This executable notebook will guide you through Pset_2 - The Relationship between Surprisal and RTs:

Reminder, a few Colab-specific things to note about execution before we get started:

- Google offers free compute (including GPU compute!) on this notebook, but *only for a limited time*. Your session will be automatically closed after 12 hours. That means you'll want to finish within 12 hours of starting, or make sure to save your intermediate work (see the next bullet).
- You can save and write files from this notebook, but they are *not guaranteed to persist*. For this reason, we'll mount a Google Drive account and write to that Drive when any files need to be kept permanently (e.g. model checkpoints, surprisal data, etc.).
- You should keep this tab open until you're completely finished with the notebook. If you close the tab, your session will be marked as "Idle" and may be terminated.

Getting started

First, make a copy of this notebook so you can make your own changes. Click *File -> Save a copy in Drive*.

What you need to do

Read through this notebook and execute each cell in sequence, making modifications and adding code where necessary. You should execute all of the code as instructed, and make sure to write code or textual responses wherever the text **TODO** shows up in text and code cells.

When you're finished, download the notebook as a PDF file by running the script in the last cell, or alternatively download it as an .ipynb file and locally convert it to PDF.

Load ngram surprisals

Let's fetch the `ngram` surprisal file:

In [46]:
```
import pandas as pd
surprisals = pd.read_csv('https://gist.githubusercontent.com/omershubi/f19f77f5157f7ba7ea1adf72a72847da/raw/c
surprisals
```

Out[46]:

	sentence_id	token_id	token	surprisal
0	1	1	In	4.57937
1	1	2	<unk>	7.45049
2	1	3	County	12.65410
3	1	4	<unk>	6.11317
4	1	5	near	12.22380
...	...	...	...	...
7693	464	17	a	3.23962
7694	464	18	leader	12.81650
7695	464	19	and	5.90348
7696	464	20	<unk>	4.62292
7697	464	21	</s>	11.10650

7698 rows × 4 columns

Load RT data

Let's fetch also the Brown_RTs dataset and see how it looks like

In [47]:
```
sprt = pd.read_csv('https://gist.githubusercontent.com/omershubi/01b55eab89b81dc882055e0d27d61016/raw/046dbb7
sprt
```

Out[47]:

	word	code	subject	text_id	text_pos	word_in_exp	time
50709	In	17000	s014	0	0	1394	501.59
71402	In	17000	s019	0	0	1252	291.95
88505	In	17000	s023	0	0	883	357.57
113707	In	17000	s029	0	0	2171	293.10
65569	In	17000	s018	0	0	0	541.18
...	...	...	...	...	...	...	...
30985	captain.	35763	s008	12	763	4009	246.55
124545	captain.	35763	s032	12	763	2502	206.54
50708	captain.	35763	s014	12	763	1393	374.83
107720	captain.	35763	s028	12	763	763	690.71
1655	captain.	35763	s001	12	763	1654	520.17

136907 rows × 7 columns

Harmonize N-gram surprisal and RT data

We have the model-derived surprisal values. To align it with human reading times, complete the following cell. This will create for us a data frame containing both metrics in sync.

In `surprisals` each row represents a word. In `sprt` each row represents a word that was displayed in a trial. Therefore, in `sprt` there are multiple row for each word - one for each subject.

Note that the words are ordered the same in both files (i.e. they both start with 'In', then 'Ireland's'/'<unk>, then 'County', and so on. However, there are differences, such as a special token for end of sentence which appears only in `surprisals` , among others.

See the PDF instructions for more details.

To preprocess the data we removed unknown tokens and split lines that contain more then one word.

In [48]:
```
import re
import pandas as pd

def sanitize_phrase(text):
    return text if text == '<unk>' else re.sub(r'[^\w\s]', '', text)

def flag_and_explode_by_space(df: pd.DataFrame):
    df['has_space'] = df['word'].str.contains(' ')
    df = df.assign(word=df['word'].str.split()).explode('word').reset_index(drop=True)
    return df

def join_dataframes(df1: pd.DataFrame, df2: pd.DataFrame):
    return pd.merge(df1.reset_index(drop=True), df2.reset_index(drop=True), left_index=True, right_index=True

def process_and_join(surprisal_df: pd.DataFrame, reaction_time_df: pd.DataFrame):
    rt_processed = reaction_time_df.drop(columns=['subject', 'word_in_exp', 'time']).drop_duplicates()
    rt_processed = flag_and_explode_by_space(rt_processed)
    rt_processed['word'] = rt_processed['word'].apply(sanitize_phrase)
    surprisal_df['token'] = surprisal_df['token'].apply(sanitize_phrase)
    merged_df = join_dataframes(surprisal_df, rt_processed)
    cleaned_df = merged_df[~merged_df['has_space'] & (merged_df['token'] != '<unk>')]
    return cleaned_df
```

