Tutorial – 4

TSA

1. Write the python code to build a multiclass text classification system.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

data = pd.read_csv("your_twitter_sentiment_dataset.csv")

# Splitting the dataset into features (X) and labels (y)
X = data['text']
y = data['label']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Text preprocessing & vectorization using TF-IDF
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Train a Multinomial Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_tfidf, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test_tfidf)

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

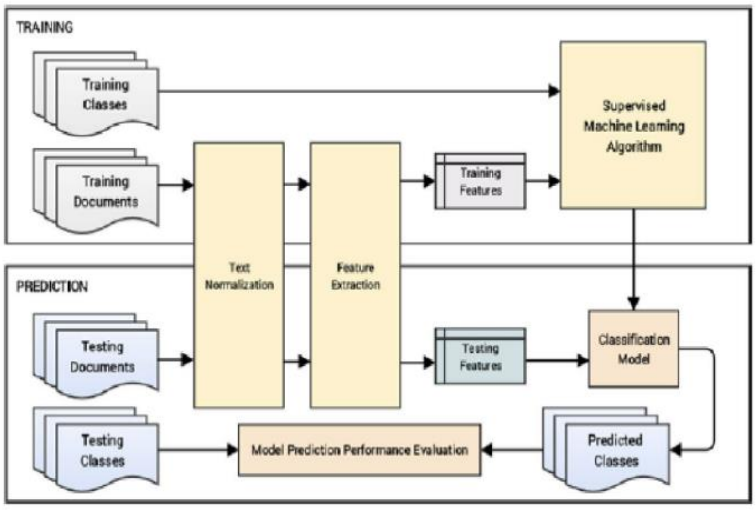
2.Write the python code to build a binary class text classification system.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

data = pd.read_csv("your_binary_sentiment_dataset.csv")

# Splitting the dataset into features (X) and labels (y)
X = data['text']
y = data['label']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Text preprocessing & vectorization using TF-IDF
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)  #
Adjust max_features as needed
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Train a Multinomial Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_tfidf, y_train)
```

```
# Make predictions on the test set
y_pred = clf.predict(X_test_tfidf)

# Display a detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

3.Draw the blueprint of text classification system and explain the components in it.

Blueprint for building an automated text classification system



Text Classification Process

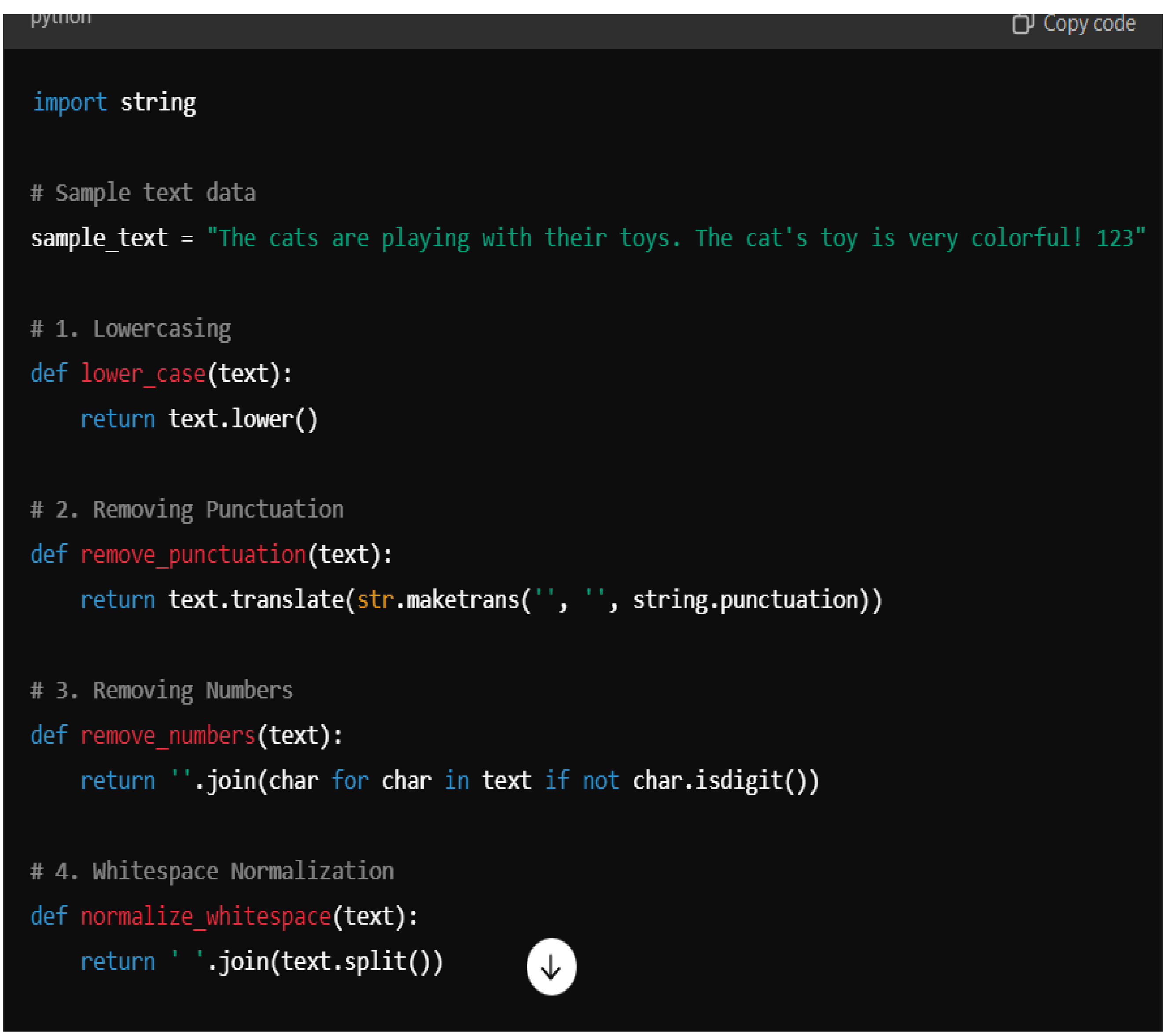
Training Process:

1. **Dataset Splitting**: Divided into training, validation (optional), and testing sets.
2. **Text Normalization**: Pre-process and standardize text.
3. **Feature Extraction**: Convert text into numerical features (e.g., TF-IDF).
4. **Model Training**: Train a supervised ML model on feature vectors and labels.
5. **Model Evaluation**: Use validation data (optional) to fine-tune performance.
6. **Hyperparameter Tuning**: Optimize model parameters.
7. **Classification Model**: The final trained model, ready for prediction.

Prediction Process:

1. **Text Normalization & Feature Extraction**: Apply the same steps as in training.
2. **Classification**: Model predicts labels for new or test documents.
3. **Model Evaluation**: Measure accuracy if true labels are available.
4. **Deployment**: Save and deploy the model for future predictions.

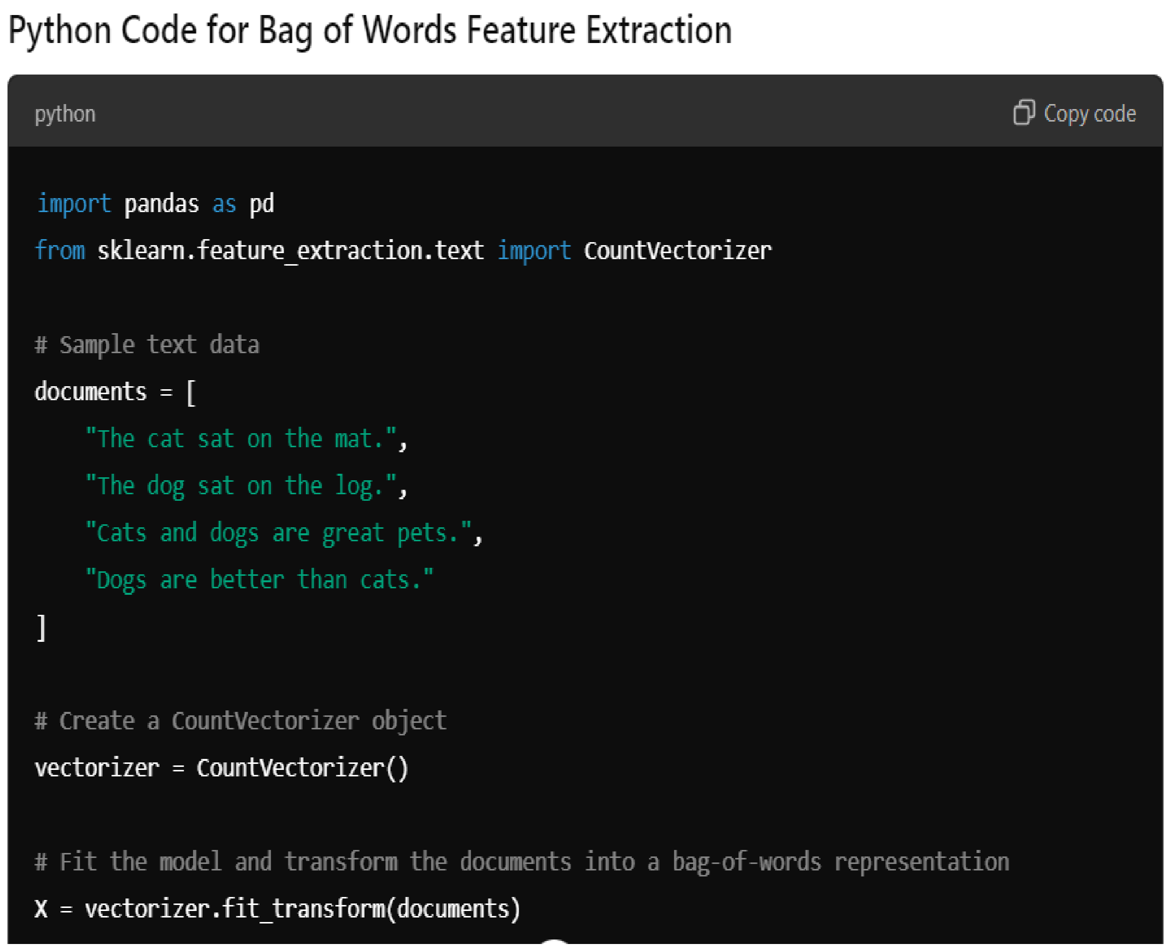
4.Write the python code to apply any four text normalization techniques into sample text data.



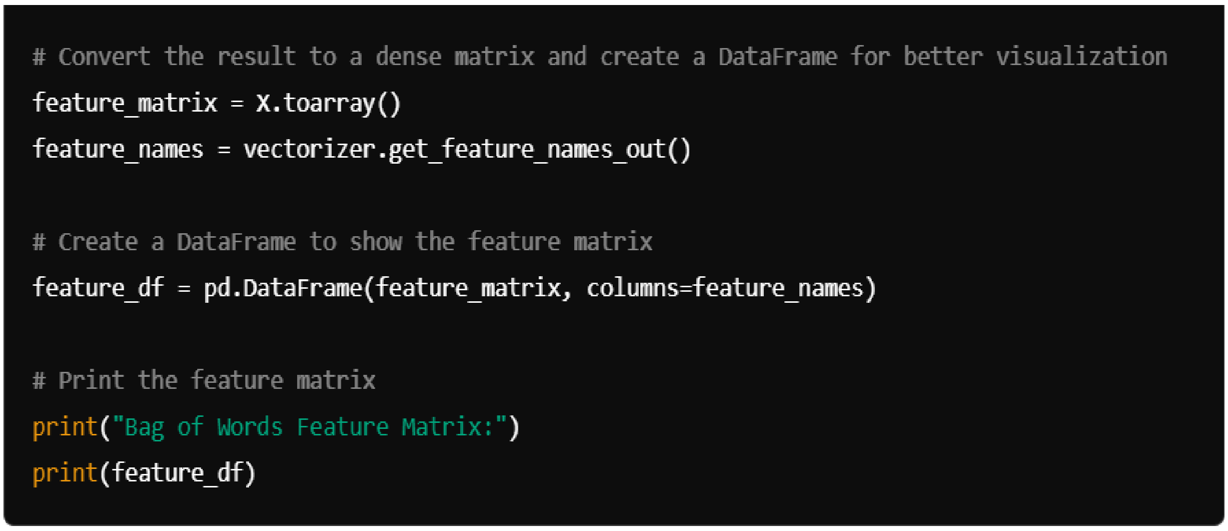
```python
import string

# Sample text data
sample_text = "The cats are playing with their toys. The cat's toy is very colorful! 123"

# 1. Lowercasing
def lower_case(text):
    return text.lower()

# 2. Removing Punctuation
def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

# 3. Removing Numbers
def remove_numbers(text):
    return ''.join(char for char in text if not char.isdigit())

# 4. Whitespace Normalization
def normalize_whitespace(text):
    return ' '.join(text.split())
```

```python
# 4. Whitespace Normalization
def normalize_whitespace(text):
    return ' '.join(text.split())

# Apply normalization techniques
lowercased_text = lower_case(sample_text)
punctuation_removed_text = remove_punctuation(lowercased_text)
numbers_removed_text = remove_numbers(punctuation_removed_text)
normalized_whitespace_text = normalize_whitespace(numbers_removed_text)

# Print the results
print("Original Text:\n", sample_text)
print("\nLowercased Text:\n", lowercased_text)
print("\nPunctuation Removed:\n", punctuation_removed_text)
print("\nNumbers Removed:\n", numbers_removed_text)
print("\nNormalized Whitespace:\n", normalized_whitespace_text)
```

5. Write the python code to perform feature extraction using Bag or Words model.

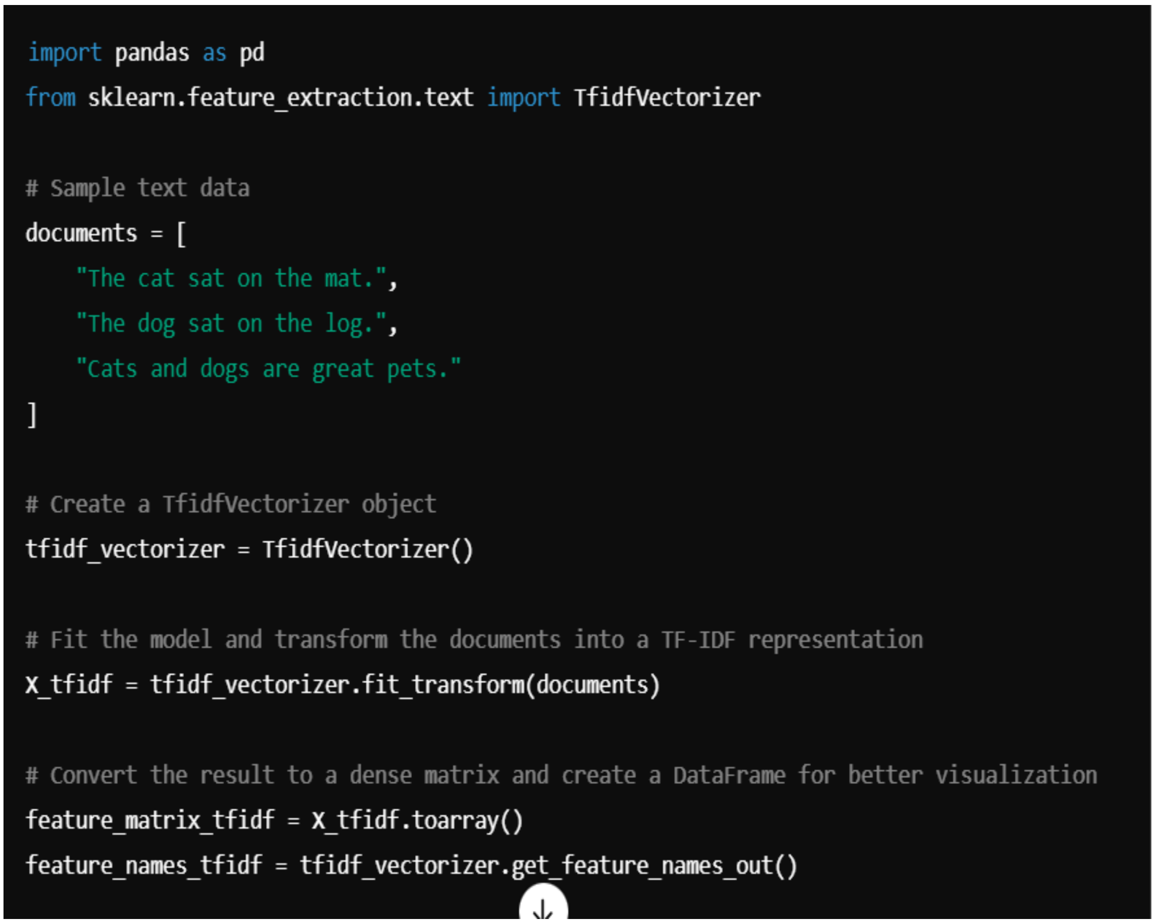

Python Code for Bag of Words Feature Extraction

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Sample text data
documents = [
    "The cat sat on the mat.",
    "The dog sat on the log.",
    "Cats and dogs are great pets.",
    "Dogs are better than cats."
]

# Create a CountVectorizer object
vectorizer = CountVectorizer()

# Fit the model and transform the documents into a bag-of-words representation
X = vectorizer.fit_transform(documents)
```

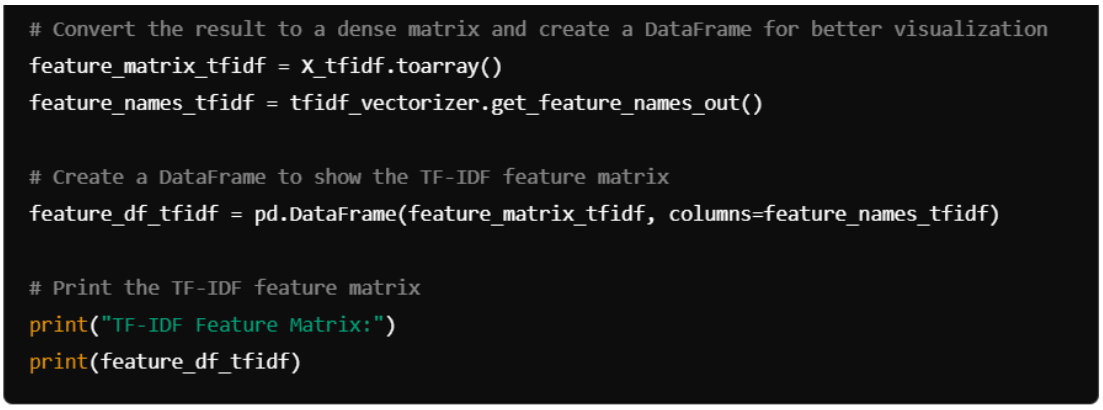


```python
# Convert the result to a dense matrix and create a DataFrame for better visualization
feature_matrix = X.toarray()
feature_names = vectorizer.get_feature_names_out()

# Create a DataFrame to show the feature matrix
feature_df = pd.DataFrame(feature_matrix, columns=feature_names)

# Print the feature matrix
print("Bag of Words Feature Matrix:")
print(feature_df)
```

6. Write the python code to perform feature extraction using TF-IDF model



```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample text data
documents = [
    "The cat sat on the mat.",
    "The dog sat on the log.",
    "Cats and dogs are great pets."
]

# Create a TfidfVectorizer object
tfidf_vectorizer = TfidfVectorizer()

# Fit the model and transform the documents into a TF-IDF representation
X_tfidf = tfidf_vectorizer.fit_transform(documents)

# Convert the result to a dense matrix and create a DataFrame for better visualization
feature_matrix_tfidf = X_tfidf.toarray()
feature_names_tfidf = tfidf_vectorizer.get_feature_names_out()
```



```python
# Convert the result to a dense matrix and create a DataFrame for better visualization
feature_matrix_tfidf = X_tfidf.toarray()
feature_names_tfidf = tfidf_vectorizer.get_feature_names_out()
```

```python

# Create a DataFrame to show the TF-IDF feature matrix
feature_df_tfidf = pd.DataFrame(feature_matrix_tfidf, columns=feature_names_tfidf)

# Print the TF-IDF feature matrix
print("TF-IDF Feature Matrix:")
print(feature_df_tfidf)
```