In [21]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from dateutil.relativedelta import relativedelta

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

import plotly.express as px
from mpl_toolkits.mplot3d import Axes3D

from itertools import cycle
from itertools import product

from googletrans import Translator

%matplotlib inline
```

In [22]:
```python
#Read in .csv files into pandas dataframes
train        = pd.read_csv('sales_train.csv')
test         = pd.read_csv('test.csv').set_index('ID')
submission   = pd.read_csv('sample_submission.csv')
items        = pd.read_csv('items.csv')
item_cats    = pd.read_csv('item_categories.csv')
shops        = pd.read_csv('shops.csv')
shops_t      = pd.read_csv('shops_translated.csv')
items_t      = pd.read_csv('items_translated_text.csv')
```

In [23]:
```python
#Down casts the data entries from int64 to int32 and float64 to float32
#This reduces the size of the records by almost half. (From 134mb to 61mb)
def downcast_dtypes(df):
    float_cols = [c for c in df if df[c].dtype == "float64"]
    int_cols = [c for c in df if df[c].dtype in ["int64", "int32"]]
    df[float_cols] = df[float_cols].astype(np.float32)
    df[int_cols] = df[int_cols].astype(np.int16)
    return df
```

In [24]:
```python
#Calls the downcasting function
train       = downcast_dtypes(train)
test        = downcast_dtypes(test)
submission = downcast_dtypes(submission)
items       = downcast_dtypes(items)
item_cats   = downcast_dtypes(item_cats)
shops       = downcast_dtypes(shops)
shops_t     = downcast_dtypes(shops_t)
items_t     = downcast_dtypes(items_t)
```

In [25]:
```python
#adds item id to main train dataframe and drops the string name column
train = train.merge(items, on='item_id')
train = train.drop(columns = 'item_name')
```

In [26]:
```python
train
```

Out[26]:

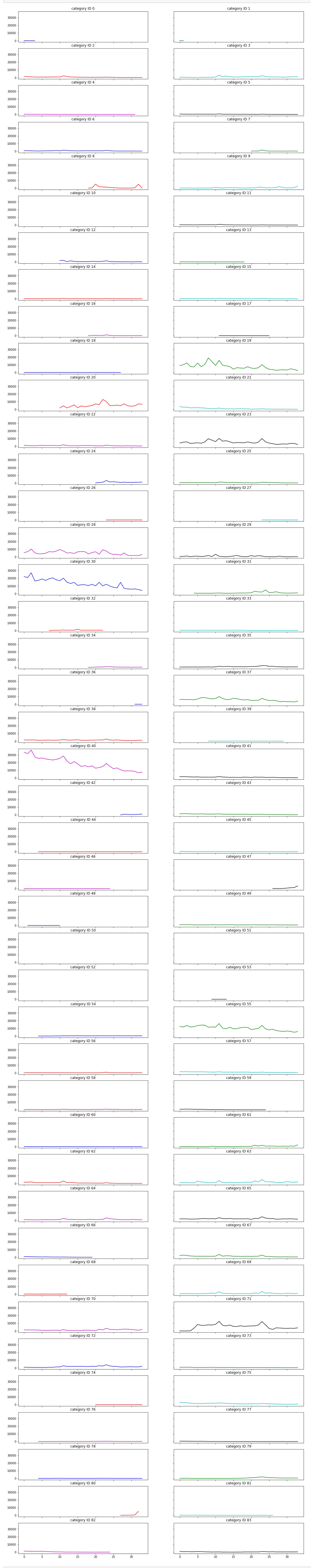
	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_category_id
0	02.01.2013	0	59	22154	999.0	1.0	37
1	23.01.2013	0	24	22154	999.0	1.0	37
2	20.01.2013	0	27	22154	999.0	1.0	37
3	02.01.2013	0	25	22154	999.0	1.0	37
4	03.01.2013	0	25	22154	999.0	1.0	37
...	...	...	...	...	...	...	...
2935844	17.10.2015	33	25	8428	249.0	1.0	40
2935845	01.10.2015	33	25	7903	12198.0	1.0	15
2935846	29.10.2015	33	25	7610	2890.0	1.0	64
2935847	22.10.2015	33	25	7635	2100.0	1.0	64
2935848	01.10.2015	33	25	7640	4040.0	1.0	64

2935849 rows × 7 columns

In [8]:
```python
### Exploratory Merging and graphing cell for preliminary data analysis ###

# group data by month and shop_id, return sum of items sold per shop per month
month_group = pd.DataFrame(train.groupby(['date_block_num', 'shop_id'])['item_cnt_day'].sum().reset_index())

# added the item_category into sales_train
merged = pd.merge(train, items[['item_id', 'item_category_id']], on = 'item_id')

# group data by month and category_id, return sum of items sold per category per month
category_group = pd.DataFrame(merged.groupby(['date_block_num', 'item_category_id'])['item_cnt_day'].sum().reset_index())
```

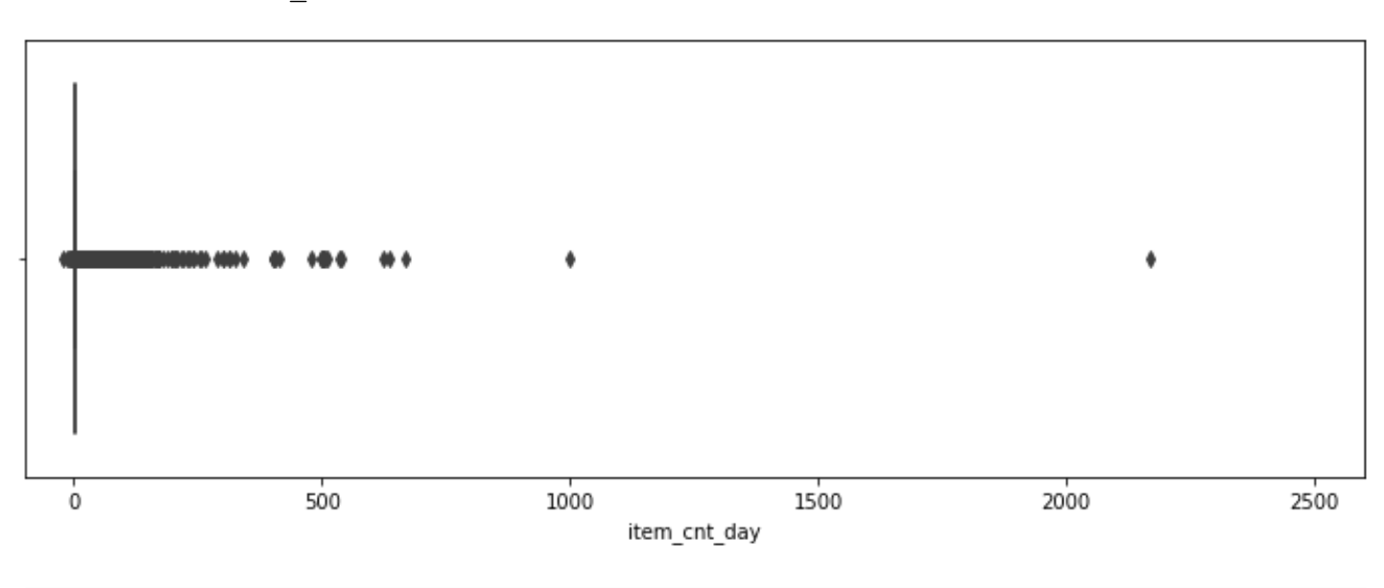
In [9]:
```python
# plotting data for visual reference