In [49]:
```
def harmonize(rt_data: pd.DataFrame, surprs_data: pd.DataFrame) -> pd.DataFrame:
    filtered_surprisals = surprs_data[surprs_data.token != '</s>'].copy()
    rt_copy = rt_data.copy()
    surprisal_rt_merged = process_and_join(filtered_surprisals, rt_copy)
    time_by_code = rt_copy.drop(columns=['word', 'subject', 'text_id', 'text_pos', 'word_in_exp'])
    average_rt_by_code = time_by_code.groupby('code')['time'].mean()

    harmonized_data = surprisal_rt_merged.merge(average_rt_by_code, on='code')[['word', 'surprisal', 'time']]
    harmonized_data = harmonized_data.rename(columns={'time': 'mean_rt'})

    return harmonized_data

harmonized_df = harmonize(sprt, surprisals)
harmonized_df
```

Out[49]:

	word	surprisal	mean_rt
0	In	4.57937	380.275294
1	County	12.65410	296.042941
2	near	12.22380	403.553529
3	the	1.98095	306.075882
4	River	15.70900	289.048235
...	...	...	...
5456	failed	8.25341	292.772500
5457	as	9.42416	284.470833
5458	a	3.23962	282.622083
5459	leader	12.81650	279.445417
5460	and	5.90348	299.705000

5461 rows × 3 columns

When you are done with this step, save the result using the following code

In [50]:
```
harmonized_df.to_csv("harmonized_ngram.csv")
```

Great, now you're ready to start doing analysis on this output data!

Analyses

Now that we've obtained our harmonized surprisal-vs-RT files, let's perform some analysis on the data.

1. Univariate linear regression

Here is an overview of the analysis we want you to run.

- For each of `metric` in `{surprisal, raw_probability}` :
  - Fit a linear regression model to predict RTs from the metric. You should report the

coefficient for the metric term (slope) and a corresponding t-score and p-value (to determine whether it is significantly different from 0), as well as an R^2-score (the coefficient of determination) of the model.; * Draw metric-RT scatterplot with best-fit line, **without** binning RT values; and * Draw metric-RT scatterplot with best-fit line, **with** binning RT values.

`Metric = Surprisal`

Fit a linear regression

In [51]:
```
import numpy as np
import statsmodels.api as sm
import pandas as pd

data = pd.read_csv("harmonized_ngram.csv")

X = data['surprisal']
y = data['mean_rt']
X = sm.add_constant(X)
lin_model = sm.OLS(y, X).fit()
print(lin_model.summary())
```

