```
"""
Chlorhexidine Survival Analysis

Blocks:
  1) Imports & config
  2) Helpers
  3) Load data
  4) Build duration & event columns (preprocessing)
  5) Kaplan-Meier plotting (overall + by arm)
  6) Log-rank test
  7) CoxPH model fit, summary, forest, partial effects
  8) PH assumption check + INLINE plotting of PH assumptions
  9) Main loop iterating analysis_vars
 10) wrapup
"""
```

Imports & Configuration

```
# ----------------------------
# 1) Imports & config
# ----------------------------

# In a Google Colab cell, we must first install the lifelines library
!pip install lifelines

# Import the necessary libraries

import os
import gc
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from lifelines import KaplanMeierFitter


from lifelines import KaplanMeierFitter, CoxPHFitter
from lifelines.statistics import logrank_test

OUTPUT_INFO = "Plots displayed directly in notebook output."

DATA_PATH = "/content/Data form Chlorhexidine Trial.xlsx"

# Define analysis variations (name, prefix, max_day)
analysis_vars = [
    ('TLC', 'TLC Day', 10),
    ('Band Form', 'Band Form Day', 10),
    ('Chest X ray', 'Chest X ray Day', 8),
    ('CPIS', 'CPIS Day', 10),
    ('ABG', 'ABG Day', 10),
    # NOTE: original had 'Cluture Day' — consider correcting to 'Culture Day' in  data
    ('Culture', 'Cluture Day', 10),
    ('O S Microbial Load', 'O S Microbial Load Day', 10),
    ('Ulcer', 'Ulcer Day', 10),
]
```

```
Start coding or generate with AI.
```

Helper Functions (Duration & Column Pickers)

```
# ----------------------------
# 2) Helpers
# ----------------------------
def calculate_duration(row, duration_columns):
    """Return the last day index (1..n) where any value exists among duration_columns.
        duration_columns must be an existing-column list (valid_duration_cols)."""
    duration = 0
    for day_idx, col in enumerate(duration_columns, start=1):
        if pd.notna(row.get(col, np.nan)):
            duration = day_idx
    return duration

def pick_column(candidates, df):
    """Return the first candidate name that appears in DataFrame df.columns, else None."""
    for name in candidates:
        if name in df.columns:
            return name
    return None

def safe_fit_cox(df_cox_input, duration_col='Duration', event_col='Event'):
    """Fit CoxPH and return fitted model or raise for caller to handle."""
    cph = CoxPHFitter()
    cph.fit(df_cox_input, duration_col=duration_col, event_col=event_col)
    return cph
```

```
Start coding or generate with AI.
```

Data Loading

```
# ----------------------------
# 3) Load data
# ----------------------------
try:
    df = pd.read_excel("/content/Data form Chlorhexidine Trial.xlsx")
except FileNotFoundError:
    raise SystemExit(f"Error: Excel file not found at {DATA_PATH}")
except Exception as e:
    raise SystemExit(f"Error reading file: {e}")
```

```
Start coding or generate with AI.
```

Preprocessing Function

```
# ----------------------------
# 4) Per-analysis preprocessing function
# ----------------------------
def prepare_analysis_df(df, prefix, max_day, name):
    """Return a cleaned DataFrame with Duration and Event columns for this analysis."""
    # Build candidate duration column list
    if name == 'ABG':
        # ABG uses a mixed naming convention
        duration_cols = [f'{prefix} 1'] + [f'Day {i}' for i in range(2, max_day + 1)]
    else:
        duration_cols = [f'{prefix} {i}' for i in range(1, max_day + 1)]

    # Keep only those that actually exist in the DataFrame
    valid_duration_cols = [col for col in duration_cols if col in df.columns]
    if not valid_duration_cols:
        return None, duration_cols

    # compute Duration and Event
    df_work = df.copy()
    df_work['Duration'] = df_work.apply(lambda r: calculate_duration(r, valid_duration_cols), axis=1)
    df_work['Event'] = np.where(df_work.get('Outcome of current episode 2') == 'Death', 1, 0)

    # Select columns needed for analysis and drop rows with missing Duration or Event
    selected = ['Trial Arm', 'Age', 'APACHE II Score', 'Duration', 'Event', 'Gender']
    available = [c for c in selected if c in df_work.columns]
    df_analysis = df_work[available].dropna(subset=['Duration', 'Event'])

    return df_analysis, valid_duration_cols

print("Preprocessing function defined.")
```

```
Start coding or generate with AI.
```

