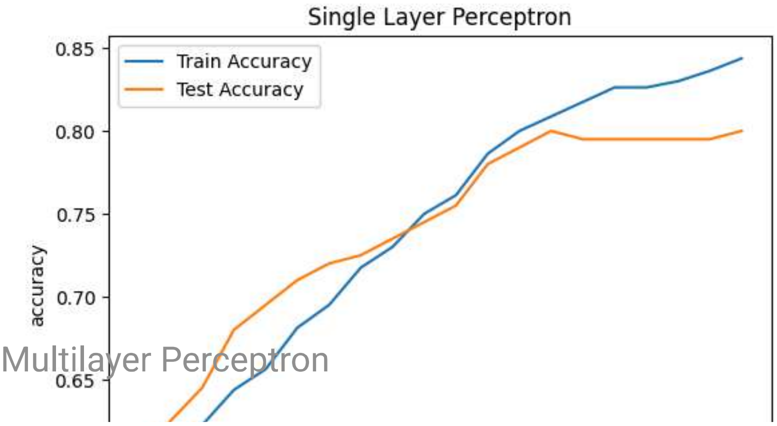
Single Layer Perceptron

```python
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
x,y = make_classification(n_samples=1000,n_features=10,random_state=42)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
scaler=StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
model = Sequential([Dense(1,activation='sigmoid',input_shape=(x_train.shape[1],))])

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
train = model.fit(x_train,y_train,epochs=20,batch_size=16,validation_data=(x_test,y_test))

test_loss,test_acc = model.evaluate(x_test,y_test,verbose=2)
print("Test Accuracy")

plt.plot(train.history['accuracy'],label="Train Accuracy")
plt.plot(train.history['val_accuracy'],label="Test Accuracy")
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.title('Single Layer Perceptron')
plt.show()
```

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5548 - loss: 0.7977 - val_accuracy: 0.5900 - val_loss: 0.7638
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5856 - loss: 0.7392 - val_accuracy: 0.6250 - val_loss: 0.7216
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6285 - loss: 0.7045 - val_accuracy: 0.6450 - val_loss: 0.6850
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6378 - loss: 0.6649 - val_accuracy: 0.6800 - val_loss: 0.6517
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6361 - loss: 0.6438 - val_accuracy: 0.6950 - val_loss: 0.6227
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6609 - loss: 0.6367 - val_accuracy: 0.7100 - val_loss: 0.5965
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6698 - loss: 0.6002 - val_accuracy: 0.7200 - val_loss: 0.5739
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7173 - loss: 0.5298 - val_accuracy: 0.7250 - val_loss: 0.5541
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7354 - loss: 0.5182 - val_accuracy: 0.7350 - val_loss: 0.5360
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7642 - loss: 0.4963 - val_accuracy: 0.7450 - val_loss: 0.5199
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7503 - loss: 0.4756 - val_accuracy: 0.7550 - val_loss: 0.5055
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7517 - loss: 0.5367 - val_accuracy: 0.7800 - val_loss: 0.4929
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7783 - loss: 0.4739 - val_accuracy: 0.7900 - val_loss: 0.4823
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8199 - loss: 0.4338 - val_accuracy: 0.8000 - val_loss: 0.4723
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7947 - loss: 0.4614 - val_accuracy: 0.7950 - val_loss: 0.4633
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8350 - loss: 0.4289 - val_accuracy: 0.7950 - val_loss: 0.4558
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8204 - loss: 0.4150 - val_accuracy: 0.7950 - val_loss: 0.4491
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8280 - loss: 0.4175 - val_accuracy: 0.7950 - val_loss: 0.4429
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8248 - loss: 0.4124 - val_accuracy: 0.7950 - val_loss: 0.4377
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8331 - loss: 0.4058 - val_accuracy: 0.8000 - val_loss: 0.4328
7/7 - 0s - 7ms/step - accuracy: 0.8000 - loss: 0.4328
Test Accuracy
```



Multilayer Perceptron

```
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv('/content/HR_comma_sep.csv')
data['salary'] = preprocessing.LabelEncoder().fit_transform(data['salary'])
data['sales'] = preprocessing.LabelEncoder().fit_transform(data['sales'])
print(data.head())
x = data[['satisfaction_level','last_evaluation','number_project','average_montly_hours','time_spend_company','Work_accident','promotior
y = data['left']

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
model =MLPClassifier(hidden_layer_sizes=(6,5),random_state=5,verbose=True,learning_rate_init=0.01)
model.fit(x_train,y_train)

ypred = model.predict(x_test)
print("Accuracy:",accuracy_score(y_test,ypred))
```

Show hidden output

Implementation of Feedforward Neural Network using Tensorflow

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_california_housing
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

california = fetch_california_housing()
df = pd.DataFrame(california.data,columns=california.feature_names)
x=df
y=california.target

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 42)

scaler =StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

model = Sequential([
    Dense(64,activation='relu',input_shape=(x_train.shape[1],)),
    Dense(32,activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam',loss='mse',metrics=['mae'])

model.fit(x_train,y_train,epochs=20,batch_size=10,validation_data=(x_test,y_test))

loss,mae = model.evaluate(x_test,y_test)
print("Mean Absolute error :",mae)


```

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.9348 - mae: 0.6560 - val_loss: 0.3976 - val_mae: 0.4576
Epoch 2/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.3894 - mae: 0.4441 - val_loss: 0.3787 - val_mae: 0.4466
Epoch 3/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 0.3669 - mae: 0.4283 - val_loss: 0.4840 - val_mae: 0.4264
Epoch 4/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.3807 - mae: 0.4257 - val_loss: 0.3310 - val_mae: 0.3992
Epoch 5/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.3197 - mae: 0.3986 - val_loss: 0.3186 - val_mae: 0.3891
Epoch 6/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.3235 - mae: 0.3994 - val_loss: 0.3445 - val_mae: 0.3917
Epoch 7/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 0.3256 - mae: 0.3888 - val_loss: 0.3266 - val_mae: 0.4052
Epoch 8/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.3000 - mae: 0.3803 - val_loss: 0.3038 - val_mae: 0.3752
```

```
Epoch 9/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 0.2921 - mae: 0.3761 - val_loss: 0.3061 - val_mae: 0.3887
Epoch 10/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.2836 - mae: 0.3692 - val_loss: 0.3123 - val_mae: 0.4051
Epoch 11/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.3032 - mae: 0.3779 - val_loss: 0.3000 - val_mae: 0.3899
Epoch 12/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 0.2827 - mae: 0.3694 - val_loss: 0.3018 - val_mae: 0.3903
Epoch 13/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 0.2789 - mae: 0.3665 - val_loss: 0.2959 - val_mae: 0.3688
Epoch 14/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.2892 - mae: 0.3702 - val_loss: 0.2960 - val_mae: 0.3679
Epoch 15/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 0.2801 - mae: 0.3646 - val_loss: 0.2985 - val_mae: 0.3634
Epoch 16/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 0.2809 - mae: 0.3650 - val_loss: 0.2869 - val_mae: 0.3619
Epoch 17/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 0.2778 - mae: 0.3622 - val_loss: 0.2986 - val_mae: 0.3663
Epoch 18/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 0.2860 - mae: 0.3642 - val_loss: 0.2782 - val_mae: 0.3603
Epoch 19/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.2747 - mae: 0.3549 - val_loss: 0.3056 - val_mae: 0.3734
Epoch 20/20
1445/1445 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 0.2720 - mae: 0.3567 - val_loss: 0.3119 - val_mae: 0.3799
194/194 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3107 - mae: 0.3793
Mean Absolute error : 0.37985995411872864
```

Forward Neural Network for Binary Classification

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

x,y = make_classification(n_samples=1000,n_features=20,n_classes=2,random_state=42)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)

scaler=StandardScaler()

x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)

model=Sequential([
    Dense(64,activation='relu',input_shape=(x_train.shape[1],)),
    Dense(32,activation='relu'),
    Dense(1,activation='sigmoid')
])

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(x_train,y_train,epochs=10,batch_size=32,validation_data=(x_test,y_test))
loss,accuracy=model.evaluate(x_test,y_test)
print("Accoracy:",accuracy)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.6456 - loss: 0.6346 - val_accuracy: 0.7667 - val_loss: 0.5581
Epoch 2/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7827 - loss: 0.5246 - val_accuracy: 0.7967 - val_loss: 0.4762
Epoch 3/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8521 - loss: 0.4195 - val_accuracy: 0.8100 - val_loss: 0.4320
Epoch 4/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8829 - loss: 0.3504 - val_accuracy: 0.8133 - val_loss: 0.4048
Epoch 5/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9074 - loss: 0.3151 - val_accuracy: 0.8300 - val_loss: 0.3913
Epoch 6/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9046 - loss: 0.2878 - val_accuracy: 0.8367 - val_loss: 0.3879
Epoch 7/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8898 - loss: 0.2758 - val_accuracy: 0.8333 - val_loss: 0.3878
Epoch 8/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9149 - loss: 0.2546 - val_accuracy: 0.8333 - val_loss: 0.3886
Epoch 9/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9007 - loss: 0.2626 - val_accuracy: 0.8300 - val_loss: 0.3837
```

```
Epoch 10/10
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9048 - loss: 0.2409 - val_accuracy: 0.8267 - val_loss: 0.3928
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8180 - loss: 0.3927
Accoracy: 0.8266666531562805
```

Anomaly Detection using Autoencoder

```python
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_squared_error

dataset = load_breast_cancer()
df = pd.DataFrame(dataset.data,columns=dataset.feature_names)

scaler=MinMaxScaler()
x=scaler.fit_transform(df.values)

x_train,x_test = train_test_split(x,test_size=0.3,random_state=42)
encoder=Sequential([
    Dense(64,activation='relu',input_shape=(x_train.shape[1],)),
    Dense(32,activation='relu'),
    Dense(16,activation='relu')
])

decoder = Sequential([
    Dense(32,activation='relu',input_shape=(16,)),
    Dense(64,activation='relu'),
    Dense(x_train.shape[1],activation='sigmoid')
])

model = Sequential([encoder,decoder])
model.compile(optimizer='adam',loss='mse')
model.fit(x_train,x_train,epochs=20,batch_size=32,validation_data=(x_test,x_test))
y_pred=model.predict(x_test)
mse_error = mean_squared_error(x_test,y_pred)
print("reconstruction loss:",mse_error)
```

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
13/13 ━━━━━━━━━━━━━━━━━━━━ 4s 26ms/step - loss: 0.0972 - val_loss: 0.0811
Epoch 2/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0747 - val_loss: 0.0495
Epoch 3/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0436 - val_loss: 0.0360
Epoch 4/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0338 - val_loss: 0.0304
Epoch 5/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0289 - val_loss: 0.0267
Epoch 6/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0246 - val_loss: 0.0227
Epoch 7/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0228 - val_loss: 0.0184
Epoch 8/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0158 - val_loss: 0.0162
Epoch 9/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0152 - val_loss: 0.0151
Epoch 10/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0139 - val_loss: 0.0140
Epoch 11/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0121 - val_loss: 0.0132
Epoch 12/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0132 - val_loss: 0.0124
Epoch 13/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0116 - val_loss: 0.0113
Epoch 14/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0097 - val_loss: 0.0103
Epoch 15/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0098 - val_loss: 0.0092
Epoch 16/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0093 - val_loss: 0.0082
```

```
Epoch 17/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0077 - val_loss: 0.0076
Epoch 18/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0070 - val_loss: 0.0072
Epoch 19/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0067 - val_loss: 0.0070
Epoch 20/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0063 - val_loss: 0.0069
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
reconstruction loss: 0.0068841951407840225
```

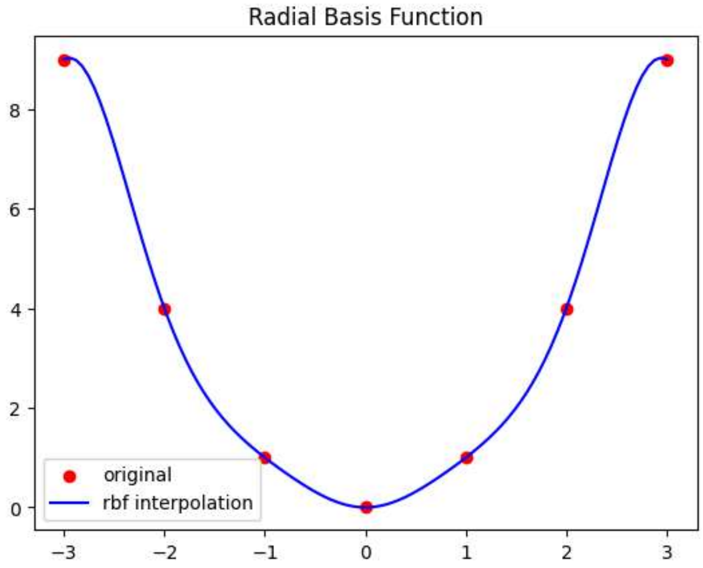
Support Vector Machine (SVM)

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data,columns = iris.feature_names)
x = df
y = iris.target

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)

model = SVC(kernel = 'linear', C = 1)
model.fit(x_train,y_train)

svm_pred = model.predict(x_test)
accuracy = model.score(x_test,y_test)

print("svm accuracy",accuracy)

```

```
svm accuracy 1.0
```

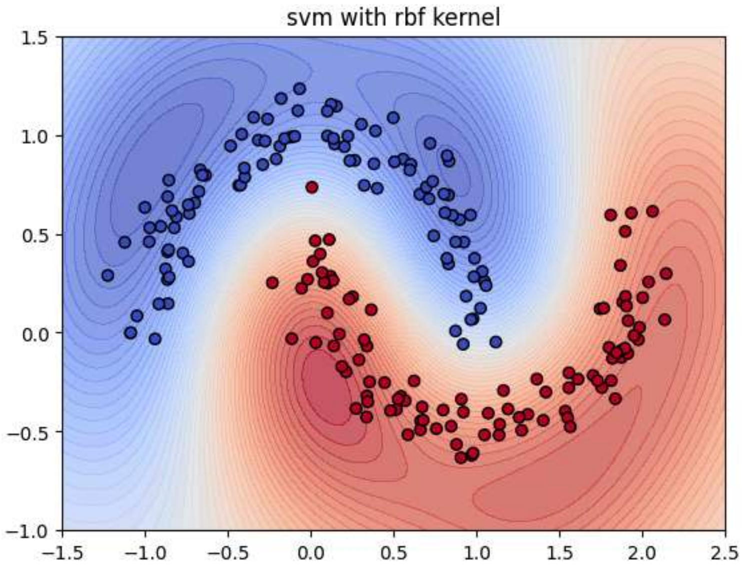
Radial Basis Function (RBF) Interpolation with Gaussian Kernel

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import Rbf

x = np.array([-3,-2,-1,0,1,2,3])
y = np.array([9,4,1,0,1,4,9])
rbf =Rbf(x,y,function = 'gaussian')

x_new = np.linspace(-3,3,100)
y_new = rbf(x_new)
plt.scatter(x,y,color = 'red',label='original')
plt.plot(x_new,y_new,color = 'blue', label = 'rbf interpolation')
plt.legend()
plt.title('Radial Basis Function')
plt.show()
```



Radial Basis Function using SVM Kernel

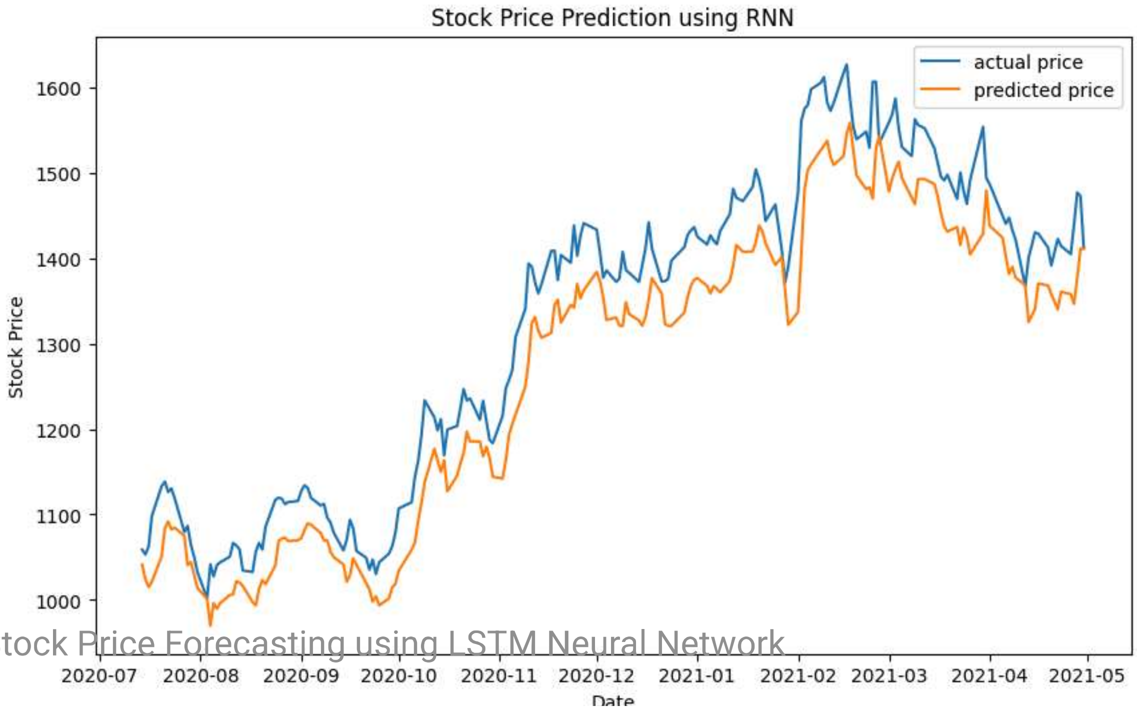
```
from sklearn.svm import SVC
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
import numpy as np
x,y = make_moons(n_samples=200,noise=0.1,random_state=42)

svm_rbf =SVC(kernel ='rbf',gamma = 1.0)
svm_rbf.fit(x,y)
xx,yy = np.meshgrid(np.linspace(-1.5,2.5,100),np.linspace(-1,1.5,100))
z= svm_rbf.decision_function(np.c_[xx.ravel(),yy.ravel()])
z=z.reshape(xx.shape)
plt.contourf(xx,yy,z,levels=np.linspace(z.min(),z.max(),50),cmap="coolwarm",alpha=0.7)
plt.scatter(x[:,0],x[:,1],c=y,edgecolors='k',cmap='coolwarm')
plt.title("svm with rbf kernel")
plt.show()
```



Stock Price Prediction using Recurrent Neural Network (RNN)

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np

data = pd.read_csv('/content/HDFCBANK.csv', parse_dates=['Date'], index_col='Date')
data = data[['Close']]

scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

def create_sequences(data, time_steps=60):
    x, y = [], []
    for i in range(len(data) - time_steps):
        x.append(data[i:i+time_steps].reshape(-1, 1))
        y.append(data[i+time_steps])
    return np.array(x), np.array(y)

time_steps = 60

x, y = create_sequences(data_scaled, time_steps)

x_train, x_test = x[:-200], x[-200:]
y_train, y_test = y[:-200], y[-200:]


model = Sequential([
    SimpleRNN(50, activation='relu', return_sequences=True, input_shape=(time_steps, 1)),
    SimpleRNN(50, activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')

model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))

predictions = model.predict(x_test)

predicted_prices = scaler.inverse_transform(predictions)
actual_prices = scaler.inverse_transform(y_test.reshape(-1, 1))

plt.figure(figsize=(10, 6))
plt.plot(data.index[-200:], actual_prices, label="actual price")
plt.plot(data.index[-200:], predicted_prices, label="predicted price")
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Stock Price Prediction using RNN')
plt.legend()
plt.show()

```

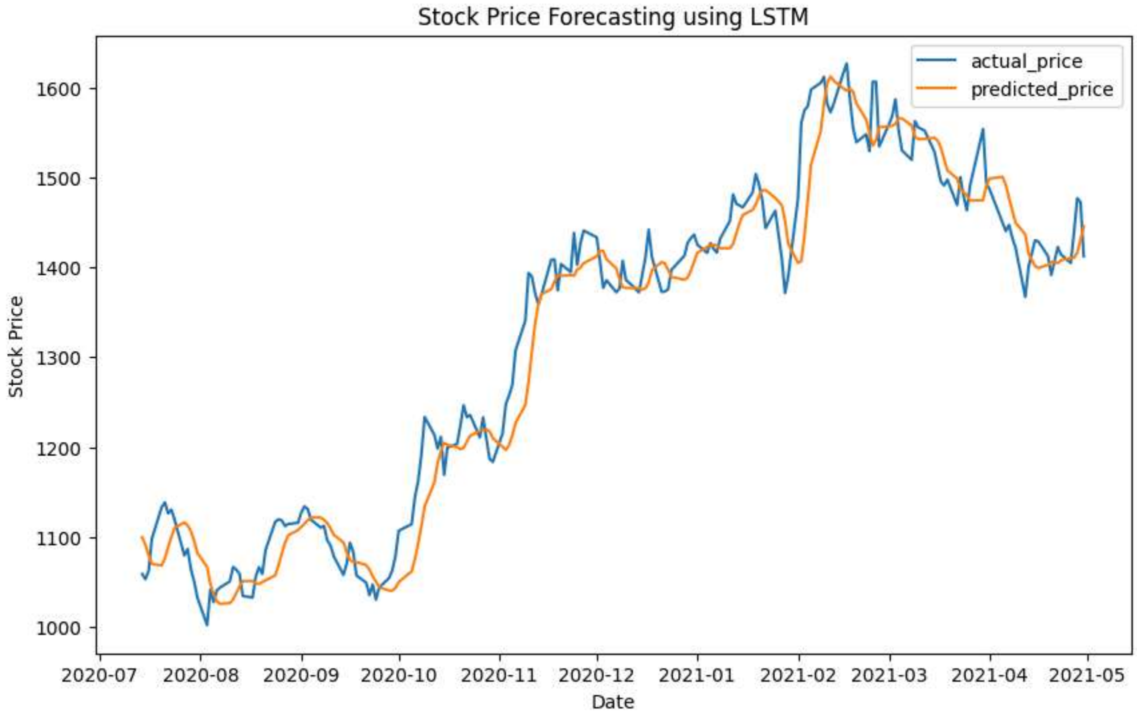
```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
158/158 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 0.0156 - val_loss: 1.6638e-04
Epoch 2/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 3.0420e-04 - val_loss: 1.5564e-04
Epoch 3/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 2.7124e-04 - val_loss: 1.3413e-04
Epoch 4/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 3.1942e-04 - val_loss: 1.3087e-04
Epoch 5/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 1.1716e-04 - val_loss: 0.0024
Epoch 6/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 6.3842e-04 - val_loss: 1.1898e-04
Epoch 7/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 3.9358e-04 - val_loss: 1.2142e-04
Epoch 8/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 1.1178e-04 - val_loss: 3.7473e-04
Epoch 9/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 3.9093e-04 - val_loss: 1.2011e-04
Epoch 10/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 1.7103e-04 - val_loss: 6.4024e-04
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step
```



Stock Price Forecasting using LSTM Neural Network

```python
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np

data = pd.read_csv('/content/HDFCBANK.csv',parse_dates=['Date'],index_col='Date')
data = data[['Close']]

scaler = MinMaxScaler(feature_range=(0,1))

data_scaled = scaler.fit_transform(data)

def create_sequences(data,time_steps=60):
  x,y=[],[]
  for i in range(len(data)-time_steps):
    x.append(data[i:i+time_steps].reshape(-1,1))
    y.append(data[i+time_steps])
  return np.array(x),np.array(y)

x,y = create_sequences(data_scaled,60)
x_train,x_test = x[:-200],x[-200:]
y_train,y_test = y[:-200],y[-200:]

model = Sequential([
    LSTM(50,activation='relu',return_sequences=True,input_shape=(60,1)),
    LSTM(50,activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam',loss='mse')

model.fit(x_train,y_train,epochs=10,batch_size=32,validation_data=(x_test,y_test))

predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
actual_prices = scaler.inverse_transform(y_test.reshape(-1,1))

plt.figure(figsize=(10,6))

plt.plot(data.index[-200:],actual_prices,label="actual_price")
plt.plot(data.index[-200:],predictions,label="predicted_price")
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Stock Price Forecasting using LSTM')
plt.legend()
plt.show()
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
158/158 ━━━━━━━━━━━━━━━━━━━━ 12s 30ms/step - loss: 0.0542 - val_loss: 4.9899e-04
Epoch 2/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - loss: 0.0012 - val_loss: 0.0012
Epoch 3/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0010 - val_loss: 4.5544e-04
Epoch 4/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0010 - val_loss: 3.8728e-04
Epoch 5/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.7163e-04 - val_loss: 5.2300e-04
Epoch 6/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 6.3780e-04 - val_loss: 3.5852e-04
Epoch 7/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 7.0148e-04 - val_loss: 3.0141e-04
Epoch 8/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 5.5206e-04 - val_loss: 3.0349e-04
Epoch 9/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 7.2926e-04 - val_loss: 2.6905e-04
Epoch 10/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 9.7683e-04 - val_loss: 2.6784e-04
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 101ms/step
```

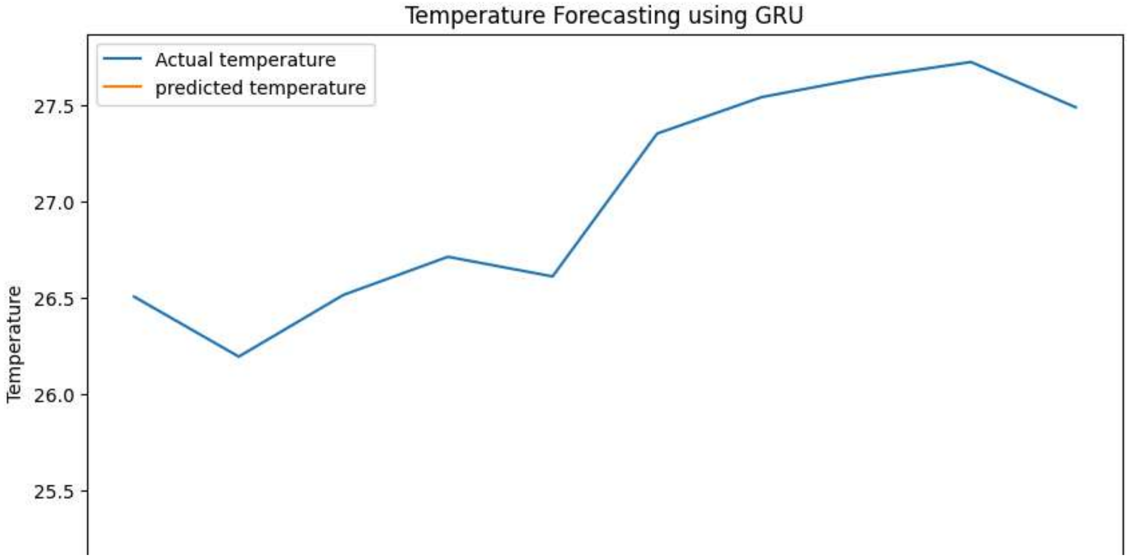


Temperature Forecasting using GRU Neural Networks

```
import pandas as pd
import numpy as np

# Set random seed for reproducibility
np.random.seed(42)

# Generate synthetic years from 1920 to 2020
years = np.arange(1920, 2021)

# Generate synthetic temperature data:
# Simulate a slow upward trend + seasonal sine wave + noise
temps = 25 + 0.02 * (years - 1920) + np.sin((years - 1920) / 5) + np.random.normal(0, 0.3, len(years))

# Create DataFrame
synthetic_df = pd.DataFrame({
    'YEAR': years,
    'ANNUAL': temps
})

# Save as CSV (you can save to your preferred path)
synthetic_df.to_csv('temperatures.csv', index=False)

```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU,Dense
data = pd.read_csv('/content/temperatures.csv',usecols=['YEAR','ANNUAL'])
data.dropna(inplace = True)

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data[['ANNUAL']])

def create_sequences(data,time_steps=60):
  x,y=[],[]
  for i in range(len(data)-time_step):
    x.append(data[i:i+time_step].reshape(time_step,1))
    y.append(data[i+time_steps])
  return np.array(x),np.array(y)

time_step = 60
x,y = create_sequences(scaled_data,time_steps)

x_train,x_test= x[:-10],x[-10:]
y_train,y_test = y[:-10],y[-10:]

model = Sequential([
    GRU(20,activation='relu',return_sequences=True,input_shape=(time_steps,1)),
    GRU(20,activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam',loss='mse')

model.fit(x_train,y_train,epochs=5,batch_size=8,validation_data=(x_test,y_test))
predictions = model.predict(x_test)
pred_temp =scaler.inverse_transform(predictions)
actual_temp = scaler.inverse_transform(y_test.reshape(-1,1))

plt.figure(figsize=(10,6))
plt.plot(actual_temp,label="Actual temperature")
plt.plot(predicted_temp,label="predicted temperature")
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title('Temperature Forecasting using GRU')
plt.legend()
plt.show()


```

```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 969ms/step - loss: 0.4303 - val_loss: 0.6191
Epoch 2/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - loss: 0.4504 - val_loss: 0.5654
Epoch 3/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 0.3544 - val_loss: 0.5143
Epoch 4/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 0.3341 - val_loss: 0.4674
Epoch 5/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 0.3341 - val_loss: 0.4209
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 610ms/step
```


Temperature Forecasting using GRU
Actual temperature
predicted temperature
Temperature