Natural Language Processing (NLP) Course Cheatsheet with Code Examples

📋 Course Overview

This comprehensive cheatsheet covers 5 sessions with practical code examples for every concept.

🎯 Session 1: NLP Introduction & Text Pre-processing

What is NLP?

- **Definition**: Computer's ability to understand, interpret, and generate human language
- **Goal**: Bridge the gap between human communication and computer understanding

Why NLP Matters

- **Need**: Massive amounts of unstructured text data require automated processing
- **Applications**:
  - Chatbots & Virtual Assistants
  - Sentiment Analysis
  - Machine Translation
  - Text Summarization
  - Information Extraction

Text Pre-processing with Code Examples

1. Basic Text Cleaning

python

```python
import re
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk

# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

def preprocess_text(text):
    # 1. Convert to lowercase
    text = text.lower()

    # 2. Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # 3. Remove user mentions and hashtags
    text = re.sub(r'@\w+|#\w+', '', text)

    # 4. Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # 5. Remove extra whitespace
    text = ' '.join(text.split())

    return text

# Example usage
raw_text = "Hello World! This is an AMAZING example. Visit https://example.com @user #NLP"
cleaned = preprocess_text(raw_text)
print(f"Original: {raw_text}")
print(f"Cleaned: {cleaned}")
# Output: hello world this is an amazing example
```

2. Tokenization

python

```python
def tokenize_text(text):
    # Word tokenization
    tokens = word_tokenize(text)
    return tokens

# Example
text = "Natural Language Processing is fascinating!"
tokens = tokenize_text(text)
print(f"Tokens: {tokens}")
# Output: ['Natural', 'Language', 'Processing', 'is', 'fascinating', '!']
```

3. Stop Words Removal

python

```python
def remove_stopwords(tokens):
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
    return filtered_tokens

# Example
tokens = ['natural', 'language', 'processing', 'is', 'fascinating']
filtered = remove_stopwords(tokens)
print(f"Without stopwords: {filtered}")
# Output: ['natural', 'language', 'processing', 'fascinating']
```

4. Stemming and Lemmatization

python

```python
def stem_and_lemmatize(tokens):
    stemmer = PorterStemmer()
    lemmatizer = WordNetLemmatizer()

    stemmed = [stemmer.stem(token) for token in tokens]
    lemmatized = [lemmatizer.lemmatize(token) for token in tokens]

    return stemmed, lemmatized

# Example
tokens = ['running', 'ran', 'easily', 'fairly']
stemmed, lemmatized = stem_and_lemmatize(tokens)
print(f"Original: {tokens}")
print(f"Stemmed: {stemmed}")
print(f"Lemmatized: {lemmatized}")
# Stemmed: ['run', 'ran', 'easili', 'fairli']
# Lemmatized: ['running', 'ran', 'easily', 'fairly']
```

5. Complete Text Preprocessing Pipeline

python

```python
def complete_preprocessing(text):
    # Clean text
    text = preprocess_text(text)

    # Tokenize
    tokens = tokenize_text(text)

    # Remove stopwords
    tokens = remove_stopwords(tokens)

    # Lemmatize
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return tokens

# Example
raw_text = "I am running quickly to the store! It's amazing how fast I can run."
processed = complete_preprocessing(raw_text)
print(f"Processed tokens: {processed}")
# Output: ['running', 'quickly', 'store', 'amazing', 'fast', 'run']
```

Web Scraping with Beautiful Soup

python

```python
from bs4 import BeautifulSoup
import requests

def scrape_website(url):
    try:
        # Send GET request
        response = requests.get(url)
        response.raise_for_status()

        # Parse HTML
        soup = BeautifulSoup(response.content, 'html.parser')

        # Extract text from paragraphs
        paragraphs = soup.find_all('p')
        text = ' '.join([p.get_text() for p in paragraphs])

        return text

    except requests.RequestException as e:
        print(f"Error scraping {url}: {e}")
        return None

# Example usage
# url = "https://example.com"
# scraped_text = scrape_website(url)
```

Named Entity Recognition (NER)

python

```python
import spacy

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

def extract_entities(text):
    doc = nlp(text)
    entities = []

    for ent in doc.ents:
        entities.append({
            'text': ent.text,
            'label': ent.label_,
            'description': spacy.explain(ent.label_)
        })

    return entities

# Example
text = "Apple Inc. was founded by Steve Jobs in Cupertino, California in 1976."
entities = extract_entities(text)
for entity in entities:
    print(f"{entity['text']} -> {entity['label']} ({entity['description']})")
# Apple Inc. -> ORG (Companies, agencies, institutions, etc.)
# Steve Jobs -> PERSON (People, including fictional)
# Cupertino -> GPE (Countries, cities, states)
```

WordCloud Generation

python

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

def generate_wordcloud(text, title="Word Cloud"):
    # Create WordCloud object
    wordcloud = WordCloud(
        width=800,
        height=400,
        background_color='white',
        max_words=100,
        colormap='viridis'
    ).generate(text)

    # Display
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(title)
    plt.axis('off')
    plt.show()

# Example
text = "machine learning artificial intelligence data science python programming"
generate_wordcloud(text, "NLP WordCloud")
```

🔢 Session 2: Vectorization & Word Embeddings

Count Vectorizer

python

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

def count_vectorizer_example():
    # Sample documents
    corpus = [
        'I love machine learning',
        'Machine learning is amazing',
        'I hate bad weather',
        'Weather is unpredictable'
    ]

    # Create CountVectorizer
    vectorizer = CountVectorizer()

    # Fit and transform
    count_matrix = vectorizer.fit_transform(corpus)

    # Get feature names
    feature_names = vectorizer.get_feature_names_out()

    # Convert to DataFrame for better visualization
    df = pd.DataFrame(count_matrix.toarray(), columns=feature_names)

    print("Count Vectorizer Results:")
    print(df)

    return vectorizer, count_matrix

count_vectorizer_example()
```

TF-IDF Vectorizer

python

```python
from sklearn.feature_extraction.text import TfidfVectorizer

def tfidf_example():
    corpus = [
        'I love machine learning',
        'Machine learning is amazing',
        'I hate bad weather',
        'Weather is unpredictable'
    ]

    # Create TF-IDF vectorizer
    tfidf_vectorizer = TfidfVectorizer()

    # Fit and transform
    tfidf_matrix = tfidf_vectorizer.fit_transform(corpus)

    # Get feature names
    feature_names = tfidf_vectorizer.get_feature_names_out()

    # Convert to DataFrame
    df = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)

    print("TF-IDF Results:")
    print(df.round(3))

    return tfidf_vectorizer, tfidf_matrix

tfidf_example()
```

Word2Vec Implementation

python

```python
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence
import numpy as np

def word2vec_example():
    # Sample sentences (tokenized)
    sentences = [
        ['machine', 'learning', 'is', 'fun'],
        ['deep', 'learning', 'is', 'powerful'],
        ['natural', 'language', 'processing', 'is', 'interesting'],
        ['python', 'programming', 'is', 'useful'],
        ['data', 'science', 'is', 'exciting']
    ]

    # Train Word2Vec model
    # sg=1: Skip-gram, sg=0: CBOW
    model = Word2Vec(
        sentences=sentences,
        vector_size=100,    # Embedding dimension
        window=5,           # Context window size
        min_count=1,        # Minimum word frequency
        sg=1,               # Skip-gram
        workers=4
    )

    # Get word vector
    try:
        vector = model.wv['learning']
        print(f"Vector for 'learning': {vector[:5]}...")  # Show first 5 dimensions

        # Find similar words
        similar_words = model.wv.most_similar('learning', topn=3)
        print(f"Words similar to 'learning': {similar_words}")

        # Calculate similarity
        similarity = model.wv.similarity('machine', 'learning')
        print(f"Similarity between 'machine' and 'learning': {similarity:.3f}")

    except KeyError as e:
        print(f"Word not in vocabulary: {e}")

    return model

model = word2vec_example()
```

Using Pre-trained Word Embeddings

python

```python
import gensim.downloader as api

def load_pretrained_embeddings():
    # Load pre-trained GloVe embeddings
    # This might take some time for first download
    print("Loading pre-trained embeddings...")

    # Available models: 'word2vec-google-news-300', 'glove-wiki-gigaword-100', etc.
    model = api.load("glove-wiki-gigaword-50")

    # Example usage
    try:
        # Get word vector
        vector = model['computer']
        print(f"Vector for 'computer': {vector[:5]}...")

        # Find similar words
        similar = model.most_similar('computer', topn=5)
        print("Words similar to 'computer':")
        for word, score in similar:
            print(f"  {word}: {score:.3f}")

        # Word analogy: king - man + woman = ?
        result = model.most_similar(positive=['king', 'woman'], negative=['man'], topn=1)
        print(f"king - man + woman = {result[0][0]}")

    except KeyError as e:
        print(f"Word not found: {e}")

# Uncomment to run (requires internet connection)
# load_pretrained_embeddings()
```

Keras Text Processing

python

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
import tensorflow as tf

def keras_text_processing():
    # Sample texts
    texts = [
        'I love this movie',
        'This movie is terrible',
        'Great acting and plot',
        'Boring and predictable',
        'Amazing cinematography'
    ]

    labels = [1, 0, 1, 0, 1]  # 1: positive, 0: negative

    # Tokenize texts
    tokenizer = Tokenizer(num_words=1000, oov_token='<OOV>')
    tokenizer.fit_on_texts(texts)

    # Convert to sequences
    sequences = tokenizer.texts_to_sequences(texts)
    print(f"Sequences: {sequences}")

    # Pad sequences
    padded = pad_sequences(sequences, maxlen=10, padding='post')
    print(f"Padded sequences:\n{padded}")

    # Word index
    word_index = tokenizer.word_index
    print(f"Word index: {word_index}")

    return tokenizer, padded, labels

tokenizer, sequences, labels = keras_text_processing()
```

Sentiment Analysis Example

python

```python
def simple_sentiment_model():
    # Build simple sentiment model
    model = Sequential([
        Embedding(input_dim=1000, output_dim=16, input_length=10),
        LSTM(32),
        Dense(1, activation='sigmoid')
    ])

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

    print(model.summary())

    # Note: This is just model architecture
    # For actual training, you'd need more data
    return model

model = simple_sentiment_model()
```

🔄 Session 3: Sequence-to-Sequence Learning

Simple RNN Implementation

python

```python
import tensorflow as tf
from tensorflow.keras.layers import SimpleRNN, Dense, Embedding
from tensorflow.keras.models import Sequential
import numpy as np

def simple_rnn_example():
    # Create sample data
    vocab_size = 1000
    embedding_dim = 64
    max_length = 20

    model = Sequential([
        Embedding(vocab_size, embedding_dim, input_length=max_length),
        SimpleRNN(50, return_sequences=True),  # return_sequences for stacked RNNs
        SimpleRNN(50),
        Dense(1, activation='sigmoid')
    ])

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

    print("Simple RNN Architecture:")
    print(model.summary())

    return model

rnn_model = simple_rnn_example()
```

LSTM Implementation

python

```python
from tensorflow.keras.layers import LSTM

def lstm_example():
    vocab_size = 1000
    embedding_dim = 64
    max_length = 20

    model = Sequential([
        Embedding(vocab_size, embedding_dim, input_length=max_length),
        LSTM(50, return_sequences=True, dropout=0.2),
        LSTM(50, dropout=0.2),
        Dense(1, activation='sigmoid')
    ])

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

    print("LSTM Architecture:")
    print(model.summary())

    return model

lstm_model = lstm_example()
```

Bidirectional LSTM

python

```python
from tensorflow.keras.layers import Bidirectional

def bidirectional_lstm_example():
    vocab_size = 1000
    embedding_dim = 64
    max_length = 20

    model = Sequential([
        Embedding(vocab_size, embedding_dim, input_length=max_length),
        Bidirectional(LSTM(50, return_sequences=True, dropout=0.2)),
        Bidirectional(LSTM(50, dropout=0.2)),
        Dense(1, activation='sigmoid')
    ])

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

    print("Bidirectional LSTM Architecture:")
    print(model.summary())

    return model

bi_lstm_model = bidirectional_lstm_example()
```

Seq2Seq Model for Translation

python

```python
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.models import Model

def seq2seq_model():
    # Encoder-Decoder architecture
    latent_dim = 256

    # Encoder
    encoder_inputs = Input(shape=(None,))
    encoder_embedding = Embedding(1000, latent_dim)(encoder_inputs)
    encoder_lstm = LSTM(latent_dim, return_state=True)
    encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
    encoder_states = [state_h, state_c]

    # Decoder
    decoder_inputs = Input(shape=(None,))
    decoder_embedding = Embedding(1000, latent_dim)(decoder_inputs)
    decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
    decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
    decoder_dense = Dense(1000, activation='softmax')
    decoder_outputs = decoder_dense(decoder_outputs)

    # Model
    model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

    model.compile(
        optimizer='rmsprop',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    print("Seq2Seq Model Architecture:")
    print(model.summary())

    return model

seq2seq = seq2seq_model()
```

Text Generation with LSTM

python

```python
def text_generation_model():
    # Sample text data preparation
    text = "hello world this is a sample text for generation"
    chars = sorted(list(set(text)))
    char_to_idx = {ch: i for i, ch in enumerate(chars)}
    idx_to_char = {i: ch for i, ch in enumerate(chars)}

    # Prepare sequences
    sequence_length = 10
    sequences = []
    next_chars = []

    for i in range(len(text) - sequence_length):
        sequences.append([char_to_idx[ch] for ch in text[i:i + sequence_length]])
        next_chars.append(char_to_idx[text[i + sequence_length]])

    X = np.array(sequences)
    y = tf.keras.utils.to_categorical(next_chars, num_classes=len(chars))

    # Model
    model = Sequential([
        Embedding(len(chars), 50, input_length=sequence_length),
        LSTM(128),
        Dense(len(chars), activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    print("Text Generation Model:")
    print(model.summary())

    return model, char_to_idx, idx_to_char

gen_model, char_to_idx, idx_to_char = text_generation_model()
```

⚡ Session 4: Transformers

Self-Attention Mechanism

python

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

def scaled_dot_product_attention(q, k, v, mask=None):
    """
    Calculate scaled dot-product attention
    """
    # Calculate attention scores
    matmul_qk = tf.matmul(q, k, transpose_b=True)

    # Scale by square root of key dimension
    dk = tf.cast(tf.shape(k)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

    # Apply mask if provided
    if mask is not None:
        scaled_attention_logits += (mask * -1e9)

    # Apply softmax
    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)

    # Apply attention to values
    output = tf.matmul(attention_weights, v)

    return output, attention_weights

# Example usage
def attention_example():
    # Sample input (batch_size, seq_length, d_model)
    batch_size, seq_length, d_model = 2, 4, 8

    # Random Q, K, V matrices
    q = tf.random.normal((batch_size, seq_length, d_model))
    k = tf.random.normal((batch_size, seq_length, d_model))
    v = tf.random.normal((batch_size, seq_length, d_model))

    output, weights = scaled_dot_product_attention(q, k, v)

    print(f"Input shape: {q.shape}")
    print(f"Output shape: {output.shape}")
    print(f"Attention weights shape: {weights.shape}")

    return output, weights

output, weights = attention_example()
```

Multi-Head Attention

python

```
class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
        self.d_model = d_model

        assert d_model % self.num_heads == 0

        self.depth = d_model // self.num_heads

        self.wq = tf.keras.layers.Dense(d_model)
        self.wk = tf.keras.layers.Dense(d_model)
        self.wv = tf.keras.layers.Dense(d_model)

        self.dense = tf.keras.layers.Dense(d_model)

    def split_heads(self, x, batch_size):
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
        return tf.transpose(x, perm=[0, 2, 1, 3])

    def call(self, v, k, q, mask=None):
        batch_size = tf.shape(q)[0]

        q = self.wq(q)
        k = self.wk(k)
        v = self.wv(v)

        q = self.split_heads(q, batch_size)
        k = self.split_heads(k, batch_size)
        v = self.split_heads(v, batch_size)

        scaled_attention, attention_weights = scaled_dot_product_attention(
            q, k, v, mask)

        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])

        concat_attention = tf.reshape(scaled_attention,
                                      (batch_size, -1, self.d_model))

        output = self.dense(concat_attention)

        return output, attention_weights

# Example usage
def multi_head_attention_example():
    d_model = 512
```

```
    num_heads = 8

    mha = MultiHeadAttention(d_model, num_heads)

    # Sample input
    batch_size, seq_length = 2, 10
    x = tf.random.normal((batch_size, seq_length, d_model))

    output, weights = mha(x, x, x)

    print(f"Multi-head attention output shape: {output.shape}")
    print(f"Attention weights shape: {weights.shape}")

    return output, weights

mha_output, mha_weights = multi_head_attention_example()
```

Transformer Block

python

```
def point_wise_feed_forward_network(d_model, dff):
    return tf.keras.Sequential([
        tf.keras.layers.Dense(dff, activation='relu'),
        tf.keras.layers.Dense(d_model)
    ])

class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, dff, rate=0.1):
        super(TransformerBlock, self).__init__()

        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask=None):
        attn_output, _ = self.mha(x, x, x, mask)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(x + attn_output)

        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        out2 = self.layernorm2(out1 + ffn_output)

        return out2

# Example Transformer model
def simple_transformer_model():
    d_model = 512
    num_heads = 8
    dff = 2048
    vocab_size = 10000
    max_seq_length = 100

    inputs = tf.keras.Input(shape=(max_seq_length,))

    # Embedding and positional encoding
    embedding = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
    embedding *= tf.math.sqrt(tf.cast(d_model, tf.float32))

    # Add positional encoding (simplified)
    pos_encoding = tf.range(max_seq_length, dtype=tf.float32)
```

```
    pos_encoding = tf.reshape(pos_encoding, [1, -1, 1])
    embedding += pos_encoding

    # Transformer blocks
    x = TransformerBlock(d_model, num_heads, dff)(embedding)
    x = TransformerBlock(d_model, num_heads, dff)(x)

    # Global average pooling and classification
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    outputs = tf.keras.layers.Dense(2, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    print("Simple Transformer Model:")
    print(model.summary())

    return model

transformer_model = simple_transformer_model()
```

Using Hugging Face Transformers

python

```python
from transformers import AutoTokenizer, AutoModel, pipeline
import torch

def hugging_face_examples():
    # 1. Using pre-trained BERT
    print("1. BERT Example:")
    tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
    model = AutoModel.from_pretrained('bert-base-uncased')

    text = "Hello, how are you today?"
    tokens = tokenizer(text, return_tensors='pt')
    print(f"Tokens: {tokens}")

    with torch.no_grad():
        outputs = model(**tokens)
        embeddings = outputs.last_hidden_state
        print(f"BERT embeddings shape: {embeddings.shape}")

    # 2. Sentiment Analysis Pipeline
    print("\n2. Sentiment Analysis:")
    sentiment_pipeline = pipeline("sentiment-analysis")

    texts = [
        "I love this product!",
        "This is terrible.",
        "It's okay, nothing special."
    ]

    results = sentiment_pipeline(texts)
    for text, result in zip(texts, results):
        print(f"'{text}' -> {result}")

    # 3. Text Generation
    print("\n3. Text Generation:")
    generator = pipeline("text-generation", model="gpt2")

    prompt = "The future of artificial intelligence is"
    generated = generator(prompt, max_length=50, num_return_sequences=1)

    for text in generated:
        print(f"Generated: {text['generated_text']}")

    # 4. Question Answering
    print("\n4. Question Answering:")
    qa_pipeline = pipeline("question-answering")
```

```
    context = "Python is a programming language. It was created by Guido van Rossum."
    question = "Who created Python?"

    answer = qa_pipeline(question=question, context=context)
    print(f"Q: {question}")
    print(f"A: {answer['answer']} (confidence: {answer['score']:.3f})")

# Uncomment to run (requires transformers and torch)
# hugging_face_examples()
```

🤖 Session 5: Generative AI & Large Language Models

Working with OpenAI GPT (Example Structure)

python

```python
# Note: This requires OpenAI API key
# pip install openai

import openai
from typing import List, Dict

class GPTWrapper:
    def __init__(self, api_key: str):
        openai.api_key = api_key

    def generate_text(self, prompt: str, max_tokens: int = 100) -> str:
        try:
            response = openai.Completion.create(
                engine="text-davinci-003",
                prompt=prompt,
                max_tokens=max_tokens,
                temperature=0.7
            )
            return response.choices[0].text.strip()
        except Exception as e:
            return f"Error: {e}"

    def chat_completion(self, messages: List[Dict]) -> str:
        try:
            response = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",
                messages=messages,
                max_tokens=150
            )
            return response.choices[0].message.content
        except Exception as e:
            return f"Error: {e}"

# Example usage (requires API key)
def gpt_examples():
    # gpt = GPTWrapper("your-api-key-here")

    # Text generation
    prompt = "Explain machine learning in simple terms:"
    # generated = gpt.generate_text(prompt)
    # print(f"Generated: {generated}")

    # Chat completion
    messages = [
        {"role": "system", "content": "You are a helpful AI assistant."},
        {"role": "user", "content": "What is natural language processing?"}
```

```
    ]
    # response = gpt.chat_completion(messages)
    # print(f"Chat response: {response}")

    print("GPT examples require API key to run")

gpt_examples()
```

Prompt Engineering Techniques

python

```
def prompt_engineering_examples():
    """
    Examples
```