```
!pip install google-cloud-vision
```

```
Collecting google-cloud-vision
  Downloading https://files.pythonhosted.org/packages/0d/7f/e10d602c2dc3f749f1b78377a
     |████████████████████████████████| 440kB 2.7MB/s
Requirement already satisfied: google-api-core[grpc]<2.0.0dev,>=1.14.0 in /usr/local/
Requirement already satisfied: pytz in /usr/local/lib/python3.6/dist-packages (from g
Requirement already satisfied: setuptools>=34.0.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in /usr/local/lib/python3.6
Requirement already satisfied: google-auth<2.0dev,>=0.4.0 in /usr/local/lib/python3.6
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in /usr/local/
Requirement already satisfied: protobuf>=3.4.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: grpcio<2.0dev,>=1.8.2; extra == "grpc" in /usr/local/l
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: cachetools<3.2,>=2.0.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: rsa<4.1,>=3.1.4 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.6/dist-package
Installing collected packages: google-cloud-vision
Successfully installed google-cloud-vision-1.0.0
WARNING: The following packages were previously imported in this runtime:
  [google]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
import pandas as pd
import numpy as np
import json
from google.colab import drive
from glob import glob
import errno
import time
import os
import re
import io
from enum import Enum
from google.cloud import vision
```

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="/content/drive/My Drive/Nearest Word/My_Firs
```

```
class FeatureType(Enum):
    PAGE = 1
    BLOCK = 2
    PARA = 3
    WORD = 4
```

```
    SYMBOL = 5


def ocr_google(filename):


    text = None
    # Read document content
    with io.open(filename, 'rb') as image_file:
        content = image_file.read()
    image = vision.types.Image(content=content)

    #GCP client
    client = vision.ImageAnnotatorClient()

    # Call Google OCR
    response = client.text_detection(image=image)

    #print(response)

    # Print detected text
    text = response.text_annotations
    print('Texts:{}'.format(text))
    return text[0].description


filein = "/content/drive/My Drive/Nearest Word/6919178.jpg"


def get_document_bounds(image_file, feature):
    """Returns document bounds given an image."""
    client = vision.ImageAnnotatorClient()

    bounds = []

    with io.open(image_file, 'rb') as image_file:
        content = image_file.read()

    image = vision.types.Image(content=content)

    response = client.document_text_detection(image=image)
    document = response.full_text_annotation

    # Collect specified feature bounds by enumerating all document features
    for page in document.pages:
        for block in page.blocks:
            for paragraph in block.paragraphs:
                for word in paragraph.words:
                    for symbol in word.symbols:
                        if (feature == FeatureType.SYMBOL):
                            bounds.append(symbol.bounding_box)

                    if (feature == FeatureType.WORD):
                        bounds.append(word.bounding_box)

                if (feature == FeatureType.PARA):
```

```
                    bounds.append(paragraph.bounding_box)

            if (feature == FeatureType.BLOCK):
                bounds.append(block.bounding_box)

    # The list `bounds` contains the coordinates of the bounding boxes.
    return bounds


page = get_document_bounds(filein, FeatureType.PAGE)
blocks = get_document_bounds(filein, FeatureType.BLOCK)
paras = get_document_bounds(filein, FeatureType.PARA)
words = get_document_bounds(filein, FeatureType.WORD)
symbols = get_document_bounds(filein, FeatureType.SYMBOL)


def detect_text(path):
    """Detects text in the file."""
    from google.cloud import vision
    import io
    client = vision.ImageAnnotatorClient()

    with io.open(path, 'rb') as image_file:
        content = image_file.read()

    image = vision.types.Image(content=content)

    response = client.text_detection(image=image)
    texts = response.text_annotations
    print('Texts:')
    df_columns = ['bottom', 'left', 'right', 'top','text']
    words_df = pd.DataFrame(columns = df_columns )
    for text in texts:
        #print('\n"{}"'.format(text.description))

        vertices = (['({},{})'.format(vertex.x, vertex.y)
                    for vertex in text.bounding_poly.vertices])
        vertices = ([(vertex.x, vertex.y)
                    for vertex in text.bounding_poly.vertices])

        #print('bounds: {}'.format(','.join(vertices)))
        #print(vertices)
        #print(text.bounding_poly.vertices)
        bottom = vertices[2][1]
        left = vertices[0][0]
        right = vertices[2][0]
        top  = vertices[0][1]
        text = text.description

        words_df = words_df.append(pd.Series([bottom,left,right,top,text], index = df_colu
        #print(words_df)

    if response.error.message:
        raise Exception(
            '{}\nFor more info on error messages, check: '
            'https://cloud.google.com/apis/design/errors'.format(
                response.error.message))
```

```
                response.error.message))
    return words_df

words_df = detect_text(filein)
```

```
Texts:
```

```
words_df.head(20)
```

	bottom	left	right	top	text
0	3789	124	3223	17	2 него\nPRINCE MOTORS\nJL NQ 88,Ni 34,JADUPUR,...
1	178	2600	2724	17	2
2	187	2728	3220	35	него
3	162	130	311	129	PRINCE
4	161	332	545	127	MOTORS
5	217	128	177	182	JL
6	215	196	265	181	NQ
7	215	282	417	182	88,Ni
8	214	438	1013	181	34,JADUPUR,GABGACHI
9	272	130	297	240	MALDA
10	271	314	481	238	MALDA
11	272	498	641	237	WEST
12	270	658	855	236	BENGAL
13	269	874	1035	235	732103
14	268	1054	1179	230	State
15	267	1200	1339	229	Code:
16	267	1366	1421	230	19
17	266	1440	1609	231	Contact
18	265	1624	1651	230	#
19	265	1666	1941	231	8158051859

```
words = words_df
```

```
c=0
```

```
for i,irow in words.iterrows():
    try:
        #print(1)
        right_new=words['left'][i+1]
        #print(right_new)
        #print(irow['left'])
```

```
            #print(irow['left'])
            gap=int(right_new)-int(irow['right'])
            #print(gap)
            words.loc[i,'gap']=int(gap)

            if gap>40 or gap<0:
                words.loc[i,'indexx']=c
                c=c+1


            else:
                words.loc[i,'indexx']=c


        except:
            words.loc[i,'gap']="NA"
            words.loc[i,'indexx']="NA"


    byAS =words.groupby('indexx')
    result = byAS.agg({
                      'text': ' '.join,
                      'left': np.min,
                      'right':np.max,
                      'top':np.min,
                      'bottom':np.max
    })


    result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 122 entries, 0.0 to NA
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    122 non-null    object
 1   left    122 non-null    int64 
 2   right   122 non-null    int64 
 3   top     122 non-null    int64 
 4   bottom  122 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 5.7+ KB
```

```
    #result.iloc[20:70,:]


    result.head()
```

indexx	text	left	right	top	bottom
0.0	2 него\nPRINCE MOTORS\nJL NQ 88,Ni 34,JADUPUR,...	124	3223	17	3789
1.0	2 него	2600	3220	17	187
2.0	PRINCE MOTORS	130	545	127	162
3.0	JL NQ 88,Ni 34,JADUPUR,GABGACHI	128	1013	181	217
4.0	MALDA MALDA WEST BENGAL 732103 State Code: 19 ...	130	1941	229	272

```
result = result.iloc[1:,:]
```

```
result.to_csv("/content/drive/My Drive/Nearest Word/6919178.csv")
```

```
def is_numeric(text):
  return bool(re.search(r'\d', text))
```

```
type(result.text)
```

```
pandas.core.series.Series
```

```
'UOM' in list(result.text)
```

```
False
```

```
def find_closest_at_right(entity_key,coordinate_df):
  #print(coordinate_df[coordinate_df.text==entity_key])
  if(entity_key in list(coordinate_df.text)):
    key_right = coordinate_df[coordinate_df.text==entity_key]['right']
    #print(key_right)
    key_right = int(key_right)
    key_top = coordinate_df[coordinate_df.text==entity_key]['top']
    key_top = int(key_top)

    value_indx = 0
    min_distance = 1000000

    for i,irow in coordinate_df.iterrows():
      horizontal_distance = irow['left'] - key_right
      vertical_distance = irow['top'] - key_top
      #print('distance:',horizontal_distance)
      #print('min_distance:',min_distance)
      if(horizontal_distance>0 and horizontal_distance<=min_distance and abs(vertical_dist
        #print(irow)
        min_distance = horizontal_distance
        value_index = i
        #print('inside if min_distance:',min_distance)
        #print(coordinate_df.loc[value_index,'text'])
        #break
```

```
    #print(value_index)
    return min_distance,coordinate_df.loc[value_index,'text'],is_numeric(coordinate_df.loc
```

```
def find_closest_at_bottom(entity_key,coordinate_df):
  if(entity_key in list(coordinate_df.text)):
    key_bottom = coordinate_df[coordinate_df.text==entity_key]['bottom']
    key_bottom = int(key_bottom)
    key_left = coordinate_df[coordinate_df.text==entity_key]['left']
    key_left = int(key_left)
    key_right = coordinate_df[coordinate_df.text==entity_key]['right']
    key_right = int(key_right)
    key_top = coordinate_df[coordinate_df.text==entity_key]['bottom']
    key_top = int(key_top)

    value_indx = 0
    min_distance = 1000000

    for i,irow in coordinate_df[coordinate_df['text']!=entity_key].iterrows():
      horizontal_distance = irow['left'] - key_left
      vertical_distance = irow['top']  - key_bottom
      #print('vertical_distance:',vertical_distance)
      #print('min_distance:',min_distance)
      #print(irow.text)
      #print(abs(horizontal_distance))
      #print(vertical_distance)
      if(vertical_distance>0 and vertical_distance<=min_distance and abs(horizontal_distan
        #print(irow.text)
        #print(horizontal_distance)
        #print(vertical_distance)
        min_distance = vertical_distance
        value_index = i
        #print('inside if min_distance:',min_distance)
        #print(coordinate_df.loc[value_index,'text'])
        #break
    #print(value_index)
    return min_distance,coordinate_df.loc[value_index,'text'],is_numeric(coordinate_df.loc


print(find_closest_at_right('Invoice #',result))
print(find_closest_at_bottom('Invoice #',result))
```

```
(71, '11962BL19S33', True)
(22, 'Date', False)
```

```
print(find_closest_at_right('Date',result))
print(find_closest_at_bottom('Date',result))
```

```
(175, '20/12/2019 16:39:16', True)
(72, 'UIN', False)
```

```
print(find_closest_at_right('Engine#',result))
print(find_closest_at_bottom('Engine#',result))
```

```
(293, 'Chassis #', False)
(47, 'JF33ADKGL03122', True)
```

```
print(find_closest_at_right('Chassis #',result))
print(find_closest_at_bottom('Chassis #',result))
```

```
(142, 'CGST %', False)
(57, 'MBLJFN051KGL02967', True)
```

```
print(find_closest_at_right('Grand Total',result))
print(find_closest_at_bottom('Grand Total',result))
```

```
(2606, '65,085.00', True)
(36, 'Rupees Sixty Five Thousand Eighty Five Only', False)
```

```
def find_nearest_correct_value(entity_key,coordinate_df,value_type):
  right_value_tuple = find_closest_at_right(entity_key,coordinate_df)
  bottom_value_tuple = find_closest_at_bottom(entity_key,coordinate_df)
  #if both right and bottom are numeric
  if(value_type is 'Alphanumeric'):
    if(right_value_tuple and right_value_tuple[2] and bottom_value_tuple and bottom_value_
      if(right_value_tuple[0]<bottom_value_tuple[0]):
        nearest_tuple = right_value_tuple
        return right_value_tuple[1]
      else:
        nearest_tuple = bottom_value_tuple
        return bottom_value_tuple[1]
    if(right_value_tuple and right_value_tuple[2]):
       return right_value_tuple[1]
    if(bottom_value_tuple and bottom_value_tuple[2]):
       return bottom_value_tuple[1]
  return ""
```

```
print(find_nearest_correct_value('Invoice #',result,'Alphanumeric'))
```

```
11962BL19S33
```

```
print(find_nearest_correct_value('Date',result,'Alphanumeric'))
```

```
20/12/2019 16:39:16
```

```
print(find_nearest_correct_value('Engine#',result,'Alphanumeric'))
```

```
JF33ADKGL03122
```

```
print(find_nearest_correct_value('Chassis #',result,'Alphanumeric'))
```

```
MBLJFN051KGL02967
```

```
print(find_nearest_correct_value('Grand Total',result,'Alphanumeric'))
```

```
65,085.00
```

```
print(find_nearest_correct_value('UIN No.',result,'Alphanumeric'))
```

```
694644042116
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.