In [2]:
```
import numpy as np
import pandas as pd
import sklearn
import seaborn as sns
from sklearn.metrics import r2_score, mean_squared_error, confusion_matrix, roc_curve, classification_report
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,BaggingClassifier,GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from collections import Counter
from sklearn.metrics import r2_score
```

In [3]:
```
df=pd.read_csv("Downloads/Social_Network_Ads_1.csv")
```

In [4]:
```
df
```

Out[4]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

In [5]:
```
df.dtypes
```

Out[5]:
```
Age                int64
EstimatedSalary    int64
Purchased          int64
dtype: object
```

In [6]:
```
df.head(20)
```

Out[6]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
5	27	58000	0
6	27	84000	0
7	32	150000	1
8	25	33000	0
9	35	65000	0
10	26	80000	0
11	26	52000	0
12	20	86000	0
13	32	18000	0
14	18	82000	0
15	29	80000	0
16	47	25000	1
17	45	26000	1
18	46	28000	1
19	48	29000	1

In [7]:
```
df.tail()
```

Out[7]:

	Age	EstimatedSalary	Purchased
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

In [8]:
```
df.describe
```

Out[8]:
```
<bound method NDFrame.describe of      Age  EstimatedSalary  Purchased
0     19            19000          0
1     35            20000          0
2     26            43000          0
3     27            57000          0
4     19            76000          0
..   ...              ...        ...
395   46            41000          1
396   51            23000          1
397   50            20000          1
398   36            33000          0
399   49            36000          1

[400 rows x 3 columns]>
```

In [9]:
```
df.shape
```

Out[9]:
```
(400, 3)
```

In [10]:
```
df.info
```

Out[10]:
```
<bound method DataFrame.info of      Age  EstimatedSalary  Purchased
0     19            19000          0
1     35            20000          0
2     26            43000          0
3     27            57000          0
4     19            76000          0
..   ...              ...        ...
395   46            41000          1
396   51            23000          1
397   50            20000          1
398   36            33000          0
399   49            36000          1

[400 rows x 3 columns]>
```

In [11]:
```
X = df.iloc[:, :-1].values
X
```

Out[11]:
```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],
       [    32, 150000],
       [    25,  33000],
       [    35,  65000],
       [    26,  80000],
       [    26,  52000],
       [    20,  86000],
       [    32,  18000],
       [    18,  82000],
       [    29,  80000],
       [    47,  25000],
       [    45,  26000],
       [    46,  28000],
```

In [12]:
```
y = df.iloc[:, -1].values
y
```

Out[12]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1], dtype=int64)
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.5,random_state = 0)
```

In [14]:
```
X_train
```

Out[14]:
```
array([[    28,  59000],
       [    40,  57000],
       [    59, 143000],
       [    57,  26000],
       [    52,  38000],
       [    47, 113000],
       [    53, 143000],
       [    35,  27000],
       [    58, 101000],
       [    45,  45000],
       [    23,  82000],
       [    46,  23000],
       [    42,  65000],
       [    28,  84000],
       [    38,  59000],
       [    26,  84000],
       [    29,  28000],
       [    37,  71000],
       [    22,  55000],
```

In [15]:
```
X_test
```

Out[15]:
```
array([[    30,  87000],
       [    38,  50000],
       [    35,  75000],
       [    30,  79000],
       [    35,  50000],
       [    27,  20000],
       [    31,  15000],
       [    36, 144000],
       [    18,  68000],
       [    47,  43000],
       [    30,  49000],
       [    28,  55000],
       [    37,  55000],
       [    39,  77000],
       [    20,  86000],
       [    32, 117000],
       [    37,  77000],
       [    19,  85000],
       [    55, 130000],
```

In [16]:
```
Y_train
```

Out[16]:
```
array([0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0], dtype=int64)
```

In [17]:
```
Y_test
```

Out[17]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1], dtype=int64)
```

In [18]:
```
print(X_train)
print(X_test)
```

```
[[    28  59000]
 [    40  57000]
 [    59 143000]
 [    57  26000]
 [    52  38000]
 [    47 113000]
 [    53 143000]
 [    35  27000]
 [    58 101000]
 [    45  45000]
 [    23  82000]
 [    46  23000]
 [    42  65000]
 [    28  84000]
 [    38  59000]
 [    26  84000]
 [    29  28000]
 [    37  71000]
 [    22  55000]
```

In [24]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

```
[[-0.95215868 -0.22005819]
 [ 0.21613418 -0.27905503]
 [ 2.06593121  2.25780886]
 [ 1.87121573 -1.19350596]
 [ 1.38442704 -0.83952496]
 [ 0.89763835  1.37285634]
 [ 1.48178478  2.25780886]
 [-0.27065451 -1.16400755]
 [ 1.96857347  1.01887533]
 [ 0.70292287 -0.63303603]
 [-1.43894737  0.4584054 ]
 [ 0.80028061 -1.28200121]
 [ 0.41084966 -0.04306769]
 [-0.95215868  0.51740224]
 [ 0.0214187  -0.22005819]
 [-1.14687416  0.51740224]
 [-0.85480094 -1.13450913]
 [-0.07593904  0.13392281]
 [-1.53630511 -0.33805186]
```

In [25]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
```

Out[25]:
```
GaussianNB()
```

In [26]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

In [27]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), Y_test.reshape(len(Y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
```

In [28]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test, y_pred)
```

```
[[115   8]
 [ 20  57]]
```

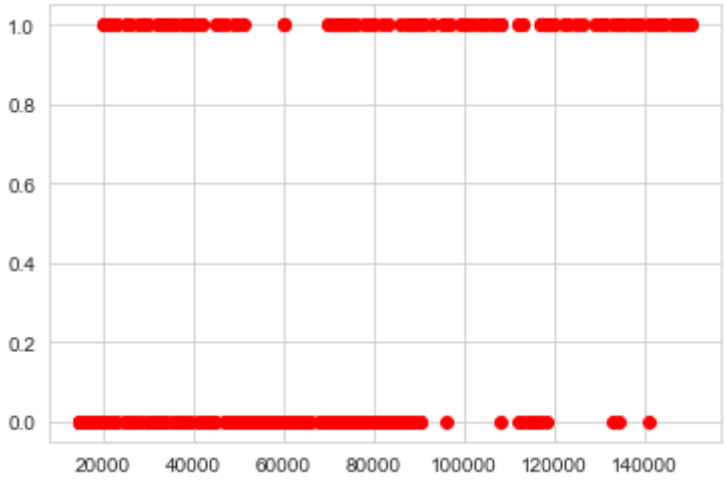
Out[28]:
```
0.86
```

In [29]:
```
accuracy_score(Y_test, y_pred)
```

Out[29]:
```
0.86
```

In [76]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(df['EstimatedSalary'],df['Purchased'],color='red')
```

Out[76]:
```
<matplotlib.collections.PathCollection at 0x1ed57a32cd0>
```



In [31]:
```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier()
model.fit(X_train,Y_train)
```

Out[31]:
```
RandomForestClassifier()
```

In [32]:
```
model.score(X_test,Y_test)
```

Out[32]:
```
0.875
```

In [33]:
```
pd.get_dummies(df.Purchased)
```

Out[33]:

	0	1
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
395	0	1
396	0	1
397	0	1
398	1	0
399	0	1

