In [1]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [4]:
```
data = pd.DataFrame(boston.data)
```

In [5]:
```
data.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [6]:
```
#Adding the feature names to the dataframe
data.columns = boston.feature_names
data.head()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.

In [7]:
```
#Adding target variable to dataframe
data['PRICE'] = boston.target
```

In [8]:
```
#Check the shape of dataframe
data.shape
```

Out[8]:
```
(506, 14)
```

In [9]:
```
data.columns
```

Out[9]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
X',
       'PTRATIO', 'B', 'LSTAT', 'PRICE'],
      dtype='object')
```

In [10]:
```
data.dtypes
```

Out[10]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
```

In [11]:
```
# Identifying the unique number of values in the dataset
data.nunique()
```

Out[11]:
```
CRIM       504
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        356
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      455
PRICE      229
dtype: int64
```

In [12]:
```
# Check for missing values
data.isnull().sum()
```

Out[12]:
```
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
```

In [13]:
```
# See rows with missing values
data[data.isnull().any(axis=1)]
```

Out[13]:

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE

In [14]:
```
# Viewing the data statistics
data.describe()
```

Out[14]:

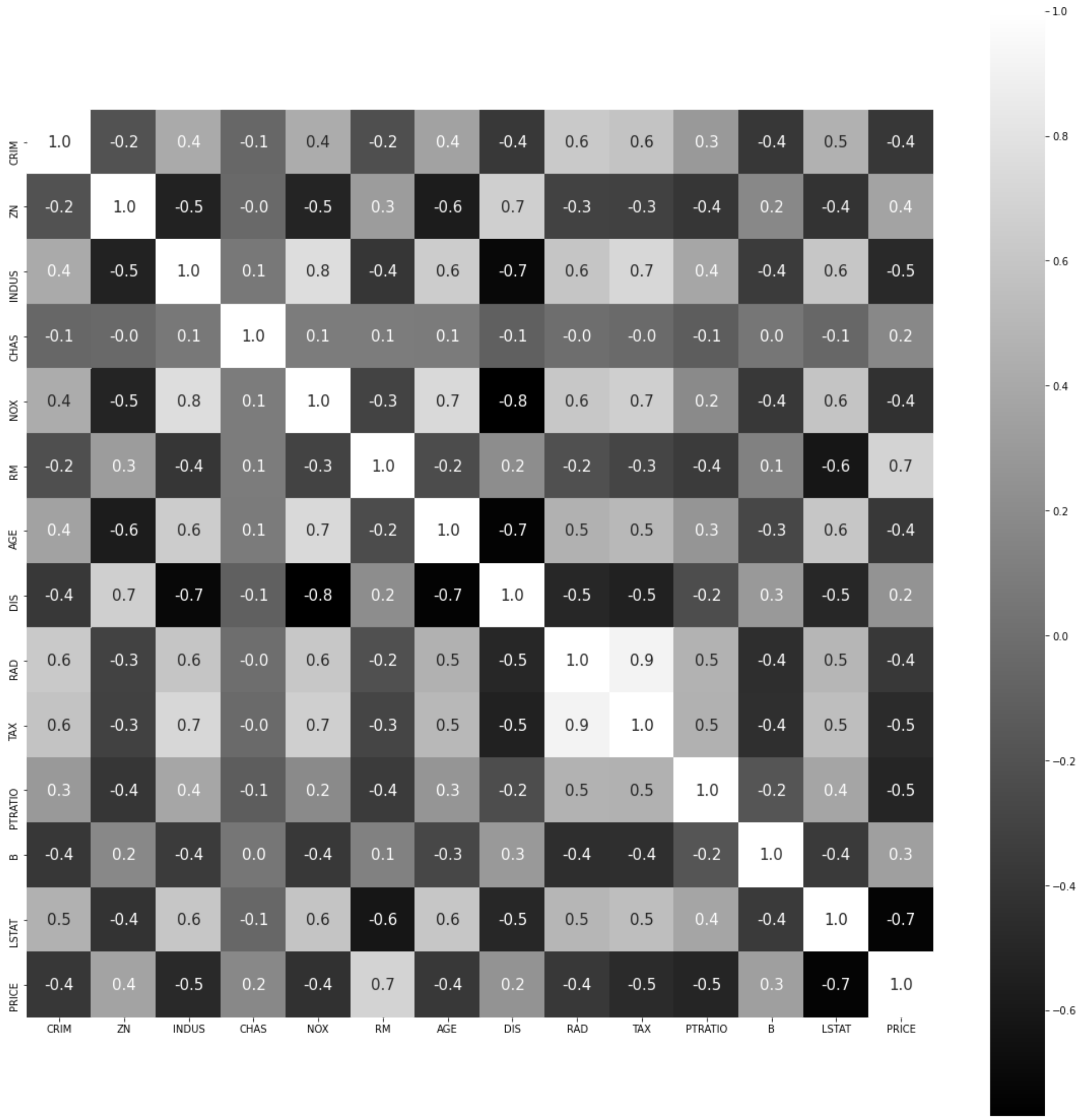
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12

In [15]:
```
# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

Out[15]:
```
(14, 14)
```

In [16]:
```
# Plotting the heatmap of correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'
```

Out[16]: <AxesSubplot:>



In [17]:
```
# Spliting target variable and independent variables
X = data.drop(['PRICE'], axis = 1)
y = data['PRICE']
```

In [18]:
```
# Splitting to training and testing data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, ranc
```

In [19]:
```
# Import library for Linear Regression
from sklearn.linear_model import LinearRegression
```

In [20]:
```
# Create a Linear regressor
lm = LinearRegression()

# Train the model using the training sets
lm.fit(X_train, y_train)
```

Out[20]: LinearRegression()

In [21]:
```
# Value of y intercept
lm.intercept_
```

Out[21]: 36.357041376594815

In [22]:
```
#Converting the coefficient values to a dataframe
coeffcients = pd.DataFrame([X_train.columns,lm.coef_]).T
coeffcients = coeffcients.rename(columns={0: 'Attribute', 1: 'Coefficients'})
coeffcients
```

Out[22]:

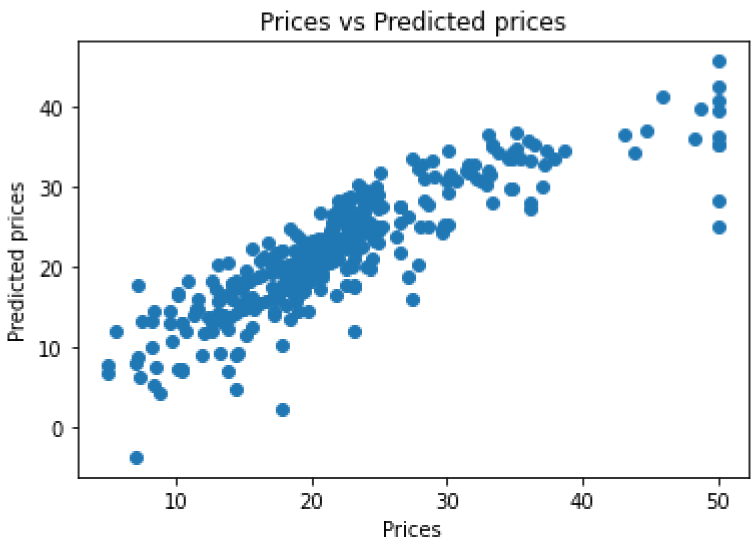
	Attribute	Coefficients
0	CRIM	-0.12257
1	ZN	0.055678
2	INDUS	-0.008834
3	CHAS	4.693448
4	NOX	-14.435783
5	RM	3.28008
6	AGE	-0.003448
7	DIS	-1.552144
8	RAD	0.32625
9	TAX	-0.014067
10	PTRATIO	-0.803275
11	B	0.009354
12	LSTAT	-0.523478

In [23]:
```
# Model prediction on train data
y_pred = lm.predict(X_train)
```

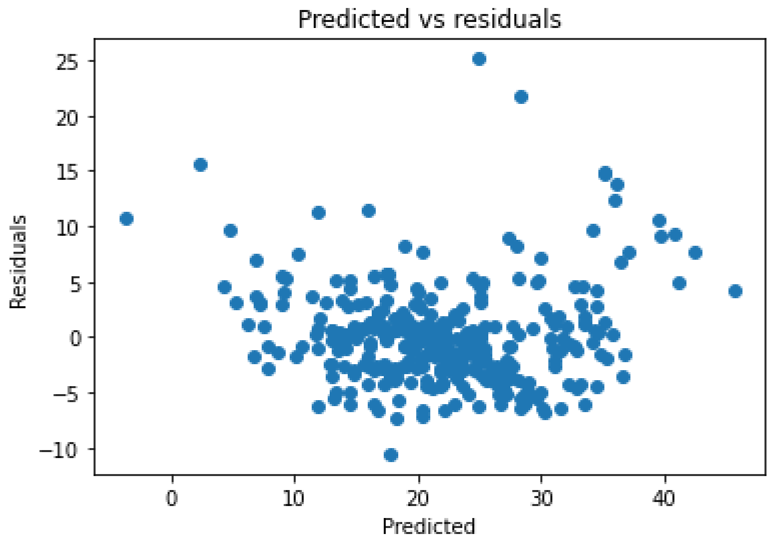
In [24]:
```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.7465991966746854
Adjusted R^2: 0.736910342429894
MAE: 3.08986109497113
MSE: 19.07368870346903
RMSE: 4.367343437774162
```

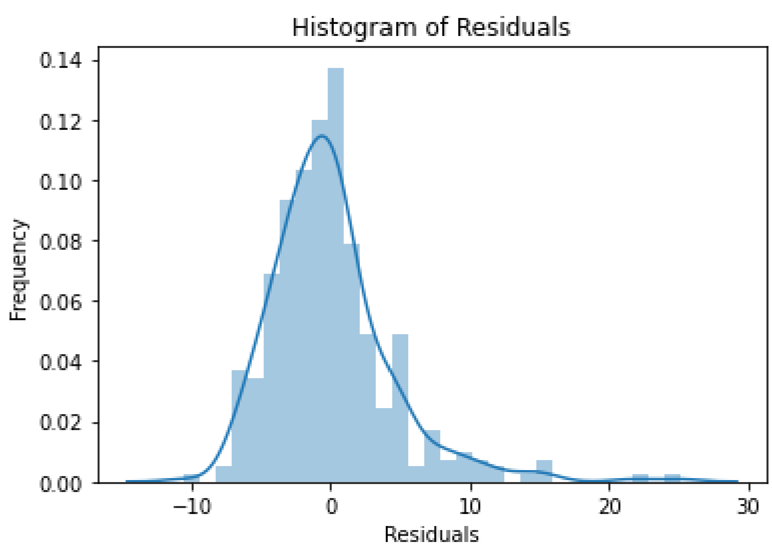
In [25]:
```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```



In [26]:
```
# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [27]:
```
# Checking Normality of errors
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [28]:
```
# Predicting Test data with the model
y_test_pred = lm.predict(X_test)
```

In [29]:
```
# Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_tes
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7121818377409185
Adjusted R^2: 0.6850685326005702
MAE: 3.8590055923707407
MSE: 30.053993307712424
RMSE: 5.482152251362985
```

Classifying movie reviews: a binary classification example ¶

Two-class classification, or binary classification, may be the most widely applied kind of machine-learning problem. In this example, you'll learn to classify movie reviews as positive or negative, based on the text content of the reviews.

The IMDB dataset You'll work with the IMDB dataset:

a set of 50,000 highly polarized reviews from the Internet Movie Database. They're split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.

In [1]:
```
from tensorflow.keras.datasets import imdb

# Load the data, keeping only 10,000 of the most frequently occuring words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_word
```

The argument *num_words=10000* means you'll only keep the top 10,000 most frequently occurring words in the training data. Rare words will be discarded. This allows you to work with vector data of manageable size.

In [2]:
```
train_data[0]
```

Out[2]:
```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
```

In [3]:
```
train_labels[0]
```

Out[3]: 1

Because you’re restricting yourself to the top 10,000 most frequent words, no word index will exceed 10,000:

In [4]:
```
max([max(sequence) for sequence in train_data])
```

Out[4]: 9999

In [5]:
```
# Let's quickly decode a review

# step 1: load the dictionary mappings from word to integer index
word_index = imdb.get_word_index()

# step 2: reverse word index to map integer indexes to their respective words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()

# Step 3: decode the review, mapping integer indices to words
'''Decodes the review. Note that the indices are offset by 3
        because 0, 1, and 2 are reserved indices for “padding,” “start of sequ
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_dat
```

In [6]:
```
decoded_review
```

Out[6]:
```
"? this film was just brilliant casting location scenery story direction ever
yone's really suited the part they played and you could just imagine being th
ere robert ? is an amazing actor and now the same being director ? father cam
e from the same scottish island as myself so i loved the fact there was a rea
l connection with this film the witty remarks throughout the film were great
it was just brilliant so much that i bought the film as soon as it was releas
ed for ? and would recommend it to everyone to watch and the fly fishing was
amazing really cried at the end it was so sad and you know what they say if y
ou cry at a film it must have been good and this definitely was also ? to the
two little boy's that played the ? of norman and paul they were just brillian
t children are often left out of the ? list i think because the stars that pl
ay them all grown up are such a big profile for the whole film but these chil
dren are amazing and should be praised for what they have done don't you thin
k the whole story was so lovely because it was true and was someone's life af
ter all that was shared with us all"
```

Preparing the data:

You can’t feed lists of integers into a neural network. You have to turn your lists into tensors. There are two ways to do that:

- Pad your lists so that they all have the same length, turn them into an integer tensor of shape (samples, word_indices), and then use as the first layer in your network a layer capable of handling such integer tensors (the Embedding layer, which we'll cover in detail later in the book).

- One-hot encode your lists to turn them into vectors of 0s and 1s. This would mean, for instance, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s. Then you could use as the first layer in your

In [7]:
```
#Encoding the integer sequences into a binary matrix
'''Explaination: I first created 2D matrix of shape(number of examples,10000)
        then I looped over each word of each example, if it exist put 1 in its
        if not just leave it as 0
        ITS JUST ONE HOT ENCODER'''
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an all zero m
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                        # Sets specific indices

    return results

# Vectorize training Data
X_train = vectorize_sequences(train_data)

# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

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

Out[8]:
```
(25000, 10000)
```

In [9]:
```
#vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test  = np.asarray(test_labels).astype('float32')
```

In [10]:
```
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Compiling the model

In [11]:
```
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import metrics

model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss = losses.binary_crossentropy,
              metrics = [metrics.binary_accuracy])
```

Validating your approach

In order to monitor during training the accuracy of the model on data it has never seen before, you'll create a validation set by setting apart 10,000 samples from the original training data.

In [12]:
```
# Input for Validation
X_val = X_train[:10000]
partial_X_train = X_train[10000:]

# Labels for validation
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [13]:
```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 39ms/step - loss: 0.5154 - binary
_accuracy: 0.7789 - val_loss: 0.3948 - val_binary_accuracy: 0.8479
Epoch 2/20
30/30 [==============================] - 0s 13ms/step - loss: 0.3140 - binary
_accuracy: 0.8953 - val_loss: 0.3320 - val_binary_accuracy: 0.8685
Epoch 3/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2371 - binary
_accuracy: 0.9179 - val_loss: 0.2880 - val_binary_accuracy: 0.8858
Epoch 4/20
30/30 [==============================] - 0s 14ms/step - loss: 0.1897 - binary
_accuracy: 0.9356 - val_loss: 0.2908 - val_binary_accuracy: 0.8832
Epoch 5/20
30/30 [==============================] - 0s 13ms/step - loss: 0.1577 - binary
_accuracy: 0.9475 - val_loss: 0.2888 - val_binary_accuracy: 0.8848
Epoch 6/20
30/30 [==============================] - 0s 12ms/step - loss: 0.1348 - binary
_accuracy: 0.9557 - val_loss: 0.3059 - val_binary_accuracy: 0.8793
Epoch 7/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1131 - binary
_accuracy: 0.9652 - val_loss: 0.3020 - val_binary_accuracy: 0.8847
Epoch 8/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0941 - binary
_accuracy: 0.9737 - val_loss: 0.3591 - val_binary_accuracy: 0.8741
Epoch 9/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0821 - binary
_accuracy: 0.9773 - val_loss: 0.3465 - val_binary_accuracy: 0.8738
Epoch 10/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0698 - binary
_accuracy: 0.9806 - val_loss: 0.3604 - val_binary_accuracy: 0.8749
Epoch 11/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0540 - binary
_accuracy: 0.9874 - val_loss: 0.4028 - val_binary_accuracy: 0.8681
Epoch 12/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0526 - binary
_accuracy: 0.9858 - val_loss: 0.4011 - val_binary_accuracy: 0.8745
Epoch 13/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0391 - binary
_accuracy: 0.9915 - val_loss: 0.4501 - val_binary_accuracy: 0.8752
Epoch 14/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0337 - binary
_accuracy: 0.9931 - val_loss: 0.4582 - val_binary_accuracy: 0.8692
Epoch 15/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0315 - binary
_accuracy: 0.9921 - val_loss: 0.4697 - val_binary_accuracy: 0.8731
Epoch 16/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0282 - binary
_accuracy: 0.9935 - val_loss: 0.4905 - val_binary_accuracy: 0.8741
Epoch 17/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0157 - binary
_accuracy: 0.9987 - val_loss: 0.5538 - val_binary_accuracy: 0.8597
Epoch 18/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0195 - binary
_accuracy: 0.9961 - val_loss: 0.5443 - val_binary_accuracy: 0.8725
Epoch 19/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0145 - binary
_accuracy: 0.9977 - val_loss: 0.5803 - val_binary_accuracy: 0.8626
```

```
Epoch 20/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0086 - binary
_accuracy: 0.9997 - val_loss: 0.7641 - val_binary_accuracy: 0.8530
```

Note that the call to *model.fit()* returns a *History* object. This object has a member *history*, which is a dictionary containing data about everything that happened during training. Let’s look at it:

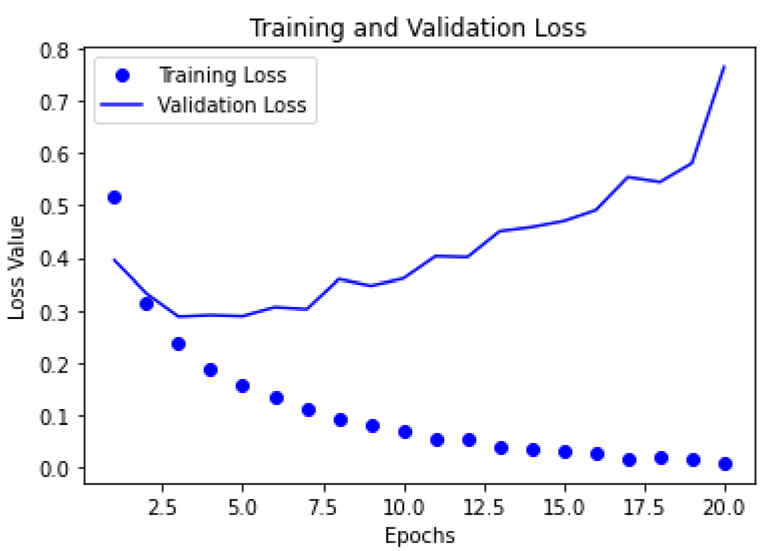
In [14]:
```
history_dict = history.history
history_dict.keys()
```

Out[14]:
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

Plotting the training and validation loss

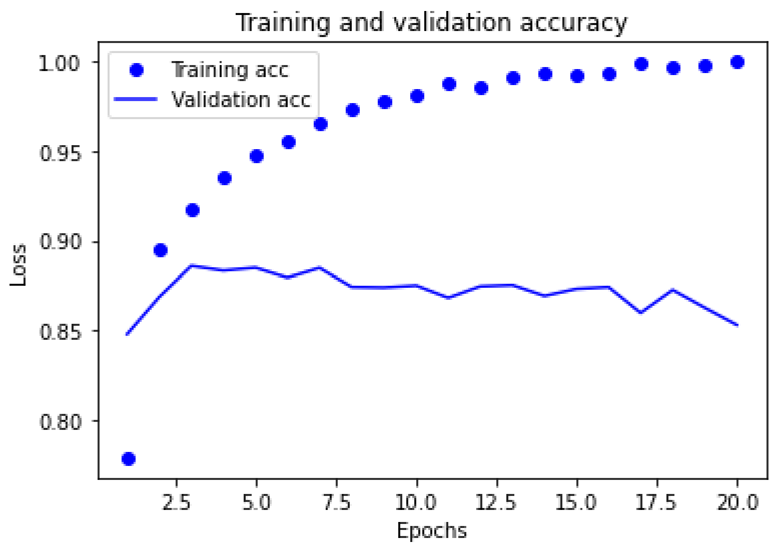
In [15]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [16]:
```
# Plotting Losses
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'bo', label="Training Loss")
plt.plot(epochs, val_loss_values, 'b', label="Validation Loss")

plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss Value')
plt.legend()

plt.show()
```



Plotting the training and validation accuracy

In [17]:
```
plt.clf() #Clears the figure
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Retraining a model from scratch

In [18]:
```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur
model.fit(X_train, y_train, epochs=4, batch_size=512)
```

```
Epoch 1/4
49/49 [==============================] - 1s 8ms/step - loss: 0.4666 - accurac
y: 0.8150
Epoch 2/4
49/49 [==============================] - 0s 7ms/step - loss: 0.2716 - accurac
y: 0.9035
Epoch 3/4
49/49 [==============================] - 0s 7ms/step - loss: 0.2108 - accurac
y: 0.9240
Epoch 4/4
49/49 [==============================] - 0s 7ms/step - loss: 0.1813 - accurac
y: 0.9338
```

Out[18]:
```
<keras.callbacks.History at 0x218812dbf10>
```

In [19]:
```
results = model.evaluate(X_test, y_test)
```

```
782/782 [==============================] - 1s 1ms/step - loss: 0.2846 - accur
acy: 0.8868
```

Using a trained network to generate predictions on new data

In [20]:
```
# Making Predictions for testing data
np.set_printoptions(suppress=True)
result = model.predict(X_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

In [21]:
```
result
```

Out[21]:
```
array([[0.20367871],
       [0.9999613 ],
       [0.91047627],
       ...,
       [0.10980374],
       [0.0675016 ],
       [0.67221063]], dtype=float32)
```

In [22]:
```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = np.asarray([round(x) for x in score])
```

In [23]:
```
y_pred
```

Out[23]:
```
array([0., 1., 1., ..., 0., 0., 1.])
```

In [24]:
```
from tensorflow.keras.metrics import mean_absolute_error
mae = mean_absolute_error = (y_pred, y_test)
```

In [25]:
```
# Error
mae
```

Out[25]:
```
(array([0., 1., 1., ..., 0., 0., 1.]),
 array([0., 1., 1., ..., 0., 0., 0.], dtype=float32))
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
fashion_train_df= pd.read_csv('fashion-mnist_train.csv')
```

In [4]:
```
fashion_test_df = pd.read_csv('fashion-mnist_test.csv')
```

In [5]:
```
fashion_train_df.head()
```

Out[5]:

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

5 rows × 785 columns

In [6]:
```
fashion_train_df.tail()
```

Out[6]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pi:
59995	9	0	0	0	0	0	0	0	0	0	...	0	
59996	1	0	0	0	0	0	0	0	0	0	...	73	
59997	8	0	0	0	0	0	0	0	0	0	...	160	
59998	8	0	0	0	0	0	0	0	0	0	...	0	
59999	7	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [7]:
```
fashion_train_df.shape
```

Out[7]: (60000, 785)

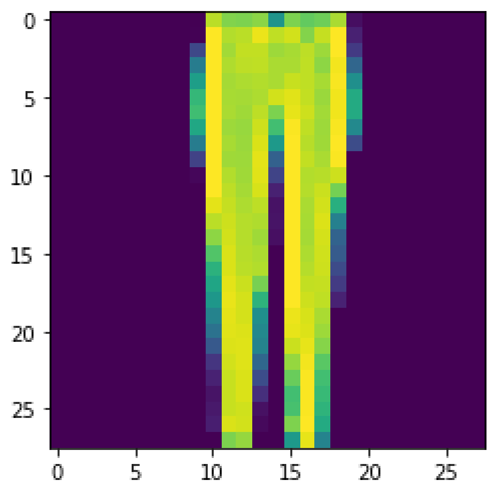
In [8]:
```
fashion_test_df.shape
```
Out[8]: (10000, 785)

In [9]:
```
training = np.array(fashion_train_df,dtype='float32')
testing = np.array(fashion_test_df,dtype='float32')
```

In [10]:
```
training.shape
```
Out[10]: (60000, 785)

In [11]:
```
import random
```

In [12]:
```
i = random.randint(0,60001)
plt.imshow(training[i,1:].reshape(28,28))
label = training[i,1]
label
```
Out[12]: 0.0



i = random.randint(0,60001) plt.imshow(training[i,1:].reshape(28,28)) label = training[i,1] label

In [13]:
```
W_grid = 7
L_grid = 7

fig,axes = plt.subplots(L_grid,W_grid,figsize =(17,17))

axes = axes.ravel()
n_training = len(training)



for i in np.arange(0,W_grid*L_grid):
        index = np.random.randint(0,n_training)
        axes[i].imshow(training[index,1:].reshape((28,28)))
        axes[i].set_title(training[index,0],fontsize = 8)
        axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```

In [14]:
```
X_train = training[:,1:]/255
y_train = training[:,0]
X_test = testing[:,1:]/255
y_test = testing[:,0]
```

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train,t
```

In [16]:
```
X_train = X_train.reshape(X_train.shape[0],*(28,28,1))
X_test = X_test.reshape(X_test.shape[0],*(28,28,1))
X_validate = X_validate.reshape(X_validate.shape[0],*(28,28,1))
```

In [17]:
```
X_train.shape
```

Out[17]:
```
(48000, 28, 28, 1)
```

In [18]:
```
X_test.shape
```

Out[18]:
```
(10000, 28, 28, 1)
```

In [19]:
```
X_validate.shape
```

Out[19]:
```
(12000, 28, 28, 1)
```

In [20]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
```

In [21]:
```
cnn_model = Sequential()
cnn_model.add(Conv2D(32,3,3,input_shape = (28,28,1),activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size= (2,2)))
cnn_model.add(Flatten())
cnn_model.add(Dense(32,activation = 'relu'))
cnn_model.add(Dense(10,activation = 'sigmoid'))
cnn_model.compile(loss ='sparse_categorical_crossentropy',optimizer = Adam(lea
```

In [22]:
```
epochs = 200
```

In [23]:
```
cnn_model.fit(X_train,y_train,batch_size =512,epochs = epochs,verbose = 1,vali
```

```
Epoch 1/200
94/94 [==============================] - 2s 12ms/step - loss: 1.3900 - acc
uracy: 0.5820 - val_loss: 0.7855 - val_accuracy: 0.7256
Epoch 2/200
94/94 [==============================] - 1s 9ms/step - loss: 0.6848 - accu
racy: 0.7535 - val_loss: 0.6168 - val_accuracy: 0.7744
Epoch 3/200
94/94 [==============================] - 1s 9ms/step - loss: 0.5819 - accu
racy: 0.7889 - val_loss: 0.5550 - val_accuracy: 0.8021
Epoch 4/200
94/94 [==============================] - 1s 9ms/step - loss: 0.5311 - accu
racy: 0.8065 - val_loss: 0.5156 - val_accuracy: 0.8148
Epoch 5/200
94/94 [==============================] - 1s 9ms/step - loss: 0.5023 - accu
racy: 0.8161 - val_loss: 0.4916 - val_accuracy: 0.8238
Epoch 6/200
94/94 [==============================] - 1s 8ms/step - loss: 0.4793 - accu
racy: 0.8253 - val_loss: 0.4733 - val_accuracy: 0.8345
Epoch 7/200
```

In [24]:
```
evaluation = cnn_model.evaluate(X_test,y_test)
print('Test Accuracy : {:.3f}'.format(evaluation[1]))
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.3384 - accur
acy: 0.8835
Test Accuracy : 0.883
```

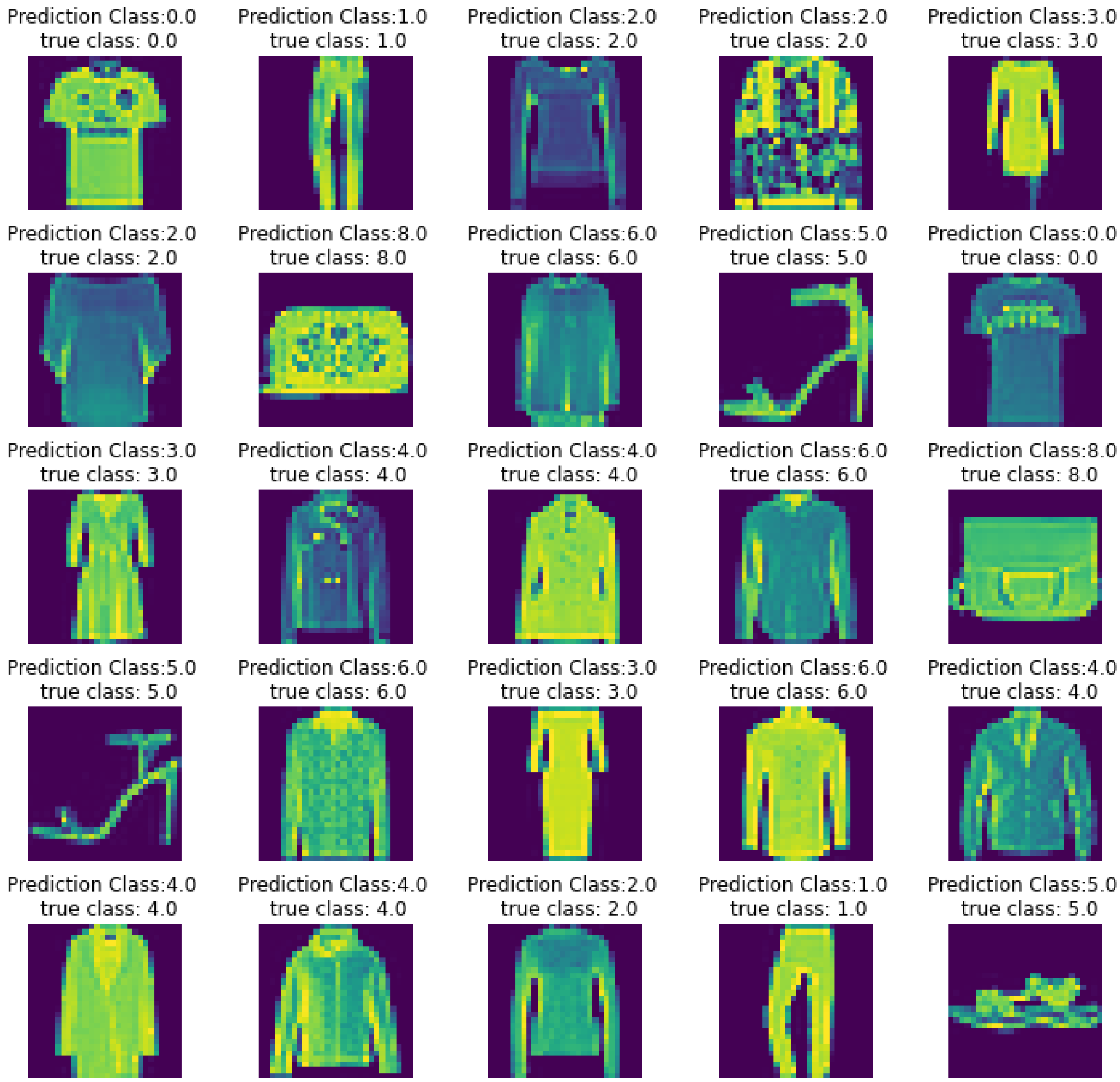
In [25]:
```
predicted_classes = np.argmax(cnn_model.predict(X_test),axis=-1)
```

```
313/313 [==============================] - 0s 947us/step
```

In [26]:
```
predicted_classes
```

Out[26]:
```
array([0, 1, 2, ..., 8, 8, 1], dtype=int64)
```

In [27]:
```
L = 5
W = 5

fig,axes = plt.subplots(L,W,figsize = (12,12))
axes = axes.ravel()
for i in np.arange(0,L*W):
    axes[i].imshow(X_test[i].reshape(28,28))
    axes[i].set_title('Prediction Class:{1} \n true class: {1}'.format(predict
    axes[i].axis('off')
plt.subplots_adjust(wspace = 0.5)
```



In [28]:
```
from sklearn.metrics import classification_report

classes = 10
targets = ["Class {}".format(i) for i in range(classes)]
print(classification_report(y_test, predicted_classes, target_names = targets))
```

```
              precision    recall  f1-score   support

     Class 0       0.86      0.81      0.83      1000
     Class 1       0.98      0.97      0.98      1000
     Class 2       0.79      0.86      0.82      1000
     Class 3       0.88      0.89      0.89      1000
     Class 4       0.84      0.79      0.81      1000
     Class 5       0.98      0.93      0.95      1000
     Class 6       0.69      0.71      0.70      1000
     Class 7       0.92      0.95      0.93      1000
     Class 8       0.97      0.97      0.97      1000
     Class 9       0.94      0.96      0.95      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

1. Problem statement

- We are given Google stock price from 01/2012 to 12/2017.
- The task is to predict the trend of the stock price for 01-06 2018.

2. 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
```

3. Data processing

3.0 import the data

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

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

Out[3]:

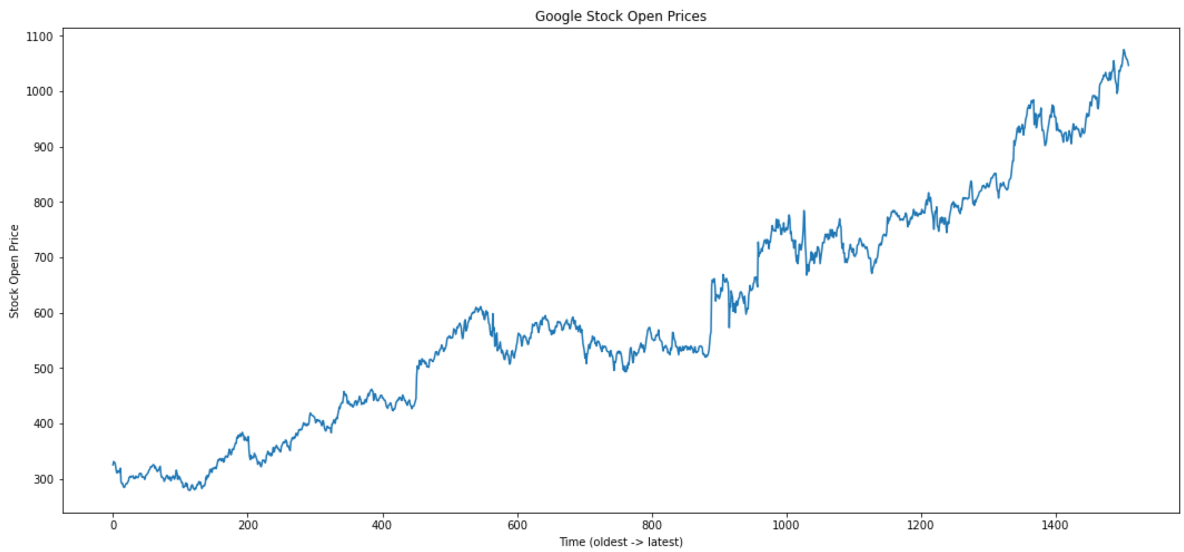
	Date	Open	High	Low	Close	Volume
0	01/03/2012	325.25	332.83	324.97	663.59	7,380,500
1	01/04/2012	331.27	333.87	329.08	666.45	5,749,400
2	01/05/2012	329.83	330.75	326.89	657.21	6,590,300
3	01/06/2012	328.34	328.77	323.68	648.24	5,405,900
4	01/09/2012	322.04	322.29	309.46	620.76	11,688,800

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()
```



3.1 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)
```

3.2 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,)
```

3.3 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)
```

4. Create & Fit Model

4.1 Create model

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

In [15]:
```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

4.2 Model fit

In [16]:
```
regressor.fit(x = X_train, y = y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
46/46 [==============================] - 23s 142ms/step - loss: 0.0276
Epoch 2/100
46/46 [==============================] - 7s 146ms/step - loss: 0.0039
Epoch 3/100
46/46 [==============================] - 6s 131ms/step - loss: 0.0032
Epoch 4/100
46/46 [==============================] - 6s 137ms/step - loss: 0.0032
Epoch 5/100
46/46 [==============================] - 7s 143ms/step - loss: 0.0029
Epoch 6/100
46/46 [==============================] - 7s 148ms/step - loss: 0.0031
Epoch 7/100
46/46 [==============================] - 7s 148ms/step - loss: 0.0028
Epoch 8/100
46/46 [==============================] - 7s 146ms/step - loss: 0.0028
Epoch 9/100
46/46 [==============================] - 7s 142ms/step - loss: 0.0032
Epoch 10/100
```

4.3 Model evaluation

4.3.1 Read and convert

In [17]:
```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
```

In [18]:
```
dataset_test.head()
```

Out[18]:

	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 [19]:
```
#keras only takes numpy array
real_stock_price = dataset_test.iloc[:, 1: 2].values
real_stock_price.shape
```

Out[19]: (125, 1)

4.3.2 Concat and convert

In [20]:
```
#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
```

4.3.3 Reshape and scale

In [21]:
```
#reshape data to only have 1 col
inputs = inputs.reshape(-1, 1)

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

In [22]:
```
len(inputs)
```

Out[22]: 185

4.3.4 Create test data strucutre

In [23]:
```
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 [24]:
```
X_test.shape
```

Out[24]:
```
(125, 60, 1)
```

4.3.5 Model prediction

In [25]:
```
predicted_stock_price = regressor.predict(X_test)
```

```
4/4 [==============================] - 1s 24ms/step
```

In [26]:
```
#inverse the scaled value
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

4.3.6 Result visualization

In [27]:
```
##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()
```