Problem Statement:

1.) Prepare a classification model using the Naive Bayes algorithm for the salary dataset. Train and test datasets are given separately. Use both for model building.

Data Pre-processing

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer

# Loading the data set
salary_test = pd.read_csv("D:/360Digi/naive bayes/SalaryData_Test.csv")
salary_train = pd.read_csv("D:\\360Digi\\naive bayes\\SalaryData_Train.csv")

salary_train.isnull().sum()
salary_test.isnull().sum()

salary_train.count()
salary_test.count()

salary_train.occupation.value_counts()

salary_train.workclass.unique()

salary_train.columns
salary_test.columns

from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()

string_columns=['workclass','education','maritalstatus','occupation','relationshi

for i in string_columns:
    salary_train[i] = labelencoder.fit_transform(salary_train[i])
    salary_test[i] = labelencoder.fit_transform(salary_test[i])
```

EDA

In [8]:
```
salary_train.isnull().sum()
```

Out[8]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

In [9]:
```
salary_test.isnull().sum()
```

Out[9]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

In [10]:
```
salary_train.describe()
```

Out[10]:

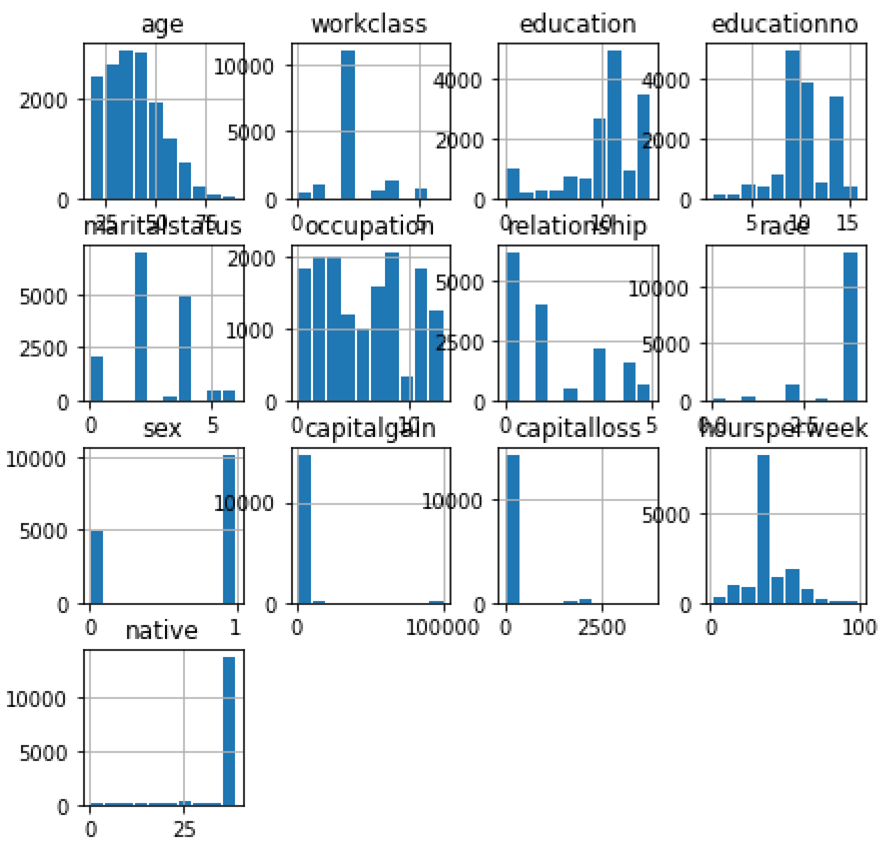