400 rows × 2 columns

In [34]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

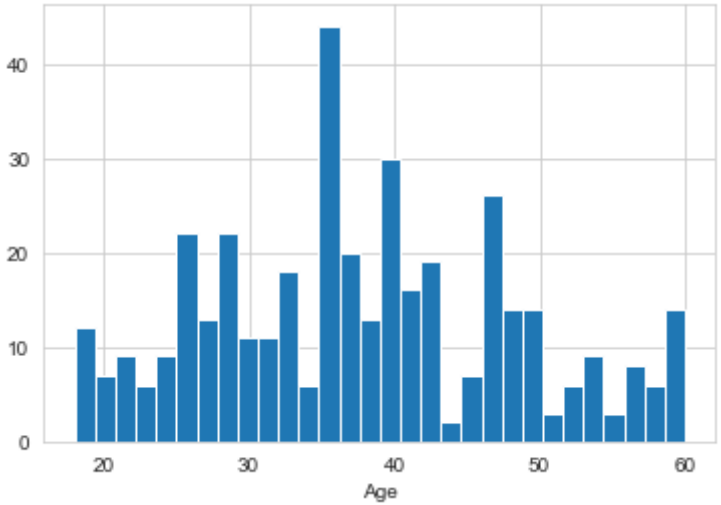
```
[0]
```

In [35]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), Y_test.reshape(len(Y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
```

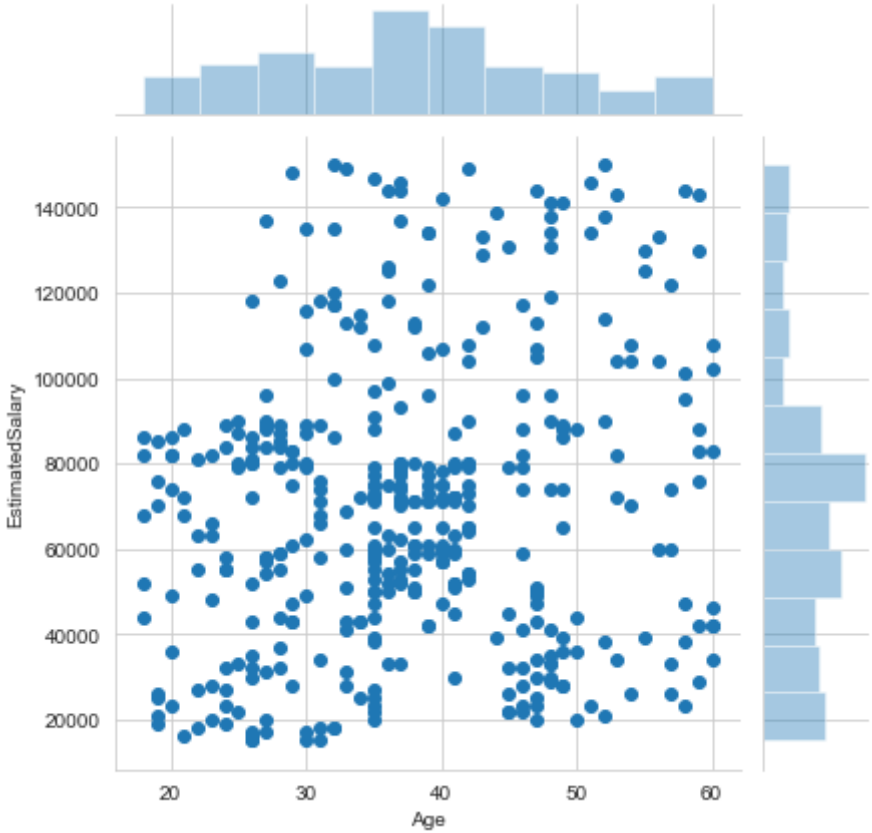
In [36]:
```
sns.set_style('whitegrid')
df['Age'].hist(bins=30)
plt.xlabel('Age')
```

Out[36]:
```
Text(0.5, 0, 'Age')
```



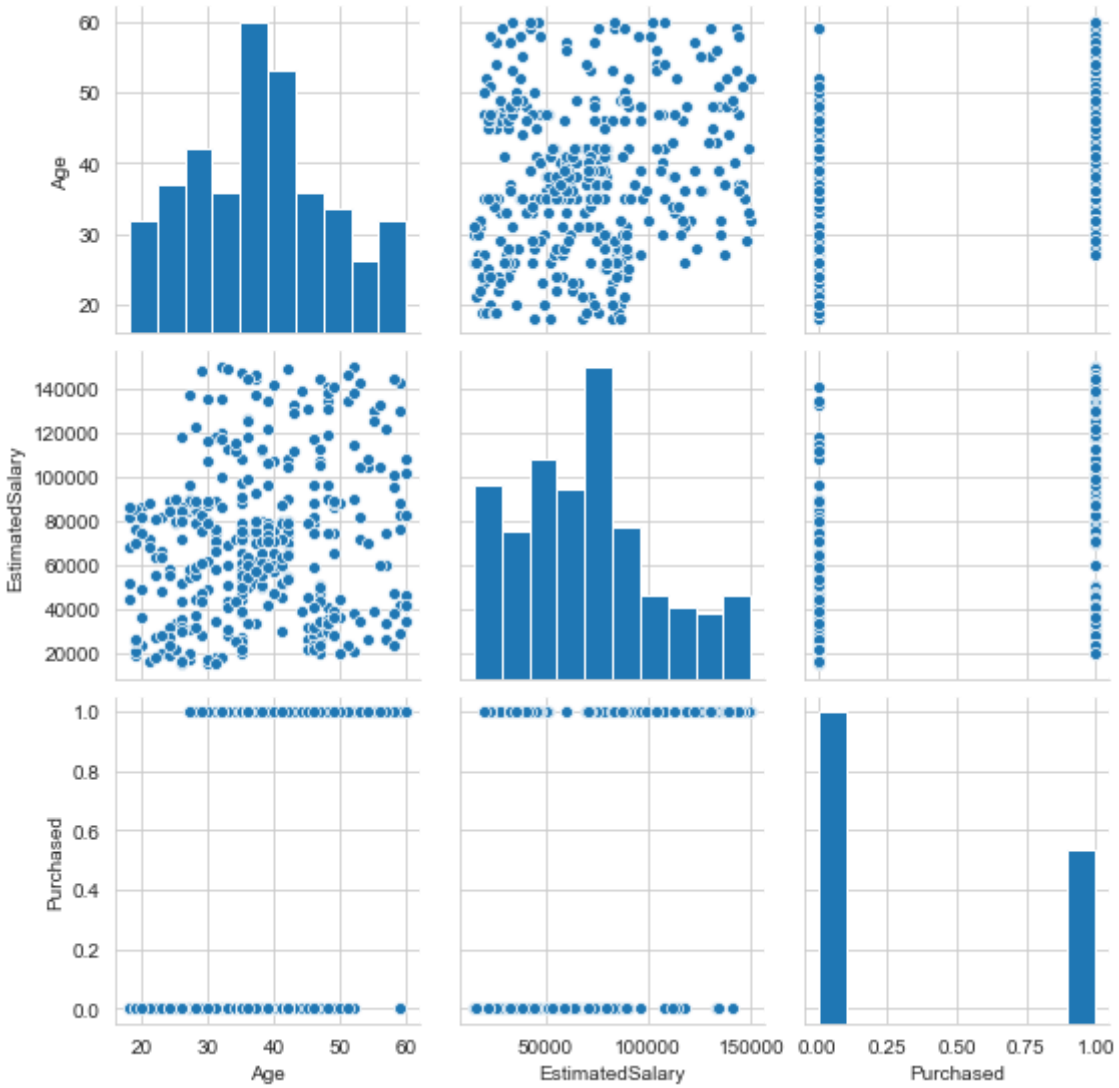
In [37]:
```
sns.jointplot(x='Age',y='EstimatedSalary',data=df)
```

Out[37]:
```
<seaborn.axisgrid.JointGrid at 0x1ed55eafeb0>
```



In [77]:
```
sns.pairplot(df)
```

Out[77]: <seaborn.axisgrid.PairGrid at 0x1ed57d37790>



In [39]:
```
from sklearn.svm import SVC
```

In [40]:
```
model  = SVC()
```

In [41]:
```
model.fit(X_train,Y_train)
```

Out[41]:
```
SVC()
```

In [42]:
```
pred = model.predict(X_test)
```

In [43]:
```
from sklearn.metrics import classification_report , confusion_matrix
```

In [44]:
```
print( confusion_matrix (Y_test , pred))
```

