_Value',␣
 ↪hue='BP_Type', palette='muted')
plt.title('Mean BP by Hypertension Status')
plt.xlabel('Prevalent Hypertension')
plt.ylabel('Blood Pressure')
plt.show()
```



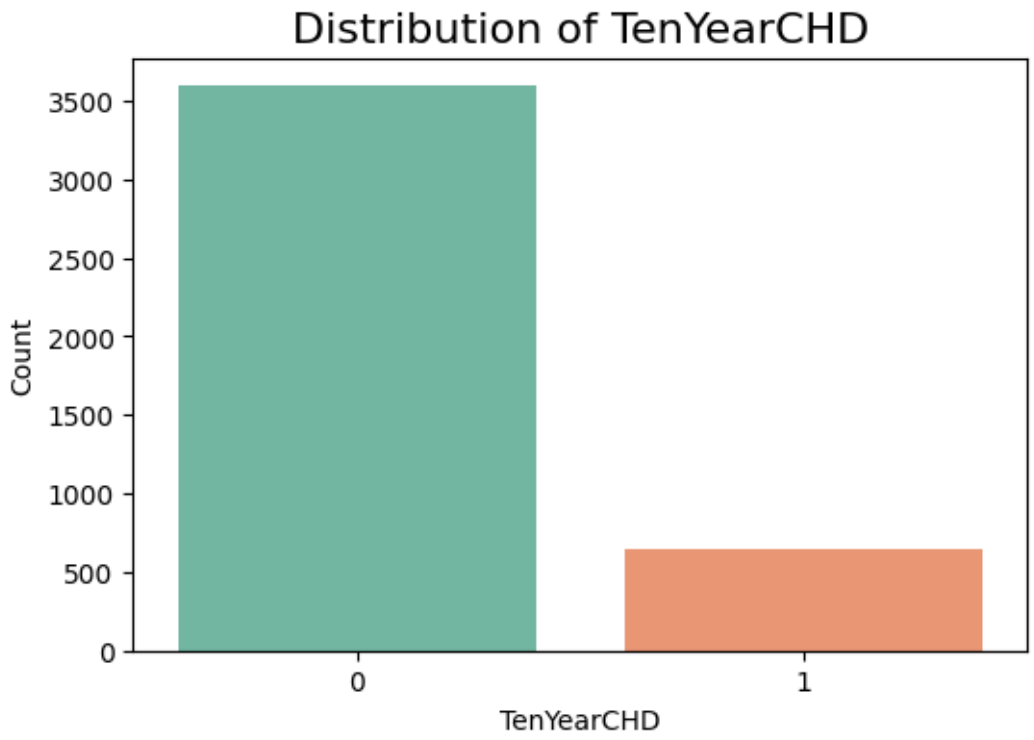
[ ]:
```
data['age_group'] = pd.cut(data['age'], bins=[30, 40, 50, 60, 70, 80],␣
 ↪labels=['30-40', '40-50', '50-60', '60-70', '70-80'])
mean_heart_rate = data.groupby('age_group')['heartRate'].mean().reset_index()
plt.figure(figsize=(8, 6))
sns.lineplot(data=mean_heart_rate, x='age_group', y='heartRate', marker='o',␣
 ↪color='purple')
plt.title('Mean Heart Rate Across Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Mean Heart Rate')
plt.show()
```

```
<ipython-input-71-aa077e30f61d>:2: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  mean_heart_rate = data.groupby('age_group')['heartRate'].mean().reset_index()
```

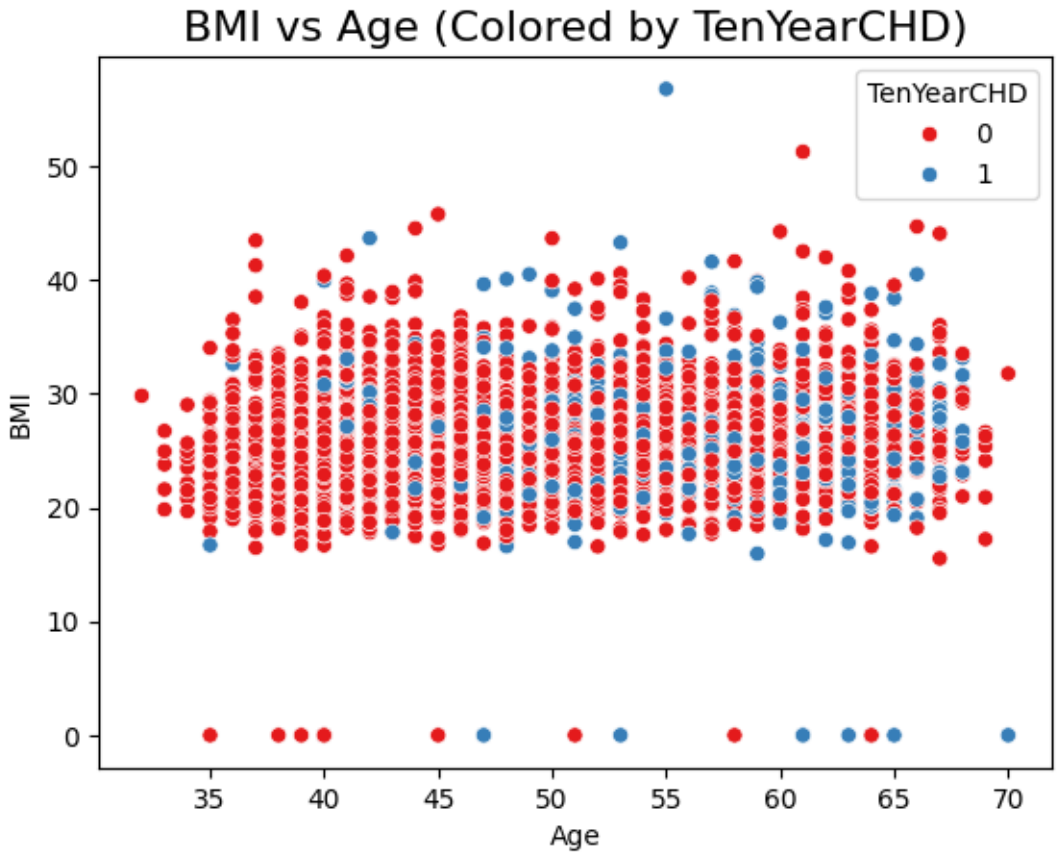


[ ]:
```
plt.figure(figsize=(6, 4))
sns.countplot(data=data, x='TenYearCHD', palette='Set2')
plt.title('Distribution of TenYearCHD', fontsize=16)
plt.xlabel('TenYearCHD')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-72-73342d59329d>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=data, x='TenYearCHD', palette='Set2')
```

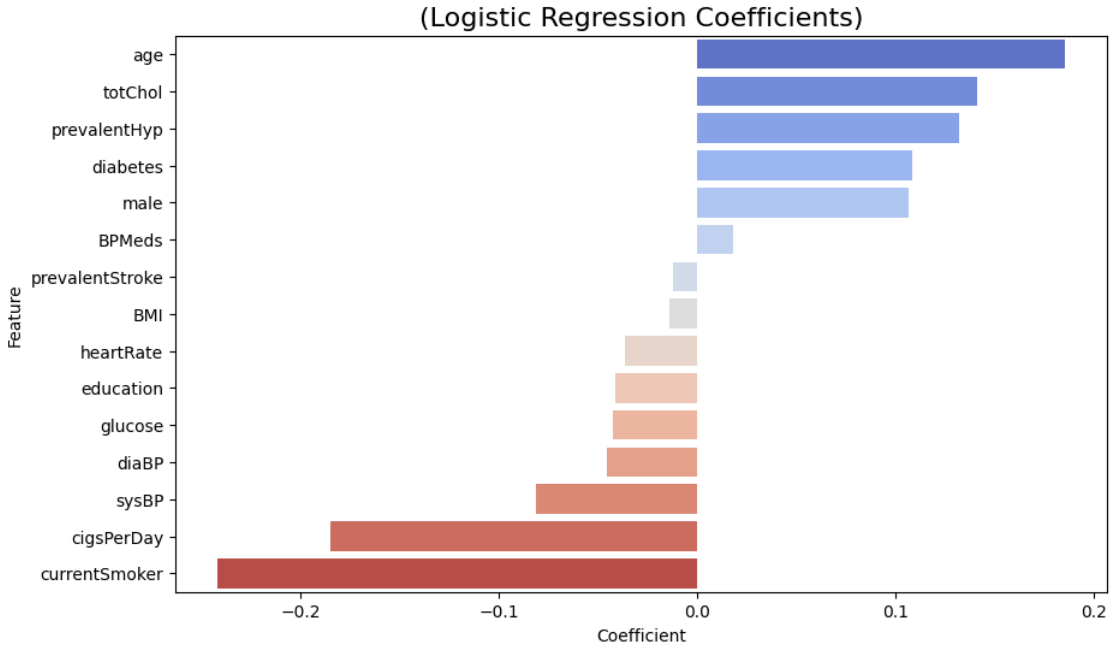


[ ]:
```
sns.scatterplot(data=data, x='age', y='BMI', hue='TenYearCHD', palette='Set1')
plt.title('BMI vs Age (Colored by TenYearCHD)', fontsize=16)
plt.xlabel('Age')
plt.ylabel('BMI')
plt.show()
```



[ ]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=importance, x='Coefficient', y='Feature', palette='coolwarm')
plt.title('(Logistic Regression Coefficients)', fontsize=16)
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.show()
```

