Problem Statement: -

This dataset contains information of users in a social network. This social network has several business clients which can post ads on it. One of the clients has a car company which has just launched a luxury SUV for a ridiculous price. Build a Bernoulli Naïve Bayes model using this dataset and classify which of the users of the social network are going to purchase this luxury SUV. 1 implies that there was a purchase and 0 implies there wasn't a purchase.

Data Pre-processing

In [121]:
```
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

# Loading the data set
caradd = pd.read_csv("D:/360Digi/naive bayes/NB_Car_Ad.csv")

caradd.isnull().sum()
caradd.head()
```

Out[121]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

EDA

In [122]:
```
caradd= caradd.drop("User ID", axis=1)
caradd.head()
```

Out[122]:

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

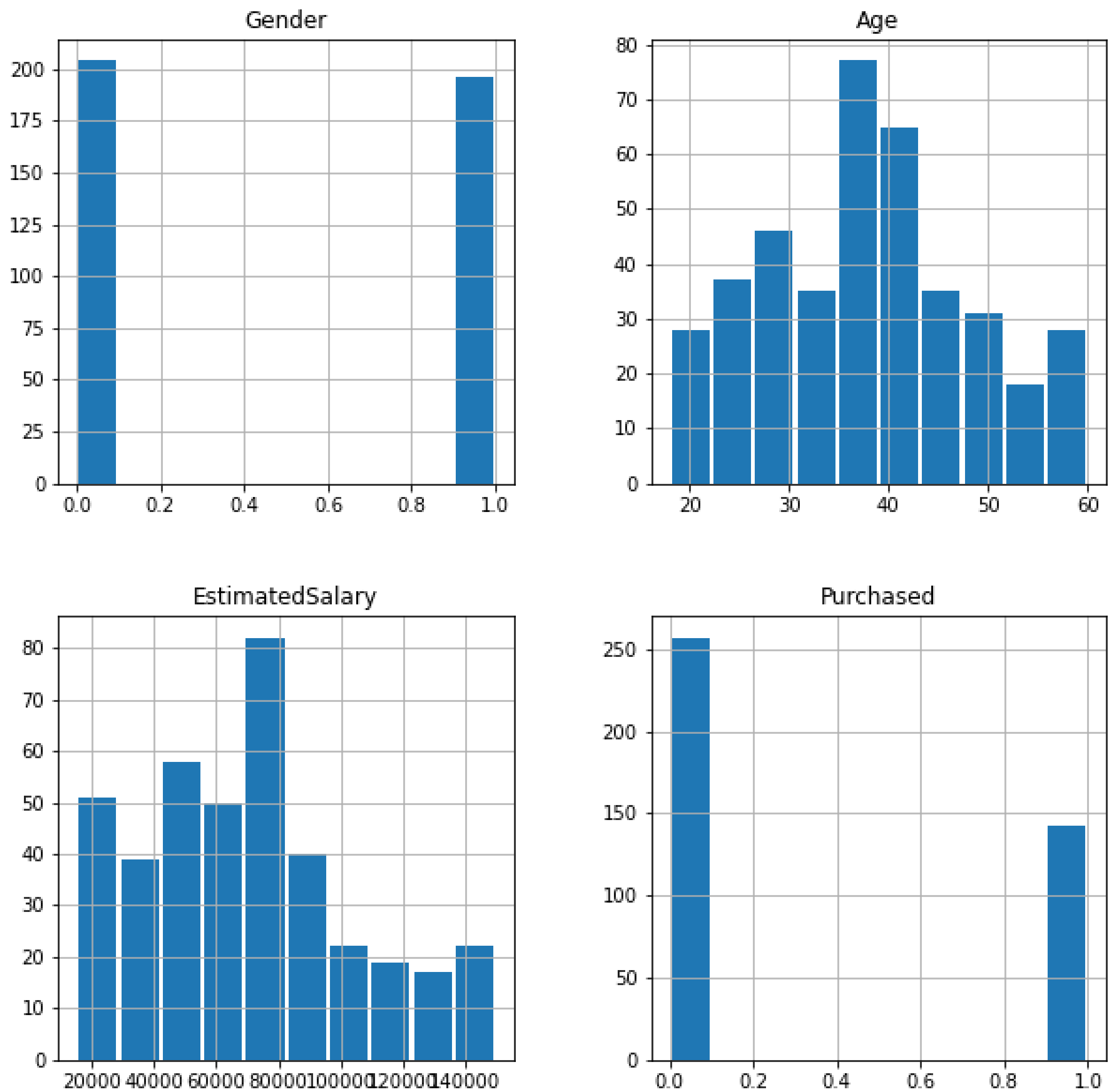
In [98]:
```
caradd.describe()
```

Out[98]:

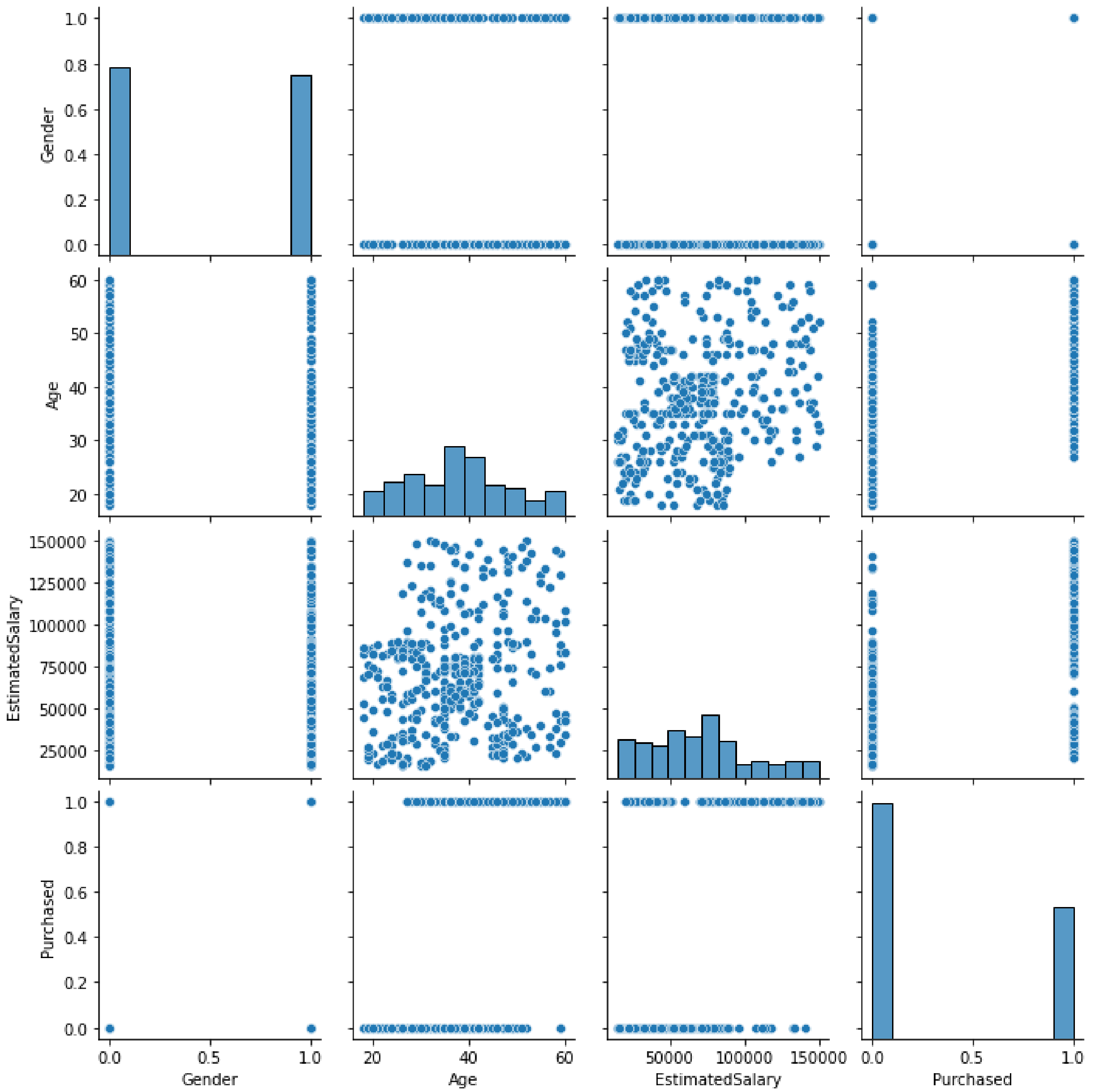
	Gender	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000	400.000000
mean	0.490000	37.655000	69742.500000	0.357500
std	0.500526	10.482877	34096.960282	0.479864
min	0.000000	18.000000	15000.000000	0.000000
25%	0.000000	29.750000	43000.000000	0.000000
50%	0.000000	37.000000	70000.000000	0.000000
75%	1.000000	46.000000	88000.000000	1.000000
max	1.000000	60.000000	150000.000000	1.000000

In [99]:
```
caradd.hist(grid=True, rwidth=0.9, figsize=(10,10))
```

Out[99]:
```
array([[<AxesSubplot:title={'center':'Gender'}>,
        <AxesSubplot:title={'center':'Age'}>],
       [<AxesSubplot:title={'center':'EstimatedSalary'}>,
        <AxesSubplot:title={'center':'Purchased'}>]], dtype=object)
```



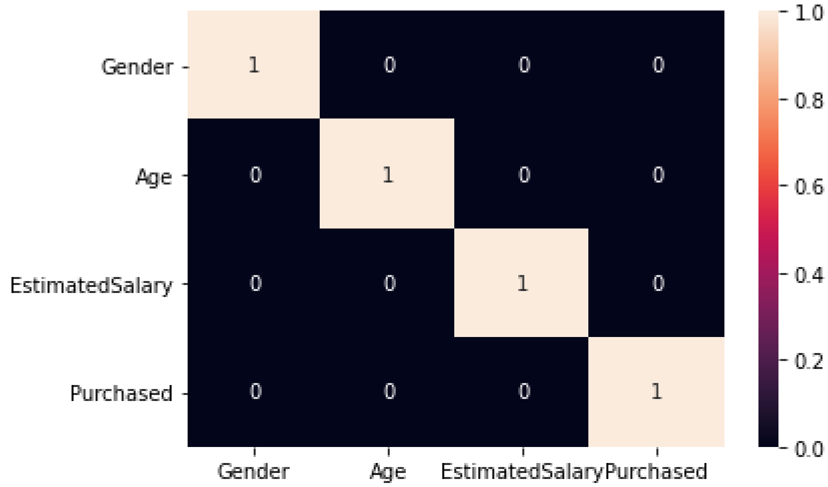
In [100]:
```
import seaborn as sns

sns.pairplot(caradd)
plt.figure(figsize=(8,8))
plt.show()
```



<Figure size 576x576 with 0 Axes>

In [102]:
```
a = caradd.corr(method ='pearson')
sns.heatmap(a>0.85,annot=True)
```

Out[102]: <AxesSubplot:>



In [ ]:

Model Building

In [103]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
caradd['Gender']= labelencoder.fit_transform(caradd['Gender'])
caradd.head()
```

Out[103]:

	Gender	Age	EstimatedSalary	Purchased
0	1	19	19000	0
1	1	35	20000	0
2	0	26	43000	0
3	0	27	57000	0
4	1	19	76000	0

In [ ]:

In [104]:

```
X = caradd.iloc[:, [0,1,2]].values
y = caradd.iloc[:, 3].values

# splitting the data set into training and test set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)

# feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

# Fitting cassifer to trining set
# fitting Bernoulli Bayes to the training set
classifier = BernoulliNB()
classifier.fit(X_train, y_train)
```

Out[104]:

```
BernoulliNB()
```

In [105]:

```
# the model is bulit on training data.Now it is time to see the result on our tes
# prdicting the Test result
y_pred = classifier.predict(X_test)
y_pred

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix", cm)


acc_score = accuracy_score(y_test, y_pred)
print("Accuracy score",acc_score)
####################################################
```

```
confusion matrix [[49  9]
 [ 6 16]]
Accuracy score 0.8125
```

In [106]:

```
# training
x_pred = classifier.predict(X_train)
cm = confusion_matrix(y_train, x_pred)
print("confusion matrix", cm)
acc_score = accuracy_score(y_train, classifier.predict(X_train))
print("Accuracy score",acc_score)
```

```
confusion matrix [[161  38]
 [ 42  79]]
Accuracy score 0.75
```

In [ ]:

In [111]:

```
# train_acc=np.mean(x_pred==y_train)

########################################################################

# The confusion matrix seems to be good. we just need to focus on the diagonal th
# and false positve rates

report = classification_report(y_test, y_pred)
print("REPORT", report)

# A quick recap on the performance metrics:
# Accuracy = Correct predictions / Total predictions
# Precision = True Positives / (True Positives + False Positives); lower precisio
# Recall = True Positives / (True Positives + False Negatives); low recall means
# F1-score = Average between Precision and Recall (weights can be applied if one
# Support = Number of actual observations in that class
```

```
REPORT               precision    recall  f1-score   support

           0       0.89      0.84      0.87        58
           1       0.64      0.73      0.68        22

    accuracy                           0.81        80
   macro avg       0.77      0.79      0.77        80
weighted avg       0.82      0.81      0.82        80
```

In [ ]:

In [ ]:

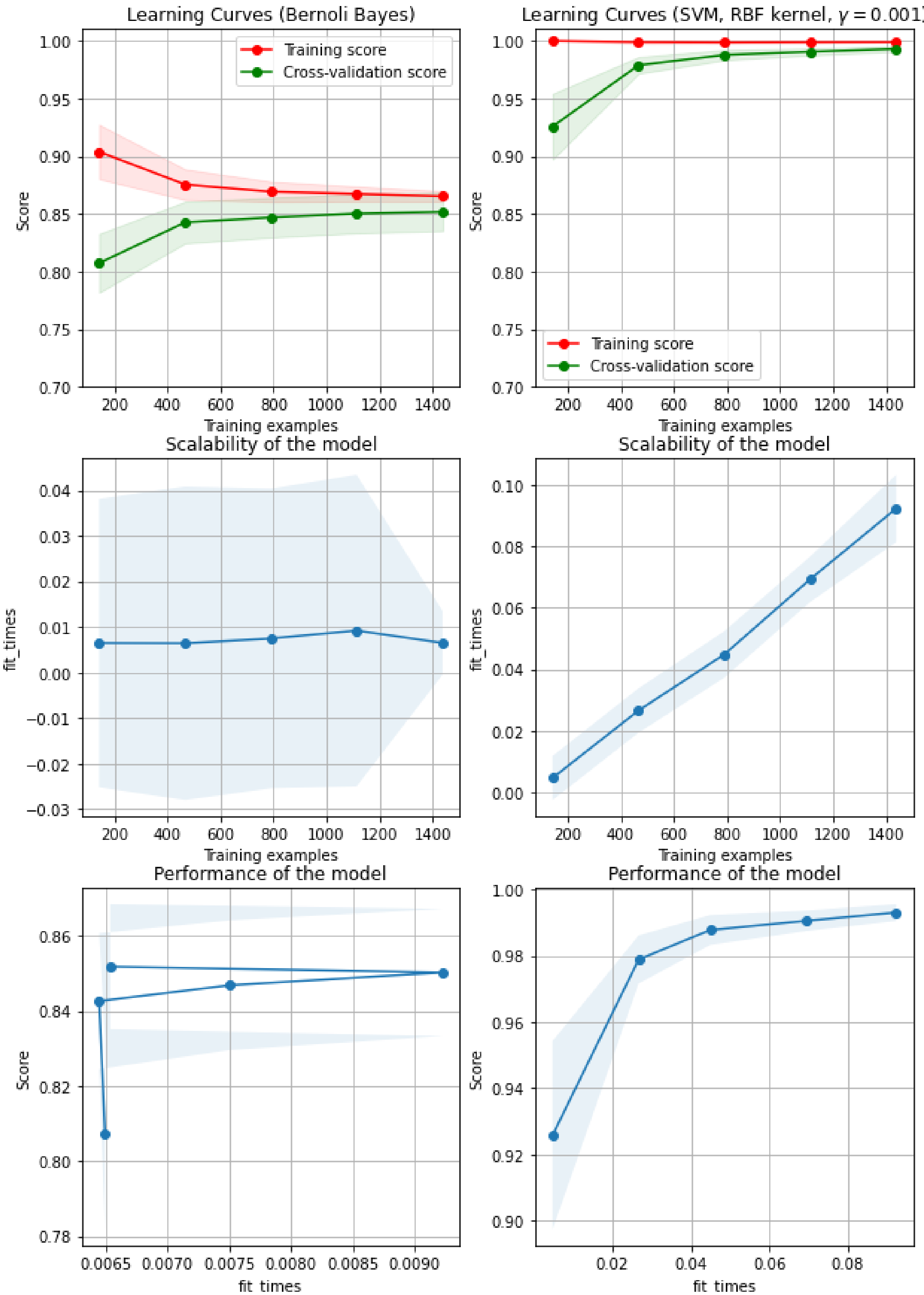
In [ ]:

In [119]:

```
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 (Bernoli Bayes)"
# 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 = BernoulliNB()
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 (Bernoli Bayes)
Training score
Cross-validation score
Score
Training examples
Learning Curves (SVM, RBF kernel, $\gamma = 0.001$)
Scalability of the model
fit_times
Performance of the model

In [ ]:

In [ ]:

In [ ]:

Summary

Predict Classify which of the users of the social network are going to purchase this luxury SUV. 1 implies that there was a purchase and 0 implies there wasn’t a purchase

Advantages of Bernoulli Naive Bayes:

They are extremely fast as compared to other classification models As in Bernoulli Naive Bayes each feature is treated independently with binary values only, it explicitly gives penalty to the model for non-occurrence of any of the features which are necessary for predicting the output y. And the other multinomial variant of Naive Bayes ignores this features instead of penalizing. In case of small amount of data or small documents(for example in text classification), Bernoulli Naive Bayes gives more accurate and precise results as compared to other models. It is fast and are able to make to make real-time predictions It can handle irrelevant features nicely Results are self explanatory Disdvantages of Bernoulli Naive Bayes:

Being a naive(showing a lack of experience) classifier, it sometimes makes a strong assumption based on the shape of data If at times the features are dependent on each other then Naive Bayes assumptions can affect the prediction and accuracy of the model and is sensitive to the given input data If there is a categorial variable which is not present in training dataset, it results in zero frequency problem. This problem can be easily solved by Laplace estimation. From all the above results we can see Bernoulli Naive Bayes is a very good classifier for problems where the features are binary. It gives very good accuracy and can be esaily trained.