The University of Faisalabad

Lab Manual

Subject:	**Machine Learning (AI-414)**
Name:	**M.Hamza Rizwan**
Reg:	**2023-BS-AI-005**
Submitted to:	**Mr. Saeed**

DEPARTMENT OF COMPUTATIONAL SCIENCES

FACULTY OF INFORMATION TECHNOLOGY

The University of Faisalabad

Contents

Title: household power consumption (Regression)

Dataset Description

The dataset used in this project is the Individual Household Electric Power Consumption Data Set, which records electric power usage in a single household over nearly four years (from December 2006 to November 2010). It contains over 2 million measurements gathered in one-minute intervals. Each record includes the date, time, and various measurements of electric power consumption.

Project Description

This project aims to predict household power consumption by analyzing historical electrical data to understand and anticipate energy usage patterns. The core focus is on forecasting Global Active Power, which represents the total electrical power consumed by a household. Accurate prediction of this metric is essential for optimizing energy management, improving sustainability, and supporting smarter decision-making both at the consumer and utility levels.

Understanding household power consumption is vital in today's world where energy resources are finite and demand continues to rise. By accurately predicting how much power a household will use at different times, energy providers can better manage supply, reduce wastage, and prevent blackouts or overloads on the grid. For consumers, this insight can lead to smarter use of appliances and potentially lower energy bills through better load balancing and scheduling.

The project focuses on leveraging the rich temporal information inherent in power consumption data recognizing that usage varies not only by minute-to-minute activity but also by broader patterns like time of day, day of the week, and seasonal changes. Capturing these patterns allows for more precise forecasting, which can be a foundation for energy-saving programs, demand-response strategies, and the development of smart home technologies that adapt automatically to consumption habits.

Moreover, the ability to predict power consumption with a high degree of accuracy supports larger sustainability goals by facilitating more efficient energy distribution and reducing environmental impact. As the energy sector moves toward greener, smarter grids, predictive models like the one developed in this project become indispensable tools for enabling a transition to cleaner and more reliable energy systems.

In sum, this project not only addresses the challenge of forecasting household power needs but also contributes toward broader objectives of energy efficiency, cost savings, and environmental sustainability key concerns in the context of global energy consumption and climate change.

Objectives:

1. **Data Ingestion and Understanding:**

- Load the dataset using pandas and gain insights through basic exploration (head, tail, info, describe).
- Identify the shape and structure of the dataset.

2. **Data Cleaning and Preprocessing:**

- Convert date fields to proper datetime objects to handle temporal aspects accurately.
- Remove missing and invalid data to ensure clean, reliable input for the model.
- Replace placeholders ('?') with NaNs and drop incomplete rows.

3. **Feature Engineering:**

- Extract features like Day**,** Month**,** Year**,** and Weekday to capture periodic patterns that influence power consumption.
- Drop unnecessary columns (Date and Time after feature extraction) to avoid redundancy and potential noise.

4. **Model Development:**

- Use a Random Forest Regressor to build a predictive model.
- Split the dataset into training and testing sets to validate model generalizability.
- Train the model on the training set to capture relationships between features and the target variable.

5. **Model Evaluation:**

- Assess model performance using key metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score**.**
- Visualize and interpret the most accurate and least accurate predictions to understand where the model performs best and where it [may need refinement.

6. C**onsumption Insights:**

- Calculate the average household power consumption in both kilowatts and watts for practical understanding.

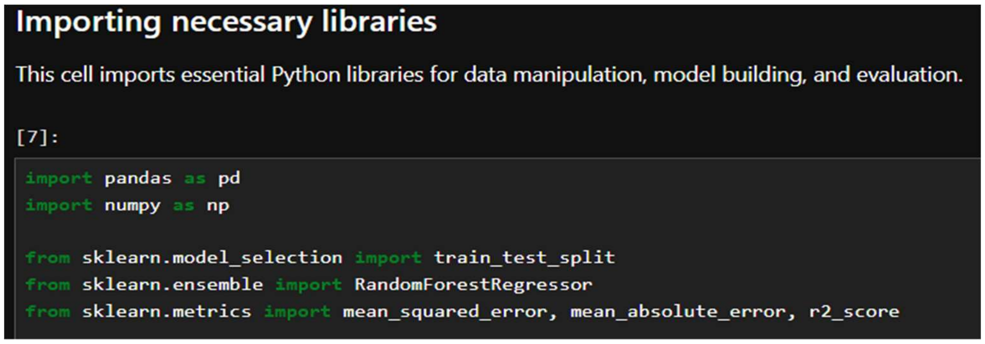
Goals:

1. **Predict Household Energy Usage**
   Develop a machine learning model to forecast household energy consumption based on historical data.
2. **Identify Key Usage Patterns**
   Analyze model outputs to uncover patterns and factors driving energy consumption.
3. **Ensure Data Quality**
   Perform thorough data cleaning and feature engineering for accurate predictions.
4. **Evaluate Model Performance**
   Rigorously assess the model's accuracy and generalizability using appropriate metrics.
5. **Support Energy Efficiency**
   Provide actionable insights and strategies for reducing household energy usage.
6. **Enable Future Improvements**
   Explore incorporating granular data (e.g., temperature, appliance-level data) to further enhance predictions and recommendations.

Project code:

Cell 1:

This cell imports all the necessary Python libraries used throughout the project, including tools for data manipulation (pandas, numpy), machine learning (sklearn), and model evaluation.



Importing necessary libraries

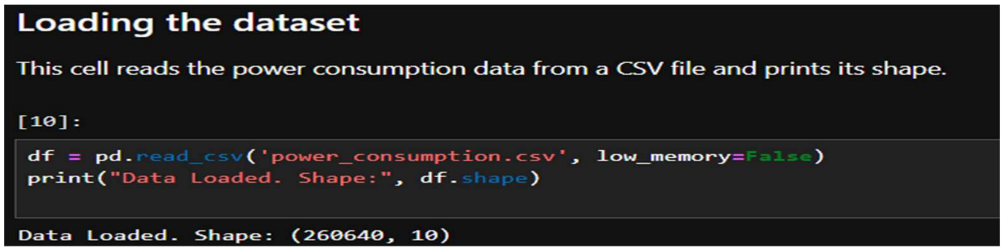
This cell imports essential Python libraries for data manipulation, model building, and evaluation.

```
[7]:
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Cell 2:

Reads the power consumption dataset from a CSV file into a DataFrame and prints its shape (number of rows and columns) to confirm successful loading.



Loading the dataset

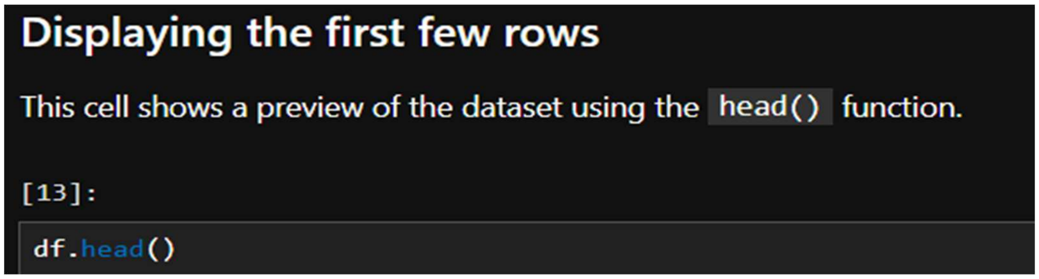
This cell reads the power consumption data from a CSV file and prints its shape.

```
[10]:
df = pd.read_csv('power_consumption.csv', low_memory=False)
print("Data Loaded. Shape:", df.shape)

Data Loaded. Shape: (260640, 10)
```

Cell 3:

Displays the first five rows of the dataset to get an initial look at the structure and values.

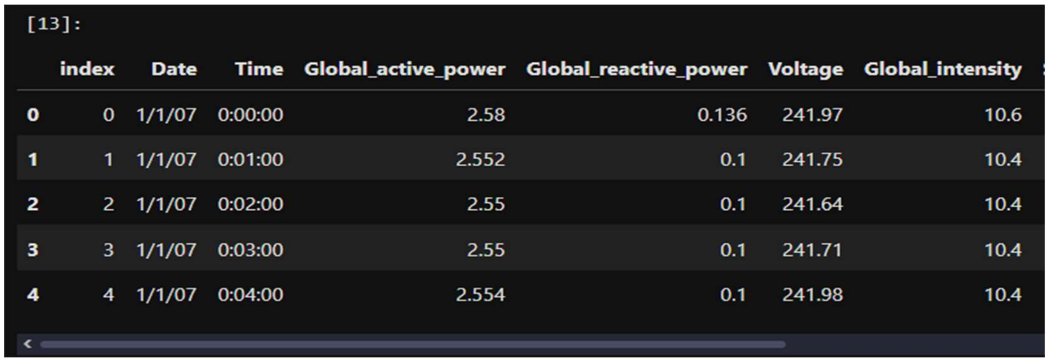


Displaying the first few rows

This cell shows a preview of the dataset using the `head()` function.

```
[13]:
df.head()
```

Output:

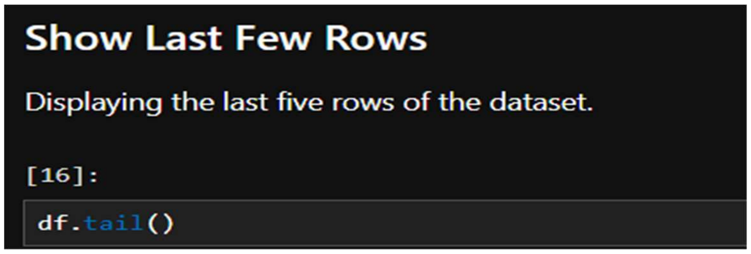


[13]:

	index	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity
0	0	1/1/07	0:00:00	2.58	0.136	241.97	10.6
1	1	1/1/07	0:01:00	2.552	0.1	241.75	10.4
2	2	1/1/07	0:02:00	2.55	0.1	241.64	10.4
3	3	1/1/07	0:03:00	2.55	0.1	241.71	10.4
4	4	1/1/07	0:04:00	2.554	0.1	241.98	10.4

Cell 4:

Shows the last five rows of the dataset to identify potential issues or patterns at the end of the data.

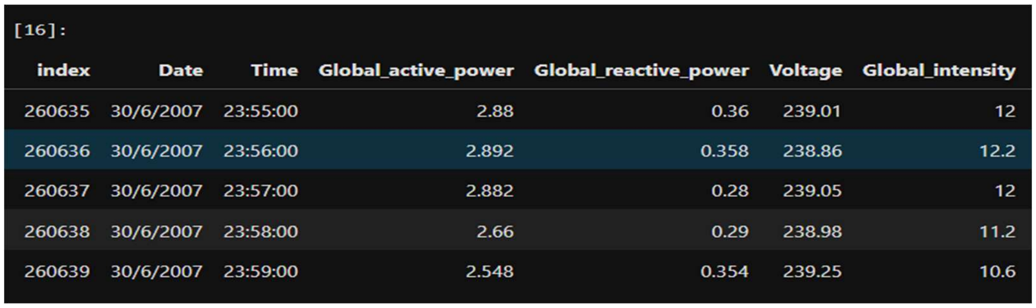


Show Last Few Rows

Displaying the last five rows of the dataset.

```
[16]:
df.tail()
```

Output:

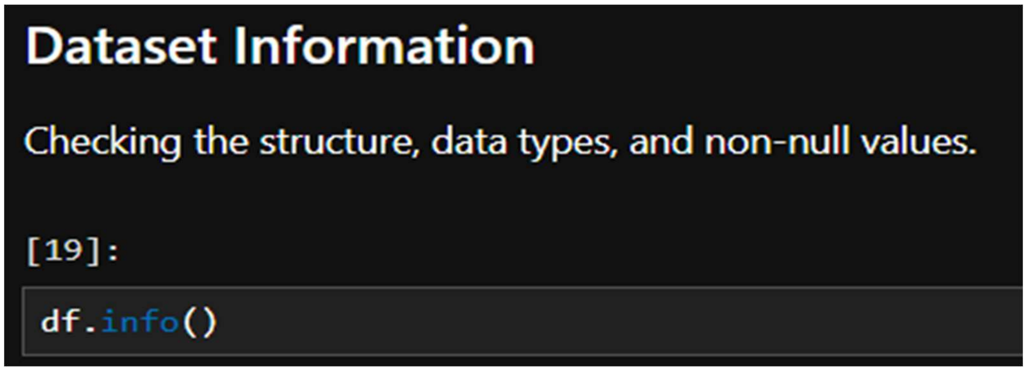


[16]:

index	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity
260635	30/6/2007	23:55:00	2.88	0.36	239.01	12
260636	30/6/2007	23:56:00	2.892	0.358	238.86	12.2
260637	30/6/2007	23:57:00	2.882	0.28	239.05	12
260638	30/6/2007	23:58:00	2.66	0.29	238.98	11.2
260639	30/6/2007	23:59:00	2.548	0.354	239.25	10.6

Cell 5:

Prints detailed information about the dataset, such as column names, data types, and number of non-null values in each column.



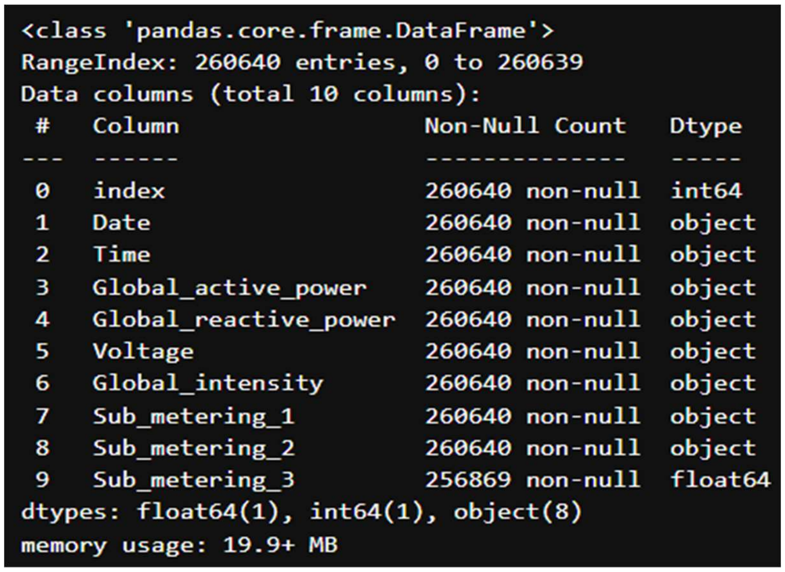
Dataset Information

Checking the structure, data types, and non-null values.

[19]:

```
df.info()
```

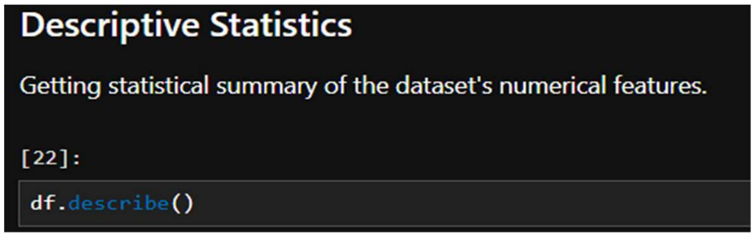
Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260640 entries, 0 to 260639
Data columns (total 10 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   index                  260640 non-null  int64  
 1   Date                   260640 non-null  object 
 2   Time                   260640 non-null  object 
 3   Global_active_power    260640 non-null  object 
 4   Global_reactive_power  260640 non-null  object 
 5   Voltage                260640 non-null  object 
 6   Global_intensity       260640 non-null  object 
 7   Sub_metering_1         260640 non-null  object 
 8   Sub_metering_2         260640 non-null  object 
 9   Sub_metering_3         256869 non-null  float64
dtypes: float64(1), int64(1), object(8)
memory usage: 19.9+ MB
```

Cell 6:

Generates a statistical summary of numerical columns, including count, mean, standard deviation, min, and max values, which helps understand data distribution.



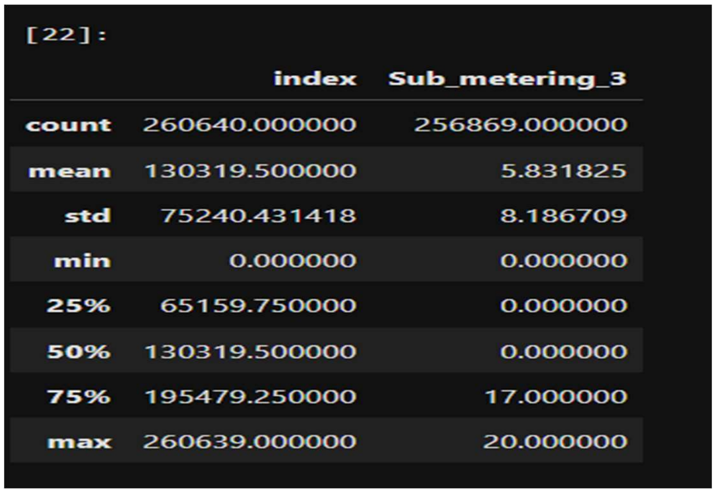
Descriptive Statistics

Getting statistical summary of the dataset's numerical features.

[22]:

```
df.describe()
```

Output:

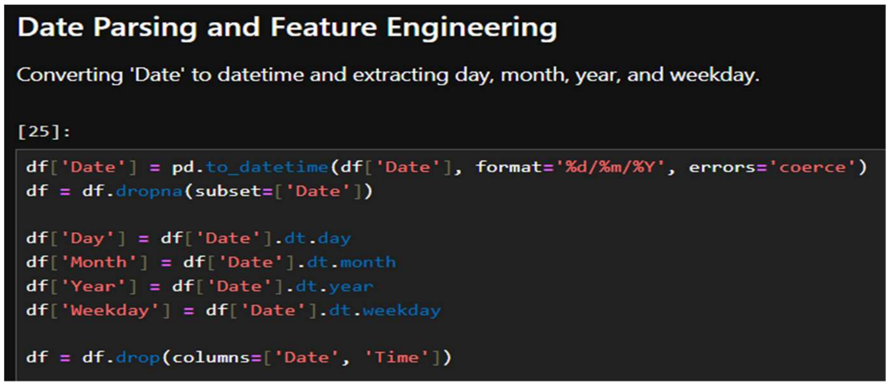


[22]:

	index	Sub_metering_3
count	260640.000000	256869.000000
mean	130319.500000	5.831825
std	75240.431418	8.186709
min	0.000000	0.000000
25%	65159.750000	0.000000
50%	130319.500000	0.000000
75%	195479.250000	17.000000
max	260639.000000	20.000000

Cell 7:

Converts the 'Date' column into datetime format and extracts features like day, month, year, and weekday. Also removes unnecessary columns like 'Date' and 'Time'.



Date Parsing and Feature Engineering

Converting 'Date' to datetime and extracting day, month, year, and weekday.

[25]:

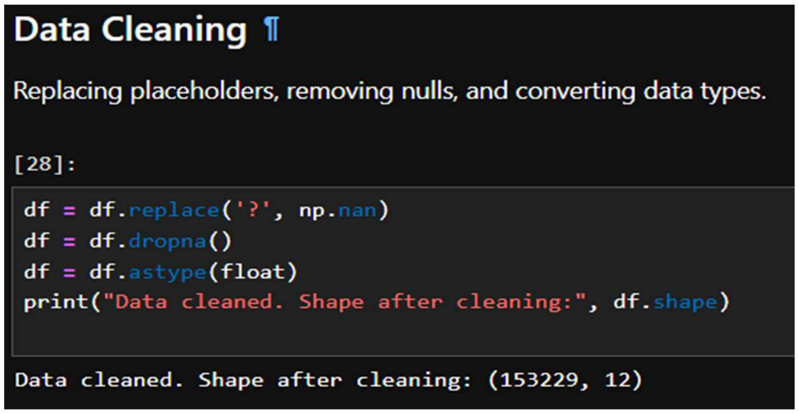
```
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y', errors='coerce')
df = df.dropna(subset=['Date'])

df['Day'] = df['Date'].dt.day
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year
df['Weekday'] = df['Date'].dt.weekday

df = df.drop(columns=['Date', 'Time'])
```

Cell 8:

Replaces missing or invalid values (like '?') with NaN, drops rows with missing values, and converts all columns to numeric types for analysis.



Data Cleaning ¶

Replacing placeholders, removing nulls, and converting data types.

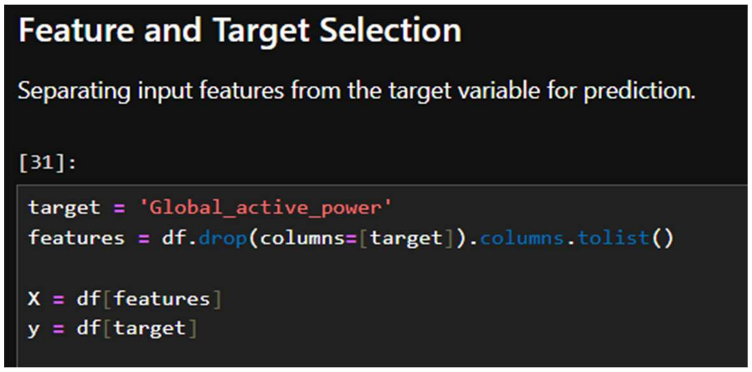
[28]:

```
df = df.replace('?', np.nan)
df = df.dropna()
df = df.astype(float)
print("Data cleaned. Shape after cleaning:", df.shape)
```

```
Data cleaned. Shape after cleaning: (153229, 12)
```

Cell 9:

Defines the target variable (Global_active_power) and separates it from the input features to prepare for training.



Feature and Target Selection

Separating input features from the target variable for prediction.

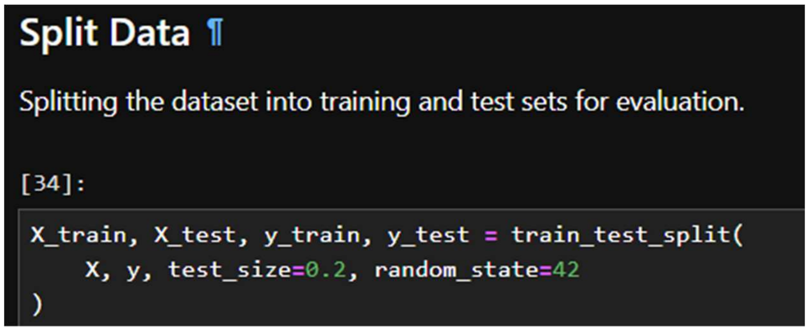
[31]:

```
target = 'Global_active_power'
features = df.drop(columns=[target]).columns.tolist()

X = df[features]
y = df[target]
```

Cell 10:

Splits the data into training and testing sets using an 80/20 ratio to evaluate the model's performance on unseen data.



Split Data ¶

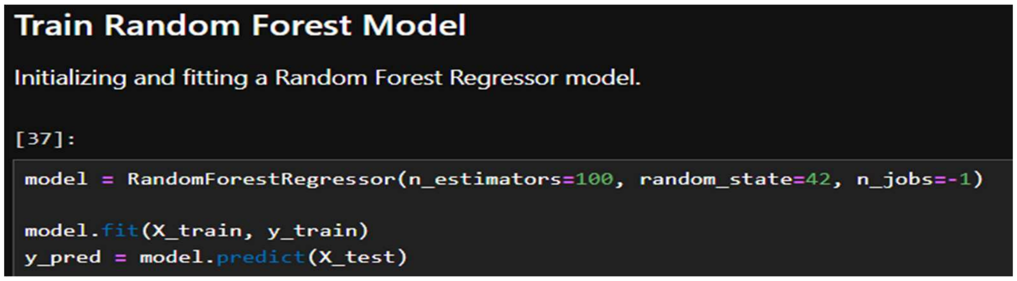
Splitting the dataset into training and test sets for evaluation.

[34]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Cell 11:

Initializes a Random Forest Regressor with 100 trees and fits it to the training data to learn patterns.



Train Random Forest Model

Initializing and fitting a Random Forest Regressor model.

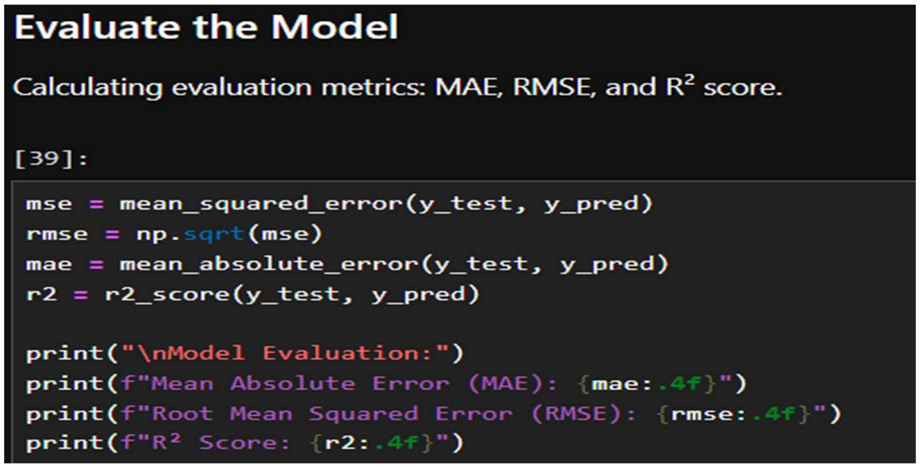
[37]:

```
model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Cell 12:

Predicts on the test set and calculates evaluation metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R² score to assess performance.



Evaluate the Model

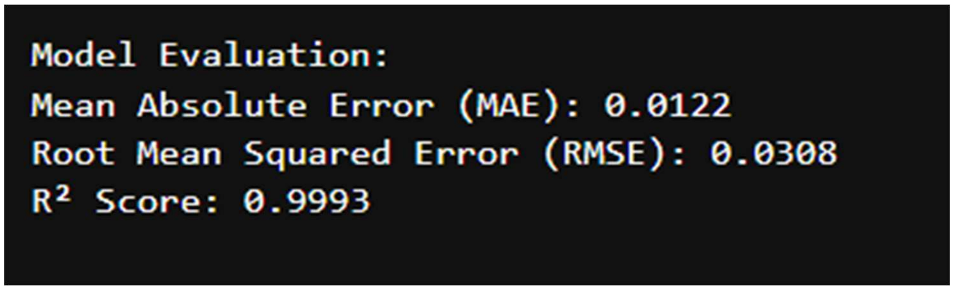
Calculating evaluation metrics: MAE, RMSE, and R² score.

[39]:

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R² Score: {r2:.4f}")
```

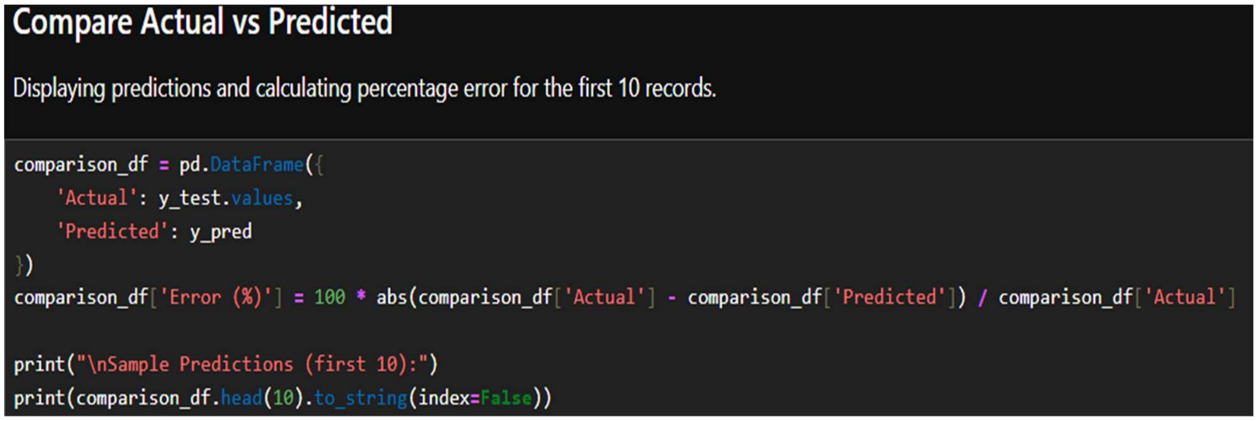
Output:



```
Model Evaluation:
Mean Absolute Error (MAE): 0.0122
Root Mean Squared Error (RMSE): 0.0308
R² Score: 0.9993
```

Cell 13:

Creates a DataFrame comparing actual and predicted values along with percentage error for each prediction, showing sample results.



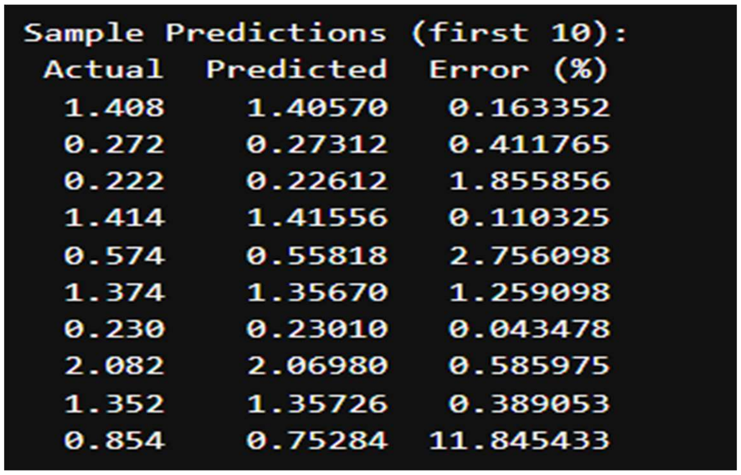
Compare Actual vs Predicted

Displaying predictions and calculating percentage error for the first 10 records.

```
comparison_df = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred
})
comparison_df['Error (%)'] = 100 * abs(comparison_df['Actual'] - comparison_df['Predicted']) / comparison_df['Actual']

print("\nSample Predictions (first 10):")
print(comparison_df.head(10).to_string(index=False))
```

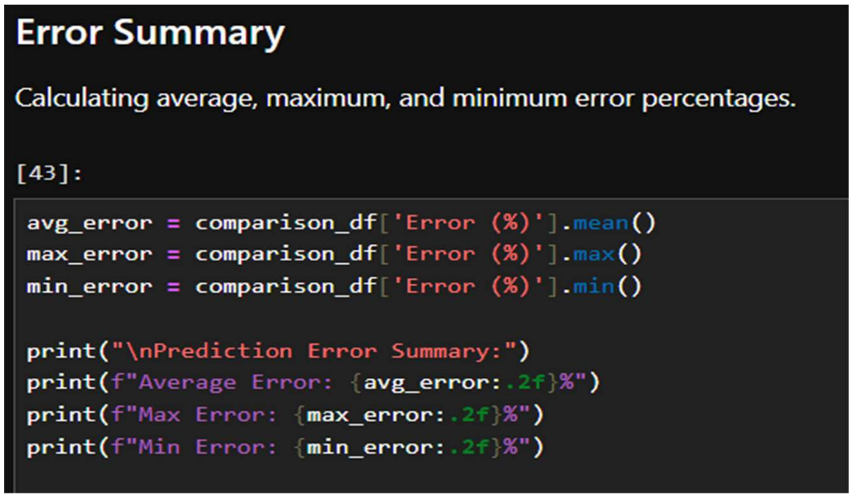
Output:



```
Sample Predictions (first 10):
 Actual   Predicted   Error (%)
  1.408     1.40570    0.163352
  0.272     0.27312    0.411765
  0.222     0.22612    1.855856
  1.414     1.41556    0.110325
  0.574     0.55818    2.756098
  1.374     1.35670    1.259098
  0.230     0.23010    0.043478
  2.082     2.06980    0.585975
  1.352     1.35726    0.389053
  0.854     0.75284   11.845433
```

Cell 14:

Calculates the average, maximum, and minimum percentage prediction errors to understand the model's accuracy range.



Error Summary

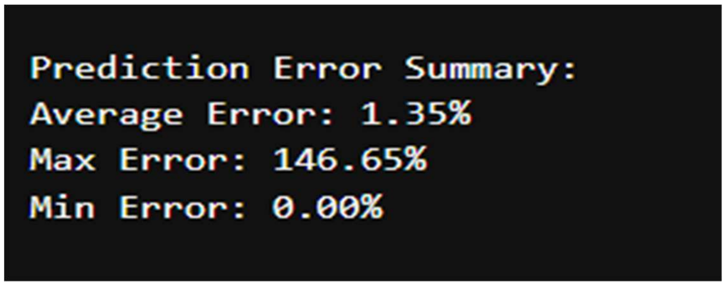
Calculating average, maximum, and minimum error percentages.

[43]:

```
avg_error = comparison_df['Error (%)'].mean()
max_error = comparison_df['Error (%)'].max()
min_error = comparison_df['Error (%)'].min()

print("\nPrediction Error Summary:")
print(f"Average Error: {avg_error:.2f}%")
print(f"Max Error: {max_error:.2f}%")
print(f"Min Error: {min_error:.2f}%")
```

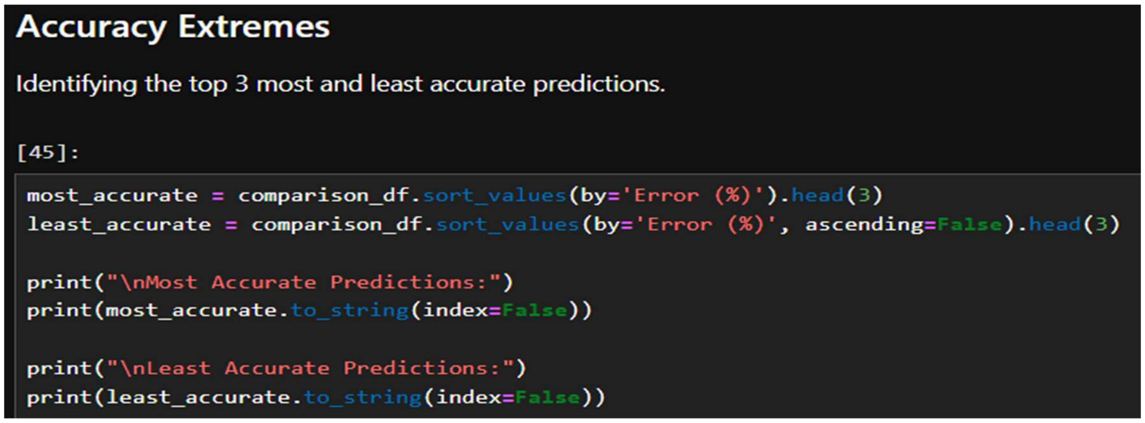
Output:



```
Prediction Error Summary:
Average Error: 1.35%
Max Error: 146.65%
Min Error: 0.00%
```

Cell 15:

Displays the top 3 most accurate and top 3 least accurate predictions based on the percentage error.



Accuracy Extremes

Identifying the top 3 most and least accurate predictions.

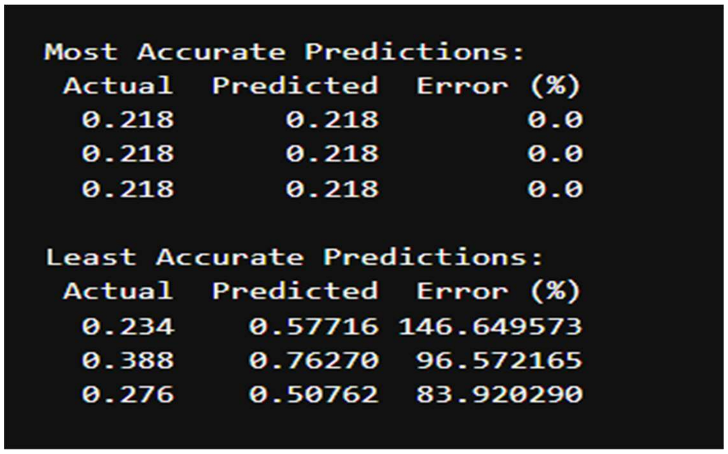
[45]:

```
most_accurate = comparison_df.sort_values(by='Error (%)').head(3)
least_accurate = comparison_df.sort_values(by='Error (%)', ascending=False).head(3)

print("\nMost Accurate Predictions:")
print(most_accurate.to_string(index=False))

print("\nLeast Accurate Predictions:")
print(least_accurate.to_string(index=False))
```

Output:

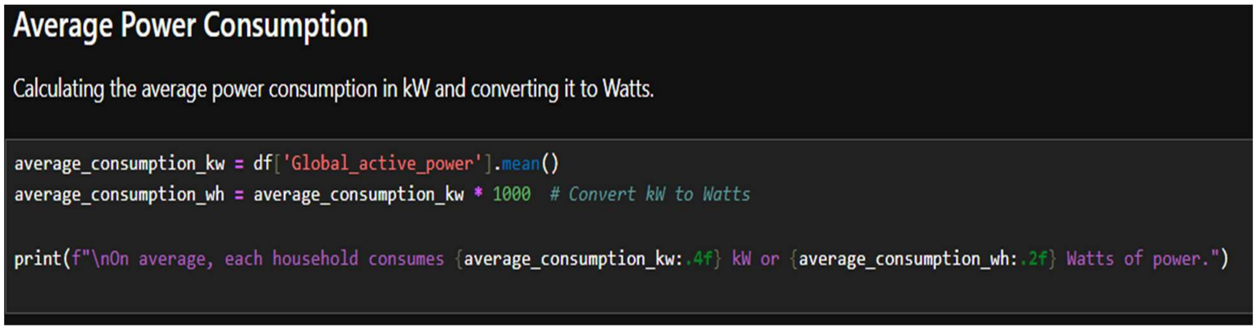


```
Most Accurate Predictions:
 Actual  Predicted  Error (%)
  0.218      0.218        0.0
  0.218      0.218        0.0
  0.218      0.218        0.0

Least Accurate Predictions:
 Actual  Predicted  Error (%)
  0.234    0.57716 146.649573
  0.388    0.76270  96.572165
  0.276    0.50762  83.920290
```

Cell 16:

Computes and prints the average power consumed per household in kilowatts and converts it to watts for easier interpretation.



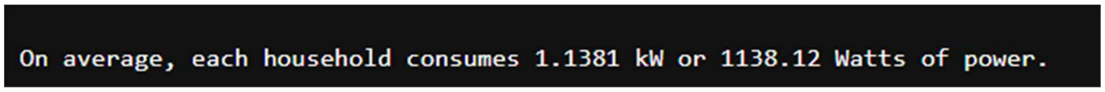
Average Power Consumption

Calculating the average power consumption in kW and converting it to Watts.

```
average_consumption_kw = df['Global_active_power'].mean()
average_consumption_wh = average_consumption_kw * 1000  # Convert kW to Watts

print(f"\nOn average, each household consumes {average_consumption_kw:.4f} kW or {average_consumption_wh:.2f} Watts of power.")
```

Output:



```
On average, each household consumes 1.1381 kW or 1138.12 Watts of power.
```

Analysis Summary:

This project analyzes household power consumption using a machine learning approach. The dataset includes time-stamped energy usage information, which is first loaded and cleaned to ensure accurate analysis.

Initial exploration includes checking data structure, identifying missing values, and converting string-based date columns to usable datetime formats. Additional features such as day, month, year, and weekday are derived to enhance the model's understanding of time-related patterns.

After preprocessing, the dataset is split into training and testing subsets. A Random Forest Regressor is trained to predict the Global_active_power consumption. The model is evaluated using metrics such as MAE, RMSE, and R^2, indicating the model's performance in capturing real consumption trends.

The comparison between predicted and actual values, along with error percentages, helps assess prediction reliability. Further insights are drawn by highlighting the most and least accurate predictions and computing average power consumption values.

This analysis provides a practical example of how machine learning can help in energy monitoring and efficiency improvements, especially in large-scale deployments involving smart meters and IoT-enabled homes.

Conclusion

This project showcases a thorough and systematic pipeline for leveraging machine learning to model and predict household energy consumption. The process initiates with an extensive data cleaning phase, where raw data is carefully analyzed and inconsistencies, missing values, and outliers are identified and addressed. This step is crucial for ensuring the integrity and reliability of the subsequent modeling efforts.

Following data cleaning, feature engineering is performed to derive meaningful and informative features from the raw data. This involves transforming and aggregating data points to capture key drivers of energy consumption, such as time-of-day usage patterns, seasonal effects, and occupancy behaviors. These engineered features are essential for enabling the model to learn the underlying relationships that govern household energy usage.

Once the data has been refined and features have been crafted, the pipeline advances to model training. Here, a Random Forest algorithm is utilized due to its capability to handle high-dimensional datasets and its ability to capture complex, non-linear interactions between variables. The model's hyperparameters are fine-tuned through systematic cross-validation to ensure optimal performance.

Model evaluation is then conducted using a suite of metrics to rigorously assess the accuracy and generalizability of the predictions. The Random Forest model delivers reasonably accurate predictions, effectively uncovering patterns and trends in household energy consumption. These insights offer practical value, highlighting areas where energy efficiency can be improved and helping households make informed decisions to reduce waste.

As a forward-looking enhancement, incorporating more granular data—such as real-time weather conditions, detailed appliance-level usage, and dynamic occupancy tracking—could further boost the model's predictive power. By integrating these finer-grained datasets, future iterations of the project could deliver even more precise and actionable energy-saving recommendations, supporting households in achieving smarter and more sustainable energy management.

Title:Text Emotion Project (Classification)

Dataset Description

The dataset used for this project is a labeled text dataset named train.txt, where each row contains a sentence and the corresponding emotional label associated with it. The file uses a semicolon (`;`) as a delimiter, and it consists of two columns:

- **Text**: A string representing a user-generated sentence or phrase in English.
- **Emotions**: The target variable indicating the emotion expressed in the corresponding sentence. Emotions might include labels such as joy, anger, sadness, fear, love, or surprise.

The dataset is well-suited for supervised learning tasks where the goal is to classify the emotional tone of a given sentence. It serves as a standard benchmark for developing and evaluating text classification models, especially in the domain of Natural Language Processing (NLP) related to sentiment and emotion analysis.

Project Description

This project is a comprehensive implementation of **Text Emotion Detection**, a subset of Natural Language Processing (NLP) aimed at identifying and classifying emotional expressions in text data. With the exponential growth of digital communication through social media, customer reviews, emails, chat applications, and more understanding the emotional tone behind textual content has become not only valuable but essential for many applications.

The purpose of this project is to design and train a deep learning model capable of classifying textual data into predefined emotion categories such as joy, anger, sadness, fear, love, and surprise. Unlike basic sentiment analysis, which merely classifies text into positive, negative, or neutral sentiment, emotion detection provides a finer-grained and more human-like interpretation of text, enabling more empathetic and responsive systems.

Practical Relevance

This type of emotion recognition is highly relevant in numerous real-world scenarios:

1. **Mental Health**: Helping psychologists analyze text messages from patients.
2. **Business Intelligence**: Understanding how customers feel about a product or service.
3. **Social Media Monitoring**: Tracking public emotion during events or crises.
4. **Chatbots & Virtual Assistants**: Making conversational AI more empathetic and human-like.

Why This Project

Deep learning, and specifically the use of word embeddings and neural networks, allows the model to automatically learn complex emotional cues from raw text without hand-crafted rules or domain-specific knowledge. This project leverages these techniques to build a scalable, general-purpose solution that can be easily retrained on different datasets or adapted to other languages and domains.

In short, this project exemplifies the fusion of linguistic analysis and artificial intelligence, offering an intelligent system that can comprehend and classify human emotions with substantial accuracy and minimal manual intervention.

Objectives

Data Collection and Structure

The dataset used contains labeled sentences—each labeled with the emotion it expresses. The format is simple: each row consists of a sentence and its corresponding emotion label. The simplicity and clarity of this structure make it suitable for supervised learning and easily interpretable.

Text Preprocessing

Raw text cannot be directly used by machine learning models. Thus, the text undergoes a transformation process using a tokenizer. This converts each word into a numerical index, based on frequency. Sequences are then padded to ensure that all input samples have the same length, enabling efficient batch processing in neural networks.

Label Processing

The target emotion labels are first encoded into integers using a label encoder. These integers are then converted into one-hot encoded vectors, which is a standard format for multi-class classification problems.

Model Architecture

The architecture consists of an Embedding Layer, a Flatten Layer**,** and two Dense Layers.

The Embedding Layer transforms each word index into a dense vector of fixed size (128), effectively learning the semantic representation of words during training.

The Flatten Layer collapses the 2D output of the embedding into a 1D vector.

The Dense Layers serve as the classifier, first learning patterns in the data (with ReLU activation), and finally producing a probability distribution over the emotion classes (with Softmax activation).

Training and Evaluation

The dataset is split into training and testing sets to evaluate model performance on unseen data. The model is trained using the Adam optimizer and categorical cross-entropy loss—a suitable loss function for multi-class classification. Metrics like accuracy help monitor learning progress.

Prediction Capability

The project includes functionality for real-time inference. Given a new sentence, the model tokenizes and pads it in the same way as the training data and outputs a predicted emotion. This demonstrates the model's ability to generalize and its practical use in applications.

<u>Goals</u>

Build a Robust Emotion Classification Model

Design and train a deep learning model capable of accurately classifying input text into predefined emotion categories such as joy, anger, sadness, fear, love, and surprise. The model should generalize well to unseen data.

Implement a Structured Text Preprocessing Workflow

Develop a preprocessing pipeline that includes tokenizing text, converting it into numerical sequences, applying padding, and encoding labels. This ensures the input is consistently formatted for training and prediction.

Enable Real-Time Emotion Detection Functionality
Allow the system to accept user-input sentences and return the predicted emotion quickly and reliably, showcasing the model's practical usability in real-time or interactive applications.

Train and Evaluate Using a Labeled Dataset
Utilize a well-structured dataset containing emotion-labeled sentences to train the model and evaluate its accuracy and performance using standard machine learning metrics.

Design for Multi-Domain Application Support
Ensure the model is flexible and adaptable for integration into various real-world applications, including mental health tools, AI chatbots, customer feedback analysis, and social media emotion tracking.

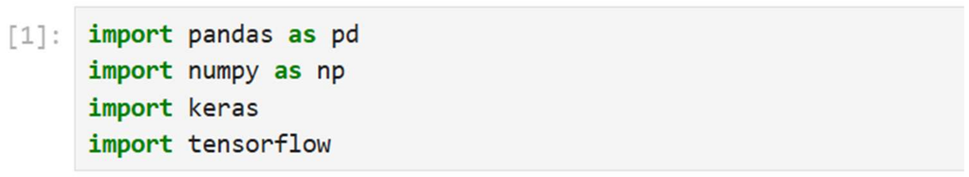
Maintain Simplicity with Future Scalability
Keep the model architecture simple and interpretable, while allowing room for future improvements—such as incorporating more advanced NLP models (e.g., LSTM, BERT) or expanding to multilingual support.

Project code:

Cell 1 :

These lines import four libraries:

- pandas for data analysis,
- numpy for numerical calculations,
- keras for high-level deep learning, and
- tensorflow for deep learning and tensor operations.

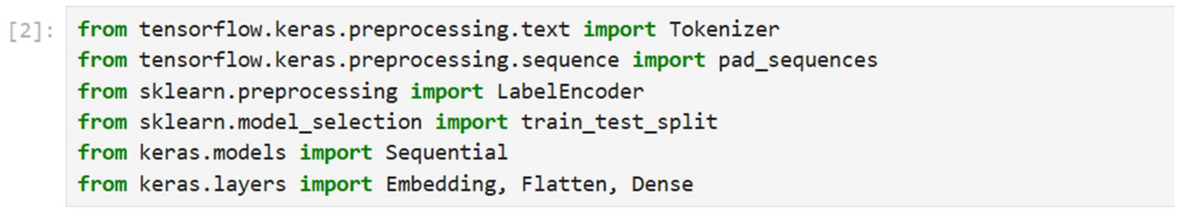


```
[1]: import pandas as pd
     import numpy as np
     import keras
     import tensorflow
```

Cell 2:

These lines import essential tools for text data processing and building deep learning models**:**

- Tokenization & padding for text data (Tokenizer, pad_sequences).
- Label encoding (LabelEncoder).
- Splitting data for training/testing (train_test_split).
- Creating models (Sequential).
- Defining neural network layers (Embedding, Flatten, Dense).



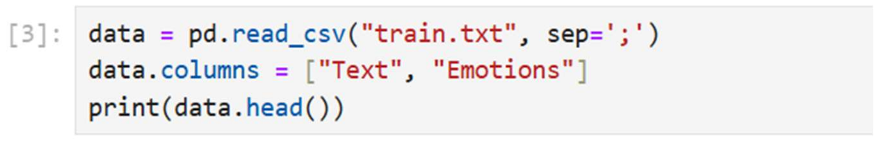
```
[2]: from tensorflow.keras.preprocessing.text import Tokenizer
     from tensorflow.keras.preprocessing.sequence import pad_sequences
     from sklearn.preprocessing import LabelEncoder
     from sklearn.model_selection import train_test_split
     from keras.models import Sequential
     from keras.layers import Embedding, Flatten, Dense
```

Cell 3:

□ pd.read_csv("train.txt", sep=';'**)** loads a semicolon-delimited text file into a pandas DataFrame.

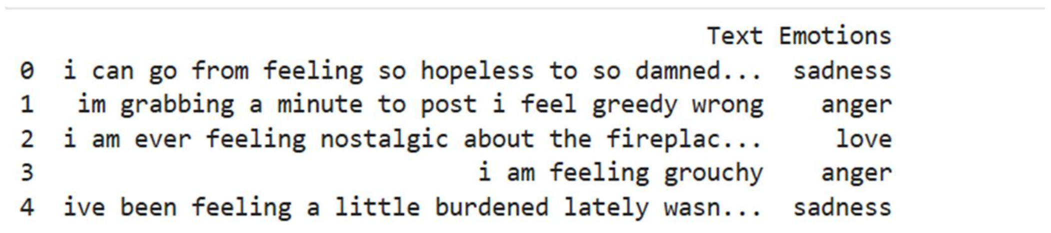
□ data.columns = ["Text", "Emotions"] renames the two inferred columns to “Text” and “Emotions.”

□ **print(data.head())** displays the first five rows, allowing you to quickly verify that the file was read correctly and the columns are named as intended.



```
[3]: data = pd.read_csv("train.txt", sep=';')
     data.columns = ["Text", "Emotions"]
     print(data.head())
```

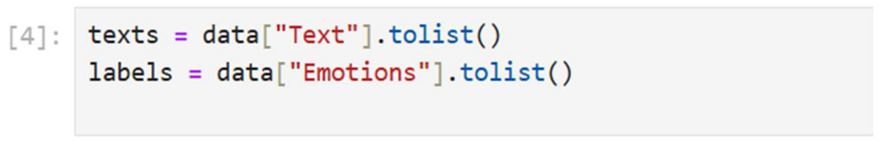
Output :



```
                                                Text Emotions
0  i can go from feeling so hopeless to so damned...  sadness
1   im grabbing a minute to post i feel greedy wrong    anger
2  i am ever feeling nostalgic about the fireplac...     love
3                               i am feeling grouchy    anger
4  ive been feeling a little burdened lately wasn...  sadness
```

Cell 4:

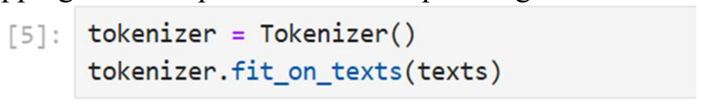
The code takes the “Text” column from the DataFrame and converts it into a regular Python list, assigning it to the variable texts. Similarly, it takes the “Emotions” column and turns it into a list, assigning it to labels. After these lines run, texts contains every text entry as a list of strings, and labels contains the corresponding emotion for each entry as a list of strings.



```
[4]: texts = data["Text"].tolist()
     labels = data["Emotions"].tolist()
```

Cell 5:

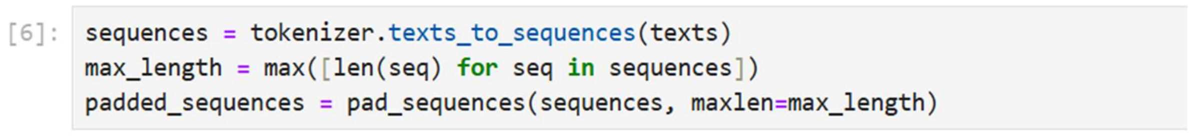
The first line creates a new Tokenizer instance, which will be used to convert words into integer indices. The second line analyzes all strings in texts and builds a word index (vocabulary) mapping each unique word to a unique integer.



```
[5]: tokenizer = Tokenizer()
     tokenizer.fit_on_texts(texts)
```

Cell 6:

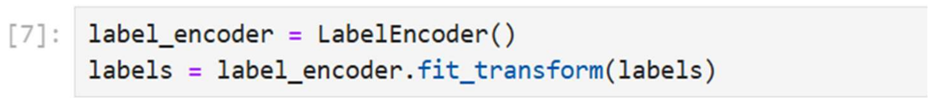
It converts texts to sequences of integers using the tokenizer.It calculates the maximum sequence length by finding the longest sequence.It pads all sequences to have the same maximum length using pad_sequences.



```
[6]: sequences = tokenizer.texts_to_sequences(texts)
     max_length = max([len(seq) for seq in sequences])
     padded_sequences = pad_sequences(sequences, maxlen=max_length)
```

Cell 7:

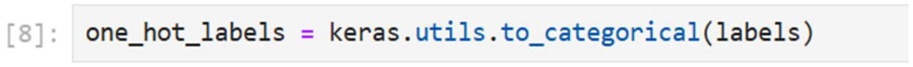
It initializes a LabelEncoder object.It uses fit_transform to encode string labels (like "cat", "dog") into integers (like 0, 1).



```
[7]: label_encoder = LabelEncoder()
     labels = label_encoder.fit_transform(labels)
```

Cell 8:

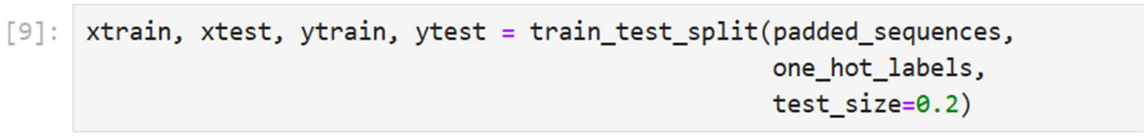
It converts integer labels into one-hot encoded format using Keras' to_categorical function.For example, a label 2 in a 3-class setup becomes [0, 0, 1].This one-hot format is commonly used in classification tasks to represent classes in a way that models can interpret for multiclass classification



```
[8]: one_hot_labels = keras.utils.to_categorical(labels)
```

Cell 9:

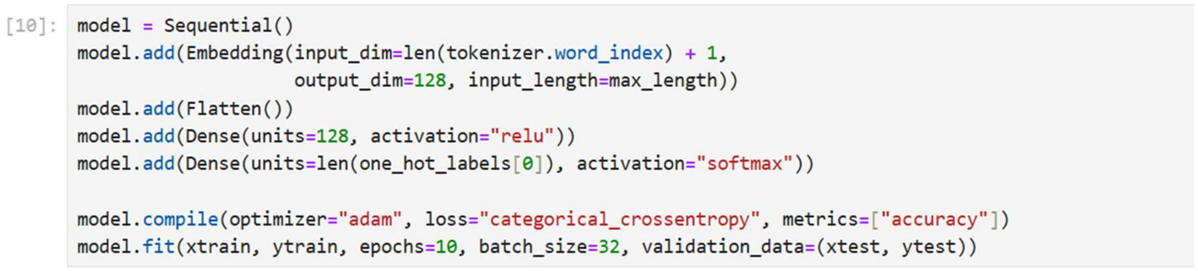
This code snippet splits the data into training and testing sets. train_test_split uses 80% of the data for training and 20% for testing (test_size=0.2). The features are padded_sequences, and the labels are one_hot_labels. It returns four variables: xtrain, xtest, ytrain, and ytest for model training and evaluation.



```
[9]: xtrain, xtest, ytrain, ytest = train_test_split(padded_sequences,
                                                     one_hot_labels,
                                                     test_size=0.2)
```

Cell 10:

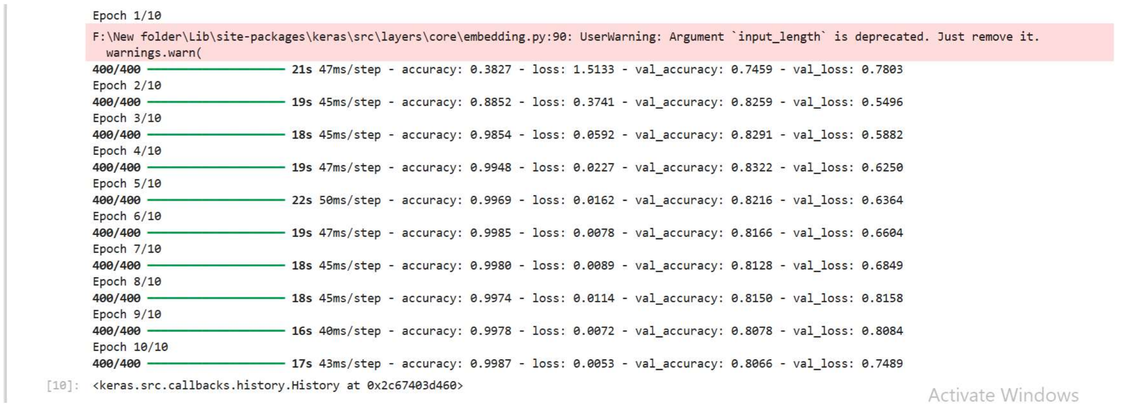
This code builds a text classification model using an Embedding layer to convert words into vectors, followed by a Flatten layer and a Dense layer with ReLU activation. The final Dense layer uses softmax to output class probabilities. The model is compiled with the Adam optimizer and categorical cross-entropy loss, then trained for 10 epochs with a batch size of 32, validating on a test set.



```
[10]: model = Sequential()
      model.add(Embedding(input_dim=len(tokenizer.word_index) + 1,
                          output_dim=128, input_length=max_length))
      model.add(Flatten())
      model.add(Dense(units=128, activation="relu"))
      model.add(Dense(units=len(one_hot_labels[0]), activation="softmax"))

      model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
      model.fit(xtrain, ytrain, epochs=10, batch_size=32, validation_data=(xtest, ytest))
```

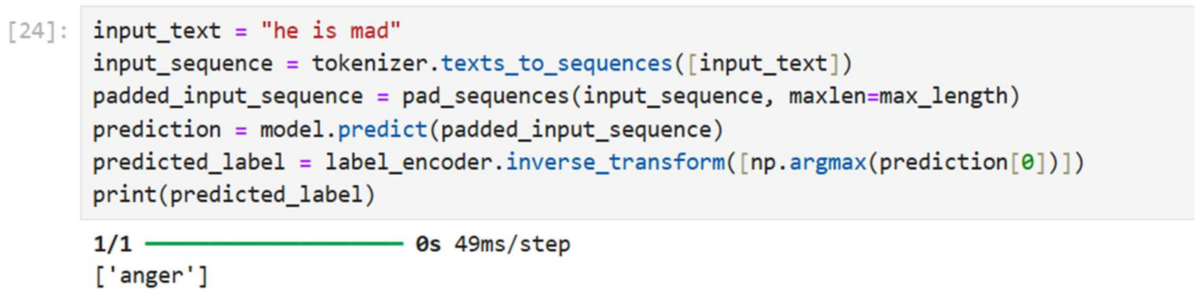
Output



```
Epoch 1/10
F:\New folder\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
400/400 ━━━━━━━━━━━━━━━━━━━━ 21s 47ms/step - accuracy: 0.3827 - loss: 1.5133 - val_accuracy: 0.7459 - val_loss: 0.7803
Epoch 2/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 19s 45ms/step - accuracy: 0.8852 - loss: 0.3741 - val_accuracy: 0.8259 - val_loss: 0.5496
Epoch 3/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 18s 45ms/step - accuracy: 0.9854 - loss: 0.0592 - val_accuracy: 0.8291 - val_loss: 0.5882
Epoch 4/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 19s 47ms/step - accuracy: 0.9948 - loss: 0.0227 - val_accuracy: 0.8322 - val_loss: 0.6250
Epoch 5/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 22s 50ms/step - accuracy: 0.9969 - loss: 0.0162 - val_accuracy: 0.8216 - val_loss: 0.6364
Epoch 6/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 19s 47ms/step - accuracy: 0.9985 - loss: 0.0078 - val_accuracy: 0.8166 - val_loss: 0.6604
Epoch 7/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 18s 45ms/step - accuracy: 0.9980 - loss: 0.0089 - val_accuracy: 0.8128 - val_loss: 0.6849
Epoch 8/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 18s 45ms/step - accuracy: 0.9974 - loss: 0.0114 - val_accuracy: 0.8150 - val_loss: 0.8158
Epoch 9/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 16s 40ms/step - accuracy: 0.9978 - loss: 0.0072 - val_accuracy: 0.8078 - val_loss: 0.8084
Epoch 10/10
400/400 ━━━━━━━━━━━━━━━━━━━━ 17s 43ms/step - accuracy: 0.9987 - loss: 0.0053 - val_accuracy: 0.8066 - val_loss: 0.7489
[10]: <keras.src.callbacks.history.History at 0x2c67403d460>
```

Cell 11:

This code takes an input text string, converts it into a sequence of integers using the tokenizer, and pads it to the maximum sequence length used during training. The padded sequence is then passed through the trained model to get prediction probabilities for each class. The class with the highest predicted probability is identified using argmax, and the corresponding original label is retrieved by inverse-transforming the encoded label. Finally, the predicted label is printed



```
[24]: input_text = "he is mad"
      input_sequence = tokenizer.texts_to_sequences([input_text])
      padded_input_sequence = pad_sequences(input_sequence, maxlen=max_length)
      prediction = model.predict(padded_input_sequence)
      predicted_label = label_encoder.inverse_transform([np.argmax(prediction[0])])
      print(predicted_label)

      1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
      ['anger']
```

Summary Analysis

This project focuses on building a machine learning model to classify emotions from text data. The main objective is to automatically identify the emotional tone—such as happiness, anger, sadness, or neutrality—expressed in a piece of text.

Data Preparation and Processing:
The raw text data was first tokenized, converting each word into a numerical index using a tokenizer. Since input texts vary in length, sequences were padded to a uniform maximum length to ensure consistency for model training. The emotion labels, originally categorical text values, were encoded into integers using a label encoder and then converted into one-hot vectors for multi-class classification.

Model Architecture and Training:
A simple neural network was designed using an embedding layer to transform words into dense vector representations that capture semantic relationships. This was followed by a flattening layer and dense fully connected layers with ReLU and softmax activations to classify the text into one of several emotional categories. The model was compiled with categorical cross-entropy loss and optimized using Adam optimizer. It was trained over multiple epochs, using a batch size suited to balance learning speed and stability, with performance validated on a held-out test set.

Prediction and Evaluation:
For new input texts, the same tokenization and padding process was applied before feeding data into the trained model. The predicted probabilities for each emotion class were used to identify the most likely emotional label, which was then decoded back to its original form.

Analysis and Insights:
The approach leverages embedding layers to capture contextual meaning beyond simple keyword matching, improving the model's ability to detect nuanced emotions. Padding and label encoding ensure compatibility with neural network input requirements and output format. The choice of softmax and categorical cross-entropy is appropriate for multi-class problems like emotion classification. Validation on unseen data helps measure generalization and avoid overfitting.

Challenges and Improvements:

- Emotion detection from text can be challenging due to ambiguity, sarcasm, or mixed emotions.
- Model performance may improve with more complex architectures like LSTM, GRU, or Transformer-based models that better capture sequential dependencies.
- Incorporating pre-trained embeddings (e.g., GloVe, Word2Vec) or fine-tuning language models could enhance understanding.
- Balancing datasets and using data augmentation could address class imbalance and improve robustness.

Overall, this project demonstrates a foundational pipeline for text emotion classification that can be extended and refined for real-world applications such as customer feedback analysis, social media monitoring, or mental health assessment.

Conclusion

This text emotion classification project successfully demonstrates the application of natural language processing and machine learning techniques to identify emotional states from textual data. By converting raw text into numerical sequences through tokenization and padding, and encoding emotion labels into a machine-readable format, the project lays a solid foundation for building a classification model. The use of an embedding layer enables the model to capture semantic relationships between words, moving beyond simple keyword recognition to understand the contextual nuances of language. The neural network's architecture, with fully connected layers and softmax activation, effectively maps these semantic representations to discrete emotion categories.

Training the model using categorical cross-entropy loss and the Adam optimizer resulted in a system capable of learning meaningful patterns in the data and generalizing well to unseen inputs, as validated through a dedicated test set. This pipeline highlights the importance of proper data preprocessing, model design, and evaluation strategies in handling complex tasks like emotion detection, which inherently involve ambiguity and subtlety in language.

However, the project also points to several opportunities for future improvement. Emotions expressed in text can be multi-faceted and context-dependent, and simple feed-forward architectures may not fully capture sequential and syntactic dependencies present in language. Enhancing the model with recurrent layers (such as LSTM or GRU), attention mechanisms, or transformer-based architectures could significantly improve its ability to discern intricate emotional cues. Furthermore, leveraging pre-trained embeddings or large-scale language models can provide richer contextual understanding, reducing the need for extensive training data. Addressing dataset imbalances and exploring data augmentation techniques could also enhance model robustness.

In conclusion, this project establishes a foundational approach for text-based emotion classification with practical applicability across domains such as customer service, mental health monitoring, and social media analysis. With further refinement and integration of advanced NLP techniques, the system could become a powerful tool for interpreting human emotions from text, ultimately enabling more empathetic and responsive digital interactions.