```
[[112  11]
 [  9  68]]
```

In [45]:
```
print( classification_report(Y_test , pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       123
           1       0.86      0.88      0.87        77

    accuracy                           0.90       200
   macro avg       0.89      0.90      0.89       200
weighted avg       0.90      0.90      0.90       200
```

In [46]:
```
from sklearn .metrics import roc_auc_score

y_true = [1, 1, 0, 0, 1, 0]

y_pred = [0.95, 0.90, 0.85, 0.81, 0.78, 0.70]

auc = np.round(roc_auc_score(y_true, y_pred), 3)

print("Auc for our sample data is {}". format(auc))
```

```
Auc for our sample data is 0.778
```

In [47]:
```
from sklearn.naive_bayes import GaussianNB
model_naive = GaussianNB()
model_naive.fit(X_train, Y_train)
```

Out[47]:
```
GaussianNB()
```

In [48]:
```
log_reg=LogisticRegression(random_state=0,max_iter=900)
```

In [49]:
```
log_reg.fit(X_train,Y_train)
```

Out[49]:
```
LogisticRegression(max_iter=900, random_state=0)
```

In [50]:
```
y_pred=log_reg.predict(X_test)
```

In [51]:
```
log_reg.score(X_test,Y_test)
```

Out[51]:
```
0.825
```

In [52]:
```
params = {'C': np.logspace(-3, 3, 7), 'penalty': ['l1', 'l2']}
lr_model = LogisticRegression(random_state = 0)
lr_cv = GridSearchCV(lr_model,params,cv = 5).fit(X_train,Y_train)
lr_cv.best_params_
```

```
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
```

```
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1304, in fit
```

```
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\abc\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 442, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
```

Out[52]:
```
{'C': 100.0, 'penalty': 'l2'}
```

In [53]:
```
log_reg2 = LogisticRegression(C = .01,random_state = 0,penalty= 'l2')
log_reg2.fit(X_train,Y_train)
```

Out[53]:
```
LogisticRegression(C=0.01, random_state=0)
```

In [54]:
```
log_reg2.predict(X_test)
```

Out[54]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [55]:
```
log_reg2.score(X_test,Y_test)
```

Out[55]:
```
0.705
```

In [56]:
```
knn = KNeighborsClassifier(n_neighbors = 2).fit(X_train,Y_train)
knn.score(X_test,Y_test)
```

Out[56]:
```
0.835
```

In [57]:
```
tree = DecisionTreeClassifier(random_state = 0).fit(X_train,Y_train)
tree.score(X_test,Y_test)
```

Out[57]: 0.855

In [58]:
```
tree_tuned = DecisionTreeClassifier(max_depth = 1, min_samples_leaf = 7, min_samples_split = 2,random_state = 42).fit(X_
tree_tuned.score(X_test,Y_test)
```

Out[58]: 0.805

In [59]:
```
rf = RandomForestClassifier(random_state = 0).fit(X_train,Y_train)
rf.score(X_test,Y_test)
```

Out[59]: 0.87

In [60]:
```
svm = SVC(random_state = 42).fit(X_train,Y_train)
svm.score(X_test,Y_test)
```

Out[60]: 0.9

In [61]:
```
gbm = GradientBoostingClassifier(random_state = 42).fit(X_train,Y_train)
gbm.score(X_test,Y_test)
```

Out[61]: 0.86

In [62]:
```
gbm_tuned = GradientBoostingClassifier(max_depth = 2, learning_rate = 0.01, min_samples_split = 2, n_estimators = 100, r
gbm_tuned.score(X_test,Y_test)
```

Out[62]: 0.89

In [63]:
```
ada = AdaBoostClassifier(random_state = 42).fit(X_train,Y_train)
ada.score(X_test,Y_test)
```

Out[63]: 0.86

In [64]:
```
ada_tuned = AdaBoostClassifier(learning_rate = 0.01,n_estimators = 1000,random_state = 42).fit(X_train,Y_train)
ada_tuned.score(X_test,Y_test)
```

Out[64]: 0.86

In [65]:
```
bag = BaggingClassifier(random_state = 42).fit(X_train,Y_train)
bag.score(X_test,Y_test)
```

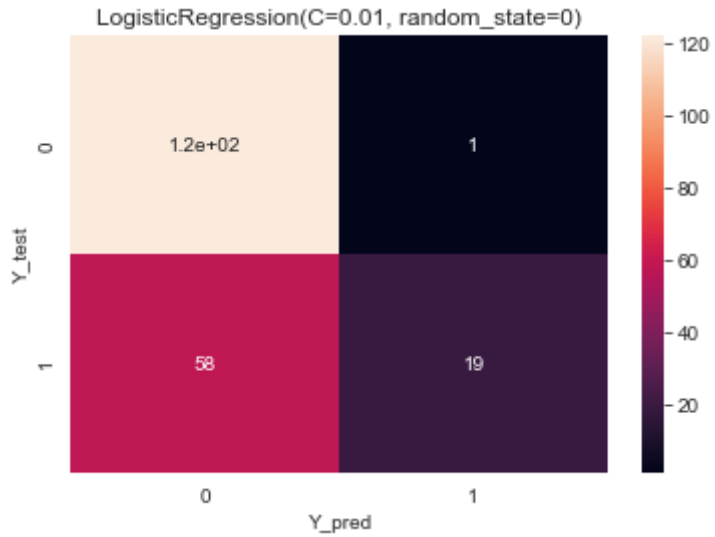
Out[65]: 0.88

In [66]:
```
bag_tuned = BaggingClassifier(n_estimators = 45,random_state = 42).fit(X_train,Y_train)
bag_tuned.score(X_test,Y_test)
```

Out[66]: 0.865

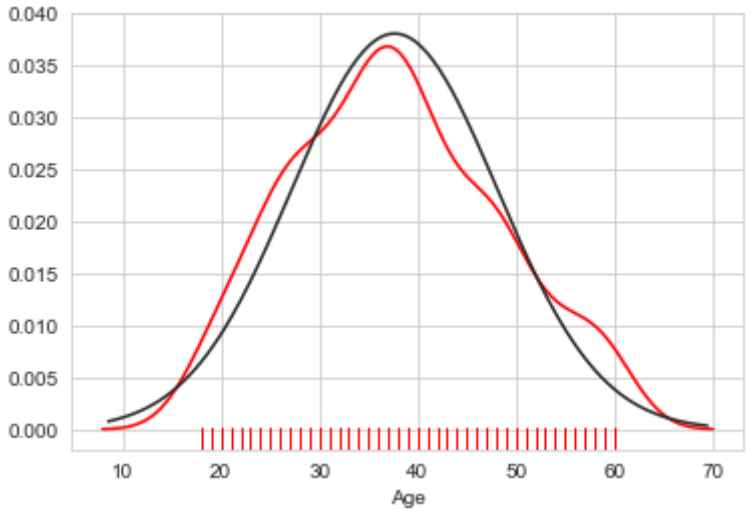
In [67]:
```
pred_list = [log_reg2,knn,rf,gbm_tuned,svm,ada_tuned,bag_tuned]