# Plot for items sold per shop per month
fig1, axes = plt.subplots(30, 2, figsize = (18,80), sharex = True, sharey = True)
shop = 0
axis = []

for i in range(int(np.max(month_group['shop_id'])/2+1)):
    axis.append([[[],[]],[[],[]]])

for row in range(30):
    for col in range(2):
        for date in range(len(month_group['shop_id'])):
            if month_group['shop_id'][date] == shop:
                axis[row][col][0].append(month_group['date_block_num'][date])
                axis[row][col][1].append(month_group['item_cnt_day'][date])
        shop+=1

shop = 0
cycol = cycle('bgrcmk')
for row in range(30):
    for col in range(2):
        axes[row, col].plot(axis[row][col][0], axis[row][col][1], color = next(cycol))
        axes[row, col].set_title('shop ' + str(shop))
        shop += 1
```

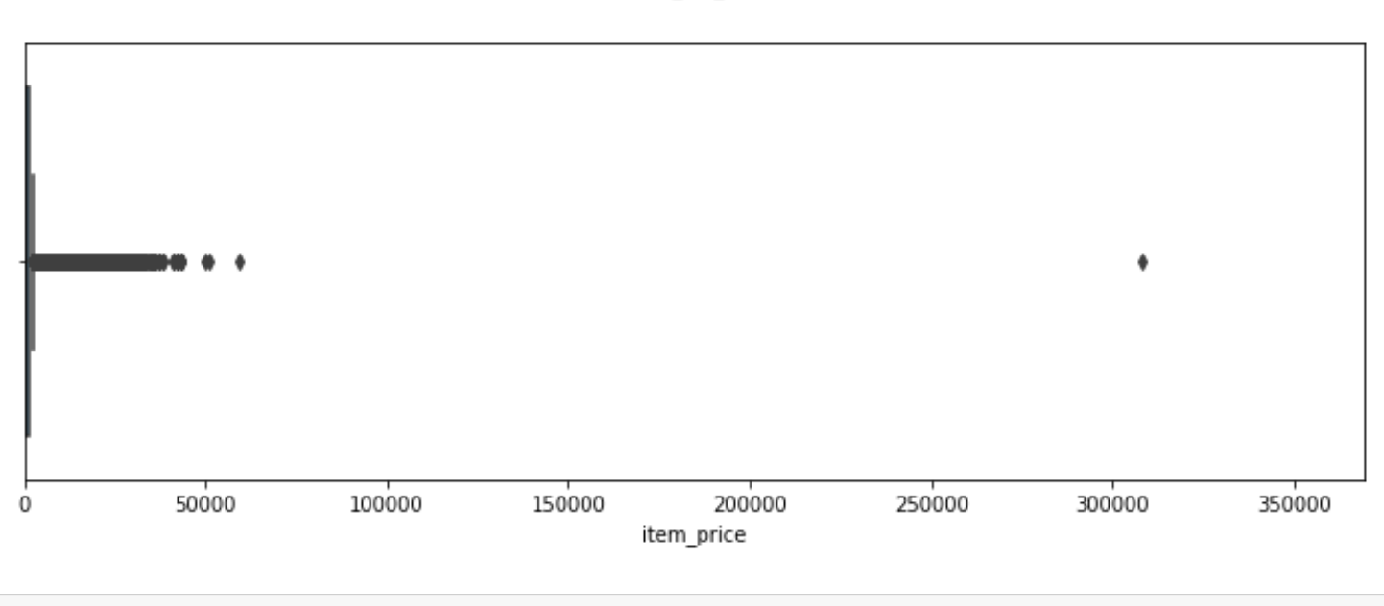
In [10]:
```python
# plotting data for visual reference

