```
import streamlit as st
import pandas as pd
import nltk
nltk.download('all')
import plotly.express as px
import plotly.graph_objects as go



APP_NAME = "Sentiment Analysis!"

st.set_page_config(
    page_title=APP_NAME,
    page_icon=":bar_chart:",
    layout="wide",
    initial_sidebar_state="expanded",
)

#


st.sidebar.title(APP_NAME)

st.header("Sentiment Analysis for Earnings Call Transcript!")

with open(r'C:\Stock Market Prediction\tesla1.txt', encoding="utf8") as
file:
    data = file.read()

transcript_dict = dict()
for value in data.split('\n\n'):
    speakers_data = value.split('\n')
    transcript_dict[speakers_data[0]] = speakers_data[1]

key = st.selectbox("Select Speaker", transcript_dict.keys())

transcript = transcript_dict[key]
# print(transcript)

show_text = st.checkbox("Show Transcript")
if show_text:
    st.subheader('Transcript')
    st.markdown(transcript)



# split the transcript into sentences
sentences = [' '.join(sent.split()).strip() for sent in
transcript.replace('\n', '').split('. ')]

# convert to dataframe
df = pd.DataFrame(sentences, columns=['content'])
import string
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```
# return the wordnet object value corresponding to the POS tag
def get_wordnet_pos(pos_tag):
    if pos_tag.startswith('J'):
        return wordnet.ADJ
    elif pos_tag.startswith('V'):
        return wordnet.VERB
    elif pos_tag.startswith('N'):
        return wordnet.NOUN
    elif pos_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN


# return the cleaned text
def clean_text(text, digits=False, stop_words=False, lemmatize=False,
only_noun=False):
    # lower text
    text = str(text).lower()

    # tokenize text and remove puncutation
    text = [word.strip(string.punctuation) for word in text.split(" ")]

    # remove words that contain numbers
    if digits:
        text = [word for word in text if not any(c.isdigit() for c in
word)]

    # remove stop words
    if stop_words:
        stop = stopwords.words('english')
        text = [x for x in text if x not in stop]

    # remove empty tokens
    text = [t for t in text if len(t) > 0]

    # pos tag text
    if lemmatize:
        pos_tags = pos_tag(text)
        # lemmatize text
        text = [WordNetLemmatizer().lemmatize(t[0],
get_wordnet_pos(t[1])) for t in pos_tags]

    if only_noun:
        # select only nouns
        is_noun = lambda pos: pos[:2] == 'NN'
        text = [word for (word, pos) in pos_tag(text) if is_noun(pos)]

    # remove words with only one letter
    text = [t for t in text if len(t) > 1]

    # join all
    text = " ".join(text)

    return (text)
# clean text data
```

```
df['content_clean'] = df['content'].apply(lambda x: clean_text(x,
digits=True, stop_words=True, lemmatize=True))

# import NLTK library for importing the SentimentIntensityAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# create the instance of SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()

# get sentiment score for each category
df['sentiment']= df['content_clean'].apply(lambda x:
sid.polarity_scores(x))
df = pd.concat([df.drop(['sentiment'], axis=1),
df['sentiment'].apply(pd.Series)], axis=1)
df = df.rename(columns={'neu': 'neutral', 'neg': 'negative', 'pos':
'positive'})

# add sentiment based on max score
df['confidence'] = df[["negative", "neutral", "positive"]].max(axis=1)
df['sentiment'] = df[["negative", "neutral",
"positive"]].idxmax(axis=1)


# create data for plot
grouped = pd.DataFrame(df['sentiment'].value_counts()).reset_index()
grouped.columns = ['sentiment', 'count']

# Display percentage of positive, negative and neutral sentiments
fig = px.pie(grouped, values='count', names='sentiment',
title='Sentiments')
fig.show()
# calculate sentiment ratio
sentiment_ratio =
df['sentiment'].value_counts(normalize=True).to_dict()
for key in ['negative', 'neutral', 'positive']:
    if key not in sentiment_ratio:
        sentiment_ratio[key] = 0.0

## Display sentiment score
sentiment_score = (sentiment_ratio['neutral'] +
sentiment_ratio['positive']) - sentiment_ratio['negative']

fig = go.Figure(go.Indicator(
    mode = "number+delta",
    value = sentiment_score,
    delta = {"reference": 0.5},
    title = {"text": "Sentiment Score"},))

fig.show()
## Display negative sentence locations
fig = px.scatter(df, y='sentiment', color='sentiment',
size='confidence', hover_data=['content'],
color_discrete_map={"negative":"firebrick","neutral":"navajowhite","pos
itive":"darkgreen"})


fig.update_layout(
    width=800,
```

```
    height=300)
fig.show()
```