```
!pip install -q google-cloud-bigquery google-cloud-bigquery-storage db-dtypes pandas pyarrow
import pandas as pd
import numpy as np
import altair as alt
from google.cloud import bigquery
from google.colab import auth
from google.cloud.bigquery import magics
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import warnings
warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarning)
```

```
auth.authenticate_user()
print('Authenticated')
project_id = 'elated-life-455020-m0' #@param {type: "string"}
# Set the default project id for %bigquery magic
magics.context.project = project_id
client = bigquery.Client(project=project_id)
```

project_id: " elated-life-455020-m0 "

```
Authenticated
```

```
%load_ext google.cloud.bigquery
```

```
/usr/local/lib/python3.11/dist-packages/google/cloud/bigquery/__init__.py:237: FutureWarning: %load_ext google.cloud.big
  warnings.warn(
```

MACHINE EVENTS TABLE DATA -

```
%%bigquery machine_events_data
SELECT * FROM `google.com:google-cluster-data`.clusterdata_2019_a.machine_events
```

Job ID 0a509c86-e227-4c79-861c-c92f6749f5e7 successfully executed: 100%

Downloading: 100%

MACHINE EVENTS TABLE INSIGHTS -

```
%%bigquery machine_events_insights
SELECT capacity.cpus AS cpu_cap, capacity.memory AS memory_cap, COUNT(DISTINCT machine_id) AS num_machines
FROM `google.com:google-cluster-data`.clusterdata_2019_a.machine_events GROUP BY 1,2
```

Job ID 9a8bdf71-10c6-4c3d-b897-153829dc896b successfully executed: 100%

Downloading: 100%

COLLECTION EVENTS TABLE DATA -

```
%%bigquery collection_events_data
SELECT * FROM `google.com:google-cluster-data`.clusterdata_2019_a.collection_events
```

Job ID 6bd87bef-a14c-4ae3-878f-012c3399a601 successfully executed: 100%

Downloading: 100%

COLLECTION EVENTS TABLE INSIGHTS -

```
%%bigquery collection_events_insights
SELECT COUNT(DISTINCT collection_id) AS collections FROM
`google.com:google-cluster-data`.clusterdata_2019_a.collection_events
```

Job ID bfd24a87-2489-42a5-b0c6-7996ea2d25ed successfully executed: 100%

Downloading: 100%

INSTANCE EVENTS TABLE DATA -

```
%%bigquery instance_events_data
SELECT * FROM `google.com:google-cluster-data`.clusterdata_2019_a.instance_events LIMIT 10000000
```

Job ID e672d95d-fbd8-4874-963f-4240099864d0 successfully executed: 100%

Downloading: 100%

MACHINE ATTRIBUTES TABLE DATA -

```
%%bigquery machine_attributes_data
SELECT * FROM `google.com:google-cluster-data`.clusterdata_2019_a.machine_attributes
```

Job ID 25973c01-52c2-4bad-8882-57449130a933 successfully executed: 100%

Downloading: 100%

INSTANCE USAGE DATA -

```
%%bigquery instance_usage_data
SELECT * FROM `google.com:google-cluster-data`.clusterdata_2019_a.instance_usage LIMIT 10000000
```

Job ID 9c98ad3a-69d5-4bbb-894b-5e9929f85201 successfully executed: 100%

Downloading: 100%

DataFrames -

machine_events_data, machine_events_insights, collection_events_data, collection_events_insights, instance_events_data, machine_attributes_data

```
machine_events_data = machine_events_data.dropna(subset=['machine_id', 'capacity'])
# # machine_events_data['machine_id'].value_counts()[lambda x: x > 1]
machine_events_data['cpu_capacity'] = machine_events_data['capacity'].apply(lambda x: x['cpus'])
machine_events_data['mem_capacity'] = machine_events_data['capacity'].apply(lambda x: x['memory'])
machine_events_data = machine_events_data.drop(columns=['switch_id', 'platform_id', 'missing_data_reason', 'capacity'], errc
machine_events_data = machine_events_data[(machine_events_data['cpu_capacity'] > 0) & (machine_events_data['mem_capacity'] >
machine_events_data
```

