```
text="""
Training a large-scale AI model like amazing DeepSeek (Giving us hope
that technology is making great moves everyday) from scratch at a
reduced cost involves implementing innovative strategies across data
management, model architecture, hardware utilization, and training
methodologies. DeepSeek’s approach provides valuable insights into
achieving high performance with cost efficiency. Here’s how you can
emulate their success:

. Innovative Model Architecture
Mixture of Experts (MoE): DeepSeek employed a “mixture of experts”
approach, distributing tasks among specialized sub-models rather than
relying on a single, monolithic model. This strategy reduces
computational load and enhances efficiency.
Multi-Stage Training: Implement a phased training process, where each
stage focuses on specific improvements such as accuracy or alignment.
For instance, start with general text data pretraining, followed by
reinforcement learning on user feedback to enhance conversational
abilities.
Avoid R&D cost: Implementing something for the second time is
significantly cheaper since minimal research and development (R&D) is
required. R&D is typically the biggest expense in any innovation. Take
the invention of the light bulb or mobile phone as an example –
initially, they were sold at extremely high prices due to the
substantial R&D costs. However, over time, as development costs were
recovered, prices dropped, making them affordable for as little as $50
per mobile phone. Initially, selling them at such a low price was
impossible due to the high investment in R&D and as OPENAI is still
under their R&D process of achieving more great things, so they can
not still make them opensource or lower cost that much, as they have
their own R&D cost of doing things first time in the world. Once the
thing is done first time, there are many researches available done by
universities and institutes and are opensource, those can really
reduce the company cost of R&D.
2. Efficient Data Management
High-Quality Curated Datasets: Utilize well-curated datasets to ensure
the model learns from accurate and relevant information, reducing the
need for extensive data cleaning and processing.
Synthetic Data Generation: Generate synthetic data to augment training
datasets, especially in specialized domains, to enhance model
robustness without incurring high data acquisition costs.
3. Hardware Optimization
Leverage Available Hardware: DeepSeek capitalized on available high-
performance chips, such as Nvidia’s H800, to optimize their training
process. Assess the hardware at your disposal and optimize your
training algorithms to make the most of it.
Mixed-Precision Training: Implement mixed-precision arithmetic to
reduce memory usage and increase computational speed without
compromising model accuracy.
```

```
4. Cost-Effective Training Strategies
Programming Shortcuts: DeepSeek utilized innovative programming
shortcuts to reduce data-processing requirements, significantly
cutting down training costs. Explore algorithmic optimizations that
can streamline computations.
Open-Source Tools: Leverage open-source frameworks and tools to build
and train your models, minimizing software development costs.
To make training a model like DeepSeek even cheaper, you can focus on
reducing computational costs, optimizing data processing, and
leveraging alternative training approaches. Here are additional cost-
saving techniques:




"""

from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Sample text


# Generate Word Cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text)

# Display the Word Cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```


```
df
```

{"type":"dataframe","variable_name":"df"}

```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Load the Excel file
file_path = "/content/Person_Data.xlsx"  # Update with the correct
file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Combine all attribute columns into a single text corpus
text_data = df.iloc[:, 2:].astype(str).values.flatten()  # Extract
only attribute columns
text_corpus = " ".join(text_data)  # Convert to a single text string

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text_corpus)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```


```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import pandas as pd

# Load the Excel file
file_path = "/content/Person_Data.xlsx"  # Update with the correct file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Combine all attribute columns into a single text corpus
text_data = df.iloc[:, 2:].astype(str).values.flatten()  # Extract only attribute columns
text_corpus = " ".join(text_data)  # Convert to a single text string

# Define a list of words to remove
stopwords = ["nan", "years", "those", "who", "are", "not", "is", "a", "the", "of", "to", "in", "and" , "kg","Type","Married","Telgu","AP","Emp","Mov","Job","new","Glasses","55 years","sister","non vegetarian","eye","Upma","Hobbies","Feet","Age","thing","Bindi","Govt","Song","Vijaywada","makes","Cm"]

# Generate the word cloud, excluding specified words
wordcloud = WordCloud(width=800, height=400, background_color='black', stopwords=stopwords, colormap='plasma').generate(text_corpus) # Added colormap

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
```

```
plt.axis("off")
plt.show()
```

Word Embedding

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
True
```

```
import pandas as pd
import re
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import nltk
```

```
# Download necessary NLP resources


# Load the dataset
file_path = "/content/Person_Data.xlsx"  # Update with correct path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Combine all attribute columns into a single text corpus
text_data = df.iloc[:, 2:].astype(str).values.flatten()
text_corpus = " ".join(text_data)

# Preprocessing function
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = word_tokenize(text)  # Tokenization
    words = [word for word in words if word not in
stopwords.words('english')]  # Remove stopwords
    return words

# Tokenize the text data
tokenized_text = [preprocess_text(text_corpus)]

# Train Word2Vec model
model = Word2Vec(sentences=tokenized_text, vector_size=100, window=5,
min_count=1, workers=4)

# Example: Get the vector for a specific word
word = "funny"
if word in model.wv:
    print(f"Word Embedding for '{word}':\n", model.wv[word])
else:
    print(f"'{word}' not in vocabulary.")
```

```
Word Embedding for 'funny':
 [-1.35412747e-02  1.34696485e-02  5.80059132e-03  1.54062361e-02
  4.15413873e-03 -9.77190025e-03  1.44205149e-02  1.15330480e-02
 -1.39973098e-02  2.27990351e-03  7.01297307e-03 -1.51008954e-02
  6.75419951e-03  2.13081785e-03  9.29751247e-03 -5.16930036e-03
 -4.11134120e-03 -6.18518470e-03 -1.49353808e-02 -2.05062348e-02
  7.93468114e-03  1.04032177e-02  1.62800662e-02 -9.89919715e-03
  3.83141125e-03  1.12242869e-03 -5.43676177e-03 -1.15296058e-02
 -8.72911979e-03 -1.13410782e-02  9.26283188e-03 -4.64141415e-03
  1.26346340e-02  1.85411295e-03 -7.00963102e-03  1.20512489e-02
  3.77075025e-03 -4.08245505e-05 -8.87390343e-04 -2.24144906e-02
 -6.86466601e-03  2.43294242e-04 -6.68100221e-03 -3.94213479e-03
  6.07200572e-03 -3.58241796e-03 -4.94506722e-03  3.80493631e-03
  2.01061764e-03  2.88210716e-03  2.42949766e-03  1.38200819e-03
 -7.86697783e-04 -4.13587410e-03  4.56904247e-03 -2.41229404e-03
  5.06435754e-03  2.30270103e-04 -4.38500009e-03 -8.58711312e-04
```

```
 -7.36160018e-03 -5.85627276e-03  1.06771924e-02 -1.03852805e-02
 -1.96005125e-02  6.04619272e-03 -1.71658897e-03  1.61374565e-02
 -1.96585432e-02  1.21542485e-02 -8.09338596e-03  1.06564043e-02
  9.00491234e-03  5.53255482e-03  3.99842905e-03  6.85977375e-06
 -2.17792392e-03 -9.85410158e-03 -1.61376260e-02 -4.15380858e-03
 -5.62671758e-03 -7.68851861e-03  1.81067106e-03  8.73718038e-03
  1.63270521e-03  9.81622841e-04  4.66932124e-03 -6.58745319e-03
  4.94372426e-03  9.56346840e-03  6.26955740e-03  5.40368026e-03
  5.02691371e-03  1.16730342e-03  2.14569941e-02  1.78989489e-02
  7.21993332e-04  5.31508937e-04  6.46259030e-03 -5.74859418e-03]
```

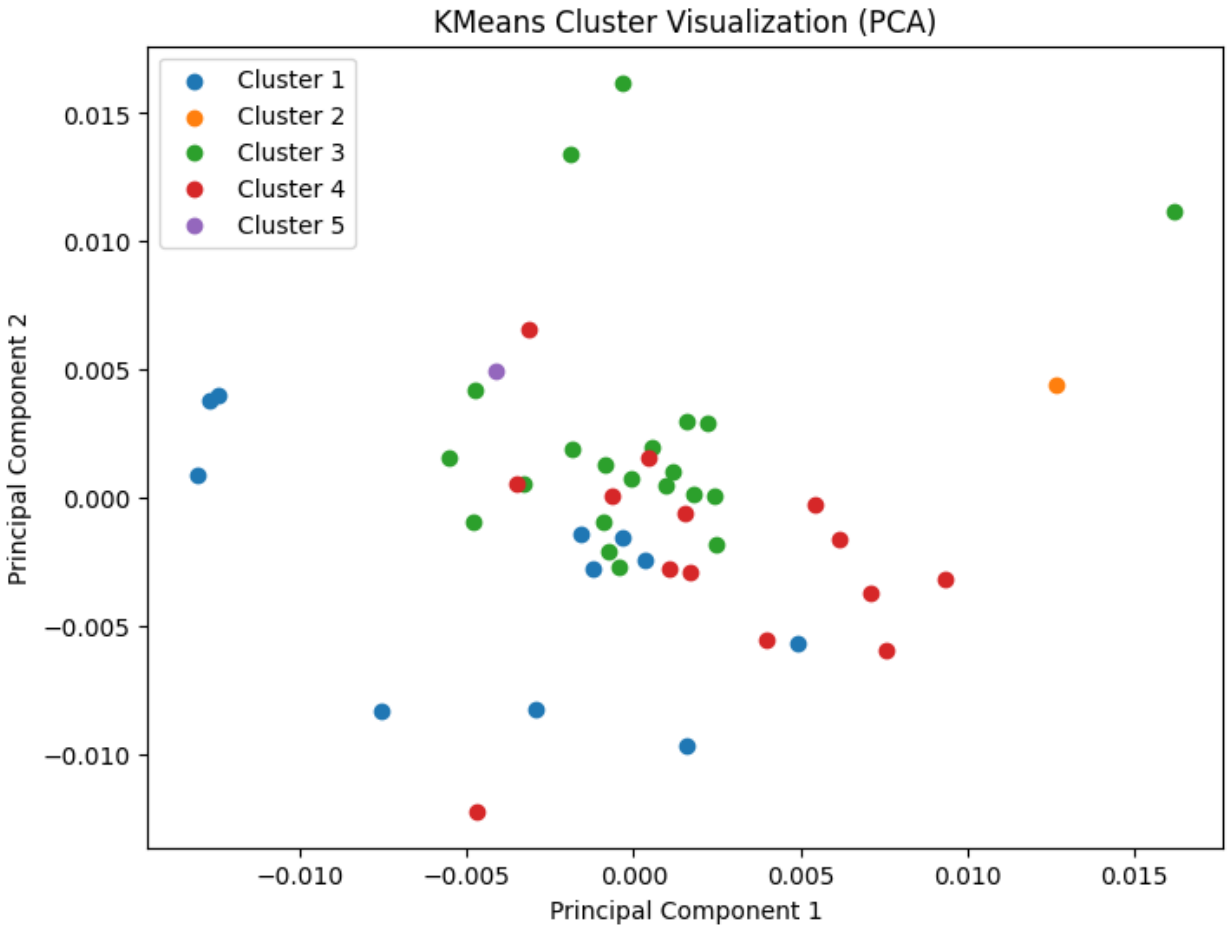
```python
import numpy as np

class KMeans:
    def __init__(self, k=5, max_iters=100, tol=1e-4):
        self.k = k  # Number of clusters
        self.max_iters = max_iters  # Maximum iterations
        self.tol = tol  # Convergence tolerance
        self.centroids = None  # Cluster centroids
        self.labels = None  # Cluster assignments

    def fit(self, X):
        np.random.seed(42)  # Set seed for reproducibility
        self.centroids = X[np.random.choice(len(X), self.k,
replace=False)]  # Initialize centroids

        for iteration in range(self.max_iters):
            labels = self._assign_clusters(X)  # Assign points to the
nearest centroid
            new_centroids = self._compute_new_centroids(X, labels)  #
Compute new centroids

            # Check for convergence
            if np.linalg.norm(self.centroids - new_centroids) <
self.tol:
                print(f"Converged after {iteration} iterations.")
                break

            self.centroids = new_centroids  # Update centroids

        self.labels = labels  # Store final cluster assignments

    def _assign_clusters(self, X):
        return np.array([np.argmin(np.linalg.norm(point -
self.centroids, axis=1)) for point in X])  # Assign clusters

    def _compute_new_centroids(self, X, labels):
        return np.array([X[labels == i].mean(axis=0) for i in
range(self.k)])  # Compute new centroids
```

```
    def predict(self, X):
        return self._assign_clusters(X)  # Predict cluster labels for
new data

    def get_centroids(self):
        return self.centroids  # Return final centroids

import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from gensim.models import Word2Vec
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans  # Importing KMeans

# Download necessary resources
nltk.download('stopwords')

# Load the Excel file
file_path = "/content/Person_Data.xlsx"  # Update with the correct
file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Text Preprocessing
text_data = df.iloc[:, 2:].fillna("").astype(str).apply(lambda x: "
".join(x), axis=1).tolist()

custom_stopwords = {"nan", "years", "those", "who", "are", "not",
"is", "a", "the", "of", "to", "in", "and" ,
"kg","Type","Married","Telgu","AP","Emp","Mov","Job","new","Glasses","
55 years","sister","non
vegetarian","eye","Upma","Hobbies","Feet","Age","thing","Bindi","Govt"
,"Song","Vijaywada","makes","Cm"}

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = text.split()  # Split text into words based on whitespace
    all_stopwords =
set(stopwords.words('english')).union(custom_stopwords)
    words = [word for word in words if word not in all_stopwords and
len(word) > 1]
    return words

# Filter out empty texts
preprocessed_text = [preprocess_text(text) for text in text_data if
```

```
text.strip()]

# Train Word2Vec Model
model = Word2Vec(sentences=preprocessed_text, vector_size=100,
window=5, min_count=1, workers=4)

# Generate Document Vectors
document_vectors = []
cleaned_corpus = []

for document in preprocessed_text:
    word_vectors = [model.wv[word] for word in document if word in
model.wv]
    if word_vectors:
        document_vectors.append(np.mean(word_vectors, axis=0))
        cleaned_corpus.append(document)
    else:
        document_vectors.append(np.zeros(model.vector_size))
        cleaned_corpus.append([])

X = np.array(document_vectors)

# Apply KMeans Clustering
k = 5
kmeans = KMeans(n_clusters=k, max_iter=100, tol=1e-4)  # Corrected to
use n_clusters instead of k
kmeans.fit(X)

# Function to get top words per cluster
def get_top_words(centroid, model, top_n=10):
    words = list(model.wv.index_to_key)
    word_vectors = np.array([model.wv[word] for word in words])
    similarities = cosine_similarity([centroid], word_vectors)[0]
    top_indices = similarities.argsort()[-top_n:][::-1]
    return [words[i] for i in top_indices]

# Generate Word Clouds for Each Cluster
fig, axes = plt.subplots(1, k, figsize=(15, 5))

for i in range(k):
    cluster_indices = np.where(kmeans.labels_ == i)[0]  # Corrected to
use labels_ instead of labels

    cluster_words = []
    for index in cluster_indices:
        cluster_words.extend(cleaned_corpus[index])

    word_freq = " ".join(cluster_words)

    wordcloud = WordCloud(width=400, height=400,
```

```
background_color="white").generate(word_freq)

    axes[i].imshow(wordcloud, interpolation="bilinear")
    axes[i].axis("off")
    axes[i].set_title(f"Cluster {i+1}")

plt.show()

# PCA for Cluster Visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.figure(figsize=(8, 6))
for i in range(k):
    plt.scatter(X_pca[kmeans.labels_ == i, 0], X_pca[kmeans.labels_ ==
i, 1], label=f'Cluster {i+1}')

plt.legend()
plt.title("KMeans Cluster Visualization (PCA)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```


KMeans Cluster Visualization (PCA)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Principal Component 2
Principal Component 1
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.010
−0.005
0.000
0.005
0.010
0.015