Kaplan-Meier Plotting Helpers

```
# ----------------------------
# 5) KM plotting helpers
# ----------------------------
def plot_km_overall(df_analysis, name):
    kmf = KaplanMeierFitter()
    fig, ax = plt.subplots(figsize=(10, 6))
    kmf.fit(df_analysis['Duration'], event_observed=df_analysis['Event'], label='Overall Survival')
    kmf.plot_survival_function(ci_show=True, ax=ax)
    ax.set_title(f'[{name} Plot 1] Overall Kaplan-Meier Survival Curve')
    plt.tight_layout()
    plt.show() # Display plot
```

```
        plt.close(fig); del fig; gc.collect()

    def plot_km_by_arm(df_analysis, name):
        kmf = KaplanMeierFitter()
        fig, ax = plt.subplots(figsize=(10, 6))
        groups = df_analysis['Trial Arm'].dropna().unique()
        if len(groups) == 0:
            plt.close(fig)
            return
        for arm in groups:
            subset = df_analysis[df_analysis['Trial Arm'] == arm]
            if len(subset) == 0:
                continue
            kmf.fit(subset['Duration'], event_observed=subset['Event'], label=str(arm))
            kmf.plot_survival_function(ci_show=False, ax=ax)
        ax.set_title(f'[{name} Plot 2] Kaplan-Meier Survival Curves by Trial Arm')
        ax.legend(title='Trial Arm')
        plt.tight_layout()
        plt.show() # Display plot
        plt.close(fig); del fig; gc.collect()

    print("KM plotting functions defined.")
```

```
    Start coding or generate with AI.
```

Log-Rank Test Helper

```
    # ----------------------------
    # 6) Log-rank helper
    # ----------------------------
    def do_logrank(df_analysis):
        unique_arms = df_analysis['Trial Arm'].dropna().unique()
        if len(unique_arms) == 2:
            arm1 = df_analysis[df_analysis['Trial Arm'] == unique_arms[0]]
            arm2 = df_analysis[df_analysis['Trial Arm'] == unique_arms[1]]
            if len(arm1) > 0 and len(arm2) > 0:
                lr = logrank_test(arm1['Duration'], arm2['Duration'],
                                  event_observed_A=arm1['Event'], event_observed_B=arm2['Event'])
                return unique_arms[0], unique_arms[1], lr.p_value
        return None

    print("Log-rank function defined.")
```

```
    Start coding or generate with AI.
```

CoxPH Preparation and Summary Functions

```
    # ----------------------------
    # 7) CoxPH Prep, Summary, and Forest Plot
    # ----------------------------
    def prepare_cox_inputs(df_analysis):
        """Return df_cox_input and covariates list for CoxPH fitting."""
        df_cox = df_analysis.copy()

        # Impute APACHE II Score mean if present
        if 'APACHE II Score' in df_cox.columns:
            if df_cox['APACHE II Score'].isnull().any():
                df_cox['APACHE II Score'] = df_cox['APACHE II Score'].fillna(df_cox['APACHE II Score'].mean())

        # Binary encodings
        df_cox['Trial Arm_Group 2'] = np.where(df_cox.get('Trial Arm') == 'Group 2', 1, 0)
        df_cox['Gender_Male'] = np.where(df_cox.get('Gender') == 'Male', 1, 0)

        # Define covariates in the order you want plotted
        covariates = []
        for c in ['Trial Arm_Group 2', 'Age', 'APACHE II Score', 'Gender_Male']:
            if c in df_cox.columns:
                covariates.append(c)

        df_cox_input = df_cox[['Duration', 'Event'] + covariates].dropna()
        return df_cox_input, covariates

    def show_cox_plots_and_summary(cph, covariates, name):
        # 1) Print summary selected columns
        summary = cph.summary
        ci_lower_col = pick_column(['exp(coef) lower 95%', 'exp(coef) lower 95% CI', 'lower 95%'], summary)
        ci_upper_col = pick_column(['exp(coef) upper 95%', 'exp(coef) upper 95% CI', 'upper 95%'], summary)
        exp_coef_col = 'exp(coef)'
        p_col = pick_column(['p', 'p-value', 'p-value (two-sided)'], summary)
        select_cols = [col for col in [exp_coef_col, ci_lower_col, ci_upper_col, p_col] if col]
        print(f"\nCox PH Model Summary for {name} (selected columns):")
        if select_cols:
            print(summary[select_cols].to_markdown(floatfmt=".3f"))

        # 2) Forest/HR plot
        fig, ax = plt.subplots(figsize=(10, 6))
        try:
            cph.plot(ax=ax)
            ax.set_title(f'[{name} Plot 3] Forest Plot of Hazard Ratios (Adjusted)')
            ax.grid(axis='x', linestyle='--')
            plt.tight_layout()
            plt.show() # Display plot
        finally:
            plt.close(fig); del fig; gc.collect()

    print("Cox PH preparation and summary functions defined.")
```

```
Start coding or generate with AI.
```

