Welcome to Colab!

(New) Try the Gemini API

- Generate a Gemini API key
- Talk to Gemini with the Speech-to-Text API
- Compare Gemini with ChatGPT
- More notebooks

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

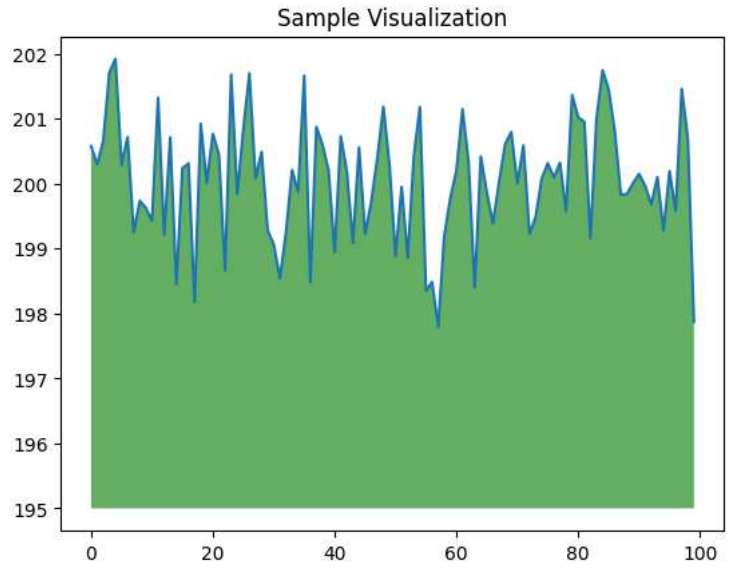
With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colaboratory
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

Featured examples

- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
import pandas as pd
```

```
df=pd.read_excel("/content/DoorDash.xlsx")
df
```

	customer_placed_order_datetime	placed_order_with_restaurant_datetime	driver_at_restaurant_datetime	delivered_to_consumer_datetir
0	2020-05-19 01:06:00	2020-05-19 01:07:00	2020-05-19 01:15:00	2020-05-19 01:27:(
1	2020-05-12 00:47:00	2020-05-12 00:49:00	NaT	2020-05-12 01:24:(
2	2020-05-04 00:13:00	2020-05-04 00:13:00	2020-05-04 00:24:00	2020-05-04 01:01:(
3	2020-05-11 01:53:00	2020-05-11 01:59:00	2020-05-11 02:31:00	2020-05-11 02:52:(
4	2020-05-18 04:44:00	2020-05-18 04:45:00	2020-05-18 04:51:00	2020-05-18 05:05:(
...	...	...	...	
495	2020-05-30 17:04:00	2020-05-30 18:12:00	NaT	2020-05-30 19:07:(
496	2020-05-26 04:09:00	2020-05-26 04:13:00	NaT	2020-05-26 05:03:(
497	2020-05-18 03:20:00	2020-05-18 03:26:00	2020-05-18 03:39:00	2020-05-18 03:51:(
498	2020-05-10 00:02:00	2020-05-10 00:06:00	2020-05-10 00:11:00	2020-05-10 00:43:(
499	2020-05-29 23:23:00	2020-05-30 01:36:00	2020-05-30 01:46:00	2020-05-30 02:12:(

500 rows × 14 columns

```
df.head(5)
```

	customer_placed_order_datetime	placed_order_with_restaurant_datetime	driver_at_res
0	2020-05-19 01:06:00	2020-05-19 01:07:00	2
1	2020-05-12 00:47:00	2020-05-12 00:49:00	
2	2020-05-04 00:13:00	2020-05-04 00:13:00	2
3	2020-05-11 01:53:00	2020-05-11 01:59:00	2
4	2020-05-18 04:44:00	2020-05-18 04:45:00	2

```
df.shape
```

```
(500, 14)
```

```
df.isnull().sum()
```

```
customer_placed_order_datetime             0
placed_order_with_restaurant_datetime      1
driver_at_restaurant_datetime            116
delivered_to_consumer_datetime             0
driver_id                                  0
restaurant_id                              0
consumer_id                                0
is_new                                     0
delivery_region                            2
is_asap                                    0
order_total                                0
discount_amount                            0
tip_amount                                 0
refunded_amount                            0
dtype: int64
```

```
df.drop(['placed_order_with_restaurant_datetime','driver_at_restaurant_datetime','delivery_region'], axis=1, inplace=True)
```

```
df.columns
```

```
Index(['customer_placed_order_datetime', 'delivered_to_consumer_datetime',
       'driver_id', 'restaurant_id', 'consumer_id', 'is_new', 'is_asap',
       'order_total', 'discount_amount', 'tip_amount', 'refunded_amount'],
      dtype='object')
```

```
df.isnull().sum()
```

```
customer_placed_order_datetime    0
delivered_to_consumer_datetime    0
driver_id                         0
restaurant_id                     0
consumer_id                       0
is_new                            0
is_asap                           0
order_total                       0
discount_amount                   0
tip_amount                        0
refunded_amount                   0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 14 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   customer_placed_order_datetime        500 non-null    datetime64[ns]
 1   placed_order_with_restaurant_datetime  499 non-null    datetime64[ns]
 2   driver_at_restaurant_datetime         384 non-null    datetime64[ns]
 3   delivered_to_consumer_datetime        500 non-null    datetime64[ns]
 4   driver_id                             500 non-null    int64
 5   restaurant_id                         500 non-null    int64
 6   consumer_id                           500 non-null    int64
 7   is_new                                500 non-null    bool
 8   delivery_region                       498 non-null    object
 9   is_asap                               500 non-null    bool
```

```
 10  order_total                 500 non-null    float64
 11  discount_amount             500 non-null    float64
 12  tip_amount                  500 non-null    float64
 13  refunded_amount             500 non-null    float64
dtypes: bool(2), datetime64[ns](4), float64(4), int64(3), object(1)
memory usage: 48.0+ KB
```

Q1:Which regions have the highest and lowest demand for delivery?

```
region_demand = df['delivery_region'].value_counts()
region_demand
```

```
Palo Alto        321
Mountain View    101
San Jose          76
Name: delivery_region, dtype: int64
```

from above , there is highest demand region is Palo Alto with 321 figure and lowest region is San Jose with figure value is 76

Q2:Which regions have the lowest demand for delivery?

```
lowest_demand_count=region_demand.min()
lowest_demand_count
```

```
76
```

Q3:Which regions have the highest demand for delivery?

```
highest_demand_count=region_demand.max()
highest_demand_count
```

```
321
```

```
#Is there a correlation between delivery region and order value?
#correlation = df['delivery_region'].corr(df[''])
#correlation
```

Q4: How many old and new consumers of restaurant?

```
df['is_new']
```

```
0      False
1      False
2      False
3      False
4      False
       ...
495     True
496    False
497    False
498    False
499    False
Name: is_new, Length: 500, dtype: bool
```

```
df['is_new'].value_counts()
```

```
False    403
True      97
Name: is_new, dtype: int64
```

from above 403 conusmers are old , however , 97 are new to restaurants.

Q5: How many consumers needs to delivery the food asap and how many dont want at asap condition?

```
df['is_asap'].value_counts()
```

```
True     388
False    112
Name: is_asap, dtype: int64
```

Q6: how many old and new consumers who wants asap delivery of food?

```
new = df.groupby('is_asap')['is_new'].value_counts()
new
```

```
is_asap  is_new
False    False      90
         True       22
True     False     313
         True       75
Name: is_new, dtype: int64
```

```
new = df.groupby('is_asap')['is_new'].count()
new
```

```
is_asap
False    112
True     388
Name: is_new, dtype: int64
```

from above , 112 consumers are old and wants asap delivery and 388 consumers are new and wants asap delivery of food

```
#df_sorted = df.sort_values(by='tip_amount', ascending=True)
#df_sorted['tip_amount'].reset_index()
```

Q7: which consumer giving maximum and minimum tip and how much? **bold text

```
new1 = df.groupby('consumer_id')['tip_amount'].agg('max').sort_values(ascending=False).reset_index()
new1
```

	consumer_id	tip_amount
0	12871	24.69
1	14519	21.28
2	3392	18.94
3	13447	18.15
4	91848	15.62
...	...	...
462	106751	0.00
463	78977	0.00
464	79146	0.00
465	93791	0.00
466	199532	0.00

467 rows × 2 columns

Q8: which restaurant giving highest discount to consumers and how much ? and which restro giving least discount to them?

```
new2 = df.groupby('restaurant_id')['discount_amount'].agg('max').sort_values(ascending=False).reset_index()
new2
```

	restaurant_id	discount_amount
0	5	84.74
1	29	50.00
2	259	18.45
3	240	11.00
4	195	6.00
...	...	...
146	94	0.00
147	91	0.00
148	90	0.00
149	88	0.00
150	356	0.00

151 rows × 2 columns

```
new2 = df.groupby('restaurant_id')['discount_amount'].agg('min').sort_values(ascending=False).reset_index()
new2
```

	restaurant_id	discount_amount
0	240	11.0
1	32	6.0
2	322	6.0
3	282	6.0
4	284	6.0
...	...	...
146	86	0.0
147	87	0.0
148	88	0.0
149	90	0.0
150	356	0.0

151 rows × 2 columns

Q9:What is the average tip amount?

```
avg_tip_amt_per_day=df['tip_amount'].mean()
avg_tip_amt_per_day
```

```
3.50714
```

```
# What is the total revenue generated over a specific period, including tips and discounts?
# What is the total number of orders placed over a specific period, and how does it vary over time?
# discount corr with order placed?
```

Q10: what is a total revenue generated by the restuarants, where including discounts and tip amount?

```
# Total Revenue = (Total Sales - Discounts) + Tips
# order_total is Total Sales here , dis & tips are also given...
df[['order_total','discount_amount']]
```

	order_total	discount_amount
0	24.71	6.0
1	14.65	6.0
2	29.33	0.0
3	58.16	0.0
4	24.16	0.0
...	...	...
495	93.22	0.0
496	37.75	0.0
497	23.89	0.0
498	40.69	0.0
499	59.67	0.0

500 rows × 2 columns

```
df = df.rename(columns={'order_total': 'total_sales'})
df
```

	customer_placed_order_datetime	delivered_to_consumer_datetime	driver_id	restaur
0	2020-05-19 01:06:00	2020-05-19 01:27:00	156	
1	2020-05-12 00:47:00	2020-05-12 01:24:00	345	
2	2020-05-04 00:13:00	2020-05-04 01:01:00	325	
3	2020-05-11 01:53:00	2020-05-11 02:52:00	249	
4	2020-05-18 04:44:00	2020-05-18 05:05:00	311	
...	...	...	...	
495	2020-05-30 17:04:00	2020-05-30 19:07:00	297	
496	2020-05-26 04:09:00	2020-05-26 05:03:00	144	
497	2020-05-18 03:20:00	2020-05-18 03:51:00	49	
498	2020-05-10 00:02:00	2020-05-10 00:43:00	248	
499	2020-05-29 23:23:00	2020-05-30 02:12:00	56	

500 rows × 11 columns

```
df1=df[['total_sales','discount_amount']]
```

```
df1['total_revenue']=df['total_sales']-df['discount_amount']+df['tip_amount']
df1
```

```
<ipython-input-253-e83c953cf10e>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  df1['total_revenue']=df['total_sales']-df['discount_amount']+df['tip_amount']
```

	total_sales	discount_amount	total_revenue
0	24.71	6.0	20.43
1	14.65	6.0	9.04
2	29.33	0.0	31.47
3	58.16	0.0	66.88
4	24.16	0.0	26.66
...	...	...	...
495	93.22	0.0	101.24
496	37.75	0.0	40.67
497	23.89	0.0	25.08
498	40.69	0.0	44.76
499	59.67	0.0	65.64

```
df.head(4)
```

	customer_placed_order_datetime	delivered_to_consumer_datetime	driver_id	restauran
0	2020-05-19 01:06:00	2020-05-19 01:27:00	156	
1	2020-05-12 00:47:00	2020-05-12 01:24:00	345	
2	2020-05-04 00:13:00	2020-05-04 01:01:00	325	
3	2020-05-11 01:53:00	2020-05-11 02:52:00	249	

Q11:What is the total number of orders placed over a specific period, and how does it vary over time?From 4 may 2020 to 29 may 2020? calculate the number of order in each day in specific period of time? **bold text

```
# specific period:
start_date = '2020-05-04'
end_date = '2020-05-11'
```

```
period_data = df[(df['customer_placed_order_datetime'] >= start_date) & (df['customer_placed_order_datetime'] <= end_date)]
orders_per_date = period_data.groupby(period_data['customer_placed_order_datetime'].dt.date).size()
orders_per_date
```

```
customer_placed_order_datetime
2020-05-04    20
2020-05-05    17
2020-05-06    15
2020-05-07    13
2020-05-08    16
2020-05-09    19
2020-05-10    16
dtype: int64
```

from above we have calculated the no of orders placed per day for one week which is starts from 4 of may 2020 to 11 of may 2020, there is initial orders placed with 20 size then comes to 17 then again down to 15,13, after fourth day it comes high with 19 orders and again down to 16 on last day of week , it seems so fluctuated ...

```
df['customer_placed_order_datetime'] = pd.to_datetime(df['customer_placed_order_datetime'])
```

Q12:What is the discount correlation with order placed?

```
df_sorted = df.sort_values(by='customer_placed_order_datetime', ascending=True)
df_sorted
```

	customer_placed_order_datetime	delivered_to_consumer_datetime	driver_id	restaur
233	2020-05-01 00:47:00	2020-05-01 01:47:00	166	
49	2020-05-01 01:06:00	2020-05-01 01:37:00	308	
429	2020-05-01 01:36:00	2020-05-01 03:18:00	249	
165	2020-05-01 02:06:00	2020-05-01 03:30:00	222	
458	2020-05-01 02:27:00	2020-05-01 03:19:00	317	
...	...	...	...	
21	2020-05-30 16:29:00	2020-05-30 18:41:00	434	
495	2020-05-30 17:04:00	2020-05-30 19:07:00	297	
484	2020-05-30 18:24:00	2020-05-30 19:10:00	434	
358	2020-05-30 20:27:00	2020-05-30 20:59:00	325	
194	2020-05-31 23:14:00	2020-05-01 00:53:00	324	

500 rows × 11 columns

```
df_sorted['customer_placed_order_datetime']
```

```
233   2020-05-01 00:47:00
49    2020-05-01 01:06:00
429   2020-05-01 01:36:00
165   2020-05-01 02:06:00
458   2020-05-01 02:27:00
            ...
21    2020-05-30 16:29:00
495   2020-05-30 17:04:00
484   2020-05-30 18:24:00
358   2020-05-30 20:27:00
194   2020-05-31 23:14:00
Name: customer_placed_order_datetime, Length: 500, dtype: datetime64[ns]
```

```
df['Month'] = df['customer_placed_order_datetime'].dt.month
df['Date'] = pd.to_datetime(df['customer_placed_order_datetime'])

df['customer_placed_order_datetime']
# 4may to 29 may 2020
```

```
0     2020-05-19 01:06:00
1     2020-05-12 00:47:00
2     2020-05-04 00:13:00
3     2020-05-11 01:53:00
4     2020-05-18 04:44:00
            ...
495   2020-05-30 17:04:00
496   2020-05-26 04:09:00
497   2020-05-18 03:20:00
498   2020-05-10 00:02:00
499   2020-05-29 23:23:00
Name: customer_placed_order_datetime, Length: 500, dtype: datetime64[ns]
```

```
df['customer_placed_order_datetime'] = pd.to_datetime(df['customer_placed_order_datetime'])
no_of_orders = df.groupby(df['customer_placed_order_datetime'].dt.date).size()
no_of_orders.reset_index()
```

	customer_placed_order_datetime	0
0	2020-05-01	11
1	2020-05-02	13
2	2020-05-03	15
3	2020-05-04	20
4	2020-05-05	17
5	2020-05-06	15
6	2020-05-07	13
7	2020-05-08	16
8	2020-05-09	19
9	2020-05-10	16
10	2020-05-11	12
11	2020-05-12	20
12	2020-05-13	12
13	2020-05-14	16
14	2020-05-15	15
15	2020-05-16	21
16	2020-05-17	14
17	2020-05-18	31
18	2020-05-19	20
19	2020-05-20	23
20	2020-05-21	10
21	2020-05-22	25
22	2020-05-23	20
23	2020-05-24	17
24	2020-05-25	17
25	2020-05-26	12
26	2020-05-27	16
27	2020-05-28	10
28	2020-05-29	17
29	2020-05-30	16
30	2020-05-31	1

Q13:What is discount correlation with order placed?

```
correlation = df['total_sales'].corr(df['discount_amount'])
correlation
```

```
1.5074151772216868e-05
```

from above corr of total sales and discount is 1.50, it is positive corr, it suggests a pattern where higher discounts are associated with higher total sales. This could potentially indicate that customers are more inclined to purchase items when discounts are offered.Higher Sales happens when discount is in much amount.

Q14:What is the correlation of difference of number of orders placed and delivered at consumer's door?

```
df['quick_time']=(df['customer_placed_order_datetime']-df['delivered_to_consumer_datetime']).dt.days
```

```
df['quick_time'].corr(df['tip_amount'])
```

```
0.0514763855315534
```

from above,the diiference between the number of orders placed and delivered to the consumer's door , there is corr is zero which , shows that no linear relationship between these variables.

Q15:What is the consumers correlation with discount amount?

```
df['customer_placed_order_datetime'] = pd.to_datetime(df['customer_placed_order_datetime'])
```

```
start_date = '2020-05-01'
end_date = '2020-06-01'
period_data = df[(df['customer_placed_order_datetime'] >= start_date) & (df['customer_placed_order_datetime'] <= end_date)]
```

```
orders_and_discounts_per_date = period_data.groupby(period_data['customer_placed_order_datetime'].dt.date).agg({'consumer_id': 'count', 'dis
```

```
orders_and_discounts_per_date
```

```
                                    consumer id  discount amount
correlation = orders_and_discounts_per_date['consumer_id'].corr(orders_and_discounts_per_date['discount_amount'])
```

```
correlation
```

```
     0.427734281827071
```

from above, a correlation coefficient of 0.4 indicates that there is a moderate positive relationship between the two variables, but it's not strong enough to imply a direct and precise relationship between them

```
                2020-05-06                          15              24.00
```

Q16:What is the corr between consumer and discount?

```
df['consumer_id'].corr(df['discount_amount']) # other way to find correlation
```