Practical 1

Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

Importing libraries and the dataset

In [1]:
```
#Importing the pandas for data processing and numpy for numerical computing
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
# Importing the Boston Housing dataset from the sklearn
from sklearn.datasets import fetch_openml

# Load the Boston housing dataset from OpenML
boston = fetch_openml(data_id=531)
```

```
C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\sklearn\datasets\_openml.py:932: FutureWarning: The default value of `parser` will change from `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is not installed. Note that the pandas parser may return different data types. See the Notes Section in fetch_openml's API doc for details.
  warn(
```

In [3]:
```
#Converting the data into pandas dataframe
data = pd.DataFrame(boston.data)
```

First look at the dataset

In [4]:
```
#First look at the data
data.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33

In [5]:
```
#Adding the feature names to the dataframe
data.columns = boston.feature_names
```

In [6]:
```
#Adding the target variable to the dataset
data['PRICE'] = boston.target
```

In [7]:
```
#Looking at the data with names and target variable
data.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

In [8]:
```
#Shape of the data
print(data.shape)
```

```
(506, 14)
```

In [9]:
```
#Checking the null values in the dataset
data.isnull().sum()
```

Out[9]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

No null values in the dataset, no missing value treatement needed

In [10]:
```
#Checking the statistics of the data
data.describe()
```

Out[10]:

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	TAX
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.554695	6.284634	68.574901	3.795043	408.237154
std	8.601545	23.322453	6.860353	0.115878	0.702617	28.148861	2.105710	168.537116
min	0.006320	0.000000	0.460000	0.385000	3.561000	2.900000	1.129600	187.000000
25%	0.082045	0.000000	5.190000	0.449000	5.885500	45.025000	2.100175	279.000000
50%	0.256510	0.000000	9.690000	0.538000	6.208500	77.500000	3.207450	330.000000
75%	3.677083	12.500000	18.100000	0.624000	6.623500	94.075000	5.188425	666.000000
max	88.976200	100.000000	27.740000	0.871000	8.780000	100.000000	12.126500	711.000000

This is sometimes very useful, for example if you look at the CRIM the max is 88.97 and 75% of the value is below 3.677083 and mean is 3.613524 so it means the max values is actually an outlier or there are outliers present in the column

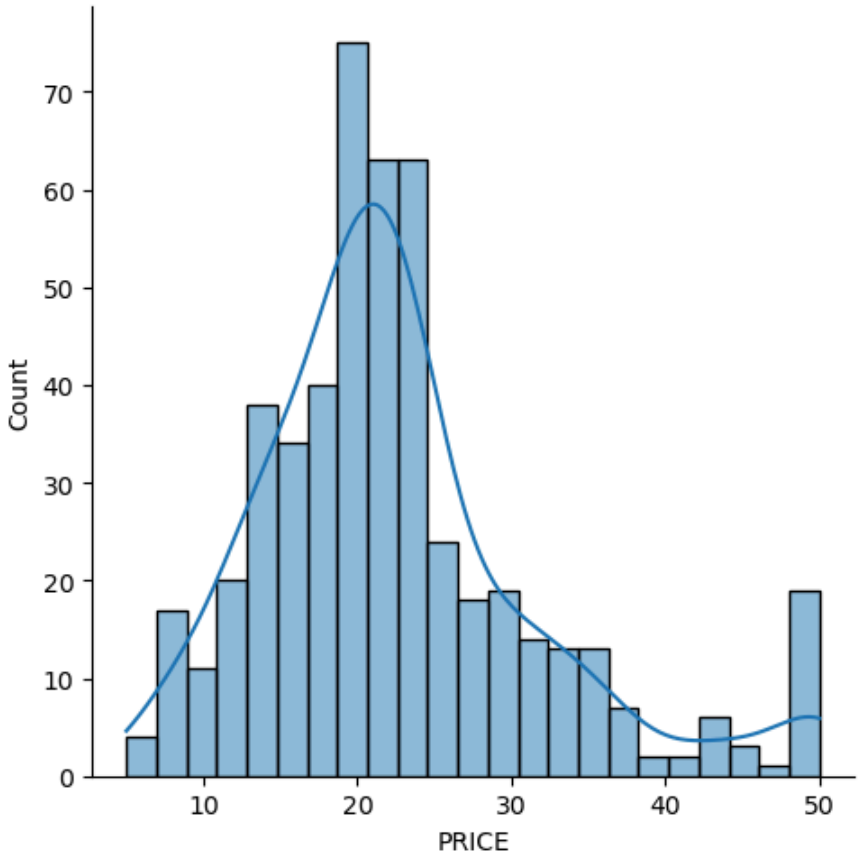
In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype   
---  ------   --------------  -----   
 0   CRIM     506 non-null    float64 
 1   ZN       506 non-null    float64 
 2   INDUS    506 non-null    float64 
 3   CHAS     506 non-null    category
 4   NOX      506 non-null    float64 
 5   RM       506 non-null    float64 
 6   AGE      506 non-null    float64 
 7   DIS      506 non-null    float64 
 8   RAD      506 non-null    category
 9   TAX      506 non-null    float64 
 10  PTRATIO  506 non-null    float64 
 11  B        506 non-null    float64 
 12  LSTAT    506 non-null    float64 
 13  PRICE    506 non-null    float64 
dtypes: category(2), float64(12)
memory usage: 49.0 KB
```

Visualisation

In [12]:
```
#checking the distribution of the target variable
import seaborn as sns
sns.displot(data['PRICE'], kde=True)
```

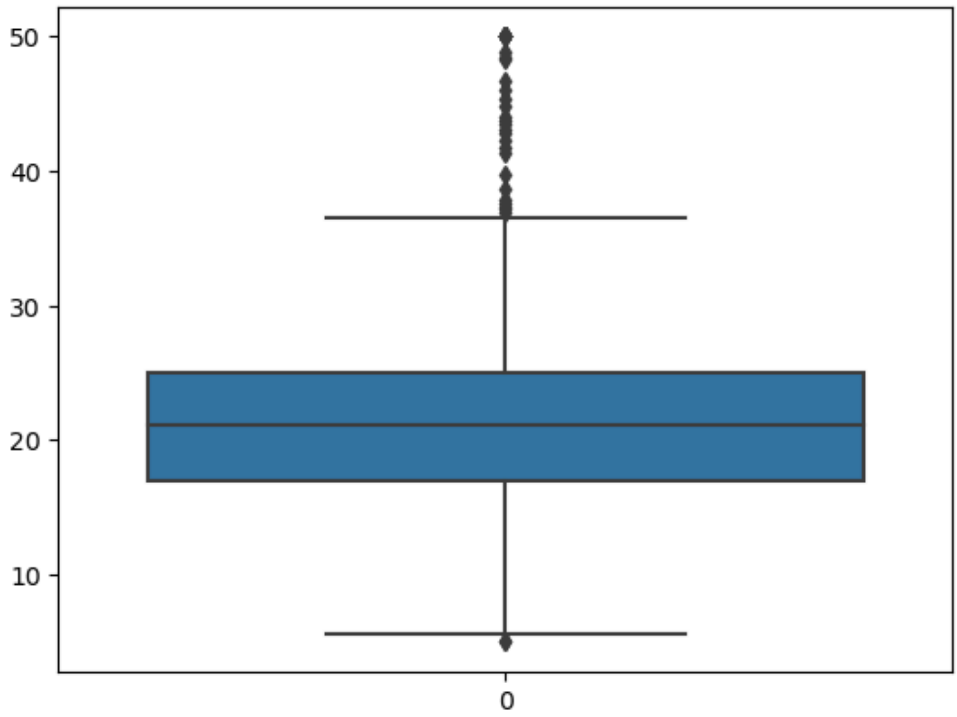
Out[12]:
```
<seaborn.axisgrid.FacetGrid at 0x216a12168f0>
```



The distribution seems normal, has not be the data normal we would have perform log transformation or took to square root of the data to make the data normal. Normal distribution is need for the machine learning for better predictiblity of the model

In [13]:
```
#Distribution using box plot
sns.boxplot(data.PRICE)
```

Out[13]:
```
<Axes: >
```



Checking the correlation of the independent feature with the dependent feature

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.An intelligent correlation analysis can lead to a greater understanding of your data

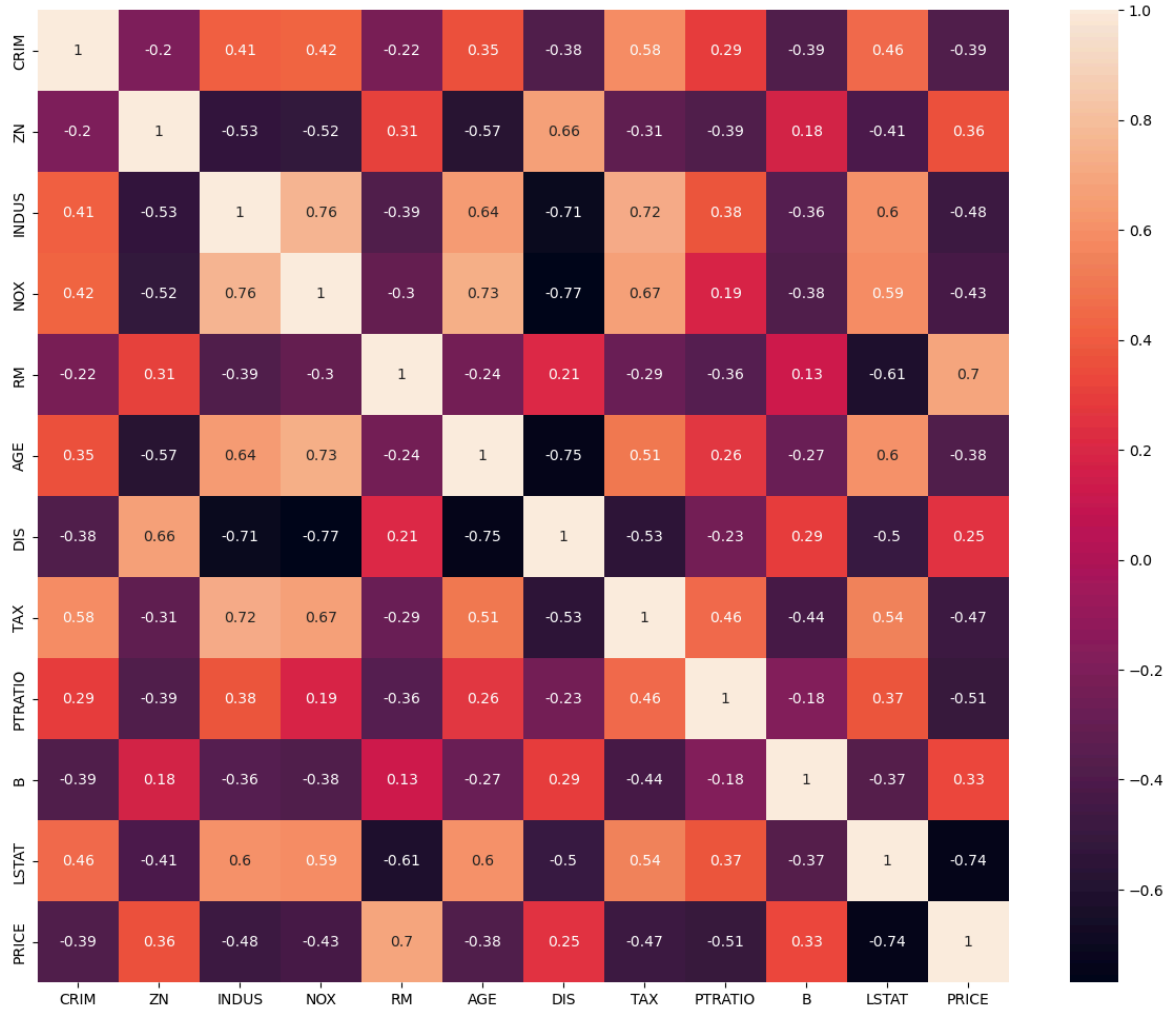
In [14]:
```
#checking Correlation of the data
correlation = data.corr()
correlation.loc['PRICE']
```