	time	machine_id	type	cpu_capacity	mem_capacity
459	0	92126449465	1	1.000000	0.500000
460	1115701818978	92126449465	1	1.000000	0.500000
461	0	92065249319	1	1.000000	0.500000
462	1040224094227	92065249319	1	1.000000	0.500000
463	0	92046587157	1	1.000000	0.500000
...	...	...	...	...	...
46214	2294902521428	398063445417	3	0.591797	0.333496
46215	2374544210585	398963927990	3	0.591797	0.333496
46216	1689220940652	385613830661	3	0.591797	0.333496
46217	1963973255182	394444272211	3	0.591797	0.333496
46218	2284387499406	398044535458	3	0.591797	0.333496

45760 rows × 5 columns

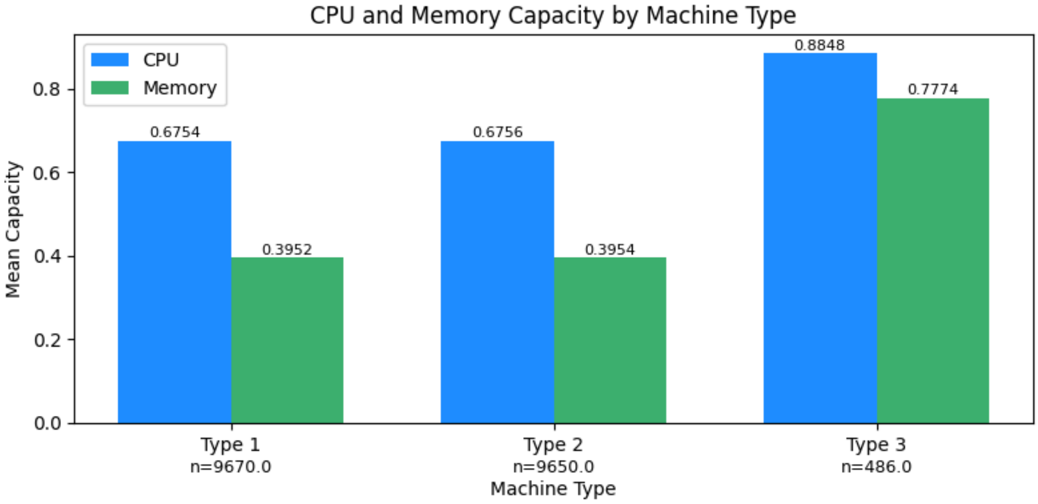
Next steps: Generate code with machine_events_data | View recommended plots | New interactive sheet

```
machine_means = machine_events_data.groupby(['machine_id', 'type']).agg({'cpu_capacity': 'mean','mem_capacity': 'mean'}).res
capacity_by_type = machine_means.groupby('type').agg({'cpu_capacity': 'mean','mem_capacity': 'mean'}).reset_index()
machine_counts = machine_means['type'].value_counts().sort_index()
capacity_by_type['count'] = capacity_by_type['type'].map(machine_counts)
```

```
print("Mean capacities by machine type:")
print(capacity_by_type)
```

```
Mean capacities by machine type:
   type  cpu_capacity  mem_capacity  count
0     1      0.675413      0.395243   9670
1     2      0.675600      0.395387   9650
2     3      0.884770      0.777398    486
```

```
fig, ax = plt.subplots(figsize=(8, 4))
bar_width = 0.35
x = np.arange(len(capacity_by_type))
cpu_bars = ax.bar(x - bar_width/2, capacity_by_type['cpu_capacity'],  bar_width, label='CPU', color='dodgerblue')
mem_bars = ax.bar(x + bar_width/2, capacity_by_type['mem_capacity'], bar_width, label='Memory', color='mediumseagreen')

ax.set_xlabel('Machine Type', labelpad=14)
ax.set_ylabel('Mean Capacity')
ax.set_title('CPU and Memory Capacity by Machine Type')
ax.set_xticks(x)
ax.set_xticklabels([f'Type {t}' for t in capacity_by_type['type']])
ax.legend(loc='upper left')

for i in range(len(capacity_by_type)):
    counts = capacity_by_type.iloc[i]['count']
    ax.text(i, -0.12, f'n={counts}', ha='center', fontsize=9)

for bar in cpu_bars:
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.008, f'{bar.get_height():.4f}', ha='center', fontsize=8)

for bar in mem_bars:
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.008, f'{bar.get_height():.4f}', ha='center', fontsize=8)

plt.tight_layout()
plt.show()
```




```
machine_data = machine_events_data.drop_duplicates(subset=['machine_id'])
machine_data.dropna(inplace=True)
machine_data
```