CoxPH Partial Effects Plotting

```
# ----------------------------
# 7) CoxPH Partial Effects plotting helpers (continued)
# ----------------------------
def show_partial_effects(cph, df_cox_input, name, covariates):
    # Age effects
    if 'Age' in covariates:
        fig, ax = plt.subplots(figsize=(10, 6))
        age_min, age_max = int(df_cox_input['Age'].min()), int(df_cox_input['Age'].max())
        if age_min == age_max:
            ages = [age_min]
        else:
            ages = list(range(age_min, age_max + 1, max(1, (age_max - age_min)//2)))
        cph.plot_partial_effects_on_outcome(covariates='Age', values=ages, ax=ax)
        ax.set_title(f"[{name} Plot 4] Adjusted Survival by Age")
        ax.grid(True)
        plt.tight_layout()
        plt.show() # Display plot
        plt.close(fig); del fig; gc.collect()

    # Trial Arm
    if 'Trial Arm_Group 2' in covariates:
        fig, ax = plt.subplots(figsize=(10, 6))
        cph.plot_partial_effects_on_outcome(covariates='Trial Arm_Group 2', values=[0, 1], ax=ax)
        ax.legend(['Group 1 (Reference)', 'Group 2'])
        ax.set_title(f"[{name} Plot 5] Adjusted Survival by Trial Arm")
        ax.grid(True)
        plt.tight_layout()
        plt.show() # Display plot
        plt.close(fig); del fig; gc.collect()

    # Gender
    if 'Gender_Male' in covariates:
        fig, ax = plt.subplots(figsize=(10, 6))
        cph.plot_partial_effects_on_outcome(covariates='Gender_Male', values=[0, 1], ax=ax)
        ax.legend(['Female (Reference)', 'Male'])
        ax.set_title(f"[{name} Plot 6] Adjusted Survival by Gender")
        ax.grid(True)
        plt.tight_layout()
        plt.show() # Display plot
        plt.close(fig); del fig; gc.collect()

print("Partial effects plotting functions defined.")
```

```
Start coding or generate with AI.
```

Proportional Hazards (PH) Assumption Check

```
# ----------------------------
# 8) PH assumption check helper
# ----------------------------
def show_ph_assumption_plots(cph, df_cox_input, name, covariates):
    print("Running PH assumption checks...")
    figs_before = set(plt.get_fignums())
    orig_figsize = plt.rcParams.get('figure.figsize', [6.4, 4.8])
    plt.rcParams['figure.figsize'] = [12, 10]
    try:
        # This will create figures
        cph.check_assumptions(df_cox_input, show_plots=True, p_value_threshold=0.05)

        figs_after = set(plt.get_fignums())
        new_figs = sorted(list(figs_after - figs_before))

        # Manually show and close each figure created
        for i, fig_id in enumerate(new_figs):
            fig = plt.figure(fig_id)
            covariate_name = covariates[i] if i < len(covariates) else f"covariate_{i}"
            fig.tight_layout(rect=[0, 0.05, 1, 0.95])
            fig.suptitle(f'[{name} Plot {7 + i}] PH Check: {covariate_name}', y=0.99, fontsize=14)

            plt.show() # Display plot
            plt.close(fig); del fig; gc.collect()

    finally:
        plt.rcParams['figure.figsize'] = orig_figsize

print("PH assumption checking function defined.")
```

Start coding or generate with AI.

Main Execution Loop

```
# ----------------------------
# 9) Main loop & 10) Wrap up
# ----------------------------
print(f"Starting analysis for {len(analysis_vars)} different outcomes. {OUTPUT_INFO}")

for name, prefix, max_day in analysis_vars:
    print("\n" + "="*70)
    print(f"Running Survival Analysis for Duration based on: {name} (up to Day {max_day})")
    print("="*70)

    # Prepare data (Cell 4)
    df_analysis, used_duration_cols = prepare_analysis_df(df, prefix, max_day, name)
    if df_analysis is None or len(df_analysis) == 0:
        print(f"Skipping {name}: no valid patients after preparation. Tried: {used_duration_cols}")
        continue

    print(f"Total Patients for Survival Analysis: {len(df_analysis)}, Total Events (Deaths): {int(df_analysis['Event'].sum()

    # KM plots (Cell 5)
    plot_km_overall(df_analysis, name)
    plot_km_by_arm(df_analysis, name)

    # Log-rank (Cell 6)
    lr_res = do_logrank(df_analysis)
    if lr_res:
        a1, a2, pval = lr_res
        print(f"Log-rank test between {a1} and {a2}: p-value = {pval:.4f}")

    # Cox PH preparation (Cell 7)
    df_cox_input, covariates = prepare_cox_inputs(df_analysis)
    if df_cox_input is None or len(df_cox_input) < 3:
        print("Skipping Cox PH: not enough data after covariate preparation.")
        continue

    # Fit CoxPH
    try:
        cph = safe_fit_cox(df_cox_input)
    except Exception as e:
        print(f"Failed to fit CoxPH for {name}: {e}")
        continue

    # Show Cox outputs (Cells 7 & 8)
    show_cox_plots_and_summary(cph, covariates, name)
    show_partial_effects(cph, df_cox_input, name, covariates)

    # PH assumption checks (Cell 9)
    try:
        show_ph_assumption_plots(cph, df_cox_input, name, covariates)
    except Exception as e:
        print(f"PH assumption plotting failed for {name}: {e}")

    gc.collect()

print("\n" + "="*70)
```

```
    print("Analysis complete. All analysis variables processed and results displayed inline.")
    print("="*70)
```