```
C:\Users\Yashwardhan Deshmukh\AppData\Local\Temp\ipykernel_11992\1690761071.py:2: Fu
tureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  correlation = data.corr()
```

Out[14]:
```
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.737663
PRICE      1.000000
Name: PRICE, dtype: float64
```

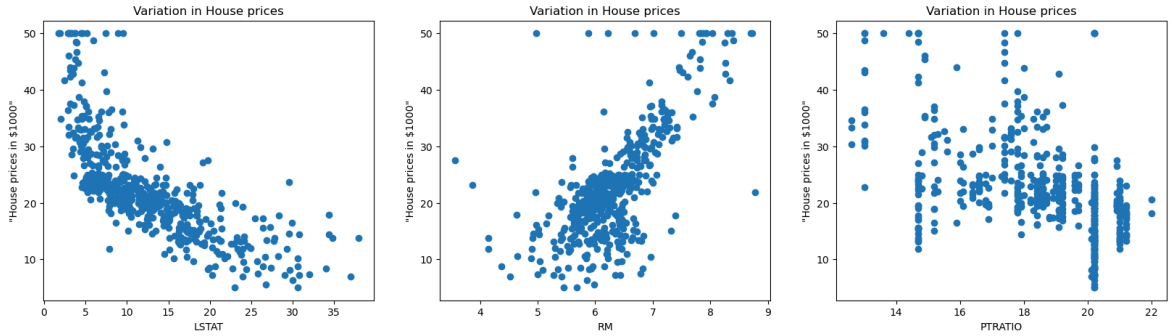
In [15]:
```
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

Out[15]: <Axes: >



By looking at the correlation plot LSAT is negatively correlated with -0.75 and RM is positively correlated to the price and PTRATIO is correlated negatively with -0.51

In [16]:
```
# Checking the scatter plot with the most correlated features
plt.figure(figsize = (20,5))
features = ['LSTAT','RM','PTRATIO']
for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = data[col]
    y = data.PRICE
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```



Splitting the dependent feature and independent feature

In [17]:
```
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.PRICE
```

Splitting the data for Model Validation

In [18]:
```
# Splitting the data into train and test for building the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state
```

Building the Model

In [19]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [20]:
```
#Fitting the model
regressor.fit(X_train,y_train)
```

Out[20]:
```
▾ LinearRegression

LinearRegression()
```

Model Evaluation

In [21]:
```
# Convert X_test to a NumPy array
X_test = np.array(X_test)
#Prediction on the test dataset
y_pred = regressor.predict(X_test)
```

```
C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\sklearn\base.py:420: UserW
arning: X does not have valid feature names, but LinearRegression was fitted with fe
ature names
  warnings.warn(
```

In [22]:
```
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

```
5.041784121402046
```

In [23]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.7263451459702515
```

Neural Networks

In [24]:
```
!pip install tensorflow
!pip install keras
```

```
=2.15.0->tensorflow) (2.23.4)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users
\yashwardhan deshmukh\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->
tensorflow-intel==2.15.0->tensorflow) (0.7.2)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in c:\users\yashwardh
an deshmukh\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow
-intel==2.15.0->tensorflow) (1.1.0)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\yashwardhan deshmu
kh\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==
2.15.0->tensorflow) (2.28.1)
Requirement already satisfied: markdown>=2.6.8 in c:\users\yashwardhan deshmukh\a
naconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.
0->tensorflow) (3.4.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\yashwardhan deshmukh\a
naconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.
0->tensorflow) (2.2.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\yashwardhan des
hmukh\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>
=2.15->tensorflow-intel==2.15.0->tensorflow) (5.3.2)
```

In [25]:
```
#Creating the neural network model
import keras
from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential

model = Sequential()

model.add(Dense(128,activation  = 'relu',input_dim =13))
model.add(Dense(64,activation  = 'relu'))
model.add(Dense(32,activation  = 'relu'))
model.add(Dense(16,activation  = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'adam',loss = 'mean_squared_error')
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecate
d. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.c
ompat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Pleas
e use tf.compat.v1.train.Optimizer instead.
```

In [26]:
```
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\sklearn\base.py:420: UserW
arning: X does not have valid feature names, but StandardScaler was fitted with feat
ure names
  warnings.warn(
```

In [27]:
```
results=model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 91/100
13/13 [==============================] - 0s 3ms/step - loss: 3.9828
Epoch 92/100
13/13 [==============================] - 0s 2ms/step - loss: 4.0298
Epoch 93/100
13/13 [==============================] - 0s 2ms/step - loss: 3.8867
Epoch 94/100
13/13 [==============================] - 0s 2ms/step - loss: 3.9619
Epoch 95/100
13/13 [==============================] - 0s 2ms/step - loss: 3.7390
Epoch 96/100
13/13 [==============================] - 0s 2ms/step - loss: 3.8308
Epoch 97/100
13/13 [==============================] - 0s 3ms/step - loss: 3.9280
Epoch 98/100
13/13 [==============================] - 0s 3ms/step - loss: 3.6752
Epoch 99/100

13/13 [==============================] - 0s 3ms/step - loss: 3.6492
Epoch 100/100
```

Evaluation of the model

In [28]:
```
y_pred = model.predict(X_test)
```

```
4/4 [==============================] - 0s 2ms/step
```

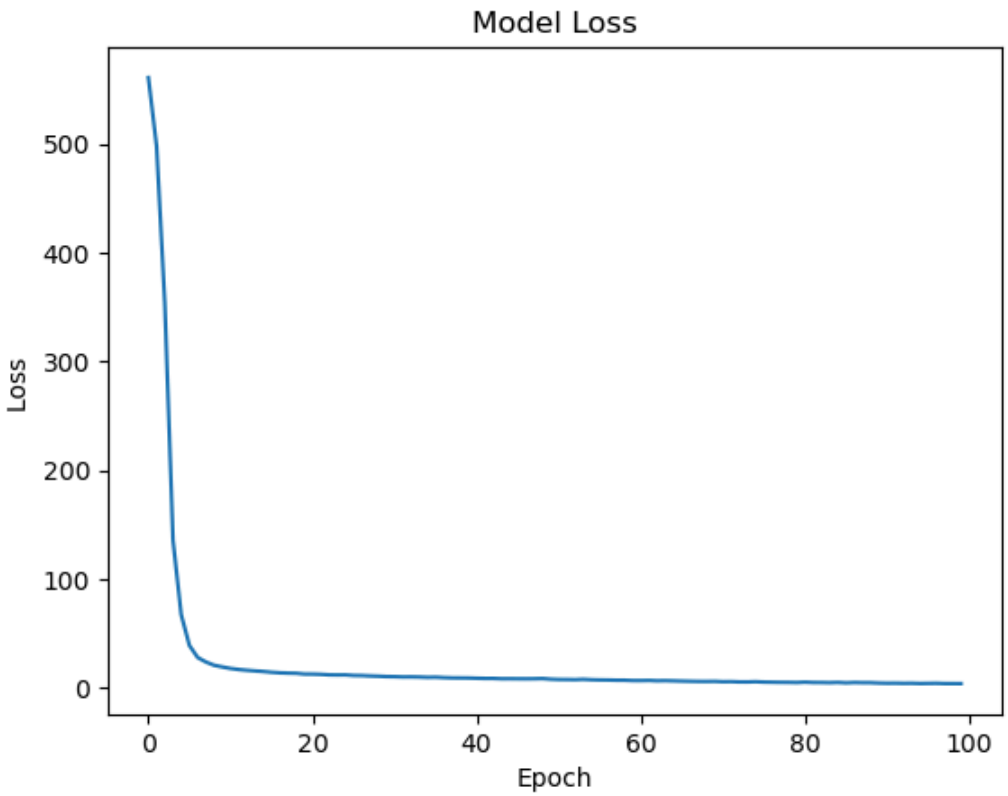
In [29]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.8848322436428823
```

In [30]:
```
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

```
3.270755808161393
```

In [31]:
```
plt.plot(results.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss ')
plt.xlabel('Epoch')
plt.show()
```

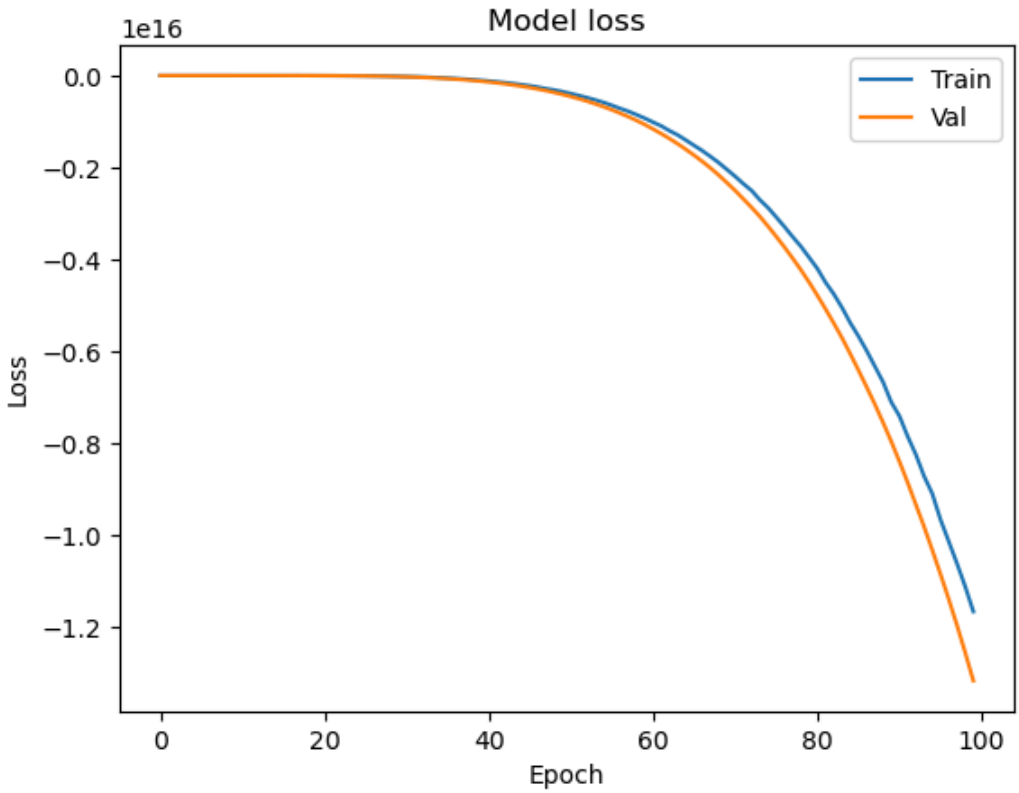


In [32]:
```
from keras.layers import Dropout
from keras import regularizers


model_3 = Sequential([
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01), input_sha
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)),
])
```

In [33]:
```
model_3.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist_3 = model_3.fit(X_train, y_train,
          batch_size=32, epochs=100,
          validation_data=(X_test, y_test))
```

```
13/13 [==============================] - 1s 46ms/step - loss: -3292891833171968.0
000 - accuracy: 0.0000e+00 - val_loss: -3737344175767552.0000 - val_accuracy: 0.0
000e+00
Epoch 78/100
13/13 [==============================] - 1s 48ms/step - loss: -3512145081794560.0
000 - accuracy: 0.0000e+00 - val_loss: -3980877948256256.0000 - val_accuracy: 0.0
000e+00
Epoch 79/100
13/13 [==============================] - 1s 48ms/step - loss: -3720911899328512.0
000 - accuracy: 0.0000e+00 - val_loss: -4232192859308032.0000 - val_accuracy: 0.0
000e+00
Epoch 80/100
13/13 [==============================] - 1s 48ms/step - loss: -3960872259026944.0
000 - accuracy: 0.0000e+00 - val_loss: -4501129484304384.0000 - val_accuracy: 0.0
000e+00
Epoch 81/100
13/13 [==============================] - 1s 49ms/step - loss: -4202862796079104.0
000 - accuracy: 0.0000e+00 - val_loss: -4787134040899584.0000 - val_accuracy: 0.0
000e+00
Epoch 82/100
```

In [34]:
```
plt.plot(hist_3.history['loss'])
plt.plot(hist_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Practical 2

Classification using Deep neural network: Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews.Use IMDB dataset

Import the required libraries

In [1]:
```
!pip install git+https://github.com/tensorflow/tensorflow.git@v2.15.0

!pip install keras
```

```
Collecting git+https://github.com/tensorflow/tensorflow.git@v2.15.0
  Cloning https://github.com/tensorflow/tensorflow.git (https://github.com/tensorflow/tensorflow.git) (to revision v2.15.0) to c:\users\yashwardhan deshmukh\appdata\local\temp\pip-req-build-75a22au7

  Running command git clone --filter=blob:none --quiet https://github.com/tensorflow/tensorflow.git (https://github.com/tensorflow/tensorflow.git) 'C:\Users\Yashwardhan Deshmukh\AppData\Local\Temp\pip-req-build-75a22au7'
  error: 27943 bytes of body are still expected
  fetch-pack: unexpected disconnect while reading sideband packet
  fatal: early EOF
  fatal: fetch-pack: invalid index-pack output
  error: subprocess-exited-with-error

  git clone --filter=blob:none --quiet https://github.com/tensorflow/tensorflow.git (https://github.com/tensorflow/tensorflow.git) 'C:\Users\Yashwardhan Deshmukh\AppData\Local\Temp\pip-req-build-75a22au7' did not run successfully.
  exit code: 128

  See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
error: subprocess-exited-with-error

git clone --filter=blob:none --quiet https://github.com/tensorflow/tensorflow.git (https://github.com/tensorflow/tensorflow.git) 'C:\Users\Yashwardhan Deshmukh\AppData\Local\Temp\pip-req-build-75a22au7' did not run successfully.
exit code: 128

See above for output.

note: This error originates from a subprocess, and is likely not a problem with pip.

Requirement already satisfied: keras in c:\users\yashwardhan deshmukh\anaconda3\lib\site-packages (2.15.0)
```

In [2]:
```
import pandas as pd
import numpy as np
import keras
import tensorflow as tf
from matplotlib import pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Read the Data

In [3]:
```
df_train=pd.read_csv('Train.csv')
df_val=pd.read_csv('Valid.csv')
df_train.head()
```

Out[3]:

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

In [4]:
```
df_val.head()
```

Out[4]:

	text	label
0	It's been about 14 years since Sharon Stone aw...	0
1	someone needed to make a car payment... this i...	0
2	The Guidelines state that a comment must conta...	0
3	This movie is a muddled mish-mash of clichés f...	0
4	Before Stan Laurel became the smaller half of ...	0

In [5]:
```
X_train=df_train['text'].values
Y_train=df_train['label'].values
```

In [6]:
```
X_val=df_val['text'].values
Y_val=df_val['label'].values
```

In [7]:
```
(X_train.shape,Y_train.shape),(X_val.shape,Y_val.shape)
```

Out[7]:
```
(((40000,), (40000,)), ((5000,), (5000,)))
```

Analyse the Data

In [8]:
```
df_train.iloc[:,1].describe()
```

Out[8]:
```
count    40000.000000
mean         0.499525
std          0.500006
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: label, dtype: float64
```

In [9]:
```
df_val.iloc[:,1].describe()
```

Out[9]:
```
count    5000.000000
mean        0.502800
std         0.500042
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: label, dtype: float64
```

In [10]:
```
X_val_len=[len(str(i).split()) for i in X_val]
df1=pd.DataFrame(X_val_len,columns=['len'])
df1.describe()
```

Out[10]:

	len
count	5000.00000
mean	228.93260
std	169.33721
min	10.00000
25%	126.00000
50%	171.00000
75%	274.00000
max	1601.00000

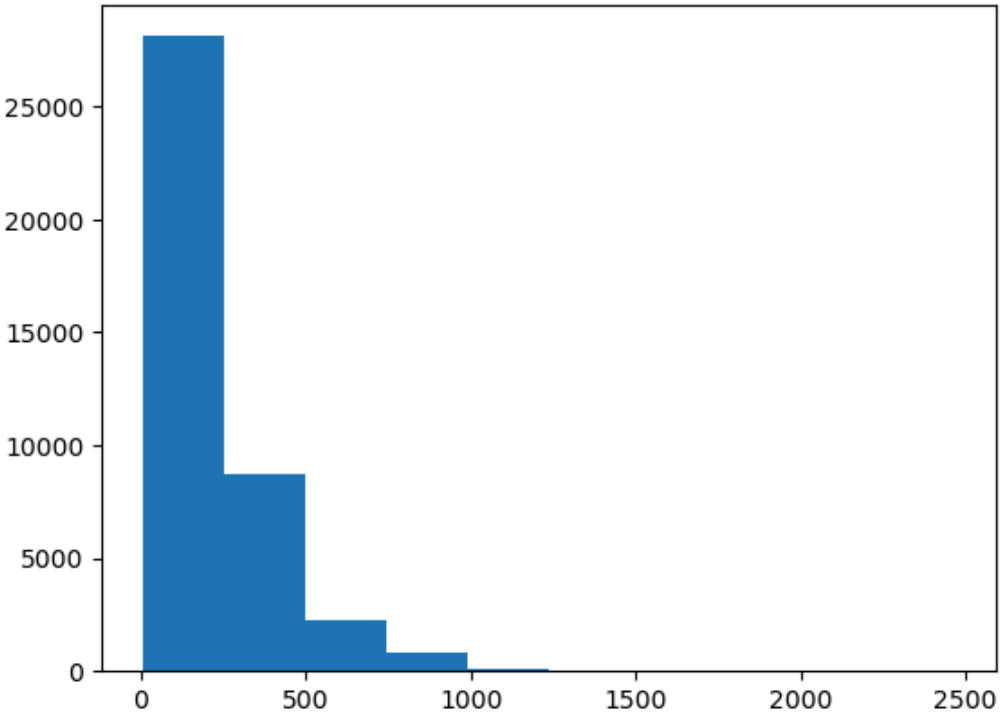
In [11]:
```
X_train_len=[len(str(i).split()) for i in X_train]
df=pd.DataFrame(X_train_len,columns=['len'])
df.describe()
```

Out[11]:

	len
count	40000.000000
mean	231.339250
std	171.194123
min	4.000000
25%	126.000000
50%	173.000000
75%	282.000000
max	2470.000000

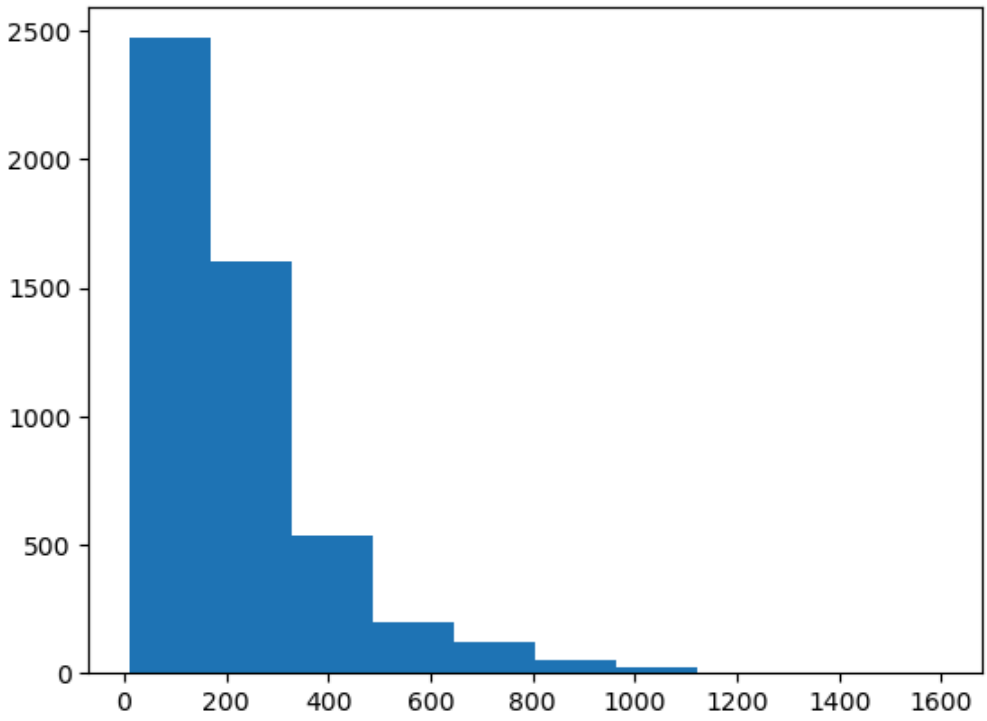
In [12]:
```
X_train_len=[len(str(i).split()) for i in X_train]
plt.hist(X_train_len)
```

Out[12]:
```
(array([2.8097e+04, 8.6960e+03, 2.2520e+03, 8.0100e+02, 1.3800e+02,
        7.0000e+00, 3.0000e+00, 3.0000e+00, 1.0000e+00, 2.0000e+00]),
 array([   4. ,  250.6,  497.2,  743.8,  990.4, 1237. , 1483.6, 1730.2,
        1976.8, 2223.4, 2470. ]),
 <BarContainer object of 10 artists>)
```



In [13]:
```
X_val_len=[len(str(i).split()) for i in X_val]
plt.hist(X_val_len)
```

Out[13]:
```
(array([2.470e+03, 1.602e+03, 5.350e+02, 1.970e+02, 1.180e+02, 5.100e+01,
        2.300e+01, 1.000e+00, 2.000e+00, 1.000e+00]),
 array([  10. ,  169.1,  328.2,  487.3,  646.4,  805.5,  964.6, 1123.7,
        1282.8, 1441.9, 1601. ]),
 <BarContainer object of 10 artists>)
```



Setting the parameters

In [14]:
```
from tensorflow.keras.datasets import imdb
# Load the IMDb movie review dataset
(X_train, y_train), (X_val, y_val) = imdb.load_data()

# Load word index to convert integer sequences back to text
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

# Retrieve the actual text data from integer sequences
X_train_text = [' '.join([reverse_word_index.get(i - 3, '') for i in sequence]) for s
X_val_text = [' '.join([reverse_word_index.get(i - 3, '') for i in sequence]) for seq

# Define tokenizer and preprocess the sequences
vocab_size = 30000  #went for an average vocab size
embedding_dimension = 64 #high dimensions would result in finding better parameters f
max_length = 120 #used a maximum length of 120 words
turnc = 'post' #preprocessing step for pad_sequences
oov_tok = '<OOV>' #oov stands for out of vocabulary tokens
```

Tokenizing and converting the data into Sequences

In [15]:
```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(X_train_text)

X_train_seq = tokenizer.texts_to_sequences(X_train_text)
X_train_padded = pad_sequences(X_train_seq, maxlen=max_length, padding='post', trunca
X_val_seq = tokenizer.texts_to_sequences(X_val_text)
X_val_padded = pad_sequences(X_val_seq, maxlen=max_length, padding='post', truncating
```

In [16]:
```
X_train_padded.shape,X_val_padded.shape
```

Out[16]:
```
((25000, 120), (25000, 120))
```

The Model

In [17]:
```
from tensorflow.keras.layers import LSTM, Bidirectional, Embedding, Dense, SpatialDro
from tensorflow.keras.models import Sequential

# Define the model
model = Sequential()
embedding_layer = Embedding(input_dim=vocab_size, output_dim=embedding_dimension, inpu
model.add(embedding_layer)
model.add(SpatialDropout1D(0.4))
model.add(Bidirectional(LSTM(120, activation='tanh', return_sequences=True)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(120, activation='tanh', return_sequences=False)))
model.add(Dropout(0.2))
model.add(Dense(300, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary
print(model.summary())
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.c
ompat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Pleas
e use tf.compat.v1.train.Optimizer instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 120, 64)           1920000

 spatial_dropout1d (Spatial  (None, 120, 64)           0
 Dropout1D)

 bidirectional (Bidirection  (None, 120, 240)          177600
 al)

 dropout (Dropout)           (None, 120, 240)          0

 bidirectional_1 (Bidirecti  (None, 240)               346560
 onal)

 dropout_1 (Dropout)         (None, 240)               0

 dense (Dense)               (None, 300)               72300

 dropout_2 (Dropout)         (None, 300)               0

 dense_1 (Dense)             (None, 1)                 301

=================================================================
Total params: 2516761 (9.60 MB)
Trainable params: 2516761 (9.60 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [18]:
```
hist=model.fit(X_train_padded,y_train,epochs=7,batch_size=16,validation_data=(X_val_p
```

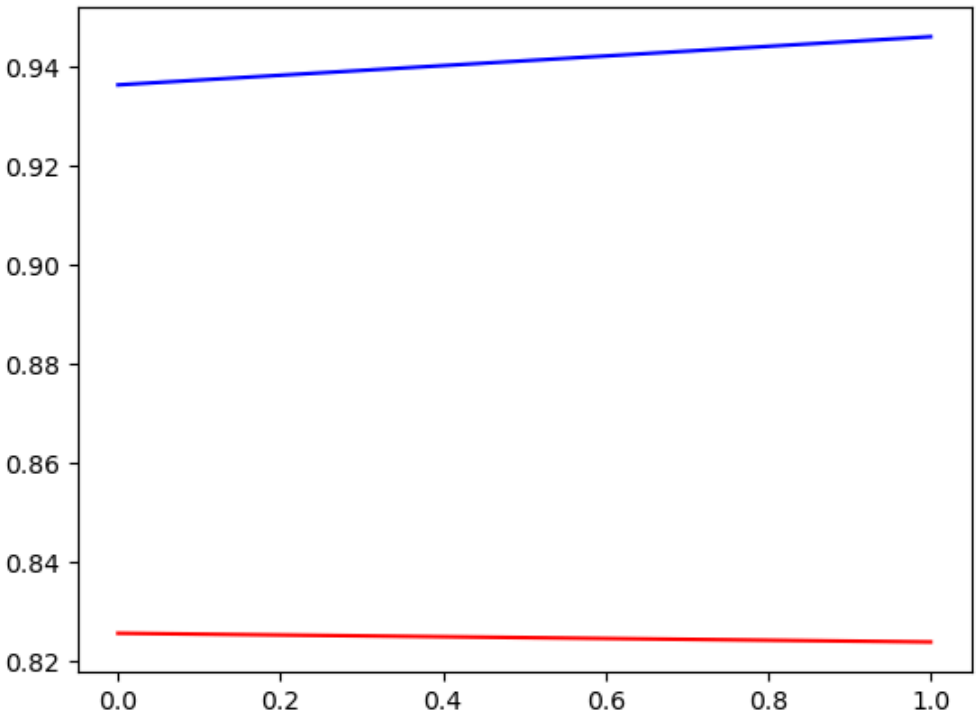
```
Epoch 1/7
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. P
lease use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functi
ons is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions inste
ad.

1563/1563 [==============================] - 794s 497ms/step - loss: 0.5323 - accura
cy: 0.7200 - val_loss: 0.3863 - val_accuracy: 0.8290
Epoch 2/7
1563/1563 [==============================] - 691s 442ms/step - loss: 0.3555 - accura
cy: 0.8502 - val_loss: 0.3885 - val_accuracy: 0.8173
Epoch 3/7
1563/1563 [==============================] - 650s 416ms/step - loss: 0.3143 - accura
cy: 0.8745 - val_loss: 0.3784 - val_accuracy: 0.8286
Epoch 4/7
1563/1563 [==============================] - 679s 435ms/step - loss: 0.2795 - accura
cy: 0.8894 - val_loss: 0.6437 - val_accuracy: 0.7637
Epoch 5/7
1563/1563 [==============================] - 955s 611ms/step - loss: 0.2483 - accura
cy: 0.9058 - val_loss: 0.3735 - val_accuracy: 0.8371
Epoch 6/7
1563/1563 [==============================] - 919s 588ms/step - loss: 0.2270 - accura
cy: 0.9177 - val_loss: 0.4115 - val_accuracy: 0.8309
Epoch 7/7
1563/1563 [==============================] - 622s 398ms/step - loss: 0.2004 - accura
cy: 0.9272 - val_loss: 0.4894 - val_accuracy: 0.8236
```

In [19]:
```
hist=model.fit(X_train_padded,y_train,epochs=2,batch_size=16,validation_data=(X_val_p
```

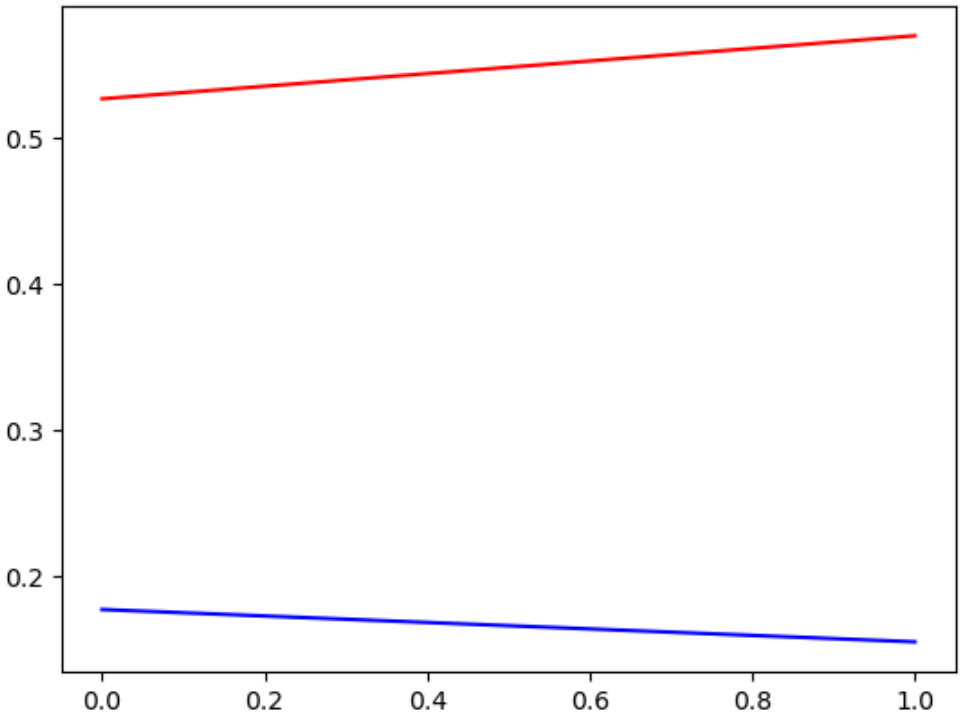
```
Epoch 1/2
1563/1563 [==============================] - 510s 327ms/step - loss: 0.1773 - accura
cy: 0.9362 - val_loss: 0.5264 - val_accuracy: 0.8256
Epoch 2/2
1563/1563 [==============================] - 548s 351ms/step - loss: 0.1552 - accura
cy: 0.9460 - val_loss: 0.5695 - val_accuracy: 0.8239
```

This Plot is for the last two Epochs

In [20]:
```
plt.plot(hist.history['accuracy'],c='b')
plt.plot(hist.history['val_accuracy'],c='r')
plt.show()
```



In [21]:
```
plt.plot(hist.history['loss'],c='b')
plt.plot(hist.history['val_loss'],c='r')
plt.show()
```



Reading the Test Data

In [22]:
```
df_test=pd.read_csv('Test.csv')
df_test.head()
```

Out[22]:

	text	label
0	I always wrote this series off as being a comp...	0
1	1st watched 12/7/2002 - 3 out of 10(Dir-Steve ...	0
2	This movie was so poorly written and directed ...	0
3	The most interesting thing about Miryang (Secr...	1
4	when i first read about "berlin am meer" i did...	0

In [23]:
```
X_test=df_test['text'].values
Y_test=df_test['label'].values
```

Converting into Sequential Data

In [24]:
```
X_test_seq=tokenizer.texts_to_sequences(X_test)
```

In [25]:
```
X_test_padded=pad_sequences(X_test_seq,maxlen=max_length,padding='post',truncating=tu
X_test_padded[0]
```

Out[25]:
```
array([   11,   208,  1040,    12,   199,   123,    15,   110,     4,
         598,  7221,  3338,    86,  1239,  5390,    14,   573,     9,
          10,     3,  2697,    19,    93,    29,   249,     4,  1576,
       13809,  3847,   101,     4,   426,  5744,   498,  1052,    11,
        1703,    13,     2,  2951,    14,    30,     2,    94,    21,
           2,    83,   497,     5,     2,   671,   818,   148,    11,
          98,    26,    41,  1838,    54,     3,  2530,   636,     2,
         671,     6,    77,     2,  2951,    40,    58,     6,     2,
         246,     6,   469,     2,  1306,    19,    93,   136,    22,
          41,    77,    54,     3,  1133,   636,     2,   702,     6,
         104,   246,     9,   158,  1109,  4764,     6,    13,    11,
         299,    36,    11,   870,     6,    41,  3250,  2704,    21,
           2,  5734,     3,   191,   842,  1947,    67,     9,  1132,
          16,    70,    49])
```

In [26]:
```
X_test_padded.shape
```

Out[26]:
```
(5000, 120)
```

In [27]:
```
model.evaluate(X_test_padded,Y_test)
```

```
157/157 [==============================] - 22s 140ms/step - loss: 0.3230 - accuracy:
0.8952
```

Out[27]:
```
[0.32298383116722107, 0.8952000141143799]
```

Check for your own Reviews

In [34]:
```
def Check(x):
    turnc = 'post'  # Define the truncating method, e.g., 'post' or 'pre'
    test_case1 = [x]
    test_case = tokenizer.texts_to_sequences(test_case1)
    max_sequence_length = 120  # Define the maximum sequence length expected by the m
    test_case_padded = pad_sequences(test_case, maxlen=max_sequence_length, padding='
    predict_x = model.predict(test_case_padded)
    print(predict_x)
    if predict_x >= 0.5:
        print("Positive")
    else:
        print("Negative")
```

In [37]:
```
test_review=str(input("Enter the review :  "))
Check(test_review)
```

```
Enter the review :  I learned about this movie couple of weeks ago through a faceboo
k page. The storyline caught my attention and decided to watch it.and am glad i did.
It is dark,dramatic and a bit of humor.I had to mention humor because it wasn't like
those comedy scenes that you find in other Bollywood films.It made sense and that we
nt with the story.'Shaitan' is realistic in every sense.  Movie has five main charac
ters and the whole story revolves around the 'events' that they did.It shows the lif
e of 5 young friends who is living life like hell.They drink,they shop lift,they foo
l around,pretty much everything that makes you feel alive.But a bit too much adrenal
in rush followed by some unwise decisions turns their life around.
1/1 [==============================] - 0s 47ms/step
[[0.99900836]]
Positive
```

I just checked for one random imdb review

In [38]:
```
test_review="You will get A to Z all details of this scam, i may be wrong but due to t
Check(test_review)
```

```
1/1 [==============================] - 0s 51ms/step
[[0.07417779]]
Negative
```

Practical 3

Convolutional neural network (CNN): Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

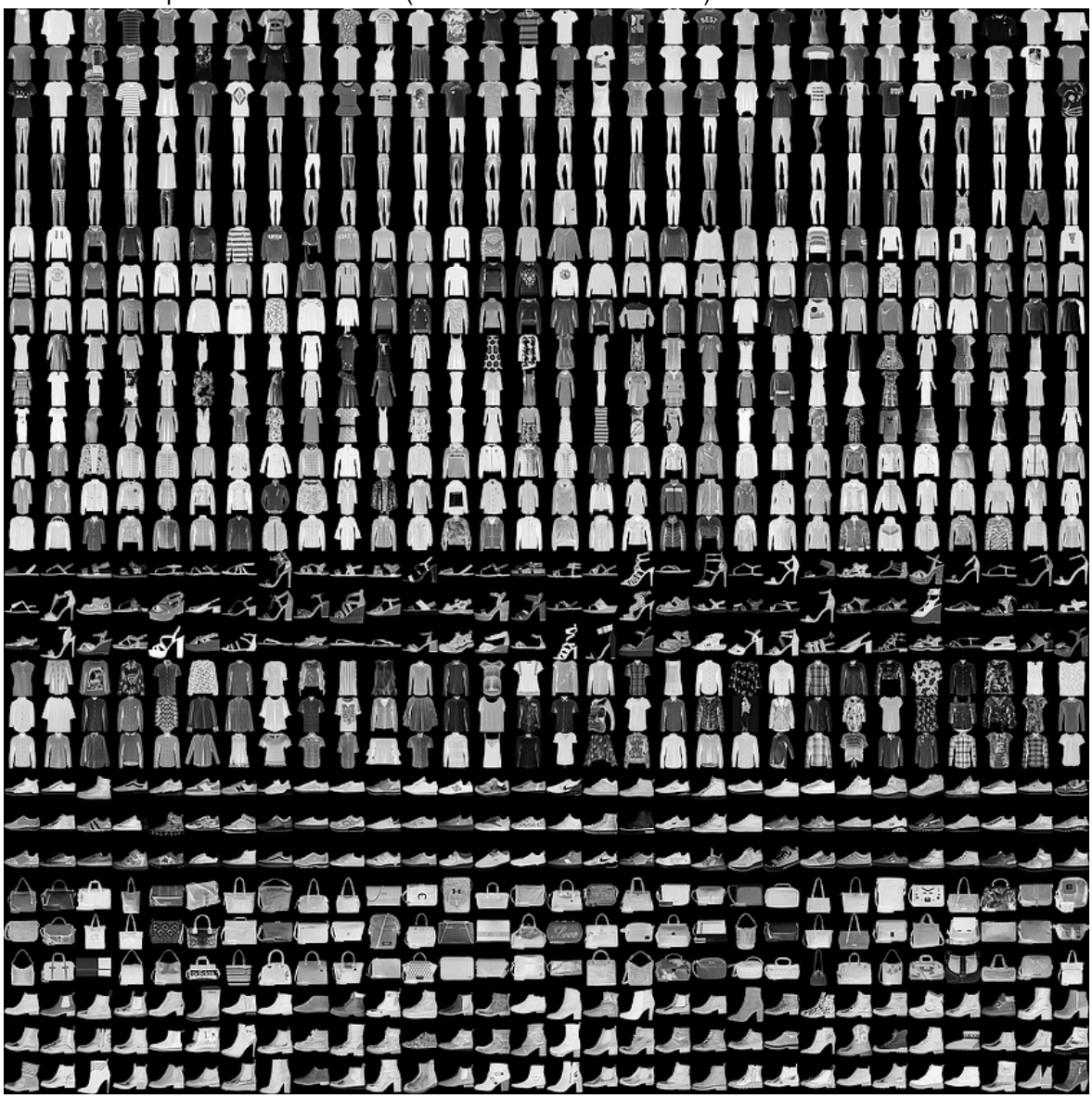
Import the required libraries

In [ ]:
```
#!pip install git+https://github.com/tensorflow/tensorflow.git@v2.15.0

#!pip install keras
```

Fashion MNIST Classification

Fashion-MNIST is a dataset of Zalando's article images—consisting of a **training set of 60,000** examples and a **test set of 10,000 examples**. Each example is a **28x28 grayscale** image, associated with a label from **10 classes**. We intend Fashion-MNIST to serve as a direct drop-in **replacement for the original MNIST** dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Here's an example how the data looks (each class takes three-rows):



Step # 1 - Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sbn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Flatten
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
from keras.utils import to_categorical
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecate
d. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Step # 2 - Load Data

In [2]:
```
fashion_train_df = pd.read_csv('dataset/fashion-mnist_train.csv', sep=',')
fashion_test_df = pd.read_csv('dataset//fashion-mnist_test.csv', sep=',')
```

Now that we have loaded the datasets, lets check some parameters about the datasets.

In [3]:
```
fashion_train_df.shape   # Shape of the dataset
```

Out[3]:
```
(60000, 785)
```

In [4]:
```
fashion_train_df.columns   # Name of the columns of the DataSet.
```

Out[4]:
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6',
       'pixel7', 'pixel8', 'pixel9',
       ...
       'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779', 'pixel780',
       'pixel781', 'pixel782', 'pixel783', 'pixel784'],
      dtype='object', length=785)
```

In [5]:
```
print(set(fashion_train_df['label']))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [6]:
```
print([fashion_train_df.drop(labels='label', axis=1).min(axis=1).min(),
      fashion_train_df.drop(labels='label', axis=1).max(axis=1).max()])
```

```
[0, 255]
```

So we have 0 to 255 which is the color values for grayscale. 0 being white and 255 being black.

Now lets check some of the rows in tabular format

In [7]:
```
fashion_train_df.head()
```

Out[7]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel77
0	2	0	0	0	0	0	0	0	0	0	...	0	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	
3	0	0	0	0	1	2	0	0	0	0	...	3	0	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	

5 rows × 785 columns

So evry other things of the test dataset are going to be the same as the train dataset except the shape.

In [8]:
```
fashion_test_df.shape
```

Out[8]:
```
(10000, 785)
```

So here we have 10000 images instead of 60000 as in the train dataset.

Lets check first few rows.

In [9]:
```
fashion_test_df.head()
```

Out[9]:

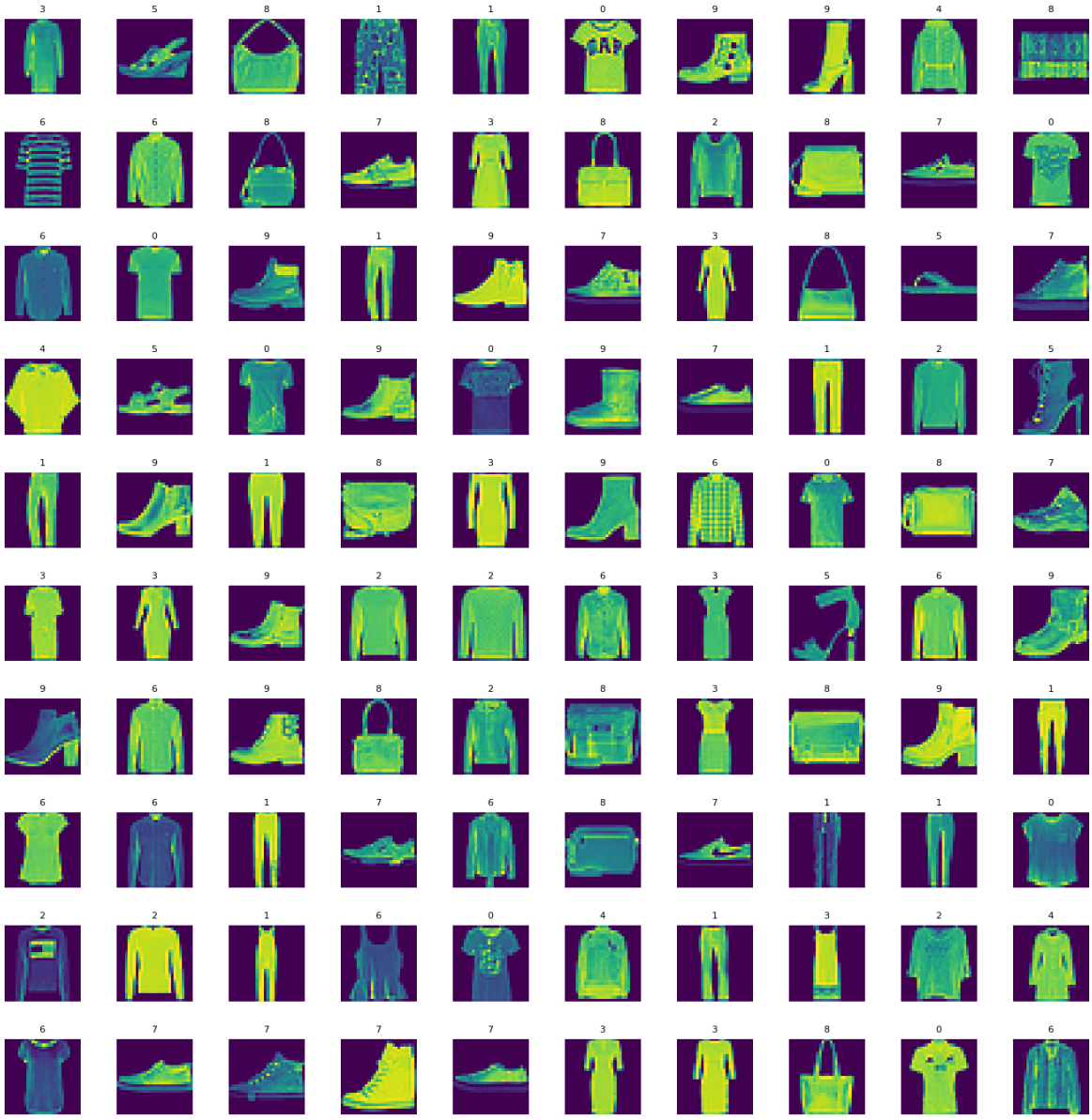
	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel77
0	0	0	0	0	0	0	0	0	9	8	...	103	87	5
1	1	0	0	0	0	0	0	0	0	0	...	34	0	
2	2	0	0	0	0	0	0	14	53	99	...	0	0	
3	2	0	0	0	0	0	0	0	0	0	...	137	126	14
4	3	0	0	0	0	0	0	0	0	0	...	0	0	

5 rows × 785 columns

Step # 3 - Visualization

Now that we have loaded the data and also got somewhat acquainted with it lets visualize the actual images. We are going to use **Matplotlib** library for this.

In [10]:
```
# Convert the dataframe ti numpy array
training = np.asarray(fashion_train_df, dtype='float32')

# Lets show multiple images in a 15x15 grid
height = 10
width = 10

fig, axes = plt.subplots(nrows=width, ncols=height, figsize=(17,17))
axes = axes.ravel()  # this flattens the 15x15 matrix into 225
n_train = len(training)

for i in range(0, height*width):
    index = np.random.randint(0, n_train)
    axes[i].imshow(training[index, 1:].reshape(28,28))
    axes[i].set_title(int(training[index, 0]), fontsize=8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.5)
```



Step # 4 - Preprocess Data

Visualized the images. So now we can start preparing for creating our model. But before that we need to preprocess our data so that we can fit our model easily. Lets do that first.

Since we are dealing with image data and our task is to recognize and classify images our model should be a Convolutional Neural Network. For that our images should have atleast 3 dimensions (**height** x **width** x **color_channels**). But our images are flattened in one dimension, **784 pixel (28x28x1)** values per row. So we need to reshape the data into its original format

In [11]:
```
# convert to numpy arrays and reshape
training = np.asarray(fashion_train_df, dtype='float32')
X_train = training[:, 1:].reshape([-1,28,28,1])
X_train = X_train/255   # Normalizing the data
y_train = training[:, 0]

testing = np.asarray(fashion_test_df, dtype='float32')
X_test = testing[:, 1:].reshape([-1,28,28,1])
X_test = X_test/255    # Normalizing the data
y_test = testing[:, 0]
```

Also we need to have three different sets of data for **training, validatin** and **testing**. We already have different sets for training and testing. So we are going to split the training dataset further into two sets and will use one set of training and the other for validation.

In [12]:
```
# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, ra
```

In [13]:
```
# Lets check the shape of all three datasets
print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(48000, 28, 28, 1) (12000, 28, 28, 1) (10000, 28, 28, 1)
(48000,) (12000,) (10000,)
```

Step # 5 - Create and Train the Model

Create the model

In [14]:
```
cnn_model = Sequential()
cnn_model.add(Conv2D(filters=64, kernel_size=(3,3), input_shape=(28,28,1), activation=
cnn_model.add(MaxPooling2D(pool_size = (2,2)))
cnn_model.add(Dropout(rate=0.3))
cnn_model.add(Flatten())
cnn_model.add(Dense(units=32, activation='relu'))
cnn_model.add(Dense(units=10, activation='sigmoid'))
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.c
ompat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated.
Please use tf.nn.max_pool2d instead.
```

compile the model

In [15]:
```
cnn_model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crosse
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 max_pooling2d (MaxPooling2  (None, 13, 13, 64)        0
 D)

 dropout (Dropout)           (None, 13, 13, 64)        0

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 32)                346144

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 347114 (1.32 MB)
Trainable params: 347114 (1.32 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train the model

In [16]:
```
cnn_model.fit(x=X_train, y=y_train, batch_size=512, epochs=50, validation_data=(X_val
```

```
94/94 [==============================] - 47s 502ms/step - loss: 0.1344 - accurac
y: 0.9523 - val_loss: 0.2502 - val_accuracy: 0.9178
Epoch 45/50
94/94 [==============================] - 44s 467ms/step - loss: 0.1326 - accurac
y: 0.9524 - val_loss: 0.2528 - val_accuracy: 0.9173
Epoch 46/50
94/94 [==============================] - 41s 434ms/step - loss: 0.1341 - accurac
y: 0.9507 - val_loss: 0.2559 - val_accuracy: 0.9142
Epoch 47/50
94/94 [==============================] - 44s 468ms/step - loss: 0.1280 - accurac
y: 0.9540 - val_loss: 0.2514 - val_accuracy: 0.9165
Epoch 48/50
94/94 [==============================] - 45s 483ms/step - loss: 0.1267 - accurac
y: 0.9542 - val_loss: 0.2551 - val_accuracy: 0.9158
Epoch 49/50
94/94 [==============================] - 43s 462ms/step - loss: 0.1251 - accurac
y: 0.9543 - val_loss: 0.2518 - val_accuracy: 0.9164
Epoch 50/50
94/94 [==============================] - 43s 457ms/step - loss: 0.1253 - accurac
y: 0.9547 - val_loss: 0.2481 - val_accuracy: 0.9200
```

Step # 5 - Evaluate the Model

Get the accuracy of the model

In [17]:
```
eval_result = cnn_model.evaluate(X_test, y_test)
print("Accuracy : {:.3f}".format(eval_result[1]))
```

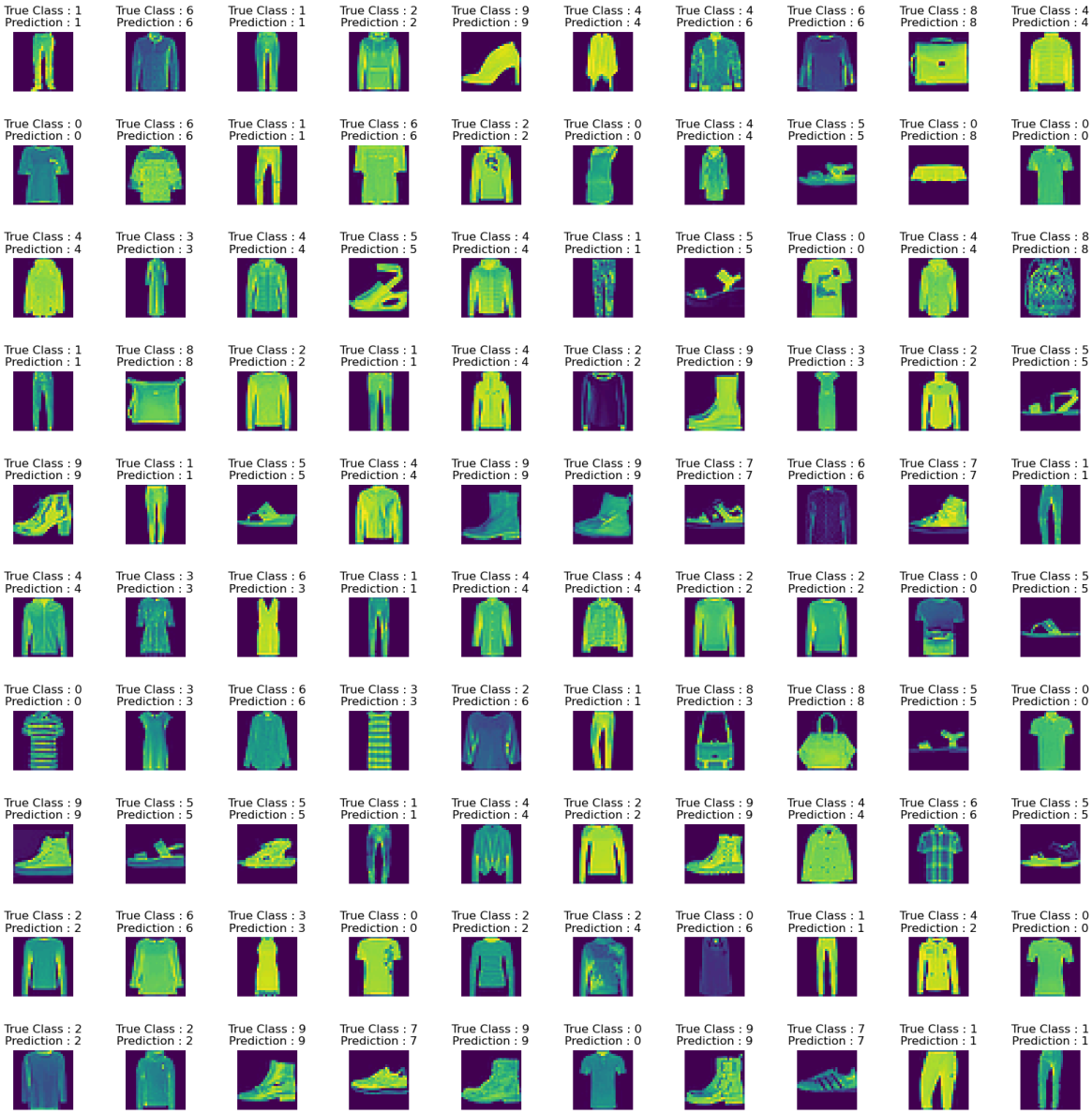
```
313/313 [==============================] - 5s 14ms/step - loss: 0.2446 - accuracy:
0.9210
Accuracy : 0.921
```

Visualize the model's predictions

In [18]:
```
predict_x=cnn_model.predict(X_test)
classes_x=np.argmax(predict_x,axis=1)
```

```
313/313 [==============================] - 6s 12ms/step
```

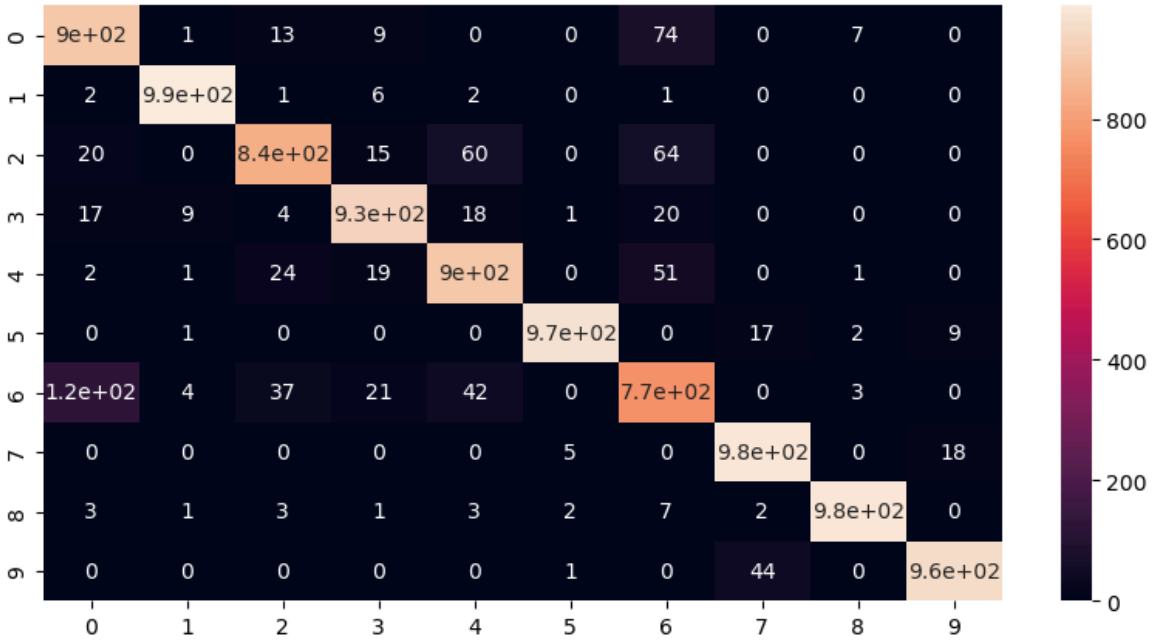
In [19]:
```
height = 10
width = 10

fig, axes = plt.subplots(nrows=width, ncols=height, figsize=(20,20))
axes = axes.ravel()
for i in range(0, height*width):
    index = np.random.randint(len(classes_x))
    axes[i].imshow(X_test[index].reshape((28,28)))
    axes[i].set_title("True Class : {:0.0f}\nPrediction : {:d}".format(y_test[index],
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.9, wspace=0.5)
```



Plot Confusin Matrix

In [20]:
```
cm = confusion_matrix(y_test, classes_x)
plt.figure(figsize=(10,5))
sbn.heatmap(cm, annot=True)
```

Out[20]: <Axes: >



Classification Report

In [21]:
```
num_classes = 10
class_names = ["class {}".format(i) for i in range(num_classes)]
cr = classification_report(y_test, classes_x, target_names=class_names)
print(cr)
```

```
              precision    recall  f1-score   support

     class 0       0.84      0.90      0.87      1000
     class 1       0.98      0.99      0.99      1000
     class 2       0.91      0.84      0.87      1000
     class 3       0.93      0.93      0.93      1000
     class 4       0.88      0.90      0.89      1000
     class 5       0.99      0.97      0.98      1000
     class 6       0.78      0.77      0.78      1000
     class 7       0.94      0.98      0.96      1000
     class 8       0.99      0.98      0.98      1000
     class 9       0.97      0.95      0.96      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

Practical 4

Recurrent neural network (RNN): Use the Google stock prices dataset and design a timecseriesanalysis and prediction system using RNN.

1. Import library

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecate
d. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2. Data processing

2.0 import the data

In [2]:
```
dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
```

In [3]:
```
dataset_train.head()
```

Out[3]:

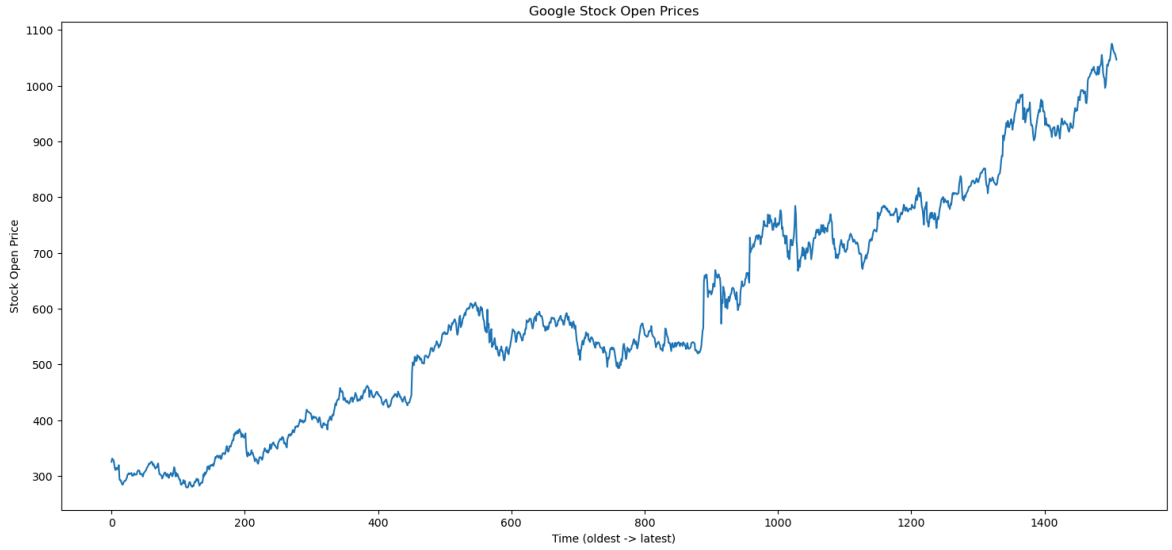
	Date	Open	High	Low	Close	Volume
0	01/03/2012	325.25	332.83	324.97	663.59	7,380,500
1	01/04/2012	331.27	333.87	329.08	666.45	5,749,400
2	01/05/2012	329.83	330.75	326.89	657.21	6,590,300
3	01/06/2012	328.34	328.77	323.68	648.24	5,405,900
4	01/09/2012	322.04	322.29	309.46	620.76	11,688,800

In [4]:
```
#keras only takes numpy array
training_set = dataset_train.iloc[:, 1: 2].values
```

In [5]:
```
training_set.shape
```

Out[5]:
```
(1509, 1)
```

In [6]:
```
plt.figure(figsize=(18, 8))
plt.plot(dataset_train['Open'])
plt.title("Google Stock Open Prices")
plt.xlabel("Time (oldest -> latest)")
plt.ylabel("Stock Open Price")
plt.show()
```



2.1 Feature scaling

In [7]:
```
import os
if os.path.exists('config.py'):
    print(1)
else:
    print(0)
```

```
0
```

In [8]:
```
sc = MinMaxScaler(feature_range = (0, 1))
#fit: get min/max of train data
training_set_scaled = sc.fit_transform(training_set)
```

2.2 Data structure creation

- taking the reference of past 60 days of data to predict the future stock price.
- It is observed that taking 60 days of past data gives us best results.
- In this data set 60 days of data means 3 months of data.
- Every month as 20 days of Stock price.
- X train will have data of 60 days prior to our date and y train will have data of one day after our date

In [9]:
```
## 60 timesteps and 1 output
X_train = []
y_train = []
for i in range(60, len(training_set_scaled)):
    X_train.append(training_set_scaled[i-60: i, 0])
    y_train.append(training_set_scaled[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
```

In [10]:
```
X_train.shape
```

Out[10]:
```
(1449, 60)
```

In [11]:
```
y_train.shape
```

Out[11]:
```
(1449,)
```

2.3 Data reshaping

In [12]:
```
X_train = np.reshape(X_train, newshape =
                     (X_train.shape[0], X_train.shape[1], 1))
```

1. Number of stock prices - 1449
2. Number of time steps - 60
3. Number of Indicator - 1

In [13]:
```
X_train.shape
```

Out[13]:
```
(1449, 60, 1)
```

3. Create & Fit Model

3.1 Create model

In [14]:
```
regressor = Sequential()
#add 1st lstm layer
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1
regressor.add(Dropout(rate = 0.2))

##add 2nd lstm layer: 50 neurons
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(rate = 0.2))

##add 3rd lstm layer
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(rate = 0.2))

##add 4th lstm layer
regressor.add(LSTM(units = 50, return_sequences = False))
regressor.add(Dropout(rate = 0.2))

##add output layer
regressor.add(Dense(units = 1))
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.c
ompat.v1.get_default_graph instead.
```

In [15]:
```
regressor.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 50)            10400

 dropout (Dropout)           (None, 60, 50)            0

 lstm_1 (LSTM)               (None, 60, 50)            20200

 dropout_1 (Dropout)         (None, 60, 50)            0

 lstm_2 (LSTM)               (None, 60, 50)            20200

 dropout_2 (Dropout)         (None, 60, 50)            0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 71051 (277.54 KB)
Trainable params: 71051 (277.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [16]:
```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
WARNING:tensorflow:From C:\Users\Yashwardhan Deshmukh\anaconda3\lib\site-packages\ke
ras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Pleas
e use tf.compat.v1.train.Optimizer instead.
```

3.2 Model fit

In [17]:
```
regressor.fit(x = X_train, y = y_train, batch_size = 32, epochs = 100)
```

```
Epoch 92/100
46/46 [==============================] - 13s 274ms/step - loss: 9.1957e-04
Epoch 93/100
46/46 [==============================] - 13s 282ms/step - loss: 9.3358e-04
Epoch 94/100
46/46 [==============================] - 13s 294ms/step - loss: 0.0010
Epoch 95/100

46/46 [==============================] - 14s 310ms/step - loss: 9.5691e-04
Epoch 96/100
46/46 [==============================] - 14s 300ms/step - loss: 8.3259e-04
Epoch 97/100
46/46 [==============================] - 12s 261ms/step - loss: 9.6896e-04
Epoch 98/100
46/46 [==============================] - 11s 234ms/step - loss: 8.9036e-04
Epoch 99/100
46/46 [==============================] - 11s 236ms/step - loss: 9.6050e-04
Epoch 100/100
46/46 [==============================] - 11s 237ms/step - loss: 9.2182e-04
```

3.3 Model evaluation

3.3.1 Read and convert

In [18]:
```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
```

In [19]:
```
dataset_test.head()
```

Out[19]:

	Date	Open	High	Low	Close	Volume
0	02/01/2018	1048.339966	1066.939941	1045.229980	1065.000000	1237600
1	03/01/2018	1064.310059	1086.290039	1063.209961	1082.479980	1430200
2	04/01/2018	1088.000000	1093.569946	1084.001953	1086.400024	1004600
3	05/01/2018	1094.000000	1104.250000	1092.000000	1102.229980	1279100
4	08/01/2018	1102.229980	1111.270020	1101.619995	1106.939941	1047600

In [20]:
```
#keras only takes numpy array
real_stock_price = dataset_test.iloc[:, 1: 2].values
real_stock_price.shape
```

Out[20]:
```
(125, 1)
```

3.3.2 Concat and convert

In [21]:
```
#vertical concat use 0, horizontal uses 1
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']),
                          axis = 0)
##use .values to make numpy array
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
```

3.3.3 Reshape and scale

In [22]:
```
#reshape data to only have 1 col
inputs = inputs.reshape(-1, 1)

#scale input
inputs = sc.transform(inputs)
```

In [23]:
```
len(inputs)
```

Out[23]:
```
185
```

3.3.4 Create test data strucutre

In [24]:
```
X_test = []
for i in range(60, len(inputs)):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
#add dimension of indicator
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

In [25]:
```
X_test.shape
```

Out[25]:
```
(125, 60, 1)
```

3.3.5 Model prediction

In [26]:
```
predicted_stock_price = regressor.predict(X_test)
```

```
4/4 [==============================] - 26s 92ms/step
```

In [27]:
```
#inverse the scaled value
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

3.3.6 Result visualization

In [28]:
```
##visualize the prediction and real price
plt.plot(real_stock_price, color = 'red', label = 'Real price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted price')

plt.title('Google price prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```