	time	machine_id	type	cpu_capacity	mem_capacity
459	0	92126449465	1	1.000000	0.500000
461	0	92065249319	1	1.000000	0.500000
463	0	92046587157	1	1.000000	0.500000
465	0	92048363107	1	1.000000	0.500000
467	0	92035409124	1	1.000000	0.500000
...	...	...	...	...	...
46208	2545554299697	399833620242	3	0.591797	0.333496
46214	2294902521428	398063445417	3	0.591797	0.333496
46215	2374544210585	398963927990	3	0.591797	0.333496
46217	1963973255182	394444272211	3	0.591797	0.333496
46218	2284387499406	398044535458	3	0.591797	0.333496

10001 rows × 5 columns

Next steps: Generate code with `machine_data` View recommended plots New interactive sheet

```
instance_events_data.head()
```

	time	type	collection_id	scheduling_class	missing_type	collection_type	priority	alloc_collection_id	inst
0	443298738600	6	330587180564	2	<NA>	1	101	0	
1	261559863676	6	330587183065	2	<NA>	1	101	0	
2	2653416634575	10	330587135936	2	<NA>	1	101	0	
3	258894487488	3	39516997747	2	<NA>	0	0	0	
4	39753013568	2	39516997747	2	<NA>	0	0	0	

```
instance_e_data = instance_events_data.drop(columns=['missing_type', 'alloc_instance_index', 'constraint'])
instance_e_data
```

	time	type	collection_id	scheduling_class	collection_type	priority	alloc_collection_id	instance_ind
0	443298738600	6	330587180564	2	1	101	0	1
1	261559863676	6	330587183065	2	1	101	0	7
2	2653416634575	10	330587135936	2	1	101	0	3
3	258894487488	3	39516997747	2	0	0	0	15
4	39753013568	2	39516997747	2	0	0	0	36
...	...	...	...	...	...	...	...	
9999995	1201558986210	0	330587209928	2	0	360	330587118384	3
9999996	2259436429051	0	330587209928	2	0	360	330587118384	1
9999997	2424050564914	4	330587209928	2	0	360	330587118384	4
9999998	2017038004484	0	330587209928	2	0	360	330587118384	7
9999999	89917935773	0	330587209928	2	0	360	330587118384	18

10000000 rows × 10 columns

```
instance_data = pd.merge(instance_usage_data, instance_e_data, on=['instance_index', 'collection_id', 'machine_id'], how='ir
instance_data = instance_data.dropna(subset=['resource_request'])
instance_data
```

	start_time	end_time	collection_id	instance_index	machine_id	alloc_collection_id_x	alloc_instance_inde
0	2399100000000	2399400000000	6779010793	2	62183393288	0	-
1	1158900000000	1159200000000	381666622735	10	1579760836	0	-
2	2406300000000	2406600000000	220585809472	38	71877861914	220585668772	3
3	905100000000	905400000000	278244174157	18	3098304099	278244173892	1
4	905100000000	905400000000	278244174157	18	3098304099	278244173892	1
...	...	...	...	...	...	...	.
861147	1209883000000	1209886000000	383065953964	1880	77443022129	0	-
861148	1791900000000	1792200000000	95393003654	134	10880737648	0	-
861149	2583300000000	2583600000000	399893024462	2955	71877557146	0	-
861150	1676400000000	1676700000000	376385662036	284	375997349832	0	-
861151	1583918000000	1583919000000	384244482746	46	290229834696	0	-

