```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
```

```
!pip install transformers
```

```
Collecting transformers
  Downloading https://files.pythonhosted.org/packages/81/89/f07e7a884072ad37b1b6b1578637ab36152e0251d74a
     |████████████████████████████████| 1.8MB 8.2MB/s
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/di
Requirement already satisfied: dataclasses; python_version < "3.7" in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: filelock in /usr/local/lib/python3.6/dist-packages (from transformers) (3
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from transformers) (2
Collecting tokenizers<0.11,>=0.10.1
  Downloading https://files.pythonhosted.org/packages/fd/5b/44baae602e0a30bcc53fbdbc60bd940c15e143d252d6
     |████████████████████████████████| 3.2MB 33.7MB/s
Collecting sacremoses
  Downloading https://files.pythonhosted.org/packages/7d/34/09d19aff26edcc8eb2a01bed8e98f13a1537005d31e9
     |████████████████████████████████| 890kB 53.3MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.6/dist-packages (from transfo
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-packages (from transformers) (
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in /usr/local/lib/pythor
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from importlib-metac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from reques
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->tr
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from request
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from sacremoses->transform
Requirement already satisfied: click in /usr/local/lib/python3.6/dist-packages (from sacremoses->transfo
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from sacremoses->transf
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.6/dist-packages (from packagir
Building wheels for collected packages: sacremoses
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.43-cp36-none-any.whl size=893261 sha256=e29b7e5a
  Stored in directory: /root/.cache/pip/wheels/29/3c/fd/7ce5c3f0666dab31a50123635e6fb5e19ceb42ce38d4e58f
Successfully built sacremoses
Installing collected packages: tokenizers, sacremoses, transformers
Successfully installed sacremoses-0.0.43 tokenizers-0.10.1 transformers-4.3.1
```

```
from transformers import pipeline
```

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

Reading the dataset which is in JSON format and converting it to a dataframe

```
import json

f_train = open("/content/drive/MyDrive/Python programs/COVID-QA.json")
train_data = json.load(f_train)
train = pd.DataFrame.from_dict(train_data, orient='index')
train.reset_index(level=0, inplace=True)
f_train.close()
print(len(train_data))

train.head()
```

```
1
```

index	0	1	2	3	4	5	6	
0	data	{'paragraphs': [{'qas': [{'question':	{'paragraphs': [{'qas': [{'question':	{'paragraphs': [{'qas': [{'question':	{'paragraphs': [{'qas': [{'question':	{'paragraphs': [{'qas': [{'question':	{'paragraphs': [{'qas': [{'question':	{'paragraphs': [{'qas': [{'question':	{'para [{'c

Converting the data in question - answer format

```
que = []
ans = []
context=[]
answer_start=[]
for i in range(146):
  temp = train[[i]]
  tt = ((temp[i]))
  aa= (tt[0])
  xx = aa['paragraphs']
  question=[]
  answer=[]
  xx=xx[0]
  ll = xx['qas']
  for y in range(len(ll)):
    context.append(xx['context'])
    pp = (ll[y])
    # print(pp)
    if 'question' in pp.keys():
      que.append(pp['question'])
      anslist = pp['answers']
      anslist = anslist[0]
      ans.append(anslist['text'])
      answer_start.append(anslist['answer_start'])
```

```
print(que)
print(ans)
```

```
['What is the main cause of HIV-1 infection in children?', 'What plays the crucial role in the Mother to
['Mother-to-child transmission (MTCT) is the main cause of HIV-1 infection in children worldwide. ', 'DC
```

```
print((len(context[0])))
```

```
31035
```

```
data = pd.DataFrame(data=list(zip(que, ans,context,answer_start)), columns=['q
print(data.shape)
data.head()
```

```
(2014, 4)
```

	questions	answers	context	starting_index
0	What is the main cause of HIV-1 infection in c...	Mother-to-child transmission (MTCT) is the mai...	Functional Genetic Variants in DC-SIGNR Are As...	370
1	What plays the crucial role in the Mother to C...	DC-SIGNR plays a crucial role in MTCT of HIV-1...	Functional Genetic Variants in DC-SIGNR Are As...	2003
2	How many children were infected by HIV-1 in 20...	more than 400,000 children were infected world...	Functional Genetic Variants in DC-SIGNR Are As...	2291
3	What is the role of C-C Motif Chemokine Ligand...	High copy numbers of CCL3L1, a potent HIV-1 su...	Functional Genetic Variants in DC-SIGNR Are As...	28143
4	What is DC-GENR and where is it expressed?	Dendritic cell-specific ICAM-grabbing non-inte...	Functional Genetic Variants in DC-SIGNR Are As...	3207

```
que_lengths = data['questions'].str.split().apply(len)
ans_lengths = data['answers'].str.split().apply(len)


print(np.percentile(que_lengths, 99))
print(np.percentile(ans_lengths, 99))
```

```
20.0
68.60999999999967
```

```
con_lengths = data['context'].str.split().apply(len)
print(np.percentile(con_lengths, 95))
```

```
11368.0
```

Here we can see the 99 percent of the questions have lenth of 20 and 99 percent of answers have length of 69.

Creating Train,test and validation set to check the vocabulary

```
msk = np.random.rand(len(data)) < 0.8
model = data[msk]
test = data[~msk]


test.shape
```

```
(423, 4)
```

```
model.shape
```

```
(1591, 4)
```

```
msk = np.random.rand(len(model)) < 0.8
train = model[msk]
validation = model[~msk]


train_vocab = dict(train.questions.str.split(expand=True).stack().value_counts


validation_vocab = dict(validation.questions.str.split(expand=True).stack().va


test_vocab = dict(test.questions.str.split(expand=True).stack().value_counts()
```

Total number of unique words present in train, validation and test set are -

```
print(len(train_vocab))
print(len(validation_vocab))
print(len(test_vocab))
```

```
2852
1132
1436
```

```
train_vocab_words = set(train_vocab.keys())
validation_vocab_words = set(validation_vocab.keys())
test_vocab_words = set(test_vocab.keys())


total_in_train_validation = train_vocab_words.union(validation_vocab_words)
print("total unique words present in train and validation set are = ",len(tota
common_train_valid_words = train_vocab_words.intersection(validation_vocab_wor
print("Common words which are present in both train and validation set = ",len
```

```
    total unique words present in train and validation set are =  3248
    Common words which are present in both train and validation set =  736


total_in_train_test = train_vocab_words.union(test_vocab_words)
print("total unique words present in train and test set are = ",len(total_in_t
common_train_test_words = train_vocab_words.intersection(test_vocab_words)
print("Common words which are present in both train and test set = ",len(commc

    total unique words present in train and test set are =  3436
    Common words which are present in both train and test set =  852
```

Calculating Cosine similarity between 2 questions

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# X = input("Enter first string: ").lower()
# Y = input("Enter second string: ").lower()
def cosine_sim(X,Y):
  # X ="I love horror movies"
  # Y ="Lights out is a horror movie"

  # tokenization
  X_list = word_tokenize(X)
  Y_list = word_tokenize(Y)
  # print("here")
  # sw contains the list of stopwords
  sw = stopwords.words('english')
  l1 =[];l2 =[]

  # remove stop words from the string
  X_set = {w for w in X_list if not w in sw}
  Y_set = {w for w in Y_list if not w in sw}

  # form a set containing keywords of both strings
  rvector = X_set.union(Y_set)
  for w in rvector:
      if w in X_set: l1.append(1) # create a vector
      else: l1.append(0)
      if w in Y_set: l2.append(1)
      else: l2.append(0)
  c = 0

  # cosine formula
  for i in range(len(rvector)):
          c+= l1[i]*l2[i]
  cosine = c / float((sum(l1)*sum(l2))**0.5)
  # print("similarity: ", cosine)
  return cosine


cosine_score=[]
temp_te_q=[]
temp_tr_q=[]
temp_sc=[]
train_ques = train['questions'].values
test_ques = test['questions'].values
for q in train_ques:
  for qq in test_ques:
    x = cosine_sim(q,qq)
    cosine_score.append(x)
    if x > 0.5:
      temp_te_q.append(qq)
      temp_tr_q.append(q)
      temp_sc.append(x)
```

```
sim_ques = pd.DataFrame(data=list(zip(temp_tr_q, temp_te_q,temp_sc)), columns=
print(sim_ques.shape)
sim_ques.head()
```

```
(9037, 3)
```

	train_ques	test_ques	sim_score
0	What is the main cause of HIV-1 infection in c...	What is the main cause of death in the neonata...	0.534522
1	What is the main cause of HIV-1 infection in c...	What are the symptoms of HBoV1 infection?	0.507093
2	What is the main cause of HIV-1 infection in c...	What is the most common infection in childhood?	0.507093
3	What is the main cause of HIV-1 infection in c...	What is the cause of Lassa fever?	0.507093
4	What is DC-GENR and where is it expressed?	What is hepcidin?	0.577350

```
cosine_score=[]
temp_val_q=[]
temp_tr_q=[]
temp_sc=[]
train_ques = train['questions'].values
val_ques = validation['questions'].values
for q in train_ques:
  for qq in val_ques:
    x = cosine_sim(q,qq)
    cosine_score.append(x)
    if x > 0.5:
      temp_val_q.append(qq)
      temp_tr_q.append(q)
      temp_sc.append(x)
```

```
sim_ques_tr_val = pd.DataFrame(data=list(zip(temp_tr_q, temp_val_q,temp_sc)),
print(sim_ques_tr_val.shape)
sim_ques_tr_val.head()
```

```
(6796, 3)
```

	train_ques	val_ques	sim_score
0	What is the main cause of HIV-1 infection in c...	What is the role of antibodies during infection?	0.507093
1	What is the main cause of HIV-1 infection in c...	What are the main groups for Mammarenaviruses?	0.507093
2	What is the main cause of HIV-1 infection in c...	What do the acute exacerbations cause?	0.507093
3	What is the main cause of HIV-1 infection in c...	What does the epithelial proteins cause?	0.507093
4	What is the main cause of HIV-1 infection in c...	What does infection of respiratory viruses cause?	0.617213

As the context is the passage which have answer for more than 10 question creating new context so that we can fit that in a smaller model

```
new_context=[]
ques = []
answer = []
start_ind=[]
end_ind=[]
var = 350
# data.shape[0]
for i in range(data.shape[0]):
    ans = data.loc[i,'answers']
    que = data.loc[i,'questions']
    con = data.loc[i,'context']
```

```
        abstract = con.find("Abstract:") +10
        con = con[abstract:]
        con = con.replace("BACKGROUND: ","")
        con = con.replace("CONCLUSION: ","")

        con = con.replace("METHODS AND FINDINGS: ","")
        con = con.replace("Text: ","")
        con = re.sub(r'\[.\]','',con)
        con = re.sub(r'\[..\]','',con)
        con = re.sub(r'\[..,...\]','',con)
        con = re.sub(r'\n',' ',con)
        con = re.sub(r'  ','',con)

        ans = re.sub(r'\[.\]','',ans)
        ans = re.sub(r'\[..\]','',ans)
        ans = re.sub(r'\[..,...\]','',ans)
        ans = re.sub(r'\n',' ',ans)
        ans = re.sub(r'  ','',ans)

        index = con.find(ans)
        if index < var:
            num = var-index
            text = con[index-index:index+600+num]
            reverse =text[::-1]
            stop = reverse.find(' .')
            text = (text[:-stop])
            index = text.find(ans)
            end_index = index + len(ans)
            new_context.append(text)
            start_ind.append(index)
            end_ind.append(end_index)
            answer.append(ans)
            ques.append(que)
        else:
            temp = len(ans)
            text = con[index-var:index+temp+var]
            stop_f = text.find('. ')
            if stop_f < var:
                text=text[stop_f+2:]
            reverse =text[::-1]
            stop_e = reverse.find(' .')
            text = (text[:-stop_e])
            index = text.find(ans)
            end_index = index + len(ans)
            new_context.append(text)
            start_ind.append(index)
            end_ind.append(end_index)
            answer.append(ans)
            ques.append(que)

        # print(i)
        # print("context - ",text)
        # print("len of con = ",len(text))
        # print("que = ",que)
        # print("r ans = ",ans)
        # print("c ans = ",text[index:end_index])


    f_data = pd.DataFrame(data=list(zip(ques, answer,new_context,start_ind,end_ind
    print(f_data.shape)
    f_data.index.name = 'id'
    f_data.head()
```

```
(2014, 5)
```

id	questions	answers	context	starting_index	ending_index
0	What is the main cause of HIV-1 infection in c...	Mother-to-child transmission (MTCT) is the mai...	Mother-to-child transmission (MTCT) is the mai...	0	96
1	What plays the crucial role in the Mother to C...	DC-SIGNR plays a crucial role in MTCT of HIV-1...	In homozygous H1 infants bearing both the p-19...	294	420
2	How many children were	more than 400,000 children	These results suggest that		
3	What is the role of C-C	High copy numbers of	CCR5 promoter variants		

```
f_data = f_data[f_data['starting_index']>-1]
```

```
f_data.shape
```

```
(1854, 5)
```

```
f_data['tol_len'] = f_data['ending_index'] - f_data['starting_index']
```

```
f_data.head()
```

id	questions	answers	context	starting_index	ending_index	tol_len
0	What is the main cause of HIV-1 infection in c...	Mother-to-child transmission (MTCT) is the mai...	Mother-to-child transmission (MTCT) is the mai...	0	96	96
1	What plays the crucial role in the Mother to C...	DC-SIGNR plays a crucial role in MTCT of HIV-1...	The promoter variant reduced transcriptional a...	358	484	126
2	How many children were infected by HIV-1 in 20...	more than 400,000 children were infected world...	In homozygous H1 infants bearing both the p-19...	569	686	117
3	What is the role of C-C Motif Chemokine	High copy numbers of CCL3L1, a potent HIV-1	Genetic variants in CCR5 have been shown	402	582	180

```
starting_index=[]
ending_index=[]
data=None
for i in range(f_data.shape[0]):
    f_dict = dict()
    ans = f_data.loc[i,'answers']
    que = f_data.loc[i,'questions']
    con = f_data.loc[i,'context']
    st = f_data.loc[i,'starting_index']
    en = f_data.loc[i,'ending_index']
    new_st_ind = len(con[:st].split())
    new_end_ind = len(con[:en].split())
    ll = con.split()
    tt = ll[new_st_ind:new_end_ind]
    starting_index.append(new_st_ind)
    ending_index.append(new_end_ind)
    # m_f_data.append(list(f_dict))
    # data  = json.dumps(f_dict)
```

```
f_data['starting_index'] = starting_index
f_data['ending_index'] = ending_index
f_data.head()
```

	questions	answers	context	starting_index	ending_index
0	What is the main cause of HIV-1 infection in c...	Mother-to-child transmission (MTCT) is the mai...	Mother-to-child transmission (MTCT) is the mai...	0	13
1	What plays the crucial role in the Mother to C...	DC-SIGNR plays a crucial role in MTCT of HIV-1...	The promoter variant reduced transcriptional a...	49	68

```
print(f_data.loc[3,'answers'])
print(f_data.loc[3,'questions'])
```

```
    High copy numbers of CCL3L1, a potent HIV-1 suppressive ligand for CCR5, are associated with higher chem
    What is the role of C-C Motif Chemokine Ligand 3 Like 1 (CCL3L1) in mother to child transmission of HIV-
```

```
f_data = f_data[f_data['tol_len']<200]
```

```
f_data = f_data[f_data['con_que_len']<512]
```

```
f_data.shape
```

```
    (1679, 5)
```

```
f_data = f_data[f_data['starting_index']>-1]
```

```
msk = np.random.rand(len(f_data)) < 0.02
test = f_data[msk]
all = f_data[~msk]
```

```
test.shape
```

```
    (36, 5)
```

```
msk = np.random.rand(len(all)) < 0.2
val = all[msk]
train = all[~msk]
```

```
train.shape
```

```
    (1491, 5)
```

```
train_contexts = list(train['context'].values)
train_questions = list(train['questions'].values)
val_contexts = list(val['context'].values)
val_questions = list(val['questions'].values)
```

```
t = list(train['answers'].values)
ass = list(train['starting_index'].values)
ae = list(train['ending_index'].values)
```

```
train_answers = []
for i in range(len(t)):
    f_dict = dict()
    text = t[i]
    answer_start = ass[i]
    answer_end = ae[i]
    f_dict['text'] = text
    f_dict['answer_start']= answer_start
    f_dict['answer_end']= answer_end
    train_answers.append(f_dict)
```

```
train_answers[0]
```

```
{'answer_end': 96,
 'answer_start': 0,
 'text': 'Mother-to-child transmission (MTCT) is the main cause of HIV-1 infection in children worldwide
```

```
t = list(val['answers'].values)
ass = list(val['starting_index'].values)
ae = list(val['ending_index'].values)


val_answers = []
for i in range(len(t)):
    f_dict = dict()
    text = t[i]
    answer_start = ass[i]
    answer_end = ae[i]
    f_dict['text'] = text
    f_dict['answer_start']= answer_start
    f_dict['answer_end']= answer_end
    val_answers.append(f_dict)


from transformers import DistilBertTokenizerFast
from transformers import AutoTokenizer
# distilbert-base-uncased-distilled-squad 2.8
# tokenizer = AutoTokenizer.from_pretrained('bert-large-uncased-whole-word-mas
# from transformers import AutoTokenizer, AutoModelForQuestionAnswering

tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")


train_encodings = tokenizer(train_contexts, train_questions,  padding=True)
val_encodings = tokenizer(val_contexts, val_questions,  padding=True)
```

```
Downloading: 100%                    232k/232k [00:01<00:00, 155kB/s]

Downloading: 100%                    466k/466k [00:00<00:00, 582kB/s]
```

```
tokenizer.model_max_length
```

```
512
```

```
def add_token_positions(encodings, answers):
    start_positions = []
    end_positions = []
    for i in range(len(answers)):
        # print(answers[i]['answer_start'])
        start_positions.append(encodings.char_to_token(i, answers[i]['answer_s
        end_positions.append(encodings.char_to_token(i, answers[i]['answer_end

        # if start position is None, the answer passage has been truncated
        if start_positions[-1] is None:
            # start_positions[-1] = tokenizer.model_max_length
            start_positions[-1] = 512

        # if end position is None, the 'char_to_token' function points to the
        if end_positions[-1] is None:
            # end_positions[-1] = tokenizer.model_max_length
            end_positions[-1] = 512
    encodings.update({'start_positions': start_positions, 'end_positions': end

add_token_positions(train_encodings, train_answers)
add_token_positions(val_encodings, val_answers)
```

```
import torch

class SquadDataset(torch.utils.data.Dataset):
    def __init__(self, encodings):
        self.encodings = encodings

    def __getitem__(self, idx):
        return {key: torch.tensor(val[idx]) for key, val in self.encodings.ite

    def __len__(self):
        return len(self.encodings.input_ids)

train_dataset = SquadDataset(train_encodings)
val_dataset = SquadDataset(val_encodings)


from transformers import DistilBertForQuestionAnswering, Trainer, TrainingArgu
# from transformers import AutoModelForQuestionAnswering, TrainingArguments, T

# model = AutoModelForQuestionAnswering.from_pretrained('dmis-lab/biobert-base
model = DistilBertForQuestionAnswering.from_pretrained("distilbert-base-uncase

training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=1,
    learning_rate=2e-5,
    # evaluation_strategy = "epoch",            # total number of training epc
    per_device_train_batch_size=1,  # batch size per device during training
    per_device_eval_batch_size=1,   # batch size for evaluation
    warmup_steps=200,                # number of warmup steps for learning rat
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    logging_steps=1,
    do_eval=True,
    # evaluate_during_training=True

)

# model = DistilBertForQuestionAnswering.from_pretrained("distilbert-base-unca

trainer = Trainer(
    model=model,                         # the instantiated 🤗 Transformers m
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=val_dataset             # evaluation dataset
)

trainer.train()


tokenizer.save_pretrained('/content/results')

     ('/content/results/tokenizer_config.json',
      '/content/results/special_tokens_map.json',
      '/content/results/vocab.txt',
      '/content/results/added_tokens.json')


trainer.save_model()


from transformers import AutoTokenizer, AutoModelForQuestionAnswering

# Load the fine-tuned model
tokenizer = AutoTokenizer.from_pretrained("/content/results")
model = AutoModelForQuestionAnswering.from_pretrained("/content/results")


from transformers import DistilBertTokenizerFast, DistilBertForQuestionAnsweri

# Load the fine-tuned model
```

```
tokenizer = DistilBertTokenizerFast.from_pretrained("/content/results")
model = DistilBertForQuestionAnswering.from_pretrained("/content/results")


context = 'Although antiretrovirals can reduce MTCT to 2%, limited access to t
ques = 'What is DC-GENR and where is  it expressed?'
nlp_qa = pipeline('question-answering',model=model,tokenizer=tokenizer)
ans = nlp_qa(context=context, question=ques)
print(ans)
```

```
{'score': 0.060548827052116394, 'start': 597, 'end': 638, 'answer': 'in placental capillary endothelial
```

Trying to pridict from the trained model directly but taking the index from context provided with the question rather than the tokenized context

```
# xx = tokenizer(  padding=True)
# yy = SquadDataset(train_encodings)
inputs = tokenizer.encode_plus(context,ques, return_tensors="pt")


x = model(**inputs)


answer_start_scores = (x['start_logits'])
answer_end_scores = (x['end_logits'])


answer_start = torch.argmax(answer_start_scores)  # get the most likely beginn
answer_end = torch.argmax(answer_end_scores) + 1  # get the most likely end of
print(answer_start)
print(answer_end)
# 3. GET THE ANSWER SPAN
# once we have the most likely start and end tokens, we grab all the tokens be
# and convert tokens back to words!
# tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["i
te = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs
print(context[answer_start:answer_end])
```

```
tensor(23)
tensor(155)
5 have been shown to influence vertical transmission of HIV-1. CCR5 promoter variants resulting in highe
```

```
p_list=[]
r_list=[]
dp = {'id':10,'prediction_text':'5 have been shown to influence vertical trans
dr = {'id':10,'answers':{'text':['dsad dasda das dasd daksd asdisd asd sa did
p_list.append(dp)
r_list.append(dr)


context='Genetic variants in CCR5 have been shown to influence vertical transm
question='What is the role of C-C Motif Chemokine Ligand 3 Like 1 (CCL3L1) in
nlp_qa = pipeline('question-answering',model=model,tokenizer=tokenizer)
nlp_qa(context=context,question=question )
```

```
{'answer': '.', 'end': 583, 'score': 0.00016867084195837379, 'start': 582}
```

Now lets check the results using pre-trained model on same question whether we have improvement on them or not

Using Pretrain model for question and answer testing how good it is providing the result on couple of examples-

```
context = 'Although antiretrovirals can reduce MTCT to 2%, limited access to t
ques = 'What is DC-GENR and where is  it expressed?'
nlp_qa = pipeline('question-answering')
```

```
nlp_qa = pipeline('question-answering')
nlp_qa(context=context, question=ques)
```

```
Downloading: 100%  473/473 [00:05<00:00, 94.1B/s]
Downloading: 100%  261M/261M [00:04<00:00, 58.0MB/s]
Downloading: 100%  213k/213k [00:02<00:00, 99.6kB/s]
Downloading: 100%  436k/436k [00:01<00:00, 297kB/s]
```

```
{'answer': 'in placental capillary endothelial cells',
 'end': 637,
 'score': 0.06756021082401276,
 'start': 597}
```

```
nlp_qa = pipeline('question-answering')
nlp_qa(context='Genetic variants in CCR5 have been shown to influence vertical
```

```
{'answer': 'higher chemokine production',
 'end': 522,
 'score': 0.5519561767578125,
 'start': 495}
```

Bellow following another tutorial from hugging face where we can handle overlap between two part of the context and also while evaluation we can get top 20 probablities for start and end index.

```
pip install datasets
```

```
Collecting datasets
  Downloading https://files.pythonhosted.org/packages/06/9b/d097f2238fc3c028495cf5f8c65378972b9f1b2cbb27
     |████████████████████████████████| 163kB 8.1MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.6/dist-packages (from datasets) (1.
Requirement already satisfied: multiprocess in /usr/local/lib/python3.6/dist-packages (from datasets) (0
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.6/dist-packages (from datasets
Requirement already satisfied: dill in /usr/local/lib/python3.6/dist-packages (from datasets) (0.3.3)
Collecting xxhash
  Downloading https://files.pythonhosted.org/packages/f7/73/826b19f3594756cb1c6c23d2fbd8ca6a77a9cd3b650c
     |████████████████████████████████| 245kB 13.6MB/s
Requirement already satisfied: tqdm<4.50.0,>=4.27 in /usr/local/lib/python3.6/dist-packages (from datase
Collecting pyarrow>=0.17.1
  Downloading https://files.pythonhosted.org/packages/33/67/2f4fcce1b41bcc7e88a6bfdb42046597ae72e5bc95c2
     |████████████████████████████████| 20.7MB 1.3MB/s
Requirement already satisfied: dataclasses; python_version < "3.7" in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (from datasets) (1.1.5)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/di
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from request
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from reques
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-packages (from pa
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas->data
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in /usr/local/lib/pythor
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from importlib-metac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>
Installing collected packages: xxhash, pyarrow, datasets
  Found existing installation: pyarrow 0.14.1
    Uninstalling pyarrow-0.14.1:
      Successfully uninstalled pyarrow-0.14.1
Successfully installed datasets-1.2.1 pyarrow-3.0.0 xxhash-2.0.0
```

```
from datasets import load_dataset,load_metric
```

Here the data is preprocessed in a way that load_dataset function can read it.

```
datasets = load_dataset('csv', data_files={'train': '/content/drive/MyDrive/Py
```

```
Downloading:  5.33k/? [00:00<00:00, 48.5kB/s]
```

```
Using custom data configuration default

Downloading and preparing dataset csv/default-5cfdb25bb9458455 (download: Unknown size, generated: Unknc
Dataset csv downloaded and prepared to /root/.cache/huggingface/datasets/csv/default-5cfdb25bb9458455/0.
```

```
datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'questions', 'context', 'answers'],
        num_rows: 1477
    })
    test: Dataset({
        features: ['id', 'questions', 'context', 'answers'],
        num_rows: 377
    })
})
```

```
datasets["train"][0]
```

```
{'answers': "{'text': 'DC-SIGNR plays a crucial role in MTCT of HIV-1 and that impaired placental DC-SIG
 'context': 'In homozygous H1 infants bearing both the p-198A and int2-180A mutations, we observed a 4-f
 'id': 1,
 'questions': 'What plays the crucial role in the Mother to Child Transmission of HIV-1 and what increas
```

```
from transformers import AutoTokenizer
model_checkpoint = 'distilbert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
Downloading: 100%  442/442 [00:00<00:00, 13.5kB/s]

Downloading: 100%  232k/232k [00:00<00:00, 337kB/s]

Downloading: 100%  466k/466k [00:00<00:00, 1.48MB/s]
```

```
max_length = 384 # The maximum length of a feature (question and context)
doc_stride = 128 # The authorized overlap between two part of the context when
batch_size = 16
```

```
pad_on_right = tokenizer.padding_side == "right"
```

```
def prepare_train_features(examples):
    # Tokenize our examples with truncation and padding, but keep the overflow
    # in one example possible giving several features when a context is long,
    # context that overlaps a bit the context of the previous feature.
    tokenized_examples = tokenizer(
        examples["questions" if pad_on_right else "context"],
        examples["context" if pad_on_right else "question"],
        truncation="only_second" if pad_on_right else "only_first",
        max_length=max_length,
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )
    # print(examples["questions"])
    # print(examples["context"])
    # Since one example might give us several features if it has a long contex
```

```
        # its corresponding example. This key gives us just that.
        sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")
        # The offset mappings will give us a map from token to character position
        # help us compute the start_positions and end_positions.
        offset_mapping = tokenized_examples.pop("offset_mapping")

        # Let's label those examples!
        tokenized_examples["start_positions"] = []
        tokenized_examples["end_positions"] = []

        for i, offsets in enumerate(offset_mapping):
            # We will label impossible answers with the index of the CLS token.
            input_ids = tokenized_examples["input_ids"][i]
            cls_index = input_ids.index(tokenizer.cls_token_id)

            # Grab the sequence corresponding to that example (to know what is the
            sequence_ids = tokenized_examples.sequence_ids(i)

            # One example can give several spans, this is the index of the example
            sample_index = sample_mapping[i]
            answers = examples["answers"][sample_index]
            reverse =answers[::-1]
            stop_e = reverse.find(' :')
            start_i = int(reverse[1:stop_e])
            # print(start_i)
            # print((answers))
            xx = answers.find(': ')
            yy = answers.find('answer_start')

            st = (answers[xx+3:yy-4])

            # print(answers[-2])
            start_char = start_i
            end_char = start_char + len(st)

            # Start token index of the current span in the text.
            token_start_index = 0
            while sequence_ids[token_start_index] != (1 if pad_on_right else 0):
                token_start_index += 1

            # End token index of the current span in the text.
            token_end_index = len(input_ids) - 1
            while sequence_ids[token_end_index] != (1 if pad_on_right else 0):
                token_end_index -= 1

            # Detect if the answer is out of the span (in which case this feature
            if not (offsets[token_start_index][0] <= start_char and offsets[token_
                tokenized_examples["start_positions"].append(cls_index)
                tokenized_examples["end_positions"].append(cls_index)
            else:
                # Otherwise move the token_start_index and token_end_index to the
                # Note: we could go after the last offset if the answer is the las
                while token_start_index < len(offsets) and offsets[token_start_ind
                    token_start_index += 1
                tokenized_examples["start_positions"].append(token_start_index - 1
                while offsets[token_end_index][1] >= end_char:
                    token_end_index -= 1
                tokenized_examples["end_positions"].append(token_end_index + 1)

        return tokenized_examples


    tokenized_datasets = datasets.map(prepare_train_features, batched=True, remove
```

```
tokenized_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['attention_mask', 'end_positions', 'input_ids', 'start_positions'],
        num_rows: 1477
    })
    test: Dataset({
        features: ['attention_mask', 'end_positions', 'input_ids', 'start_positions'],
        num_rows: 377
    })
})
```

```
from transformers import AutoModelForQuestionAnswering, TrainingArguments, Tra

model = AutoModelForQuestionAnswering.from_pretrained(model_checkpoint)
```

```
Downloading: 100%  268M/268M [02:20<00:00, 1.91MB/s]

Some weights of the model checkpoint at distilbert-base-uncased were not used when initializing DistilBe
- This IS expected if you are initializing DistilBertForQuestionAnswering from the checkpoint of a model
- This IS NOT expected if you are initializing DistilBertForQuestionAnswering from the checkpoint of a m
Some weights of DistilBertForQuestionAnswering were not initialized from the model checkpoint at distilb
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and infe
```

```
args = TrainingArguments(
    f"test-squad",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=10,
    weight_decay=0.01,
)
```

```
from transformers import default_data_collator

data_collator = default_data_collator
```

```
trainer = Trainer(
    model,
    args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```
trainer.train()
```

[930/930 10:03, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Runtime	Samples Per Second
1	No log	3.589539	4.539200	83.055000
2	No log	3.565194	4.684900	80.471000
3	No log	3.647276	4.787500	78.747000

```
import torch
for batch in trainer.get_eval_dataloader():
    break
batch = {k: v.to(trainer.args.device) for k, v in batch.items()}
with torch.no_grad():
    output = trainer.model(**batch)
    print(output['loss'])
output.keys()
```

```
tensor(7.0231, device='cuda:0')
odict_keys(['loss', 'start_logits', 'end_logits'])
```

```
TrainOutput(global_step=930, training_loss=2.553336654170867, metrics={'train_runtime': 604.8239, 'train
```

```
output.start_logits.shape, output.end_logits.shape
```

```
(torch.Size([16, 384]), torch.Size([16, 384]))
```

```
n_best_size = 20
```

```
import numpy as np

start_logits = output.start_logits[0].cpu().numpy()
end_logits = output.end_logits[0].cpu().numpy()
# Gather the indices the best start/end logits:
start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 : -1].tolist()
end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].tolist()
valid_answers = []
for start_index in start_indexes:
    for end_index in end_indexes:
        if start_index <= end_index: # We need to refine that test to check th
            valid_answers.append(
                {
                    "score": start_logits[start_index] + end_logits[end_index]
                    "text": "" # We need to find a way to get back the origina
                }
            )
```

```
def prepare_validation_features(examples):
    # Tokenize our examples with truncation and maybe padding, but keep the ov
    # in one example possible giving several features when a context is long,
    # context that overlaps a bit the context of the previous feature.
    tokenized_examples = tokenizer(
        examples["questions" if pad_on_right else "context"],
        examples["context" if pad_on_right else "question"],
        truncation="only_second" if pad_on_right else "only_first",
        max_length=max_length,
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    # Since one example might give us several features if it has a long contex
    # its corresponding example. This key gives us just that.
    sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")

    # We keep the example_id that gave us this feature and we will store the c
    tokenized_examples["example_id"] = []
```

```
        for i in range(len(tokenized_examples["input_ids"])):
            # Grab the sequence corresponding to that example (to know what is the
            sequence_ids = tokenized_examples.sequence_ids(i)
            context_index = 1 if pad_on_right else 0

            # One example can give several spans, this is the index of the example
            sample_index = sample_mapping[i]
            tokenized_examples["example_id"].append(examples["id"][sample_index])

            # Set to None the offset_mapping that are not part of the context so i
            # position is part of the context or not.
            tokenized_examples["offset_mapping"][i] = [
                (o if sequence_ids[k] == context_index else None)
                for k, o in enumerate(tokenized_examples["offset_mapping"][i])
            ]

        return tokenized_examples


    validation_features = datasets["test"].map(
        prepare_validation_features,
        batched=True,
        remove_columns=datasets["test"].column_names
    )
```

100% 1/1 [00:00<00:00, 1.47ba/s]

```
    raw_predictions = trainer.predict(validation_features)
```

[24/24 00:04]

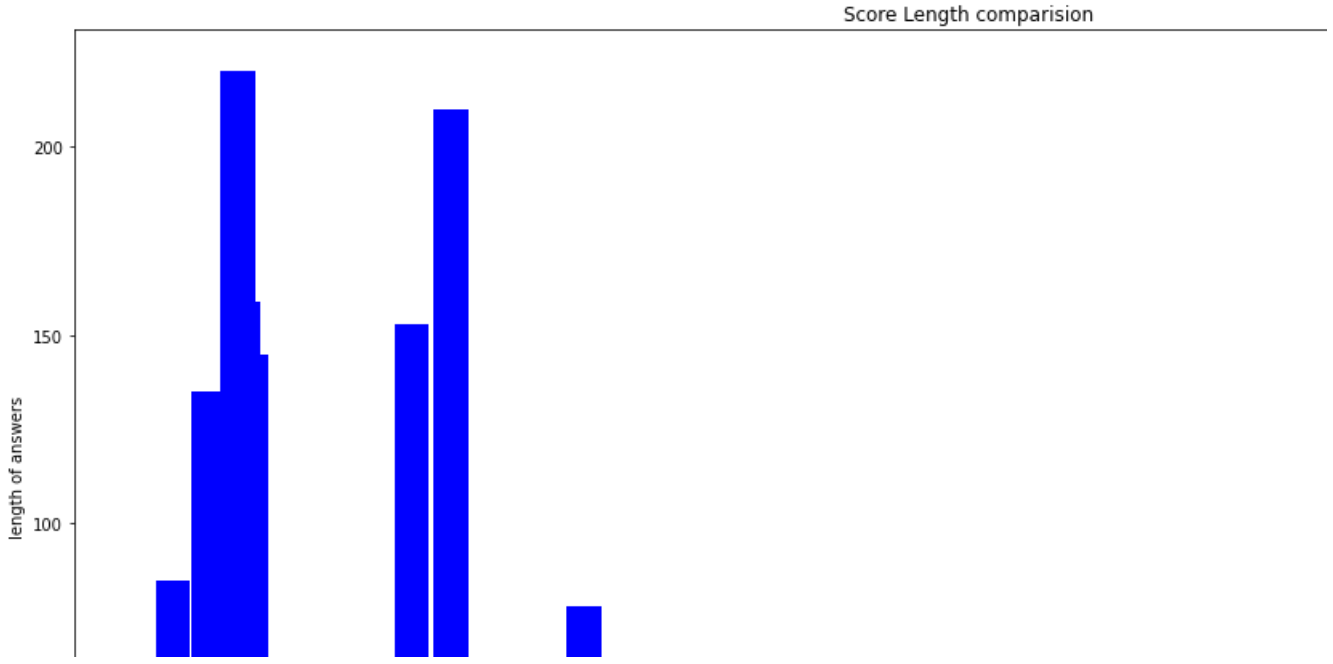
```
    validation_features.set_format(type=validation_features.format["type"], column


    max_answer_length = 60


    start_logits = output.start_logits[0].cpu().numpy()
    end_logits = output.end_logits[0].cpu().numpy()
    offset_mapping = validation_features[0]["offset_mapping"]
    # The first feature comes from the first example. For the more general case, w
    # an example index
    context = datasets["test"][0]["context"]

    # Gather the indices the best start/end logits:
    start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 : -1].tolist()
    end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].tolist()
    valid_answers = []
    for start_index in start_indexes:
        for end_index in end_indexes:
            # Don't consider out-of-scope answers, either because the indices are
            # to part of the input_ids that are not in the context.
            if (
                start_index >= len(offset_mapping)
                or end_index >= len(offset_mapping)
                or offset_mapping[start_index] is None
                or offset_mapping[end_index] is None
            ):
                continue
            # Don't consider answers with a length that is either < 0 or > max_ans
            if end_index < start_index or end_index - start_index + 1 > max_answer
                continue
            if start_index <= end_index: # We need to refine that test to check th
                start_char = offset_mapping[start_index][0]
                end_char = offset_mapping[end_index][1]
                valid_answers.append(
                    {
```

```
                `
                    "score": start_logits[start_index] + end_logits[end_index]
                    "text": context[start_char: end_char]
                }
            )

valid_answers = sorted(valid_answers, key=lambda x: x["score"], reverse=True)[
valid_answers
```

```
[{'score': 4.8534594, 'text': 'worldwide'},
 {'score': 3.0185757, 'text': 'infection in children worldwide'},
 {'score': 2.5086603, 'text': 'worldwide.'},
 {'score': 1.6493661, 'text': 'worldwide. Given'},
 {'score': 1.4629529, 'text': 'infection'},
 {'score': 1.4372299,
  'text': 'transmission (MTCT) is the main cause of HIV-1 infection in children worldwide'},
 {'score': 1.2819654, 'text': 'of HIV-1 infection in children worldwide'},
 {'score': 1.1000681, 'text': 'children worldwide'},
 {'score': 1.0542297,
  'text': 'investigate the potential role of DC-SIGNR in MTCT of HIV-1, we carried out a genetic associa
 {'score': 1.0454994, 'text': 'Hara'},
 {'score': 0.94033813,
  'text': 'worldwide. Given that the C-type lectin receptor, dendritic cell-specific ICAM-grabbing non-i
 {'score': 0.83317846, 'text': 'investigate'},
 {'score': 0.67377675, 'text': 'infection in children worldwide.'},
 {'score': 0.4824338, 'text': 'HIV-1 infection in children worldwide'},
 {'score': 0.47425163,
  'text': 'pathogens including HIV-1 and is expressed at the maternal-fetal interface, we hypothesized t
 {'score': 0.4509455,
  'text': 'interact with pathogens including HIV-1 and is expressed at the maternal-fetal interface, we
 {'score': 0.4361335,
  'text': 'ymph node–specific ICAM-grabbing non-integrin (L-SIGN)), can interact with pathogens includir
 {'score': 0.3817718,
  'text': ') is the main cause of HIV-1 infection in children worldwide'},
 {'score': 0.35343444,
  'text': 'including HIV-1 and is expressed at the maternal-fetal interface, we hypothesized that it cou
 {'score': 0.25012124,
  'text': 'investigate the potential role of DC-SIGNR in MTCT of HIV-1, we carried out a genetic'}]
```

```
score=[]
length =[]
for i in range(20):
  score.append(valid_answers[i]['score'])
  length.append(len(valid_answers[i]['text']))

print(score)
print(length)
fig = plt.figure(figsize = (20, 10))

# creating the bar plot
plt.bar(score, length, color ='blue',
        width = 0.1)

plt.xlabel("Scores range")
plt.ylabel("length of answers")
plt.title("Score Length comparision")
plt.show()
```

```
[4.8534594, 3.0185757, 2.5086603, 1.6493661, 1.4629529, 1.4372299, 1.2819654, 1.1000681, 1.0542297, 1.04
[9, 31, 10, 16, 9, 78, 40, 18, 210, 4, 153, 11, 32, 37, 145, 159, 220, 60, 135, 85]
```



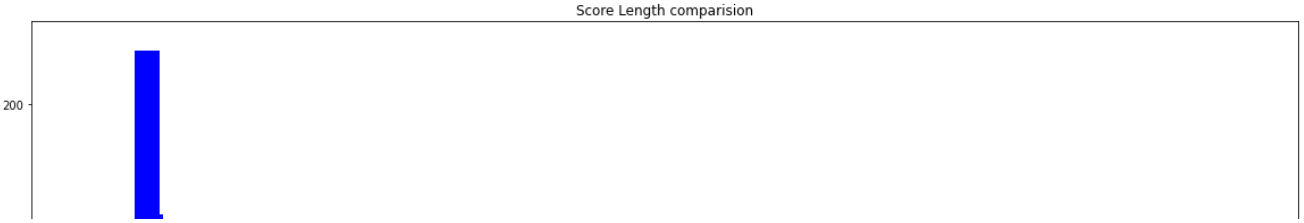

```
start_logits = output.start_logits[0].cpu().numpy()
end_logits = output.end_logits[0].cpu().numpy()
offset_mapping = validation_features[0]["offset_mapping"]
# The first feature comes from the first example. For the more general case, w
# an example index
context = datasets["test"][5]["context"]

# Gather the indices the best start/end logits:
start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 : -1].tolist()
end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].tolist()
valid_answers = []
for start_index in start_indexes:
    for end_index in end_indexes:
        # Don't consider out-of-scope answers, either because the indices are
        # to part of the input_ids that are not in the context.
        if (
            start_index >= len(offset_mapping)
            or end_index >= len(offset_mapping)
            or offset_mapping[start_index] is None
            or offset_mapping[end_index] is None
        ):
            continue
        # Don't consider answers with a length that is either < 0 or > max_ans
        if end_index < start_index or end_index - start_index + 1 > max_answer
            continue
        if start_index <= end_index: # We need to refine that test to check th
            start_char = offset_mapping[start_index][0]
            end_char = offset_mapping[end_index][1]
            valid_answers.append(
                {
                    "score": start_logits[start_index] + end_logits[end_index]
                    "text": context[start_char: end_char]
                }
            )

valid_answers = sorted(valid_answers, key=lambda x: x["score"], reverse=True)[
valid_answers
```

```
[{'score': 4.8534594, 'text': 'bled and '},
 {'score': 3.0185757, 'text': '. Sequences were assembled and '},
 {'score': 2.5086603, 'text': 'bled and a'},
 {'score': 1.6493661, 'text': 'bled and annotat'},
 {'score': 1.4629529, 'text': '. Sequenc'},
 {'score': 1.4372299,
  'text': 'ions were performed as previously described (3). Sequences were assembled and '},
```

```
{'score': 1.2819654, 'text': 'ribed (3). Sequences were assembled and '},
{'score': 1.1000681, 'text': 'ere assembled and '},
{'score': 1.0542297,
 'text': 'coronavirus (DcCoV) HKU23-23-362F strain from the United Arab Emirates (accession no. KF90625
{'score': 1.0454994, 'text': ''},
{'score': 0.94033813,
 'text': 'bled and annotated using the Geneious software (version 5.1.6). We obtained a sequence counti
{'score': 0.83317846, 'text': 'coronavirus'},
{'score': 0.67377675, 'text': '. Sequences were assembled and a'},
{'score': 0.4824338, 'text': 'ed (3). Sequences were assembled and '},
{'score': 0.47425163,
 'text': 'to these analyses, the orf1ab (20kb nucleotides, located at the 5= side of the genome) gene i
{'score': 0.4509455,
 'text': 'is. According to these analyses, the orf1ab (20kb nucleotides, located at the 5= side of the
{'score': 0.4361335,
 'text': ' genes of the obtained BCoV were submitted to a Blastn analysis. According to these analyses,
{'score': 0.3817718,
 'text': 'd as previously described (3). Sequences were assembled and '},
{'score': 0.35343444,
 'text': 'nalyses, the orf1ab (20kb nucleotides, located at the 5= side of the genome) gene is closely
{'score': 0.25012124,
 'text': 'coronavirus (DcCoV) HKU23-23-362F strain from the United Arab Emirates (accession no.'}]
```

```
score=[]
length =[]
for i in range(20):
  score.append(valid_answers[i]['score'])
  length.append(len(valid_answers[i]['text']))

print(score)
print(length)
fig = plt.figure(figsize = (20, 10))

# creating the bar plot
plt.bar(score, length, color ='blue',
        width = 0.1)

plt.xlabel("Scores range")
plt.ylabel("length of answers")
plt.title("Score Length comparision")
plt.show()
```

```
[4.8534594, 3.0185757, 2.5086603, 1.6493661, 1.4629529, 1.4372299, 1.2819654, 1.1000681, 1.0542297, 1.04
[9, 31, 10, 16, 9, 78, 40, 18, 135, 0, 153, 11, 32, 37, 145, 159, 220, 60, 135, 85]
```



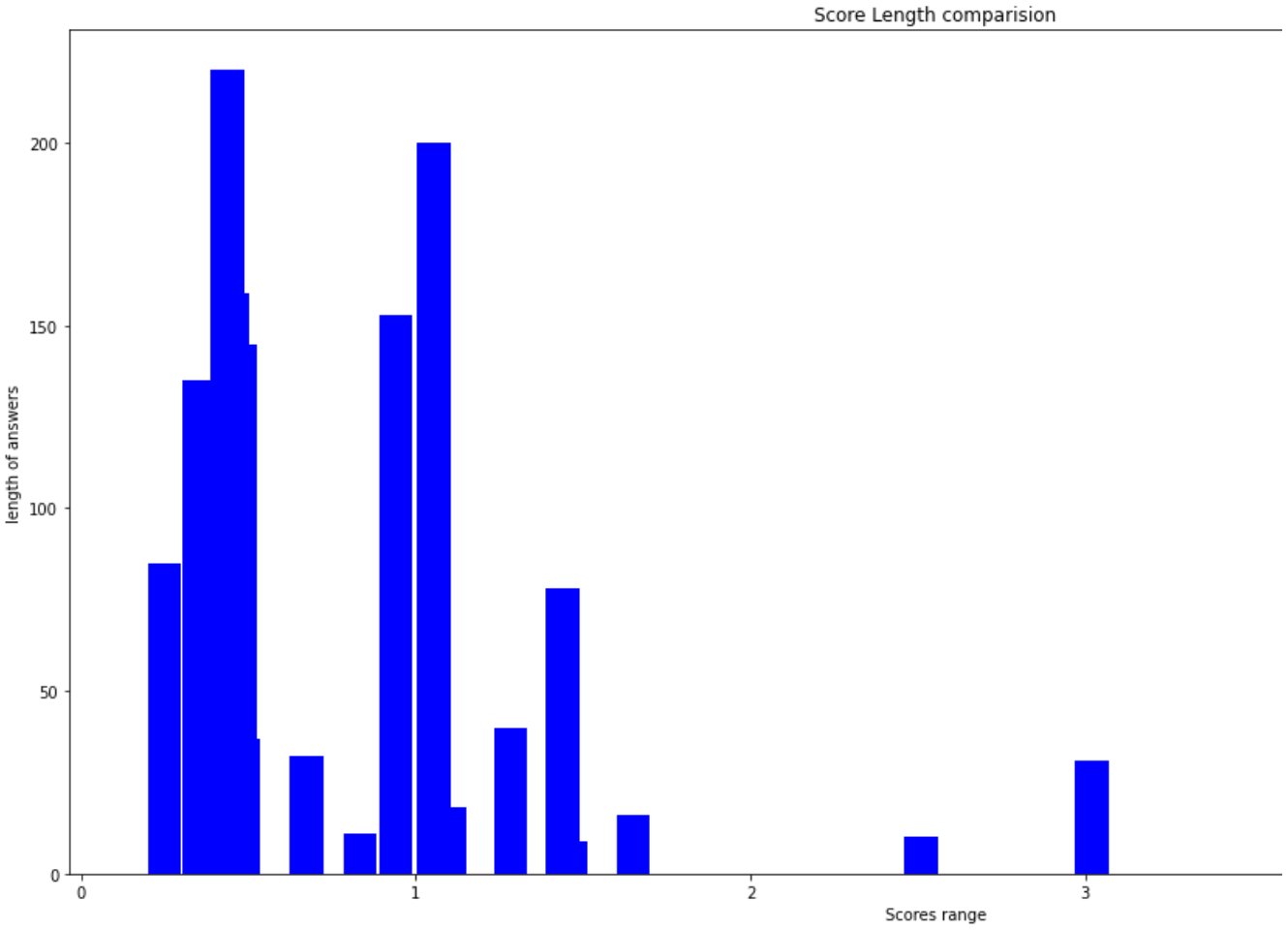

```
start_logits = output.start_logits[0].cpu().numpy()
end_logits = output.end_logits[0].cpu().numpy()
offset_mapping = validation_features[0]["offset_mapping"]
# The first feature comes from the first example. For the more general case, w
# an example index
context = datasets["test"][10]["context"]

# Gather the indices the best start/end logits:
start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 : -1].tolist()
end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].tolist()
valid_answers = []
for start_index in start_indexes:
    for end_index in end_indexes:
        # Don't consider out-of-scope answers, either because the indices are
        # to part of the input_ids that are not in the context.
        if (
            start_index >= len(offset_mapping)
            or end_index >= len(offset_mapping)
            or offset_mapping[start_index] is None
            or offset_mapping[end_index] is None
        ):
            continue
        # Don't consider answers with a length that is either < 0 or > max_ans
        if end_index < start_index or end_index - start_index + 1 > max_answer
            continue
        if start_index <= end_index: # We need to refine that test to check th
            start_char = offset_mapping[start_index][0]
            end_char = offset_mapping[end_index][1]
            valid_answers.append(
                {
                    "score": start_logits[start_index] + end_logits[end_index]
                    "text": context[start_char: end_char]
                }
            )

valid_answers = sorted(valid_answers, key=lambda x: x["score"], reverse=True)[
valid_answers
```

```
[{'score': 4.8534594, 'text': '. An ELIS'},
 {'score': 3.0185757, 'text': 's available clinically. An ELIS'},
 {'score': 2.5086603, 'text': '. An ELISA'},
 {'score': 1.6493661, 'text': '. An ELISA-array'},
 {'score': 1.4629529, 'text': 's availab'},
 {'score': 1.4372299,
  'text': ' are still no multiple antigen detection methods available clinically. An ELIS'},
 {'score': 1.2819654, 'text': 'on methods available clinically. An ELIS'},
 {'score': 1.1000681, 'text': 'linically. An ELIS'},
 {'score': 1.0542297,
  'text': 'he ELISA-array. The ELISA-array assay is based on a "sandwich" ELISA format and consists of v
 {'score': 1.0454994, 'text': ''},
 {'score': 0.94033813,
  'text': '. An ELISA-array, which detects multiple antigens, is easy to handle, and inexpensive, has er
 {'score': 0.83317846, 'text': 'he ELISA-ar'},
 {'score': 0.67377675, 'text': 's available clinically. An ELISA'},
 {'score': 0.4824338, 'text': 'methods available clinically. An ELIS'},
 {'score': 0.47425163,
  'text': 'ed in this study. Seven monoclonal antibodies against five encephalitis-associated viruses we
 {'score': 0.4509455,
  'text': 'es was developed in this study. Seven monoclonal antibodies against five encephalitis-associa
 {'score': 0.4361335,
  'text': 'hod for the simultaneous detection of five encephalitis viruses was developed in this study.
```

```
{'score': 0.3817718,
 'text': 'iple antigen detection methods available clinically. An ELIS'},
{'score': 0.35343444,
 'text': ' study. Seven monoclonal antibodies against five encephalitis-associated viruses were prepare
{'score': 0.25012124,
 'text': 'he ELISA-array. The ELISA-array assay is based on a "sandwich" ELISA format and consi'}]
```

```
score=[]
length =[]
for i in range(20):
  score.append(valid_answers[i]['score'])
  length.append(len(valid_answers[i]['text']))

print(score)
print(length)
fig = plt.figure(figsize = (20, 10))

# creating the bar plot
plt.bar(score, length, color ='blue',
        width = 0.1)

plt.xlabel("Scores range")
plt.ylabel("length of answers")
plt.title("Score Length comparision")
plt.show()
```

```
[4.8534594, 3.0185757, 2.5086603, 1.6493661, 1.4629529, 1.4372299, 1.2819654, 1.1000681, 1.0542297, 1.04
[9, 31, 10, 16, 9, 78, 40, 18, 200, 0, 153, 11, 32, 37, 145, 159, 220, 60, 135, 85]
```




```
start_logits = output.start_logits[0].cpu().numpy()
end_logits = output.end_logits[0].cpu().numpy()
offset_mapping = validation_features[0]["offset_mapping"]
# The first feature comes from the first example. For the more general case, w
```

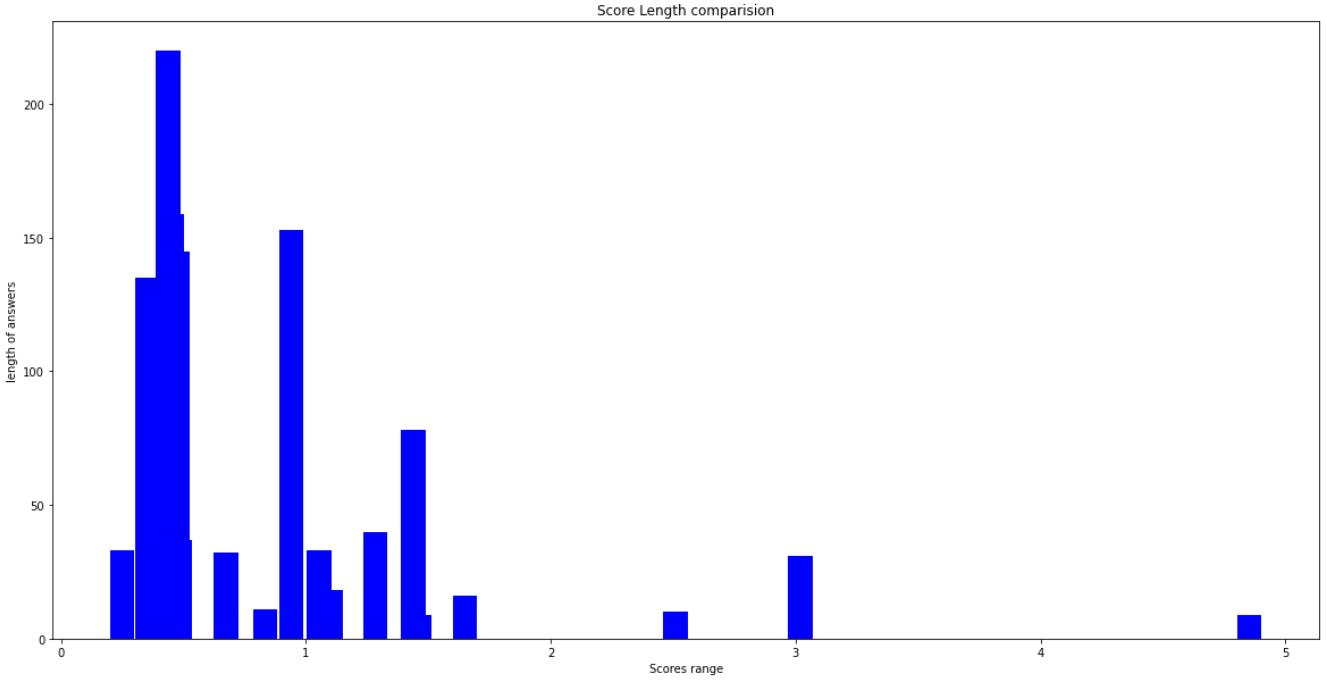
```
# an example index
context = datasets["test"][20]["context"]

# Gather the indices the best start/end logits:
start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 : -1].tolist()
end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].tolist()
valid_answers = []
for start_index in start_indexes:
    for end_index in end_indexes:
        # Don't consider out-of-scope answers, either because the indices are
        # to part of the input_ids that are not in the context.
        if (
            start_index >= len(offset_mapping)
            or end_index >= len(offset_mapping)
            or offset_mapping[start_index] is None
            or offset_mapping[end_index] is None
        ):
            continue
        # Don't consider answers with a length that is either < 0 or > max_ans
        if end_index < start_index or end_index - start_index + 1 > max_answer
            continue
        if start_index <= end_index: # We need to refine that test to check th
            start_char = offset_mapping[start_index][0]
            end_char = offset_mapping[end_index][1]
            valid_answers.append(
                {
                    "score": start_logits[start_index] + end_logits[end_index]
                    "text": context[start_char: end_char]
                }
            )

valid_answers = sorted(valid_answers, key=lambda x: x["score"], reverse=True)[
valid_answers
```

```
[{'score': 4.8534594, 'text': ' from 0.2'},
 {'score': 3.0185757, 'text': 'ncidence of pneumonia, from 0.2'},
 {'score': 2.5086603, 'text': ' from 0.29'},
 {'score': 1.6493661, 'text': ' from 0.29 episo'},
 {'score': 1.4629529, 'text': 'ncidence '},
 {'score': 1.4372299,
  'text': 'est that there has been a 25% decrease in the incidence of pneumonia, from 0.2'},
 {'score': 1.2819654, 'text': ' in the incidence of pneumonia, from 0.2'},
 {'score': 1.1000681, 'text': 'neumonia, from 0.2'},
 {'score': 1.0542297, 'text': 'lion in 2000 to 900,000 in 2013. '},
 {'score': 1.0454994, 'text': ''},
 {'score': 0.94033813,
  'text': ' from 0.29 episodes per child year in low-and middle-income countries in 2000, to 0.22 episoc
 {'score': 0.83317846, 'text': 'lion in 200'},
 {'score': 0.67377675, 'text': 'ncidence of pneumonia, from 0.29'},
 {'score': 0.4824338, 'text': ' the incidence of pneumonia, from 0.2'},
 {'score': 0.47425163,
  'text': '0 and 2013, from 186 million to 78 million as estimated in the Global Burden of Disease study
 {'score': 0.4509455,
  'text': 'rs between 1990 and 2013, from 186 million to 78 million as estimated in the Global Burden of
 {'score': 0.4361335,
  'text': 'decrease in pneumonia-associated disability-adjusted life years between 1990 and 2013, from 1
 {'score': 0.3817718,
  'text': ' been a 25% decrease in the incidence of pneumonia, from 0.2'},
 {'score': 0.35343444,
  'text': ', from 186 million to 78 million as estimated in the Global Burden of Disease study. Pneumoni
 {'score': 0.25012124, 'text': 'lion in 2000 to 900,000 in 2013. '}]
```

```
score=[]
length =[]
for i in range(20):
  score.append(valid_answers[i]['score'])
  length.append(len(valid_answers[i]['text']))

print(score)
print(length)
```

```
fig = plt.figure(figsize = (20, 10))

# creating the bar plot
plt.bar(score, length, color ='blue',
        width = 0.1)

plt.xlabel("Scores range")
plt.ylabel("length of answers")
plt.title("Score Length comparision")
plt.show()
```

```
[4.8534594, 3.0185757, 2.5086603, 1.6493661, 1.4629529, 1.4372299, 1.2819654, 1.1000681, 1.0542297, 1.04
[9, 31, 10, 16, 9, 78, 40, 18, 33, 0, 153, 11, 32, 37, 145, 159, 220, 60, 135, 33]
```




```
datasets["test"][20]["answers"]
```

```
'{'text': 'Recent data suggest that there has been a 25% decrease in the incidence of pneumonia, from
0.29 episodes per child year in low-and middle-income countries in 2000, to 0.22 episodes per child yea
r in 2010. This is substantiated by a 58% decrease in pneumonia-associated disability-adjusted life yea
rs between 1990 and 2013, from 186 million to 78 million ', 'answer_start': 0}'
```

Here I saw the higer score usually gives small answers and less score is giving you little longer answers