	age	workclass	education	educationno	maritalstatus	occupation	relationshi
›unt	30161.000000	30161.00000	30161.00000	30161.000000	30161.000000	30161.000000	30161.00000
ean	38.438115	2.19933	10.33361	10.121316	2.580087	5.959849	1.41832
std	13.134830	0.95394	3.81226	2.550037	1.498018	4.029633	1.60136
min	17.000000	0.00000	0.00000	1.000000	0.000000	0.000000	0.00000
25%	28.000000	2.00000	9.00000	9.000000	2.000000	2.000000	0.00000
50%	37.000000	2.00000	11.00000	10.000000	2.000000	6.000000	1.00000
75%	47.000000	2.00000	12.00000	13.000000	4.000000	9.000000	3.00000
nax	90.000000	6.00000	15.00000	16.000000	6.000000	13.000000	5.00000

In [11]:
```
salary_test.isnull().sum()
```

Out[11]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

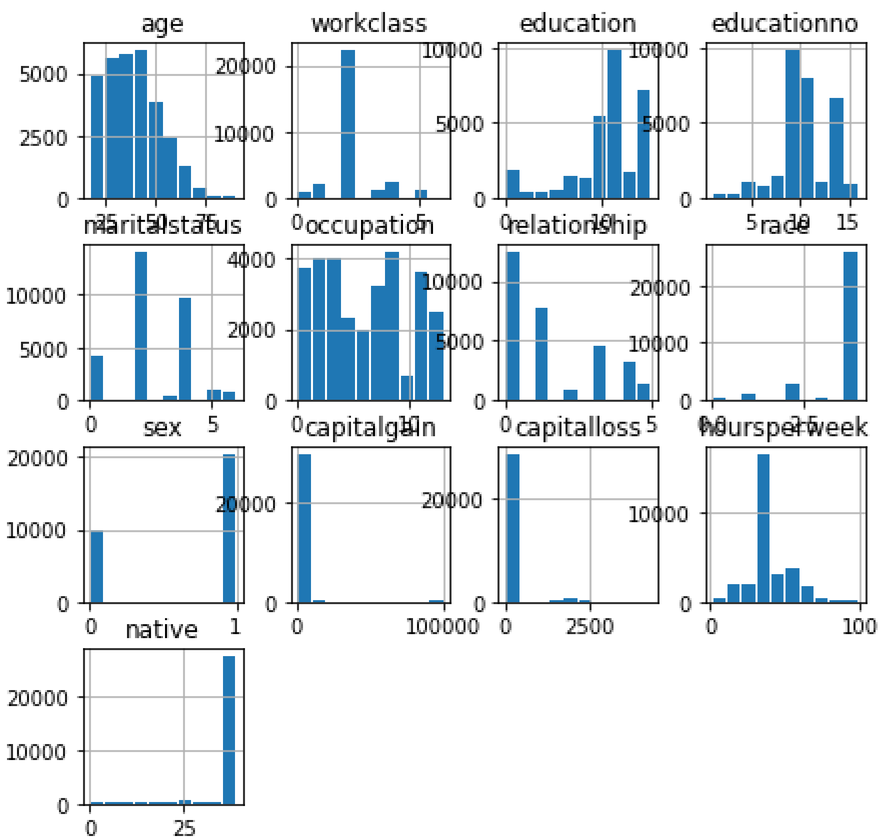
In [32]:
```
salary_test.hist(grid=True, rwidth=0.9, figsize=(7,7))
```

Out[32]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'workclass'}>,
        <AxesSubplot:title={'center':'education'}>,
        <AxesSubplot:title={'center':'educationno'}>],
       [<AxesSubplot:title={'center':'maritalstatus'}>,
        <AxesSubplot:title={'center':'occupation'}>,
        <AxesSubplot:title={'center':'relationship'}>,
        <AxesSubplot:title={'center':'race'}>],
       [<AxesSubplot:title={'center':'sex'}>,
        <AxesSubplot:title={'center':'capitalgain'}>,
        <AxesSubplot:title={'center':'capitalloss'}>,
        <AxesSubplot:title={'center':'hoursperweek'}>],
       [<AxesSubplot:title={'center':'native'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```

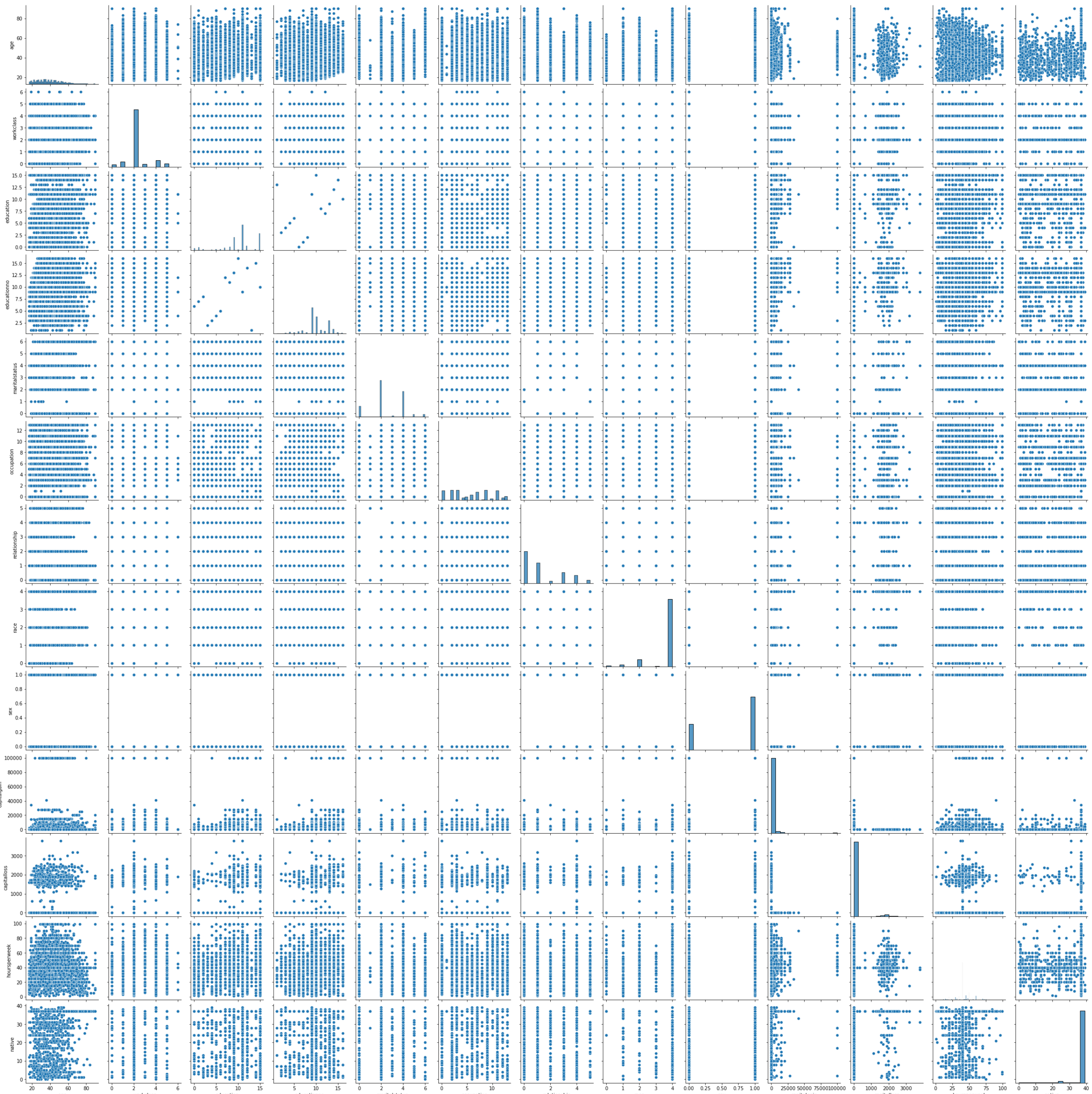


In [33]:
```python
salary_train.hist(grid=True, rwidth=0.9, figsize=(7,7))
```

Out[33]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'workclass'}>,
        <AxesSubplot:title={'center':'education'}>,
        <AxesSubplot:title={'center':'educationno'}>],
       [<AxesSubplot:title={'center':'maritalstatus'}>,
        <AxesSubplot:title={'center':'occupation'}>,
        <AxesSubplot:title={'center':'relationship'}>,
        <AxesSubplot:title={'center':'race'}>],
       [<AxesSubplot:title={'center':'sex'}>,
        <AxesSubplot:title={'center':'capitalgain'}>,
        <AxesSubplot:title={'center':'capitalloss'}>,
        <AxesSubplot:title={'center':'hoursperweek'}>],
       [<AxesSubplot:title={'center':'native'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```

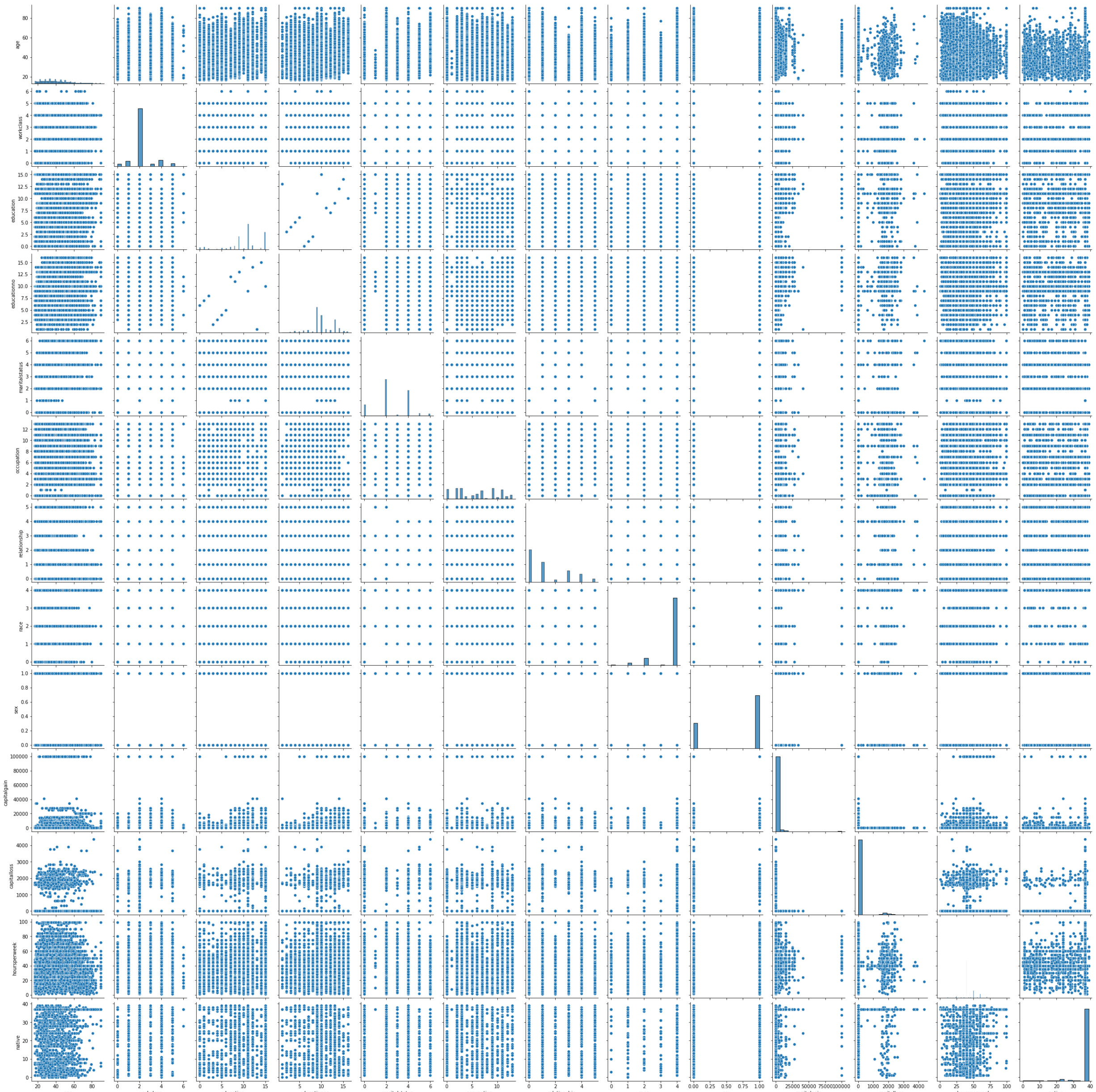


In [34]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(salary_test)
plt.figure(figsize=(7,7))
plt.show()
```




```
<Figure size 504x504 with 0 Axes>
```

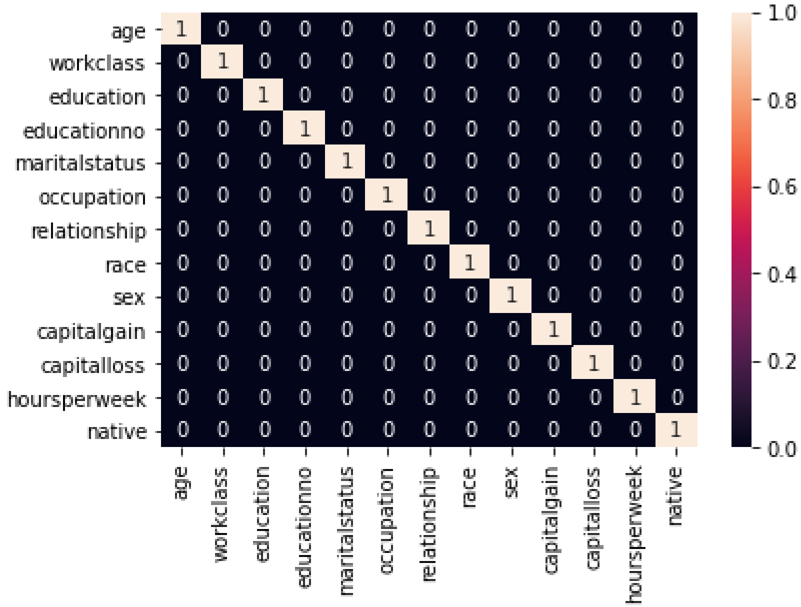
In [35]:
```
import seaborn as sns

sns.pairplot(salary_train)
plt.figure(figsize=(7,7))
plt.show()
```




```
<Figure size 504x504 with 0 Axes>
```

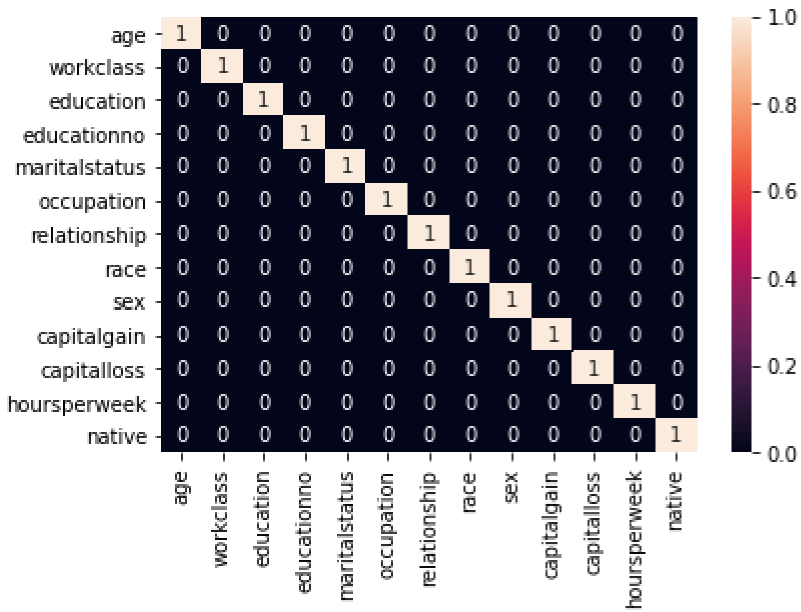
In [15]:
```
a = salary_train.corr(method ='pearson')
sns.heatmap(a>0.85,annot=True)
```

Out[15]: <AxesSubplot:>



In [14]:
```
a = salary_test.corr(method ='pearson')
sns.heatmap(a>0.85,annot=True)
```

Out[14]: <AxesSubplot:>



In [ ]:

In [16]:
```
col_names = list(salary_train.columns)


train_X=salary_train[col_names[0:13]]
train_Y=salary_train[col_names[13]]
test_x=salary_test[col_names[0:13]]
test_y=salary_test[col_names[13]]
```

In [ ]:

Model Building

In [17]:

```
#Gaussian Naive Bayes

from sklearn.naive_bayes import GaussianNB
Gmodel=GaussianNB()
train_pred_gau=Gmodel.fit(train_X,train_Y).predict(train_X)
test_pred_gau=Gmodel.fit(train_X,train_Y).predict(test_x)  #pridiction of test da

train_acc_gau=np.mean(train_pred_gau==train_Y)
test_acc_gau=np.mean(test_pred_gau==test_y)
train_acc_gau#0.795
test_acc_gau#0.794
```

Out[17]: 0.7946879150066402

In [22]:

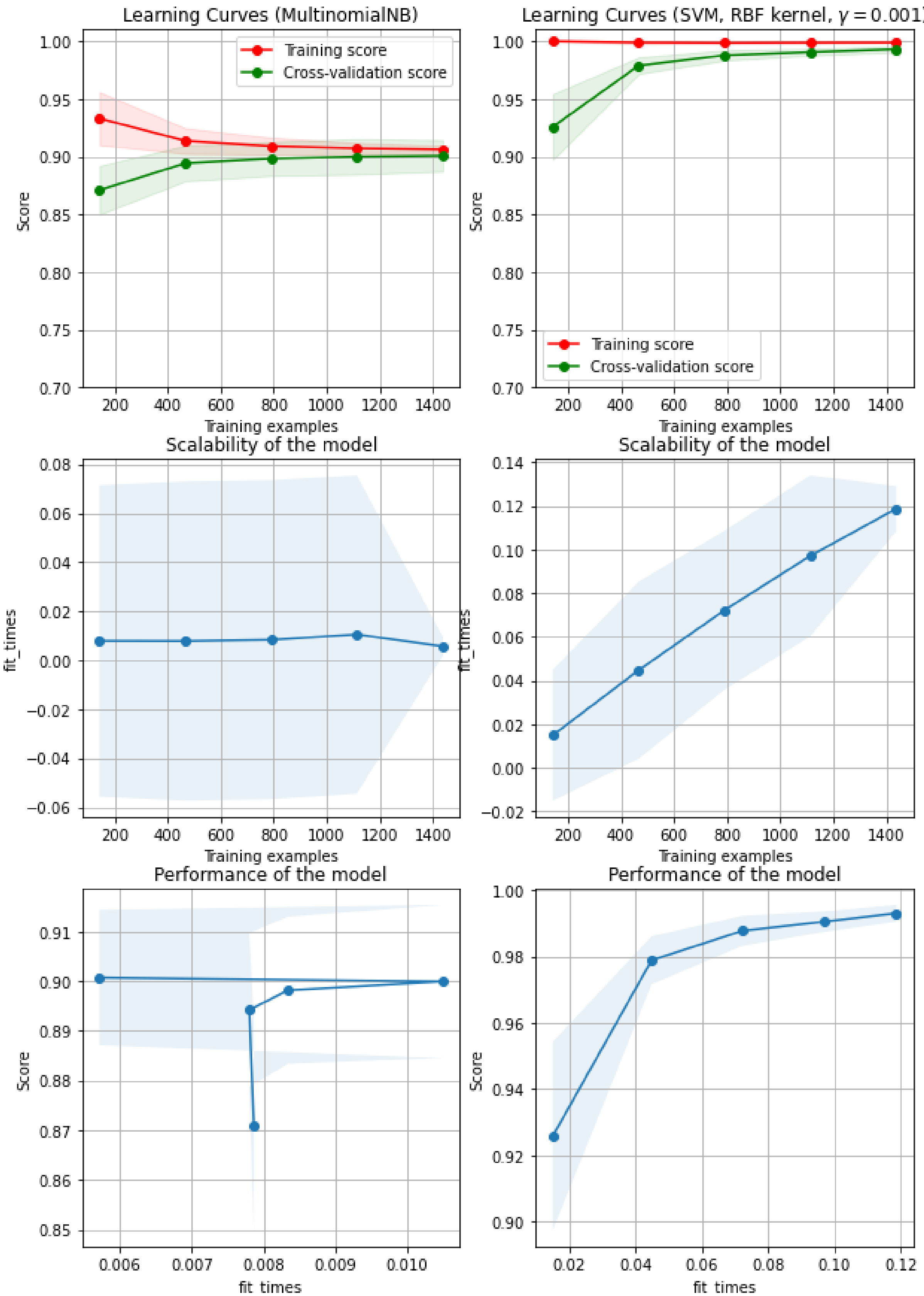
```
#Multinomial Naive Bayes

from sklearn.naive_bayes import MultinomialNB
Mmodel=MultinomialNB()
train_pred_multi=Mmodel.fit(train_X,train_Y).predict(train_X)
test_pred_multi=Mmodel.fit(train_X,train_Y).predict(test_x)

train_acc_multi=np.mean(train_pred_multi==train_Y)
test_acc_multi=np.mean(test_pred_multi==test_y)
train_acc_multi#0.772
test_acc_multi#0.774
```

Out[22]: 0.7749667994687915

In [28]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as mtp
import numpy as nm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sco
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve
from sklearn.datasets import load_digits
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer


def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate 3 plots: the test and training learning curve, the training
    samples vs fit times curve, the fit times vs score curve.

    Parameters
    ----------
    estimator : estimator instance
        An estimator instance implementing `fit` and `predict` methods which
        will be cloned for each validation.

    title : str
        Title for the chart.

    X : array-like of shape (n_samples, n_features)
        Training vector, where ``n_samples`` is the number of samples and
        ``n_features`` is the number of features.

    y : array-like of shape (n_samples) or (n_samples, n_features)
        Target relative to ``X`` for classification or regression;
        None for unsupervised learning.

    axes : array-like of shape (3,), default=None
        Axes to use for plotting the curves.

    ylim : tuple of shape (2,), default=None
        Defines minimum and maximum y-values plotted, e.g. (ymin, ymax).

    cv : int, cross-validation generator or an iterable, default=None
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

          - None, to use the default 5-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.
```

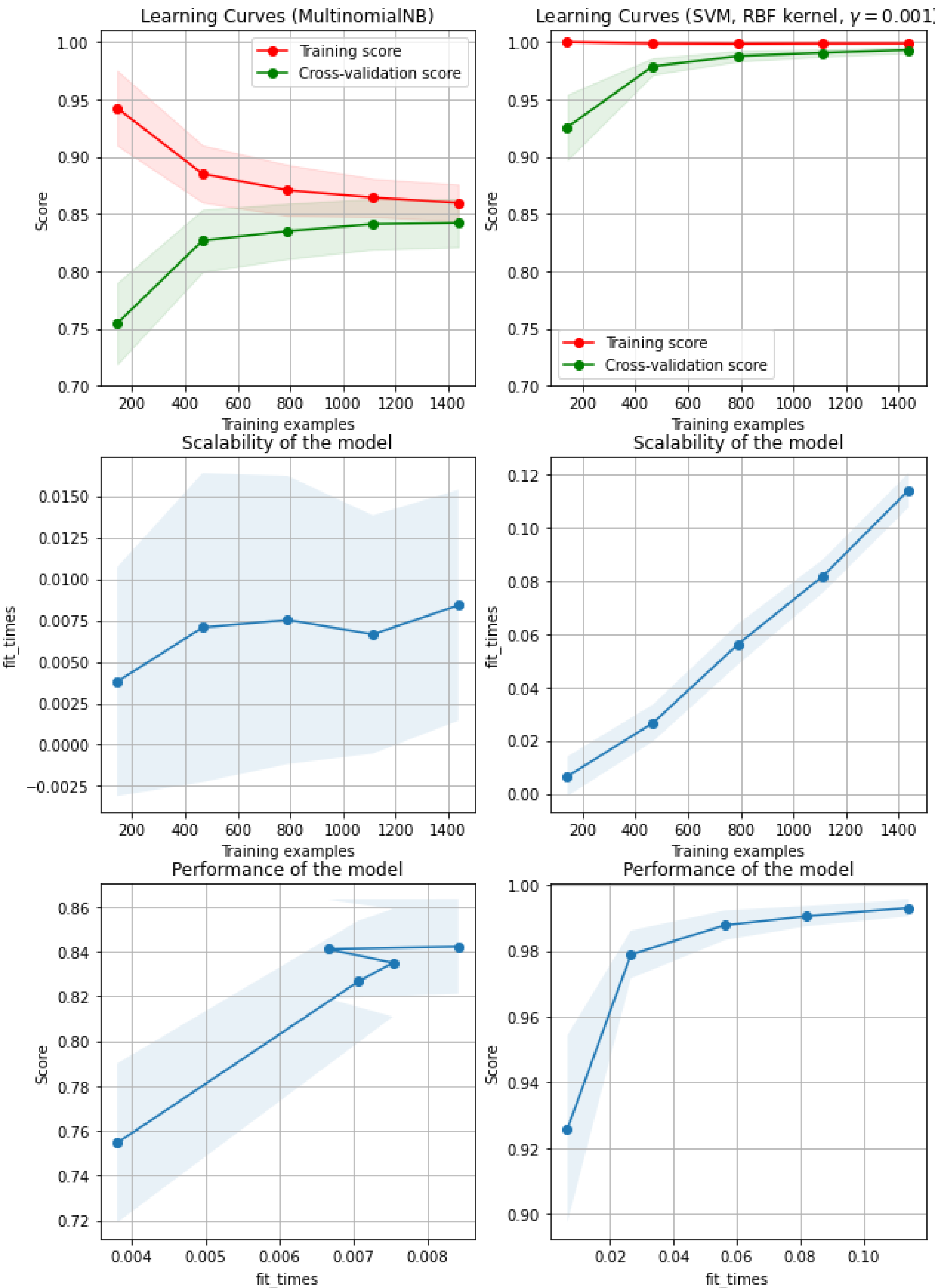
```
        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, default=None
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like of shape (n_ticks,)
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the ``dtype`` is float, it is regarded
        as a fraction of the maximum size of the training set (that is
        determined by the selected validation method), i.e. it has to be within
        (0, 1]. Otherwise it is interpreted as absolute sizes of the training
        sets. Note that for classification the number of samples usually have
        to be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")
```