861136 rows × 25 columns

```
instance_data['cpus_util'] = instance_data['average_usage'].apply(lambda x: x['cpus'])
instance_data['mem_util'] = instance_data['average_usage'].apply(lambda x: x['memory'])
instance_data['cpus_req'] = instance_data['resource_request'].apply(lambda x: x['cpus'])
instance_data['mem_req'] = instance_data['resource_request'].apply(lambda x: x['memory'])

instance_data['cpus_max_util'] = instance_data['maximum_usage'].apply(lambda x: x['cpus'])
instance_data['mem_max_util'] = instance_data['maximum_usage'].apply(lambda x: x['memory'])
```

```
instance_data['start_hour'] = ((instance_data['start_time'] / (1000000)) // 3600).astype(int)
instance_data['end_hour'] = ((instance_data['end_time'] / (1000000)) // 3600).astype(int)
instance_data = instance_data.sort_values(by=['start_hour', 'end_hour'])
instance_data = instance_data.dropna(subset=['start_hour', 'end_hour'])
```

```
instance_data['cpus_util_perc'] = (instance_data['cpus_util'] * 100) / instance_data['cpus_req']
instance_data['mem_util_perc'] = (instance_data['mem_util'] * 100) / instance_data['mem_req']
instance_data = instance_data[(instance_data['cpus_util_perc'] <= 100) & (instance_data['mem_util_perc'] <= 100)]

instance_data['cpus_max_util_perc'] = (instance_data['cpus_max_util'] * 100) / instance_data['cpus_req']
instance_data['mem_max_util_perc'] = (instance_data['mem_max_util'] * 100) / instance_data['mem_req']
instance_data = instance_data[(instance_data['cpus_max_util_perc'] <= 100) & (instance_data['mem_max_util_perc'] <= 100)]
instance_data
```

	start_time	end_time	collection_id	instance_index	machine_id	alloc_collection_id_x	alloc_instance_inde
10992	2100000000	2330000000	226455519451	484	198555643071	0	-
23528	300000000	600000000	215354841713	2	22338307	0	-
29237	600000000	900000000	244773171840	1109	1128087483	244773164927	110
33397	2987000000	2988000000	104894292360	0	20935939	104894291782	
37634	600000000	900000000	220585838132	518	71880674234	220585668772	51
...	...	...	...	...	...	...	
600197	2678700000000	2679000000000	4982357443	1775	33687330218	0	-
657307	2678400000000	2678700000000	128325471862	10	1579921599	0	-
698895	2678700000000	2679000000000	399595436096	3836	1375623189	0	-
721199	2678700000000	2679000000000	4982357443	2172	70536604388	0	-
778979	2678656000000	2678660000000	399637245522	959	92140157512	0	-

334067 rows × 37 columns

```
data_to_plot = instance_data[["start_time", "end_time", "collection_id", "machine_id", "type", "scheduling_class",
                              "priority", "cpus_util_perc", "mem_util_perc", "cpus_max_util_perc", "mem_max_util_perc"]]
data_to_plot.head()
```

	start_time	end_time	collection_id	machine_id	type	scheduling_class	priority	cpus_util_perc	mem_util_perc
10992	2100000000	2330000000	226455519451	198555643071	2	2	200	5.708092	60.644531
23528	300000000	600000000	215354841713	22338307	2	1	200	6.961634	1.644036
29237	600000000	900000000	244773171840	1128087483	7	3	200	39.500942	50.132802
33397	2987000000	2988000000	104894292360	20935939	3	3	200	0.000000	0.000000
37634	600000000	900000000	220585838132	71880674234	0	2	205	0.713554	4.707792

```
instance_data.head()
```

	start_time	end_time	collection_id	instance_index	machine_id	alloc_collection_id_x	alloc_instance_index	col
10992	2100000000	2330000000	226455519451	484	198555643071	0	-1	
23528	300000000	600000000	215354841713	2	22338307	0	-1	
29237	600000000	900000000	244773171840	1109	1128087483	244773164927	1109	
33397	2987000000	2988000000	104894292360	0	20935939	104894291782	0	
37634	600000000	900000000	220585838132	518	71880674234	220585668772	518	

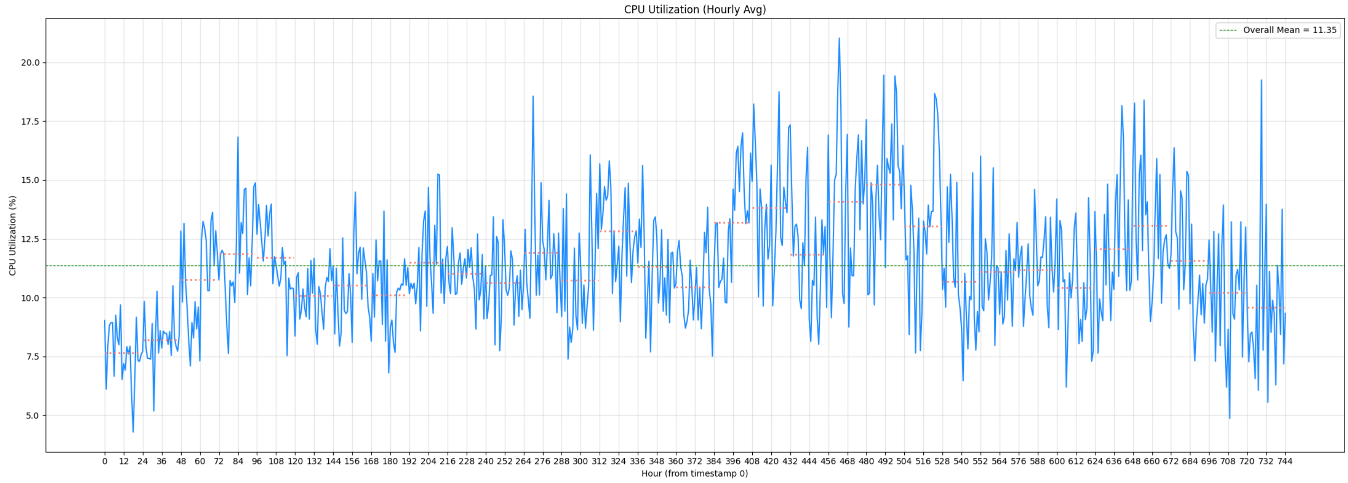
5 rows × 37 columns

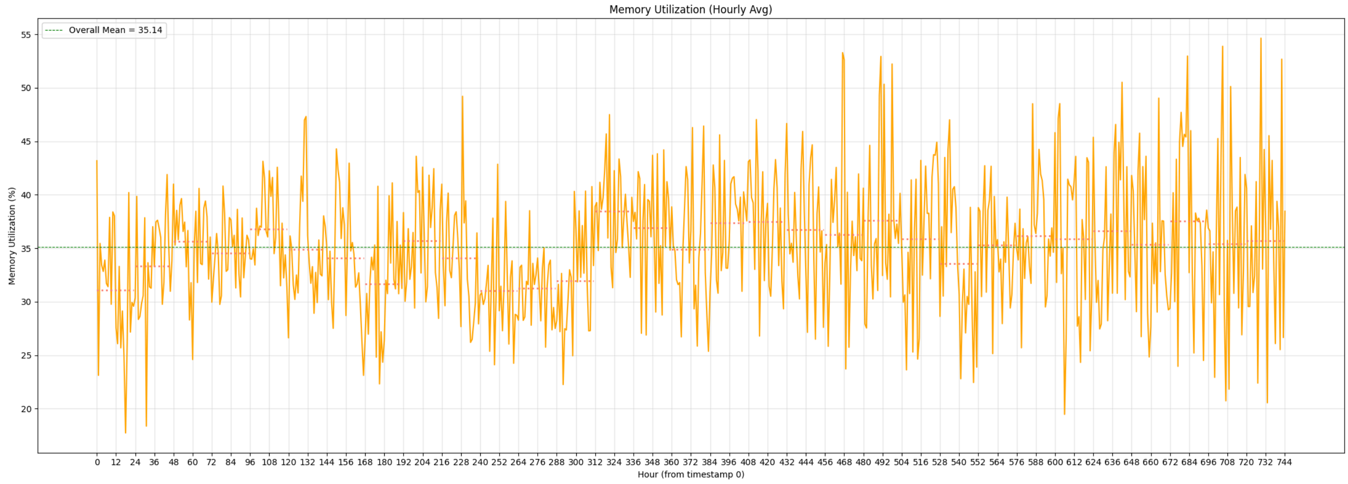
```
data_timeseries_util_plot = instance_data[['start_hour', 'end_hour', 'cpus_util_perc', 'mem_util_perc', 'cpus_max_util_perc'
data_timeseries_util_plot = (data_timeseries_util_plot.assign(hour=lambda df: df.apply(
    lambda row: range(int(row['start_hour']), int(row['end_hour']) + 1), axis=1)).explode('hour'))

data_timeseries_util_plot
```