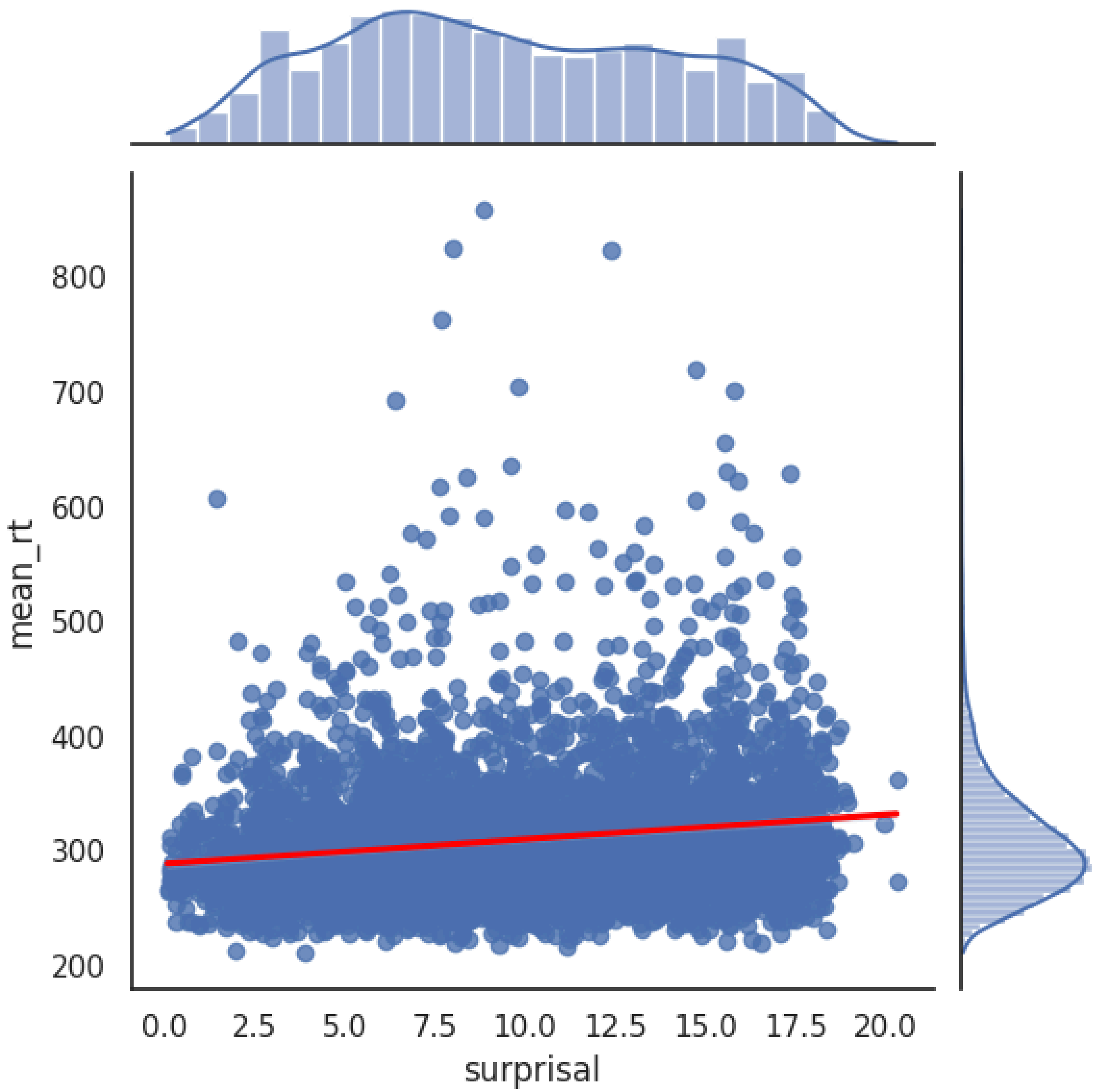
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                mean_rt   R-squared:                       0.032
Model:                            OLS   Adj. R-squared:                  0.032
Method:                 Least Squares   F-statistic:                     182.9
Date:                Sun, 04 May 2025   Prob (F-statistic):           5.10e-41
Time:                        14:23:39   Log-Likelihood:                -29512.
No. Observations:                5461   AIC:                         5.903e+04
Df Residuals:                    5459   BIC:                         5.904e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        289.1501      1.662    174.002      0.000     285.892     292.408
surprisal      2.1422      0.158     13.524      0.000       1.832       2.453
==============================================================================
Omnibus:                     2969.351   Durbin-Watson:                   1.282
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            36969.135
Skew:                           2.334   Prob(JB):                         0.00
Kurtosis:                      14.861   Cond. No.                         24.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The function `.summary()` outputs a variety of metrices and statistical tests. Here we are intrested in model's parameters (the coefficients), their t score, and the corresponding p-values, as well as in the overall R^2 - score of the model.