# Plot for items sold per category per month
fig2, axes2 = plt.subplots(42, 2, figsize = (18,100), sharex = True, sharey = True)
category = 0
axis2 = []

for i in range(int(np.max(category_group['item_category_id'])/2+1)):
    axis2.append([[[],[]],[[],[]]])

for row in range(42):
    for col in range(2):
        for date in range(len(category_group['item_category_id'])):
            if category_group['item_category_id'][date] == category:
                axis2[row][col][0].append(category_group['date_block_num'][date])
                axis2[row][col][1].append(category_group['item_cnt_day'][date])
        category+=1

category = 0
cycol = cycle('bgrcmk')
for row in range(42):
    for col in range(2):
        axes2[row, col].plot(axis2[row][col][0], axis2[row][col][1], color = next(cycol))
        axes2[row, col].set_title('category ID ' + str(category))
        category += 1
```



In [14]:
```python
# Boxplot to show outliers in item_cnt_day and item_price
# This shows the outlier in price and the one instance of an unusually high sales count
plt.figure(figsize=(12,4))
plt.xlim(-100, train.item_cnt_day.max()*1.2)
sns.boxplot(x=train.item_cnt_day)

plt.figure(figsize=(12,4))
plt.xlim(train.item_price.min()*1.1, train.item_price.max()*1.2)
sns.boxplot(x=train.item_price)
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2509f7f4f88>
```





In [30]:
```python
#Removes outliers from train
train = train[train.item_price < 90000]
train = train[train.item_cnt_day < 999]

#replaces the negative price item with the median item_price of all items with the id of 2973 and in shop id 32
median = train[(train.shop_id==32)&(train.item_id==2973)&(train.date_block_num==4)&(train.item_price>0)].item_price.median()
```

In [31]:
```python
# moves item_cnt_day to the last column
train_cnt = train['item_cnt_day']
train.drop(labels=['item_cnt_day'], axis=1, inplace = True)
train.insert(6, 'item_cnt_day', train_cnt)
```

In [32]:
```python
train
```

Out[32]:

	date	date_block_num	shop_id	item_id	item_price	item_category_id	item_cnt_day
0	02.01.2013	0	59	22154	999.0	37	1.0
1	23.01.2013	0	24	22154	999.0	37	1.0
2	20.01.2013	0	27	22154	999.0	37	1.0
3	02.01.2013	0	25	22154	999.0	37	1.0
4	03.01.2013	0	25	22154	999.0	37	1.0
...	...	...	...	...	...	...	...
2935844	17.10.2015	33	25	8428	249.0	40	1.0
2935845	01.10.2015	33	25	7903	12198.0	15	1.0
2935846	29.10.2015	33	25	7610	2890.0	64	1.0
2935847	22.10.2015	33	25	7635	2100.0	64	1.0
2935848	01.10.2015	33	25	7640	4040.0	64	1.0

2935846 rows × 7 columns

In [36]:
```python
#sums up daily sales into monthly sales in column item_cnt_day and renames it item_cnt_month

train_grouped_month = pd.DataFrame(train.groupby(['date_block_num', 'shop_id', 'item_category_id', 'item_id', 'item_price'])['item_cnt_day'].sum().reset_index())
train_grouped_month.rename(columns = {'item_cnt_day':'item_cnt_month'}, inplace = True)
```

In [37]:
```python
train_grouped_month
```

Out[37]:

	date_block_num	shop_id	item_category_id	item_id	item_price	item_cnt_month
0	0	0	2	5572	1322.0	10.0
1	0	0	2	5573	560.0	1.0
2	0	0	2	5575	806.0	4.0
3	0	0	2	5576	2231.0	5.0
4	0	0	2	5609	2381.0	1.0
...	...	...	...	...	...	...
1739014	33	59	79	17717	1250.0	4.0
1739015	33	59	79	17717	1999.0	1.0
1739016	33	59	83	22087	119.0	6.0
1739017	33	59	83	22088	119.0	2.0
1739018	33	59	83	22091	179.0	1.0

1739019 rows × 6 columns

In [39]:
```python
#Translate entire csv frame of russian names into english
#Already done.
#def translate_and_save():
#     t = Translator()
#     items['english_name'] = items['item_name'].apply(lambda x: t.translate(x).text)
#     #shops['english_name'] = shops['shop_name'].apply(lambda x: t.translate(x).text)
#     items.to_csv('items_translated_text.csv')
```

In [40]:
```python
# Changes numerical, categorical features into strings to properly be represented as categorical in onehotencoding
# nominal intergers can not be converted to binary encoding, convert to string
train_grouped_month['date_block_num']    = [('month ' + str(i)) for i in train_grouped_month['date_block_num']]
train_grouped_month['shop_id']           = [('shop ' + str(i)) for i in train_grouped_month['shop_id']]
train_grouped_month['item_category_id'] = [('item_category ' + str(i)) for i in train_grouped_month['item_category_id']]
train_grouped_month['item_id']           = [('item ' + str(i)) for i in train_grouped_month['item_id']]
```

In [41]:
```python
train_grouped_month
```

Out[41]:

	date_block_num	shop_id	item_category_id	item_id	item_price	item_cnt_month
0	month 0	shop 0	item_category 2	item 5572	1322.0	10.0
1	month 0	shop 0	item_category 2	item 5573	560.0	1.0
2	month 0	shop 0	item_category 2	item 5575	806.0	4.0
3	month 0	shop 0	item_category 2	item 5576	2231.0	5.0
4	month 0	shop 0	item_category 2	item 5609	2381.0	1.0
...	...	...	...	...	...	...
1739014	month 33	shop 59	item_category 79	item 17717	1250.0	4.0
1739015	month 33	shop 59	item_category 79	item 17717	1999.0	1.0
1739016	month 33	shop 59	item_category 83	item 22087	119.0	6.0
1739017	month 33	shop 59	item_category 83	item 22088	119.0	2.0
1739018	month 33	shop 59	item_category 83	item 22091	179.0	1.0

1739019 rows × 6 columns

In [44]:
```python
# stores all feature columns in the x variable, and stores the target variable as the last column in y variable
x = train_grouped_month.iloc[:, :-1].values
y = train_grouped_month.iloc[:,-1].values
```

In [45]:
```python
# Encoding categorical data
# provides a value to data that can be then used in regression equations, eg. Friday = 1 etc.

ct = ColumnTransformer([('encoder', OneHotEncoder(), [0, 1, 2, 3])], remainder = 'passthrough')
x = ct.fit_transform(x)
```

In [46]:
```python
# split data set into training set and testing set
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

In [55]:
```python
# fitting multiple linear regression to the training set
regressor = LinearRegression()
regressor.fit(X_train, Y_train)

# prediction of test split from original data, train_grouped_month
y_prediction = regressor.predict(X_test)

# The output of the this is the r squared
regressor.score(X_test, Y_test)
```

Out[55]:
```
0.24532067463255935
```

In [ ]:
```python

```

In [48]:
```python
# This is for playing around with different items and shops
# sample format of test prediction = ['month 0', 'shop 0', 'item_category 2', 'item 5572', 1322]

# A simple program to see how the predictor would work

#from scipy import stats
#def auto_predictor():
#    global ct
#    month = input('Please enter the month you want to predict. Enter the digit of the month, eg. Jan = 0 : ')
#    shop = input('Please enter the shop ID, eg. 2 : ')
#    item = input("Please enter the item ID you wish to predict, eg. for item 55, enter '55' : ")
#    item_cat = (items.loc[items['item_id'] == int(item), ['item_category_id']].values)[0][0]
#    prices = train.loc[train['item_id'] == int(item), ['item_price']].values
#    price = (stats.mode(prices))[0][0][0]

#    z = ['month ' + str(month), 'shop ' + str(shop), 'item_category ' + str(item_cat), 'item ' + str(item), price]
#    z = np.array(z, dtype = object).reshape(1, -1)
#    z = ct.transform(z)
#    z_pred = regressor.predict(z)

#    return(z_pred[0])

# Things to check!!!
# ***There are multiple prices available for the item in that specific shop, provide an option to choose price
```

In [25]: %pandas_info
```
Pandas 2.2.2, numpy 1.26.4, pyarrow 17.0.0, scipy 1.13.1, pytz 2025.2, dateutil 2.9.0.post0, pip 25.1.1, Cython 3.0.11, sqlalchemy 2.0.41, scipy.ext.numba 0.59.1, jinja2 3.1.6, bottleneck 1.4.2, numexpr 2.10.2, matplotlib 3.10.0, tzdata 2025.2, xlsxwriter 3.2.9, fastparquet 2024.11.0, fsspec 2025.3.0, gcsfs 2025.3.0, pandas_gbq 0.29.1, pyarrow_hotfix 0.6
```
Out[25]:
```
None
```

In [26]: %matplotlib inline
```
Using matplotlib backend: inline
```
In [27]: import os, sys, time, re
```
os 3.10.6(v3.10.6:9c7b4bd164, Aug 1 2022, 17:13:49)
sys 3.10.6 (main, Aug 22 2025, 17:24:27) [GCC 11.4.0]
time 2.6.21
re 2.2.1
```
In [28]: %load_ext dotenv, %dotenv -v -o -override
```
dotenv 1.0.1, 0.1.0(v0.1.0-2-g7a1a1a1), 2025-05-16T15:51:34Z
```
In [29]: %load_ext google.cloud.bigquery.magics, %bigquery --project=$GOOGLE_CLOUD_PROJECT --verbose --max_results=10 --use_bqstorage_api --use_rest_api --bytes=10000 --progressbar_type=tqdm_notebook --destination_table=$GOOGLE_CLOUD_PROJECT.$GOOGLE_CLOUD_BIGQUERY_DATASET.$GOOGLE_CLOUD_BIGQUERY_TEMP_TABLE --destination_table_ttl=86400 --dry_run --maximum_bytes_billed=10000000000 --no_cache --no_query_cache --no_extensions
```
Connected to google.cloud.bigquery.magics.bigquery.BigQueryCelladeMagic.
Project: gcp-ai-ml-dev
Location: None
Dataset: None
Table: None
Max results: 10
Maximum bytes billed: 10000000000
Use BigQuery Storage API: True
Use REST API: True
Progress bar type: tqdm_notebook
Destination table: gcp-ai-ml-dev.temp_dataset.temp_table
Destination table TTL: 86400 seconds
Dry run: True
Cache enabled: False
Extensions: False
Query output: Variable 'df' will be assigned to it.
Query:
```
In [30]: %%bigquery --project=$GOOGLE_CLOUD_PROJECT
```
SELECT
```
	table_id	dataset_id	dataset_project_id	kind	type	time_partitioning
0	thesaurusconcepts_20250424_224000_000000_000000	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
1	thesaurusconcepts_20250424_224000_000000_000001	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
2	thesaurusconcepts_20250424_224000_000000_000002	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
3	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
4	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
5	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
6	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
7	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
8	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None
9	temp_table_20250424_174700	gcp-ai-ml-dev	temp_dataset	bigquery#table	TABLE	None

In [31]: %matplotlib inline
```
Using matplotlib backend: inline
```
In [32]: %pandas_info
```
Pandas 2.2.2, numpy 1.26.4, pyarrow 17.0.0, scipy 1.13.1, pytz 2025.2, dateutil 2.9.0.post0, pip 25.1.1, Cython 3.0.11, sqlalchemy 2.0.41, scipy.ext.numba 0.59.1, jinja2 3.1.6, bottleneck 1.4.2, numexpr 2.10.2, matplotlib 3.10.0, tzdata 2025.2, xlsxwriter 3.2.9, fastparquet 2024.11.0, fsspec 2025.3.0, gcsfs 2025.3.0, pandas_gbq 0.29.1, pyarrow_hotfix 0.6
```
Out[32]:
```
None
```

In 1:
Out[1]:

In 2:
Out[2]:

In 3:
Out[3]:

In 4:
Out[4]:

In 5:
Out[5]:

In 6:
Out[6]: