# National Institute of Technology Rourkela, Odisha, India, 769008

## Department of Computer Science Engineering

### Laborotary-4

**(Data Science Laborotary)**

# Linear Regression

Importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
C:\Users\Laptop\AppData\Local\Temp\ipykernel_21056\2080034654.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas
(pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interope
rability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

Read the salary dataset

In [2]:
```
# your answer here
dataSet_salary = pd.read_csv('Salary_Data.csv')
```

Show the first 10 rows of the dataset

In [3]:
```
# your answer here
dataSet_salary.head(10)
```

Out[3]:

| | YearsExperience | Salary |
|---|---|---|
| **0** | 1.1 | 39343.0 |
| **1** | 1.3 | 46205.0 |
| **2** | 1.5 | 37731.0 |
| **3** | 2.0 | 43525.0 |
| **4** | 2.2 | 39891.0 |
| **5** | 2.9 | 56642.0 |

| | YearsExperience | Salary |
|---|---|---|
| **6** | 3.0 | 60150.0 |
| **7** | 3.2 | 54445.0 |
| **8** | 3.2 | 64445.0 |
| **9** | 3.7 | 57189.0 |

Show the dimensions (No. of rows and coulmns) of the dataset

In [4]:
```
# your answer here
dataSet_salary.shape
```

Out[4]:
```
(30, 2)
```

Print all the column names of the dataset

In [5]:
```
# your answer here
dataSet_salary.columns
```

Out[5]:
```
Index(['YearsExperience', 'Salary'], dtype='object')
```

Print general information of the dataset like column, and datatype.

In [6]:
```
# your answer here
dataSet_salary.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 612.0 bytes
```

Extract independent and dependent features and store it in two different variables.

In [7]:
```
# your answer here
yearsExperience = dataSet_salary['YearsExperience']
salary = dataSet_salary['Salary']
```

Split the dataset into train and test set

In [8]:
```
from sklearn.model_selection import train_test_split
```

In [9]:
```
# your answer here

X_train, X_test, y_train, y_test = train_test_split(yearsExperience, salary, test_size=
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(24,) (6,) (24,) (6,)
```

Training the Simple Linear Regression model on the Training set

In [10]:
```
from sklearn.linear_model import LinearRegression
```

In [11]:
```
# your answer here
model = LinearRegression()
model.fit(X_train.values.reshape(-1,1), y_train)
```

Out[11]:
```
▾ LinearRegression

LinearRegression()
```

Predict the Test set results

In [12]:
```
# your answer here
predictions = model.predict(X_test.values.reshape(-1,1))
```

Visualize the linear regression on training data using scatterplot.

In [13]:
```
# your answer here
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```


Visualize the linear regression on test data using scatterplot.

In [14]:

```
# your answer here
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```


Finding R^2 score

In [15]:
```
from sklearn.metrics import r2_score
```

In [16]:
```
# your answer here
r2_score(y_test, predictions)
```

Out[16]:
```
0.988169515729126
```

# Ridge Regression

In [17]:
```
from sklearn.linear_model import Ridge
```

In [18]:
```
model = Ridge(alpha=0.5)
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print(r2_score(y_test, predictions))
```

```
0.987891303817413
```

In [19]:
```
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
```

```
plt.ylabel('Salary')
plt.show()
```


# Huber

In [20]:

```
from sklearn.linear_model import HuberRegressor
# outlier
```

In [21]:

```
model = HuberRegressor()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print(r2_score(y_test, predictions))
```

```
0.9870632883295445
```

In [22]:

```
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```


# ElasticNet

In [23]:
```
from sklearn.linear_model import ElasticNet
```

In [24]:
```
model = ElasticNet()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print(r2_score(y_test, predictions))
```

```
0.9772686017240042
```

In [25]:
```
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```


# Lasso

In [26]:
```
from sklearn.linear_model import Lasso
```

In [27]:
```
model = Lasso()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print(r2_score(y_test, predictions))
```

```
0.988168127365881
```

In [28]:
```
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```


# Logistic Regression

Import Libraries

In [29]:
```
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.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Read the heart failure dataset

In [30]:
```
# your answer here
dataSet_heart = pd.read_csv('heart.csv')
```

Display the first five rows

In [31]:
```
# your answer here
dataSet_heart.head(5)
```

Out[31]:

| | Age | Sex | ChestPainType | RestingBP | Cholesterol | FastingBS | RestingECG | MaxHR | ExerciseAngina | Ol |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 40 | M | ATA | 140 | 289 | 0 | Normal | 172 | N | |

| | Age | Sex | ChestPainType | RestingBP | Cholesterol | FastingBS | RestingECG | MaxHR | ExerciseAngina | Ol |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 49 | F | NAP | 160 | 180 | 0 | Normal | 156 | N | |
| **2** | 37 | M | ATA | 130 | 283 | 0 | ST | 98 | N | |
| **3** | 48 | F | ASY | 138 | 214 | 0 | Normal | 108 | Y | |
| **4** | 54 | M | NAP | 150 | 195 | 0 | Normal | 122 | N | |

Check for missing values

In [32]:

```
# your answer here
empty_count = {}
for column in dataSet_heart.columns:
    empty_count[column] = dataSet_heart[column].isnull().sum()

has_empty = False
for key, value in empty_count.items():
    if value != 0:
        has_empty = True
        print(key, value)

if not has_empty:
    print('No empty data')
```

```
No empty data
```

Describe numerical features

In [33]:

```
# your answer here
dataSet_heart.describe()
```

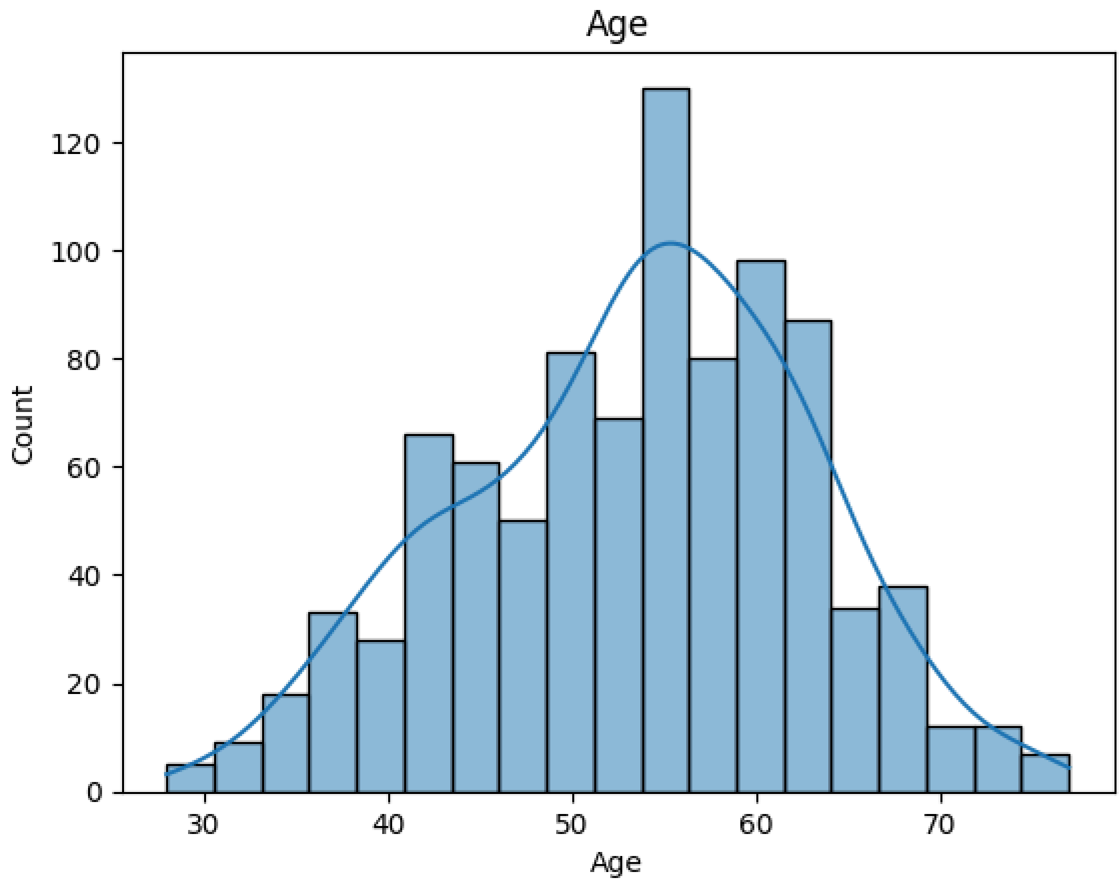
Out[33]:

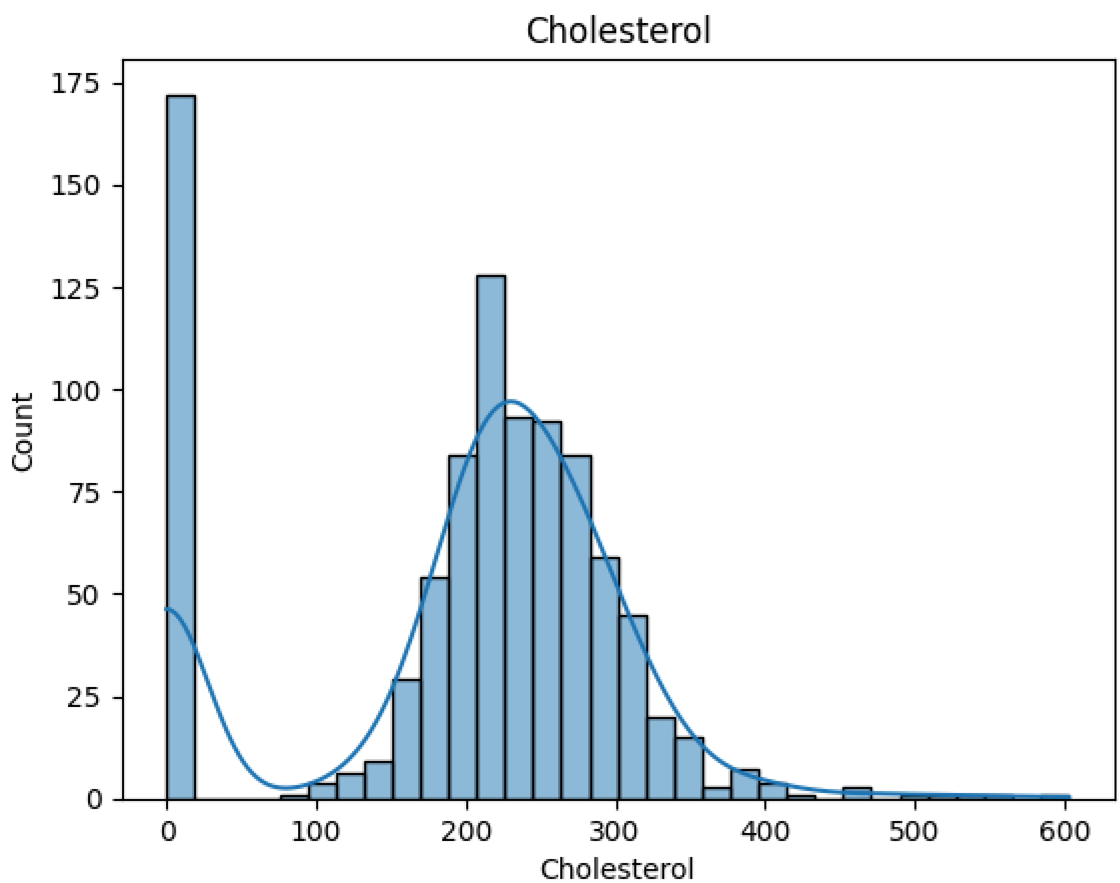
| | Age | RestingBP | Cholesterol | FastingBS | MaxHR | Oldpeak | HeartDisease |
|---|---|---|---|---|---|---|---|
| **count** | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 918.000000 |
| **mean** | 53.510893 | 132.396514 | 198.799564 | 0.233115 | 136.809368 | 0.887364 | 0.553377 |
| **std** | 9.432617 | 18.514154 | 109.384145 | 0.423046 | 25.460334 | 1.066570 | 0.497414 |
| **min** | 28.000000 | 0.000000 | 0.000000 | 0.000000 | 60.000000 | -2.600000 | 0.000000 |
| **25%** | 47.000000 | 120.000000 | 173.250000 | 0.000000 | 120.000000 | 0.000000 | 0.000000 |
| **50%** | 54.000000 | 130.000000 | 223.000000 | 0.000000 | 138.000000 | 0.600000 | 1.000000 |
| **75%** | 60.000000 | 140.000000 | 267.000000 | 0.000000 | 156.000000 | 1.500000 | 1.000000 |
| **max** | 77.000000 | 200.000000 | 603.000000 | 1.000000 | 202.000000 | 6.200000 | 1.000000 |

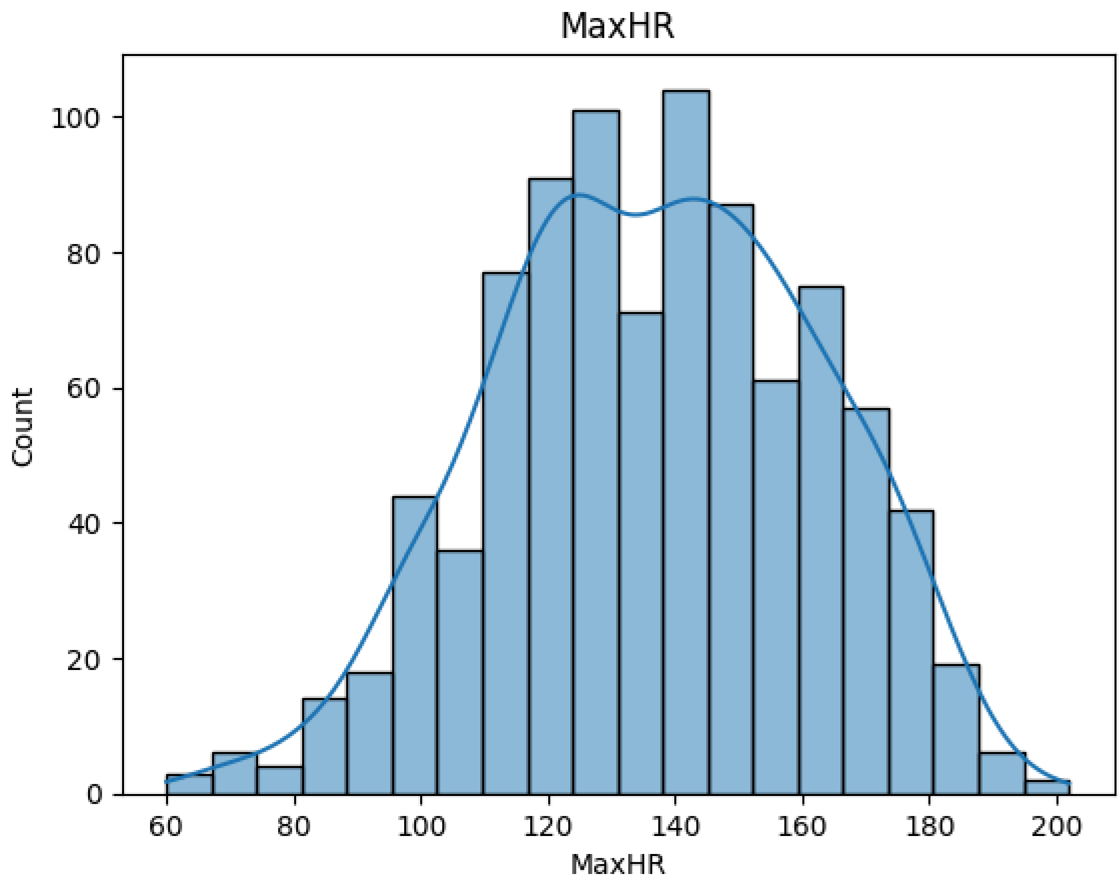
Visualize the distribution of key features (Age, Cholesterol, MaxHR) using histograms.

In [34]:

```
# your answer here
for col in ['Age', 'Cholesterol', 'MaxHR']:
    sns.histplot(dataSet_heart[col], kde=True)
    plt.title(col)
    plt.show()
```


Age
Count
Age
0
20
40
60
80
100
120
30
40
50
60
70


Cholesterol
Count
Cholesterol
0
25
50
75
100
125
150
175
0
100
200
300
400
500
600




List all categorical_features

In [35]:

```
# your answer here
for col in dataSet_heart.columns:
    if dataSet_heart[col].dtype == 'object':
        print(col)
        print(dataSet_heart[col].unique())
        print("count: ", len(dataSet_heart[col].unique()))
```

```
Sex
['M' 'F']
count:  2
ChestPainType
['ATA' 'NAP' 'ASY' 'TA']
count:  4
RestingECG
['Normal' 'ST' 'LVH']
count:  3
ExerciseAngina
['N' 'Y']
count:  2
ST_Slope
['Up' 'Flat' 'Down']
count:  3
```

Convert categorical variables into numerical format using label encoding.

In [36]:

```
# your answer here
label = {}
for col in dataSet_heart.columns:
```

```
    ind = 0
    if dataSet_heart[col].dtype == 'object':
        for val in dataSet_heart[col].unique():
            if col not in label:
                label[col] = {}
            label[col][val] = ind
            ind += 1

for col in label:
    print(label[col])

dataSet_label = dataSet_heart.copy()
for col in label:
    dataSet_label[f"{col}_label"] = dataSet_label[col].map(label[col])
    dataSet_label = dataSet_label.drop(col, axis=1)
    dataSet_label = dataSet_label.rename(columns={f"{col}_label": col})

dataSet_label.head(5)
```

```
{'M': 0, 'F': 1}
{'ATA': 0, 'NAP': 1, 'ASY': 2, 'TA': 3}
{'Normal': 0, 'ST': 1, 'LVH': 2}
{'N': 0, 'Y': 1}
{'Up': 0, 'Flat': 1, 'Down': 2}
```

Out[36]:

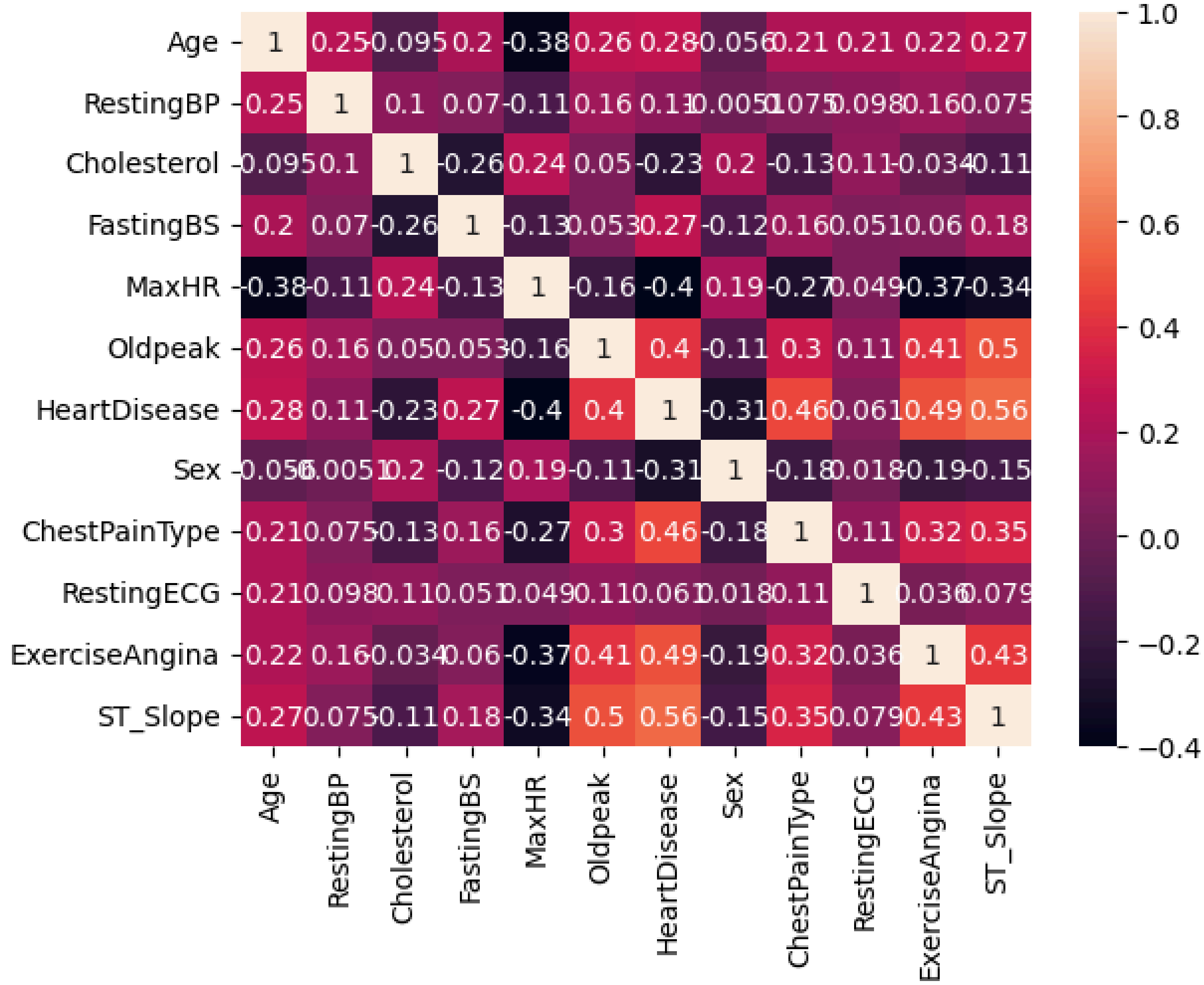
| | Age | RestingBP | Cholesterol | FastingBS | MaxHR | Oldpeak | HeartDisease | Sex | ChestPainType | Restinç |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 40 | 140 | 289 | 0 | 172 | 0.0 | 0 | 0 | 0 | |
| **1** | 49 | 160 | 180 | 0 | 156 | 1.0 | 1 | 1 | 1 | |
| **2** | 37 | 130 | 283 | 0 | 98 | 0.0 | 0 | 0 | 0 | |
| **3** | 48 | 138 | 214 | 0 | 108 | 1.5 | 1 | 1 | 2 | |
| **4** | 54 | 150 | 195 | 0 | 122 | 0.0 | 0 | 0 | 1 | |

Analyze the correlation between features using a heatmap.

In [37]:

```
# your answer here
sns.heatmap(dataSet_label.corr(), annot=True)
```

Out[37]: <Axes: >



Split the dataset into training and testing sets (80-20 split).

In [38]:
```
# your answer here
dataSet = dataSet_label.copy()
X_train, X_test, y_train, y_test = train_test_split(dataSet.drop('HeartDisease', axis=1
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(734, 11) (184, 11) (734,) (184,)
```

Perform hyperparameter tuning on logistic regression using GridSearchCV to find the best parameters

In [39]:
```
# your answer here
parameters = {'C': [0.1, 1, 5, 10, 20, 100], 'penalty': ['l1', 'l2'], "solver": ['libli
model = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(model, parameters)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.8610287950796757
```

Train the logistic regression model using the best parameters obtained from GridSearchCV and evaluate its performance on the test set using accuracy, confusion matrix, and classification report.

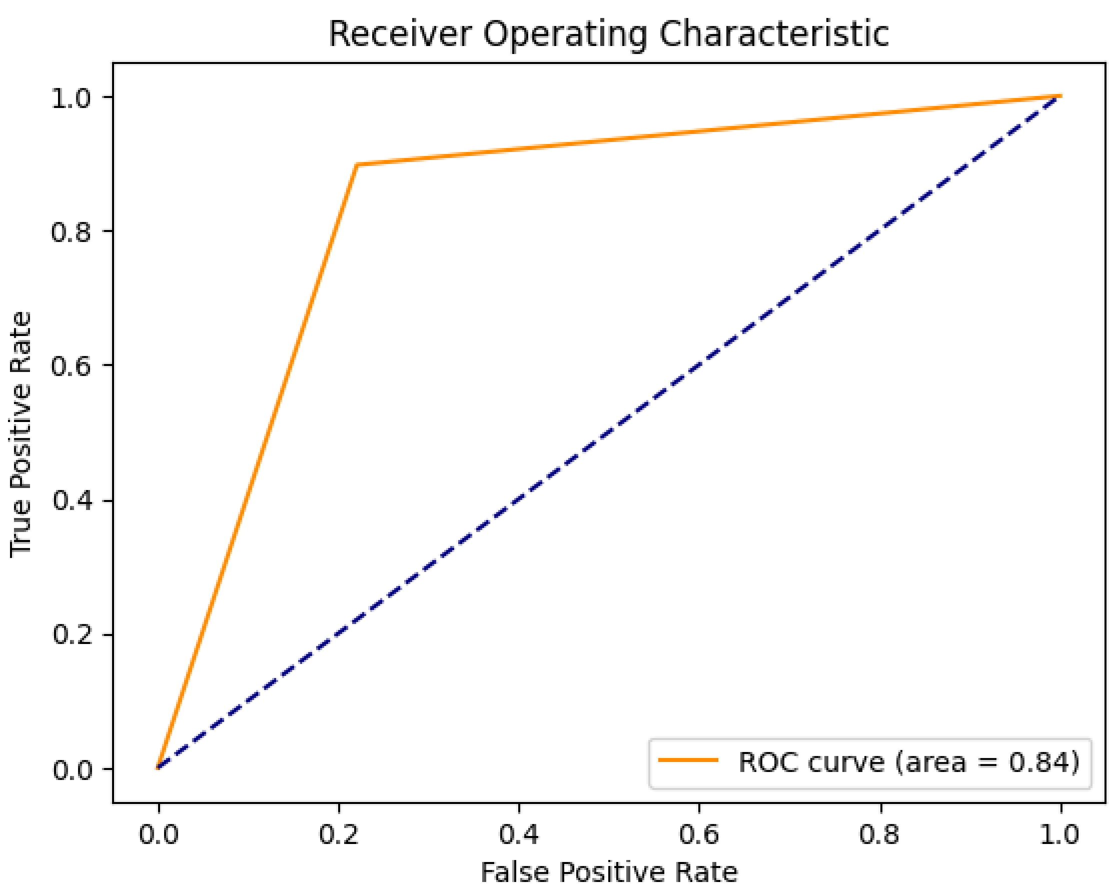
In [40]:
```
# your answer here
model = LogisticRegression(C=grid_search.best_params_['C'], penalty=grid_search.best_pa
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(accuracy_score(y_test, predictions))
```

```
0.8478260869565217
```

In [41]:
```
from sklearn.metrics import roc_curve, auc
```

In [42]:
```
# your answer here

fpr, tpr, thresholds = roc_curve(y_test, predictions)
roc_auc = auc(fpr, tpr)
print(roc_auc)
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0,1], [0,1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
0.8382085204515112
```



```
[[60 17]
 [11 96]]
              precision    recall  f1-score   support

           0       0.85      0.78      0.81        77
           1       0.85      0.90      0.87       107

    accuracy                           0.85       184
   macro avg       0.85      0.84      0.84       184
weighted avg       0.85      0.85      0.85       184
```

# Linear Regression Tuning

## Elastic

In [43]:

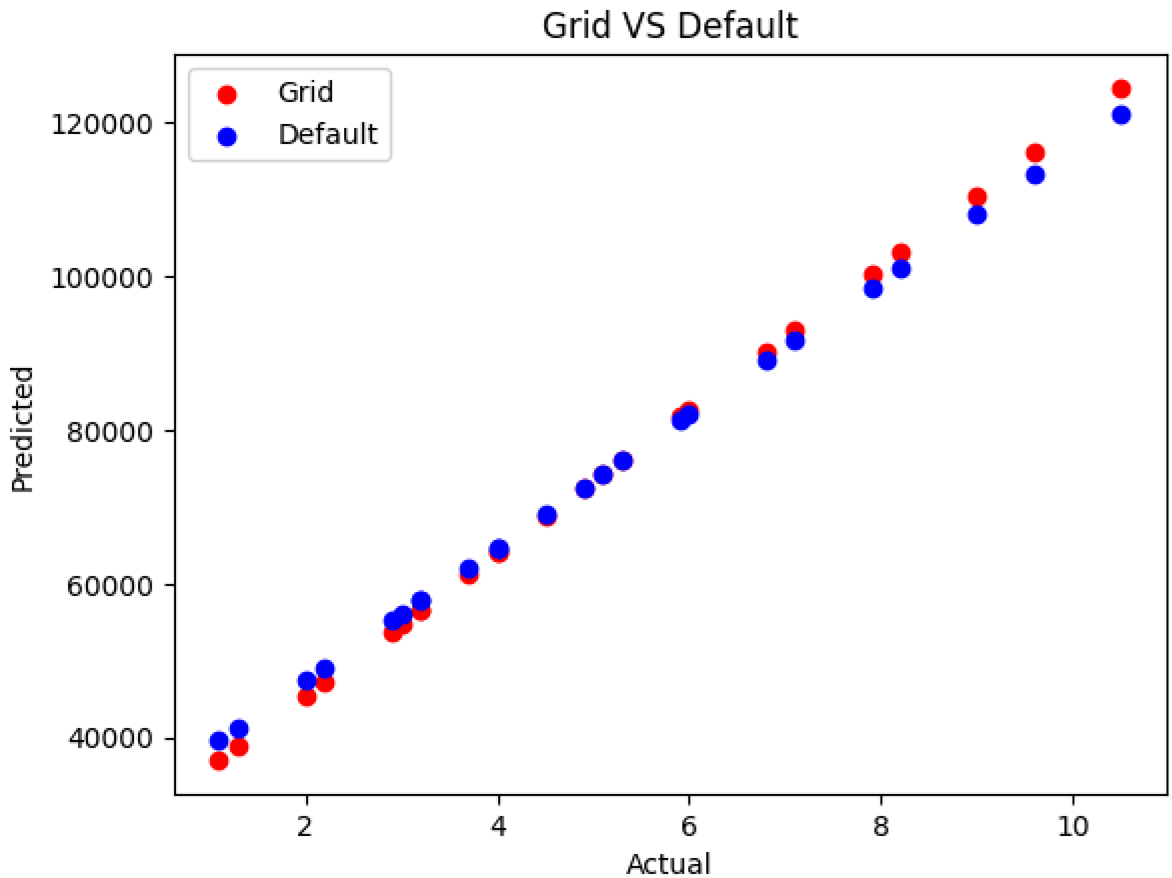
```
parameters = {'alpha': [0.001, 0.01, 0.1, 1], 'l1_ratio': [0.001, 0.1, 0.25, 0.5, 0.75,
model = ElasticNet()
grid_search = GridSearchCV(model, parameters)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'alpha': 0.01, 'l1_ratio': 0.001, 'max_iter': 1000}
0.5178328889165776
```

In [44]:

```
# comparision in scatterplot

model = ElasticNet(alpha=grid_search.best_params_['alpha'], l1_ratio=grid_search.best_p
X_train, X_test, y_train, y_test = train_test_split(dataSet_salary['YearsExperience'],
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Grid: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='red')
plt.title('Grid VS Default')
plt.xlabel('Actual')
plt.ylabel('Predicted')

model = ElasticNet()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Default: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.legend(['Grid', 'Default'])

plt.show()
```

```
Grid:  0.9880383226742803
Default:  0.9772686017240042
```




## Ridge

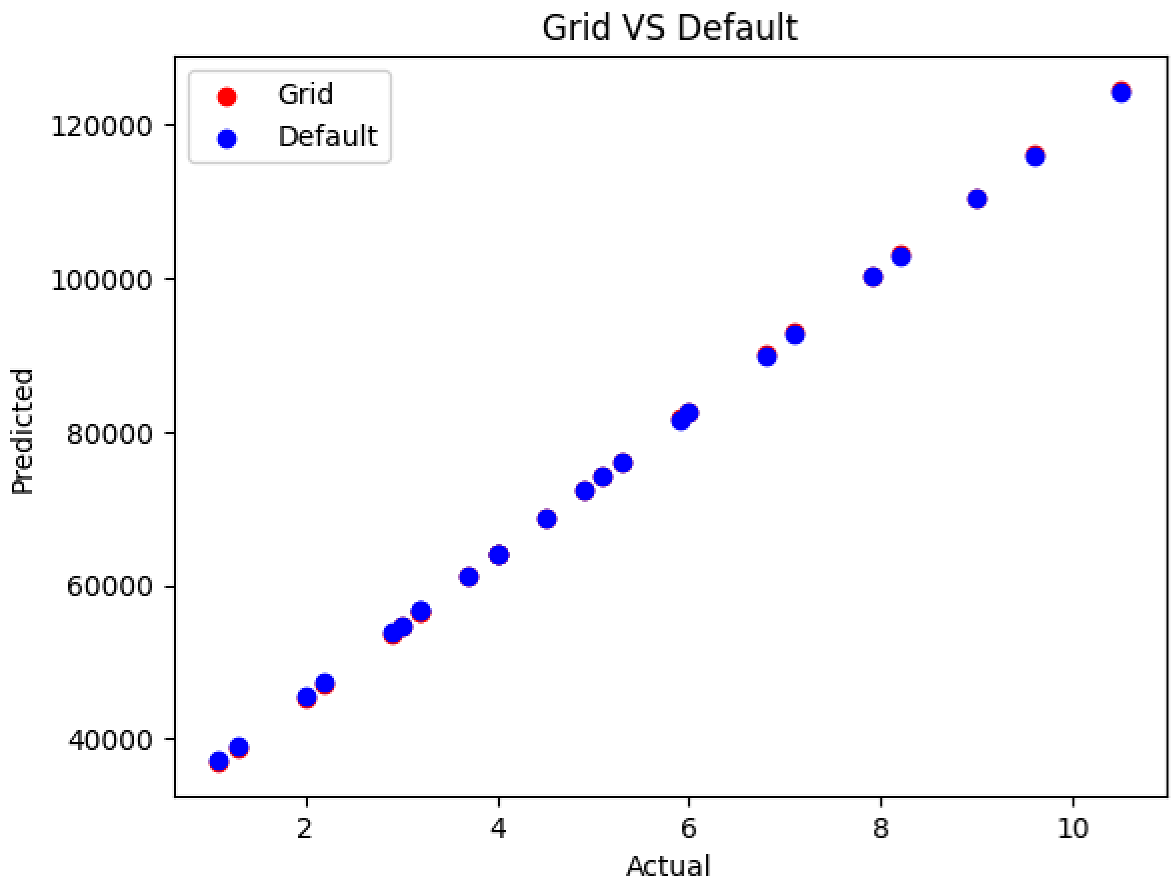
In [45]:
```
parameters = {'alpha': [0.001, 0.01, 0.1, 1], 'fit_intercept': [True, False], 'max_iter
model = Ridge()
grid_search = GridSearchCV(model, parameters)
grid_search.fit(X_train.values.reshape(-1,1), y_train)
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'alpha': 0.001, 'fit_intercept': True, 'max_iter': 1000}
0.9272137573042315
```

In [46]:
```
# comparision in scatterplot
model = Ridge(alpha=grid_search.best_params_['alpha'], fit_intercept=grid_search.best_p
X_train, X_test, y_train, y_test = train_test_split(dataSet_salary['YearsExperience'], 
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Grid: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='red')
plt.title('Grid VS Default')
plt.xlabel('Actual')
plt.ylabel('Predicted')

model = Ridge()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Default: ", r2_score(y_test, predictions))
```

```
plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.legend(['Grid', 'Default'])

plt.show()
```

```
Grid:  0.9881689770761223
Default:  0.9875955163095868
```




## Lasso

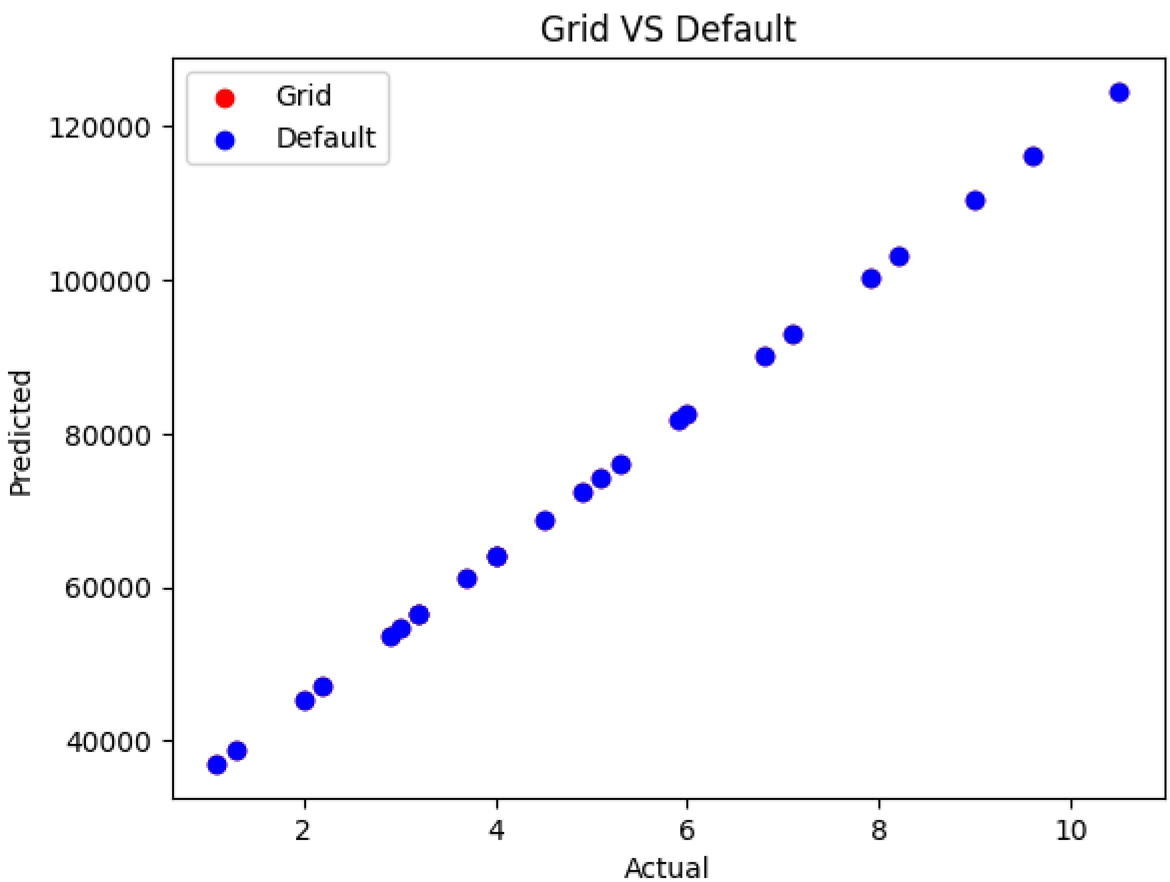
In [47]:
```
parameters = {'alpha': [0.001, 0.01, 0.1, 1], 'fit_intercept': [True, False], 'max_iter
model = Lasso()
grid_search = GridSearchCV(model, parameters)
grid_search.fit(X_train.values.reshape(-1,1), y_train)
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'alpha': 0.001, 'fit_intercept': True, 'max_iter': 1000}
0.9272138116883252
```

In [48]:
```
# comparision in scatterplot
model = Lasso(alpha=grid_search.best_params_['alpha'], fit_intercept=grid_search.best_p
X_train, X_test, y_train, y_test = train_test_split(dataSet_salary['YearsExperience'], 
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Grid: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='red')
plt.title('Grid VS Default')
```

```
plt.xlabel('Actual')
plt.ylabel('Predicted')

model = Lasso()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Default: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.legend(['Grid', 'Default'])

plt.show()
```

```
Grid:  0.988169514341023
Default:  0.988168127365881
```




# HuberRegressor

In [49]:

```
parameters = {'epsilon': [1, 1.35, 1.5, 1.75, 1.9], 'max_iter': [1000], 'alpha': [0.000
model = HuberRegressor()
grid_search = GridSearchCV(model, parameters)
grid_search.fit(X_train.values.reshape(-1,1), y_train)
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'alpha': 0.0001, 'epsilon': 1.5, 'max_iter': 1000}
0.9270353658559995
```

In [50]:

```
# comparision in scatterplot
model = HuberRegressor(epsilon=grid_search.best_params_['epsilon'], max_iter=grid_searc
```

```
X_train, X_test, y_train, y_test = train_test_split(dataSet_salary['YearsExperience'],
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Grid: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='red')
plt.title('Grid VS Default')
plt.xlabel('Actual')
plt.ylabel('Predicted')

model = HuberRegressor()
model.fit(X_train.values.reshape(-1,1), y_train)
predictions = model.predict(X_test.values.reshape(-1,1))
print("Default: ", r2_score(y_test, predictions))

plt.scatter(X_train, model.predict(X_train.values.reshape(-1,1)), color='blue')
plt.legend(['Grid', 'Default'])

plt.show()
```

```
Grid:  0.9875192872597423
Default:  0.9870632883295445
```