dlcv-programs

May 4, 2024

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

[ ]:
```python
df = pd.read_csv("/content/kc_house_data - kc_house_data.csv")
```

[ ]:
```python
df.head()
```

[ ]:
```
           id        date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  10/13/2014  221900.0         3       1.00         1180
1  6414100192   12/9/2014  538000.0         3       2.25         2570
2  5631500400   2/25/2015  180000.0         2       1.00          770
3  2487200875   12/9/2014  604000.0         4       3.00         1960
4  1954400510   2/18/2015  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0      5650     1.0           0     0  …      7        1180              0
1      7242     2.0           0     0  …      7        2170            400
2     10000     1.0           0     0  …      6         770              0
3      5000     1.0           0     0  …      7        1050            910
4      8080     1.0           0     0  …      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503
```

```
[5 rows x 21 columns]
```

[ ]:
```
df.describe()
```

[ ]:
```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.159700e+04  2.159700e+04  21597.000000  21597.000000  21597.000000
mean   4.580474e+09  5.402966e+05      3.373200      2.115826   2080.321850
std    2.876736e+09  3.673681e+05      0.926299      0.768984    918.106125
min    1.000102e+06  7.800000e+04      1.000000      0.500000    370.000000
25%    2.123049e+09  3.220000e+05      3.000000      1.750000   1430.000000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1910.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000  13540.000000

           sqft_lot        floors    waterfront          view     condition  \
count  2.159700e+04  21597.000000  21597.000000  21597.000000  21597.000000
mean   1.509941e+04      1.494096      0.007547      0.234292      3.409825
std    4.141264e+04      0.539683      0.086549      0.766390      0.650546
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000
50%    7.618000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.068500e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21597.000000  21597.000000   21597.000000  21597.000000  21597.000000
mean       7.657915   1788.596842     291.725008   1970.999676     84.464787
std        1.173200    827.759761     442.667800     29.375234    401.821438
min        3.000000    370.000000       0.000000   1900.000000      0.000000
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000

            zipcode           lat          long  sqft_living15     sqft_lot15
count  21597.000000  21597.000000  21597.000000   21597.000000   21597.000000
mean   98077.951845     47.560093   -122.213982    1986.620318   12758.283512
std       53.513072      0.138552      0.140724     685.230472   27274.441950
min    98001.000000     47.155900   -122.519000     399.000000     651.000000
25%    98033.000000     47.471100   -122.328000    1490.000000    5100.000000
50%    98065.000000     47.571800   -122.231000    1840.000000    7620.000000
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000
```

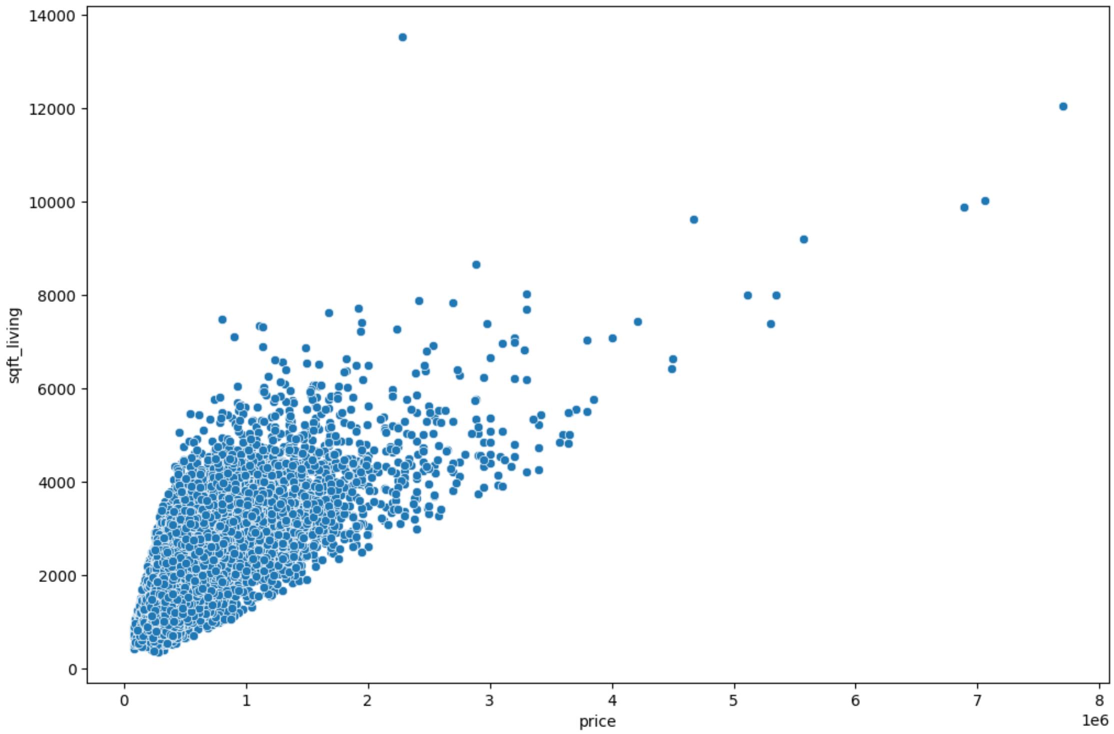
[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  object
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     21597 non-null  int64
 9   view           21597 non-null  int64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  int64
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  int64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 non-null  int64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

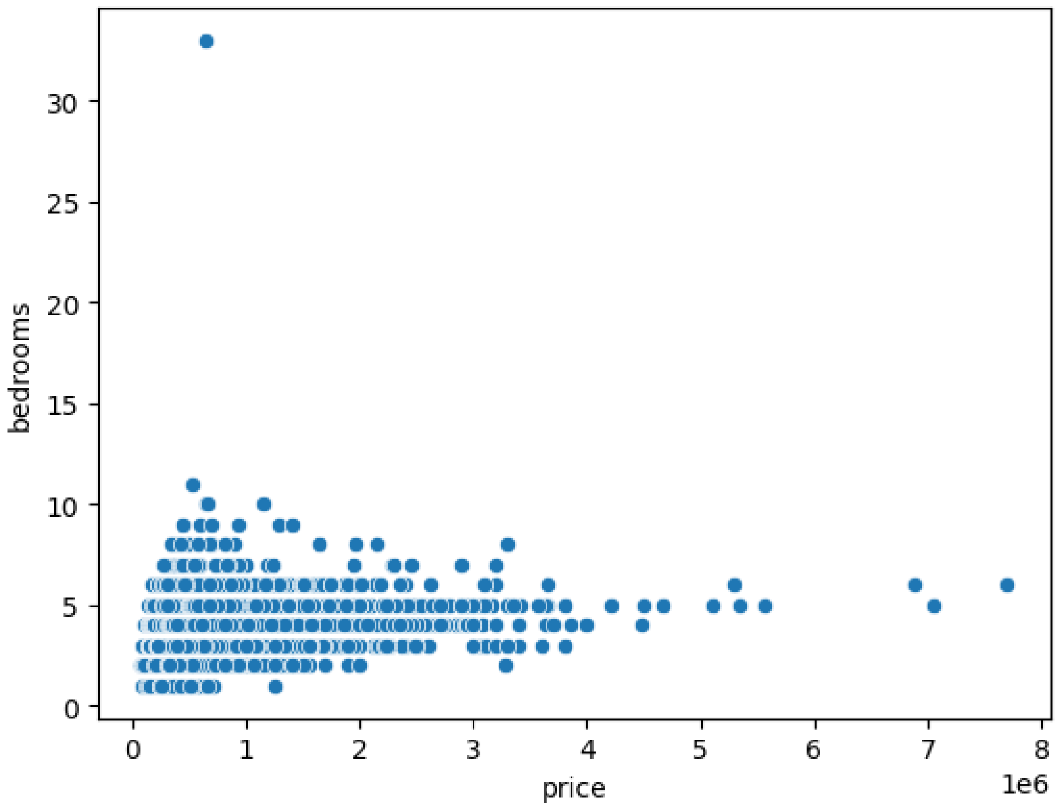
[ ]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x='price',y='sqft_living',data=df)
```

[ ]:
```
<Axes: xlabel='price', ylabel='sqft_living'>
```



[ ]:
```
sns.scatterplot(x='price',y='bedrooms',data=df)
```

[ ]: <Axes: xlabel='price', ylabel='bedrooms'>



[ ]:
```
df = df.drop('id',axis=1)
```

[ ]:
```
df = df.drop('date',axis=1)
```

[ ]:
```
df = df.drop('zipcode',axis=1)
```

[ ]:
```
X = df.drop('price',axis=1)
y = df['price']
```

[ ]:
```
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,random_state=101)
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train= scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
```

```python
from tensorflow.keras.optimizers import Adam
```

[ ]:
```python
model = Sequential()

model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(1))

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

[ ]:
```python
model.fit(x=X_train,y=y_train.values,
          validation_data=(X_test,y_test.values),
          batch_size=128,epochs=40)
```

```
Epoch 1/40
119/119 [==============================] - 2s 4ms/step - loss: 430249672704.0000
- val_loss: 418971287552.0000
Epoch 2/40
119/119 [==============================] - 0s 3ms/step - loss: 429839089664.0000
- val_loss: 417413103616.0000
Epoch 3/40
119/119 [==============================] - 0s 3ms/step - loss: 422650347520.0000
- val_loss: 400060121088.0000
Epoch 4/40
119/119 [==============================] - 0s 2ms/step - loss: 381574643712.0000
- val_loss: 327786987520.0000
Epoch 5/40
119/119 [==============================] - 0s 3ms/step - loss: 270787510272.0000
- val_loss: 188310421504.0000
Epoch 6/40
119/119 [==============================] - 0s 2ms/step - loss: 139636097024.0000
- val_loss: 98309341184.0000
Epoch 7/40
119/119 [==============================] - 0s 2ms/step - loss: 95262654464.0000
- val_loss: 89510789120.0000
Epoch 8/40
119/119 [==============================] - 0s 2ms/step - loss: 91792924672.0000
- val_loss: 88237350912.0000
Epoch 9/40
119/119 [==============================] - 0s 4ms/step - loss: 90371563520.0000
- val_loss: 86868508672.0000
Epoch 10/40
119/119 [==============================] - 0s 4ms/step - loss: 88918220800.0000
- val_loss: 85576867840.0000
Epoch 11/40
```

```
119/119 [==============================] - 0s 4ms/step - loss: 87456022528.0000
- val_loss: 84077469696.0000
Epoch 12/40
119/119 [==============================] - 0s 4ms/step - loss: 85961760768.0000
- val_loss: 82615631872.0000
Epoch 13/40
119/119 [==============================] - 0s 4ms/step - loss: 84426514432.0000
- val_loss: 81133944832.0000
Epoch 14/40
119/119 [==============================] - 0s 4ms/step - loss: 82835832832.0000
- val_loss: 79603769344.0000
Epoch 15/40
119/119 [==============================] - 0s 2ms/step - loss: 81204551680.0000
- val_loss: 78028111872.0000
Epoch 16/40
119/119 [==============================] - 0s 2ms/step - loss: 79485132800.0000
- val_loss: 76351594496.0000
Epoch 17/40
119/119 [==============================] - 0s 2ms/step - loss: 77710712832.0000
- val_loss: 74618806272.0000
Epoch 18/40
119/119 [==============================] - 0s 3ms/step - loss: 75940216832.0000
- val_loss: 72907931648.0000
Epoch 19/40
119/119 [==============================] - 0s 3ms/step - loss: 74136600576.0000
- val_loss: 71116767232.0000
Epoch 20/40
119/119 [==============================] - 0s 3ms/step - loss: 72276418560.0000
- val_loss: 69305352192.0000
Epoch 21/40
119/119 [==============================] - 0s 2ms/step - loss: 70353641472.0000
- val_loss: 67475832832.0000
Epoch 22/40
119/119 [==============================] - 0s 2ms/step - loss: 68435341312.0000
- val_loss: 65610268672.0000
Epoch 23/40
119/119 [==============================] - 0s 2ms/step - loss: 66520584192.0000
- val_loss: 63833935872.0000
Epoch 24/40
119/119 [==============================] - 0s 3ms/step - loss: 64601067520.0000
- val_loss: 61938171904.0000
Epoch 25/40
119/119 [==============================] - 0s 2ms/step - loss: 62775332864.0000
- val_loss: 60234399744.0000
Epoch 26/40
119/119 [==============================] - 0s 2ms/step - loss: 60946124800.0000
- val_loss: 58477793280.0000
Epoch 27/40
```

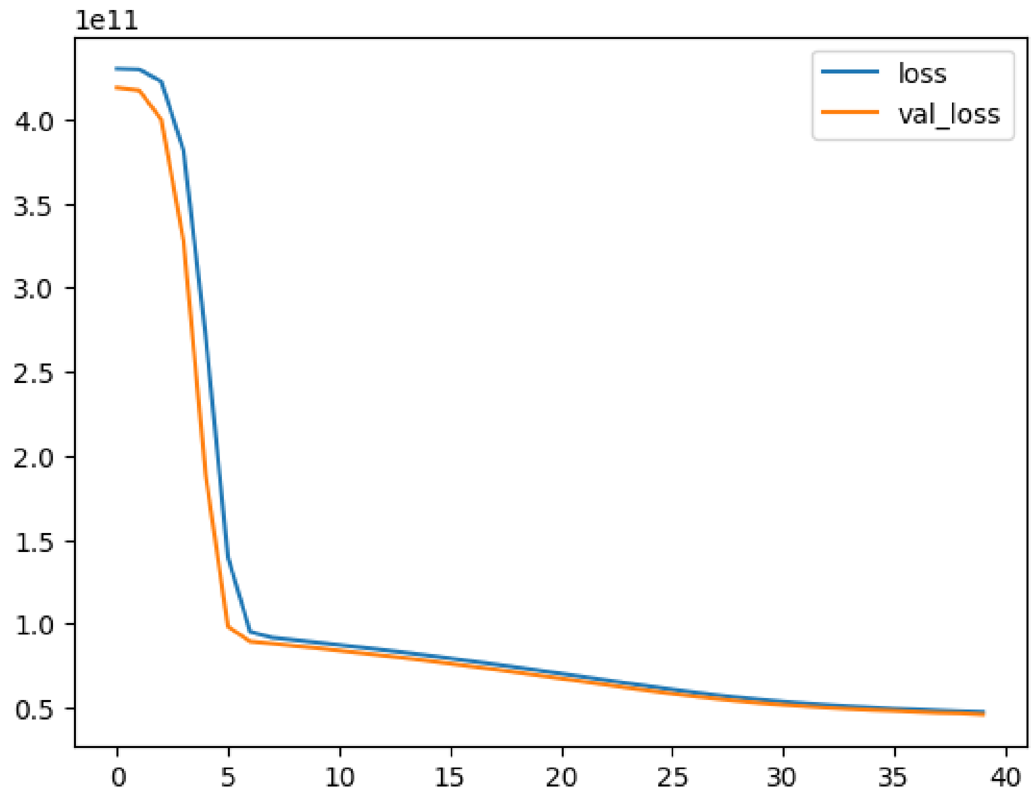
```
119/119 [==============================] - 0s 2ms/step - loss: 59250962432.0000
- val_loss: 56911044608.0000
Epoch 28/40
119/119 [==============================] - 0s 2ms/step - loss: 57630973952.0000
- val_loss: 55405420544.0000
Epoch 29/40
119/119 [==============================] - 0s 2ms/step - loss: 56191746048.0000
- val_loss: 54074175488.0000
Epoch 30/40
119/119 [==============================] - 0s 2ms/step - loss: 54857224192.0000
- val_loss: 52855857152.0000
Epoch 31/40
119/119 [==============================] - 0s 2ms/step - loss: 53691281408.0000
- val_loss: 51818115072.0000
Epoch 32/40
119/119 [==============================] - 0s 2ms/step - loss: 52659077120.0000
- val_loss: 50878496768.0000
Epoch 33/40
119/119 [==============================] - 0s 3ms/step - loss: 51745894400.0000
- val_loss: 50068185088.0000
Epoch 34/40
119/119 [==============================] - 0s 2ms/step - loss: 50916413440.0000
- val_loss: 49338159104.0000
Epoch 35/40
119/119 [==============================] - 0s 2ms/step - loss: 50248773632.0000
- val_loss: 48698892288.0000
Epoch 36/40
119/119 [==============================] - 0s 2ms/step - loss: 49509707776.0000
- val_loss: 48246181888.0000
Epoch 37/40
119/119 [==============================] - 0s 2ms/step - loss: 49016295424.0000
- val_loss: 47633457152.0000
Epoch 38/40
119/119 [==============================] - 0s 2ms/step - loss: 48410259456.0000
- val_loss: 47078539264.0000
Epoch 39/40
119/119 [==============================] - 0s 2ms/step - loss: 47886741504.0000
- val_loss: 46799671296.0000
Epoch 40/40
119/119 [==============================] - 0s 3ms/step - loss: 47501975552.0000
- val_loss: 46111432704.0000
```

[ ]:
```
<keras.src.callbacks.History at 0x7c8c100f7df0>
```

[ ]:
```
losses = pd.DataFrame(model.history.history)
```

[ ]:
```
losses.plot()
```

[ ]: <Axes: >



[ ]:
```
from sklearn.metrics import␣
↪mean_squared_error,mean_absolute_error,explained_variance_score
```

[ ]:
```
predictions = model.predict(X_test)
mean_absolute_error(y_test,predictions)
```

```
203/203 [==============================] - 1s 2ms/step
```

[ ]:
```
128543.12028763382
```

[ ]:
```
single_house = df.drop('price',axis=1).iloc[0]
single_house = scaler.transform(single_house.values.reshape(-1, 17))
model.predict(single_house)
```

```
1/1 [==============================] - 0s 21ms/step

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but MinMaxScaler was fitted with feature names
  warnings.warn(
```

[ ]: array([[255590.75]], dtype=float32)

[ ]:

2nd program

[ ]:
```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

[ ]:
```
# Step 1: Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

[ ]:
```
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255
```

[ ]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

[ ]:
```
# Step 2: Change the hyperparameters of the classification model and analyze␣
↪performance
hyperparameter_combinations = [
    {'filters': 32, 'kernel_size': (3, 3), 'pool_size': (2, 2)},
    {'filters': 64, 'kernel_size': (3, 3), 'pool_size': (2, 2)},
    {'filters': 64, 'kernel_size': (3, 3), 'pool_size': (4, 4)},
    # Add more hyperparameter combinations as needed
]
```

[ ]:
```
for hyperparameters in hyperparameter_combinations:
    model = Sequential()
    model.add(Conv2D(hyperparameters['filters'],
                     kernel_size=hyperparameters['kernel_size'],
                     activation='relu',
                     input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=hyperparameters['pool_size']))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))  # 10 classes for digits 0-9
```

```
[ ]: model.compile(optimizer='adam', loss='binary_crossentropy',␣
     ↪metrics=['accuracy'])
```

```
[ ]: # Train the model
         history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test,␣
     ↪y_test), verbose=0)
```

```
[ ]: # Evaluate the model
         test_loss, test_accuracy = model.evaluate(x_test, y_test)
```

```
[ ]: # Print metrics and hyperparameters
         print(f'Model with Hyperparameters: {hyperparameters}')
         print(f'Test Accuracy: {test_accuracy}')
         print(f'Test Loss: {test_loss}')
```

```
[ ]: # Plot training history
         plt.plot(history.history['accuracy'], label='Training Accuracy')
         plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
         plt.xlabel('Epoch')
         plt.ylabel('Accuracy')
         plt.legend()
         plt.show()
```

3rd program

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

```
[ ]: #import libraries
     from keras.datasets import imdb
     from keras.models import Sequential
     from keras.layers import Dense
     from keras.layers import LSTM
     from keras.layers import Embedding
     from keras.preprocessing import sequence
     np.random.seed(7)
```

```
[ ]: import warnings
     warnings.filterwarnings('ignore')
```

```
[ ]: top_words = 5000
     (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
```

[ ]:
```python
print('Shape of training data: ')
print(X_train.shape)
print(y_train.shape)
print('Shape of test data: ')
print(X_test.shape)
print(y_test.shape)
```

[ ]:
```python
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
```

[ ]:
```python
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout

model = Sequential()
model.add(Embedding(top_words,max_review_length,␣
 ↪input_length=max_review_length))
model.add(LSTM(16, return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(16, return_sequences=True))  # Add another LSTM layer with␣
 ↪return_sequences=True
model.add(Dropout(0.5))
model.add(LSTM(16))  # Add another LSTM layer
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
model.summary()
```

[ ]:
```python
model.compile(optimizer='rmsprop', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=10, batch_size=64)
```

[ ]:
```python
#evaluate the model
test =  model.evaluate(X_test, y_test, verbose=0)
print("Testing Accuracy: %.2f%%" % (test[1]*100))
```

[ ]:
```python
def plot_result(history, epoch):

    epoch_range = range(1, 11)

    plt.plot(epoch_range, history.history['accuracy'], label='Training acc')
    plt.plot(epoch_range, history.history['val_accuracy'], label='Validation␣
 ↪acc')
```

```
    plt.title('Training and validation accuracy')
    plt.xlabel('epochs')
    plt.ylabel('acc')
    plt.legend(loc='upper left')
    plt.savefig('acc.jpg')

    plt.show()

    plt.plot(epoch_range, history.history['loss'], label='Training loss')
    plt.plot(epoch_range, history.history['val_loss'], label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend(loc='upper left')
    plt.savefig('loss.jpg')

    plt.show()
```

[ ]:
```
plot_result(history, 10)
```

[ ]:
```

```

4th program

[ ]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk import NaiveBayesClassifier
from nltk.classify import accuracy
```

[ ]:
```
# Download necessary resources
nltk.download('punkt')
nltk.download('stopwords')
```

[ ]:
```
# Preprocessing function
def preprocess(text):
    tokens = word_tokenize(text.lower())  # Tokenization and lowercase
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.isalpha() and word not in␣
 ↪stop_words]  # Removing stopwords and non-alphabetic tokens
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]  # Stemming
```

```
    return dict([(token, True) for token in tokens])
```

[ ]:
```
# Read dataset from a text file
dataset_file = "dataset.txt"  # Path to your dataset file
dataset = []
with open(dataset_file, 'r') as file:
    for line in file:
        text, label = line.strip().split(",")
        dataset.append((text, label))
```

[ ]:
```
# Preprocess the dataset
preprocessed_dataset = [(preprocess(text), label) for text, label in dataset]
```

[ ]:
```
# Split data into training and testing sets
train_data = preprocessed_dataset[:90]
test_data = preprocessed_dataset[10:]
```

[ ]:
```
# Train the Naive Bayes Classifier
classifier = NaiveBayesClassifier.train(train_data)
```

[ ]:
```
# Test the classifier
print("Accuracy:", accuracy(classifier, test_data))
```

[ ]:
```
# Test a new text segment
text_to_classify = "The product exceeded my expectations"  # You can change
 ↪this text to test different segments
preprocessed_text = preprocess(text_to_classify)
print("Classification:", classifier.classify(preprocessed_text))
```

5th program

[ ]:
```
! python -m ipykernel install --user --name gan
```

[ ]:
```
import torch
from torch import nn

import math
import matplotlib.pyplot as plt
```

[ ]:
```
torch.manual_seed(111)
```

```
[ ]: train_data_length = 1024
train_data = torch.zeros((train_data_length, 2))
train_data[:, 0] = 2 * math.pi * torch.rand(train_data_length)
train_data[:, 1] = torch.sin(train_data[:, 0])
train_labels = torch.zeros(train_data_length)
train_set = [
    (train_data[i], train_labels[i]) for i in range(train_data_length)
]
```

```
[ ]: plt.plot(train_data[:, 0], train_data[:, 1], ".")
```

```
[ ]: batch_size = 32
train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True
)
```

```
[ ]: class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        output = self.model(x)
        return output
```

```
[ ]: discriminator = Discriminator()
```

```
[ ]: class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 16),
            nn.ReLU(),
            nn.Linear(16, 32),
            nn.ReLU(),
```

```
            nn.Linear(32, 2),
        )

    def forward(self, x):
        output = self.model(x)
        return output

generator = Generator()
```

[ ]:
```
lr = 0.001
num_epochs = 300
loss_function = nn.BCELoss()
```

[ ]:
```
optimizer_discriminator = torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)
```

[ ]:
```
for epoch in range(num_epochs):
    for n, (real_samples, _) in enumerate(train_loader):
        # Data for training the discriminator
        real_samples_labels = torch.ones((batch_size, 1))
        latent_space_samples = torch.randn((batch_size, 2))
        generated_samples = generator(latent_space_samples)
        generated_samples_labels = torch.zeros((batch_size, 1))
        all_samples = torch.cat((real_samples, generated_samples))
        all_samples_labels = torch.cat(
            (real_samples_labels, generated_samples_labels)
        )

        # Training the discriminator
        discriminator.zero_grad()
        output_discriminator = discriminator(all_samples)
        loss_discriminator = loss_function(
            output_discriminator, all_samples_labels)
        loss_discriminator.backward()
        optimizer_discriminator.step()

        # Data for training the generator
        latent_space_samples = torch.randn((batch_size, 2))

        # Training the generator
        generator.zero_grad()
        generated_samples = generator(latent_space_samples)
        output_discriminator_generated = discriminator(generated_samples)
        loss_generator = loss_function(
            output_discriminator_generated, real_samples_labels
        )
        loss_generator.backward()
```

```
        optimizer_generator.step()

        # Show loss
        if epoch % 10 == 0 and n == batch_size - 1:
            print(f"Epoch: {epoch} Loss D.: {loss_discriminator}")
            print(f"Epoch: {epoch} Loss G.: {loss_generator}")
```

[ ]:
```
latent_space_samples = torch.randn(100, 2)
generated_samples = generator(latent_space_samples)
```

[ ]:
```
generated_samples = generated_samples.detach()
plt.plot(generated_samples[:, 0], generated_samples[:, 1], ".")
```

[ ]:
```

```