for i in pred_list:
    print("Score : ",i.score(X_test,Y_test))
    y_pred = i.predict(X_test)
    sns.heatmap(confusion_matrix(Y_test,y_pred),annot = True)
    plt.xlabel("Y_pred")
    plt.ylabel("Y_test")
    plt.title(i)
    plt.show()
```

```
Score :  0.705
```



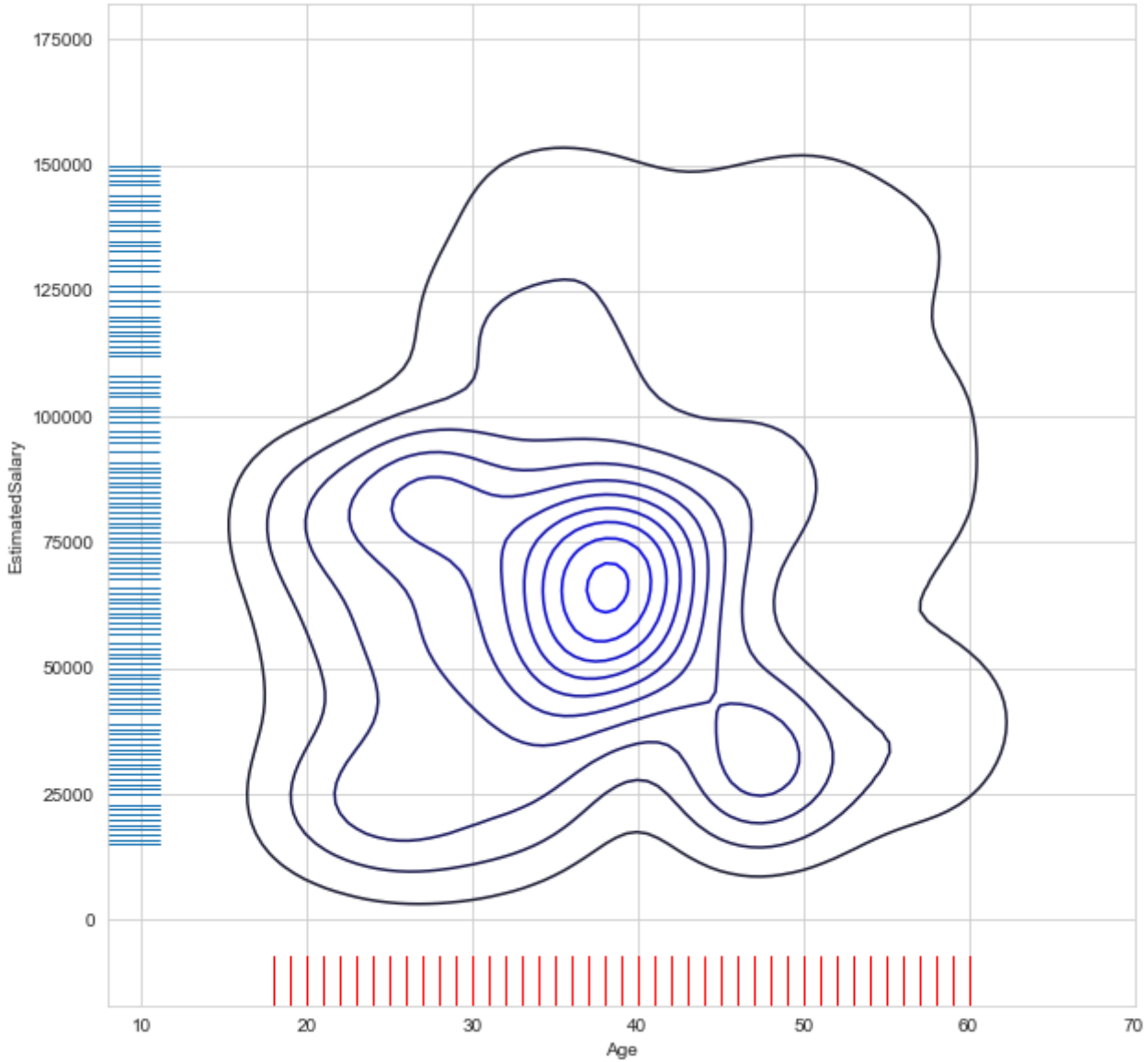

```
Score :  0.835
```

In [68]:
```
from scipy.stats import norm
sns.distplot(df['Age'], hist=False, color='r', rug=True, fit=norm);
```



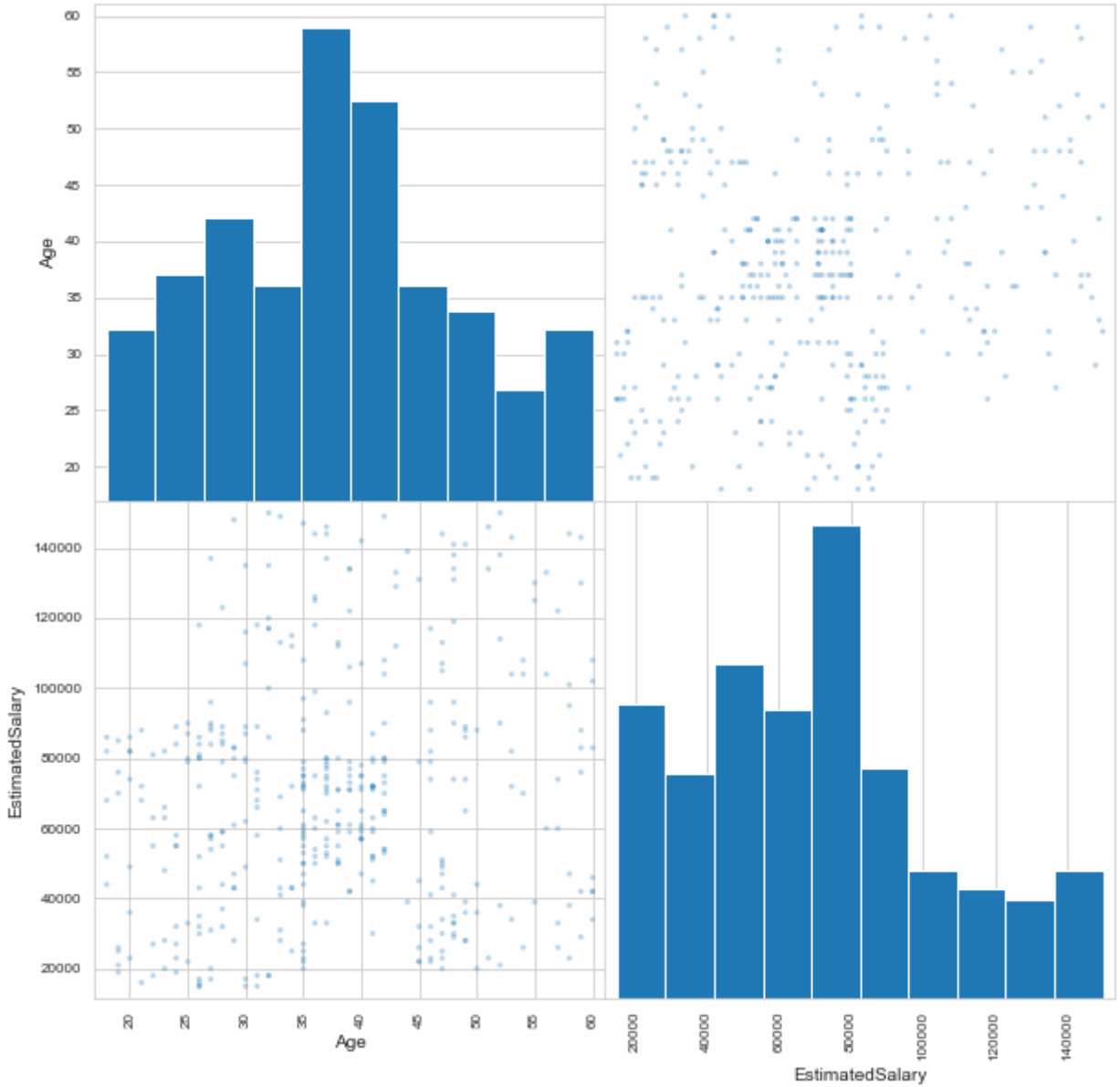
In [69]:
```
f, ax = plt.subplots(figsize=(10, 10))
sns.kdeplot(df.Age, df['EstimatedSalary'], color="b", ax=ax)
sns.rugplot(df.Age, color="r", ax=ax)
sns.rugplot(df['EstimatedSalary'],vertical=True, ax=ax)
```

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x1ed57af1c40>



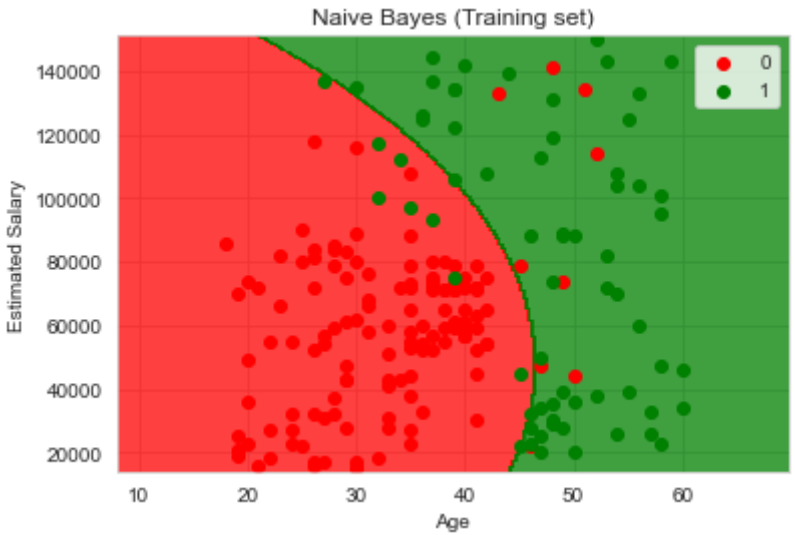
In [71]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(df[['Age', 'EstimatedSalary']],
    alpha=0.3, figsize=(10,10))
```

Out[71]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001ED56887970>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001ED57B29190>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001ED579585E0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001ED56841A30>]],
      dtype=object)
```


Age
EstimatedSalary
Age
EstimatedSalary

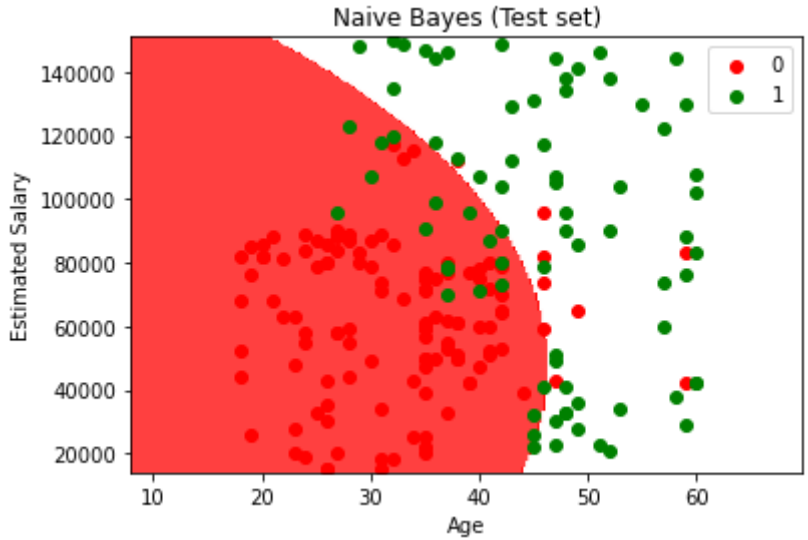
In [75]:
```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prece
dence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prece
dence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```


Naive Bayes (Training set)
Estimated Salary
Age
0
1
140000
120000
100000
80000
60000
40000
20000
10
20
30
40
50
60

In [42]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), Y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'white')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prece
dence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prece
dence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```



In [ ]:

In [ ]:

In [ ]: