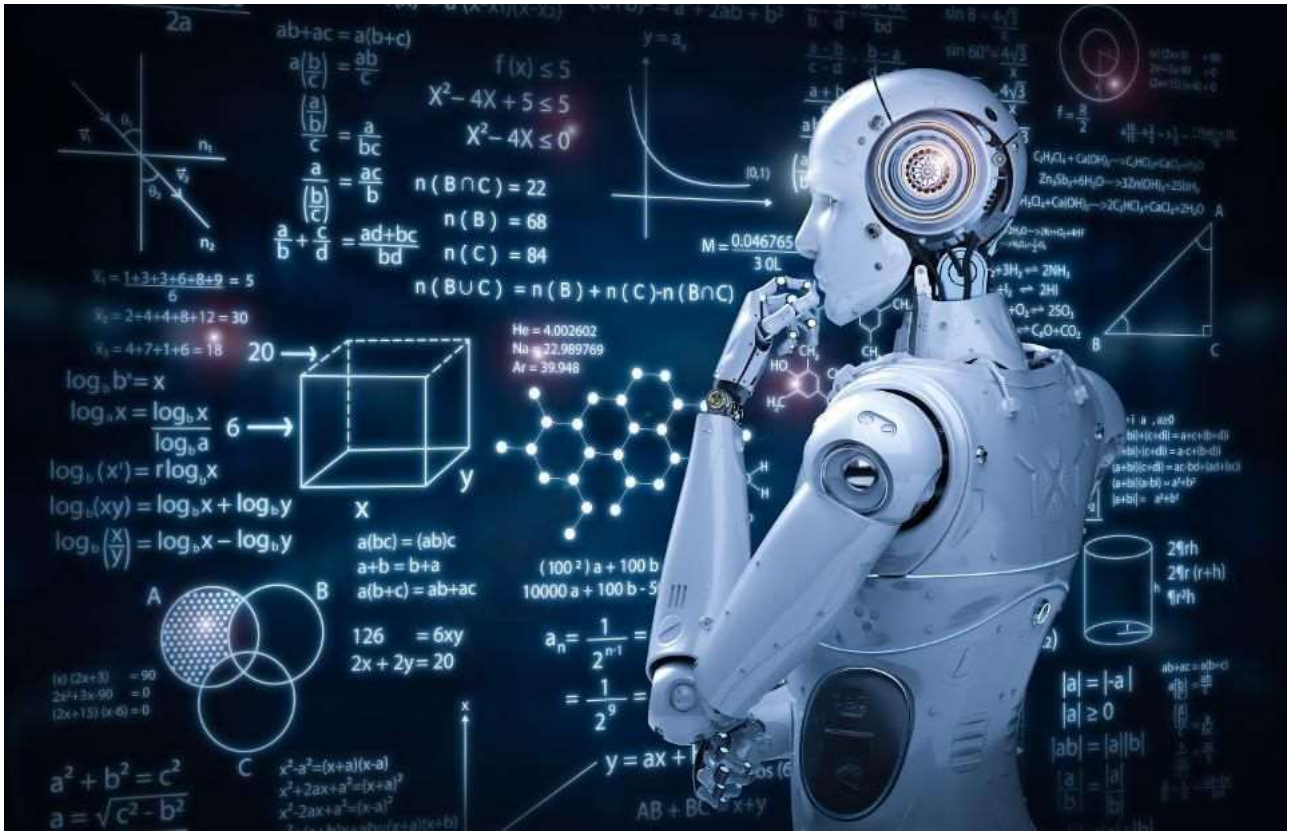
Learn machine learning algorithms in a very easy way

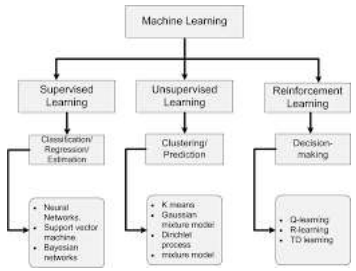
Machine Learning

Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in an autonomous fashion, by feeding them data and information in the form of observations and real-world interactions. There are many algorithms for getting machines to learn, from using basic decision trees to clustering to layers of artificial neural networks, depending on the task you're trying to accomplish and the type and amount of data that you have available.



There are three types of machine learning

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Reinforcement Machine Learning



Supervised Machine Learning

It is a type of learning in which both input and desired output data are provided. Input and output data are labeled for classification to provide a learning basis for future data processing. This algorithm consists of a target/outcome variable (or dependent variable) that is to be predicted from a given set of predictors (independent variables). Using this set of variables, we generate a function that maps inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data.

Unsupervised Machine Learning

Unsupervised learning is the training of an algorithm using information that is neither classified nor labeled, allowing the algorithm to act on that information without guidance. The main idea behind unsupervised learning is to expose the machines to large volumes of varied data and allow them to learn and infer from the data. However, the machines must first be programmed to learn from the data.

Unsupervised learning problems can be further grouped into clustering and association problems:

Clustering: A clustering problem involves discovering inherent groupings in the data, such as grouping customers by purchasing behavior. Association: An association rule learning problem aims to discover rules that describe large portions of your data, such as identifying patterns like people who buy X also tend to buy Y.

Reinforcement Machine Learning

Reinforcement Learning is a type of Machine Learning that enables machines to automatically determine the ideal behavior within a specific context to maximize their performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal. It differs from standard supervised learning in that correct input/output pairs need not be presented, and sub-optimal actions need not be explicitly corrected. Instead, the focus is on performance, which involves finding a balance between exploring uncharted territory and exploiting current knowledge.

Application of Supervised Machine Learning

1. Bioinformatics
2. Quantitative structure
3. Database marketing
4. Handwriting recognition
5. Information retrieval
6. Learning to rank
7. Information extraction
8. Object recognition in computer vision
9. Optical character recognition
10. Spam detection
11. Pattern recognition

Application of Unsupervised Machine Learning

1. Human Behaviour Analysis
2. Social Network Analysis to define groups of friends.
3. Market Segmentation of companies by location, industry, vertical.

4. Organizing computing clusters based on similar event patterns and processes.

Application of Reinforcement Machine Learning

1. Resources management in computer clusters
2. Traffic Light Control
3. Robotics
4. Web System Configuration
5. Personalized Recommendations
6. Deep Learning

We can apply machine learning model by following six steps:

1. Problem Definition
2. Analyse Data
3. Prepare Data
4. Evaluate Algorithm
5. Improve Results
6. Present Results

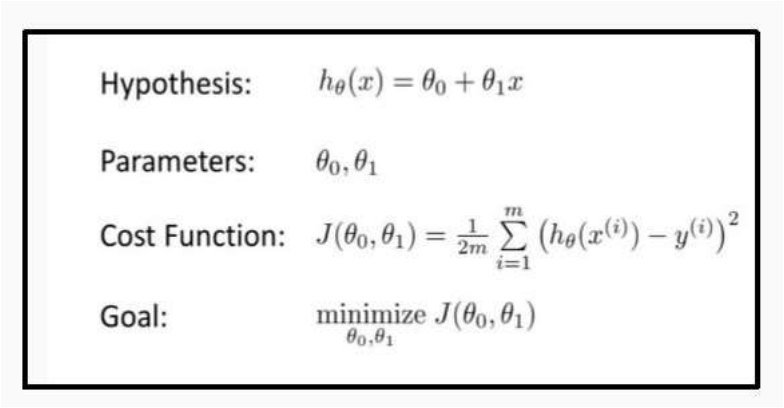
Factors help to choose algorithm

1. Type of algorithm
2. Parametrization
3. Memory size
4. Overfitting tendency
5. Time of learning
6. Time of predicting

Linear Regression

It is a basic and commonly used type of predictive analysis. These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. Y = a + bX where

- Y – Dependent Variable
- a – intercept - Bise
- X – Independent variable



- b – Slope -Weights

Example: University GPA' = (0.675)(High School GPA) + 1.097

Library and Data

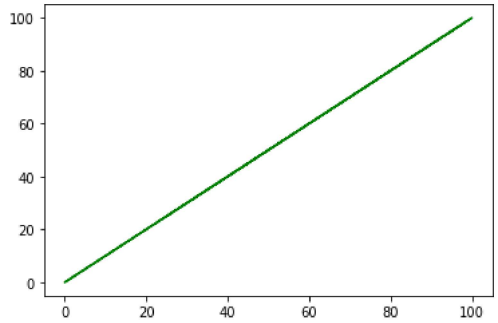
In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
train = pd.read_csv("../input/random-linear-regression/train.csv")
test = pd.read_csv("../input/random-linear-regression/test.csv")
train = train.dropna()
test = test.dropna()
train.head()
```

Out[1]:

	x	y
0	24.0	21.549452
1	50.0	47.464463
2	15.0	17.218656
3	38.0	36.586398
4	87.0	87.288984

Model with plots and accuracy

In [2]:
```
X_train = np.array(train.iloc[:, :-1].values)
y_train = np.array(train.iloc[:, 1].values)
X_test = np.array(test.iloc[:, :-1].values)
y_test = np.array(test.iloc[:, 1].values)
model = LinearRegression(fit_intercept=True, normalize=True,copy_X=True,n_jobs=-1)


model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = model.score(X_test, y_test)

plt.plot(X_train, model.predict(X_train), color='green')
plt.show()
print(accuracy)
```



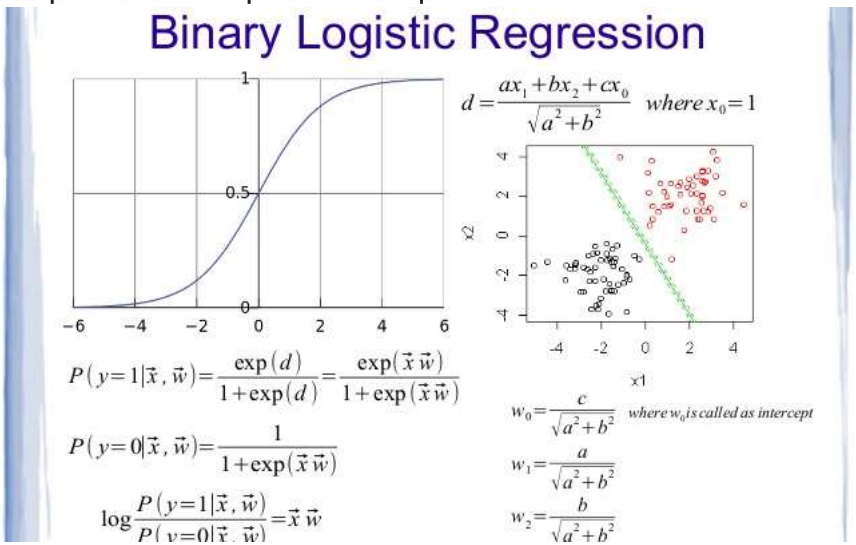

```
0.9888014444327563
```

Logistic Regression

It's a classification algorithm, that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.

- odds= p(x)/(1-p(x)) = probability of event occurrence / probability of not event occurrence

Example- When we have to predict if a student passes or fails in an exam when the number of hours spent studying is given as a feature, the response variable has two values, pass and fail.



Libraries and data

In [3]:
```
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import r2_score
from statistics import mode


train = pd.read_csv("../input/titanic/train.csv")
test  = pd.read_csv('../input/titanic/test.csv')
train.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

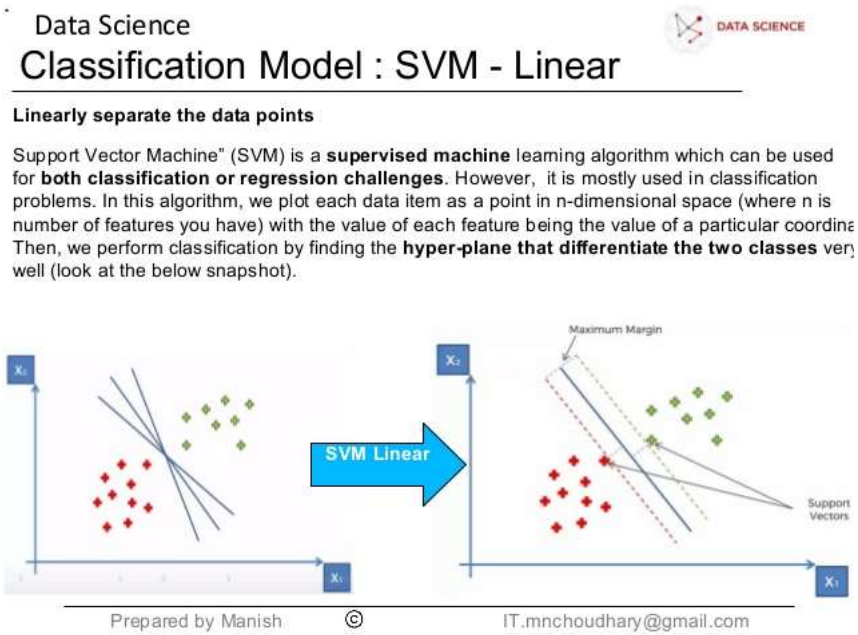
In [4]:
```
ports = pd.get_dummies(train.Embarked , prefix='Embarked')
train = train.join(ports)
train.drop(['Embarked'], axis=1, inplace=True)
train.Sex = train.Sex.map({'male':0, 'female':1})
y = train.Survived.copy()
X = train.drop(['Survived'], axis=1)
X.drop(['Cabin'], axis=1, inplace=True)
X.drop(['Ticket'], axis=1, inplace=True)
X.drop(['Name'], axis=1, inplace=True)
X.drop(['PassengerId'], axis=1, inplace=True)
X.Age.fillna(X.Age.median(), inplace=True)
```

Model and Accuracy

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)
from sklearn.linear_model import LogisticRegression
#linear_model.LogisticRegression(penalty='l2',dual=False,tol=0.0001,C=1.0,fit_intercept=True,intercept_scaling=1,
#  class_weight=None,random_state=None,solver='warn',max_iter=100,
#   multi_class='warn', verbose=0,warm_start=False, n_jobs=None)


model = LogisticRegression(max_iter = 500000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = model.score(X_test, y_test)
print(accuracy)
```

```
0.8251121076233184
```

Support Vector Machine



Support Vector Machines are perhaps one of the most popular and talked about machine learning algorithms.It is primarily a classier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables

Example: One class is linearly separable from the others like if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates

Libraries and Data

In [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
data_svm = pd.read_csv("../input/svm-classification/UniversalBank.csv")
data_svm.head()
```

Out[6]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

Model and Accuracy

In [7]:
```
X = data_svm.iloc[:,1:13].values
y = data_svm.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

'''
```

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True,
                probability=False, tol=0.001, cache_size=200, class_weight=None,verbose=False,
                max_iter=-1, decision_function_shape='ovr', random_state=None)
'''

classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
accuracies.mean()
```

Out[7]: 0.7077333333333333

Naive Bayes Algorithm

Naive Bayes

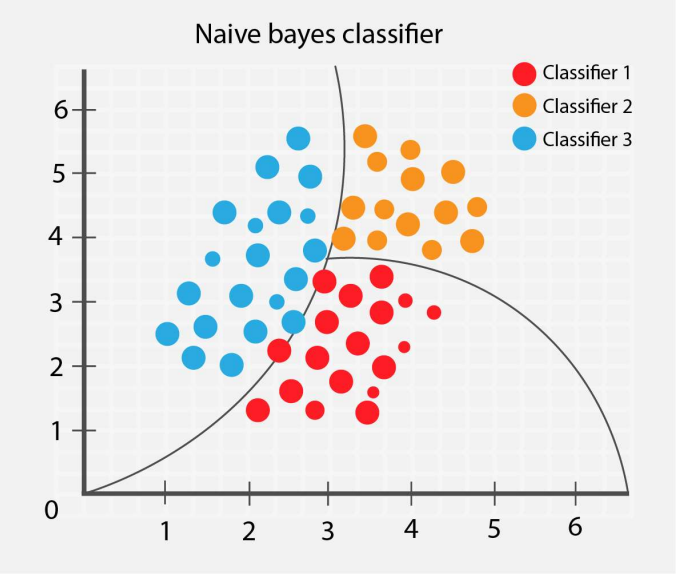
thatware.co

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



A naive Bayes classifier is not a single algorithm, but a family of machine learning algorithms which use probability theory to classify data with an assumption of independence between predictors It is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods

Example: Emails are given and we have to find the spam emails from that.A spam filter looks at email messages for certain key words and puts them in a spam folder if they match.

Libraries and Data

In [8]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
data = pd.read_csv('../input/classification-suv-dataset/Social_Network_Ads.csv')
data_nb = data
data_nb.head()
```

Out[8]:

	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

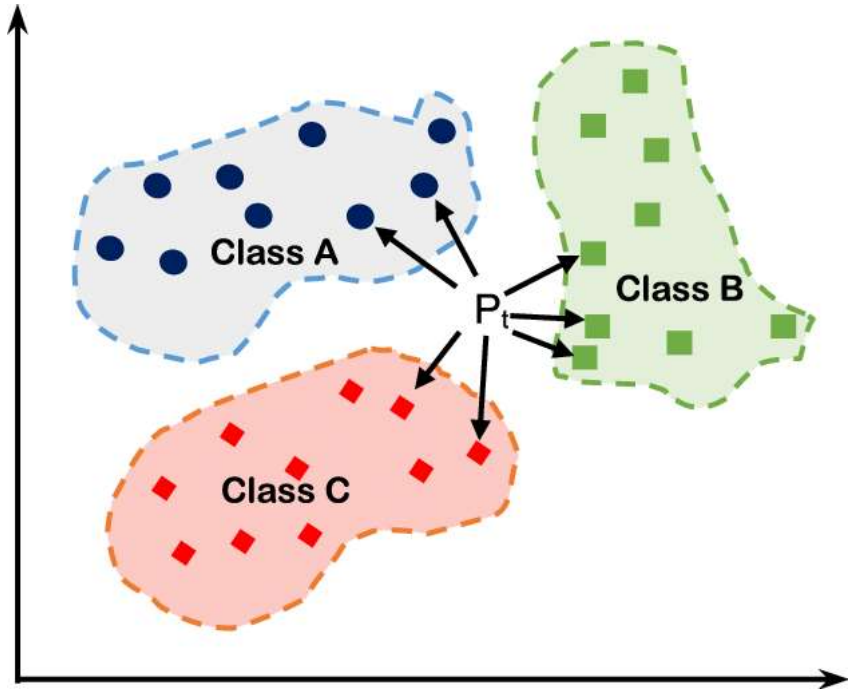
Model and Accuracy

In [9]:
```
X = data_nb.iloc[:, [2,3]].values
y = data_nb.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

'''
#sklearn.naive_bayes.GaussianNB(priors=None, var_smoothing=1e-09)
'''

classifier=GaussianNB()
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9125
```

KNN

KNN does not learn any model. and stores the entire training data set which it uses as its representation.The output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction



Example: Should the bank give a loan to an individual? Would an individual default on his or her loan? Is that person closer in characteristics to people who defaulted or did not default on their loans?

Libraries and Data

In [10]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = pd.read_csv("../input/iris/Iris.csv")
knn.head()
```

Out[10]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

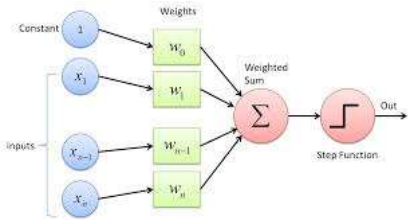
Model and Accuracy

In [11]:
```
X = knn.iloc[:, [1,2,3,4]].values
y = knn.iloc[:, 5].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


'''
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
                                       p=2, metric='minkowski', metric_params=None,n_jobs=None)
'''
```

```
classifier=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
acc=accuracy_score(y_test, y_pred)
print( acc)
```

```
1.0
```

Perceptron



It is single layer neural network and used for classification

In [12]:
```
from sklearn.linear_model import Perceptron
from sklearn.neighbors import KNeighborsClassifier
p = pd.read_csv("../input/iris/Iris.csv")
p.head()
```

Out[12]:

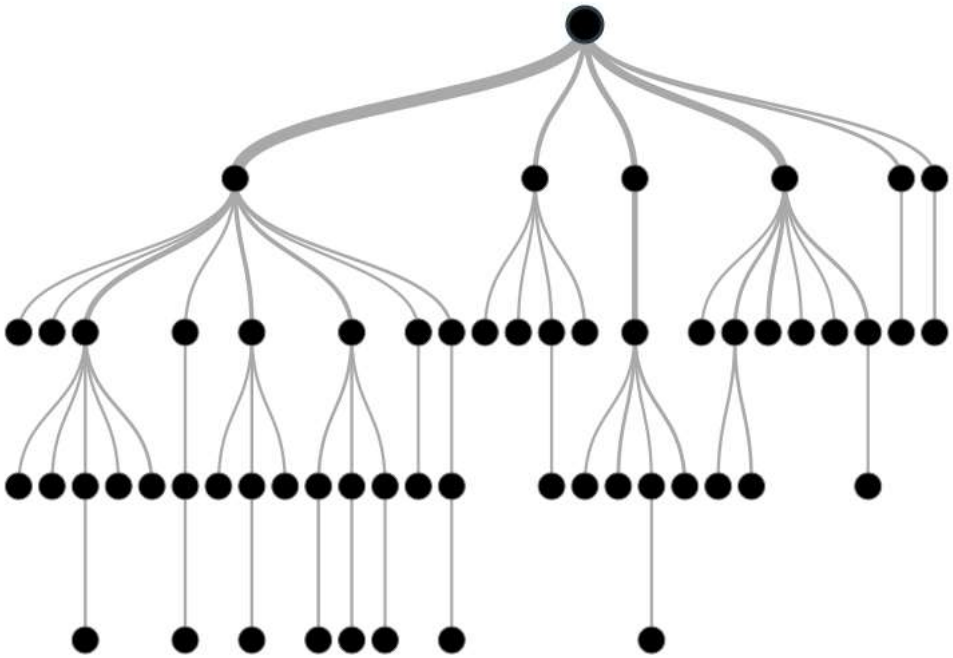
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [13]:
```
X = p.iloc[:, [1,2,3,4]].values
y = p.iloc[:, 5].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
'''
classifier=Perceptron(penalty=None, alpha=0.0001, fit_intercept=True, max_iter=None, tol=None, shuffle=True,
verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1,
n_iter_no_change=5, class_weight=None, warm_start=False, n_iter=None)
'''

classifier=Perceptron()
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9666666666666667
```

Decision Tree

Decision tree algorithm is classification algorithm under supervised machine learning and it is simple to understand and use in data.The idea of Decision tree is to split the big data(root) into smaller(leaves)



In [14]:
```
from sklearn.tree import DecisionTreeClassifier
dt = data
dt.head()
```

Out[14]:

	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

In [15]:
```
X = dt.iloc[:, [2,3]].values
y = dt.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


'''
sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None,min_samples_split=2,
                                    min_samples_leaf=1,min_weight_fraction_leaf=0.0,max_features=None,
```

```
                                random_state=None, max_leaf_nodes=None,min_impurity_decrease=0.0,
                                min_impurity_split=None, class_weight=None,presort=False)
'''


classifier=DecisionTreeClassifier(criterion="entropy",random_state=0)
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9
```

Extra Tree

Library and Data

In [16]:
```
from sklearn.ensemble import ExtraTreesClassifier
et = data
et.head()
```

Out[16]:

	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

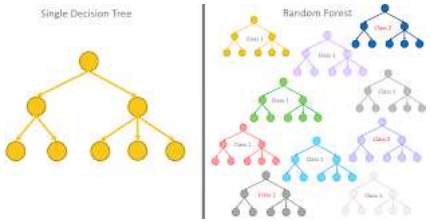
Model and Accuracy

In [17]:
```
X = et.iloc[:, [2,3]].values
y = et.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


''' ExtraTreesRegressor(n_estimators=10, max_features=32, random_state=0)'''

classifier=ExtraTreesClassifier(criterion="entropy",random_state=0)
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9
```

Random Forest

Random forest is collection of tress(forest) and it builds multiple decision trees and merges them together to get a more accurate and stable prediction.It can be used for both classification and regression problems.



Example: Suppose we have a bowl of 100 unique numbers from 0 to 99. We want to select a random sample of numbers from the bowl. If we put the number back in the bowl, it may be selected more than once.

Libraries and Data

In [18]:
```
from sklearn.ensemble import RandomForestClassifier
rf = pd.read_csv("../input/mushroom-classification/mushrooms.csv")
rf.head()
```

Out[18]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g

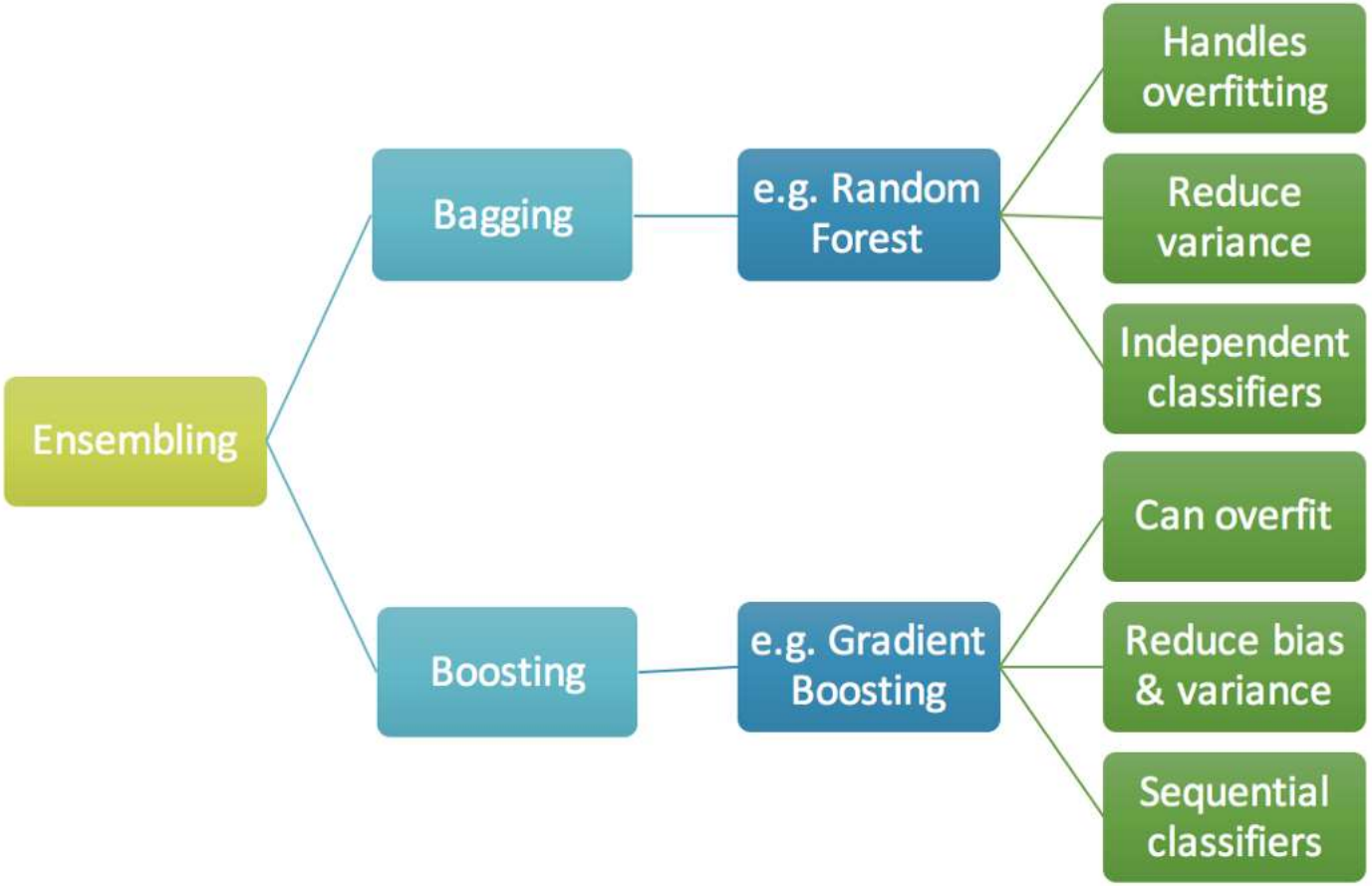
5 rows × 23 columns

Model and Accuracy

In [19]:
```
X = rf.drop('class', axis=1)
y = rf['class']
X = pd.get_dummies(X)
y = pd.get_dummies(y)
X_train, X_test, y_train, y_test = train_test_split(X, y)


'''
ensemble.RandomForestClassifier(n_estimators='warn', criterion='gini', max_depth=None,
                                min_samples_split=2, min_samples_leaf=1,min_weight_fraction_leaf=0.0,
                                max_features='auto',max_leaf_nodes=None,min_impurity_decrease=0.0,
                                min_impurity_split=None, bootstrap=True,oob_score=False, n_jobs=None,
                                random_state=None, verbose=0,warm_start=False, class_weight=None)
'''


model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=1)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Out[19]:
```
1.0
```

Gradient Boosting



Gradient boosting is an alogithm under supervised machine learning, boosting means converting weak into strong. In this new tree is boosted over the previous tree

Libraries and Data

In [20]:
```
from sklearn.ensemble import GradientBoostingClassifier
gb = data
gb.head()
```

Out[20]:

	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

Model and Accuracy

In [21]:
```
X = gb.iloc[:, [2,3]].values
y = gb.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


'''
ensemble.GradientBoostingClassifier(loss='deviance', learning_rate=0.1,n_estimators=100, subsample=1.0,
                                    criterion='friedman_mse',min_samples_split=2,min_samples_leaf=1,
                                    min_weight_fraction_leaf=0.0,max_depth=3,min_impurity_decrease=0.0,
                                    min_impurity_split=None,init=None, random_state=None,max_features=None,
                                    verbose=0, max_leaf_nodes=None,warm_start=False, presort='auto',
                                    validation_fraction=0.1,n_iter_no_change=None, tol=0.0001)
'''


gbk = GradientBoostingClassifier()
gbk.fit(X_train, y_train)
pred = gbk.predict(X_test)
acc=accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9
```

LDA

A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.Itis used in statistics, pattern recognition, and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

Library and Data

In [22]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = data
lda.head()
```

Out[22]:

	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

Model and Accuracy

In [23]:
```
X = gb.iloc[:, [2,3]].values
y = gb.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
Model=LinearDiscriminantAnalysis()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
accuracy is  0.9125
```

CNN

Library and Data

In [24]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
import tensorflow as tf
train_data = pd.read_csv("../input/digit-recognizer/train.csv")
test_data = pd.read_csv("../input/digit-recognizer/test.csv")
train_data.head()
```

```
Using TensorFlow backend.
```

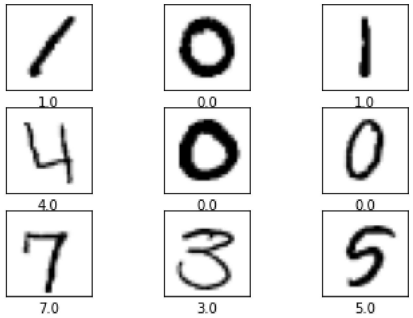
Out[24]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	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	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Preprocessing and Data Split

In [25]:
```
X = np.array(train_data.drop("label", axis=1)).astype('float32')
y = np.array(train_data['label']).astype('float32')
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X[i].reshape(28, 28), cmap=plt.cm.binary)
    plt.xlabel(y[i])
plt.show()

X = X / 255.0
X = X.reshape(-1, 28, 28, 1)
y = to_categorical(y)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
X_test = np.array(test_data).astype('float32')
X_test = X_test / 255.0
X_test = X_test.reshape(-1, 28, 28, 1)
plt.figure(figsize=(10,10))
```



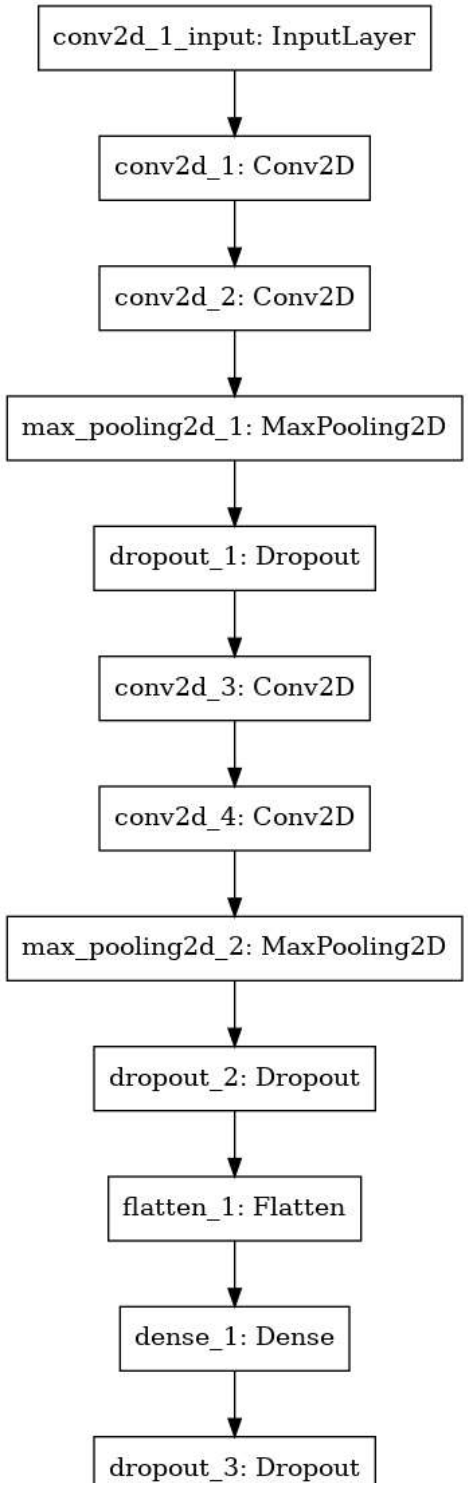
Out[25]:
```
<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>
```

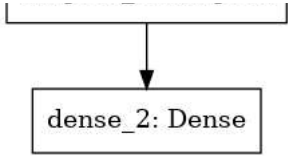
Model

In [26]:
```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
model.summary()
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model1.png')
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               803072
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 887,530
Trainable params: 887,530
Non-trainable params: 0
_________________________________________________________________
```

Out[26]:


conv2d_1_input: InputLayer
conv2d_1: Conv2D
conv2d_2: Conv2D
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_3: Conv2D
conv2d_4: Conv2D
max_pooling2d_2: MaxPooling2D
dropout_2: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_3: Dropout

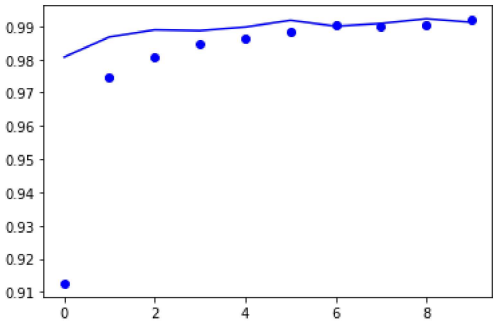


Compiling model

In [27]:
```
#increse to epochs to 30 for better accuracy
model.compile(optimizer='adam', loss="categorical_crossentropy", metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=10, batch_size=85, validation_data=(X_val, y_val))
```

```
Train on 33600 samples, validate on 8400 samples
Epoch 1/10
33600/33600 [==============================] - 6s 192us/step - loss: 0.2713 - accuracy: 0.9126 - val_loss: 0.0646 - val_accuracy: 0.9807
Epoch 2/10
33600/33600 [==============================] - 3s 91us/step - loss: 0.0834 - accuracy: 0.9747 - val_loss: 0.0444 - val_accuracy: 0.9868
Epoch 3/10
33600/33600 [==============================] - 3s 91us/step - loss: 0.0628 - accuracy: 0.9807 - val_loss: 0.0372 - val_accuracy: 0.9889
Epoch 4/10
33600/33600 [==============================] - 3s 93us/step - loss: 0.0504 - accuracy: 0.9848 - val_loss: 0.0377 - val_accuracy: 0.9887
Epoch 5/10
33600/33600 [==============================] - 3s 91us/step - loss: 0.0425 - accuracy: 0.9864 - val_loss: 0.0377 - val_accuracy: 0.9898
Epoch 6/10
33600/33600 [==============================] - 3s 91us/step - loss: 0.0386 - accuracy: 0.9884 - val_loss: 0.0331 - val_accuracy: 0.9918
Epoch 7/10
33600/33600 [==============================] - 3s 92us/step - loss: 0.0332 - accuracy: 0.9904 - val_loss: 0.0308 - val_accuracy: 0.9900
Epoch 8/10
33600/33600 [==============================] - 3s 91us/step - loss: 0.0326 - accuracy: 0.9898 - val_loss: 0.0326 - val_accuracy: 0.9908
Epoch 9/10
33600/33600 [==============================] - 3s 101us/step - loss: 0.0301 - accuracy: 0.9904 - val_loss: 0.0278 - val_accuracy: 0.9923
Epoch 10/10
33600/33600 [==============================] - 3s 93us/step - loss: 0.0262 - accuracy: 0.9918 - val_loss: 0.0296 - val_accuracy: 0.9912
```

In [28]:
```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.show()

print(model.evaluate(X_val, y_val))
```




```
8400/8400 [==============================] - 1s 73us/step
[0.02961143438459008, 0.991190493106842]
```

LSTM

LSTM blocks are part of a recurrent neural network structure. Recurrent neural networks are made to utilize certain types of artificial memory processes that can help these artificial intelligence programs to more effectively imitate human thought.It is capable of learning order dependence LSTM can be used for machine translation, speech recognition, and more.

Library and Data

In [30]:
```
import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
lstm = pd.read_csv("../input/nyse/prices.csv")
lstm = lstm[lstm['symbol']=="NFLX"]
lstm['date'] = pd.to_datetime(lstm['date'])
lstm.set_index('date',inplace=True)
lstm = lstm.reset_index()
lstm.head()
```

Out[30]:

	date	symbol	open	close	low	high	volume
0	2010-01-04	NFLX	55.519999	53.479999	52.960001	55.730000	17239600.0
1	2010-01-05	NFLX	53.570001	51.510001	50.810001	53.599998	23753100.0
2	2010-01-06	NFLX	51.530001	53.319999	50.380002	53.710001	23290400.0
3	2010-01-07	NFLX	54.120000	52.400001	52.240001	54.300001	9955400.0
4	2010-01-08	NFLX	52.490000	53.300002	52.260001	54.199999	8180900.0

Preprocessing

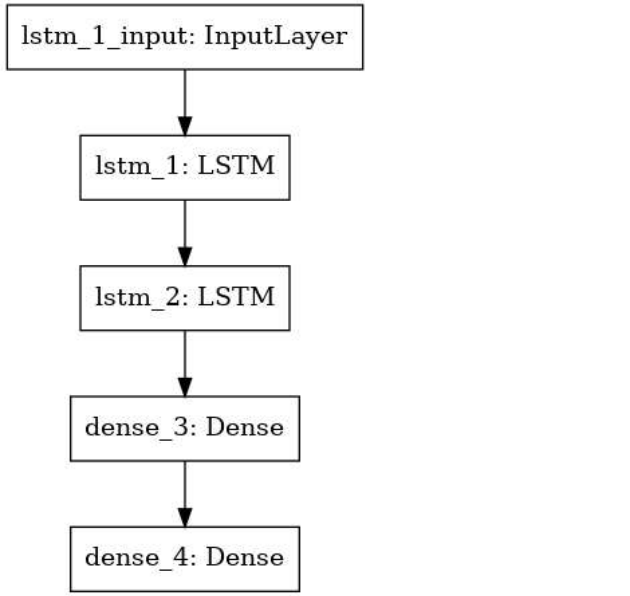
In [31]:
```
data = lstm.filter(['close'])
dataset = data.values
training_data_len = math.ceil(len(dataset)*.75)
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
train_data = scaled_data[0:training_data_len, :]
x_train = []
y_train = []
for i in range(60,len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i,0])
x_train,y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))
```

Model

In [32]:
```
model =Sequential()
model.add(LSTM(64,return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(64, return_sequences= False))
model.add(Dense(32))
model.add(Dense(1))
model.summary()
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model1.png')
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 60, 64)            16896
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 52,033
Trainable params: 52,033
Non-trainable params: 0
_________________________________________________________________
```

Out[32]:



Compiling Model

In [33]:

```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train,y_train, batch_size=85, epochs=20)
```

```
Epoch 1/20
1262/1262 [==============================] - 2s 2ms/step - loss: 0.0217
Epoch 2/20
1262/1262 [==============================] - 1s 1ms/step - loss: 0.0031
Epoch 3/20
1262/1262 [==============================] - 2s 1ms/step - loss: 0.0014
Epoch 4/20
1262/1262 [==============================] - 1s 1ms/step - loss: 0.0010
Epoch 5/20
1262/1262 [==============================] - 1s 1ms/step - loss: 9.0542e-04
Epoch 6/20
1262/1262 [==============================] - 1s 1ms/step - loss: 8.7967e-04
Epoch 7/20
1262/1262 [==============================] - 1s 1ms/step - loss: 8.4264e-04
Epoch 8/20
1262/1262 [==============================] - 1s 1ms/step - loss: 8.3065e-04
Epoch 9/20
1262/1262 [==============================] - 1s 1ms/step - loss: 7.9484e-04
Epoch 10/20
1262/1262 [==============================] - 1s 1ms/step - loss: 7.7333e-04
Epoch 11/20
1262/1262 [==============================] - 1s 1ms/step - loss: 7.6124e-04
Epoch 12/20
1262/1262 [==============================] - 1s 1ms/step - loss: 7.3006e-04
Epoch 13/20
1262/1262 [==============================] - 1s 1ms/step - loss: 7.0877e-04
Epoch 14/20
1262/1262 [==============================] - 1s 1ms/step - loss: 6.8812e-04
Epoch 15/20
1262/1262 [==============================] - 2s 2ms/step - loss: 6.5516e-04
Epoch 16/20
1262/1262 [==============================] - 1s 1ms/step - loss: 6.4585e-04
Epoch 17/20
1262/1262 [==============================] - 1s 1ms/step - loss: 6.2556e-04
Epoch 18/20
1262/1262 [==============================] - 1s 1ms/step - loss: 5.8959e-04
Epoch 19/20
1262/1262 [==============================] - 1s 1ms/step - loss: 5.7692e-04
Epoch 20/20
1262/1262 [==============================] - 1s 1ms/step - loss: 5.5900e-04
```

Out[33]:
```
<keras.callbacks.callbacks.History at 0x7cf563f028d0>
```

Prediction and Accuracy

In [34]:
```
test_data= scaled_data[training_data_len-60:, :]
x_test = []
y_test = dataset[training_data_len:,:]
for i in range(60,len(test_data)):
    x_test.append(test_data[i-60:i,0])
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
rmse = np.sqrt(np.mean(predictions - y_test)**2)
rmse
```

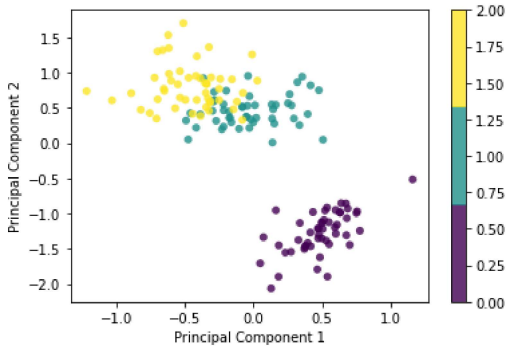
Out[34]:
```
3.9742272483293015
```

Principle Component Analysis

It's an important method for dimension reduction.It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible and to visualise high-dimensional data, it also reduces noise and finally makes other algorithms to work better because we are injecting fewer inputs.

- Example: When we have to bring out strong patterns in a data set or to make data easy to explore and visualize

In [35]:
```
from sklearn.datasets import make_blobs
from sklearn import datasets
class PCA:
  def __init__(self, n_components):
    self.n_components = n_components
    self.components = None
    self.mean = None

  def fit(self, X):
    self.mean = np.mean(X, axis=0)
    X = X - self.mean
    cov = np.cov(X.T)

    evalue, evector = np.linalg.eig(cov)

    eigenvectors = evector.T
    idxs = np.argsort(evalue)[::-1]

    evalue = evalue[idxs]
    evector = evector[idxs]
    self.components = evector[0:self.n_components]

  def transform(self, X):
    #project data
    X = X - self.mean
    return(np.dot(X, self.components.T))

data = datasets.load_iris()
X = data.data
y = data.target

pca = PCA(2)
pca.fit(X)
X_projected = pca.transform(X)



x1 = X_projected[:,0]
x2 = X_projected[:,1]

plt.scatter(x1,x2,c=y,edgecolor='none',alpha=0.8,cmap=plt.cm.get_cmap('viridis',3))
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar()
plt.show()
```



Apriori

It is a categorisation algorithm attempts to operate on database records, particularly transactional records, or records including certain numbers of fields or items.It is mainly used for sorting large amounts of data. Sorting data often occurs because of association rules.

- Example: To analyse data for frequent if/then patterns and using the criteria support and confidence to identify the most important relationships.

In [36]:
```
df = pd.read_csv('../input/supermarket/GroceryStoreDataSet.csv',names=['products'],header=None)
data = list(df["products"].apply(lambda x:x.split(',')))
data
```

Out[36]:
```
[['MILK', 'BREAD', 'BISCUIT'],
 ['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['JAM', 'MAGGI', 'BREAD', 'MILK'],
 ['MAGGI', 'TEA', 'BISCUIT'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['MAGGI', 'TEA', 'CORNFLAKES'],
 ['MAGGI', 'BREAD', 'TEA', 'BISCUIT'],
 ['JAM', 'MAGGI', 'BREAD', 'TEA'],
 ['BREAD', 'MILK'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'COCK'],
 ['BREAD', 'SUGER', 'BISCUIT'],
 ['COFFEE', 'SUGER', 'CORNFLAKES'],
 ['BREAD', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['TEA', 'MILK', 'COFFEE', 'CORNFLAKES']]
```

In [37]:
```
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data,columns=te.columns_)
df1 = apriori(df,min_support=0.01,use_colnames=True)
df1.head()
```

Out[37]:

	support	itemsets
0	0.35	(BISCUIT)
1	0.20	(BOURNVITA)
2	0.65	(BREAD)
3	0.15	(COCK)
4	0.40	(COFFEE)

Prophet

Prophet is an extremely easy tool for analysts to produce reliable forecasts

1. Prophet only takes data as a dataframe with a ds (datestamp) and y (value we want to forecast) column. So first, let's convert the dataframe to the appropriate format.
2. Create an instance of the Prophet class and then fit our dataframe to it.
3. Create a dataframe with the dates for which we want a prediction to be made with make_future_dataframe(). Then specify the number of days to forecast using the periods parameter.

4. Call predict to make a prediction and store it in the forecast dataframe. What's neat here is that you can inspect the dataframe and see the predictions as well as the lower and upper boundaries of the uncertainty interval.

Library and Data

In [38]:
```
import plotly.offline as py
import plotly.express as px
from fbprophet import Prophet
from fbprophet.plot import plot_plotly, add_changepoints_to_plot

pred = pd.read_csv("../input/coronavirus-2019ncov/covid-19-all.csv")
pred = pred.fillna(0)
predgrp = pred.groupby("Date")[["Confirmed","Recovered","Deaths"]].sum().reset_index()
pred_cnfrm = predgrp.loc[:,["Date","Confirmed"]]
pr_data = pred_cnfrm
pr_data.columns = ['ds','y']
pr_data.head()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3063: DtypeWarning:

Columns (0,1) have mixed types. Specify dtype option on import or set low_memory=False.
```

Out[38]:

	ds	y
0	2020-01-22	557.0
1	2020-01-23	1097.0
2	2020-01-24	941.0
3	2020-01-25	1437.0
4	2020-01-26	2118.0

Model and Forecast

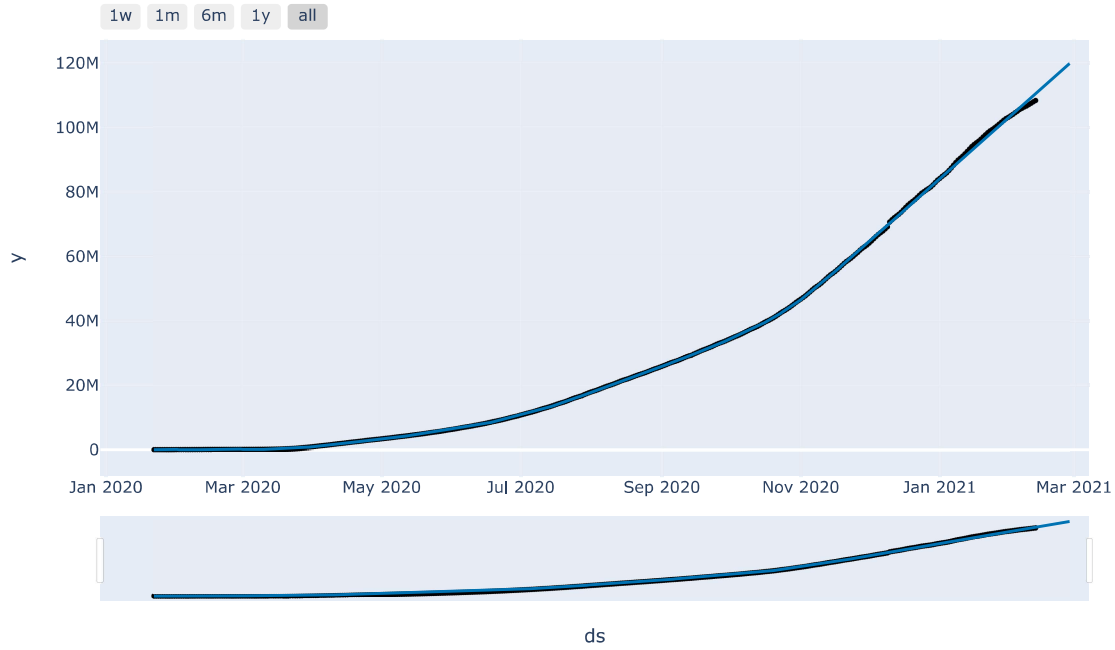
In [39]:
```
m=Prophet()
m.fit(pr_data)
future=m.make_future_dataframe(periods=15)
forecast=m.predict(future)
forecast
```

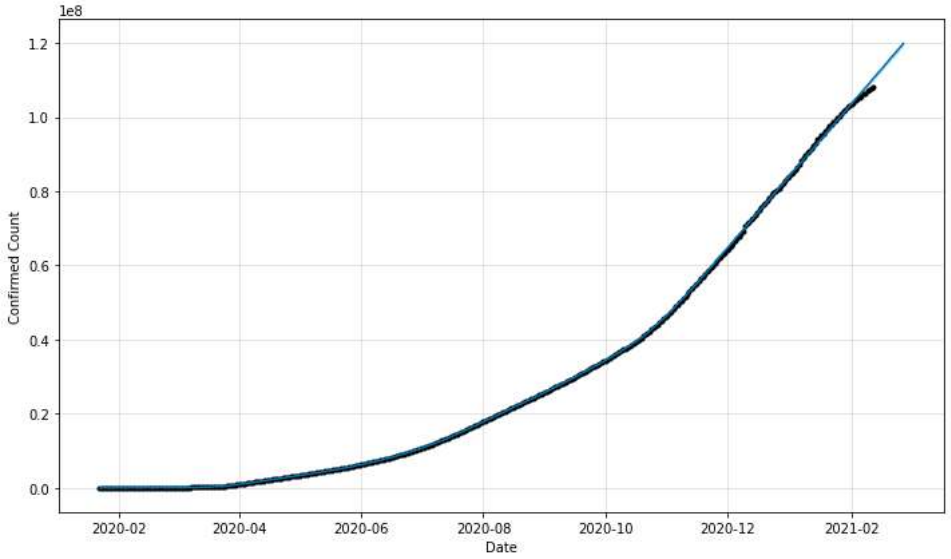
Out[39]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly	weekly_lower	weekly_upper	multiplicative_terms	multiplica
0	2020-01-22	-1.102811e+04	-4.432776e+05	3.641621e+05	-1.102811e+04	-1.102811e+04	-24354.621123	-24354.621123	-24354.621123	-24354.621123	-24354.621123	-24354.621123	0.0	
1	2020-01-23	-8.542636e+03	-4.112285e+05	3.997316e+05	-8.542636e+03	-8.542636e+03	7597.594845	7597.594845	7597.594845	7597.594845	7597.594845	7597.594845	0.0	
2	2020-01-24	-6.057162e+03	-4.282336e+05	4.078674e+05	-6.057162e+03	-6.057162e+03	24185.668431	24185.668431	24185.668431	24185.668431	24185.668431	24185.668431	0.0	
3	2020-01-25	-3.571688e+03	-3.450811e+05	4.357830e+05	-3.571688e+03	-3.571688e+03	56868.656272	56868.656272	56868.656272	56868.656272	56868.656272	56868.656272	0.0	
4	2020-01-26	-1.086214e+03	-4.022589e+05	4.423423e+05	-1.086214e+03	-1.086214e+03	10568.490321	10568.490321	10568.490321	10568.490321	10568.490321	10568.490321	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
398	2021-02-23	1.172838e+08	1.167996e+08	1.177215e+08	1.171539e+08	1.174014e+08	-41108.998576	-41108.998576	-41108.998576	-41108.998576	-41108.998576	-41108.998576	0.0	
399	2021-02-24	1.179104e+08	1.174113e+08	1.183454e+08	1.177549e+08	1.180576e+08	-24354.621123	-24354.621123	-24354.621123	-24354.621123	-24354.621123	-24354.621123	0.0	
400	2021-02-25	1.185370e+08	1.181009e+08	1.190034e+08	1.183533e+08	1.187112e+08	7597.594845	7597.594845	7597.594845	7597.594845	7597.594845	7597.594845	0.0	
401	2021-02-26	1.191637e+08	1.186935e+08	1.196907e+08	1.189454e+08	1.193734e+08	24185.668431	24185.668431	24185.668431	24185.668431	24185.668431	24185.668431	0.0	
402	2021-02-27	1.197903e+08	1.192833e+08	1.203335e+08	1.195215e+08	1.200276e+08	56868.656272	56868.656272	56868.656272	56868.656272	56868.656272	56868.656272	0.0	

403 rows × 16 columns

In [40]:
```
fig = plot_plotly(m, forecast)
py.iplot(fig)

fig = m.plot(forecast,xlabel='Date',ylabel='Confirmed Count')
```


1w
1m
6m
1y
all
120M
100M
80M
60M
40M
20M
0
y
Jan 2020
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
Mar 2021
ds


1e8
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Confirmed Count
2020-02
2020-04
2020-06
2020-08
2020-10
2020-12
2021-02
Date

Arima

Library and Data

In [41]:
```
import datetime
from statsmodels.tsa.arima_model import ARIMA
ar = pd.read_csv("../input/competitive-data-science-predict-future-sales/sales_train.csv")
ar.date=ar.date.apply(lambda x:datetime.datetime.strptime(x, '%d.%m.%Y'))
ar=ar.groupby(["date_block_num"])["item_cnt_day"].sum()
ar.index=pd.date_range(start = '2013-01-01',end='2015-10-01', freq = 'MS')
ar=ar.reset_index()
ar=ar.loc[:,["index","item_cnt_day"]]
ar.columns = ['confirmed_date','count']
ar.head()
```

Out[41]:

	confirmed_date	count
0	2013-01-01	131479.0
1	2013-02-01	128090.0
2	2013-03-01	147142.0
3	2013-04-01	107190.0
4	2013-05-01	106970.0

Model

In [42]:
```
model = ARIMA(ar['count'].values, order=(1, 2, 1))
fit_model = model.fit(trend='c', full_output=True, disp=True)
fit_model.summary()
```

Out[42]:

ARIMA Model Results

Dep. Variable:	D2.y	**No. Observations:**	32
Model:	ARIMA(1, 2, 1)	**Log Likelihood**	-367.756
Method:	css-mle	**S.D. of innovations**	22262.760
Date:	Sat, 14 Oct 2023	**AIC**	743.512
Time:	21:07:43	**BIC**	749.375
Sample:	2	**HQIC**	745.456

	coef	std err	z	P>|z|	[0.025	0.975]
const	-73.2828	329.027	-0.223	0.824	-718.165	571.599
ar.L1.D2.y	-0.2607	0.168	-1.552	0.121	-0.590	0.068
ma.L1.D2.y	-1.0000	0.084	-11.969	0.000	-1.164	-0.836

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-3.8366	+0.0000j	3.8366	0.5000
MA.1	1.0000	+0.0000j	1.0000	0.0000

Prediction

In [43]:
```
fit_model.plot_predict()
plt.title('Forecast vs Actual')
pd.DataFrame(fit_model.resid).plot()
forcast = fit_model.forecast(steps=6)
```

```
pred_y = forcast[0].tolist()
pred = pd.DataFrame(pred_y)
```