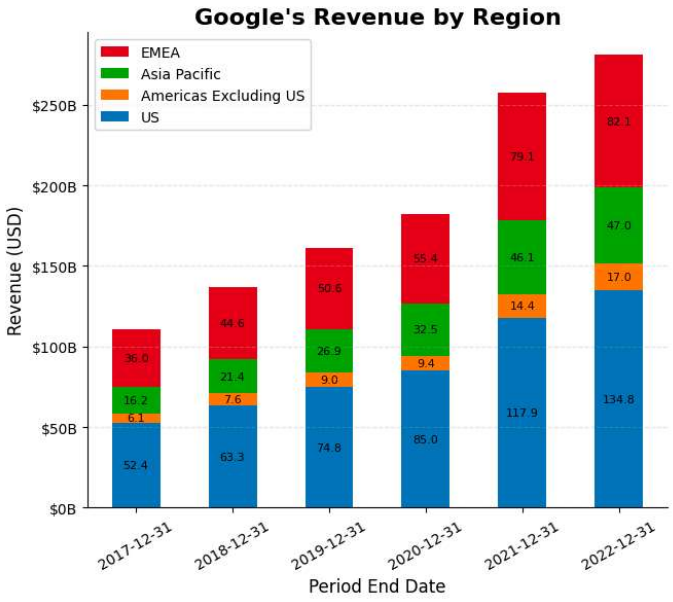
Extract Google's Revenue Metrics from 10-K Filings with Python

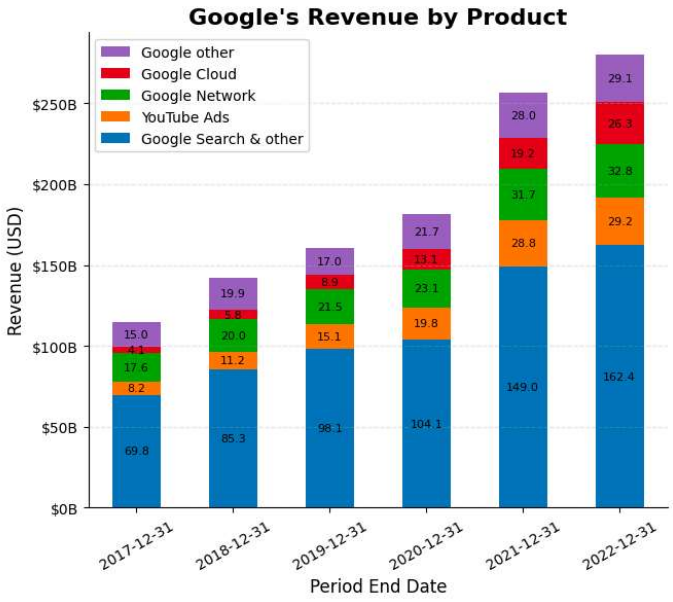
Open in Colab Download notebook

Welcome to this tutorial on how to extract revenue metrics from Google's 10-K SEC EDGAR filings and visualize them using Python's matplotlib library.

In this tutorial, we will explore how to extract revenue data by product categories and geographical regions, and create bar charts to show the progression of revenue over the years. We will look at income statements from 2016 to 2022 reported in 10-K filings from 2018 to 2022. Using the XBRL-to-JSON Converter API, we will extract income statements and revenue metrics from the XBRL data of 10-K filings and convert them into a standardized JSON format, which can be easily imported into pandas dataframes. Finally, we will extract the relevant US GAAP items and visualize the revenue data in a stacked bar chart.

Below is an illustrative example of the two bar charts that we will create to visualize Google's revenue by product categories and revenue by geographical regions, respectively.





Getting Started

While similar to our previous tutorial, Extract Financial Statements from SEC Filings with Python, this tutorial is tailored specifically towards Google's 10-K and XBRL structure.

For those already familiar with the previous tutorial, feel free to skip ahead. Otherwise, let's get started by installing the `sec-api` Python package and creating an instance of `XbrlApi`, which exposes the `.xbrl_to_json(htm_url)` function to convert XBRL data to JSON format. We will then define the function `get_income_statement(xbrl_json)` to extract the income statement from the XBRL-converted JSON data and create a pandas dataframe for us to hold all relevant metrics, such as revenue and research and development expenses per year for 2020 to 2022.

```
API_KEY = 'YOUR_API_KEY'
```

```
!pip install -q sec-api
```

```
from sec_api import XbrlApi

xbrlApi = XbrlApi(API_KEY)
```

```
# URL of Google's 10-K filings
url_10k = 'https://www.sec.gov/Archives/edgar/data/1652044/000165204423000016/goog-20221231.htm'

xbrl_json = xbrlApi.xbrl_to_json(htm_url=url_10k)
```

```
import pandas as pd

# convert XBRL-JSON of income statement to pandas dataframe
def get_income_statement(xbrl_json):
    income_statement_store = {}

    # iterate over each US GAAP item in the income statement
```

```
    for usGaapItem in xbrl_json['StatementsOfIncome']:
        values = []
        indicies = []

        for fact in xbrl_json['StatementsOfIncome'][usGaapItem]:
            # only consider items without segment. not required for our analysis.
            if 'segment' not in fact:
                index = fact['period']['startDate'] + '-' + fact['period']['endDate']
                # ensure no index duplicates are created
                if index not in indicies:
                    values.append(fact['value'])
                    indicies.append(index)

        income_statement_store[usGaapItem] = pd.Series(values, index=indicies)

    income_statement = pd.DataFrame(income_statement_store)
    # switch columns and rows so that US GAAP items are rows and each column header represents a
    return income_statement.T


income_statement_google = get_income_statement(xbrl_json)


print("Income statement from Google's 2022 10-K filing as dataframe")
print('-------------------------------------------------------------')
income_statement_google
```

```
Income statement from Google's 2022 10-K filing as dataframe
-------------------------------------------------------------
```