Now let's create a scatterplot of our data accompanied by the best-fit line

Without Binning:

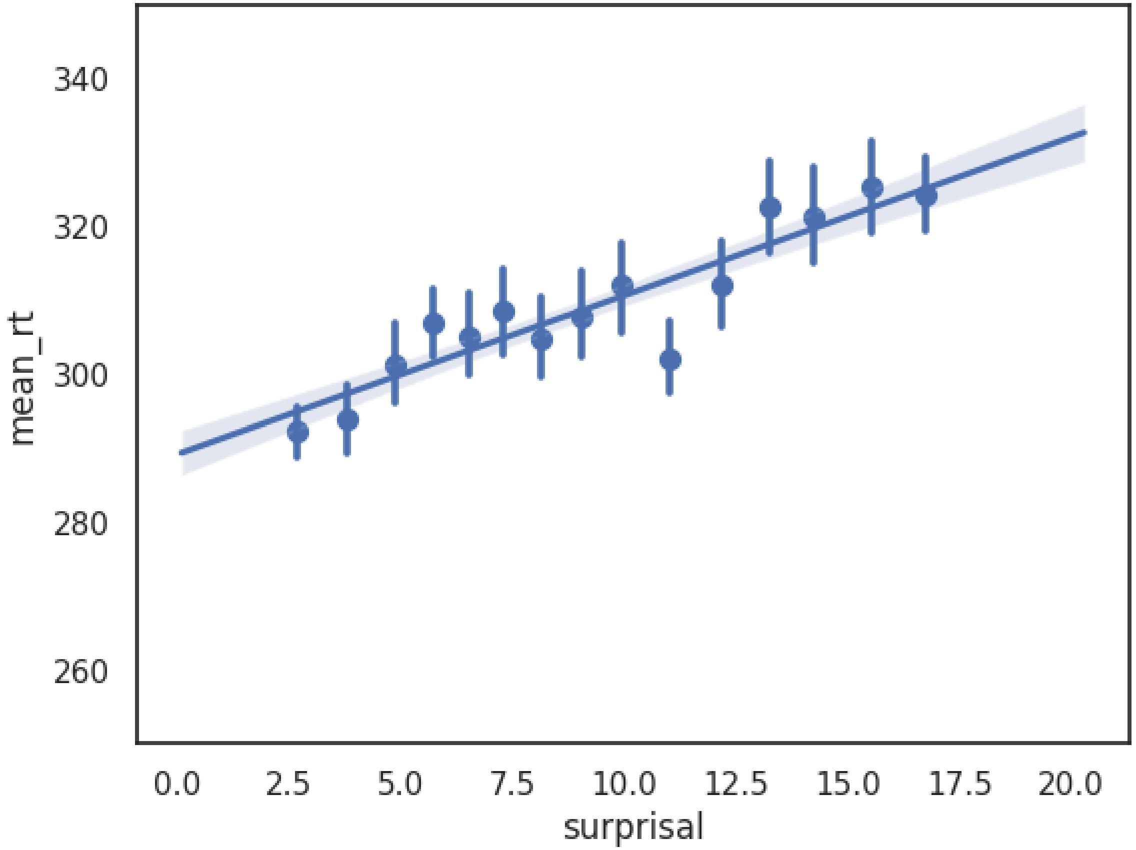
In [52]:
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(style="white", color_codes=True)

g = sns.jointplot(x="surprisal", y="mean_rt", data=data, kind='reg')
# We're going to make the regression line red so it's easier to see
regline = g.ax_joint.get_lines()[0]
regline.set_color('red')
```



With Binning:

In [53]:
```
g = sns.regplot(x="surprisal", y="mean_rt", data=data, x_bins=15)
g.set_ylim([250, 350])
```

Out[53]: (250.0, 350.0)



Metric = Raw_probability

After running the code cells above, your next task is to reproduce this analysis for `metric=raw_probability` .

Note that you can transform the surprisal values in the data frames by simply applying standard math and `numpy` operators. For example, this code takes each surprisal value to the power of 3 and adds 0.1:

In [54]:
```
#np.power(ngram.surprisal, 3) + 0.1
```

In [55]:
```
log_surprisal = data['surprisal']
exp_values = np.exp(log_surprisal)
inverse_probs = 1 / exp_values
data['estimated_prob'] = inverse_probs


X = data['estimated_prob']
y = data['mean_rt']
X = sm.add_constant(X)
lin_model = sm.OLS(y, X).fit()
print(lin_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                mean_rt   R-squared:                       0.007
Model:                            OLS   Adj. R-squared:                  0.006
Method:                 Least Squares   F-statistic:                     36.16
Date:                Sun, 04 May 2025   Prob (F-statistic):           1.94e-09
Time:                        14:23:41   Log-Likelihood:                -29584.
No. Observations:                5461   AIC:                         5.917e+04
Df Residuals:                    5459   BIC:                         5.919e+04
Df Model:                           1
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const            310.4404      0.760    408.638      0.000     308.951     311.930
estimated_prob   -61.9024     10.295     -6.013      0.000     -82.084     -41.721
==============================================================================
Omnibus:                     2953.317   Durbin-Watson:                   1.269
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            35057.722
Skew:                           2.336   Prob(JB):                         0.00
Kurtosis:                      14.500   Cond. No.                         14.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Interpret the results

- Does the univariate analysis support the hypothesis of a linear relationship between word surprisal and word reading time?
- Is that hypothesis better or worse than an alternative hypothesis of a linear relationship between raw word probability and word reading time?
- Are there other alternative hypotheses that might be even more compelling given the data?

The univariate anaylsis supports the hypothesis of a linear relationship between suprisal and word reading time, this is because we get a very small p value for the linear regression. A small p value means that we have a statistically significant relationship between the two variables.

The R squared in the suprisal model is larger then the R squared in the raw probability model. This means that there is a stronger linear relationship in between suprisal and reading time then between raw word probabilties and reading time

Another hypothesis can be that there is a non linear relationship between suprisal and word reading time this can be checked using different types of non linear regression

2. Multiple regression analysis : Adding control variables

In this stage we want to add two control variables to our linear model and reexamine the effect of surprisal *above and beyond* these variables. The two variables are **word-length** and **word log-frequency**.

First, you should write a code that creates those variables.

Word-length:

In [56]:
```
data['word_length'] = data['word'].astype(str).apply(len)

# Save the updated DataFrame back to the CSV
data.to_csv("harmonized_ngram.csv", index=False)
```

Word log-frequency:

For each word w_i in our `harmonized_ngram.csv` dataset, we want to obtain the $log(frequency(w_i))$ of w_i using a different, large corpus of text. You will first download the *tokenized* version of the **PTB** dataset (no other preprocessing stages are needed) and then write a code for

In [57]:
```
# Downloads ptb_tok_train.txt
!wget -qO ptb_tok_train.txt https://gist.githubusercontent.com/omershubi/cdd4231472d6188f03ab21e2b2729fee/raw
!head ptb_tok_train.txt
```

```
In an Oct. 19 review of `` The Misanthrope '' at Chicago 's Goodman Theatre -LRB- `` <unk> <unk> Take the S
tage in <unk> City , '' Leisure & Arts -RRB- , the role of Celimene , played by Kim <unk> , was mistakenly
attributed to Christina Haag .
Ms. Haag plays <unk> .
Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars in 1990 .
The luxury auto maker last year sold <unk> cars in the U.S.
Howard <unk> , president and chief executive officer , said he anticipates growth for the luxury auto maker
in Britain and Europe , and in Far Eastern markets .
<unk> INDUSTRIES Inc. increased its quarterly to 10 cents from seven cents a share .
The new rate will be payable Feb. 15 .
A record date has n't been set .
Bell , based in Los Angeles , makes and distributes electronic , computer and building products .
Investors are appealing to the Securities and Exchange Commission not to limit their access to information
about stock purchases and sales by corporate insiders .
```

In [58]:
```
from collections import Counter

with open("ptb_tok_train.txt", "r", encoding="utf-8") as f:
    ptb_text = f.read()

ptb_tokens = ptb_text.strip().split()

word_counts = Counter(word.lower() for word in ptb_tokens)

freq_df = pd.DataFrame(word_counts.items(), columns=["word", "frequency"])
freq_df["log_freq"] = np.log(freq_df["frequency"])

data = pd.read_csv("harmonized_ngram.csv")

data["word"] = data["word"].astype(str).str.lower().str.strip()

merged = pd.merge(data, freq_df[["word", "log_freq"]], on="word", how="left")

merged["log_freq"] = merged["log_freq"].fillna(np.log(1))

merged.to_csv("harmonized_ngram.csv", index=False)
```

Multiple regression analysis:

Based on the code above (section 1: univariate linear regression), write a new code for multiple regresion analysis.