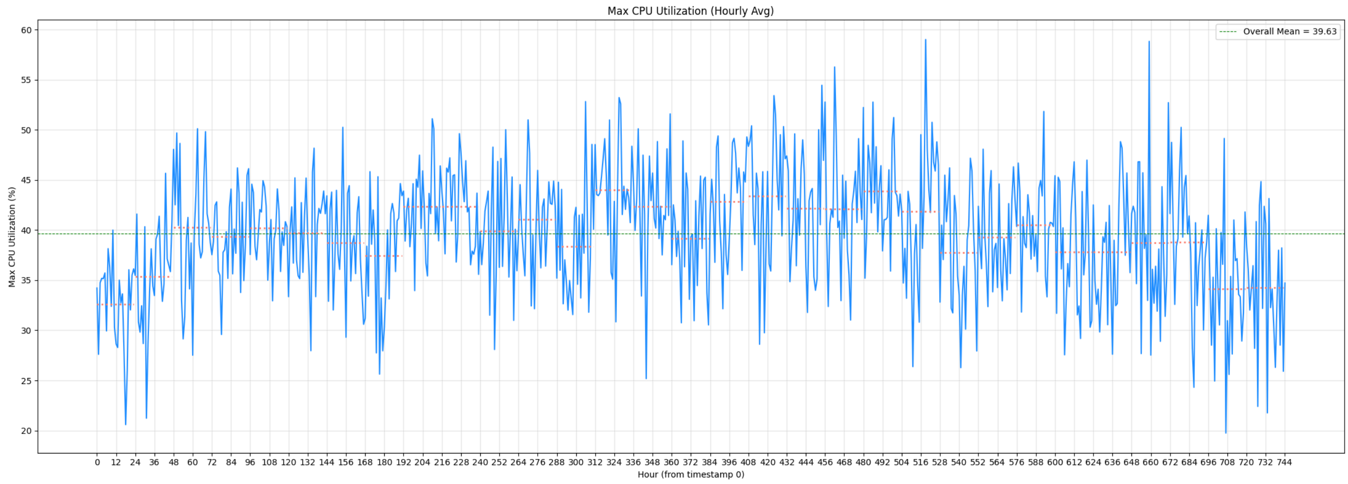
	start_hour	end_hour	cpus_util_perc	mem_util_perc	cpus_max_util_perc	mem_max_util_perc	hour
10992	0	0	5.708092	60.644531	24.566474	60.742188	0
23528	0	0	6.961634	1.644036	31.064356	1.757188	0
29237	0	0	39.500942	50.132802	86.346516	50.265604	0
33397	0	0	0.000000	0.000000	0.000000	0.000000	0
37634	0	0	0.713554	4.707792	23.126464	4.829545	0
...	...	...	...	...	...	...	...
600197	744	744	16.270860	30.804598	65.725289	30.862069	744
657307	744	744	2.095104	50.160772	17.867232	50.482315	744
698895	744	744	18.321300	29.548763	80.505415	30.131004	744
721199	744	744	21.405648	32.126437	53.979461	32.183908	744
778979	744	744	0.454939	0.065852	1.841421	0.077826	744

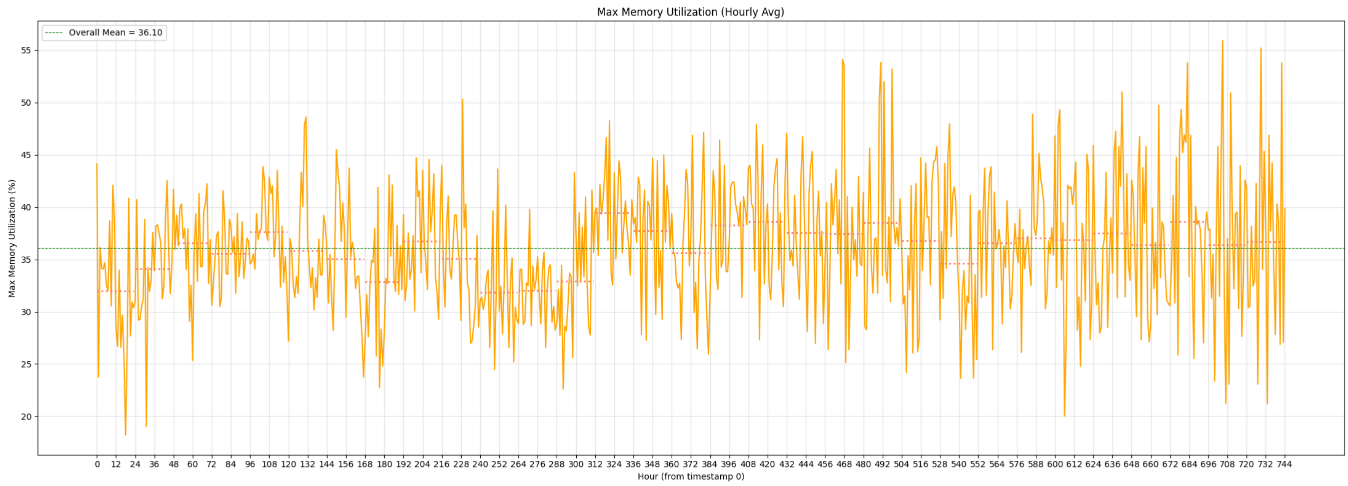
351620 rows × 7 columns

```
# Calculating hourly averages
hourly_avg = data_timeseries_util_plot.groupby('hour')[['cpus_util_perc', 'mem_util_perc', 'cpus_max_util_perc', 'mem_max_ut

def plot_with_24h_means(data, col_name, color, ylabel, title):
    plt.figure(figsize=(22, 8))
    plt.plot(data['hour'], data[col_name], color=color, linewidth=1.5)
    overall_mean = data[col_name].mean()
    plt.axhline(y=overall_mean, color='green', linestyle='--', linewidth=0.8, label=f'Overall Mean = {overall_mean:.2f}')

    max_hour = data['hour'].max()
    for start in range(0, max_hour + 1, 24):
        end = start + 24
        block = data[(data['hour'] >= start) & (data['hour'] < end)]
        if not block.empty:
            block_mean = block[col_name].mean()
            plt.hlines(y=block_mean, xmin=start, xmax=min(end - 1, max_hour), colors='salmon', linestyles='dotted', linewidt

    plt.title(title)
    plt.xlabel('Hour (from timestamp 0)')
    plt.ylabel(ylabel)
    plt.xticks(ticks=np.arange(0, max_hour + 1, 12))
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.savefig(f'Google_{col_name}_plot')
    plt.show()

# Plotting CPU Utilization
plot_with_24h_means(data=hourly_avg, col_name='cpus_util_perc', color='dodgerblue', ylabel='CPU Utilization (%)', title='CPL

# Plotting Memory Utilization
plot_with_24h_means(data=hourly_avg, col_name='mem_util_perc', color='orange', ylabel='Memory Utilization (%)', title='Memor

# Plotting Max CPU Utilization
plot_with_24h_means(data=hourly_avg, col_name='cpus_max_util_perc', color='dodgerblue', ylabel='Max CPU Utilization (%)', ti

# Plotting Max Memory Utilization
plot_with_24h_means(data=hourly_avg, col_name='mem_max_util_perc', color='orange', ylabel='Max Memory Utilization (%)', titl
```


CPU Utilization (Hourly Avg)
Overall Mean = 11.35
CPU Utilization (%)
Hour (from timestamp 0)


Memory Utilization (Hourly Avg)
Overall Mean = 35.14
Memory Utilization (%)
Hour (from timestamp 0)


Max CPU Utilization (Hourly Avg)
Overall Mean = 39.63
Max CPU Utilization (%)
Hour (from timestamp 0)


Max Memory Utilization (Hourly Avg)
Overall Mean = 36.10
Max Memory Utilization (%)
Hour (from timestamp 0)

```
!pip install xgboost
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import root_mean_squared_error
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (3.0.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.26.2.post1)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.14.1)
```

```
data_to_predict = instance_data[['start_hour', 'start_time', 'end_time', 'collection_id', 'machine_id', 'type', 'scheduling_

data_to_predict["total_time_running"] = data_to_predict['end_time'] - data_to_predict['start_time']
data_to_predict["start_hour"] = data_to_predict["start_hour"] % 24
training_data_X, testing_data_X, training_data_Y, testing_data_Y = train_test_split(data_to_predict[["start_hour", "total_ti
```

```
xgboost_regressor_model = XGBRegressor(n_estimators = 1500)
xgboost_regressor_model.fit(training_data_X, training_data_Y)
avg_cpu_prediction_values = xgboost_regressor_model.predict(testing_data_X)
print(avg_cpu_prediction_values)
root_mean_squared_error(testing_data_Y, avg_cpu_prediction_values)
```

```
[ 5.660599   1.3851129 16.359755  ...  3.7535024 13.478291  15.161158 ]
9.203477644794843
```

```
diff_in_prediction_vals_from_truth = (abs(avg_cpu_prediction_values - testing_data_Y)).to_list()
prediction_in_range_counter = 0
for curr_diff in diff_in_prediction_vals_from_truth:
    if curr_diff <= 10:
        prediction_in_range_counter = prediction_in_range_counter + 1
model_avg_cpu_pred_accuracy = prediction_in_range_counter * 100 / len(diff_in_prediction_vals_from_truth)
print("Model's Average CPU Utilization Precition accuracy is:", str(model_avg_cpu_pred_accuracy) + "%")
```

```
Model's Average CPU Utilization Precition accuracy is: 81.80675287356321%
```

```
data_to_predict = instance_data[['start_hour', 'start_time', 'end_time', 'collection_id', 'machine_id', 'type', 'scheduling_

data_to_predict["total_time_running"] = data_to_predict['end_time'] - data_to_predict['start_time']
data_to_predict["start_hour"] = data_to_predict["start_hour"] % 24
training_data_X, testing_data_X, training_data_Y, testing_data_Y = train_test_split(data_to_predict[["start_hour", "total_ti
```

```
xgboost_regressor_model = XGBRegressor(n_estimators = 1500)
xgboost_regressor_model.fit(training_data_X, training_data_Y)
avg_mem_prediction_values = xgboost_regressor_model.predict(testing_data_X)
print(avg_mem_prediction_values)
root_mean_squared_error(testing_data_Y, avg_mem_prediction_values)
```

```
[29.770962   33.72831    45.14325    ...  0.99865186 30.834314
  1.3498333 ]
15.808012696849385
```

```
diff_in_prediction_vals_from_truth = (abs(avg_mem_prediction_values - testing_data_Y)).to_list()
prediction_in_range_counter = 0
for curr_diff in diff_in_prediction_vals_from_truth:
    if curr_diff <= 10:
        prediction_in_range_counter = prediction_in_range_counter + 1
model_avg_mem_pred_accuracy = prediction_in_range_counter * 100 / len(diff_in_prediction_vals_from_truth)
print("Model's Average Memory Utilization Precition accuracy is:", str(model_avg_mem_pred_accuracy) + "%")
```

```
Model's Average Memory Utilization Precition accuracy is: 67.25933908045977%
```