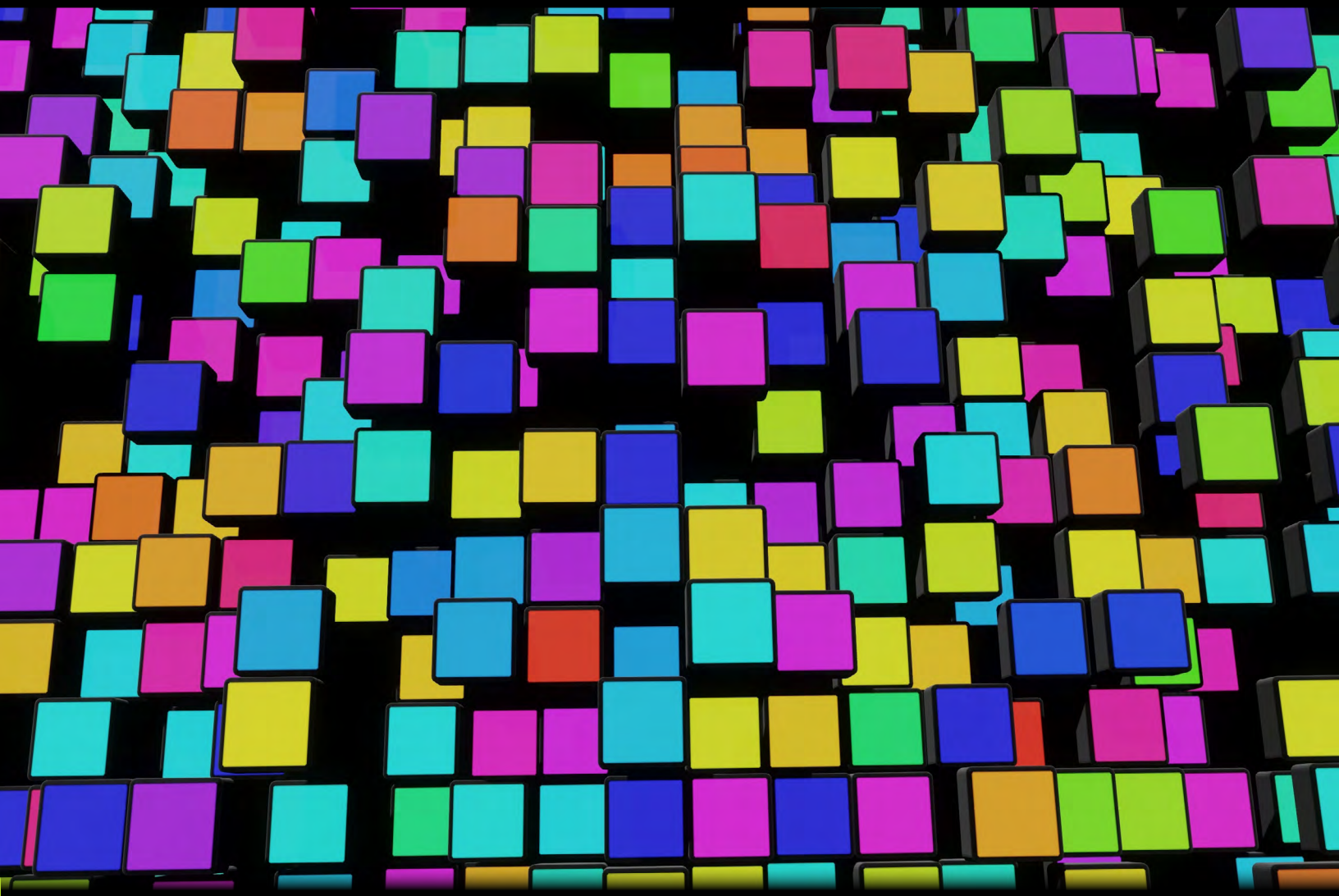
GlobalLogic®



Clustering Analysis for Customer Audio Conversations in InsurTech

by Dr. Param Jeet, Swarn Singh Bedi, and Sudhanshu Joshi

Contents

Introduction

In one way or another, we have all connected with customer care agents to inquire about a product or service.

Nowadays, whether it is a small organization confined to a specific location or a large corporation spread across multiple countries, almost every business has its own customer service team specially hired to address consumer grievances.

In some business domains (for example, banking or insurance), there are sales teams that help the company sell their products to the customer over a recorded phone line.

In this white paper, we'll focus on the insurance industry and analyze call recordings from customer sales agents. We will try to find patterns in the questions and answers that come up in conversations between customers and sales agents. This will help us discover common queries that arise when consumers purchase an insurance plan. It will also help determine customer preferences in areas related to an insurance policy.

Data Summary

Our primary source of data was the call recordings of conversations between sales agents and customers. We converted this to a transcript form using techniques that are explained in subsequent sections.

After converting the call recordings to textual form, the raw data we obtained was in the form of a python dictionary. A snapshot of the data is shown below for reference.


```
{'result_index': 0, 'results': [{'final': True, 'alternatives': [{'transcript': 'thanks for calling
Insurance Company, this is Kim ', 'confidence': 0.73, 'timestamps': [['thanks', 0.71, 1.0], ['for',
1.06, 1.23], ['calling', 1.23, 1.68], ['Pat', 1.68, 1.95], ['plan', 1.98, 2.35], ['%HESITATION',
2.35, 2.61], ['this', 2.61, 2.82], ['is', 2.82, 2.94], ['Kim', 2.94, 3.33]]}]}, {'final': True,
'alternatives': [{'transcript': 'yes Kim %HESITATION I have a new puppy eight eight weeks old ',
'confidence': 0.88, 'timestamps': [['yes', 5.42, 5.78], ['Kim', 5.78, 6.13], ['%HESITATION', 6.23,
6.66], ['I', 6.66, 6.8], ['have', 6.8, 6.99], ['a', 6.99, 7.05], ['new', 7.05, 7.21], ['puppy', 7.21,
7.84], ['eight', 8.31, 8.63], ['eight', 8.77, 8.98], ['weeks', 8.98, 9.27], ['old', 9.27, 9.63]]}]},
{'final': True, 'alternatives': [{'transcript': '%HESITATION it's %HESITATION Australian and
miniature Australian shepherd doodle necks ', 'confidence': 0.77, 'timestamps': [['%HESITATION',
10.62, 11.29], ['it's', 11.82, 12.04], ['%HESITATION', 12.04, 12.37], ['Australian', 12.45, 13.22],
['and', 13.25, 13.43], ['miniature', 13.43, 13.89], ['Australian', 13.89, 14.49], ['shepherd', 14.49,
15.13], ['doodle', 15.65, 16.01], ['necks', 16.01, 16.62]]}]}, {'final': True, 'alternatives':
[{'transcript': 'and I wanted to talk about the different plans I do have R. ', 'confidence': 0.98,
'timestamps': [['and', 17.43, 17.98], ['I', 17.98, 18.04], ['wanted', 18.04, 18.4], ['to', 18.4,
18.48], ['talk', 18.48, 18.77], ['about', 18.77, 19.02], ['the', 19.02, 19.1], ['different', 19.1,
19.45], ['plans', 19.45, 20.19], ['I', 20.69, 20.81], ['do', 20.81, 21.06], ['have', 21.06, 21.33],
['R.', 21.37, 21.73]]}]}, {'final': True, 'alternatives': [{'transcript': 'and I I see my sister got
```


As you can see from the snapshot, this data is highly disorganized and contains too much information to be useful to us. We applied certain techniques to clean this data and convert it into a more readable form. We converted the above python dictionary into a pandas dataframe, a snapshot of which is shown below.

key	speaker	transcript
1	0	thanks for calling the insurance company, this is Kim
2	1	yes Kim, I have a new puppy eight eight weeks old, it's an Australian shepherd doodle and I wanted to talk about the different plans
3	1	I see my sister got me insurance and they give a discount for her she said
4	0	sure let me help you we offer me that ten percent discount for our members
5	0	okay yeah and what we cover for your part would be any accidents feral the only thing we do not cover are pre existing conditions and preventative care
6	1	okay I thought there was a plans that cover like office visits and vaccines
7	0	well we cover the exam fee for when you're sick or injured so that's an office visit for an illness or an injury but we don't cover
8	0	just regular routine visit all right

The average duration of the call recordings was approximately 12 minutes. To convert the recordings into the raw python dictionary format, it took around 5-7 minutes per recording. The python dictionary received as an output is a nested dictionary, out of which we extracted the transcript. The pandas dataframe that we produced from this raw data has three columns: “key,” “speaker,” and “transcript.”

Approach

To analyze a conversation between a sales agent and a customer, the very first thing we did was listen to a few recordings in order to understand the context of the conversations. This helped us to understand the conversation in terms of the insurance industry domain and the customer base of this industry. This human analysis will help when making decisions or drawing conclusions from the results of the machine learning (ML) algorithms.

The second step was to convert the audio conversations to a transcript form. The transcript is in the form of speech and should identify the different speakers involved in the conversation. It could also contain some other information like the start and end time of a particular sentence by any of the speakers.

The third step was to clean and pre-process the transcript. A conversation between two individuals contains a lot of repeated words, para-phrases, sentences, etc. We do not want them to be counted every time during our analysis. Moreover, there will be a lot of useless words that will not add any meaning to a sentence (in terms of machines), and these should not be considered while we analyze a conversation using any of the ML techniques.

Then we needed to convert this cleaned data into a form that can be understood by ML models. An ideal form that can be supplied as an input to most ML models is the vectorized form. We will be talking about this in detail in subsequent sections.

The last step was to supply this vectorized form of our conversation to several ML models. There are a number of ML models that can be used for our analysis. We will apply various techniques and then choose the model that is best able to perform our desired operation. At the end, we will form a conclusion.

Tools and Technologies Used

Before we take a deep dive into the details of how the analysis was performed, allow us to give a quick introduction to the technologies and tools we used.

- **Jupyter Notebooks:** The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

- **IBM Watson™ Speech To Text:** The IBM Watson™ Speech to Text service allows APIs that use IBM's speech-recognition capabilities to produce transcripts of spoken audio. The service can transcribe speech from various languages and audio formats. In addition to basic transcription, the service can provide detailed information about many aspects of the audio.

- **Python Libraries:** We used several python libraries for different applications throughout the course of this analysis. Some of the most frequent libraries are as follows:
    - Pandas
    - Numpy
    - SkLearn
    - NLTK
    - Word2Vec

Methodologies

Sample Selection: We had a large number of recordings from different salespeople to perform our analysis. We randomly selected approximately five recordings from each salesperson. This was done so we can have a wide range of Q&As to diversify our dataset.

Speech Recognition: We used the IBM Watson™ Studio Speech Recognition API to convert the audio recordings into their individual transcripts.

Preparing our dataset: After we converted the call recordings to the python dictionary using the Watson™ Speech To Text API, we had to perform some data cleaning steps to prepare our data in a readable format so it could be consumed by different algorithms. After the data cleaning operations, we filtered out, separated, and combined the transcripts in a different dataframe and saved them in a CSV file.

```
with open("Rec1.txt", "r") as f:
    Sales1_rec1 = f.read()
    Sales1_rec1 = ast.literal_eval(Sales1_rec1)

    speakers_rec1 = pd.DataFrame(Sales1_rec1['speaker_labels']).loc[:,['from','to','speaker']]

    Conversation_rec1 = pd.DataFrame()

    for x in range(len(Sales1_rec1['results'])):
        Conversation_rec1 = Conversation_rec1.append(Sales1_rec1['results'][x]['alternatives'][0]['timestamps'])

    # Change the column names to something meaningful
    Conversation_rec1.columns =['word', 'start_time', 'end_time']

    result_rec1 = speakers_rec1.join(Conversation_rec1)

    # Sorting by the start_time of the call so we get the conversation as it proceeds
    result_rec1 = result_rec1.sort_values('start_time')
    result_rec1 = result_rec1.dropna()
    result_rec1.to_csv('result_sales1_rec1.csv')
```

Data Cleaning: After preparing our raw corpus, we applied various data cleaning techniques like tokenization, removing stop words, etc., to prepare a clean data set. These techniques are widely used when dealing with ML algorithms to analyze textual data.

- **Removing words other than stop words:** Stop words are connecting words that don't add meaning to a sentence. We used the WordCloud library to get a summary of the most common words in the corpus. These words reduce the visibility of important words.

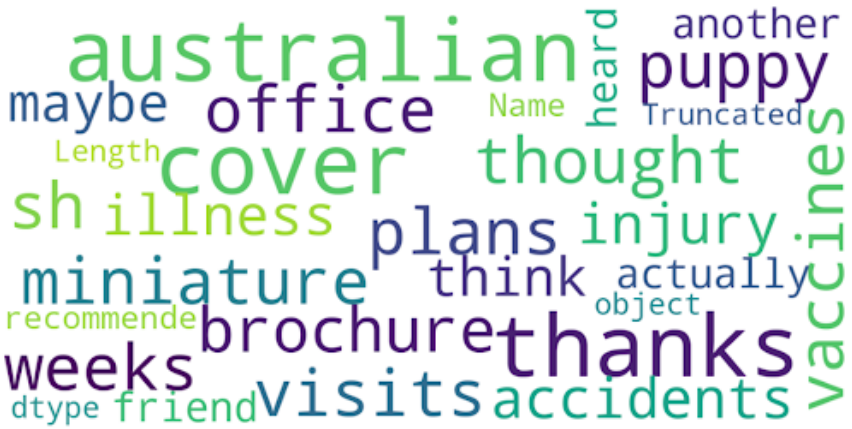
- **Tokenization:** This is a technique that's used to split text into smaller elements. A tokenizer can be a self-written method, or we can also use tokenizers that come with the NLTK library. In our case, we used both according to our use case

```
### Corpus tokenization
sentences = []
words = []
for word in customer_data['word']:
  sentences.append(word)
print(sentences)
words = np.unique(list(flatten(sentences)))
```

- **Stop Words:** This is removing words other than stop words, cleaning the data, performing TF-IDF, and cleaning the dataset for better results.

```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

# add stemming and lemmatisation in the preprocess function
def preprocess(document):
    'changes document to lower case and removes stopwords'
    #print((document))

    # remove special characters
    document = document.replace("[.,#,%,',:]", "")

    # change sentence to lower case
    document = document.lower()

    # tokenize into words
    words = word_tokenize(document)

    # remove stop words
    words = [word for word in words if word not in stopwords.words("english")]

    # removing some more words
    words = [word for word in words if word not in removeWords]

    # remove words less than 4 words
    words = [word for word in words if len(word)>4]

    # stem
    #words = [stemmer.stem(word) for word in words]

    # join words to make sentence
    document = " ".join(words)

    return document
```

- **WordCloud:** We used a WordCloud to look at the words that are most common in our corpus. WordCloud is employed to generate a visual picture of the corpus you are working on, which helps you find the main topic of a document. It also helps to see if you have a lot of words in your corpus that mean nothing for your problem statement. Using this, we found that we had certain meaningless words in our corpus, even after removing the stop words. Some examples of these words are: okay, hi, hello, thank you, one, two, three.



- **TF-IDF:** As we know, ML models work on numerical data and not actual words. To account for this, we applied the Term frequency–Inverse document frequency (TF-IDF) to our corpus to get the dense matrix and feature names from our corpus.

  TF-IDF is a smarter way than Bag of Words (BOW) to represent the importance of each word in a document. The TF-IDF score will be higher for words that are rare among multiple documents but are frequently present in one document. The TF-IDF score will be low for terms that are common among all the documents.

  We used sklearn '*TfidfVectorizer*' class to do the same thing.

```
1 vectorizer = TfidfVectorizer()
2 tfidf_model = vectorizer.fit_transform(combinedDF['Truncated'])
3 #print(tfidf_model)
```

- **Training the dataset:** We trained different traditional unsupervised learning models like KMeans, Hierarchical Clustering, and DBSCAN.

Modeling

KMeans is an unsupervised learning model that works in cases where we don't have predefined labels in our dataset. This model is used for classification purposes with unlabeled data. It cannot be used for prediction or forecasting purposes.

KMeans is also used for clustering. Clustering is an unsupervised machine learning technique that is used to place data elements into a related group without prior knowledge of the group definitions. It can be used to identify hidden patterns and structures in data.

When training using KMeans, we don't split our data into test and train datasets. The internal working of KMeans is very deep and cannot be explained here. However, there are different ways by which we can check whether our data will give us meaningful results or not if we use KMeans.

- Using Hopkins's method to check the clustering tendency of the dataset. We can use this method because a random dataset will not give good results. If the score is close to 1, then the dataset can be clustered efficiently. If the score is 0.5, it means that the dataset is randomly placed and will not give fruitful results. **We got a score of 0.94 for our dataset.**

- We can decide on the value of 'K,' i.e., the number of clusters required for best results. This can be chosen by using the 'silhouette metrics.' A score closer to 1 is considered a better score. The clusters chosen must have high cohesion, i.e., the data points inside a cluster must be close to each other, and the two different clusters must be well separated.

```
from sklearn import metrics
labels = clf.labels_
centroids = clf.cluster_centers_

silhouette_score = metrics.silhouette_score(tfidf_model.toarray(), labels, metric='euclidean')

print ("Silhouette_score: ")
print (silhouette_score)
```

- Using the elbow curve, we can decide the value of K for our model.

```
## elbow curve
ssd = []
for num_clusters in list(range(20,200)):
    model_clus = KMeans(n_clusters = num_clusters, max_iter=50)
    model_clus.fit(names)
    ssd.append(model_clus.inertia_)

plt.plot(ssd)
```

After we applied KMeans on our dataset over the matrix that we got using the TF-IDF vectorizer, we discovered many beneficial insights with this model.

```
# Kmeans with tf-idf

clf = cluster.KMeans(n_clusters=50, init='random',
              max_iter = 100, n_init=10, tol=0.001)

cluster_KMeans = pd.DataFrame(0, index=range(len(combinedDF['Truncated'])),
    columns=['Transcript', 'Cluster', 'Cluster_Frequency', 'Cluster Rank'])

cluster_KMeans.iloc[:,0] = combinedDF['Truncated']
c1 = np.array([l for l in combinedDF['Truncated']])
cluster_KMeans.iloc[:,0]  = c1
cluster_KMeans.iloc[:,1] = clf.fit_predict(tfidf_model.toarray())

cluster_KMeans_frequency = cluster_KMeans.iloc[:,1].value_counts()
cluster_KMeans_frequency.to_dict()
#cluster_KMeans_frequency

cluster_KMeans.iloc[:,2] = cluster_KMeans['Cluster']
                           .map(cluster_KMeans_frequency)
cluster_KMeans.iloc[:,3] = cluster_KMeans['Cluster_Frequency'].rank()

cluster_KMeans.to_csv("/content/Data/cluster_KMeans.csv")
```

Using a Neural Networks Approach

We also tried to improve our results using the latest neural networks.

- **Word2Vec:** This is used to represent words as a vector. It is based on the idea that if we represent words as vectors, words that are similar or are used in the same context will have vectors close to each other. The cosine angle between them will be negligible. A word is converted to its vector form using this library, and then similarities can be discovered based on the results. Here's an example using the word "glove" in **word2vec in code**.

```
## Using glove in word2vec
from gensim.scripts.glove2word2vec import glove2word2vec
glove_input_file = '/content/Data/glove.6B.100d.txt'
word2vec_output_file = '/content/Data/glove.6B.100d.w2vformat.txt'
glove2word2vec(glove_input_file, word2vec_output_file)
```

```
(400000, 100)
```

```
#model = Word2Vec(text_tokens,min_count=2,window=5,sg=1
from gensim.models.keyedvectors import KeyedVectors
model = KeyedVectors.load_word2vec_format("/content/Data/glove.6B.100d.w2vformat.txt", binary=False)
```

There are basically two types of this model: skip-gram and CBOW. We used skip-gram in our model. In simple terms, skip-gram uses the current word to predict the words in the neighborhood, while CBOW uses neighboring words to predict the current word,

Results and Conclusions

We created a dataframe with the following structure:

	Original	Truncated	cluster
0	yes Kim I have a new puppy eight eight weeks ...	puppy weeks australian miniature australian sh...	31

We used this dataframe to analyze the results. The 'Original' column shows us the original sentence, and the 'Truncated' column shows us the sentence after data cleaning. 'Cluster' tells us which cluster has been assigned to that sentence by Kmeans using the word embedding (from word2Vec) as input.

S.No	KMeans	KMeans with Word2Vec
1.	Word context was not taken into consideration, different context sentences were tagged in the same cluster.	Word context was taken into consideration.
2.	Less meaningful clusters were formed.	Clusters had more meaning to them.
3.	Fewer computations are required.	More computation power might be required.

Examples of the clusters formed:

okay so the first six months NY and have did you say	months	5
four months	months	5
the past six months let	months	5
but it will cover from this state like step from this stay six months off from when he was born six months	cover state months months	5
he's five months	months	5
is that he's got nine months	months	5
all right and so she's nine months	months	5
that's for six months okay	months	5

In the cluster above, we can see which sentences have been clustered together when the age in months is being discussed.

can so what's the deductible on that	deductible	24
to this one out three hundred so can for it is that much difference instead of a three hundred dollar deductible for two fifty deductible	difference instead dollar deductible fifty deductible	24
all right and the deductible is three hundred	deductible	24
right okay anyway with that won't be happening for a while because even if we have a charge it would we have the deductible to pay so	anyway happening charge deductible	24

The cluster shown here consists of sentences that refer to a deductible.

Using the word embeddings, we achieved similar results to what we got when using only KMeans. However, we found that using Word2Vec was more beneficial because it is the best model.

This model can be employed in the following use cases:

- **FAQ Section:** An insurance company can update the FAQ section on their website with the common questions we found in our analysis.
- **Building a Chatbot:** An insurance company can build an AI chatbot and incorporate it on its website. Customers can interact with the chatbot and get answers to general questions when buying an insurance policy.
- **Designing UI:** We can create some input parameters for FAQ about the insurer.

Some results we found to be useful:

1. Customers mainly prefer monthly or quarterly payments.
2. Inputs such as age in months are mostly asked by the customer care representative. If we intend to automate this process, we can provide a user input for the customer to enter their age.
3. Most conversations are related to deductibles and insurance details, etc.

References

1. Global Call Centers Industry
2. Call center industry: market size by region 2017
3. Speech to Text
4. Data Mining Map
5. Word2Vec
6. GloVe: Global Vectors for Word Representation

About the Authors

Dr. Param Jeet has a Ph.D. in mathematics and more than ten years of hands-on industry experience. He has published many research papers in international journals and authored a book. Currently, he is leading AI/ML practice at GlobalLogic, India, and collaborating with universities to enhance the Industry-Academic Research Collaboration program.

Swarn Singh Bedi is an experienced senior Java developer with six years of experience and a post-graduate degree in ML/AI. Software development and ML/AI are his main interests, and he endeavors to improve his skills while working with clients to provide efficient solutions.

Sudhanshu Joshi has been working as a Senior Software Engineer for the last 2.5 years with GlobalLogic. He has six years of experience in the IT industry and has worked on projects in web development using Python, Django Framework, and Cloud Technologies. He is a data science enthusiast and has been constantly developing his skills in this field.

GlobalLogic®

GlobalLogic, a Hitachi Group Company, is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating our strategic design, complex engineering, and vertical industry expertise with Hitachi's Operating Technology and Information Technology capabilities, we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses. Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the automotive, communications, financial services, healthcare & life sciences, media and entertainment, manufacturing, semiconductor, and technology industries.



www.globallogic.com