In [59]:
```
data = pd.read_csv("harmonized_ngram.csv")
X = data[['surprisal', 'word_length', 'log_freq']]
X = sm.add_constant(X)
y = data['mean_rt']
multi_model = sm.OLS(y, X).fit()
print(multi_model.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                mean_rt   R-squared:                       0.050
Model:                            OLS   Adj. R-squared:                  0.049
Method:                 Least Squares   F-statistic:                     95.12
Date:                Sun, 04 May 2025   Prob (F-statistic):           5.27e-60
Time:                        14:23:41   Log-Likelihood:                -29463.
No. Observations:                5461   AIC:                         5.893e+04
Df Residuals:                    5457   BIC:                         5.896e+04
Df Model:                           3                                         
Covariance Type:            nonrobust                                         
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const         244.9572      6.668     36.737      0.000     231.886     258.029
surprisal       2.5528      0.300      8.498      0.000       1.964       3.142
word_length     4.6725      0.484      9.656      0.000       3.724       5.621
log_freq        3.2266      0.527      6.126      0.000       2.194       4.259
==============================================================================
Omnibus:                     3019.795   Durbin-Watson:                   1.291
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            39768.583
Skew:                           2.367   Prob(JB):                         0.00
Kurtosis:                      15.344   Cond. No.                         117.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

###Interpret the results

- How does the surprisal coefficient of this model compare to the surprisal coefficient in the univariate model?
- Does your conclusion regarding the effect of suprisal on RTs from the univariate analysis still hold?

The suprisal coefficient of this model is 2.55 while in the previous model it is 2.14. In this model we still get a strong linear relationship between suprisal and reading times.

Export to PDF

Run the following cell to download the notebook as a nicely formatted pdf file.

In [60]:
```
# Add to a new cell at the end of the notebook and run the follow code,
# which will save the notebook as pdf in your google drive (allow the permissions) and download it automatica

!wget -nc https://raw.githubusercontent.com/lacclab/096222-colab-pdf/master/colab_pdf.py

from colab_pdf import colab_pdf

# If you saved the notebook in the default location in your Google Drive,
#  and didn't change the name of the file, the code should work as is. If not, adapt accordingly.
# E.g. in your case the file name may be "Copy of XXXX.ipynb"

colab_pdf(file_name='Pset_2_RT_and_surprisal.ipynb', notebookpath="drive/MyDrive/Colab Notebooks")
```

```
File ‘colab_pdf.py’ already there; not retrieving.
```

```
---------------------------------------------------------------------------
MessageError                              Traceback (most recent call last)
<ipython-input-60-650e0d1638d4> in <cell line: 0>()
     10 # E.g. in your case the file name may be "Copy of XXXX.ipynb"
     11
---> 12 colab_pdf(file_name='Pset_2_RT_and_surprisal.ipynb', notebookpath="drive/MyDrive/Colab Notebooks")

/content/colab_pdf.py in colab_pdf(file_name, notebookpath)
     16         from google.colab import drive
     17
---> 18         drive.mount(drive_mount_point)
     19
     20     # Check if the notebook exists in the Drive.

/usr/local/lib/python3.11/dist-packages/google/colab/drive.py in mount(mountpoint, force_remount, timeout_m
s, readonly)
     98 def mount(mountpoint, force_remount=False, timeout_ms=120000, readonly=False):
     99   """Mount your Google Drive at the specified mountpoint path."""
--> 100   return _mount(
    101       mountpoint,
    102       force_remount=force_remount,

/usr/local/lib/python3.11/dist-packages/google/colab/drive.py in _mount(mountpoint, force_remount, timeout_
ms, ephemeral, readonly)
    135   )
    136   if ephemeral:
--> 137     _message.blocking_request(
    138         'request_auth',
    139         request={'authType': 'dfs_ephemeral'},

/usr/local/lib/python3.11/dist-packages/google/colab/_message.py in blocking_request(request_type, request,
timeout_sec, parent)
    174       request_type, request, parent=parent, expect_reply=True
    175   )
--> 176   return read_reply_from_input(request_id, timeout_sec)

/usr/local/lib/python3.11/dist-packages/google/colab/_message.py in read_reply_from_input(message_id, timeo
ut_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
```

```
        return reply.get('data', None)


MessageError: Error: credential propagation was unsuccessful
```