Out[ ]:

	2020-0 2020-1
RevenueFromContractWithCustomerExcludingAssessedTax	1825270(
CostOfRevenue	847320(
ResearchAndDevelopmentExpense	275730(
SellingAndMarketingExpense	179460(
GeneralAndAdministrativeExpense	110520(
CostsAndExpenses	1413030(
OperatingIncomeLoss	412240(
NonoperatingIncomeExpense	68580(
IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest	480820(
IncomeTaxExpenseBenefit	78130(
NetIncomeLoss	402690(
EarningsPerShareBasic	
EarningsPerShareDiluted	

Extract Income Statements from Google's 10-K Filings

In the following section, we will extract all XBRL data from Google's 10-K filings filed between 2018 and 2022. We will use the XBRL-to-JSON Converter API to convert this data and create a consolidated dataframe that holds multiple years of income statement data ranging from 2016 to 2022. Furthermore, we will merge all US GAAP revenue items across different XBRL dimensions into a single dataframe, which will assist us in filtering revenue items based on product and region.

```
# Google's 10Ks of the last 5 years, 2018 to 2022, with data from 2016 to 2022
url_10k_2018 = 'https://www.sec.gov/Archives/edgar/data/1652044/000165204419000004/goog10-kq42018
url_10k_2019 = 'https://www.sec.gov/Archives/edgar/data/1652044/000165204420000008/goog10-k2019.h
url_10k_2020 = 'https://www.sec.gov/Archives/edgar/data/1652044/000165204421000010/goog-20201231
url_10k_2021 = 'https://www.sec.gov/Archives/edgar/data/1652044/000165204422000019/goog-20211231
url_10k_2022 = 'https://www.sec.gov/Archives/edgar/data/1652044/000165204423000016/goog-20221231

xbrl_json_2018 = xbrlApi.xbrl_to_json(htm_url=url_10k_2018)
xbrl_json_2019 = xbrlApi.xbrl_to_json(htm_url=url_10k_2019)
xbrl_json_2020 = xbrlApi.xbrl_to_json(htm_url=url_10k_2020)
xbrl_json_2021 = xbrlApi.xbrl_to_json(htm_url=url_10k_2021)
xbrl_json_2022 = xbrlApi.xbrl_to_json(htm_url=url_10k_2022)
```

In 2020, Google made a change to its reporting naming convention by using the key `Revenues` instead of the previously used `RevenueFromContractWithCustomerExcludingAssessedTax`. Therefore, in this tutorial, we will address this naming inconsistency by simply renaming the element.

```
# fix naming inconsistency
xbrl_json_2020['StatementsOfIncome']['RevenueFromContractWithCustomerExcludingAssessedTax'] = xbr
del xbrl_json_2020['StatementsOfIncome']['Revenues']
xbrl_json_2021['StatementsOfIncome']['RevenueFromContractWithCustomerExcludingAssessedTax'] = xbr
del xbrl_json_2021['StatementsOfIncome']['Revenues']
```

```
income_statement_2018 = get_income_statement(xbrl_json_2018)
income_statement_2019 = get_income_statement(xbrl_json_2019)
income_statement_2020 = get_income_statement(xbrl_json_2020)
income_statement_2021 = get_income_statement(xbrl_json_2021)
income_statement_2022 = get_income_statement(xbrl_json_2022)
```

```
income_statements_merged = pd.concat([income_statement_2018,
                                      income_statement_2019,
                                      income_statement_2020,
                                      income_statement_2021,
                                      income_statement_2022], axis=0, sort=False)

# sort & reset the index of the merged dataframe
income_statements_merged = income_statements_merged.sort_index().reset_index()

# convert cells to float
income_statements_merged = income_statements_merged.applymap(lambda x: pd.to_numeric(x, errors='


print("Merged, uncleaned financials of all income statements")
print('-----------------------------------------------------')
income_statements_merged.head(10)
```

Merged, uncleaned financials of all income statements

Out[ ]:

	index	2016-01-01-2016-12-31	2017-01-01-2017-12-31	2018-01-01-2018-12-31	2019-01-01-2019-12-31	2020-01-01-2020-12-31	2021-01-0 2021-12-
0	CostOfRevenue	NaN	4.558300e+10	5.954900e+10	7.189600e+10	NaN	Na
1	CostOfRevenue	3.513800e+10	4.558300e+10	5.954900e+10	NaN	NaN	Na
2	CostOfRevenue	NaN	NaN	NaN	NaN	8.473200e+10	1.109390e+
3	CostOfRevenue	NaN	NaN	5.954900e+10	7.189600e+10	8.473200e+10	Na
4	CostOfRevenue	NaN	NaN	NaN	7.189600e+10	8.473200e+10	1.109390e+
5	CostsAndExpenses	NaN	NaN	NaN	1.276260e+11	1.413030e+11	1.789230e+
6	CostsAndExpenses	6.655600e+10	8.470900e+10	1.104980e+11	NaN	NaN	Na
7	CostsAndExpenses	NaN	NaN	1.092950e+11	1.276260e+11	1.413030e+11	Na
8	CostsAndExpenses	NaN	8.467700e+10	1.092950e+11	1.276260e+11	NaN	Na
9	CostsAndExpenses	NaN	NaN	NaN	NaN	1.413030e+11	1.789230e+

```
income_statements = income_statements_merged.groupby('index').max()

# reindex the merged dataframe using the index of the first dataframe
income_statements = income_statements.reindex(income_statement_2019.index)

# loop over the columns
for col in income_statements.columns[1:]:
    # extract start and end dates from the column label
    splitted = col.split('-')
    start = '-'.join(splitted[:3])
    end = '-'.join(splitted[3:])

    # convert start and end dates to datetime objects
    start_date = pd.to_datetime(start)
    end_date = pd.to_datetime(end)

    # calculate the duration between start and end dates
    duration = (end_date - start_date).days / 360

    # drop the column if duration is less than a year
    if duration < 1:
        income_statements.drop(columns=[col], inplace=True)

# convert datatype of cells to readable format, e.g. "2.235460e+11" becomes "223546000000"
income_statements = income_statements.apply(lambda row: pd.to_numeric(row, errors='coerce', down


print("Income statements from Google's 10-K filings (2016 to 2022) as dataframe")
print('-----------------------------------------------------------------------')
income_statements
```

Income statements from Google's 10-K filings (2016 to 2022) as dataframe

---

Out[ ]:

	RevenueFromContractWithCustomerExcludingA	Cost	ResearchAndDevelopme	SellingAndMarketi	GeneralAndAdministrati	LossContingencyLo	CostsAn	OperatingI	NonoperatingIncor	IncomeLossFromContinuingOperationsBeforeIncomeTaxesMinorityInterestAndIncomeLossFromEquityMethodIr	IncomeTaxExpe	NetI	EarningsPer	EarningsPerSh

```
# fix naming inconsistency
# xbrl_json_2021['StatementsOfIncome']['RevenueFromContractWithCustomerExcludingAssessedTax'] = x
# del xbrl_json_2021['StatementsOfIncome']['Revenues']
```

Visualize Google's Revenue by Region from 2016 to 2022

We combine all US GAAP elements in each `RevenueFromContractWithCustomerExcludingAssessedTax` list in the XBRL-JSON data for each year into a single array and then use the `pd.json_normalize()` function to create a dataframe.

However, some segment objects in the XBRL data have multiple items with `dimension` and `value` keys, forming an array. For such cases, we use the `.explode()` method to create new rows for each array item.

For instance, if we have one row with a cell containing the array

```
[
  {
    "dimension": "srt:ProductOrServiceAxis",
    "value": "goog:AdvertisingRevenueMember"
  },
  {
    "dimension": "us-gaap:StatementBusinessSegmentsAxis",
    "value": "goog:GoogleInc.Member"
  }
]
```

we would create two new rows: the first with `dimension: srt:ProductOrServiceAxis, value: goog:AdvertisingRevenueMember`, and the second with `dimension: us-gaap:StatementBusinessSegmentsAxis, value: goog:GoogleInc.Member`.

```
all_revenues_json = xbrl_json_2018['StatementsOfIncome']['RevenueFromContractWithCustomerExcludi
                    xbrl_json_2019['StatementsOfIncome']['RevenueFromContractWithCustomerExcludi
                    xbrl_json_2020['StatementsOfIncome']['RevenueFromContractWithCustomerExcludi
                    xbrl_json_2021['StatementsOfIncome']['RevenueFromContractWithCustomerExcludi
                    xbrl_json_2022['StatementsOfIncome']['RevenueFromContractWithCustomerExcludi

all_revenues = pd.json_normalize(all_revenues_json)

# explode segment list of dictionaries
all_revenues = all_revenues.explode('segment')

segment_split = all_revenues['segment'].apply(pd.Series)
segment_split = segment_split.rename(columns={'dimension': 'segment.dimension', 'value': 'segmen
segment_split = segment_split.drop(0, axis=1)

all_revenues = all_revenues.combine_first(segment_split)
all_revenues = all_revenues.drop('segment', axis=1)
all_revenues = all_revenues.reset_index(drop=True)

all_revenues
```

Out[ ]:

	decimals	period.endDate	period.startDate	segment.dimension	
0	-6	2016-12-31	2016-01-01	NaN	
1	-6	2016-12-31	2016-01-01	srt:ProductOrServiceAxis	
2	-6	2016-12-31	2016-01-01	us-gaap:StatementBusinessSegmentsAxis	
3	-6	2016-12-31	2016-01-01	srt:ProductOrServiceAxis	goog:Goc
4	-6	2016-12-31	2016-01-01	us-gaap:StatementBusinessSegmentsAxis	
...	...	...	...	...	
211	-6	2021-12-31	2021-01-01	us-gaap:ReclassificationOutOfAccumulatedOtherC...	us-gaap:Rec
212	-6	2021-12-31	2021-01-01	us-gaap:StatementEquityComponentsAxis	gaap:Accum
213	-6	2022-12-31	2022-01-01	us-gaap:DerivativeInstrumentRiskAxis	us-
214	-6	2022-12-31	2022-01-01	us-gaap:ReclassificationOutOfAccumulatedOtherC...	us-gaap:Rec
215	-6	2022-12-31	2022-01-01	us-gaap:StatementEquityComponentsAxis	gaap:Accum

216 rows × 7 columns

Next, we aim to gain insight into the data's structure and potential values to develop a strategy for the best extraction and visualization of the revenue metrics. To achieve this, we create a pivot table along the

`segment.dimension` and `segment.value` columns, which enables us to identify all the potential values for each XBRL dimension.

```
xbrl_dimensions = all_revenues.pivot(columns='segment.dimension', values='segment.value').dropna

xbrl_dimensions = pd.DataFrame([(col, xbrl_dimensions[col].unique()) for col in xbrl_dimensions.
                               columns=['segment.dimension', 'segment.values'])

xbrl_dimensions
```

