Index

S. No	Title	Date	Sign.
1.	Linear regression: Implement linear regression on a dataset and evaluate the model's performance.		
2.	Logistic regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance.		
3.	k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance.		
4.	Decision Trees: Implement decision trees on a dataset and evaluate the model's performance		
5.	Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.		
6.	Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance		
7.	Naive Bayes: Implement Naive Bayes algorithm on a dataset and evaluate the model's performance.		
8.	Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance.		
9.	Convolutional Neural Networks (CNN): Implement CNN on an image classification dataset and evaluate the model's performance		
10.	Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance.		
11.	Long Short-Term Memory Networks (LSTM): Implement LSTM on a time-series dataset and evaluate the model's performance.		
12.	Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance.		
13.	Generative Adversarial Networks (GANs): Implement GANs on an image dataset and evaluate the model's performance.		
14.	Transfer Learning: Implement transfer learning on an image dataset and evaluate the model's performance		
15.	Implement reinforcement learning in a game environment and evaluate performance .		

Practical -1

AIM : Linear regression: Implement linear regression on a dataset and evaluate the model's performance.

Code :

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

Training the Simple Linear Regression model on the Training set

In [4]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[4]:
```
▾ LinearRegression
LinearRegression()
```

Predicting the Test set results

In [5]:
```
y_pred = regressor.predict(X_test)
```

```
In [6]: plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Visualising the Test set results

```
In [7]: plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Practical - 2

AIM : Logistic regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance.

Code :

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [3]:
```
X
```

Out[3]:
```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],
       [    32, 150000],
       [    25,  33000],
       [    35,  65000],
       [    26,  80000],
       [    26,  52000],
       [    20,  86000],
       [    32,  18000],
       [    18,  82000],
```

Splitting the dataset into the Training set and Test set

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [6]:
```
print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
 [    47  49000]
 [    59  29000]
 [    49  65000]
```

Feature Scaling

In [10]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [11]:
```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
```

Training the Logistic Regression model on the Training set

In [13]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[13]:
```
LogisticRegression
LogisticRegression(random_state=0)
```

Predicting a new result

In [14]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

In [15]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
```

Making the Confusion Matrix

In [16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 8 24]]
```

Out[16]:
```
0.89
```

Practical - 3

AIM : k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance.

Code :

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [6]:
```
print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
```

Feature Scaling

In [10]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [11]:
```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
```

Training the K-NN model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
```

Making the Confusion Matrix

In [16]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
```

Out[16]:

```
0.93
```

Practical - 4

AIM : Decision Trees: Implement decision trees on a dataset and evaluate the model's performance.

Code :

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

n [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

n [6]:
```
print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
```

Feature Scaling

In [10]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [11]:
```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
```

Training the Decision Tree Classification model on the Training set

In [13]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[13]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Predicting a new result

In [14]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

In [15]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
```

Making the Confusion Matrix

In [16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[62  6]
 [ 3 29]]
```

Out[16]:
```
0.91
```

Practical - 5

Aim : Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.

Code :

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
```

Training the Random Forest Classification model on the Training set

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
```

Making the Confusion Matrix

In [14]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[63  5]
 [ 4 28]]
```

Out[14]: 0.91

Practical - 6

AIM: Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.

Code :

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[[    19  19000]
 [    35  20000]
 [    26  43000]
 [    27  57000]
 [    19  76000]
 [    27  58000]
 [    27  84000]
 [    32 150000]
 [    25  33000]
```

Splitting the dataset into the Training set and Test set

```
In [4]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
In [5]: print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
```

Feature Scaling

```
In [9]: from sklearn.preprocessing import StandardScaler
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
```

```
In [10]: print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
```

Training the Kernel SVM model on the Training set

In [12]:
```
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[12]:
```
SVC
SVC(random_state=0)
```

Predicting a new result

In [13]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

In [14]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
```

Making the Confusion Matrix

In [15]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
```

Out[15]: 0.93

Practical - 7

Aim : Naive Bayes: Implement Naive Bayes algorithm on a dataset and evaluate the model's performance .

Code :

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [3]:
```
print(X)
```

```
[[    19  19000]
 [    35  20000]
 [    26  43000]
 [    27  57000]
 [    19  76000]
 [    27  58000]
 [    27  84000]
```

Splitting the dataset into the Training set and Test set

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [6]:
```
print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
```

Feature Scaling

In [10]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [11]:
```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
```

Training the Naive Bayes model on the Training set

In [13]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[13]:
```
▾ GaussianNB
GaussianNB()
```

Predicting a new result

In [14]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

In [15]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
```

Making the Confusion Matrix

In [16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 7 25]]
```

Out[16]: 0.9

Practical - 8

Aim : Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance.

Code :

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [2]:
```
data= pd.read_csv('Social_Network_Ads.csv')
```

In [3]:
```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

In [4]:
```
X.fillna(X.mean(), inplace=True)
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [6]:
```
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb_classifier.fit(X_train, y_train)
```

Out[6]:
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

In [7]:
```
y_pred = gb_classifier.predict(X_test)
```

In [8]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.8625
```

In [9]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.90        52
           1       0.81      0.79      0.80        28

    accuracy                           0.86        80
   macro avg       0.85      0.84      0.85        80
weighted avg       0.86      0.86      0.86        80
```

Practical - 9

Aim : Convolutional Neural Networks (CNN): Implement CNN on an image classification dataset and evaluate the model's performance.

Code :

In [1]:

```
import os
import numpy as np
from keras.preprocessing import image
import matplotlib.pyplot as plt
%matplotlib inline

def load_images_from_path(path, label):
    images = []
    labels = []

    for file in os.listdir(path):
        img = image.load_img(os.path.join(path, file), target_size=(224, 224, 3))
        images.append(image.img_to_array(img))
        labels.append((label))

    return images, labels

def show_images(images):
    fig, axes = plt.subplots(1, 8, figsize=(20, 20), subplot_kw={'xticks': [], 'yticks': []})

    for i, ax in enumerate(axes.flat):
        ax.imshow(images[i] / 255)

x_train = []
y_train = []
x_test = []
y_test = []
```

]:

```
images, labels = load_images_from_path('Data/train/arctic_fox', 0)
show_images(images)

x_train += images
y_train += labels
```

Load polar-bear training images.

]:

```
images, labels = load_images_from_path('Data/train/polar_bear', 1)
show_images(images)

x_train += images
y_train += labels
```

```python
from tensorflow.keras.utils import to_categorical

x_train = np.array(x_train) / 255
x_test = np.array(x_test) / 255

y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)
```

In [9]:

```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 128)     36992
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 128)       147584
```

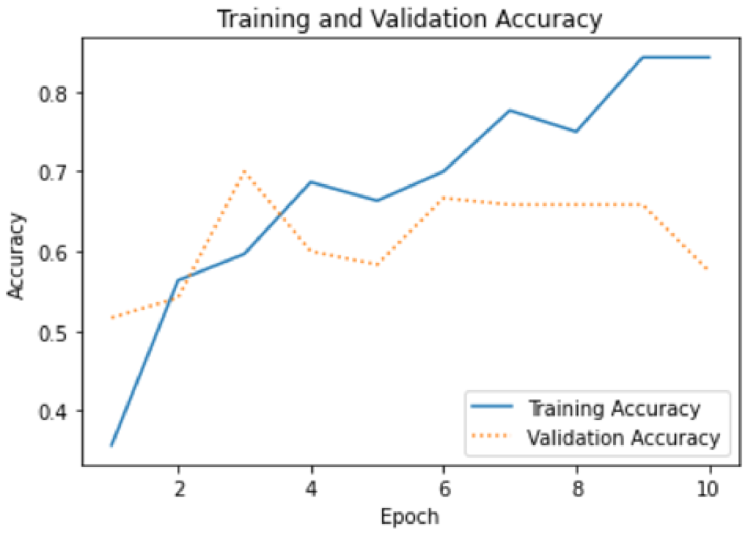
```python
hist = model.fit(x_train, y_train_encoded, validation_data=(x_test, y_test_encoded), batch_size=10, epochs=10)
```

```
Epoch 1/10
30/30 [==============================] - 18s 314ms/step - loss: 1.6569 - accuracy: 0.3303 - val_loss: 1.0713 - val_accuracy: 0.5167
Epoch 2/10
30/30 [==============================] - 9s 304ms/step - loss: 0.9401 - accuracy: 0.5681 - val_loss: 0.9088 - val_accuracy: 0.5417
Epoch 3/10
30/30 [==============================] - 9s 294ms/step - loss: 0.8122 - accuracy: 0.6006 - val_loss: 0.8025 - val_accuracy: 0.7000
Epoch 4/10
30/30 [==============================] - 9s 312ms/step - loss: 0.7639 - accuracy: 0.6580 - val_loss: 0.8886 - val_accuracy: 0.6000
Epoch 5/10
30/30 [==============================] - 9s 290ms/step - loss: 0.6776 - accuracy: 0.6477 - val_loss: 0.8573 - val_accuracy: 0.5833
Epoch 6/10
30/30 [==============================] - 9s 286ms/step - loss: 0.5531 - accuracy: 0.7462 - val_loss: 0.7939 - val_accuracy: 0.6667
Epoch 7/10
30/30 [==============================] - 9s 284ms/step - loss: 0.5155 - accuracy: 0.7691 - val_loss: 0.9490 - val_accuracy: 0.6583
Epoch 8/10
```

In [11]:

```
acc = hist.history['accuracy']
val_acc = hist.history['val_accuracy']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, '-', label='Training Accuracy')
plt.plot(epochs, val_acc, ':', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.plot()
```

Out[11]: []



Practical - 10

Aim : Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance.

Code :

In [25]:
```
DATA_PATH = 'data/toxic-comments/'
os.makedirs(DATA_PATH, exist_ok=True)
```

In [28]:
```
ZIP_DATA_PATH = f'{DATA_PATH}zip_files/'
```

In [25]:
```
!mkdir -p {ZIP_DATA_PATH} && kaggle competitions download -c jigsaw-toxic-comment-classification-challenge -p {ZIP_DATA_PATH
```

```
downloading https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/download/sample_submission.csv.zip

sample_submission.csv.zip 100% |####################| Time: 0:00:00   3.8 MiB/s

downloading https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/download/test.csv.zip

test.csv.zip 100% |#################################| Time: 0:00:00  31.6 MiB/s

downloading https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/download/train.csv.zip

train.csv.zip 100% |################################| Time: 0:00:00  39.0 MiB/s

downloading https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/download/test_labels.csv.zip

test_labels.csv.zip 100% |##########################| Time: 0:00:00   5.0 MiB/s
```

In [30]:
```
RAW_DATA_PATH = f'{DATA_PATH}raw_data_files/'
os.makedirs(RAW_DATA_PATH, exist_ok=True)
```

In [13]:
```
import glob
import zipfile
def unzip(path, targ_path):
    """ Function to unzip files in data path"""
    for file in glob.glob(path):
        zip_ref = zipfile.ZipFile(file, 'r')
        zip_ref.extractall(targ_path)
        zip_ref.close()
        print('%s unzipped' %file.split('/')[-1])
```

In [29]:
```
unzip(f'{ZIP_DATA_PATH}*.zip', f'{RAW_DATA_PATH}')
```

```
train.csv.zip unzipped
test_labels.csv.zip unzipped
sample_submission.csv.zip unzipped
test.csv.zip unzipped
```

In [5]:
```python
import pandas as pd
import numpy as np
```

In [31]:
```python
def get_file_names(path, file_format=''):
    """Function to get filenames in data path"""
    file_list=[]
    path = path+'*'+file_format
    for file in glob.glob(path):
        file_list.append(file.split('/')[-1].split('.')[0])
    return file_list
```

In [32]:
```python
file_names = get_file_names(f'{RAW_DATA_PATH}', file_format='.csv')
file_names
```

Out[32]: ['test_labels', 'test', 'train', 'sample_submission']

In [33]:
```python
tables = [pd.read_csv(f'{RAW_DATA_PATH}{fname}.csv', low_memory=False)
          for fname in file_names]
```

In [40]:
```python
import re


def clean(comment):
    # remove \n
    # torchtext cannot read the .csv files correctly if there are newline characters, so replace with " "
    comment = re.sub("\\n"," ",comment)

    # remove leaky elements like ip,user
    comment = re.sub("\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}"," ",comment)

    # removing usernames
    comment = re.sub("\[\[.*\]","",comment)

    comment = re.sub(r"[\*\"“”\n\\…\+\-\/\=\(\)‘•:\[\]\|’\!;]", " ", str(comment))

    comment = re.sub(r"[ ]+", " ", comment)

    comment = re.sub(r"\!+", "!", comment)

    comment = re.sub(r"\,+", ",", comment)

    comment = re.sub(r"\?+", "?", comment)

    return(comment)
```

In [41]:
```python
raw_train_df['comment_text'] = raw_train_df.comment_text.apply(lambda x: clean(x))
print('train cleaned ...')

raw_test_df['comment_text'] = raw_test_df.comment_text.apply(lambda x: clean(x))
print('test cleaned ...')
```

n [41]:
```python
raw_train_df['comment_text'] = raw_train_df.comment_text.apply(lambda x: clean(x))
print('train cleaned ...')

raw_test_df['comment_text'] = raw_test_df.comment_text.apply(lambda x: clean(x))
print('test cleaned ...')
```


```
train cleaned ...
test cleaned ...
```

n [43]:
```python
raw_train_df.shape
```

ut[43]: (159571, 8)

convenient. Here's how we would read data from a csv file.

In [11]:
```
trn_data_fields = [("id", None),
                   ("comment_text", TEXT), ('toxic', LABEL),
                  ('severe_toxic', LABEL), ('obscene', LABEL),
                  ('threat', LABEL), ('insult', LABEL),
                  ('identity_hate', LABEL),]

trn, vld = data.TabularDataset.splits(path=f'{PROCESSED_DATA_PATH}',
                                      train='train_ds.csv', validation='valid_ds.csv',
                                      format='csv', skip_header=True, fields=trn_data_fields)

full_trn = data.TabularDataset(path=f'{PROCESSED_DATA_PATH}full_train_ds.csv',
                                      format='csv', skip_header=True, fields=trn_data_fields)
```

In [12]:
```
test_data_fields = [("id", None),
                    ("comment_text", TEXT)]

tst = data.TabularDataset(path=f'{PROCESSED_DATA_PATH}test_ds.csv',
                                      format='csv', skip_header=True, fields=test_data_fields)
```

2.7. Load pretrained vectors & Build vocabulary

In [13]:
```
VECTOR_PATH = '.vector_cache'
!ls {VECTOR_PATH}
```

```
glove.6B.100d.txt     glove.6B.200d.txt.pt  glove.6B.50d.txt.pt
glove.6B.100d.txt.pt  glove.6B.300d.txt     wiki.en.vec
glove.6B.200d.txt     glove.6B.50d.txt
```

In [15]:
```
TEXT.build_vocab(full_trn, vectors=pretrained_vectors, max_size = MAX_CHARS)
```

Let's take a look at what the vocab looks like.

In [16]:
```
TEXT.vocab.freqs.most_common(10)
```

Out[16]:
```
[('.', 514412),
 ('the', 495846),
 (',', 470871),
 ('"', 379291),
 ('to', 297111),
 ('i', 239115),
 ('of', 224181),
 ('and', 222987),
 ('you', 219888),
 ('a', 214419)]
```

In [20]:

```python
class RNNModel(nn.Module):
    """
    Neural Network Module with an embedding layer, a recurent module and an output linear layer

    Arguments:
        rnn_type(str) -- type of rnn module to use options are ['LSTM', 'GRU', 'RNN_TANH', 'RNN_RELU']
        input_size(int) -- size of the dictionary of embeddings
        embz_size(int) -- the size of each embedding vector
        hidden_size(int) -- the number of features in the hidden state
        batch_size(int) -- the size of training batches
        output_size(int) -- the number of output classes to be predicted
        num_layers(int, optional) -- Number of recurrent layers. Default=1
        dropout(float, optional) -- dropout probabilty. Default=0
        bidirectional(bool, optional) -- If True, becomes a bidirectional RNN. Default=False
        tie_weights(bool, optional) -- if True, ties the weights of the embedding and output layer. Default=False

    Inputs: input
        input of shape (seq_length, batch_size) -- tensor containing the features of the input sequence

    Returns: output
        output of shape (batch_size, output_size) -- tensor containing the sigmoid activation on the
                                                     output features h_t from the last layer of the rnn,
                                                     for the last time-step t.
    """


    def __init__(self, rnn_type, input_size, embz_size, hidden_size, batch_size, output_size,
                 num_layers=1, dropout=0.5, bidirectional=True, tie_weights=False):
        super().__init__()

        if bidirectional: self.num_directions = 2
        else: self.num_directions = 1

        self.hidden_size, self.output_size, self.embz_size = hidden_size, output_size, embz_size
        self.bidirectional, self.rnn_type, self.num_layers = bidirectional, rnn_type, num_layers
        self.drop = nn.Dropout(dropout)
        self.embedding_layer = nn.Embedding(input_size, embz_size)
        self.output_layer = nn.Linear(hidden_size*self.num_directions, output_size)
        self.init_hidden(batch_size)

        if rnn_type in ['LSTM', 'GRU']:
            self.rnn = getattr(nn, rnn_type)(embz_size, hidden_size, num_layers=num_layers,
                              dropout=dropout, bidirectional=bidirectional)
        else:
            try:
                nonlinearity = {'RNN_TANH':'tanh', 'RNN_RELU':'relu'}[rnn_type]
            except KeyError:
                raise ValueError("""An invalid option for '--rnn_type' was supplied,
                                 options are ['LSTM', 'GRU', 'RNN_TANH', 'RNN_RELU']""")
            self.rnn = nn.RNN(embz_size, hidden_size, num_layers=num_layers,
                              dropout=dropout, bidirectional=bidirectional, nonlinearity=nonlinearity)

        if tie_weights:
            if hidden_size != embz_size:
                raise ValueError("When using the tied flag, hidden size must be equal to embeddign size")
            elif bidirectional:
                raise ValueError("When using the tied flag, set bidirectional=False")
            self.output_layer.weight = self.embedding_layer.weight

    def init_emb_weights(self, vector_weight_matrix):
        self.embedding_layer.weight.data.copy_(vector_weight_matrix)

    def init_identity_weights(self):
        if self.rnn_type == 'RNN_RELU':
            self.rnn.weight_ih_l0.data.copy_(torch.eye(self.hidden_size, self.embz_size))
            self.rnn.weight_hh_l0.data.copy_(torch.eye(self.hidden_size, self.hidden_size))

            if self.bidirectional:
                self.rnn.weight_ih_l0_reverse.data.copy_(torch.eye(self.hidden_size, self.embz_size))
                self.rnn.weight_hh_l0_reverse.data.copy_(torch.eye(self.hidden_size, self.hidden_size))
        else:
            pass
```

```python
    def init_hidden(self, batch_size):
        if self.rnn_type == 'LSTM':
            self.hidden = (V(torch.zeros(self.num_layers*self.num_directions, batch_size, self.hidden_size)),
                           V(torch.zeros(self.num_layers*self.num_directions, batch_size, self.hidden_size)))
        else:
            self.hidden = V(torch.zeros(self.num_layers*self.num_directions, batch_size, self.hidden_size))

    def forward(self, seq):
        batch_size = seq[0].size(0)
        if self.hidden[0].size(1) != batch_size:
            self.init_hidden(batch_size)
        input_tensor = self.drop(self.embedding_layer(seq))
        output, hidden = self.rnn(input_tensor, self.hidden)
        self.hidden = repackage_var(hidden)
        output = self.drop(self.output_layer(output))
        return F.sigmoid(output[-1, :, :])
```

]:

```python
vector_weight_matrix = TEXT.vocab.vectors
input_size = vector_weight_matrix.size(0)
hidden_size = vector_weight_matrix.size(1)
output_size = 6
embz_size = vector_weight_matrix.size(1)
batch_size = batch_size
rnn_type = 'GRU'
model = RNNModel(rnn_type, input_size, embz_size, hidden_size, batch_size, output_size); model
```

]:

```
RNNModel(
  (drop): Dropout(p=0.5)
  (embedding_layer): Embedding(20002, 50)
  (output_layer): Linear(in_features=100, out_features=6)
  (rnn): GRU(50, 50, dropout=0.5, bidirectional=True)
)
```

In [104…

```python
def to_label(x, threshold=0.5):
    if x > threshold:
        return 1
    else:
        return 0
```

In [107…

```python
for i in range(len(label_column_list)):
    test_ds[label_column_list[i]] = test_ds[label_column_list[i]].apply(to_label, threshold=0.45)
```

In [108…

```python
test_ds.head()
```

Out[108…

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you wi...	1	0	1	0	1	0
1	0000247867823ef7	== From RfC == The title is fine as it is, IMO.	0	0	0	0	0	0
2	00013b17ad220c46	" == Sources == * Zawe Ashton on Lapland — / "	0	0	0	0	0	0
3	00017563c3f7919a	:If you have a look back at the source, the in...	0	0	0	0	0	0
4	00017695ad8997eb	I do not anonymously edit articles at all.	0	0	0	0	0	0

In [109…

```python
for i in label_column_list:
    print(i.upper())
    display(test_ds[getattr(test_ds_, i) == 1].sample().iloc[0]['comment_text'])
```

```
TOXIC
```

Practical - 11

Aim : Long Short-Term Memory Networks (LSTM): Implement LSTM on a time-series dataset and evaluate the model's performance.

Code :

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.float_format', lambda x: '%.4f' % x)
import seaborn as sns
sns.set_context("paper", font_scale=1.3)
sns.set_style('white')
import warnings
warnings.filterwarnings('ignore')
from time import time
import matplotlib.ticker as tkr
from scipy import stats
from statsmodels.tsa.stattools import adfuller
from sklearn import preprocessing
from statsmodels.tsa.stattools import pacf
%matplotlib inline

import math
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.callbacks import EarlyStopping
```

In [2]:
```
df=pd.read_csv('household_power_consumption.txt', delimiter=';')
print('Number of rows and columns:', df.shape)
df.head(5)
```

```
Number of rows and columns: (2075259, 9)
```

Out[2]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_me
0	16/12/2006	17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	
1	16/12/2006	17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	
2	16/12/2006	17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	
3	16/12/2006	17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	
4	16/12/2006	17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	

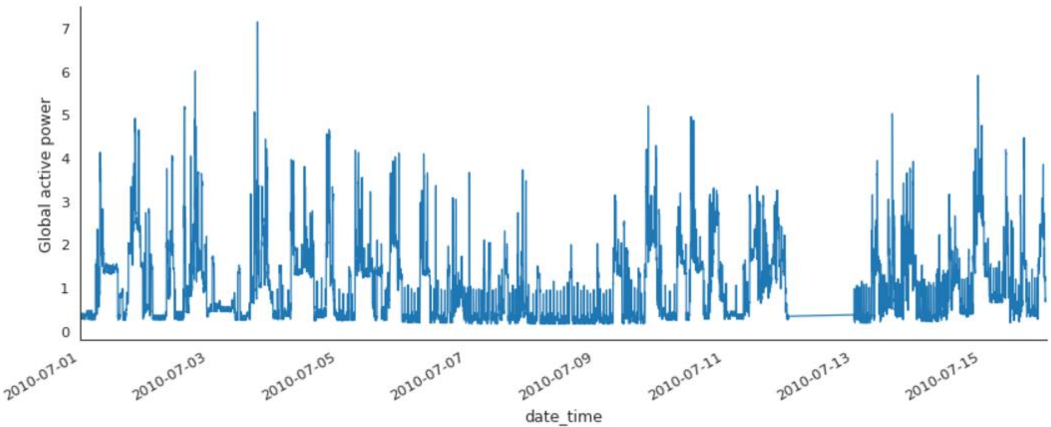
In [3]:

```
df['date_time'] = pd.to_datetime(df['Date'] + ' ' + df['Time'])
df['Global_active_power'] = pd.to_numeric(df['Global_active_power'], errors='coerce')
df = df.dropna(subset=['Global_active_power'])
df['date_time']=pd.to_datetime(df['date_time'])
df['year'] = df['date_time'].apply(lambda x: x.year)
df['quarter'] = df['date_time'].apply(lambda x: x.quarter)
df['month'] = df['date_time'].apply(lambda x: x.month)
df['day'] = df['date_time'].apply(lambda x: x.day)
df=df.loc[:,['date_time','Global_active_power', 'year','quarter','month','day']]
df.sort_values('date_time', inplace=True, ascending=True)
df = df.reset_index(drop=True)
df["weekday"]=df.apply(lambda row: row["date_time"].weekday(),axis=1)
df["weekday"] = (df["weekday"] < 5).astype(int)
print(df.shape)
print(df.date_time.min())
print(df.date_time.max())
df.tail(5)
```

```
(2049280, 7)
2006-12-16 17:24:00
2010-12-11 23:59:00
```

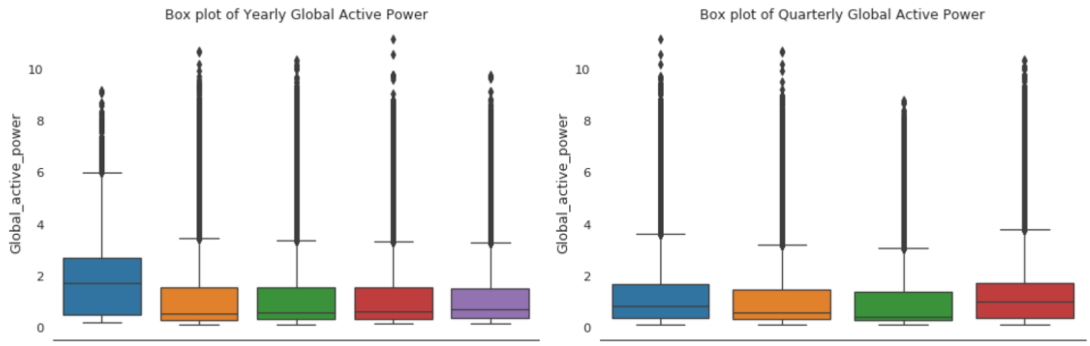
In [8]:

```
df2=df1[(df1.index>='2010-07-01') & (df1.index<'2010-7-16')]
df2.plot(figsize=(12,5));
plt.ylabel('Global active power')
plt.legend().set_visible(False)
plt.tight_layout()
sns.despine(top=True)
plt.show();
```



In [9]:

```
plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.subplots_adjust(wspace=0.2)
sns.boxplot(x="year", y="Global_active_power", data=df)
plt.xlabel('year')
plt.title('Box plot of Yearly Global Active Power')
sns.despine(left=True)
plt.tight_layout()

plt.subplot(1,2,2)
sns.boxplot(x="quarter", y="Global_active_power", data=df)
plt.xlabel('quarter')
plt.title('Box plot of Quarterly Global Active Power')
sns.despine(left=True)
plt.tight_layout();
```



In [20]:

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)
```

In [21]:

```
# reshape into X=t and Y=t+1
look_back = 30
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test = create_dataset(test, look_back)
```

In [22]:

```
X_train.shape
```

Out[22]:

```
(1639393, 30)
```

In [23]:

```
Y_train.shape
```

Out[23]:

```
(1639393,)
```

In [24]:

```
# reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

In [26]:

```python
model = Sequential()
model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(X_train, Y_train, epochs=20, batch_size=70, validation_data=(X_test, Y_test),
                    callbacks=[EarlyStopping(monitor='val_loss', patience=10)], verbose=1, shuffle=False)

# Training Phase
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_wi
th (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (fro
m tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorf
low.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 1639393 samples, validate on 409825 samples
Epoch 1/20
1639393/1639393 [==============================] - 49s 30us/step - loss: 7.5102e-04 - val_loss: 4.3052e-04
Epoch 2/20
1639393/1639393 [==============================] - 48s 29us/step - loss: 6.6376e-04 - val_loss: 4.4660e-04
Epoch 3/20
1639393/1639393 [==============================] - 48s 29us/step - loss: 6.5333e-04 - val_loss: 4.4078e-04
Epoch 4/20
1639393/1639393 [==============================] - 48s 29us/step - loss: 6.4683e-04 - val_loss: 4.4997e-04
Epoch 5/20
```

In [27]:

```python
# make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
# invert predictions
train_predict = scaler.inverse_transform(train_predict)
Y_train = scaler.inverse_transform([Y_train])
test_predict = scaler.inverse_transform(test_predict)
Y_test = scaler.inverse_transform([Y_test])

print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0], train_predict[:,0])))
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0], test_predict[:,0])))
```

```
Train Mean Absolute Error: 0.11166630031104467
Train Root Mean Squared Error: 0.26582184486052096
Test Mean Absolute Error: 0.09755654357173127
Test Root Mean Squared Error: 0.22122063258519137
```

Practical - 12

Aim : Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance.

Code :

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model
```

Load the dataset

To start, you will train the basic autoencoder using the Fashion MNIST dataset. Each image in this dataset is 28x28 pixels.

```
[ ] (x_train, _), (x_test, _) = fashion_mnist.load_data()

    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.

    print (x_train.shape)
    print (x_test.shape)
```

```
class Autoencoder(Model):
  def __init__(self, latent_dim, shape):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.shape = shape
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(tf.math.reduce_prod(shape).numpy(), activation='sigmoid'),
      layers.Reshape(shape)
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded


shape = x_test.shape[1:]
latent_dim = 64
autoencoder = Autoencoder(latent_dim, shape)
```

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

that the model is trained, let's test it by encoding and decoding images from the test set.

```
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

```
(x_train, _), (x_test, _) = fashion_mnist.load_data()
```

```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]

print(x_train.shape)
```

ıg random noise to the images

```
noise_factor = 0.2
x_train_noisy = x_train + noise_factor * tf.random.normal(shape=x_train.shape)
x_test_noisy = x_test + noise_factor * tf.random.normal(shape=x_test.shape)

x_train_noisy = tf.clip_by_value(x_train_noisy, clip_value_min=0., clip_value_max=1.)
x_test_noisy = tf.clip_by_value(x_test_noisy, clip_value_min=0., clip_value_max=1.)
```

```
class Denoise(Model):
  def __init__(self):
    super(Denoise, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Input(shape=(28, 28, 1)),
      layers.Conv2D(16, (3, 3), activation='relu', padding='same', strides=2),
      layers.Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)])

    self.decoder = tf.keras.Sequential([
      layers.Conv2DTranspose(8, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Conv2DTranspose(16, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Denoise()
```

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))
```

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):

    # display original + noise
    ax = plt.subplot(2, n, i + 1)
    plt.title("original + noise")
    plt.imshow(tf.squeeze(x_test_noisy[i]))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    bx = plt.subplot(2, n, i + n + 1)
    plt.title("reconstructed")
    plt.imshow(tf.squeeze(decoded_imgs[i]))
    plt.gray()
    bx.get_xaxis().set_visible(False)
    bx.get_yaxis().set_visible(False)
plt.show()
```

```
# Download the dataset
dataframe = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv', header=None)
raw_data = dataframe.values
dataframe.head()
```

```
# The last element contains the labels
labels = raw_data[:, -1]

# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]

train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=21
)
```

```
reconstructions = autoencoder.predict(anomalous_test_data)
test_loss = tf.keras.losses.mae(reconstructions, anomalous_test_data)

plt.hist(test_loss[None, :], bins=50)
plt.xlabel("Test loss")
plt.ylabel("No of examples")
plt.show()
```

sify an ECG as an anomaly if the reconstruction error is greater than the threshold.

```
def predict(model, data, threshold):
  reconstructions = model(data)
  loss = tf.keras.losses.mae(reconstructions, data)
  return tf.math.less(loss, threshold)

def print_stats(predictions, labels):
  print("Accuracy = {}".format(accuracy_score(labels, predictions)))
  print("Precision = {}".format(precision_score(labels, predictions)))
  print("Recall = {}".format(recall_score(labels, predictions)))
```

```
preds = predict(autoencoder, test_data, threshold)
print_stats(preds, test_labels)
```

Practical - 13

Aim : Generative Adversarial Networks (GANs): Implement GANs on an image dataset and evaluate the model's performance.

Code :

```
import tensorflow as tf
import numpy as np
import datetime
import matplotlib.pyplot as plt
%matplotlib inline

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/")
```

```
sample_image = mnist.train.next_batch(1)[0]
print(sample_image.shape)

sample_image = sample_image.reshape([28, 28])
plt.imshow(sample_image, cmap='Greys')
```

```
def discriminator(images, reuse_variables=None):
    with tf.variable_scope(tf.get_variable_scope(), reuse=reuse_variables) as scope:
        # First convolutional and pool layers
        # This finds 32 different 5 x 5 pixel features
        d_w1 = tf.get_variable('d_w1', [5, 5, 1, 32], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b1 = tf.get_variable('d_b1', [32], initializer=tf.constant_initializer(0))
        d1 = tf.nn.conv2d(input=images, filter=d_w1, strides=[1, 1, 1, 1], padding='SAME')
        d1 = d1 + d_b1
        d1 = tf.nn.relu(d1)
        d1 = tf.nn.avg_pool(d1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

        # Second convolutional and pool layers
        # This finds 64 different 5 x 5 pixel features
        d_w2 = tf.get_variable('d_w2', [5, 5, 32, 64], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b2 = tf.get_variable('d_b2', [64], initializer=tf.constant_initializer(0))
        d2 = tf.nn.conv2d(input=d1, filter=d_w2, strides=[1, 1, 1, 1], padding='SAME')
        d2 = d2 + d_b2
        d2 = tf.nn.relu(d2)
        d2 = tf.nn.avg_pool(d2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

        # First fully connected layer
        d_w3 = tf.get_variable('d_w3', [7 * 7 * 64, 1024], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b3 = tf.get_variable('d_b3', [1024], initializer=tf.constant_initializer(0))
        d3 = tf.reshape(d2, [-1, 7 * 7 * 64])
        d3 = tf.matmul(d3, d_w3)
        d3 = d3 + d_b3
        d3 = tf.nn.relu(d3)

        # Second fully connected layer
        d_w4 = tf.get_variable('d_w4', [1024, 1], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b4 = tf.get_variable('d_b4', [1], initializer=tf.constant_initializer(0))
        d4 = tf.matmul(d3, d_w4) + d_b4

        # d4 contains unscaled values
        return d4
```

```
def generator(z, batch_size, z_dim):
    g_w1 = tf.get_variable('g_w1', [z_dim, 3136], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02)
    g_b1 = tf.get_variable('g_b1', [3136], initializer=tf.truncated_normal_initializer(stddev=0.02))
    g1 = tf.matmul(z, g_w1) + g_b1
    g1 = tf.reshape(g1, [-1, 56, 56, 1])
    g1 = tf.contrib.layers.batch_norm(g1, epsilon=1e-5, scope='g_b1')
    g1 = tf.nn.relu(g1)

    # Generate 50 features
    g_w2 = tf.get_variable('g_w2', [3, 3, 1, z_dim/2], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=
    g_b2 = tf.get_variable('g_b2', [z_dim/2], initializer=tf.truncated_normal_initializer(stddev=0.02))
    g2 = tf.nn.conv2d(g1, g_w2, strides=[1, 2, 2, 1], padding='SAME')
    g2 = g2 + g_b2
    g2 = tf.contrib.layers.batch_norm(g2, epsilon=1e-5, scope='g_b2')
    g2 = tf.nn.relu(g2)
    g2 = tf.image.resize_images(g2, [56, 56])

    # Generate 25 features
    g_w3 = tf.get_variable('g_w3', [3, 3, z_dim/2, z_dim/4], dtype=tf.float32, initializer=tf.truncated_normal_initializer(s
    g_b3 = tf.get_variable('g_b3', [z_dim/4], initializer=tf.truncated_normal_initializer(stddev=0.02))
    g3 = tf.nn.conv2d(g2, g_w3, strides=[1, 2, 2, 1], padding='SAME')
    g3 = g3 + g_b3
    g3 = tf.contrib.layers.batch_norm(g3, epsilon=1e-5, scope='g_b3')
    g3 = tf.nn.relu(g3)
    g3 = tf.image.resize_images(g3, [56, 56])

    # Final convolution with one output channel
    g_w4 = tf.get_variable('g_w4', [1, 1, z_dim/4, 1], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=
    g_b4 = tf.get_variable('g_b4', [1], initializer=tf.truncated_normal_initializer(stddev=0.02))
    g4 = tf.nn.conv2d(g3, g_w4, strides=[1, 2, 2, 1], padding='SAME')
    g4 = g4 + g_b4
    g4 = tf.sigmoid(g4)

    # Dimensions of g4: batch_size x 28 x 28 x 1
    return g4
```

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    generated_image = sess.run(generated_image_output,
                               feed_dict={z_placeholder: z_batch})
    generated_image = generated_image.reshape([28, 28])
    plt.imshow(generated_image, cmap='Greys')
```

```
tf.reset_default_graph()
batch_size = 50

z_placeholder = tf.placeholder(tf.float32, [None, z_dimensions], name='z_placeholder')
# z_placeholder is for feeding input noise to the generator

x_placeholder = tf.placeholder(tf.float32, shape = [None,28,28,1], name='x_placeholder')
# x_placeholder is for feeding input images to the discriminator

Gz = generator(z_placeholder, batch_size, z_dimensions)
# Gz holds the generated images

Dx = discriminator(x_placeholder)
# Dx will hold discriminator prediction probabilities
# for the real MNIST images

Dg = discriminator(Gz, reuse_variables=True)
# Dg will hold discriminator prediction probabilities for generated images
```

```python
for i in range(100000):
    real_image_batch = mnist.train.next_batch(batch_size)[0].reshape([batch_size, 28, 28, 1])
    z_batch = np.random.normal(0, 1, size=[batch_size, z_dimensions])

    # Train discriminator on both real and fake images
    _, __, dLossReal, dLossFake = sess.run([d_trainer_real, d_trainer_fake, d_loss_real, d_loss_fake],
                                           {x_placeholder: real_image_batch, z_placeholder: z_batch})

    # Train generator
    z_batch = np.random.normal(0, 1, size=[batch_size, z_dimensions])
    _ = sess.run(g_trainer, feed_dict={z_placeholder: z_batch})

    if i % 10 == 0:
        # Update TensorBoard with summary statistics
        z_batch = np.random.normal(0, 1, size=[batch_size, z_dimensions])
        summary = sess.run(merged, {z_placeholder: z_batch, x_placeholder: real_image_batch})
        writer.add_summary(summary, i)

    if i % 100 == 0:
        # Every 100 iterations, show a generated image
        print("Iteration:", i, "at", datetime.datetime.now())
        z_batch = np.random.normal(0, 1, size=[1, z_dimensions])
        generated_images = generator(z_placeholder, 1, z_dimensions)
        images = sess.run(generated_images, {z_placeholder: z_batch})
        plt.imshow(images[0].reshape([28, 28]), cmap='Greys')
        plt.show()

        # Show discriminator's estimate
        im = images[0].reshape([1, 28, 28, 1])
        result = discriminator(x_placeholder)
        estimate = sess.run(result, {x_placeholder: im})
        print("Estimate:", estimate)
```

Practical - 14

Aim: Transfer Learning: Implement transfer learning on an image dataset and evaluate the model's performance.

Dataset Used:

Code:

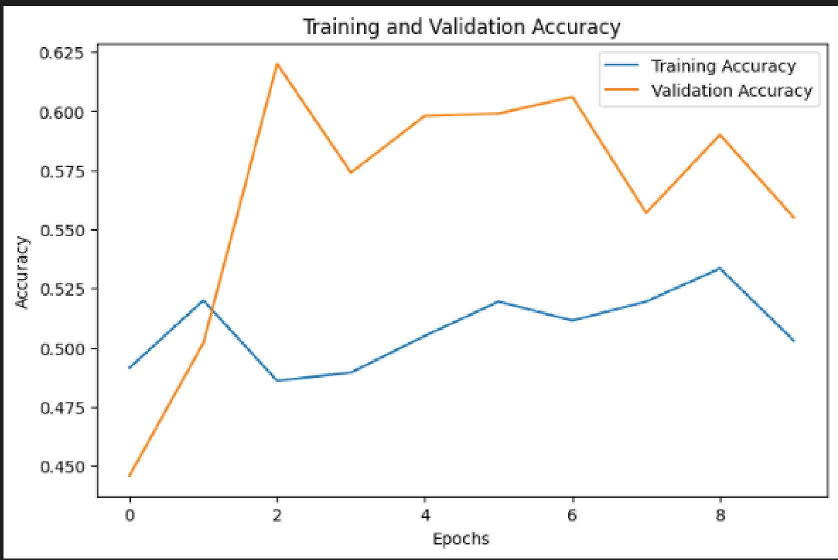
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import os

# Set paths for the Cats vs. Dogs dataset
base_dir = r"./cats_and_dogs_filtered"
train_dir = os.path.join(base_dir, "train")
validation_dir = os.path.join(base_dir, "validation")
print(f"Validation directory: {validation_dir}")

# Image data generators for preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest"
)

validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
```

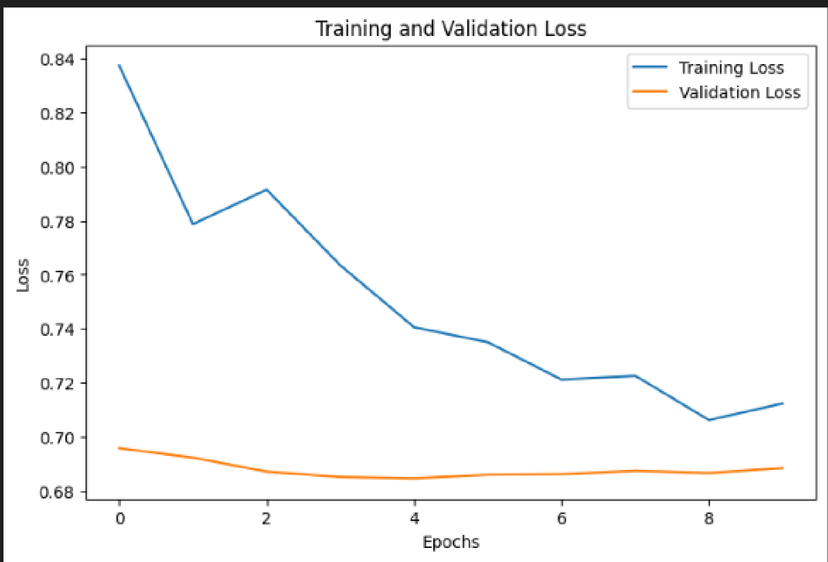
```
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

# Load the ResNet50 model pre-trained on ImageNet
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=(224,
224, 3))

# Freeze all layers of the base model
base_model.trainable = False

# Add custom layers on top
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Early stopping for better performance
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[early_stopping]
)

# Evaluate the model
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")

# Plot training and validation accuracy
```

```
plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output:

```
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 46s 726ms/step - accuracy: 0.5256 - loss: 0.7063 - val_accuracy: 0.5900 - val_loss: 0.6865
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 43s 682ms/step - accuracy: 0.5077 - loss: 0.7077 - val_accuracy: 0.5550 - val_loss: 0.6884
32/32 ━━━━━━━━━━━━━━━━━━━━ 13s 413ms/step - accuracy: 0.6039 - loss: 0.6847
Validation Loss: 0.6845
Validation Accuracy: 0.5980
```





Practical - 15

Aim: Reinforcement Learning: Implement reinforcement learning on a game environment and evaluate the model's performance.

Code:

```
import gym
import numpy as np
import warnings

# Suppress specific deprecation warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Load the environment with render mode specified
env = gym.make('CartPole-v1', render_mode="human")

# Initialize the environment to get the initial state
state = env.reset()

# Print the state space and action space
print("State space:", env.observation_space)
print("Action space:", env.action_space)


for _ in range(10):
    env.render()
    action = env.action_space.sample()

    step_result = env.step(action)

    if len(step_result) == 4:
        next_state, reward, done, info = step_result
        terminated = False
    else:
        next_state, reward, done, truncated, info = step_result
        terminated = done or truncated

    print(f"Action: {action}, Reward: {reward}, Next State: {next_state},
Done: {done}, Info: {info}")

    if terminated:
        state = env.reset()

env.close()
```

Output

1] ✓ 2.4s Pythor

```
State space: Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1
Action space: Discrete(2)
Action: 0, Reward: 1.0, Next State: [-0.00504123 -0.24225019  0.04615125  0.26604646], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.00988624 -0.04781627  0.05147218 -0.01173022], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.01084256  0.14653116  0.05123757 -0.28773913], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.00791194  0.3408864   0.04548279 -0.56383216], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.00109421  0.5353417   0.03420615 -0.84184605], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [ 0.00961262  0.3397699   0.01736923 -0.5386054 ], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [ 0.01640802  0.5346434   0.00659712 -0.82576525], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [ 0.02710089  0.7296745  -0.00991819 -1.116366  ], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [ 0.04169438  0.5346842  -0.03224551 -0.8268108 ], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [ 0.05238806  0.34001765 -0.04878172 -0.5444413 ], Done: False, Info: {}
```