Data Cleaning

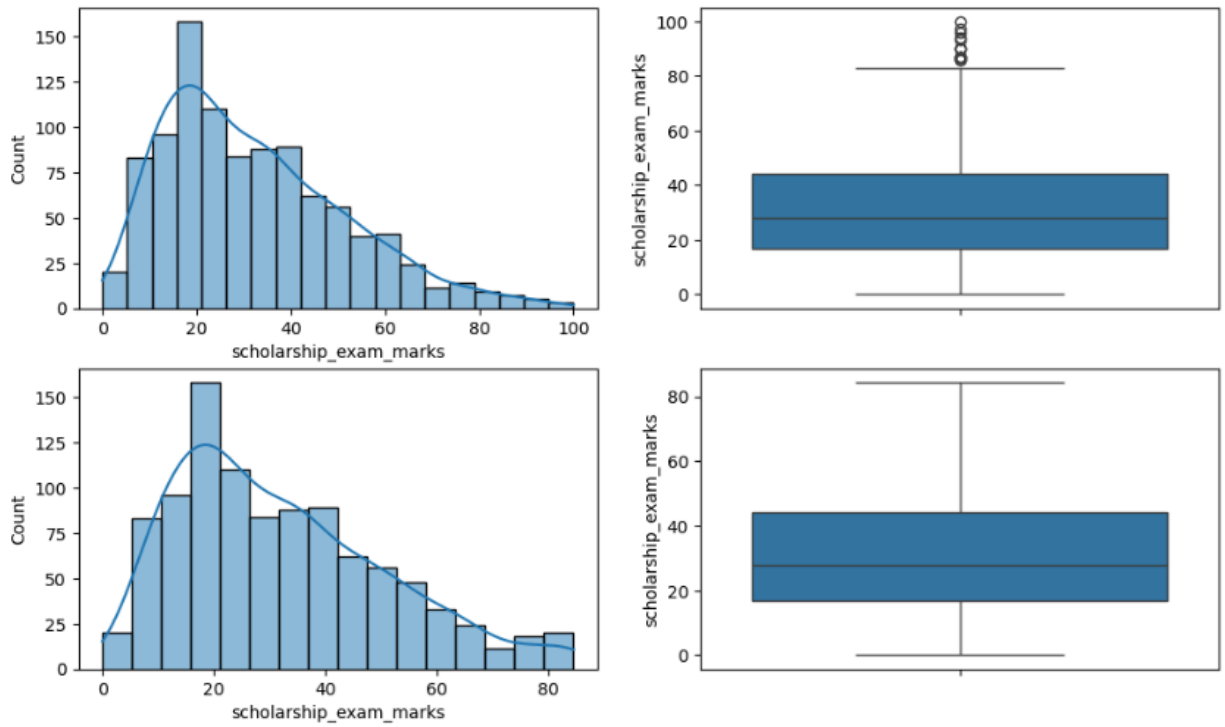
Outliers

Removing by IQR

```
capped = df.copy()
```

```
capped['scholarship_exam_marks'] = np.where(df['scholarship_exam_marks'] > max_allowed, max_allowed,
                                            np.where(df['scholarship_exam_marks'] < low_allowed, low_allowed,
                                                     df['scholarship_exam_marks']))
```

```
plt.figure(figsize = (12, 7))

plt.subplot(2, 2, 1)
sns.histplot(df['scholarship_exam_marks'], kde = True)
plt.subplot(2, 2, 2)
sns.boxplot(df['scholarship_exam_marks'])

plt.subplot(2, 2, 3)
sns.histplot(capped['scholarship_exam_marks'], kde = True)
plt.subplot(2, 2, 4)
sns.boxplot(capped['scholarship_exam_marks'])

plt.show()
```



```
Q1 = df['scholarship_exam_marks'].quantile(.25)
print('Q1  :', Q1)
Q3 = df['scholarship_exam_marks'].quantile(.75)
print('Q3  :', Q3)

IQR = Q3 - Q1
print('IQR :', IQR)
```

```
Q1  : 17.0
Q3  : 44.0
IQR : 27.0
```

```
max_allowed = Q3 + 1.5 * IQR
low_allowed = Q1 - 1.5 * IQR
```

```
outliers_iqr = df[(df['scholarship_exam_marks'] < low_allowed) | (df['scholarship_exam_marks'] > max_allowed)]
```

Feature Engineering

Encoding Categorical variables

One-Hot Encoder

One Hot Encoding is a method for converting categorical variables into a binary format. It creates new binary columns (0s and 1s) for each category in the original variable. Each category in the original column is represented as a separate column, where a value of 1 indicates the presence of that category, and 0 indicates its absence.

```
data = [['Blue'], ['Red'], ['Orange'], ['Black'], ['Blue'], ['Red'], ['Red']]
```

```
onehot_encoder = OneHotEncoder(sparse_output = False)
```

```
onehot_encoded = onehot_encoder.fit_transform(data)
```

```
print(onehot_encoded)
```

```
[[0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]]
```

Label Encoder

Each unique category in the feature is mapped to an integer value. For example, if you have a feature like room_type with categories like "Single", "Double", and "Suite", label encoding will convert them into integers like 1,2,3 etc.

```
label_encoder = LabelEncoder()
```

```
data_labeled = label_encoder.fit_transform(data)
data_labeled
```

```
C:\Users\kalat\anaconda3\Lib\site-packages\sklearn\preprocessing\_
xpected. Please change the shape of y to (n_samples, ), for exampl
  y = column_or_1d(y, warn=True)
```

```
array([1, 3, 2, 0, 1, 3, 3], dtype=int64)
```

```
df3 = df.copy()
```

```
label_encoded = label_encoder.fit_transform(df3['room_type'])
label_encoded
```

```
array([1, 0, 1, ..., 1, 0, 0])
```

Scaling Numerical Data

Standard Scaler

The StandardScaler is a feature scaling technique that standardizes the features of a dataset by removing the mean and scaling them to unit variance. This ensures that each feature has a mean of 0 and a standard deviation of 1. Standardization is especially useful when the features have different units or varying scales, as it brings them to a common scale, which helps many machine learning algorithms perform better.

```
df4 = df.copy()
```

```
std_scaler = StandardScaler()
```

```
std_scaled = std_scaler.fit_transform(df4[['price']])
```

```
std_scaled
```

```
array([[-0.01545572],
       [ 0.30104445],
       [-0.01129124],
       ...,
       [-0.28198217],
       [-0.13622552],
       [-0.052936  ]])
```

MinMax Scaler

The MinMaxScaler is another feature scaling technique that transforms features by scaling them into a specific range, typically between 0 and 1. It works by subtracting the minimum value of the feature and dividing by the range (difference between the maximum and minimum values), so that the transformed values are in the range [0, 1].

```
df5 = df.copy()
```

```
minmax_scaled = MinMaxScaler()
```

```
minmax_scaled = minmax_scaled.fit_transform(df5[['price']])
```

```
minmax_scaled
```

```
array([[0.0149],
       [0.0225],
       [0.015 ],
       ...,
       [0.0085],
       [0.012 ],
       [0.014 ]])
```

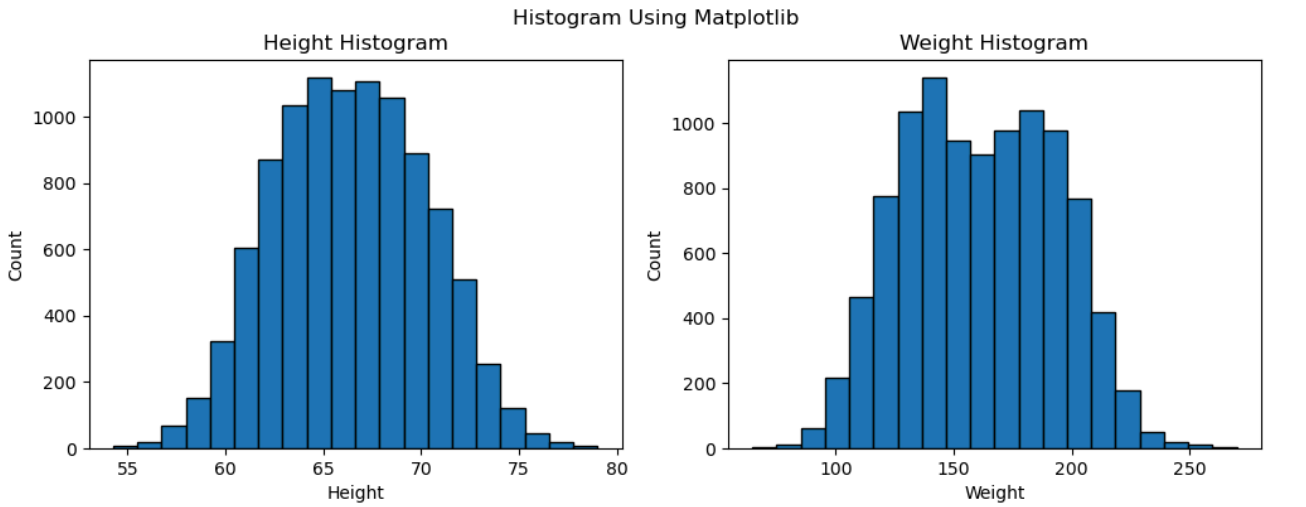
Exploratory Data Analysis

Visualizations

Histogram

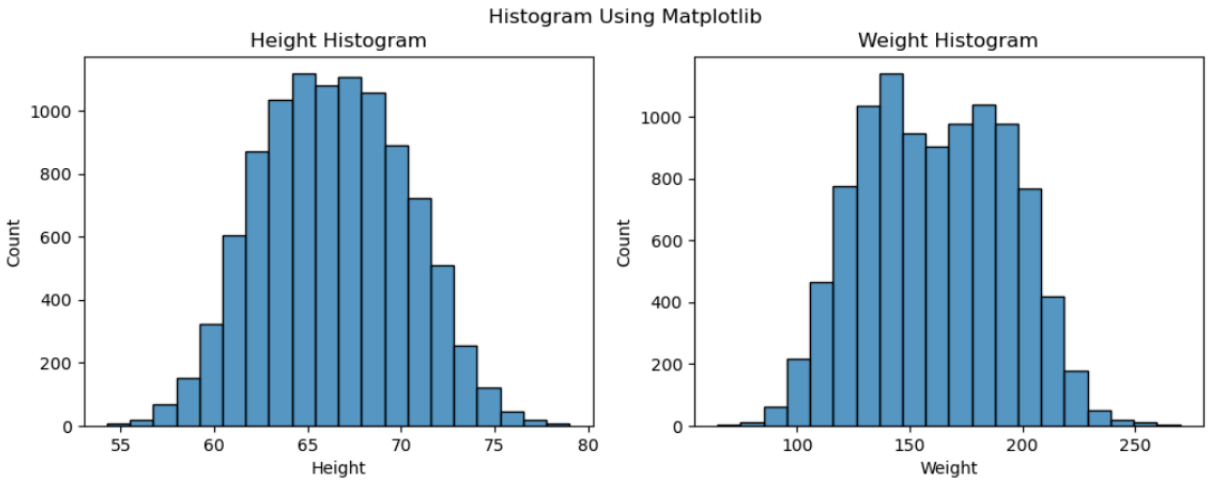
- Matplotlib

```
plt.figure(figsize = (12, 4))
plt.subplot(1, 2, 1)
plt.hist(data['Height'], bins = 20, edgecolor = 'black')
plt.xlabel('Height')
plt.ylabel('Count')
plt.title('Height Histogram')

plt.subplot(1, 2, 2)
plt.hist(data['Weight'], bins = 20, edgecolor = 'black')
plt.xlabel('Weight')
plt.ylabel('Count')
plt.title('Weight Histogram')

plt.suptitle('Histogram Using Matplotlib')
plt.show()
```



- Seaborn

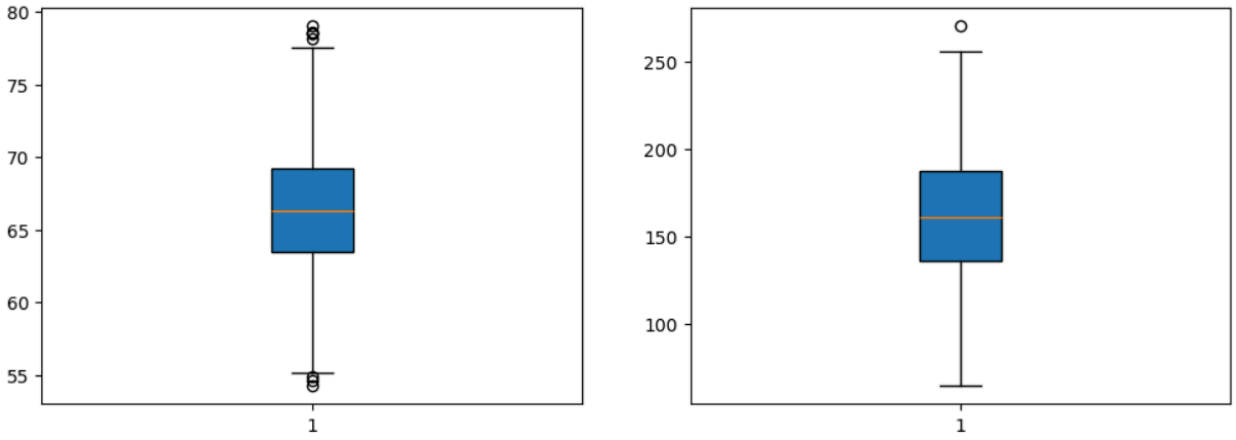
```
plt.figure(figsize = (12, 4))

plt.subplot(1, 2, 1)
sns.histplot(data['Height'], bins = 20)
plt.title('Height Histogram')

plt.subplot(1, 2, 2)
sns.histplot(data['Weight'], bins = 20)
plt.title('Weight Histogram')

plt.suptitle('Histogram Using Matplotlib')
plt.show()
```



Box Plot

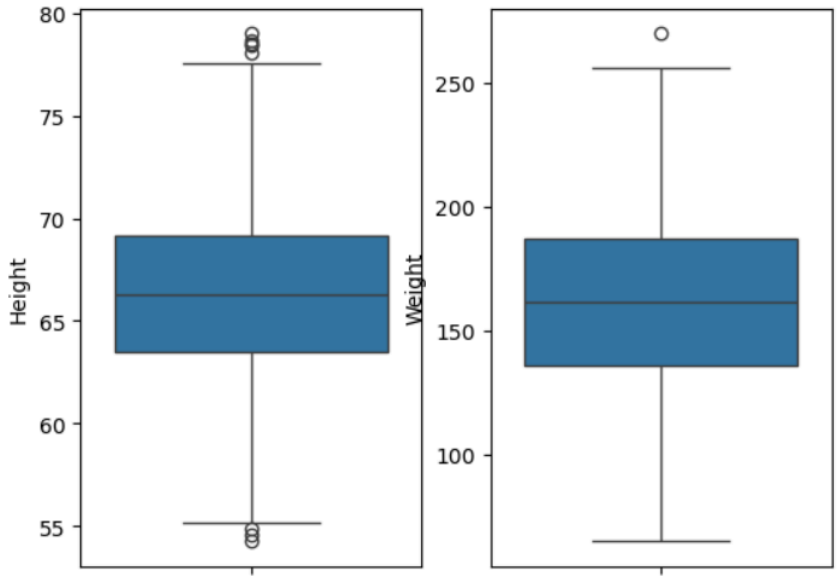
- Matplotlib

```
plt.figure(figsize = (12, 4))

plt.subplot(1, 2, 1)
plt.boxplot(data['Height'], patch_artist = True)

plt.subplot(1, 2, 2)
plt.boxplot(data['Weight'], patch_artist = True)
plt.show()
```



- Seaborn

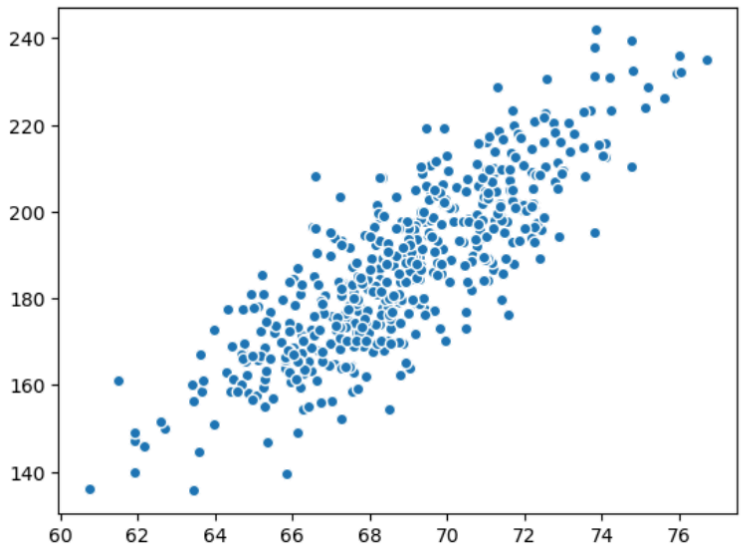
```
plt.subplot(1, 2, 1)
sns.boxplot(data['Height'])

plt.subplot(1, 2, 2)
sns.boxplot(data['Weight'])
plt.show()
```



Scatter Plot

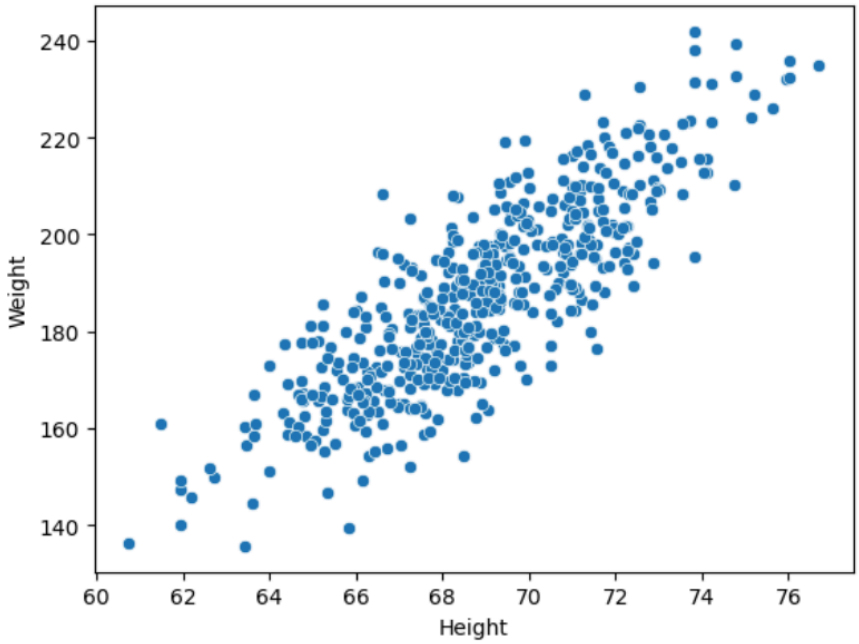
- Matplotlib

```
plt.scatter(data['Height'].iloc[:500], data['Weight'].iloc[:500], edgecolor = 'white')
plt.show()
```



- Seaborn

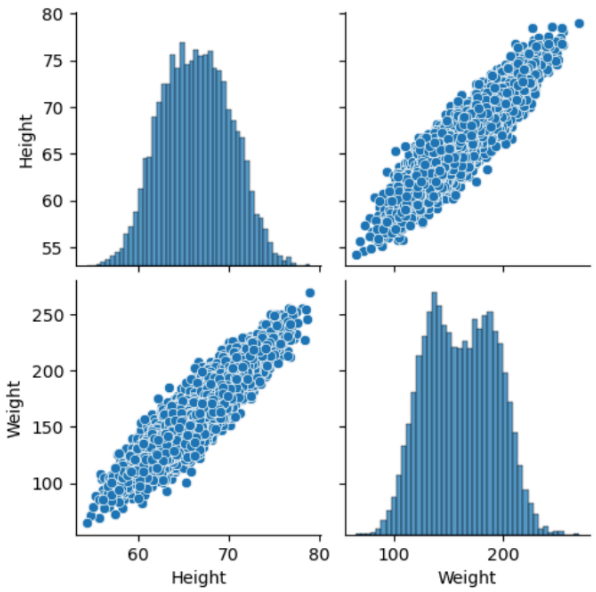
```
sns.scatterplot(data = data, x = data['Height'].iloc[:500], y = data['Weight'].iloc[:500])
plt.show()
```



Pair Plot

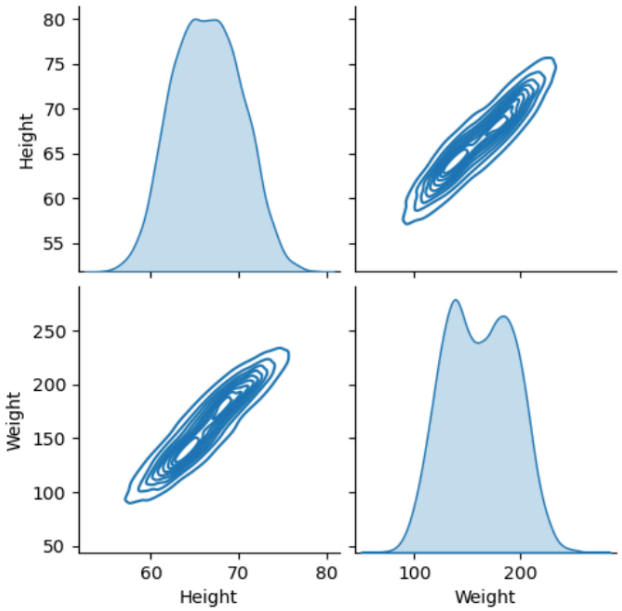
```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x241502add00>
```




```
sns.pairplot(data, kind = 'kde')
```

```
<seaborn.axisgrid.PairGrid at 0x241509098b0>
```



Word Embeddings

Word embeddings are a type of representation for text where words or phrases are mapped to vectors of real numbers. These embeddings capture the semantic meaning of words in a continuous vector space, making similar words closer together in the space. Word embeddings are a core concept in natural language processing (NLP) and are used to improve the performance of machine learning models on tasks such as text classification, machine translation, sentiment analysis, and more.

How Word Embeddings Work

- Vector Representation: Each word is represented as a dense vector (a list of numbers) in a high-dimensional space. These vectors are learned from large text corpora using specific algorithms, such as Word2Vec, GloVe, or FastText.
- Dimensionality: The vectors typically have 100 to 300 dimensions. Each dimension doesn't directly correspond to a specific feature, but the relationships between the dimensions encode semantic relationships between words.
- Semantic Similarity: Words with similar meanings are placed closer to each other in the vector space. For example, the words “king” and “queen” would be close to each other, while “king” and “car” would be farther apart.

Popular Algorithms for Word Embeddings

1. Word2Vec (Word to Vector)

Word2Vec is one of the most famous techniques for learning word embeddings. It uses two main architectures:

- Continuous Bag of Words (CBOW): Predicts the current word from a context of surrounding words.
- Skip-gram: Predicts surrounding words from the current word.

Word2Vec learns embeddings by training on a large corpus of text to minimize a loss function related to predicting words in context.

2. GloVe (Global Vectors for Word Representation)

GloVe is another popular algorithm. It constructs a word representation based on global word-word co-occurrence statistics from a corpus. It is trained using matrix factorization techniques and emphasizes the relationships between words in a corpus.

3. FastText

FastText, an extension of Word2Vec, improves the Word2Vec model by representing each word as a bag of character n-grams. This is especially useful for morphologically rich languages or when dealing with out-of-vocabulary words.

Simple Word2Vec using Gensim

```
sentences = [['human', 'interface', 'computer'],
             ['survey', 'user', 'computer', 'system', 'response', 'time'],
             ['eps', 'user', 'interface', 'system'],
             ['system', 'human', 'system', 'eps'],
             ['user', 'response', 'time'],
             ['trees'],
             ['graph', 'trees'],
             ['graph', 'minors', 'trees'],
             ['graph', 'minors', 'survey']]
```

```
model = Word2Vec(sentences = sentences, min_count = 1, vector_size = 8)
```

```
model.wv['trees']
```

```
array([ 0.03595725,  0.01239842, -0.10356519, -0.11811022,  0.09139708,
        0.06337827,  0.08447117,  0.00953582], dtype=float32)
```

```
similarity_score_hm_cp = model.wv.similarity('human', 'computer')
print('Similarity Score of Human and Computer :', similarity_score_hm_cp)
```

```
Similarity Score of Human and Computer : -0.44009817
```

```
similar_words_comp = model.wv.most_similar('computer', topn = 5)
print('Top 5 Similar Words of Computer :')
similar_words_comp
```

```
Top 5 Similar Words of Computer :
[('minors', 0.5697376132011414),
 ('response', 0.455085813999176),
 ('graph', 0.29385602474212646),
 ('user', -0.01504569686949253),
 ('interface', -0.07110846042633057)]
```

Reducing Dimensions using PCA

```
lst = []
for word in model.wv.index_to_key:
    lst.append(model.wv[word])
```

```
pca = PCA(n_components = 2)
```

```
reduced = pca.fit_transform(lst)
```

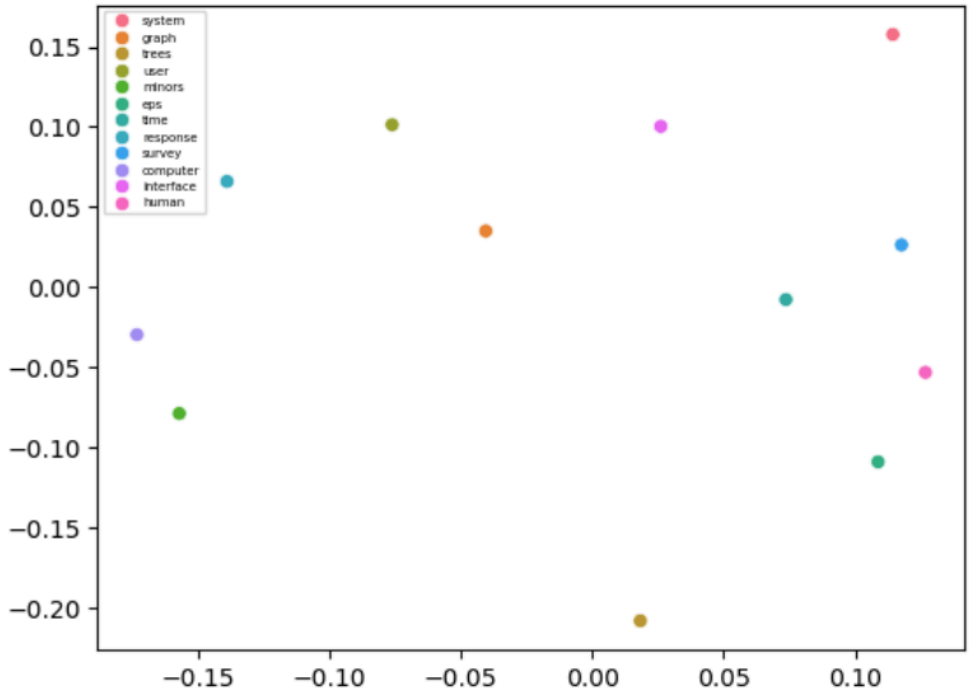
```
reduced
```

```
array([[ 0.11434661,  0.15765952],
       [-0.040465  ,  0.03502328],
       [ 0.01832792, -0.2081702 ],
       [-0.07608456,  0.10132395],
       [-0.15705209, -0.07884491],
       [ 0.10875142, -0.10900879],
       [ 0.0737009 , -0.00775762],
       [-0.13889178,  0.06594011],
       [ 0.11764772,  0.02636249],
       [-0.17316243, -0.02966088],
       [ 0.02611917,  0.10031539],
       [ 0.1267621 , -0.05318235]])
```

Visualizing Vectors on 2D plane

```
sns.scatterplot(x = reduced[:, 0], y = reduced[:, 1], hue = model.wv.index_to_key)
plt.legend(loc = 'upper left', fontsize = 5)
```

```
<matplotlib.legend.Legend at 0x2ec84c1fc20>
```



Word2Vec

Developed by researchers at Google, Word2Vec is an algorithm that transforms words into fixed-length vector representations. The main idea behind Word2Vec is encapsulated in the phrase, “You shall know a word by the company it keeps,” which emphasizes that the meaning of a word is derived from its context within a corpus.

Key Features of Word2Vec:

- Vector Representation: Each word is represented as a dense vector in a continuous vector space, where semantically similar words are located close to each other.
- Architectures: Word2Vec operates primarily through two architectures:
    - Continuous Bag of Words (CBOW): This model predicts a target word based on its surrounding context words. For instance, given the context words "the" and "cat," CBOW might predict the target word "sat."
    - Skip-Gram: In contrast, this model uses a target word to predict its surrounding context words. For example, given the target word "cat," it would predict context words like "the" and "sat."
- Training Process: The algorithm is trained on large text corpora, adjusting the vectors based on the co-occurrence of words within a specified window size. This allows it to capture semantic relationships effectively124.

GloVe (Global Vectors for Word Representation)

GloVe is another widely used algorithm for generating word embeddings, introduced by researchers at Stanford University. Unlike Word2Vec, which focuses on local context, GloVe leverages global statistical information across the entire corpus.

Key Features of GloVe:

- Co-occurrence Matrix: GloVe constructs a co-occurrence matrix that records how often words appear together in a given context. This matrix captures the frequency with which pairs of words co-occur in the corpus.
- Optimization Objective: The algorithm optimizes word vectors such that their dot product approximates the logarithm of the probability of co-occurrence between the words. This means that semantically similar words will have similar vector representations35.

- Dimensionality Options: GloVe provides pre-trained embeddings in various dimensions (e.g., 50d, 100d, 200d, 300d), allowing users to select vectors that best fit their computational resources and specific tasks36.

Applications of Word Embeddings

Word embeddings have transformed various NLP tasks by providing richer representations of text data:

- *Text Classification :* Used as features in models for tasks like sentiment analysis and spam detection.
- *Named Entity Recognition (NER) :* Enhances models' ability to identify entities by capturing semantic relationships.
- *Machine Translation :* Represents words across languages to improve translation quality.
- *Question Answering Systems :* Helps models understand context and relationships between words for accurate responses.