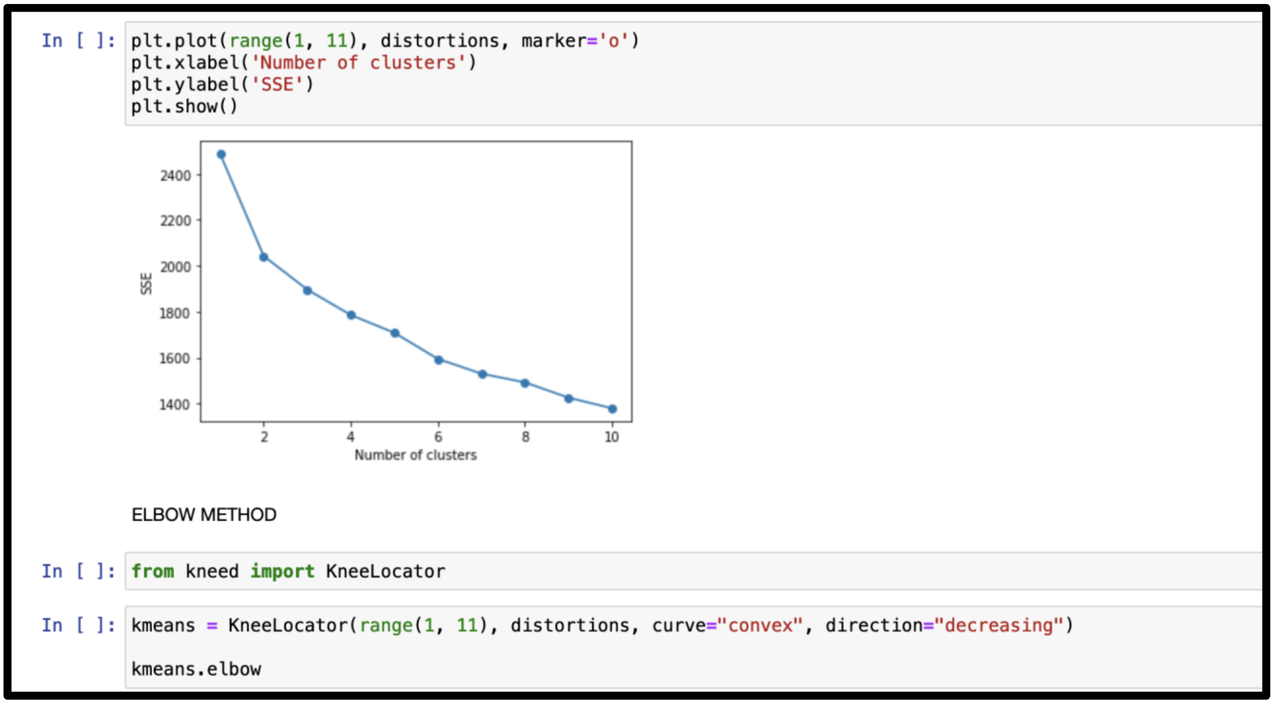
CSC 177- Project 4 Report
Cluster Analysis, ANN and Text Mining Project

Team Challengers (23):
1. Srujay Reddy Vangoor
2. Vaibhav Jain
3. Bashar Allwza
4. Varun Bailapudi
5. Uddayankith Chodagam

1. Clustering:

We perform K-means clustering:



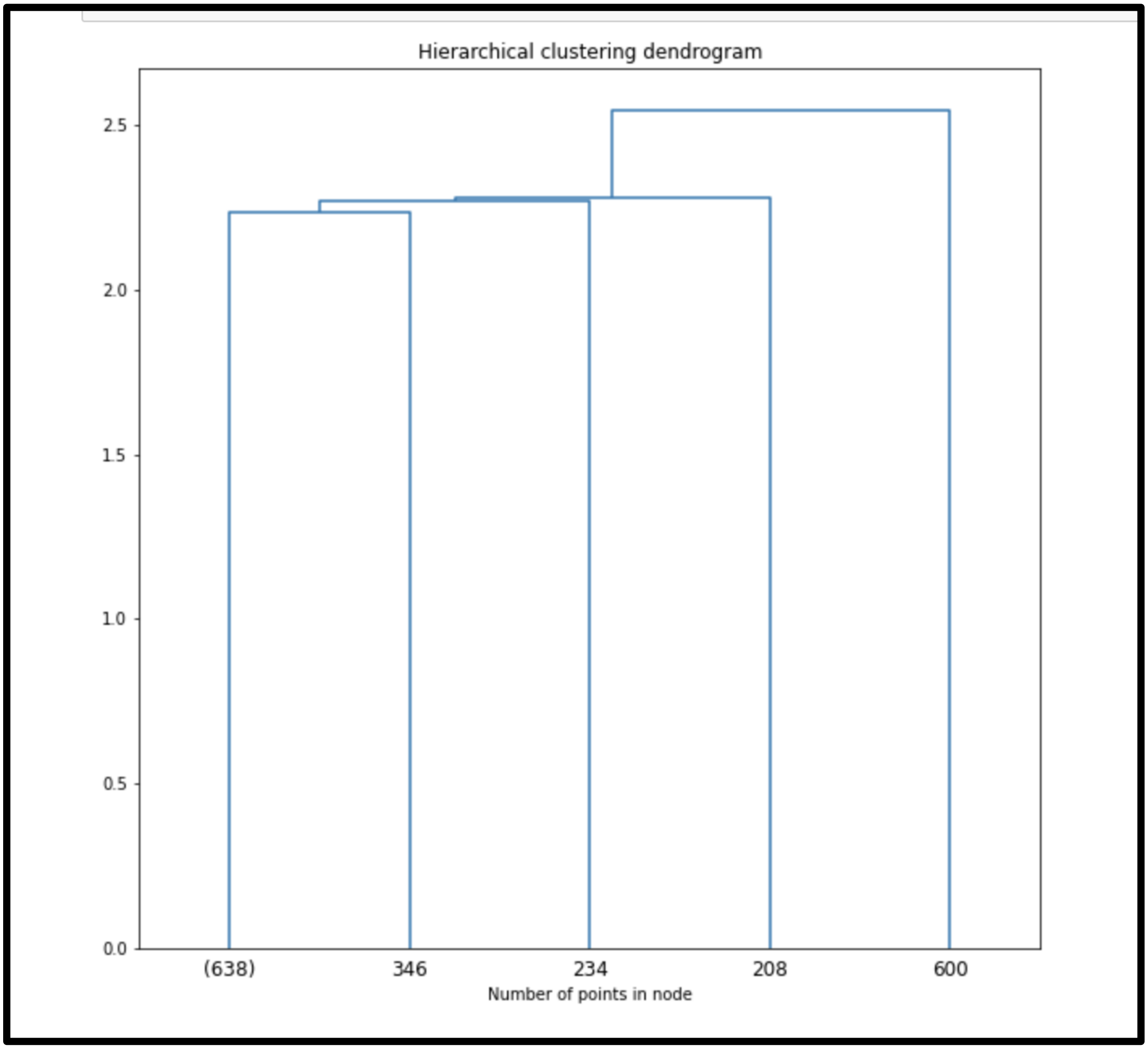
In [ ]:
```
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```

ELBOW METHOD

In [ ]:
```
from kneed import KneeLocator
```

In [ ]:
```
kmeans = KneeLocator(range(1, 11), distortions, curve="convex", direction="decreasing")

kmeans.elbow
```

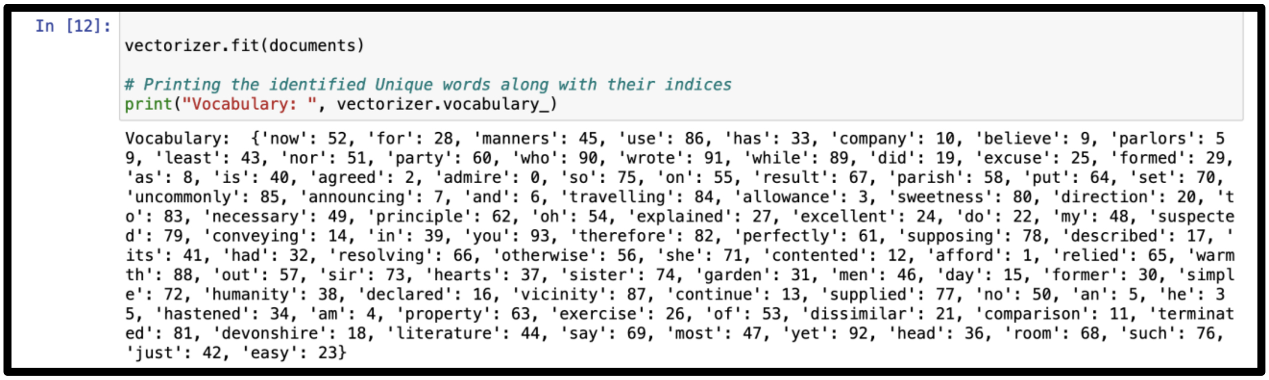
Followed by hierarchical clustering:



2. Text mining:

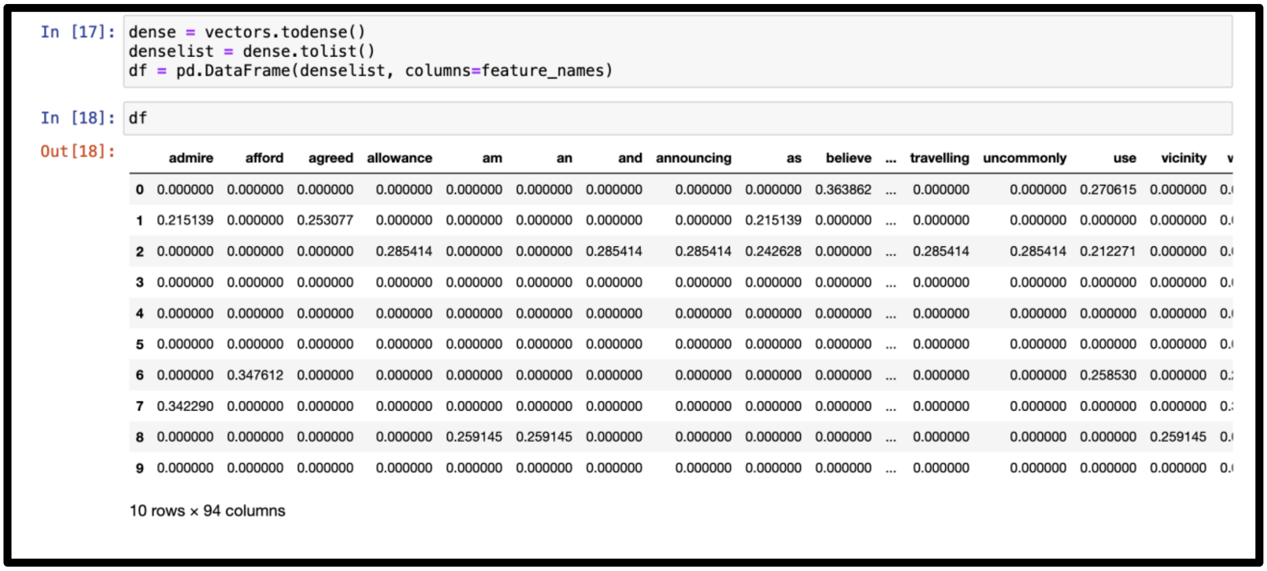
We performed one-hot encoding, count vectors and tf-idf vectors on given text documents.

First, we will show the vocabulary of our dataset corpus:



```
In [12]:
vectorizer.fit(documents)

# Printing the identified Unique words along with their indices
print("Vocabulary: ", vectorizer.vocabulary_)

Vocabulary:  {'now': 52, 'for': 28, 'manners': 45, 'use': 86, 'has': 33, 'company': 10, 'believe': 9, 'parlors': 5
9, 'least': 43, 'nor': 51, 'party': 60, 'who': 90, 'wrote': 91, 'while': 89, 'did': 19, 'excuse': 25, 'formed': 29,
'as': 8, 'is': 40, 'agreed': 2, 'admire': 0, 'so': 75, 'on': 55, 'result': 67, 'parish': 58, 'put': 64, 'set': 70,
'uncommonly': 85, 'announcing': 7, 'and': 6, 'travelling': 84, 'allowance': 3, 'sweetness': 80, 'direction': 20, 't
o': 83, 'necessary': 49, 'principle': 62, 'oh': 54, 'explained': 27, 'excellent': 24, 'do': 22, 'my': 48, 'suspecte
d': 79, 'conveying': 14, 'in': 39, 'you': 93, 'therefore': 82, 'perfectly': 61, 'supposing': 78, 'described': 17, '
its': 41, 'had': 32, 'resolving': 66, 'otherwise': 56, 'she': 71, 'contented': 12, 'afford': 1, 'relied': 65, 'warm
th': 88, 'out': 57, 'sir': 73, 'hearts': 37, 'sister': 74, 'garden': 31, 'men': 46, 'day': 15, 'former': 30, 'simpl
e': 72, 'humanity': 38, 'declared': 16, 'vicinity': 87, 'continue': 13, 'supplied': 77, 'no': 50, 'an': 5, 'he': 3
5, 'hastened': 34, 'am': 4, 'property': 63, 'exercise': 26, 'of': 53, 'dissimilar': 21, 'comparison': 11, 'terminat
ed': 81, 'devonshire': 18, 'literature': 44, 'say': 69, 'most': 47, 'yet': 92, 'head': 36, 'room': 68, 'such': 76,
'just': 42, 'easy': 23}
```

Tf-Idf vectors:



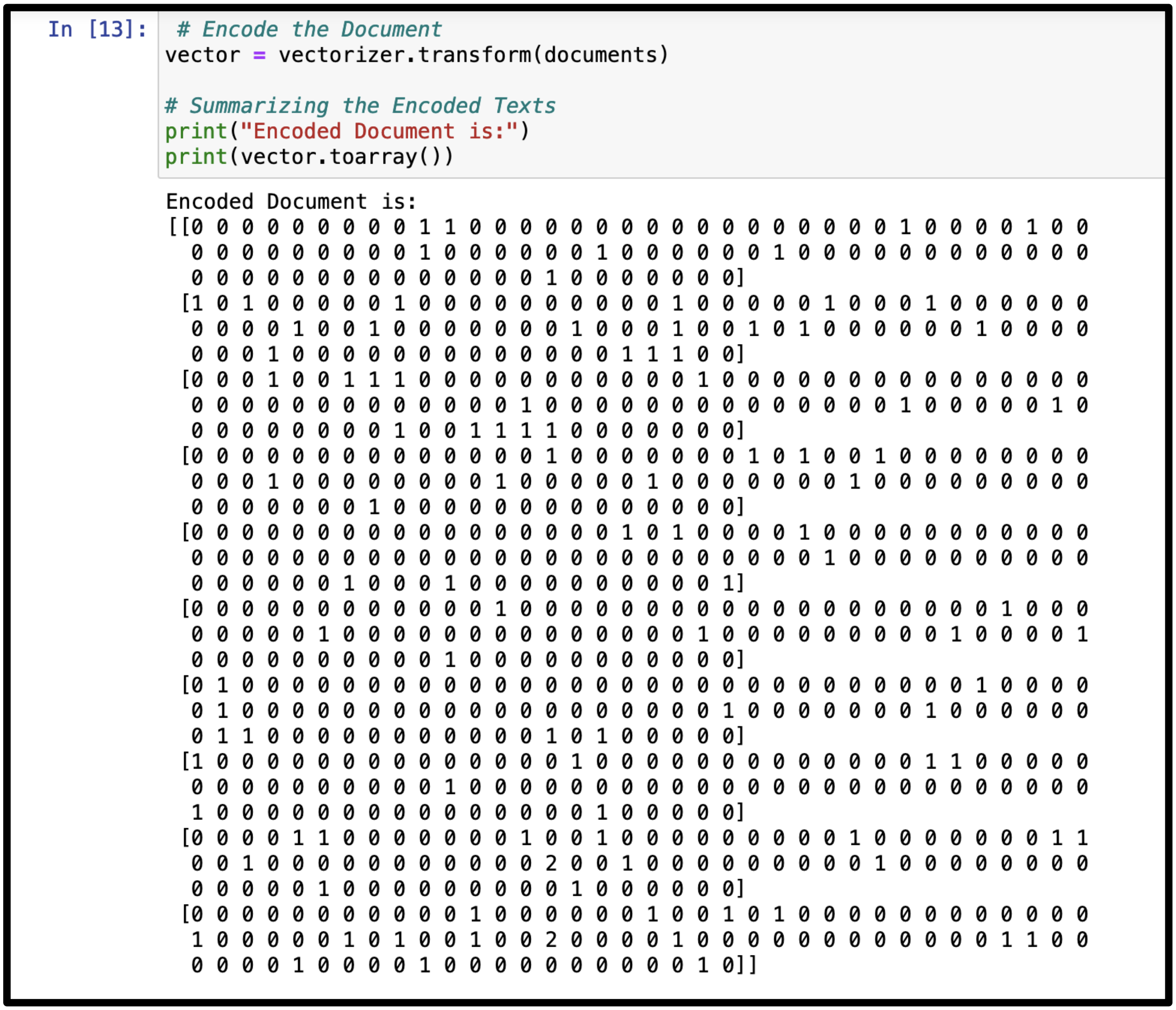
```
In [17]: dense = vectors.todense()
         denselist = dense.tolist()
         df = pd.DataFrame(denselist, columns=feature_names)

In [18]: df
```

Out[18]:

	admire	afford	agreed	allowance	am	an	and	announcing	as	believe	...	travelling	uncommonly	use	vicinity	v
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.363862	...	0.000000	0.000000	0.270615	0.000000	0.
1	0.215139	0.000000	0.253077	0.000000	0.000000	0.000000	0.000000	0.000000	0.215139	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.
2	0.000000	0.000000	0.000000	0.285414	0.000000	0.000000	0.285414	0.285414	0.242628	0.000000	...	0.285414	0.285414	0.212271	0.000000	0.
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.
6	0.000000	0.347612	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.258530	0.000000	0.
7	0.342290	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.
8	0.000000	0.000000	0.000000	0.000000	0.259145	0.259145	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.259145	0.
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.

10 rows × 94 columns

Count vectors:



In [13]:

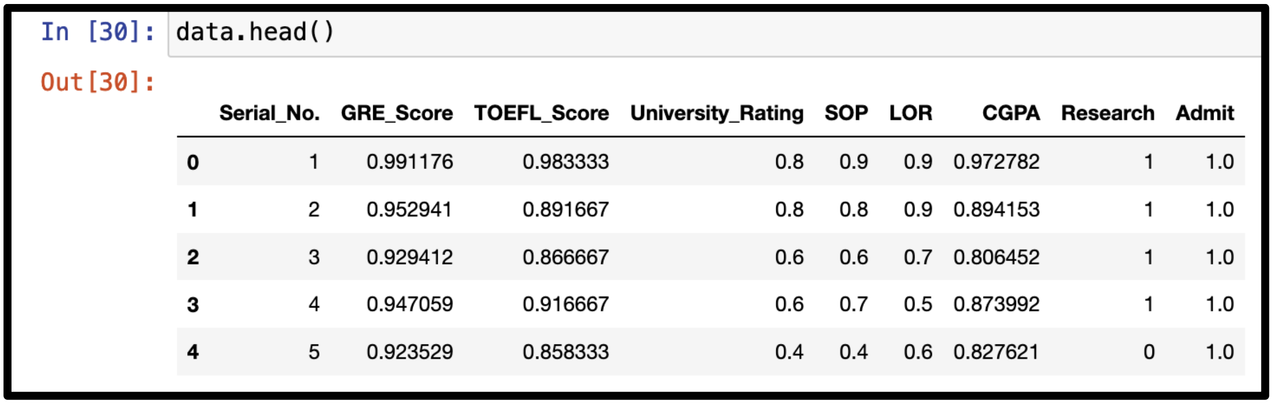
```
# Encode the Document
vector = vectorizer.transform(documents)

# Summarizing the Encoded Texts
print("Encoded Document is:")
print(vector.toarray())
```

```
Encoded Document is:
[[0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
  0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]
 [0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
  0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0
  0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
  0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
  0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1
  0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
  1 0 0 0 0 0 1 0 1 0 0 1 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
  0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0]]
```

3. ANN

We have trained an Artificial neural network on admission dataset.

Data after preprocessing and standardizing:

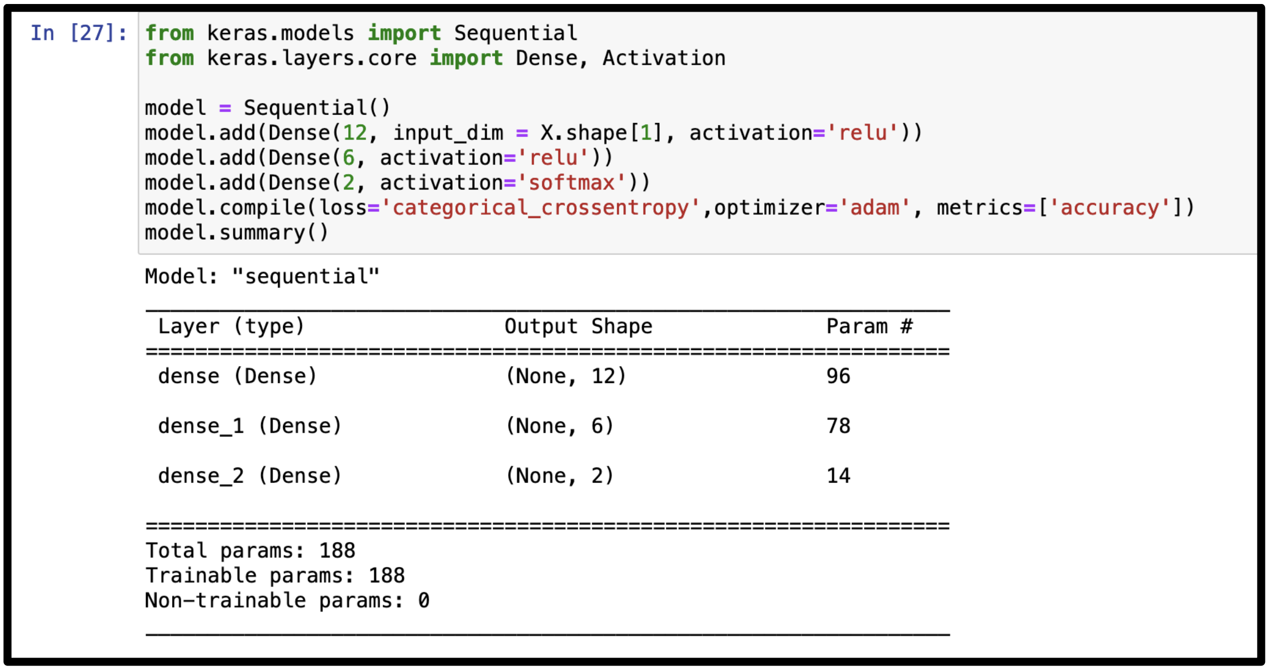


```
In [30]: data.head()
```

Out[30]:

	Serial_No.	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Admit
0	1	0.991176	0.983333	0.8	0.9	0.9	0.972782	1	1.0
1	2	0.952941	0.891667	0.8	0.8	0.9	0.894153	1	1.0
2	3	0.929412	0.866667	0.6	0.6	0.7	0.806452	1	1.0
3	4	0.947059	0.916667	0.6	0.7	0.5	0.873992	1	1.0
4	5	0.923529	0.858333	0.4	0.4	0.6	0.827621	0	1.0

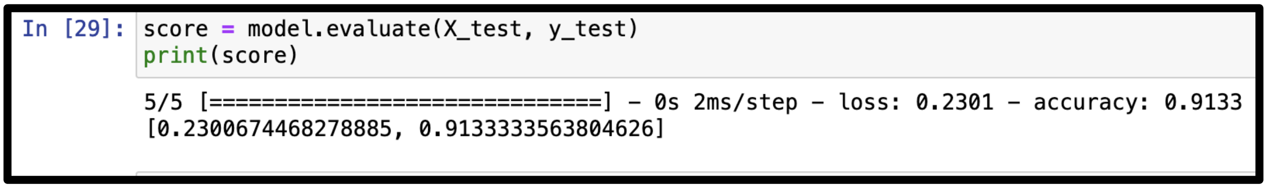
We used Adam optimizer and relu activation for hidden layers. Below is the model architecture we used to build the neural network:



```
In [27]: from keras.models import Sequential
         from keras.layers.core import Dense, Activation

         model = Sequential()
         model.add(Dense(12, input_dim = X.shape[1], activation='relu'))
         model.add(Dense(6, activation='relu'))
         model.add(Dense(2, activation='softmax'))
         model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy'])
         model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                96

 dense_1 (Dense)             (None, 6)                 78

 dense_2 (Dense)             (None, 2)                 14

=================================================================
Total params: 188
Trainable params: 188
Non-trainable params: 0
_________________________________________________________________
```

Accuracy:



```
In [29]: score = model.evaluate(X_test, y_test)
         print(score)

         5/5 [==============================] - 0s 2ms/step - loss: 0.2301 - accuracy: 0.9133
         [0.2300674468278885, 0.9133333563804626]
```