Question1

In [124]:

```
# Import scikit image modules
from skimage import data, img_as_float
from skimage import exposure

#Playing with 3 random grayscale images
from skimage import exposure
import cv2
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
```

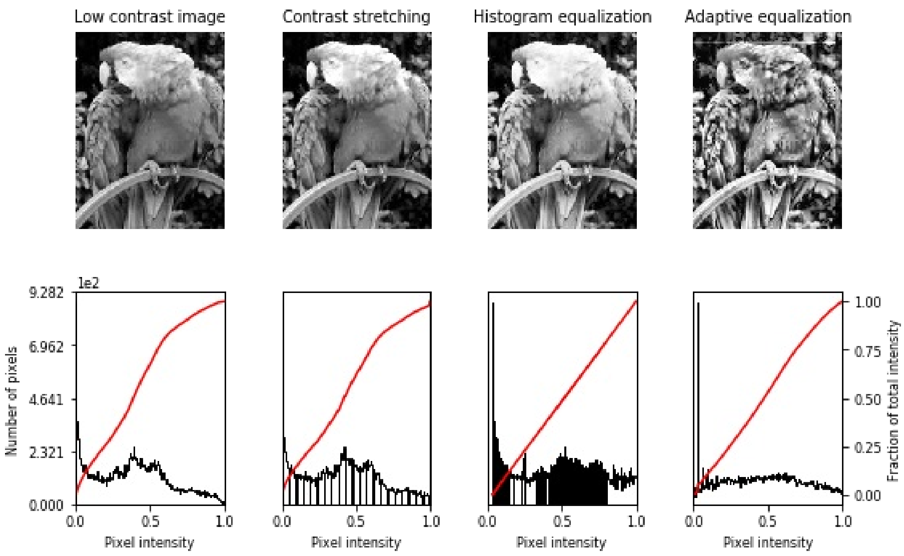
In [125]:

```
def plot_img_and_hist(img, axes, bins=256):
    """Plot an image along with its histogram and cumulative histogram.

    """
    img = img_as_float(img)
    ax_img, ax_hist = axes
    ax_cdf = ax_hist.twinx()

    # Display image
    ax_img.imshow(img, cmap=plt.cm.gray)
    ax_img.set_axis_off()
    ax_img.set_adjustable('box-forced')

    # Display histogram
    ax_hist.hist(img.ravel(), bins=bins, histtype='step', color='black')
    ax_hist.ticklabel_format(axis='y', style='scientific', scilimits=(0, 0))
    ax_hist.set_xlabel('Pixel intensity')
    ax_hist.set_xlim(0, 1)
    ax_hist.set_yticks([])

    # Display cumulative distribution
    img_cdf, bins = exposure.cumulative_distribution(img, bins)
    ax_cdf.plot(bins, img_cdf, 'r')
    ax_cdf.set_yticks([])

    return ax_img, ax_hist, ax_cdf
```

In [126]:

```
# Load an example image
img = cv2.imread('images/parrot.jpg', cv2.IMREAD_GRAYSCALE)

# Set font size for images
matplotlib.rcParams['font.size'] = 8

# Contrast stretching
p2, p98 = np.percentile(img, (2, 98))
img_rescale = exposure.rescale_intensity(img, in_range=(p2, p98))

# Histogram Equalization
img_eq1 = exposure.equalize_hist(img)

# Adaptive Equalization
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)

#### Everything below here is just to create the plot/graphs ####

# Display results
fig = plt.figure(figsize=(8, 5))
axes = np.zeros((2, 4), dtype=np.object)
axes[0, 0] = fig.add_subplot(2, 4, 1)
for i in range(1, 4):
    axes[0, i] = fig.add_subplot(2, 4, 1+i, sharex=axes[0,0], sharey=axes[0,0])
```

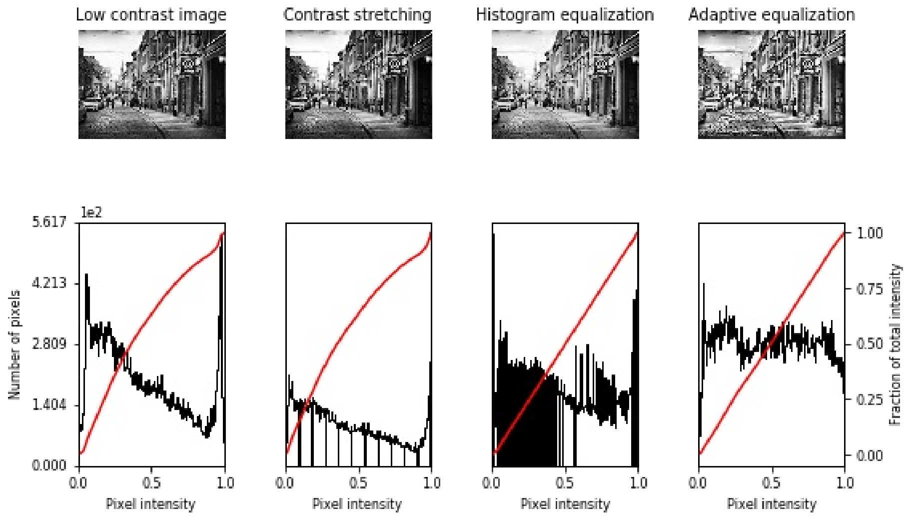
```
for i in range(0, 4):
    axes[1, i] = fig.add_subplot(2, 4, 5+i)

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img, axes[:, 0])
ax_img.set_title('Low contrast image')

y_min, y_max = ax_hist.get_ylim()
ax_hist.set_ylabel('Number of pixels')
ax_hist.set_yticks(np.linspace(0, y_max, 5))

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_rescale, axes[:, 1])
ax_img.set_title('Contrast stretching')

ax_img, ax_hist1, ax_cdf = plot_img_and_hist(img_eq1, axes[:, 2])
ax_img.set_title('Histogram equalization')

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_adapteq, axes[:, 3])
ax_img.set_title('Adaptive equalization')

ax_cdf.set_ylabel('Fraction of total intensity')
ax_cdf.set_yticks(np.linspace(0, 1, 5))

# prevent overlap of y-axis labels
fig.tight_layout()
plt.show()
```



In [129]:

```
img1 = cv2.imread('images/street.jpg', cv2.IMREAD_GRAYSCALE)

# Set font size for images
matplotlib.rcParams['font.size'] = 8

# Contrast stretching
p2, p98 = np.percentile(img1, (2, 98))
img_rescale = exposure.rescale_intensity(img1, in_range=(p2, p98))

# Histogram Equalization
img_eq2 = exposure.equalize_hist(img1)

# Adaptive Equalization
img_adapteq = exposure.equalize_adapthist(img1, clip_limit=0.03)

#### Everything below here is just to create the plot/graphs ####

# Display results
fig = plt.figure(figsize=(8, 5))
axes = np.zeros((2, 4), dtype=np.object)
axes[0, 0] = fig.add_subplot(2, 4, 1)
for i in range(1, 4):
    axes[0, i] = fig.add_subplot(2, 4, 1+i, sharex=axes[0,0], sharey=axes[0,0])
for i in range(0, 4):
    axes[1, i] = fig.add_subplot(2, 4, 5+i)

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img1, axes[:, 0])
```

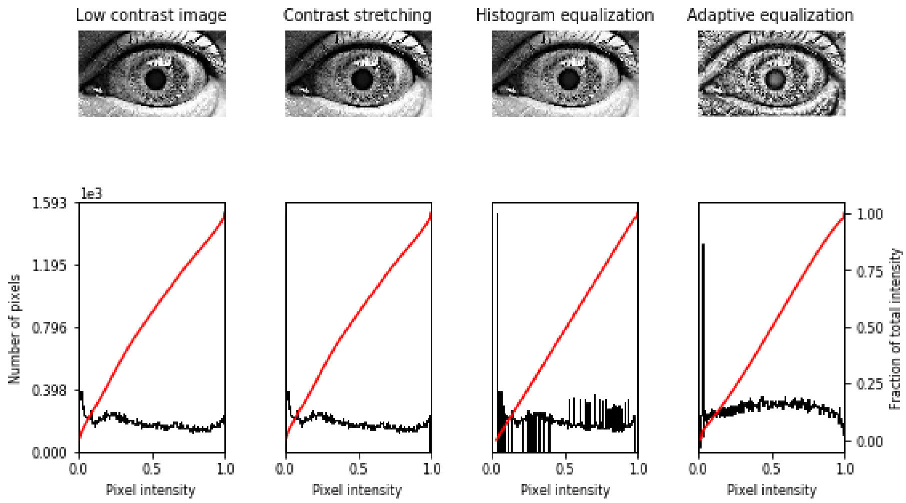
```
ax_img.set_title('Low contrast image')

y_min, y_max = ax_hist.get_ylim()
ax_hist.set_ylabel('Number of pixels')
ax_hist.set_yticks(np.linspace(0, y_max, 5))

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_rescale, axes[:, 1])
ax_img.set_title('Contrast stretching')

ax_img, ax_hist2, ax_cdf = plot_img_and_hist(img_eq2, axes[:, 2])
ax_img.set_title('Histogram equalization')

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_adapteq, axes[:, 3])
ax_img.set_title('Adaptive equalization')

ax_cdf.set_ylabel('Fraction of total intensity')
ax_cdf.set_yticks(np.linspace(0, 1, 5))

# prevent overlap of y-axis labels
fig.tight_layout()
plt.show()
```



In [34]:

```
img2 = cv2.imread('images/eye.jpg', cv2.IMREAD_GRAYSCALE)

# Set font size for images
matplotlib.rcParams['font.size'] = 8

# Contrast stretching
p2, p98 = np.percentile(img2, (2, 98))
img_rescale = exposure.rescale_intensity(img2, in_range=(p2, p98))

# Histogram Equalization
img_eq3 = exposure.equalize_hist(img2)

# Adaptive Equalization
img_adapteq = exposure.equalize_adapthist(img2, clip_limit=0.03)

#### Everything below here is just to create the plot/graphs ####

# Display results
fig = plt.figure(figsize=(8, 5))
axes = np.zeros((2, 4), dtype=np.object)
axes[0, 0] = fig.add_subplot(2, 4, 1)
for i in range(1, 4):
    axes[0, i] = fig.add_subplot(2, 4, 1+i, sharex=axes[0,0], sharey=axes[0,0])
for i in range(0, 4):
    axes[1, i] = fig.add_subplot(2, 4, 5+i)

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img2, axes[:, 0])
ax_img.set_title('Low contrast image')

y_min, y_max = ax_hist.get_ylim()
ax_hist.set_ylabel('Number of pixels')
ax_hist.set_yticks(np.linspace(0, y_max, 5))
```

```python
ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_rescale, axes[:, 1])
ax_img.set_title('Contrast stretching')

ax_img, ax_hist3, ax_cdf = plot_img_and_hist(img_eq3, axes[:, 2])
ax_img.set_title('Histogram equalization')

ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_adapteq, axes[:, 3])
ax_img.set_title('Adaptive equalization')

ax_cdf.set_ylabel('Fraction of total intensity')
ax_cdf.set_yticks(np.linspace(0, 1, 5))

# prevent overlap of y-axis labels
fig.tight_layout()
plt.show()
```



It is seen that pixel intensities are more similar for 1st and 3rd images (img and img2)

In [58]:

```python
height, width = img.shape
img=cv2.resize(img,(64,64))
img2=cv2.resize(img2,(64,64))
```

In [115]:

```python
a=img/255
b=img2/255
```

In [116]:

```python
#Calculating Softmax Probabilities
def softmax(X):
    """Calculate softmax Probabilities.

    """
    exps = np.exp(X)
    return exps / np.sum(exps)

a=softmax(a)
b=softmax(b)
```

In [112]:

```python
def cross_entropy(predictions, targets, epsilon=1e-12):
    """
    Computes cross entropy between targets (encoded as one-hot vectors)
    and predictions.
    Input: predictions (N, k) ndarray
```

```
            targets (N, k) ndarray
    Returns: scalar
    """
    predictions = np.clip(predictions, epsilon, 1. - epsilon)
    N = predictions.shape[0]
    ce = -np.sum(targets*np.log(predictions+1e-9))/N
    return ce

def KL(a, b):
    epsilon=1e-12
    a = np.asarray(a, dtype=np.float)+epsilon
    b = np.asarray(b, dtype=np.float)+epsilon
    divergence=np.sum(np.where(a != 0, a * np.log(a / b), 0))
    return divergence

predictions = a
targets = b
ans = 0.71355817782  #Correct answer
x = cross_entropy(predictions, targets)
print("Cross Entropy Loss =",x)
print("KL Divergence =",KL(a,b))
```

```
Cross Entropy Loss = 0.1299650323551208
KL Divergence = 2.8150873236249123e-16
```

In [ ]:

```

```

2. $$E = -\sum_{i=1}^{n_{out}} \left(t_i \log(y_i) + (1-t_i)\log(1-y_i)\right)$$

$$y_i = \frac{1}{1+e^{-s_i}} \; ; \quad s_i = \sum_{j=1} h_j w_{ji}$$

→ chain rule

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}}$$

$$\therefore \frac{\partial E}{\partial y_i} = \frac{-t_i}{y_i} + \left(\frac{1-t_i}{1-y_i}\right) = \frac{(y_i - t_i)}{y_i(1-y_i)}$$

$$\frac{\partial y_i}{\partial s_i} = y_i(1-y_i) ; \quad \frac{\partial s_i}{\partial w_{ji}} = h_j$$

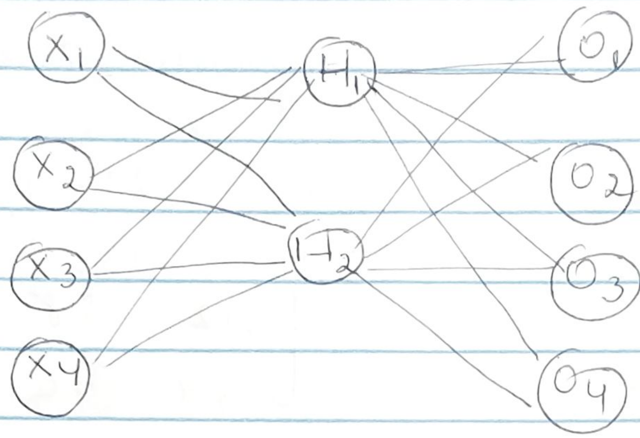
$$\therefore \frac{\partial E}{\partial w_{ji}} = h_j(y_i - t_i) = \frac{\partial E}{\partial s_i} \cdot \frac{\partial s_i}{\partial w_{ji}}$$

Now for $$\frac{\partial E}{\partial s'_j} = \sum_{i=1}^{n_{out}} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_i}$$

$$= \sum_{i=1}^{n_{out}} (y_i - t_i) \cdot w_{ji} \left(h_j(1-h_j)\right)$$

$$\frac{\partial E}{\partial h_j} = \sum_{i=1} \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{\partial x_j} = \sum_i \frac{\partial E}{\partial y_i} y_i(1-y_i) w_{ji}$$

$$\frac{\partial E}{\partial w'_{kj}} = \frac{\partial E}{\partial s'_j} \frac{\partial s'_j}{\partial w'_{kj}} = \sum_{i=1}^{n_{out}} (y_i - t_i)(w_{ji})(h_j(1-h_j))(x_k)$$

3. 

Linear Autoencoder

Input $[.5 \ .25 \ .25 \ .5] = X$

$W_{xh} = \begin{bmatrix} 1.2 & -0.5 \\ .75 & .5 \\ 0.2 & -1.2 \\ -0.5 & 1.2 \end{bmatrix}$; $W_{ho} = \begin{bmatrix} .2 & 1.5 & -0.5 & -0.2 \\ -0.4 & .6 & 1.2 & -0.5 \end{bmatrix}$

$H = X * W_{xh} + b_x \quad (b_x = 0)$

$H_1 = .5875, \quad H_2 = .175$

$Z = H * \cancel{\text{Output}}\ W_{ho} + b_o \quad (b_o = 0)$

↳ Output or O.

$O_1 = .0475, \quad O_2 = .9865, \quad O_3 = -.08375$

$O_4 = .205$

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$= \frac{[(.5-.0475)^2 + (.25-.9865)^2 + (.25+.08375)^2 + (.5-.205)^2]}{4}$$

$$= \frac{(.2047) + (.5424) + (.1114) + (.0870)}{4} = \underline{.2364}$$

4. MARKOV CHAIN

$$\begin{array}{c|cc} & \text{Vowel} & \text{Consonant} \\ \hline \text{Vowel} & \multicolumn{2}{c}{\multirow{2}{*}{$\begin{bmatrix} .12 & .88 \\ .54 & .46 \end{bmatrix}$}} \\ \text{Consonant} & & \end{array}$$

$\rightarrow$ Sum of rows is 1 & this is a Translational Probability matrix

$\rightarrow$ From the data we get 38% of letters as vowels if words were created by selecting random uniform letters & 62% as consonants.

Stationary matrix is calculated as

$$SP = S$$

$$[S_1 \ S_2] \begin{bmatrix} .12 & .88 \\ .54 & .46 \end{bmatrix} = [S_1 \ S_2]$$

$$.12S_1 + .54S_2 = S_1 \quad \text{— (i)}$$

$$.88S_1 + .46S_2 = S_2$$

We know $S_1 + S_2 = 1$ or $S_1 = 1 - S_2$ — (ii)

Substituting (ii) in (i)

$S_2 = .62$ & $\therefore S_1 = .38$, $S = [.38 \ .62]$

Question 5 RNN with an input sequence array to generate an output

In [77]:

```
import numpy as np
```

In [78]:

```
def softmax(X):
    "Returns softmax probability"
    exps = np.exp(X)
    return exps / np.sum(exps)

# def rnn(xt,waa,wax,wya,ba,by):

#     for i in xt:
#         a=np.tanh(a_prev*Waa+xt[i]*Wax)
#         y=softmax(a*Wya)
#     return a,y


# xt = np.array([0.1, 0.25, 0.35, 0.25,.10])
# a_prev = 0
# Waa = np.array([1,-0.5])
# Wax = np.array([-0.5, 0.5])
# Wya = np.array([.25, 0.75])
# ba = 0
# by = 0

# rnn(xt,Waa,Wax,Wya,ba,by)
```

In [79]:

```
#   xt is input data at timestep "t", numpy array of shape (n_x, m).
#   a_prev - Hidden state at timestep "t-1", numpy array of shape (n_a, m)
#     parameters:
#                Wax -- Weight matrix multiplying the input, numpy array of shape (n_a, n_x)
#                Waa -- Weight matrix multiplying the hidden state, numpy array of shape (n_a, n_a)
#                Wya -- Weight matrix relating the hidden-state to the output, numpy array of shape
(n_y, n_a)
#                ba --  Bias, numpy array of shape (n_a, 1)
#                by -- Bias relating the hidden-state to the output, numpy array of shape (n_y, 1)
#     Returns:
#     a_next -- next hidden state, of shape (n_a, m)
#     yt_pred -- prediction at timestep "t", numpy array of shape (n_y, m)
```

In [80]:

```
xt = np.array([0.1, 0.25, 0.35, 0.25,.10])
a_prev = 0
Waa = np.array([1,-0.5])
Wax = np.array([-0.5, 0.5])
Wya = np.array([.25, 0.75])
ba = 0
by = 0

a1=np.tanh(a_prev*Waa+xt[0]*Wax)
print(a1)
#y1=softmax(a1*Wya)
y1= np.tanh(np.dot(Wya,a1) + by)
print(y1)

a2=np.tanh(a1*Waa+xt[1]*Wax)
print(a2)
y2 = np.tanh(np.dot(Wya,a2) + by)
#y2=softmax(a2*Wya)
print(y2)
```

```
a3=np.tanh(a2*Waa+xt[2]*Wax)
print(a3)
y3 = np.tanh(np.dot(Wya,a3) + by)
#y3=softmax(a3*Wya)
print(y3)


a4=np.tanh(a3*Waa+xt[3]*Wax)
print(a4)
y4 = np.tanh(np.dot(Wya,a4) + by)
print(y4)

a5=np.tanh(a4*Waa+xt[4]*Wax)
print(a5)
y5 = np.tanh(np.dot(Wya,a5) + by)
print(y5)
```

```
[-0.04995837  0.04995837]
0.024973993438951323
[-0.17319478  0.0996886 ]
0.03145737229659703
[-0.33477361  0.12450629]
0.00968601207847469
[-0.42989968  0.06266464]
-0.06040282246432743
[-0.44616326  0.01866551]
-0.09723350402569085
```

In [81]:

```
yhat=[y1,y2,y3,y4,y5]
print('Output Sequence:', yhat)
```

```
Output Sequence: [0.024973993438951323, 0.03145737229659703, 0.00968601207847469, -
0.06040282246432743, -0.09723350402569085]
```

In [82]:

```
#Extra LSTM Autoencoder Predicition
#Output Sequence prediction after 500 epochs is amazingly accurate with the LSTM
```

In [83]:

```
#Calling Libraries for LSTM Prediction
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.utils import plot_model
```

In [84]:

```
# define input sequence
seq_in = array([0.1, 0.25, 0.35, 0.25,.10])
# reshape input into [samples, timesteps, features]
n_in = len(seq_in)
seq_in = seq_in.reshape((1, n_in, 1))
# prepare output sequence
#[:, 1:, :]
seq_out = seq_in
n_out = n_in
```

In [85]:

```
# Define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))
model.add(RepeatVector(n_out))
model.add(LSTM(100, activation='relu', return_sequences=True))
```

```
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')

# fit model
model.fit(seq_in, seq_out, epochs=500, verbose=0)

# demonstrate prediction
yhat = model.predict(seq_in, verbose=0)
print("Output Sequence ",yhat[0,:,0])
```

```
Output Sequence  [0.10000786 0.25001827 0.35003456 0.25005242 0.10006101]
```

We can see better predictions using LSTM method

Question 6

STOCK PREDICTION

In [10]:

```
import math
import pandas as pd
import numpy as np


# import data visualization libraries
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# import other libraries
import datetime as dt
import time
```

READING DATA

In [17]:

```
 def get_data(path):
     dataframe = pd.read_csv(path)
     print(dataframe.head())
     return




print('NASDAQ Data')

data = get_data('stocks/NASDAQ.csv')
```

```
NASDAQ Data
        Date         Open      Volume         High          Low        Close  \
0   3/8/2018  7422.770020  2272110000  7435.009766  7391.500000  7427.950195
1   3/9/2018  7475.979980  2302930000  7560.810059  7469.029785  7560.810059
2  3/12/2018  7581.040039  2294440000  7609.100098  7563.439941  7588.319824
3  3/13/2018  7627.520020  2448830000  7637.270020  7492.979980  7511.009766
4  3/14/2018  7539.779785  2104450000  7544.890137  7473.899902  7496.810059

     Adj Close
0  7427.950195
1  7560.810059
2  7588.319824
3  7511.009766
4  7496.810059
```

In [18]:

```
print('DOW Data')

data = get_data('stocks/Dow.csv')
```

```
DOW Data
        Date         Open     Volume         High          Low        Close  \
0   3/8/2018  24853.41016  327300000  24950.49023  24703.05078  24895.21094
1   3/9/2018  25004.89063  371570000  25336.33008  25004.89063  25335.74023
2  3/12/2018  25372.43945  362580000  25449.15039  25152.01953  25178.60938
3  3/13/2018  25257.75000  447880000  25376.40039  24947.50000  25007.02930
4  3/14/2018  25086.97070  356830000  25130.11914  24668.83008  24758.11914

     Adj Close
0  24895.21094
```

```
1  25335.74023
2  25178.60938
3  25007.02930
4  24758.11914
```

In [19]:

```
print('S&P500 Data')

data = get_data('stocks/S&P500.csv')
```

```
S&P500 Data
        Date         Open      Volume         High          Low        Close  \
0   3/8/2018  2732.750000  3212320000  2740.449951  2722.649902  2738.969971
1   3/9/2018  2752.909912  3364100000  2786.570068  2751.540039  2786.570068
2  3/12/2018  2790.540039  3185020000  2796.979980  2779.260010  2783.020020
3  3/13/2018  2792.310059  3301650000  2801.899902  2758.679932  2765.310059
4  3/14/2018  2774.060059  3391360000  2777.110107  2744.379883  2749.479980

     Adj Close
0  2738.969971
1  2786.570068
2  2783.020020
3  2765.310059
4  2749.479980
```

Analyzing Data

In [21]:

```
data1=pd.read_csv('stocks/NASDAQ.csv')
def data_analysis(data):
    print("\n")
    print("Open   => mean :", np.mean(data1['Open']),  "  \t Std: ", np.std(data1['Open']),  "  \t Max: ", np.max(data1['Open']),  "  \t Min: ", np.min(data1['Open']))
    print("High   => mean :", np.mean(data1['High']),  "  \t Std: ", np.std(data1['High']),  "  \t Max: ", np.max(data1['High']),  "  \t Min: ", np.min(data1['High']))
    print("Low    => mean :", np.mean(data1['Low']),   "  \t Std: ", np.std(data1['Low']),   "  \t Max: ", np.max(data1['Low']),   "  \t Min: ", np.min(data1['Low']))
    print("Close  => mean :", np.mean(data1['Close']), "  \t Std: ", np.std(data1['Close']), "  \t Max: ", np.max(data1['Close']), "  \t Min: ", np.min(data1['Close']))
    print("Volume => mean :", np.mean(data1['Volume']),"  \t Std: ", np.std(data1['Volume']),"  \t Max: ", np.max(data1['Volume']),"  \t Min: ", np.min(data1['Volume']))
    return

data_analysis(data1)
```

```
Open   => mean : 7448.174101996309     Std:  375.2168006753417     Max:  8094.200195     Min:  6257
.859863000001
High   => mean : 7495.134799870849     Std:  359.650585724092     Max:  8133.299805     Min:  6355.
180176
Low    => mean : 7393.75051711439     Std:  390.2330193957042     Max:  8079.310059     Min:  6190.
169922
Close  => mean : 7445.9791173911435     Std:  376.1184988923075     Max:  8109.689941     Min:  619
2.919922
Volume => mean : 2461182952.0295205     Std:  3400808566.283922     Max:  57995120000     Min:  958
950000
```

In [25]:

```
data2=pd.read_csv('stocks/Dow.csv')
data_analysis(data2)
```

```
Open   => mean : 25047.956844391145     Std:  849.3874326706024     Max:  26833.4707     Min:  2185
7.73047
High   => mean : 25187.57727472325     Std:  807.3612001276886     Max:  26951.81055     Min:  2233
9.86914
Low    => mean : 24893.883749704793     Std:  893.4132917502258     Max:  26789.08008     Min:  217
12.5293
```

```
12.5293
Close   => mean : 25045.51476785978     Std:  851.3002577896312     Max:  26828.39063     Min:  2179
2.19922
Volume => mean : 332070590.40590405     Std:  91604354.53694943     Max:  900510000     Min:  15594
0000
```

In [26]:

```python
data3=pd.read_csv('stocks/S&P500.csv')
data_analysis(data3)
```

```
Open    => mean : 25047.956844391145     Std:  849.3874326706024     Max:  26833.4707     Min:  2185
7.73047
High    => mean : 25187.57727472325     Std:  807.3612001276886     Max:  26951.81055     Min:  2233
9.86914
Low     => mean : 24893.883749704793     Std:  893.4132917502258     Max:  26789.08008     Min:  217
12.5293
Close   => mean : 25045.51476785978     Std:  851.3002577896312     Max:  26828.39063     Min:  2179
2.19922
Volume => mean : 332070590.40590405     Std:  91604354.53694943     Max:  900510000     Min:  15594
0000
```

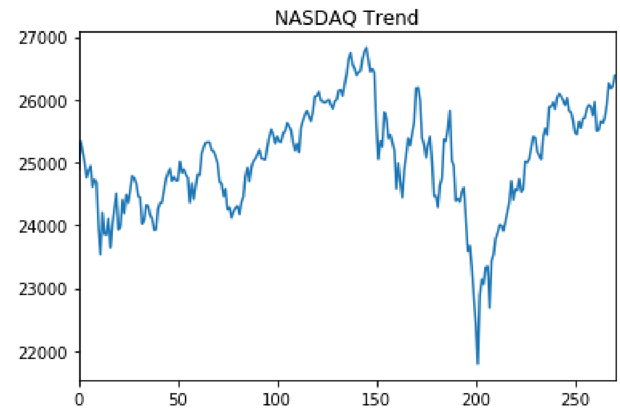
In [7]:

```
Open    => mean : 399.194655027235     Std:  246.00435389311465     Max:  1140.310059     Min:  49.6
44646
High    => mean : 402.6346727007105     Std:  247.3188660507521     Max:  1148.880005     Min:  50.9
20921
Low     => mean : 395.37474639254003     Std:  244.49253356916884     Max:  1126.660034     Min:  48
.028027
Close   => mean : 399.07631232178807     Std:  246.04368294403352     Max:  1139.099976     Min:  50
.055054
Volume => mean : 7896743.309650681     Std:  8274864.818135541     Max:  82151100     Min:  520600
```

DATA EXPLORATION

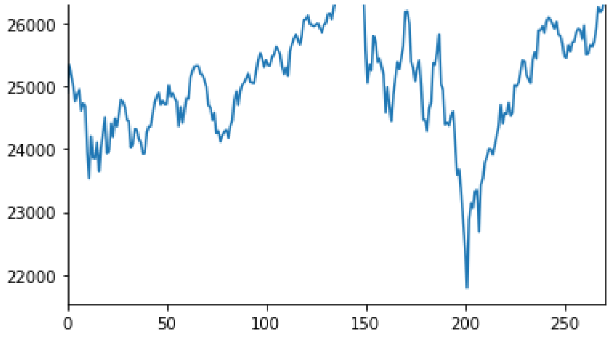
In [34]:

```python
import matplotlib.pyplot as plt
plt.title('NASDAQ Trend')
data1['Adj Close'].plot();
```



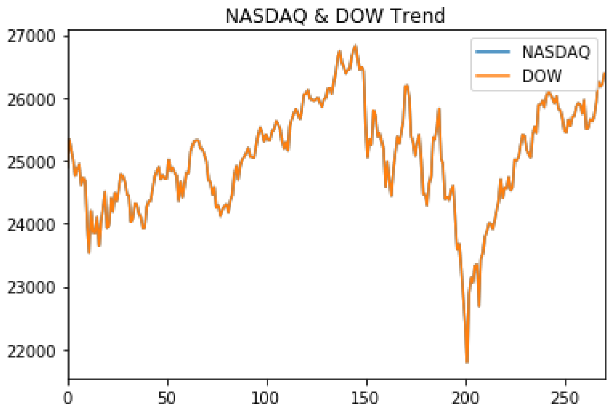
In [35]:

```python
plt.title('DOW Trend')
data2['Adj Close'].plot();
```



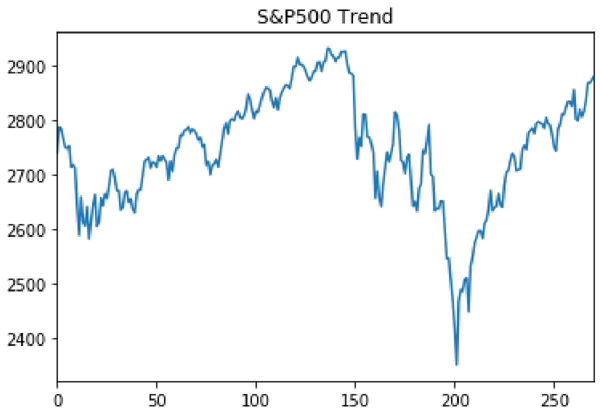
In [73]:

```
plt.title('NASDAQ & DOW Trend')
data1['Adj Close'].plot()
data2['Adj Close'].plot()
plt.legend(['NASDAQ', 'DOW']);
```



In [36]:

```
plt.title('S&P500 Trend')
data3['Adj Close'].plot();
```



PRE-PROCESSING DATA

1 BENCHMARK MODEL

In [120]:

```
from sklearn.preprocessing import MinMaxScaler
import csv
scaler = MinMaxScaler()

data_val3=data3.iloc[:,1:7].values
print(data_val3[1])
X_test = data_val3[:,1:2]
y_test = data3['Close']
```

```
y_new = []
for i in y:
    y_new.append([i])
y_new = np.array(y_new)


X_test = np.array(X_test)

X_test=scaler.fit_transform(X_test)
y_new=scaler.fit_transform(y_new)
```

```
[2.75290991e+03 3.36410000e+09 2.78657007e+03 2.75154004e+03
 2.78657007e+03 2.78657007e+03]
```

In [121]:

```
data_val1=data1.iloc[:,1:7].values
print(data_val1[1])
X = data_val1[:,1:2]
y = data1['Close']

y_new = []
for i in y:
    y_new.append([i])
y_new = np.array(y_new)


X = np.array(X)

X=scaler.fit_transform(X_test)
y_new=scaler.fit_transform(y_new)
```

```
[2.50048906e+04 3.71570000e+08 2.53363301e+04 2.50048906e+04
 2.53357402e+04 2.53357402e+04]
```

2 LSTM MODEL

In [122]:

```
new_data1 = scaler.fit_transform(data_val1)
print(new_data1[1])

train_X = new_data1[:,0:4]

train_y_old = new_data1[:,5]

train_Y = []
for i in train_y_old:
    train_Y.append([i])
train_Y = np.array(train_Y)
print(train_Y.shape)


train_X = np.array(train_X)
train_Y = np.array(train_Y)

print(train_X .shape)
print(train_Y.shape)
```

```
[0.6325009  0.2896034  0.64971791 0.64854297 0.70361524 0.70361524]
(271, 1)
(271, 4)
(271, 1)
```

LINEAR REGRESSION MODEL

a) SPLITTING DATA

In [123]:

```
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(X, y_new, test_size = 0.15,shuffle = False,ra
ndom_state = 0)
print(X_train[1])
print(y_train[1])
print(X_train.shape)
print(X_valid.shape)
print(y_train.shape)
print(y_valid.shape)
```

```
[0.28745115]
[0.70361524]
(230, 1)
(41, 1)
(230, 1)
(41, 1)
```

b) INITIALISING MODEL

In [124]:

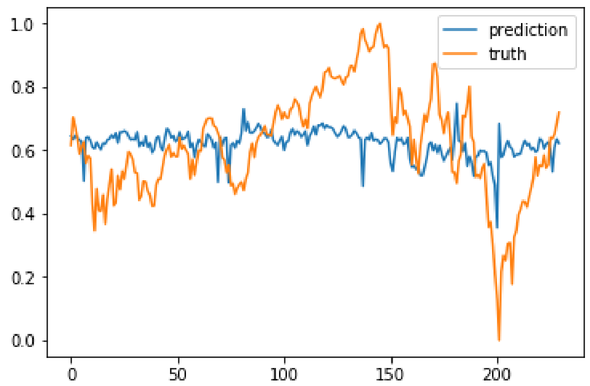
```
from sklearn.linear_model import LinearRegression
model= LinearRegression()
model =model.fit(X_train,y_train)
#predicted__valid_price = model.predict(X_valid)
predicted__test_price = model.predict(X_test)
predicted_train = model.predict(X_train)
print(model.coef_)
print(model.intercept_)
```

```
[[-0.39188622]]
[0.74741956]
```

c) DISPLAYING RESULT

In [127]:

```
import matplotlib.pyplot as plt
plt.plot(predicted_train, label='prediction')
plt.plot(y_train, label='truth')
plt.legend()
plt.show()
```



d) CALCULATING ERRORS

In [61]:

```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(y_test,predicted_price),np.sqrt(mean_squared_error(y_test,predicted_price))
```

Out[61]:

```
(-5.8961422378198005, 0.1570501991534916)
```