```
    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt


fig, axes = plt.subplots(3, 2, figsize=(10, 15))

X, y = load_digits(return_X_y=True)

title = "Learning Curves (MultinomialNB)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = MultinomialNB()
plot_learning_curve(estimator, title, X, y, axes=axes[:, 0], ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

title = r"Learning Curves (SVM, RBF kernel, $\gamma=0.001$)"
# SVC is more expensive so we do a lower number of CV iterations:
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
estimator = SVC(gamma=0.001)
plot_learning_curve(estimator, title, X, y, axes=axes[:, 1], ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()


####
```


Learning Curves (MultinomialNB)
Training score
Cross-validation score
Score
Training examples
Learning Curves (SVM, RBF kernel, $\gamma = 0.001$)
Training score
Cross-validation score
Score
Training examples
Scalability of the model
fit_times
Training examples
Scalability of the model
fit_times
Training examples
Performance of the model
Score
fit_times
Performance of the model
Score
fit_times

In [30]:

```
fig, axes = plt.subplots(3, 2, figsize=(10, 15))

X, y = load_digits(return_X_y=True)

title = "Learning Curves (GaussianNB)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = GaussianNB()
plot_learning_curve(estimator, title, X, y, axes=axes[:, 0], ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

title = r"Learning Curves (SVM, RBF kernel, $\gamma=0.001$)"
# SVC is more expensive so we do a lower number of CV iterations:
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
estimator = SVC(gamma=0.001)
plot_learning_curve(estimator, title, X, y, axes=axes[:, 1], ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()


####
```


Learning Curves (MultinomialNB)
Learning Curves (SVM, RBF kernel, $\gamma = 0.001$)
Training score
Cross-validation score
Score
Training examples
Scalability of the model
fit_times
Performance of the model

Summary:

Gaussian Naive Bayes is better then Multinomial Naive Bayes as accuracy is better.

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. ... Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality.

Multinomial Naïve Bayes uses term frequency i.e. the number of times a given term appears in a document. ... After normalization, term frequency can be used to compute maximum likelihood estimates based on the training data to estimate the conditional probability

The Accuracy of Gaussian Naive Bayes is 79%.

The Accuracy of Multinomial Naive Bayes is 77%.

In [ ]:

In [ ]: