```
from collections import Counter
import pandas as pd
from math import pi
from pdfminer.converter import TextConverter
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfpage import PDFPage
from spacy.matcher import PhraseMatcher, Matcher
from spacy.tokens import Doc, Span, Token
import spacy
from spacy.lemmatizer import Lemmatizer
from spacy.tokenizer import Tokenizer
from spacy.lang.en import English
from sklearn import decomposition
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
#from sklearn.feature_extraction import stop_words
from scipy import linalg
import numpy as np
import operator
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import seaborn as sb
import pandas as pd
import io
import os
sb.set_theme(style="whitegrid")
np.set_printoptions(precision=1)
import warnings; warnings.filterwarnings(action='once')

class Document:
    # Class attributes
    resource_manager = PDFResourceManager()
    file_handle = io.StringIO()
    converter = TextConverter(resource_manager, file_handle)
    page_interpreter = PDFPageInterpreter(resource_manager, converter)
    #nlp = spacy.load("en_core_web_sm")
    nlp = spacy.load("en_core_web_lg")
    tokenizer = Tokenizer(nlp.vocab)

    nlp.add_pipe(nlp.create_pipe('sentencizer'))
    os.chdir('/home/saul/Business')

    numberofTopics = 5
```

```
    svdTopics = []
    nmfTopics = []
    ldaTopics = []
    common_verbs = []
    common_nouns = []
    common_adjs = []

    Topics = {}
    weightsDict = {}

    def __init__(self, fileName):
        self.__convertToText(fileName)

    def __convertToText(self, fileName):
        list = []
        with open(fileName, 'rb') as fh:
            for page in PDFPage.get_pages(fh,
                                          caching=True,
                                          check_extractable=True):
                self.page_interpreter.process_page(page)

            text = self.file_handle.getvalue() # whole document in text
            list.append(text)

        self.converter.close()
        self.file_handle.close()
        self.__textAnalysis(text)

    def __textAnalysis(self, text):

        # Add law jargon and terms to stop words
        customize_stop_words = ['a.', 'b.', 'c.', 'i.', 'ii', 'iii',
        'the', 'to', " \x0c", ' ', 'Mr.', 'Dr.', 'v', 'of', 'case', 'section', 'defence',
        'trial', 'evidence', 'law', 'court', 'Court', 'criminal', 'Act', 'Article', 'UK','
    extradition', 'offence', 'information',
        '“', '-v-', 'A.', 'B.', '(', ')', 'wlr', 'wikileaks'
        ]
        for w in customize_stop_words:
            self.nlp.vocab[w].is_stop = True

        customize_non_punct = [
        'Ms.'
        ]
        for w in customize_non_punct:
```

```
            self.nlp.vocab[w].is_punct = False
        doc = self.nlp(text)

        #remove stop wods
        cleanDoc = [t.text for t in doc if t.is_stop != True and t.whitespace_ != True
and t.text.isspace() != True and t.is_punct != True
        and t.pos != "-PRON-"]

        # convert List to String not include strings less then 3
        listToStr = ' '.join([str(elem) for elem in cleanDoc if len(elem) > 2])
        cleanDoc = self.nlp(listToStr)
        self.__tokenizeDoco(cleanDoc)
        self.__svdDecomp(cleanDoc)
        self.__NMFDecomp(cleanDoc)
        self.__LDADecomp(cleanDoc)
        self.__topicAnalysis()
        self.__plotTopics()

        nouns = [t.lemma_ for t in cleanDoc if t.pos_ == "NOUN"]
        verbs = [t.lemma_ for t in cleanDoc if t.pos_ =="VERB"]
        adjectives = [t.lemma_ for t in cleanDoc if t.pos_ == "ADJ"]
        others = [t.lemma_ for t in cleanDoc if t.pos_ != "VERB" and t.pos_ != "NOUN
" and t.pos_ != "ADJ" and t.pos_ != "NUM"
        and t.pos != "-PRON-"]
        self.__verbAnalysis(verbs)
        self.__nounAnalysis(nouns)
        self.__adjectiveAnalysis(adjectives)

    def __tokenizeDoco(self, doc):

        sents_list = []
        for sent in doc.sents:
            sents_list.append(sent.text)

        #tfidf_vector = TfidfVectorizer()
        tfidf_vector = TfidfVectorizer(smooth_idf=False, sublinear_tf=False, norm=
None, analyzer='word')

        model = tfidf_vector.fit(sents_list)
        transformed_model = model.transform(sents_list) #Transform documents to
document-term matrix.

        self.weightsDict = dict(zip(model.get_feature_names(), tfidf_vector.idf_))
```

```
        #Weight of words per document
        max_val = transformed_model.max(axis=0).toarray().ravel()
        sort_by_tfidf = max_val.argsort()
        feature_names = np.array(tfidf_vector.get_feature_names())


    def __svdDecomp(self, doc):
        sents_list = []
        bow_vector = CountVectorizer(min_df =0.001, max_df=0.95, stop_words='
english') # Convert a collection of text documents to a matrix of token counts
        for sent in doc.sents:
            sents_list.append(sent.text)
        vectors = bow_vector.fit_transform(sents_list).todense()
        vocab = np.array(bow_vector.get_feature_names())
        U, s, Vh = linalg.svd(vectors, full_matrices=False)
        topics = self.__get_topics(Vh[:self.numberofTopics], vocab)
        self.__tokenizeTopics(topics, "SVD")


    def __NMFDecomp(self, doc):
        sents_list = []
        bow_vector = CountVectorizer(min_df =0.001, max_df=0.95, stop_words='
english') # Convert a collection of text documents to a matrix of token counts
        for sent in doc.sents:
            sents_list.append(sent.text)
        vectors = bow_vector.fit_transform(sents_list).todense()
        vocab = np.array(bow_vector.get_feature_names())
        m,n=vectors.shape
        topicModel = decomposition.NMF(n_components= self.numberofTopics,
random_state=1)
        fittedModel = topicModel.fit_transform(vectors)
        topicModelComps = topicModel.components_
        topics = self.__get_topics(topicModelComps, vocab)
        self.__tokenizeTopics(topics, "NMF")

    def __LDADecomp(self, doc):

        sents_list = []
        bow_vector = CountVectorizer(min_df =0.001, max_df=0.95, stop_words='
english') # Convert a collection of text documents to a matrix of token counts
        for sent in doc.sents:
            sents_list.append(sent.text)

        vectors = bow_vector.fit_transform(sents_list).todense()
```

```
        vocab = np.array(bow_vector.get_feature_names())
        m,n=vectors.shape

        topicModel = decomposition.LatentDirichletAllocation(n_components=self.
numberofTopics, max_iter=10, learning_method='online',verbose=True)
        lda_fit = topicModel.fit_transform(vectors) #Learn the vocabulary dictionary
and return document-term matrix
        topicModelComps = topicModel.components_
        topics = self.__get_topics(topicModelComps, vocab)
        self.__tokenizeTopics(topics, "LDA")

    def __tokenizeTopics(self, topics, modeltype):

        # convert List to String not include strings less than 3
        listToStr = ' '.join([str(elem) for elem in topics if len(elem) > 2])
        doc = self.nlp(listToStr)

        for sent in doc:
            if modeltype == "LDA":
                self.ldaTopics.append(sent.text)
            elif modeltype == "NMF":
                self.nmfTopics.append(sent.text)
            elif modeltype == "SVD":
                self.svdTopics.append(sent.text)

    def __topicAnalysis(self):
        self.Topics = set(self.ldaTopics) & set(self.nmfTopics) & set(self.svdTopics)

    def __get_topics(self, vector, vocab):
        num_top_words=10
        top_words = lambda t: [vocab[i] for i in np.argsort(t)[:-num_top_words-1:-1]]
        topic_words = ([top_words(t) for t in vector])
        return [' '.join(t) for t in topic_words]

        # Get Bag of Words (BoW) of top 10 words
    def __verbAnalysis(self, verbs):
        verb_freq = Counter(verbs)
        self.common_verbs = verb_freq.most_common(10)
        self.__radar(self.common_verbs, 'Top 10 Frequent Actions', 'Actions')
        self.__bar(self.common_verbs, 'Top 10 Frequent Actions', 'Actions')

    def __nounAnalysis(self, nouns):
        noun_freq = Counter(nouns)
```

```
        self.common_nouns = noun_freq.most_common(10)
        self.__radar(self.common_nouns, 'Top 10 Frequent Subjects', 'Subjects')
        self.__bar(self.common_nouns, 'Top 10 Frequent Subjects', 'Subjects')

    def __adjectiveAnalysis(self, adjectives):
        adj_freq = Counter(adjectives)
        self.common_adjs = adj_freq.most_common(10)
        self.__radar(self.common_adjs, 'Top 10 Frequent Referrals', 'Referrals')
        self.__bar(self.common_adjs, 'Top 10 Frequent Referrals', 'Referrals' )

    def __otherAnalysis(self, others):
        oth_freq = Counter(others)
        common_oths = oth_freq.most_common(10)

    def __plotTopics(self):

        mainTopics = {}

        for key in self.Topics:

            if key in self.weightsDict:
                mainTopics[key] = self.weightsDict[key]

        tt = dict(sorted(mainTopics.items(), key=lambda item: item[1])) # sort topics
with their idf

        x, y = zip(*tt.items()) # unpack a list of pairs into two tuples

        df = pd.DataFrame({"Topics":x,
                           "Inverse Term Frequency Ranks":y})

        graph = sb.PairGrid(df, x_vars= ["Inverse Term Frequency Ranks"] , y_vars=["
Topics"],
                            height=10, aspect= 0.8)

        graph.map(sb.stripplot, size=12, orient="h", jitter=False,
            palette="flare_r", linewidth=1, edgecolor="w")

        # Annotate
        plt.annotate('Mercedes Models', xy=(0.0, 11.0), xytext=(1.0, 11), xycoords='
data',
            fontsize=15, ha='center', va='center',
            bbox=dict(boxstyle='square', fc='firebrick'),
            arrowprops=dict(arrowstyle='-[, widthB=2.0, lengthB=1.5', lw=2.0, color='
```

```
steelblue'), color='white')

        # Add Patches
        p1 = patches.Rectangle((3.7, 1.5), width=0.55, height=5, alpha=.2, facecolor='
blue')
        p2 = patches.Rectangle((4.6, 10.5), width=.3, height=2, alpha=.2, facecolor='
blue')

        plt.gca().add_patch(p1)
        plt.gca().add_patch(p2)
        plt.title('Topics of Court Decision', weight='bold', fontdict={'size':11})
        plt.subplots_adjust(left = 0.16, bottom=0.16, top=0.9)
        plt.show()

    def __radar(self, words, title, subject):

        fig, axes = plt.subplots(figsize=(9, 9))
        fig.subplots_adjust(wspace=0.25, hspace=0.20, top=0.85, bottom=0.05)

        graphdata = {}
        graphdata['group'] = ['A']

        for _ in range(len(words)):
            graphdata[words[_][0]]= [words[_][1]]

        dataframe = pd.DataFrame(graphdata)

        categories=list(dataframe)[1:]
        N = len(categories)

        values=dataframe.loc[0].drop('group').values.flatten().tolist()
        values += values[:1]

        angles = [n / float(N) * 2 * pi for n in range(N)]
        angles += angles[:1]

        ax = plt.subplot(111, polar=True)
        plt.xticks(angles[:-1], categories, color='grey', size=10)
        ax.set_rlabel_position(0)
        plt.yticks([20, 60,  100, 140, 180],
        ["20", "60", "100", "140", "180"], color="grey", size=8)
        plt.ylim(0,max(values))
        ax.plot(angles, values, linewidth=1, linestyle='solid')
        ax.fill(angles, values, 'b', alpha=0.1)
```

```
        ax.set_title(title, weight='bold', size='medium', position=(0.5, 1.1),
                horizontalalignment='center', verticalalignment='center')
        plt.savefig(str(subject) + '_radar.png')
        plt.show()

    def __bar(self, words, title, subject):

        figs, ax = plt.subplots(figsize=(11, 7))
        graphdata = {}

        for _ in range(len(words)):

            graphdata[words[_][0]]= [words[_][1]]

        dataframe = pd.DataFrame(graphdata)
        categories=list(dataframe)[0:]
        values=dataframe.loc[0].values.flatten().tolist()

        y_pos = np.arange(len(categories))

        plt.barh(categories, values )

        # Show top values
        ax.invert_yaxis()
        ax.set_title(title, weight='bold', size='medium', position=(0.5, 1.1),
                horizontalalignment='center', verticalalignment='center')
        ax.set_ylabel( subject, fontweight ='bold')
        ax.set_xlabel("Term Frequency", fontweight ='bold')
        # Add Text watermark
        # Add padding between axes and labels
        ax.xaxis.set_tick_params(pad = 5)
        ax.yaxis.set_tick_params(pad = 10)
        # Add annotation to bars
        for i in ax.patches:
            plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10, fontweight ='bold',
             color ='grey')

        figs.text(0.9, 0.15, 'Seyhan AI', fontsize = 12,
         color ='grey', ha ='right', va ='bottom',
         alpha = 0.7)
        plt.subplots_adjust(bottom=0.2, top=0.9)
        plt.savefig(str(subject) + '.png')
```

```
        plt.show()


if __name__ == '__main__':

    courtdoco = Document("usaassangejudgement.pdf")
```