```
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, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv("/content/heartdata.csv")
```

```
df.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1
4	54	1	3	150	195	0	0	122	0	0.0	1	0

```
df.shape
```

```
(1190, 12)
```

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

```
np.int64(272)
```

```
df.drop_duplicates(inplace=True)
```

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

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

```
df.shape
```

```
(918, 12)
```

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

	0
age	0
sex	0
chest pain type	0
resting bp s	0
cholesterol	0
fasting blood sugar	0
resting ecg	0
max heart rate	0
exercise angina	0
oldpeak	0
ST slope	0
target	0

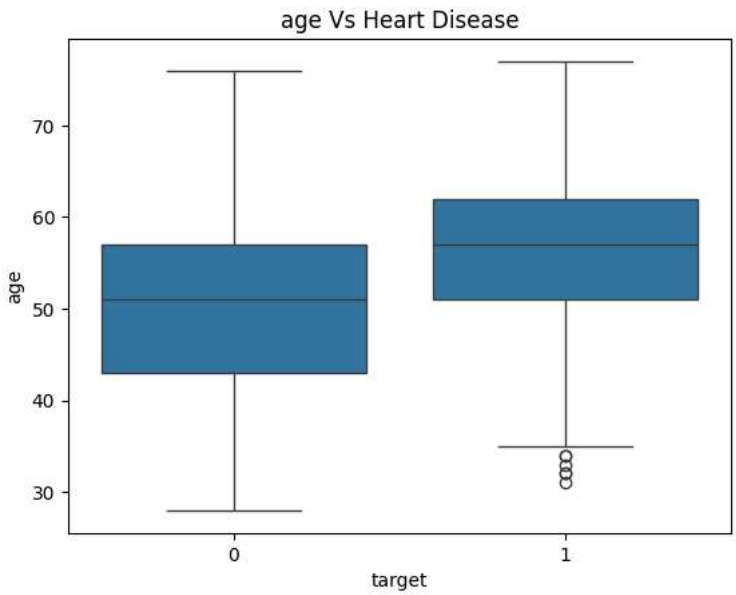
dtype: int64

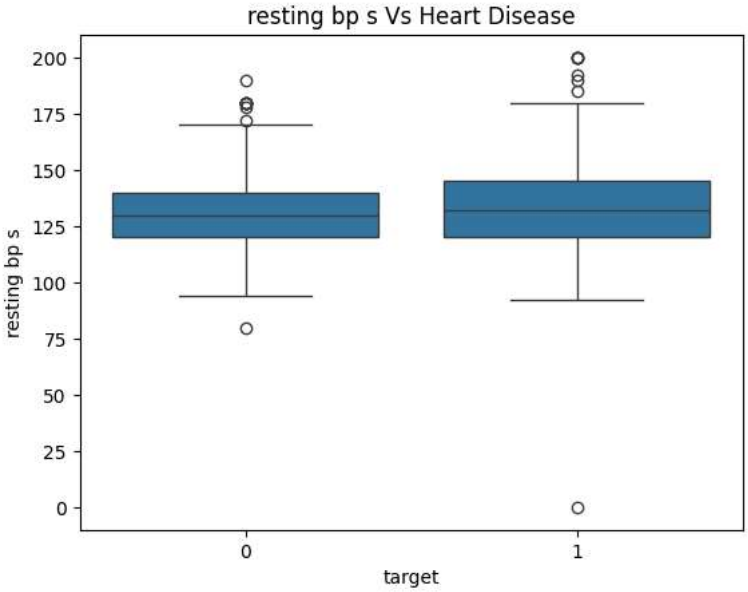
```
print('=====Describe=====')
print(df.describe())
print('=====Information=====')
print(df.info())
print('=====Corelation Matrix=====')
corr_matrix=df.corr()
print(corr_matrix['target'].sort_values(ascending=False))
```

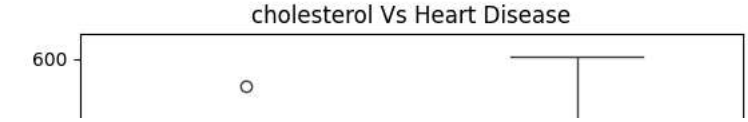
```
=====Describe=====
                age         sex  chest pain type  resting bp s  cholesterol  \
count  918.000000  918.000000       918.000000    918.000000   918.000000
mean    53.510893    0.789760         3.251634    132.396514   198.799564
std      9.432617    0.407701         0.931031     18.514154   109.384145
min     28.000000    0.000000         1.000000      0.000000     0.000000
25%     47.000000    1.000000         3.000000    120.000000   173.250000
50%     54.000000    1.000000         4.000000    130.000000   223.000000
75%     60.000000    1.000000         4.000000    140.000000   267.000000
max     77.000000    1.000000         4.000000    200.000000   603.000000

       fasting blood sugar  resting ecg  max heart rate  exercise angina  \
count           918.000000   918.000000      918.000000       918.000000
mean              0.233115     0.603486      136.809368         0.404139
std               0.423046     0.805968       25.460334         0.490992
min               0.000000     0.000000       60.000000         0.000000
25%               0.000000     0.000000      120.000000         0.000000
50%               0.000000     0.000000      138.000000         0.000000
75%               0.000000     1.000000      156.000000         1.000000
max               1.000000     2.000000      202.000000         1.000000

          oldpeak    ST slope      target
count  918.000000  918.000000  918.000000
mean     0.887364    1.636166    0.553377
std      1.066570    0.609341    0.497414
min     -2.600000    0.000000    0.000000
25%      0.000000    1.000000    0.000000
50%      0.600000    2.000000    1.000000
75%      1.500000    2.000000    1.000000
max      6.200000    3.000000    1.000000
=====Information=====
<class 'pandas.core.frame.DataFrame'>
Index: 918 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  918 non-null    int64
 1   sex                  918 non-null    int64
 2   chest pain type      918 non-null    int64
 3   resting bp s         918 non-null    int64
 4   cholesterol          918 non-null    int64
 5   fasting blood sugar  918 non-null    int64
 6   resting ecg          918 non-null    int64
 7   max heart rate       918 non-null    int64
 8   exercise angina      918 non-null    int64
 9   oldpeak              918 non-null    float64
 10  ST slope             918 non-null    int64
 11  target               918 non-null    int64
```

```
dtypes: float64(1), int64(11)
memory usage: 93.2 KB
None
=====Corelation Matrix=====
target                 1.000000
ST slope               0.553461
exercise angina        0.494282
chest pain type        0.471354
oldpeak                0.403951
sex                    0.305445
```

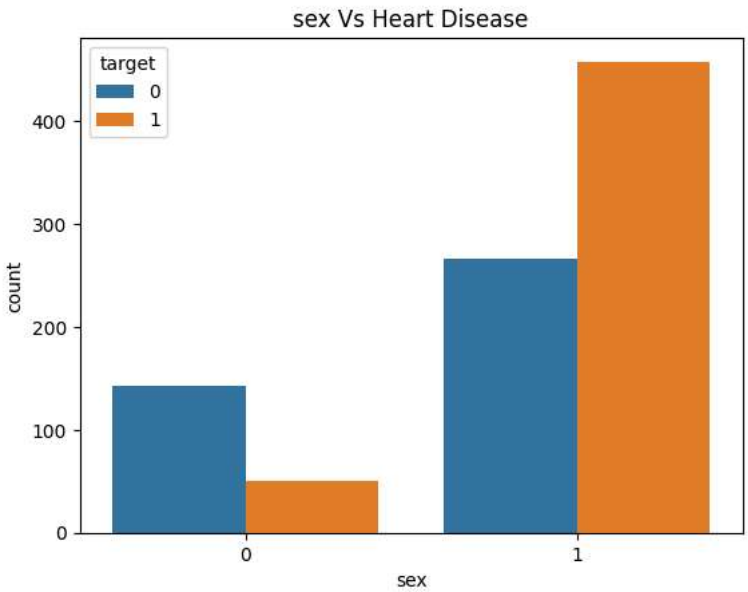
```
numerical_features=['age', 'resting bp s',  'cholesterol', 'max heart rate', 'oldpeak']
categorical_features=['sex','chest pain type','fasting blood sugar','resting ecg','exercise angina',    'ST slope']
#Numerical Feature analysis
for feature in numerical_features:
  sns.boxplot(x='target', y=feature, data=df)
  plt.title(f'{feature} Vs Heart Disease')
  plt.show()
```

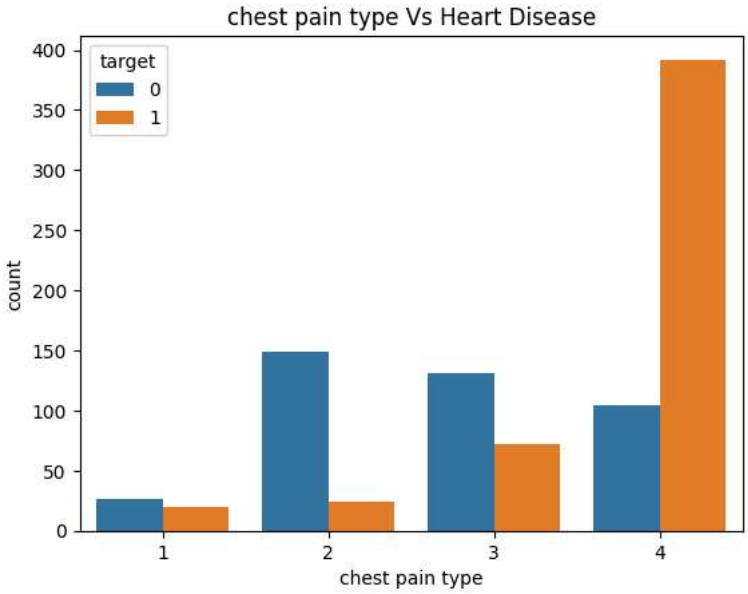


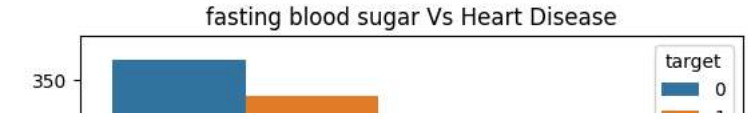





```
#for categorical feature
for feature in categorical_features:
  sns.countplot(x=feature,hue='target',data=df)
  plt.title(f'{feature} Vs Heart Disease')
  plt.show()
```





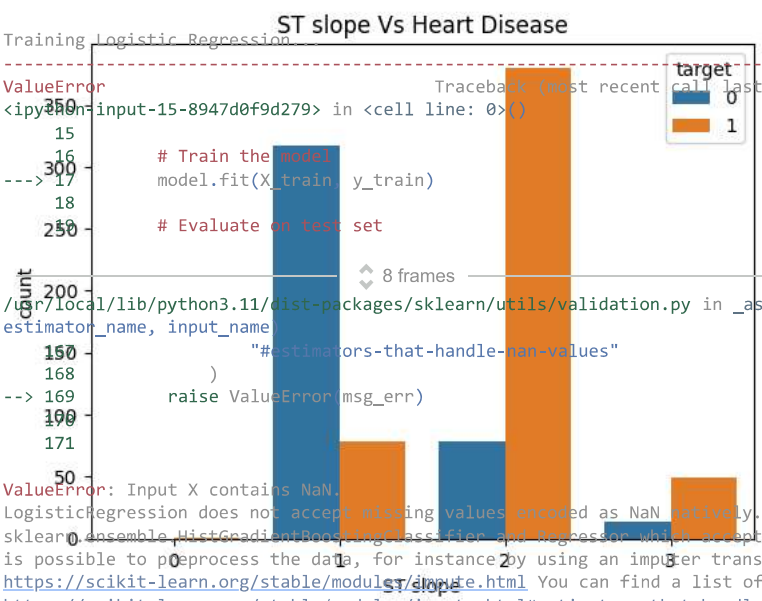



```
df['chest pain type']=df['chest pain type']-1
chest_pain_dummies=pd.get_dummies(df['chest pain type'],prefix='chest pain type')
df=pd.concat([df,chest_pain_dummies],axis=1)
df['ST slope']=df['ST slope']-1
st_slope_dummies = pd.get_dummies(df['ST slope'], prefix='ST slope')
df = pd.concat([df, st_slope_dummies], axis=1)
resting_ecg_dummies = pd.get_dummies(df['resting ecg'], prefix='resting ecg')
df = pd.concat([df, resting_ecg_dummies], axis=1)

df['rate pressure product'] = (df['resting bp s'] * df['max heart rate']) / 100
df['heart rate reserve'] = 220 - df['age'] - df['max heart rate']

df['bp category'] = pd.cut(df['resting bp s'], bins=[0, 120, 140, 160, 200], labels=[0, 1, 2, 3])
df['chol category'] = pd.cut(df['cholesterol'], bins=[0, 200, 240, 300, 600], labels=[0, 1, 2, 3])

df.drop(['chest pain type', 'ST slope', 'resting ecg'], axis=1, inplace=True)
```

```
X=df.drop('target',axis=1)
y=df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
```

```
X_train shape: (734, 13)
X_test shape: (184, 13)
```

```
models = {
        'Logistic Regression': Pipeline([
            ('scaler', StandardScaler()),
            ('classifier', LogisticRegression(max_iter=1000, random_state=42))
        ]),
        'Random Forest': Pipeline([
            ('classifier', RandomForestClassifier(random_state=42))
        ]),
        'Gradient Boosting': Pipeline([
            ('classifier', GradientBoostingClassifier(random_state=42))
        ])
    }
for name, model in models.items():
        print(f"\nTraining {name}...")

        # Train the model
        model.fit(X_train, y_train)

        # Evaluate on test set
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)

        # Cross-validation
        cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
        print(f"{name} Test Accuracy: {accuracy:.4f}")
        print(f"{name} Cross-Validation Scores: {cv_scores}")
        print(f"{name} Mean CV Score: {cv_scores.mean():.4f}")
        print(f"Classification Report:\n{classification_report(y_test, y_pred)}")
        print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")

        # ROC Curve
        if hasattr(model, "predict_proba"):
            y_prob = model.predict_proba(X_test)[:, 1]
            fpr, tpr, _ = roc_curve(y_test, y_prob)
            roc_auc = auc(fpr, tpr)
            print(f"AUC-ROC: {roc_auc:.4f}")
```




```
Training Logistic Regression...
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-8947d0f9d279> in <cell line: 0>()
     15
     16         # Train the model
---> 17         model.fit(X_train, y_train)
     18
     19         # Evaluate on test set

                              8 frames
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name, input_name)
    150                 "#estimators-that-handle-nan-values"
    168             )
--> 169         raise ValueError(msg_err)
    170
    171

ValueError: Input X contains NaN.
LogisticRegression does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider sklearn.ensemble.HistGradientBoostingClassifier and Regressor which accept missing values encoded as NaNs natively. Alternatively, it is possible to preprocess the data, for instance by using an imputer transformer in a pipeline or drop samples with missing values. See https://scikit-learn.org/stable/modules/impute.html You can find a list of all estimators that handle NaN values at the following page: https://scikit-learn.org/stable/modules/impute.html#estimators-that-handle-nan-values
```

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

	0
age	0
sex	0