Out[ ]:

	segment.dimension	segment.values
0	NaN	[nan]
1	srt:ProductOrServiceAxis	[goog:AdvertisingRevenueMember, nan, goog:Goog...
2	srt:StatementGeographicalAxis	[nan, country:US, goog:AmericasExcludingUnited...
3	us-gaap:DerivativeInstrumentRiskAxis	[nan, us-gaap:ForeignExchangeContractMember]
4	us-gaap:IncomeStatementLocationAxis	[nan, us-gaap:SalesMember]
5	us-gaap:ReclassificationOutOfAccumulatedOtherC...	[nan, us-gaap:ReclassificationOutOfAccumulated...
6	us-gaap:StatementBusinessSegmentsAxis	[nan, goog:GoogleInc.Member, us-gaap:AllOtherS...
7	us-gaap:StatementEquityComponentsAxis	[nan, us-gaap:AccumulatedGainLossNetCashFlowHe...

```
# list(all_revenues['segment.value'].drop_duplicates())
```

Based on the pivot table above, we can observe that the `srt:StatementGeographicalAxis` dimension has four different values, namely

- country:US
- goog:AmericasExcludingUnitedStatesMember
- srt:AsiaPacificMember
- us-gaap:EMEAMember

To focus only on the revenue generated from Google's geographical regions, we filter all revenue GAAP items by `srt:StatementGeographicalAxis` and create a new pivot table with `period.endDate` (the revenue reporting years) as the index and `segment.value` (the regions) as the columns. Each cell in the pivot table represents the revenue generated by Google in a specific region during a particular year.

```
all_revenues['value'] = all_revenues['value'].astype(int)

mask = all_revenues['segment.dimension'] == 'srt:StatementGeographicalAxis'

revenue_region = all_revenues[mask]

revenue_region = revenue_region.drop_duplicates(subset=['period.endDate', 'segment.value'])

# pivot the dataframe to create a new dataframe with period.endDate as the index,
# segment.value as the columns, and value as the values
revenue_region_pivot = revenue_region.pivot(index='period.endDate', columns='segment.value', val
```

```
print("Google's revenues by region from 2016 to 2022")
print('-------------------------------------------')
revenue_region_pivot
```

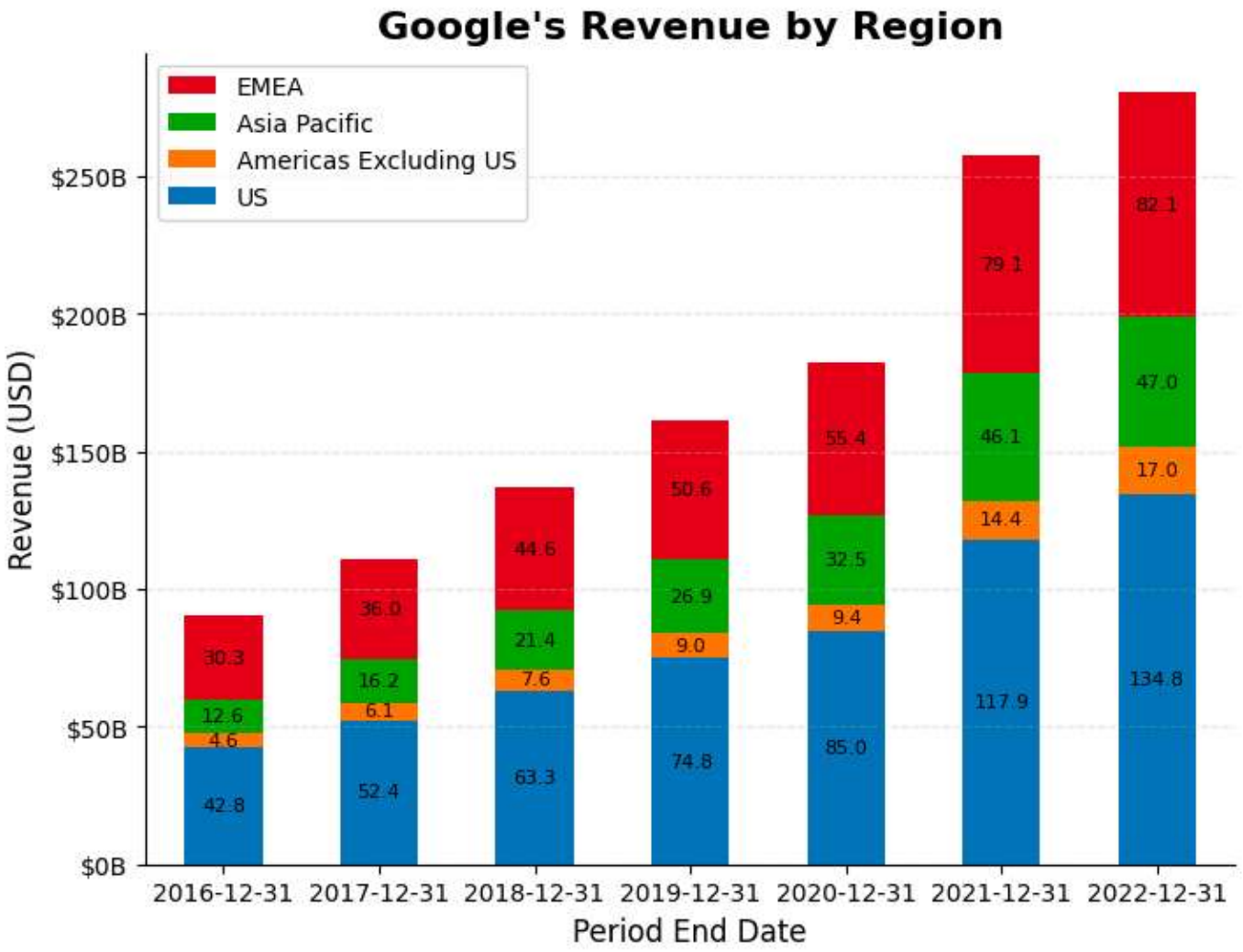
```
Google's revenues by region from 2016 to 2022
-------------------------------------------
```

Out[ ]:

segment.value	country:US	goog:AmericasExcludingUnitedStatesMember	srt:AsiaPacificMember	gaap:
period.endDate				
2016-12-31	42781000000	4628000000	12559000000	
2017-12-31	52449000000	6125000000	16235000000	
2018-12-31	63269000000	7609000000	21374000000	
2019-12-31	74843000000	8986000000	26928000000	
2020-12-31	85014000000	9417000000	32550000000	
2021-12-31	117854000000	14404000000	46123000000	
2022-12-31	134814000000	16976000000	47024000000	

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# plot the histogram bar chart
ax = revenue_region_pivot.plot(kind='bar', stacked=True, figsize=(8, 6))

# rotate the x-axis labels by 0 degrees
plt.xticks(rotation=0)

# set the title and labels for the chart
ax.set_title("Google's Revenue by Region", fontsize=16, fontweight='bold')
ax.set_xlabel('Period End Date', fontsize=12)
ax.set_ylabel('Revenue (USD)', fontsize=12)

# set the legend properties
ax.legend(title='Product Category', loc='upper left', fontsize='small', title_fontsize=10)

# add gridlines
ax.grid(axis='y', linestyle='--', alpha=0.3)

# remove the top and right spines
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# format y-axis ticks to show values in millions in dollars
formatter = ticker.FuncFormatter(lambda x, pos: '$%1.0fB' % (x*1e-9))
plt.gca().yaxis.set_major_formatter(formatter)

# map the original labels to new labels
label_map = {
    'country:US': 'US',
    'goog:AmericasExcludingUnitedStatesMember': 'Americas Excluding US',
```

```
    'srt:AsiaPacificMember': 'Asia Pacific',
    'us-gaap:EMEAMember': 'EMEA'
}

# create a list of new labels based on the original labels
new_labels = [label_map[label] for label in sorted(revenue_region['segment.value'].unique())]
handles, _ = ax.get_legend_handles_labels()
plt.legend(handles=handles[::-1], labels=new_labels[::-1])

# add the values in billions of dollars to each part of the bar
for p in ax.containers:
    ax.bar_label(p, labels=['%.1f' % (v/1e9) for v in p.datavalues],
                 label_type='center', fontsize=8)

plt.show()
```

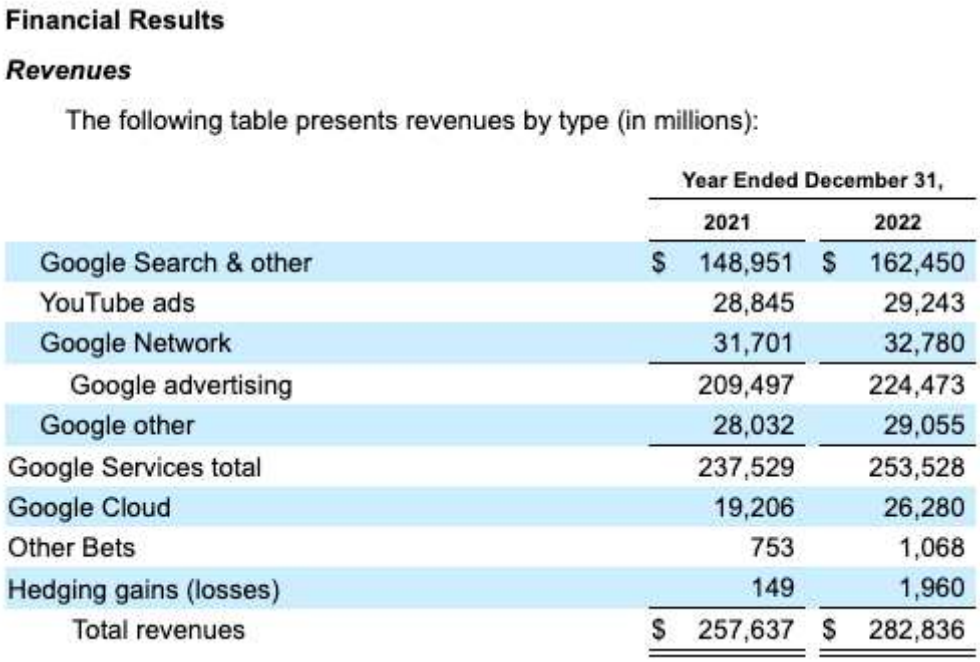


Visualize Google's Revenue by Product Category from 2016 to 2022

In this section, we will repeat most of the steps from the previous section but with a revenue item filter that only considers `segment.dimension` revenue items with `srt:ProductOrServiceAxis` or `us-gaap:StatementBusinessSegmentsAxis` as values since Google reported their cloud revenue in the `us-`

`gaap:StatementBusinessSegmentsAxis` dimension, while it reported all other metrics such as YouTube advertising revenue under the `srt:ProductOrServiceAxis` dimension.

To make it easier for you to compare the extracted values to the human-readable values in Google's 2022 10-K, we have added a screenshot of the financial metrics below.



Financial Results

Revenues

The following table presents revenues by type (in millions):

	Year Ended December 31,	
	2021	2022
Google Search & other	$ 148,951	$ 162,450
YouTube ads	28,845	29,243
Google Network	31,701	32,780
Google advertising	209,497	224,473
Google other	28,032	29,055
Google Services total	237,529	253,528
Google Cloud	19,206	26,280
Other Bets	753	1,068
Hedging gains (losses)	149	1,960
Total revenues	$ 257,637	$ 282,836

To facilitate a better understanding, we have provided a mapping between the human-readable labels and their corresponding XBRL element names, as the element names of the US GAAP items inside the XBRL data differ from the financial results table in the 10-K filing.

Human-readable metric	XBRL metric
Google Search & other	GoogleSearchOtherMember
YouTube ads	YouTubeAdvertisingRevenueMember
Google Network	GoogleNetworkMembersPropertiesMember (2016-2019), GoogleNetworkMember (2020-2022)
Google advertising	GoogleAdvertisingRevenueMember
Google other	OtherRevenuesMember
Google Cloud	GoogleCloudMember

```
GoogleAdvertisingRevenueMember =
    GoogleSearchOtherMember
  + YouTubeAdvertisingRevenueMember
  + GoogleNetworkMembersPropertiesMember OR GoogleNetworkMember
```

An inconsistency in the naming convention can be observed as the revenue reported for Google Network was previously referred to as `GoogleNetworkMembersPropertiesMember` from 2016 to 2019, which changed to `GoogleNetworkMember` since 2020. To address this inconsistency, we will merge the values of both items and create a new consolidated item named `GoogleNetwork`.

```
all_revenues['value'] = all_revenues['value'].astype(int)

mask_1 = all_revenues['segment.dimension'] == 'srt:ProductOrServiceAxis'
mask_2 = all_revenues['segment.dimension'] == 'us-gaap:StatementBusinessSegmentsAxis'

revenue_product = all_revenues[mask_1 | mask_2]

revenue_product = revenue_product.drop_duplicates(subset=['period.endDate', 'segment.value'])

# pivot the dataframe to create a new dataframe with period.endDate as the index,
# segment.value as the columns, and value as the values
revenue_product_pivot = revenue_product.pivot(index='period.endDate', columns='segment.value', v


print("Google's revenues by product from 2016 to 2022")
print('-----------------------------------------------')
revenue_product_pivot
```

```
Google's revenues by product from 2016 to 2022
-----------------------------------------------
```

Out[ ]:

segment.value	goog:AdvertisingRevenueMember	goog:GoogleAdvertisingRevenueMember	goog:Google(
period.endDate			
2016-12-31	7.938300e+10	NaN	
2017-12-31	9.537500e+10	9.557700e+10	
2018-12-31	1.163180e+11	1.164610e+11	
2019-12-31	NaN	1.348110e+11	
2020-12-31	NaN	1.469240e+11	
2021-12-31	NaN	2.094970e+11	
2022-12-31	NaN	2.244730e+11	

```
# merge: goog:GoogleCloudMember + goog:GoogleNetworkMembersPropertiesMember
revenue_product_pivot['goog:GoogleNetwork'] = revenue_product_pivot['goog:GoogleNetworkMember'].
revenue_product_pivot = revenue_product_pivot.drop(['goog:GoogleNetworkMember', 'goog:GoogleNetw

revenue_product_pivot = revenue_product_pivot[['goog:GoogleSearchOtherMember',
                                               'goog:YouTubeAdvertisingRevenueMember',
                                               'goog:GoogleNetwork',
                                               'goog:GoogleCloudMember',
                                               'goog:OtherRevenuesMember']]

# remove 2016 row
revenue_product_pivot = revenue_product_pivot.iloc[1:]

revenue_product_pivot
```

Out[ ]:

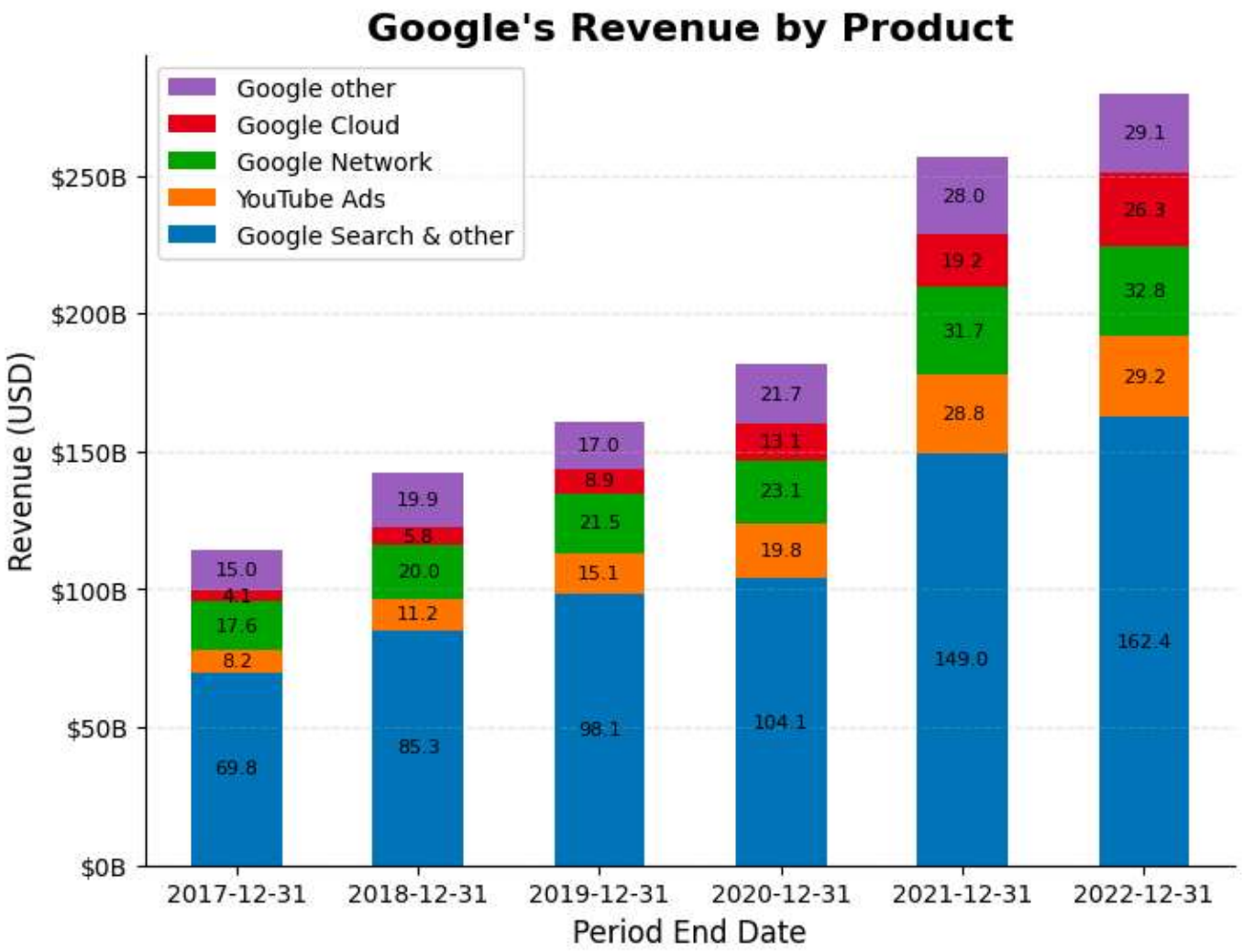
segment.value period.endDate	goog:GoogleSearchOtherMember	goog:YouTubeAdvertisingRevenueMember	goog:Google
2017-12-31	6.981100e+10	8.150000e+09	1.75
2018-12-31	8.529600e+10	1.115500e+10	1.99
2019-12-31	9.811500e+10	1.514900e+10	2.15
2020-12-31	1.040620e+11	1.977200e+10	2.30
2021-12-31	1.489510e+11	2.884500e+10	3.17
2022-12-31	1.624500e+11	2.924300e+10	3.27

```
sorted(list(revenue_product_pivot.columns.unique()))
```

Out[ ]:
```
['goog:GoogleCloudMember',
 'goog:GoogleNetwork',
 'goog:GoogleSearchOtherMember',
 'goog:OtherRevenuesMember',
 'goog:YouTubeAdvertisingRevenueMember']
```

```
# plot the histogram bar chart
ax = revenue_product_pivot.plot(kind='bar', stacked=True, figsize=(8, 6))

# rotate the x-axis labels by 0 degrees
plt.xticks(rotation=0)

# set the title and labels for the chart
ax.set_title("Google's Revenue by Product", fontsize=16, fontweight='bold')
ax.set_xlabel('Period End Date', fontsize=12)
ax.set_ylabel('Revenue (USD)', fontsize=12)

# set the legend properties
ax.legend(title='Product Category', loc='upper left', fontsize='small', title_fontsize=10)

# add gridlines
ax.grid(axis='y', linestyle='--', alpha=0.3)

# remove the top and right spines
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# format y-axis ticks to show values in millions in dollars
formatter = ticker.FuncFormatter(lambda x, pos: '$%1.0fB' % (x*1e-9))
plt.gca().yaxis.set_major_formatter(formatter)

# map the original labels to new labels
label_map = {
    'goog:GoogleCloudMember': 'Google Cloud',
    'goog:GoogleNetwork': 'Google Network',
    'goog:YouTubeAdvertisingRevenueMember': 'YouTube Ads',
    'goog:GoogleSearchOtherMember': 'Google Search & other',
    'goog:OtherRevenuesMember': 'Google other',
}

# create a list of new labels based on the original labels
```

```
new_labels = [label_map[label] for label in list(revenue_product_pivot.columns.unique())]
handles, _ = ax.get_legend_handles_labels()
plt.legend(handles=handles[::-1], labels=new_labels[::-1])

# add the values in billions of dollars to each part of the bar
for p in ax.containers:
    ax.bar_label(p, labels=['%.1f' % (v/1e9) for v in p.datavalues],
                 label_type='center', fontsize=8)

plt.show()
```



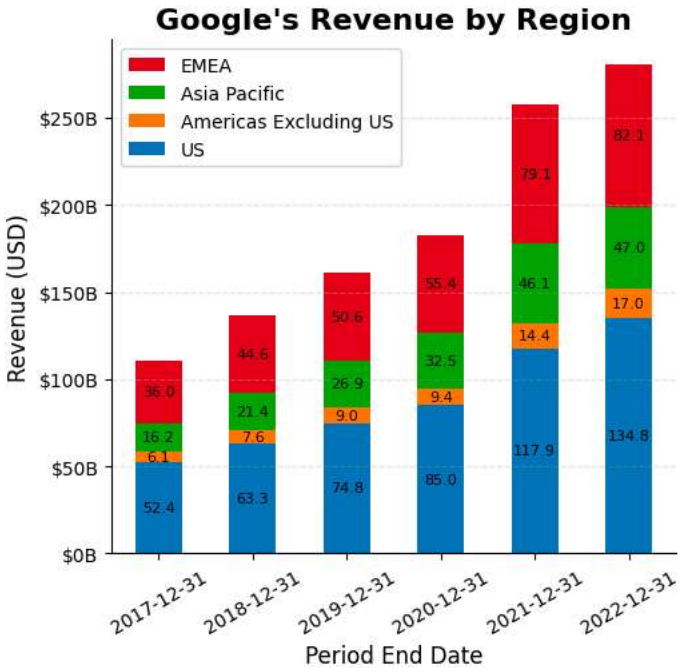
Create a Figure with Two Revenue Charts: Region and Product

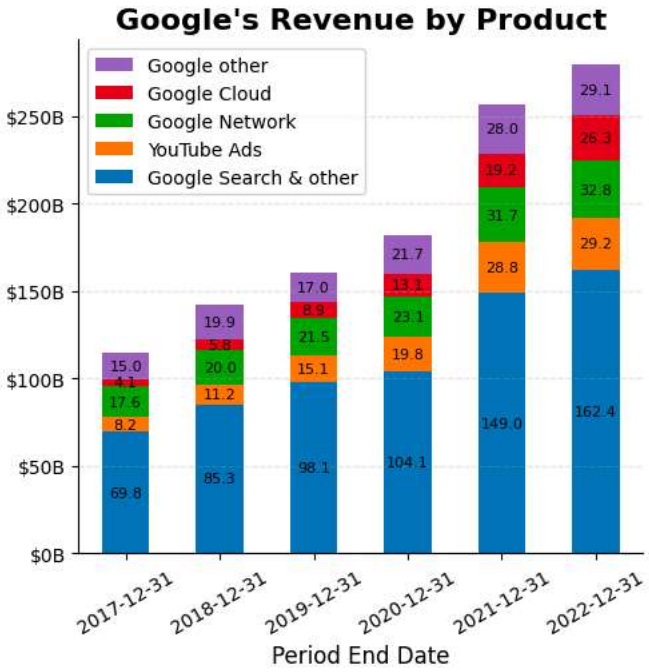
We will now combine both charts into a single figure, presenting them side by side. To ensure that the x-axis is aligned (i.e., the years from 2017 to 2022), we will remove the 2016 data points from `revenue_region_pivot`, since we don't have reliable data in `revenue_product_pivot` for that year.

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

revenue_region_pivot.iloc[1:].plot(kind='bar', stacked=True, ax=ax1)
revenue_product_pivot.plot(kind='bar', stacked=True, ax=ax2)

# set the title and labels for the chart
```

```
ax1.set_title("Google's Revenue by Region", fontsize=16, fontweight='bold')
ax1.set_xlabel('Period End Date', fontsize=12)
ax1.set_ylabel('Revenue (USD)', fontsize=12)
ax2.set_title("Google's Revenue by Product", fontsize=16, fontweight='bold')
ax2.set_xlabel('Period End Date', fontsize=12)

# set the legend properties
ax1.legend(title='Region', loc='upper left', fontsize='small', title_fontsize=10)
ax2.legend(title='Product Category', loc='upper left', fontsize='small', title_fontsize=10)

# add gridlines
ax1.grid(axis='y', linestyle='--', alpha=0.3)
ax2.grid(axis='y', linestyle='--', alpha=0.3)

# remove the top and right spines
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax2.spines['right'].set_visible(False)

# map the original labels to new labels
label_map_1 = {
    'country:US': 'US',
    'goog:AmericasExcludingUnitedStatesMember': 'Americas Excluding US',
    'srt:AsiaPacificMember': 'Asia Pacific',
    'us-gaap:EMEAMember': 'EMEA'
}
label_map_2 = {
    'goog:GoogleCloudMember': 'Google Cloud',
    'goog:GoogleNetwork': 'Google Network',
    'goog:YouTubeAdvertisingRevenueMember': 'YouTube Ads',
    'goog:GoogleSearchOtherMember': 'Google Search & other',
    'goog:OtherRevenuesMember': 'Google other',
}

# create a list of new labels based on the original labels
new_labels1 = [label_map_1[label] for label in list(revenue_region_pivot.columns.unique())]
new_labels2 = [label_map_2[label] for label in list(revenue_product_pivot.columns.unique())]
handles1, _ = ax1.get_legend_handles_labels()
handles2, _ = ax2.get_legend_handles_labels()
ax1.legend(handles=handles1[::-1], labels=new_labels1[::-1])
ax2.legend(handles=handles2[::-1], labels=new_labels2[::-1])

# add the values in billions of dollars to each part of the bar
for p in ax1.containers:
    ax1.bar_label(p, labels=['%.1f' % (v/1e9) for v in p.datavalues],
                  label_type='center', fontsize=8)

for p in ax2.containers:
    ax2.bar_label(p, labels=['%.1f' % (v/1e9) for v in p.datavalues],
                  label_type='center', fontsize=8)

# format y-axis ticks to show values in millions in dollars
formatter = ticker.FuncFormatter(lambda x, pos: '$%1.0fB' % (x*1e-9))
ax1.yaxis.set_major_formatter(formatter)
ax2.yaxis.set_major_formatter(formatter)

# rotate the x-axis labels by 0 degrees
ax1.tick_params(axis='x', labelrotation=30)
ax2.tick_params(axis='x', labelrotation=30)
```

```
plt.show()
```





Download & Render

API

Insider Trading Data

API

XBRL-to-JSON

Converter

CIK, CUSIP, Ticker

Mapping

Executive

Compensation API

Form 13D/13G

Search API

10-K/10-Q/8-K Item

Extractor

Float (Outstanding

Shares) API