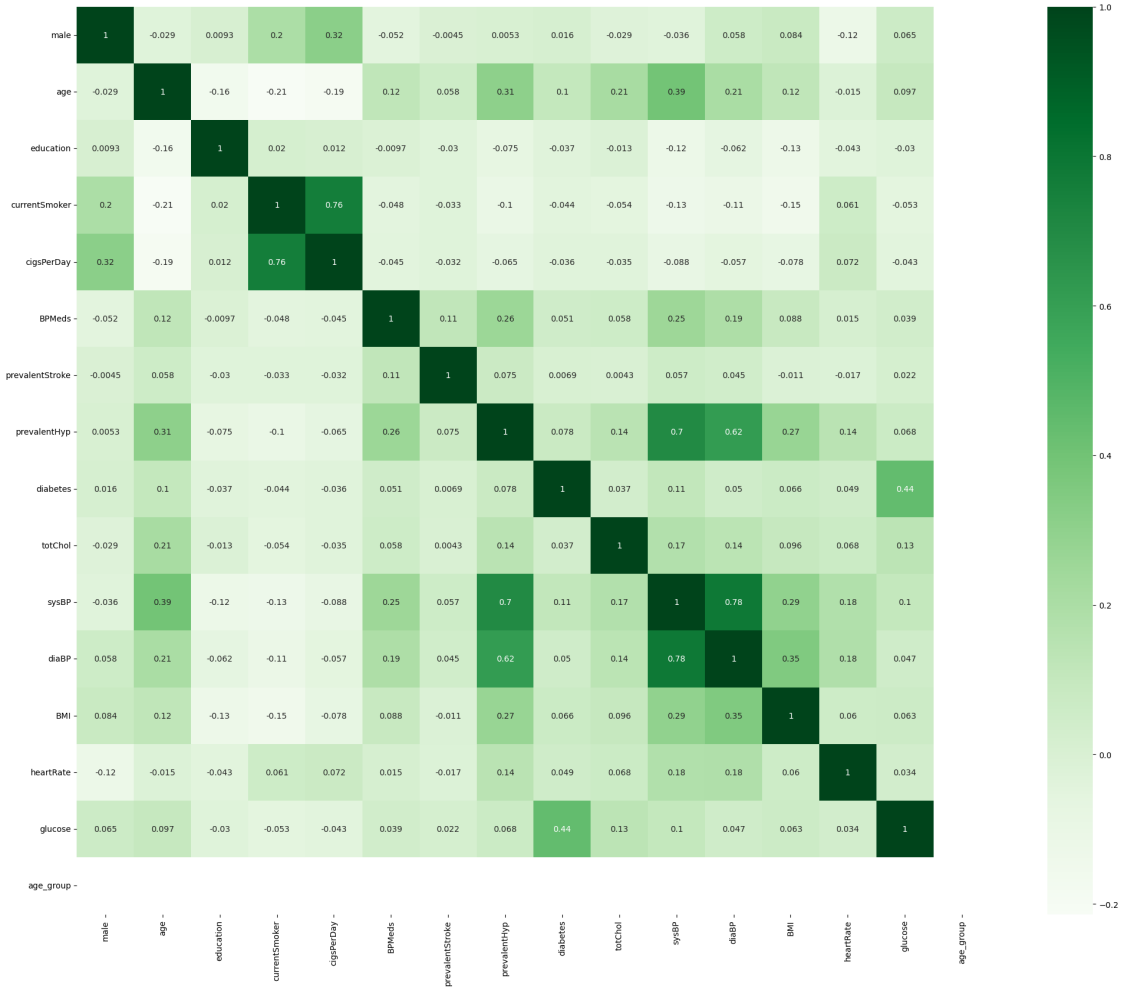
```
<ipython-input-74-29ae9def8f5d>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=importance, x='Coefficient', y='Feature', palette='coolwarm')
```



[ ]:
```
data = data.apply(pd.to_numeric, errors='coerce')
```

[ ]:
```
data = data.fillna(0)
```

[ ]:
```
corr = data.drop(columns= 'TenYearCHD').corr()
fig , ax = plt.subplots(figsize=(25 , 20))
sns.heatmap(corr ,annot= True , ax=ax , cmap= 'Greens');
```



[ ]:
```
X = data.drop(columns=['TenYearCHD'])
target = data['TenYearCHD']

X_train , X_test , y_train , y_test = train_test_split(X ,target ,test_size=0.
↪35 , random_state=35 )
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (2754, 16)
y_train shape: (2754,)
X_test shape: (1484, 16)
y_test shape: (1484,)
```

[ ]:
```
from sklearn.dummy import DummyClassifier
dummy_classifier = DummyClassifier(strategy = 'most_frequent')
dummy_classifier.fit(X_train, y_train)
y_pred = dummy_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Baseline Model Accuracy: {accuracy:.4f}")
```

```
Baseline Model Accuracy: 0.8511
```

[ ]:
```
X = data.drop("TenYearCHD", axis=1)
y = data["TenYearCHD"]
```

[91]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=35)
```

[92]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[96]:
```
model = LogisticRegression(random_state=35)
model.fit(X_train, y_train)
```

[96]:
```
LogisticRegression(random_state=35)
```

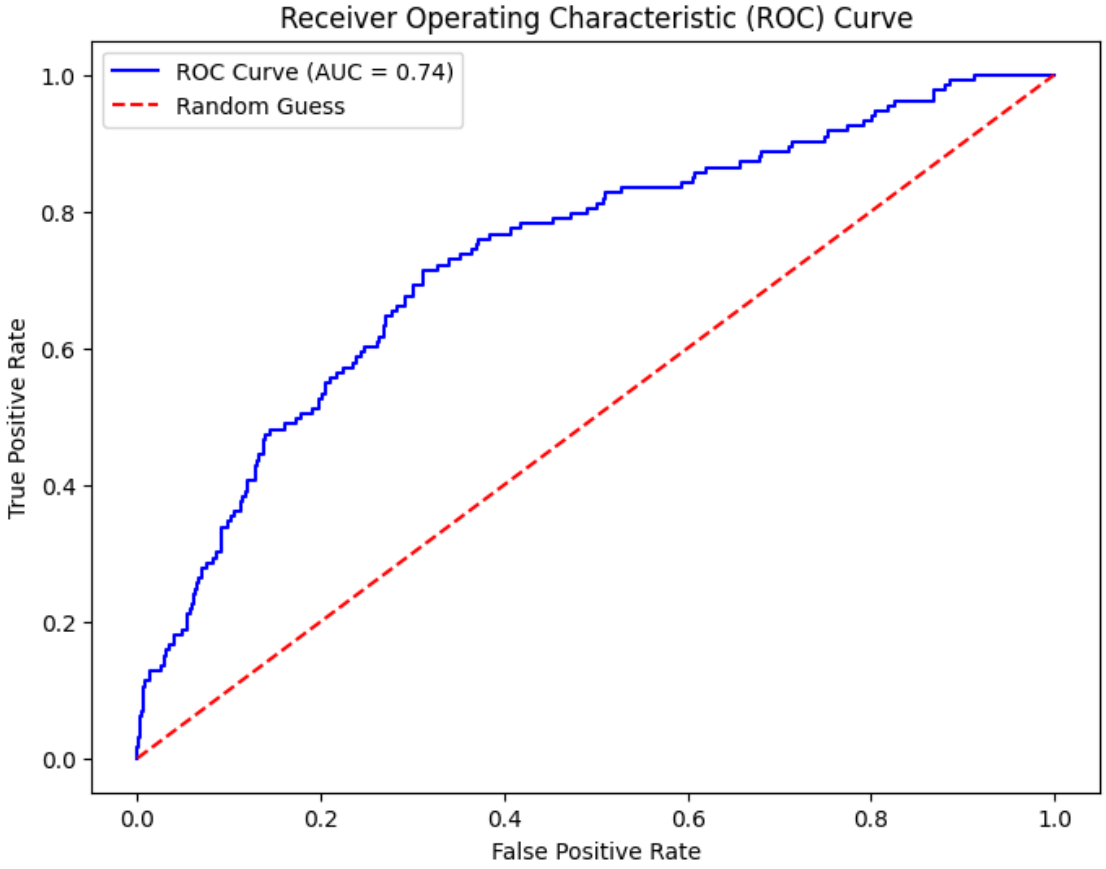
[97]:
```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)
```

[98]:
```
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)
print("ROC AUC Score:", roc_auc)
```

```
Accuracy: 0.8537735849056604

Confusion Matrix:
 [[709   6]
 [118  15]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       715
           1       0.71      0.11      0.19       133
```

```
    accuracy                           0.85       848
   macro avg       0.79      0.55      0.56       848
weighted avg       0.83      0.85      0.81       848

ROC AUC Score: 0.7354960828645041
```

[99]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Guess')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



[100]:
```
m1 = 'Logistic Regression'
lr = LogisticRegression()
model = lr.fit(X_train, y_train)
lr_predict = lr.predict(X_test)
```

```
lr_conf_matrix = confusion_matrix(y_test, lr_predict)
lr_acc_score = accuracy_score(y_test, lr_predict)
print("confussion matrix")
print(lr_conf_matrix)
print("\n")
print("Accuracy of Logistic Regression:",lr_acc_score*100,'\n')
print(classification_report(y_test,lr_predict))
```

```
confussion matrix
[[709   6]
 [118  15]]


Accuracy of Logistic Regression: 85.37735849056604

              precision    recall  f1-score   support

           0       0.86      0.99      0.92       715
           1       0.71      0.11      0.19       133

    accuracy                           0.85       848
   macro avg       0.79      0.55      0.56       848
weighted avg       0.83      0.85      0.81       848
```

[101]:
```
from sklearn.linear_model import LogisticRegression
LogisticRegressionModel = LogisticRegression(penalty='l2',solver='sag',C=1.
  ↪0,random_state=33)
LogisticRegressionModel.fit(X_train, y_train)
y_pred = LogisticRegressionModel.predict(X_test)
```

[102]:
```
print('LogisticRegressionModel Train Score is : ' , LogisticRegressionModel.
  ↪score(X_train, y_train))
print('LogisticRegressionModel Test Score is : ' , LogisticRegressionModel.
  ↪score(X_test, y_test))
```

```
LogisticRegressionModel Train Score is :  0.8533923303834808
LogisticRegressionModel Test Score is :  0.8525943396226415
```

[ ]: