ML_HOMEWORK_6

ELIZABETH AMEKE

662055975

Question_1

(a) Calculate the first two principal components of the Scikit-learn / UCI ML repository Wine Dataset using Scikit's PCA algorithm.
(b) What is the explained variance ratio of the first two principal components?
(c) Use the first two principal components to train a SVM classifier to classify the wines into three classes. Use a 60%-40% split for the train set and the test set.
(d) Compare the accuracy, precision and recall of the SVM classifier with that of a decision tree classifier that classifies the wines in to three classes using the original features. Use the same 60%-40% split for the train set and the test set for the decision tree classifier and use max_depth=3.
(e) Plot the decision tree classifier.

```
#(a)
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Wine dataset
data = load_wine()
X = data.data

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA with two components on the standardized data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Print the first two principal components
print("First principal component:", pca.components_[0])
print("Second principal component:", pca.components_[1])
```

```
First principal component: [ 0.1443294  -0.24518758 -0.00205106 -0.23932041  0.14199204  0.39466085
  0.4229343  -0.2985331   0.31342949 -0.0886167   0.29671456  0.37616741
  0.28675223]
Second principal component: [-0.48365155 -0.22493093 -0.31606881  0.0105905  -0.299634   -0.06503951
  0.00335981 -0.02877949 -0.03930172 -0.52999567  0.27923515  0.16449619
 -0.36490283]
```

.

```
#(b)
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Wine dataset
data = load_wine()
X = data.data

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA with two components on the standardized data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Get the explained variance ratio of the first two principal components
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained variance ratio of the first two principal components:", explained_variance_ratio)
```

```
Explained variance ratio of the first two principal components: [0.36198848 0.1920749 ]
```

Explained variance ratio of the first two principal components: (0.36198848, 0.1920749)

.

```
#(c)
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Wine dataset
data = load_wine()
X = data.data
y = data.target

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA with two components on the standardized data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.4, random_state=42)

# Train a multiclass SVM classifier
svm_classifier = SVC(kernel='linear', random_state=42, decision_function_shape='ovr')
svm_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9722222222222222
```

Accuracy: 0.9722222222222222

.

```
#(d)
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Load the Wine dataset
data = load_wine()
X = data.data
y = data.target

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Train SVM classifier
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm_classifier = SVC(kernel='linear', random_state=42, decision_function_shape='ovr')
svm_classifier.fit(X_train_scaled, y_train)

# Make predictions using SVM classifier
y_pred_svm = svm_classifier.predict(X_test_scaled)

# Calculate accuracy, precision, and recall for SVM classifier
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')

print("SVM Classifier:")
print("Accuracy:", accuracy_svm)
print("Precision:", precision_svm)
print("Recall:", recall_svm)

# Train Decision Tree classifier
dt_classifier = DecisionTreeClassifier(max_depth=3, random_state=42)
dt_classifier.fit(X_train, y_train)

# Make predictions using Decision Tree classifier
y_pred_dt = dt_classifier.predict(X_test)

# Calculate accuracy, precision, and recall for Decision Tree classifier
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')

print("\nDecision Tree Classifier:")
print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
```

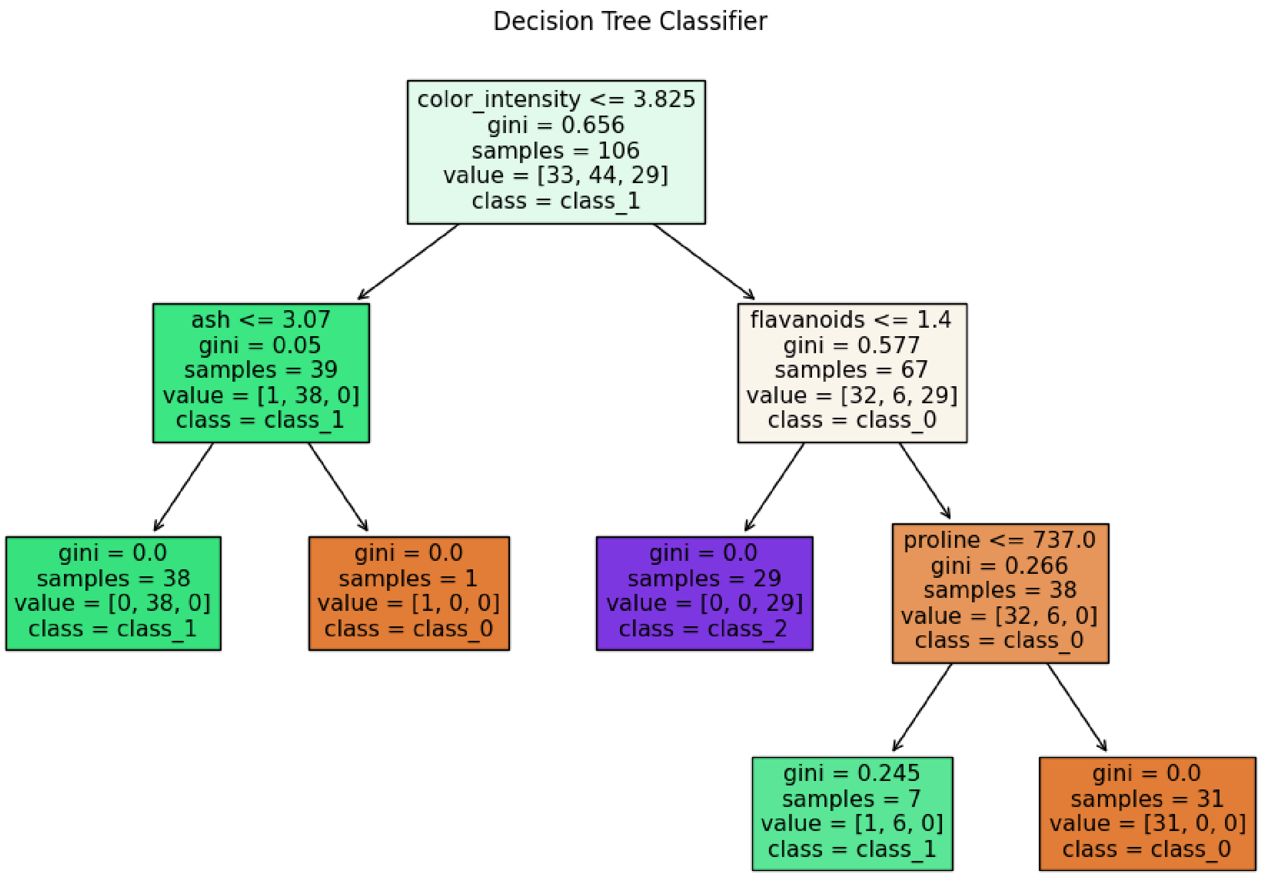
```
SVM Classifier:
Accuracy: 0.9861111111111112
Precision: 0.9868055555555555
Recall: 0.9861111111111112
```

```
Decision Tree Classifier:
Accuracy: 0.9305555555555556
Precision: 0.94140625
Recall: 0.9305555555555556
```

Judging from the values obtained, the SVM has higher metric values for accuracy, precision, and recall as compared to that of the Decision Tree.

.

```
#(e)
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Plot Decision Tree classifier
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, feature_names=data.feature_names, class_names=data.target_names, filled=True)
plt.title("Decision Tree Classifier")
plt.show()
```



.

.

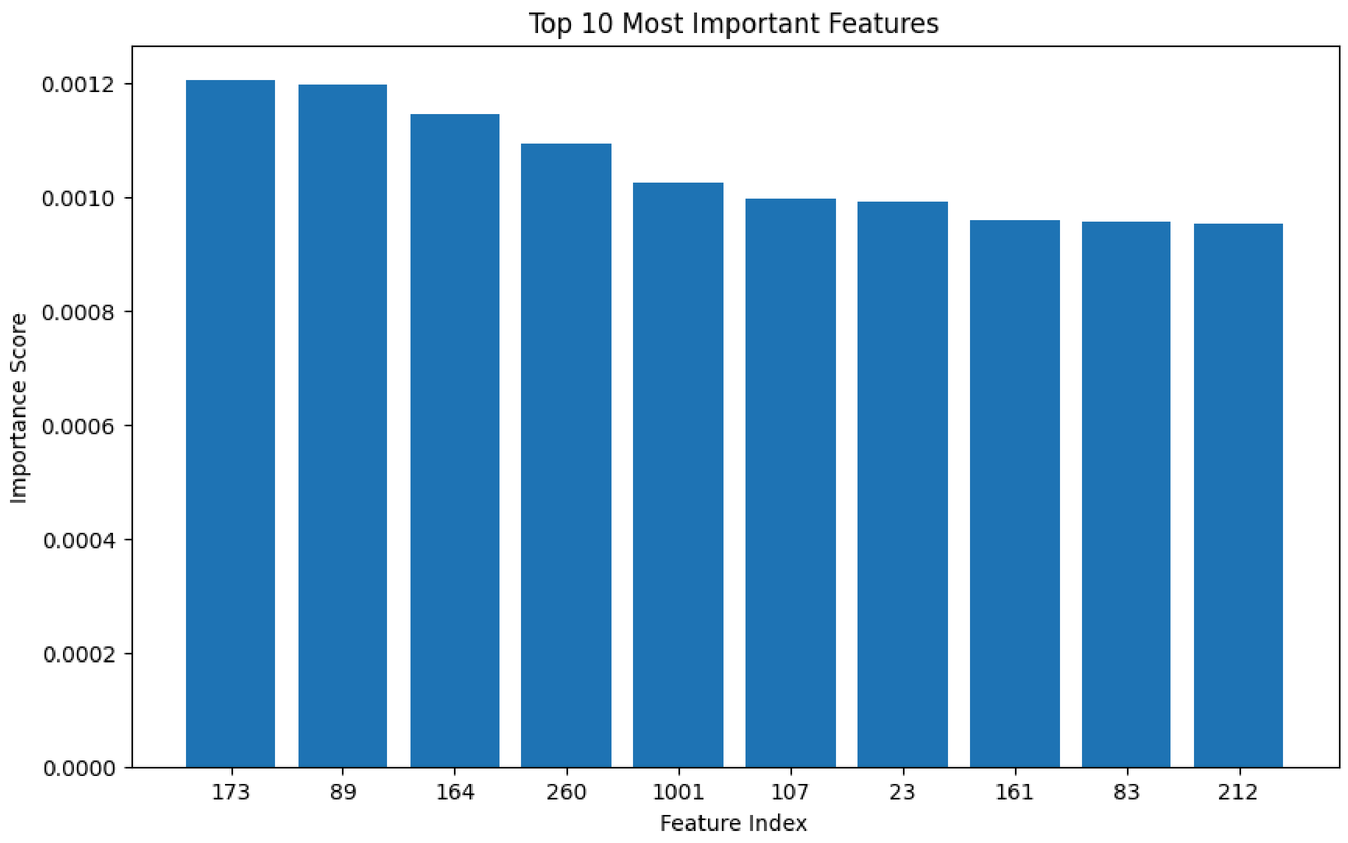
Question_2

Calculate the feature importance of the pixels in the images of the CIFAR-10 dataset (combine train and test sets) using a random forest classifier.

```
import numpy as np
from tensorflow.keras.datasets import cifar10
from sklearn.ensemble import RandomForestClassifier

# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Combine train and test sets
X_combined = np.concatenate((X_train, X_test))
y_combined = np.concatenate((y_train, y_test)).ravel()  # ravel() to convert to 1D array

# Reshape the data to 2D (flatten images)
X_combined_flattened = X_combined.reshape(X_combined.shape[0], -1)

# Create a random forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to the data
rf_classifier.fit(X_combined_flattened, y_combined)

# Get feature importances
feature_importances = rf_classifier.feature_importances_

# Sort feature importances in descending order
sorted_indices = np.argsort(feature_importances)[::-1]

# Print the top 10 most important features
print("Top 10 most important features (pixels):")
for i in range(10):
    print(f"Feature index: {sorted_indices[i]}, Importance: {feature_importances[sorted_indices[i]]}")
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
Top 10 most important features (pixels):
Feature index: 173, Importance: 0.0012051691659010467
Feature index: 89, Importance: 0.001197034768632174
Feature index: 164, Importance: 0.001143843171806588
Feature index: 260, Importance: 0.0010930119456225098
Feature index: 1001, Importance: 0.0010246100268692096
Feature index: 107, Importance: 0.000997301829705567
Feature index: 23, Importance: 0.0009905668214285513
Feature index: 161, Importance: 0.0009575925682724734
Feature index: 83, Importance: 0.0009556702012220343
Feature index: 212, Importance: 0.0009521951534418141
```

```
#Visualizing the data
import matplotlib.pyplot as plt

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(10), feature_importances[sorted_indices[:10]], align='center')
plt.xticks(range(10), sorted_indices[:10])
plt.xlabel('Feature Index')
plt.ylabel('Importance Score')
plt.title('Top 10 Most Important Features')
plt.show()
```



.

.

Question_3

Import the surface temperature data, surface_temp.npy, for a small sphere from LMS. The temperature data is given for 1000 timesteps. Predict the temperatures for the next 10 timesteps using a recurrent neural network.

```python
import numpy as np
import tensorflow as tf
from tensorflow import keras

# Load surface temperature data
surface_temp_data = np.load("/content/surface_temp.npy")

# Reshape the data to have two dimensions: (number of timesteps, number of features)
surface_temp_data = surface_temp_data.reshape(-1, 1)

# Define the number of timesteps and the number of features (temperature readings)
n_timesteps = surface_temp_data.shape[0]
n_features = surface_temp_data.shape[1]

# Define the number of timesteps to predict in the future
n_predictions = 10

# Prepare the data for training
X_train = surface_temp_data[:-n_predictions]
y_train = surface_temp_data[n_predictions:]

# Reshape the data to fit the RNN input shape (samples, timesteps, features)
X_train = X_train.reshape(1, -1, n_features)
y_train = y_train.reshape(1, -1, n_features)

# Define the RNN model
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, n_features]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(n_features))  # Apply Dense layer to each timestep
])

# Compile the model
model.compile(loss="mse", optimizer="adam")

# Train the model with increased epochs
history = model.fit(X_train, y_train, epochs=100)

# Make predictions for the next 10 timesteps
X_test = surface_temp_data[-n_predictions:].reshape(1, -1, n_features)
predictions = model.predict(X_test)

# Print predicted temperatures for the next 10 timesteps
print("Predicted temperatures for the next 10 timesteps:")
print(predictions.flatten())
```

```
Epoch 81/100
1/1 [==============================] - 0s 269ms/step - loss: 134853.4375
Epoch 82/100
1/1 [==============================] - 0s 279ms/step - loss: 134824.2500
Epoch 83/100
1/1 [==============================] - 0s 280ms/step - loss: 134795.0312
Epoch 84/100
1/1 [==============================] - 0s 269ms/step - loss: 134765.7812
Epoch 85/100
1/1 [==============================] - 0s 273ms/step - loss: 134736.4375
Epoch 86/100
1/1 [==============================] - 0s 282ms/step - loss: 134707.0312
Epoch 87/100
1/1 [==============================] - 0s 278ms/step - loss: 134677.5312
Epoch 88/100
1/1 [==============================] - 0s 273ms/step - loss: 134647.9219
Epoch 89/100
1/1 [==============================] - 0s 288ms/step - loss: 134618.2500
Epoch 90/100
1/1 [==============================] - 1s 542ms/step - loss: 134588.4219
Epoch 91/100
1/1 [==============================] - 0s 491ms/step - loss: 134558.4844
Epoch 92/100
1/1 [==============================] - 1s 534ms/step - loss: 134528.4688
Epoch 93/100
1/1 [==============================] - 1s 516ms/step - loss: 134498.3750
Epoch 94/100
1/1 [==============================] - 0s 450ms/step - loss: 134468.2031
Epoch 95/100
1/1 [==============================] - 0s 274ms/step - loss: 134438.0000
Epoch 96/100
1/1 [==============================] - 0s 291ms/step - loss: 134407.7969
Epoch 97/100
1/1 [==============================] - 0s 289ms/step - loss: 134377.6094
Epoch 98/100
1/1 [==============================] - 0s 273ms/step - loss: 134347.5000
Epoch 99/100
1/1 [==============================] - 0s 266ms/step - loss: 134317.5000
Epoch 100/100
1/1 [==============================] - 0s 291ms/step - loss: 134287.6406
1/1 [==============================] - 0s 442ms/step
Predicted temperatures for the next 10 timesteps:
[6.184396  6.835847  6.9132404 6.9149876 6.916648  6.915823  6.9162436
 6.916071  6.9161396 6.916112 ]
```

Predicted temperatures for the next 10 timesteps: [2.8620827 3.0736597 3.0565202 2.9984803 3.15436 3.23128 3.0521154 3.1545632 3.1435492 3.1309738]

.

```
import numpy as np

# Load surface temperature data
surface_temp_data = np.load("/content/surface_temp.npy")

# Print the surface temperature data
print("Surface temperature data:")
print(surface_temp_data)
```

```
 373.00780778 373.21572136 373.78699434 373.67840623 373.42325627
 373.27833984 373.70772663 373.56705856 374.18916968 373.72028711
 373.69194628 373.79698212 373.70286643 373.26821272 373.14855165
 373.33300374 373.41428092 373.5776021  373.53631026 373.59359563
 374.25085333 373.78687238 373.55528303 373.38205916 373.35694907
 373.58690988 373.77346035 373.47739981 373.29521718 373.19600435
 373.53552225 374.05623973 373.8540419  373.00770317 373.08562184
 373.58266353 373.84988686 373.2029191  371.3789552  370.00163734
 370.40565321 372.24949707 373.79347373 374.05434063 373.04683465
 372.99596136 373.05415831 373.97585139 373.66647851 373.22097278
 373.18495763 373.41381622 373.86059878 373.71879861 373.43687639
 373.11985799 373.67937549 373.96431513 373.57365343 373.27488099
 373.53661318 373.64070026 373.39692197 373.67735322 373.51579957
 373.41148474 373.70489511 373.80394925 373.31127385 373.71930899
 373.63146165 373.64404916 373.60355748 373.13226002 373.19440831
 373.27337225 373.87669829 373.25200096 373.23725973 373.30585513
 373.79917698 374.07491561 373.61207186 373.26360001 373.10281864
 373.71325898 373.85943226 373.16758878 370.91157768 370.19694412
 370.9718647  372.59841327 373.59680781 373.77387195 373.27003131
 373.21299923 373.58718311 373.74432456 373.40936364 373.73936181
 373.42259639 373.4332165  373.7245395  373.76129859 373.09595832
 372.98933894 373.69983744 373.56202182 373.84050577 373.79792197
 373.75018622 373.49758669 373.50596033 373.20660648 373.59665607
 373.37755813 373.51802524 373.68383758 373.34518061 373.62556784
 373.94673949 373.34845767 373.45003991 372.7574692  373.35151219
 373.43145235 373.75226762 373.44820259 373.23792921 373.23827543
 374.01626741 373.89843992 373.67160374 373.01926808 373.27092446
 373.80659828 374.0152379  373.04679782 371.12836088 370.27617258
 370.6982978  372.35594563 373.74839268 373.90639292 373.24643534
 372.92635997 373.47318059 374.11983138 373.55676068 373.48043608
 373.22355833 373.5694899  373.65517868 373.980168   373.4644566
 373.20021373 373.87735884 374.10201594 374.00154152 373.75749225
 373.40961846 373.25768136 373.78603389 373.67615895 373.42464421
 373.5023727  373.85016743 373.680439   373.43280911 373.49613313
 373.9211827  373.62175529 373.54188448 373.14645342 373.3557284
 374.19182929 373.67102792 373.47755617 373.21666739 373.25315774
 373.72786298 374.35348074 373.61810622 373.07600384 373.46230122
 373.82571048 373.60527173 372.59830386 370.87755827 369.78033121
 370.93727155 372.54108027 373.63580614 374.05988931 373.38733462
 373.04644041 373.44650297 373.74797573 373.55634755 373.37808843
 373.16188713 373.63453927 373.51288261 373.85167422 373.14930018
 373.26959312 373.29529926 374.11543953 373.92298043 373.80972279
 373.74817923 373.48040453 373.73817329 373.56040611 373.84541956
 373.55042552 373.41309286 373.9297378  373.39707053 373.78555655
 373.86004841 374.09604544 373.11201594 373.36629909 373.3545765
 373.91219162 373.5925467  373.27808937 373.31665023 373.98463348
 374.24335475 373.87401718 373.26464393 372.84744545 373.06179818
 373.90680004 374.12058828 373.05539028 370.92814003 370.21957238
 370.64839601 372.75197045 373.72971917 373.96377416 373.12933349
 373.26399868 373.38228056 373.84576488 373.57028062 373.14062216
 372.89543933 373.55429448 373.22831221 373.74332305 373.20636042
 373.11750323 373.87633648 373.70848333 373.65709418 373.40781485
 373.44481666 373.52747408 373.44045289 373.40815145 373.37827168
 373.71604213 373.2026854  373.62485611 373.5841981  373.53452032
 373.70541283 373.72134386 373.81302583 373.63063346 373.09935146
 373.37213136 373.37061798 373.45716198 373.38176781 373.64388132
 373.70495315 374.00151051 373.73193525 373.09785716 373.27834991
 374.18606828 374.00228441 372.33366869 370.95345815 369.85047826]
```

```python
import numpy as np

# Load surface temperature data
surface_temp_data = np.load("/content/surface_temp.npy")
```

```
# Generate numbers from 1 to 1000
numbers = np.arange(1, 1001)

# Create X and Y
X = numbers.reshape(-1, 1)  # Reshape to have a single column
Y = surface_temp_data.reshape(-1, 1)  # Reshape to have a single column

# Stack X and Y vertically
data = np.vstack((X, Y)).T

# Print the first few rows
print("First few rows of X and Y:")
print(data[:5])
```

```
First few rows of X and Y:
[[  1.           2.           3.         ... 372.33366869 370.95345815
  369.85047826]]
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Load surface temperature data
surface_temp_data = np.load("/content/surface_temp.npy")

# Generate numbers from 1 to 1000
numbers = np.arange(1, 1001)

# Create X and Y
X = numbers.reshape(-1, 1)  # Reshape to have a single column
Y = surface_temp_data.reshape(-1, 1)  # Reshape to have a single column

# Split data into training and testing sets
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

# Reshape input to be 3D [samples, timesteps, 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]))

# Define the LSTM model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(1, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# Fit the model
model.fit(X_train, Y_train, epochs=300, batch_size=1, verbose=2)

# Predict temperatures for the next 10 timesteps
future_timesteps = 10
future_X = np.arange(1001, 1001 + future_timesteps).reshape(-1, 1)
future_X = np.reshape(future_X, (future_X.shape[0], 1, future_X.shape[1]))
future_Y = model.predict(future_X)

# Print the predicted temperatures
print("Predicted temperatures for the next 10 timesteps:")
print(future_Y)
```

```
Epoch 286/300
800/800 - 2s - loss: 7.2018 - 2s/epoch - 2ms/step
Epoch 287/300
800/800 - 2s - loss: 12.4301 - 2s/epoch - 2ms/step
Epoch 288/300
800/800 - 1s - loss: 27.7461 - 1s/epoch - 2ms/step
Epoch 289/300
800/800 - 1s - loss: 311.5132 - 1s/epoch - 2ms/step
Epoch 290/300
800/800 - 1s - loss: 2.7287 - 1s/epoch - 2ms/step
Epoch 291/300
800/800 - 1s - loss: 4.1318 - 1s/epoch - 2ms/step
Epoch 292/300
800/800 - 1s - loss: 29.1877 - 1s/epoch - 2ms/step
Epoch 293/300
800/800 - 1s - loss: 10.6090 - 1s/epoch - 2ms/step
Epoch 294/300
800/800 - 1s - loss: 19.1234 - 1s/epoch - 2ms/step
Epoch 295/300
800/800 - 2s - loss: 25.4607 - 2s/epoch - 3ms/step
Epoch 296/300
800/800 - 1s - loss: 6.8856 - 1s/epoch - 2ms/step
Epoch 297/300
800/800 - 1s - loss: 8.8172 - 1s/epoch - 2ms/step
Epoch 298/300
800/800 - 1s - loss: 48.0721 - 1s/epoch - 2ms/step
Epoch 299/300
800/800 - 1s - loss: 13.5704 - 1s/epoch - 2ms/step
Epoch 300/300
800/800 - 1s - loss: 5.2360 - 1s/epoch - 2ms/step
1/1 [==============================] - 0s 175ms/step
Predicted temperatures for the next 10 timesteps:
[[386.39767]
 [386.55823]
 [386.7188 ]
 [386.8794 ]
 [387.03992]
 [387.2005 ]
 [387.36108]
 [387.52164]
 [387.6822 ]
 [387.84277]]
```

Predicted temperatures for the next 10 timesteps: [[386.39767] [386.55823] [386.7188] [386.8794] [387.03992] [387.2005] [387.36108] [387.52164] [387.6822] [387.84277]]

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Load surface temperature data
surface_temp_data = np.load("/content/surface_temp.npy")

# Generate numbers from 1 to 1000
numbers = np.arange(1, 1001)

# Create X and Y
X = numbers.reshape(-1, 1)  # Reshape to have a single column
Y = surface_temp_data.reshape(-1, 1)  # Reshape to have a single column

# Stack X and Y vertically
data = np.vstack((X, Y)).T

# Split data into training and testing sets
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

# Reshape input to be 3D [samples, timesteps, 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]))

# Define the LSTM model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(1, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# Fit the model
model.fit(X_train, Y_train, epochs=300, batch_size=1, verbose=2)

# Predict temperatures from 1001 to 1010
future_timesteps = 10
future_X = np.arange(1001, 1001 + future_timesteps).reshape(-1, 1)
future_Y = model.predict(future_X)

# Print the predicted temperatures
print("Predicted temperatures from 1001 to 1010:")
print(future_Y)
```

```
Epoch 285/300
800/800 - 1s - loss: 5.3922 - 1s/epoch - 2ms/step
Epoch 286/300
800/800 - 1s - loss: 14.0606 - 1s/epoch - 2ms/step
Epoch 287/300
800/800 - 1s - loss: 10.1142 - 1s/epoch - 2ms/step
Epoch 288/300
800/800 - 2s - loss: 8.0212 - 2s/epoch - 3ms/step
Epoch 289/300
800/800 - 1s - loss: 15.7697 - 1s/epoch - 2ms/step
Epoch 290/300
800/800 - 1s - loss: 70.6601 - 1s/epoch - 2ms/step
Epoch 291/300
800/800 - 1s - loss: 1.4089 - 1s/epoch - 2ms/step
Epoch 292/300
800/800 - 1s - loss: 4.4463 - 1s/epoch - 2ms/step
Epoch 293/300
800/800 - 1s - loss: 5.5078 - 1s/epoch - 2ms/step
Epoch 294/300
800/800 - 1s - loss: 10.7523 - 1s/epoch - 2ms/step
Epoch 295/300
800/800 - 1s - loss: 13.2319 - 1s/epoch - 2ms/step
Epoch 296/300
800/800 - 2s - loss: 19.1776 - 2s/epoch - 2ms/step
Epoch 297/300
800/800 - 2s - loss: 4.4960 - 2s/epoch - 2ms/step
Epoch 298/300
```

Predicted temperatures from 1001 to 1010: [[379.2252] [379.25662] [379.28806] [379.31946] [379.35086] [379.38232] [379.41373] [379.44516] [379.4766] [379.508]]

```
800/800 - 1s - loss: 3.2285 - 1s/epoch - 2ms/step
```

.

```
[[379.2252 ]
```

.

```
 [379.31946]
```

'

```
 [379.41373]
```

Double-click (or enter) to edit

```
 [379.508 ]]
```

Double-click (or enter) to edit

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.