Mubeen Khan

Greater Chicago Area

mubeen@mubeen.co 7136141726

linkedin.com/in/mubeen-khan

Summary

Creative and proactive management and strategy professional with experience leveraging his background in healthcare, coding, teaching, and scientific research to design and improve automated systems, streamline departmental methods, mentor and coach employees, and preempt future problems. Proven drive and ability to develop and implement creative visions using strong planning skills, automation, statistics, and soft-skills to support service launches, overhauls, and employee productivity, positioning departments for success and future expansion.

Key skills:

• Strategic planning
• Project management
• Interdepartmental mediation
• Corporate compliance
• Proactive reporting
• Automation
• Statistical/data analysis
• R/Python/SQL/C/Excel/JS
• Digital infrastructure
• Machine learning (RNN)
• Operations coordination
• Workforce development
• Leadership/team-building
• Teaching/training
• Public speaking/presentation

Experience

Access Team Lead

Northwestern Medicine

Oct 2018 - Present (3 years 4 months +)

Recruited to function in an assistant manager role. Supervises and coaches 80+ employees connecting patients to healthcare resources in a fast-paced environment.

Key contributions:

• Filled a gap and drove the transition to a work-from-home model and constructing a comprehensive, digital operations-framework on Microsoft Teams/OneNote, reducing employee-question response times by 80%, improving inbound capacity by 20%, and cutting training time by 33%.

• Took initiative to build an artificial intelligence package in R/Python to automate data collection and interpretation, quality assurance, and analysis implementation, saving 40 man-hours per week

• Stepped outside of his role to conceptualize and integrate automation/data analysis tools to measure department progress and modernize management processes (i.e. employee attendance, performance, corporate compliance). Trains senior leaders accordingly

• Led and conducted an audit that resulted in an overhaul of the quality assurance department.
• Coordinated 20 go-lives and 6 key-initiatives to improve department processes. Helped lead a key initiative that reduced patient wait times for appointments by 2 months

Mental Health Specialist

Garfield Park Behavioral Hospital

Jun 2018 - Oct 2018 (5 months)

Brought on to provide inpatient psychiatric support and counseling to 20 pediatric patients from underprivileged backgrounds. Conducted group therapy, CBT, nutrition, and entertainment while coordinating with physicians and nursing staff.

Key contributions:

- Provided documentation that facilitated a patient's successful entry into foster care.

Mental Health Worker

Heartland Alliance

Jun 2018 - Oct 2018 (5 months)

Hired to provide healthcare support to a challenging population of active substance users in an innovative affordable-housing project centered on harm-reduction. Provided medication support, counseling, care-coordination, and petty cash for 25 tenants.

Key contributions:

- Provided documentation that helped a tenant graduate to independent housing.

Senior Educator

Peer Health Exchange

Sep 2014 - Jun 2016 (1 year 10 months)

Expanded responsibilities from the previous Junior Educator role to include training educators, teaching advanced curricula, and supporting operations.

Key contributions:

- Prevented key operations from failing during a staffing shortage.
- Awarded Educator-of-the-Year and a Career Achievement Award in 2016.

Research Assistant

The University of Chicago

Apr 2014 - Jan 2016 (1 year 10 months)

Coded and analyzed gestural and eye contact data from infant interviews for communication research. Separately, collected and analyzed DNA from hamster brains to understand the effects of circadian rhythm interruption on immune systems.

Junior Educator

Peer Health Exchange

Sep 2012 - Jun 2014 (1 year 10 months)

Taught health topics to high-school students in underserved areas through the Chicago South side, including sexual communication, elementary advocacy skills, sexual health, mental health, and informed decision-making.

Research Assistant
MD Anderson Cancer Center
Jan 2010 - Aug 2013 (3 years 8 months)
• Performed wet-lab and statistical research on multiple myeloma, amyloidosis, graft-vs-host disease, and umbilical cord blood.
• Learned and performed care of cell lines, flow cytometry, mouse necropsy, identification of protein markers and genetic abnormalities, harvested cord blood cells, electrophoresis, etc.
• Performed significant data entry, review, and analysis with large datasets.
• Participated in weekly lab meetings and learned about the intellectual process behind study design and administration in the field of stem cell transplantation.
• Shadowed specialized oncologists and interviewed patients and family members.

Education

University of Chicago
Bachelor of Arts - BA, Psychology
2012 - 2016
Dean's List (2015-2016)

Loyola University Chicago
Graduate Student at Large
Aug 2016 - Jun 2017

Tufts University
Graduate Student at Large
Aug 2017 - May 2018

Licenses & Certifications

CPR Certification - American Heart Association

Skills

PostgreSQL • Relational Databases • R (Programming Language) • Python (Programming Language) • Data Analysis • Data Visualization • Extract, Transform, Load (ETL) • Business Intelligence (BI) • Customer Relationship Management (CRM) • Training

Honors & Awards

Dean's List
Jun 2016

Outstanding Educator - Peer Health Exchange
2016

Dedicated Educator Award - Peer Health Exchange
2015

Educator of the Month - Peer Health Exchange
May 2016

Code Samples

SQL

Dates, CTEs, and Partitions: Determining the number of users who make new purchases due to a marketing campaign.

```
-- we want to see whether a user has made a purchase of products on more
-- than one date AND that there is a date for multiple products
with purchase_dates as (
    select
        user_id,
        -- earliest date for a purchase by a user
        min(created_at) over (partition by user_id) as min_date,
        -- earliest date for a purchase by a user of a product
        min(created_at) over (
            partition by user_id, product_id
            ) as min_prd_date
    from marketing_campaign
),
worked as (
    select
        user_id,
        -- if the earliest purchase date is equal to the earliest purchase
        -- date for a product by each product, then that result doesn't
        -- matter. If they are not equal, then they purchased another produ
        -- on a different date
        case when min_date != min_prd_date then 1 else 0 end as worked
    from purchase_dates
)
select count(distinct(user_id)) from worked where worked = 1
```

Joins and Subqueries: Find the customer with the highest daily total order cost between 2019-02-01 to 2019-05-01.

```
with daily_totals as (
    select
        cust_id,
        date,
        sum(cost) as cost
    from (
        select
            cust_id,
            order_date as date,
            total_order_cost as cost
        from orders
        where order_date between '2019-02-01' and '2019-05-01'
    ) as costs
    group by costs.cust_id, date
)
select c.first_name, d.date, d.cost
from daily_totals as d
left join customers as c on d.cust_id = c.id
where d.cost in (select max(cost) from daily_totals);
```

CTE, Window Functions: Determine for which Microsoft employees, their directly previous employer was Google.

```
with lags as (
    select
        user_id,
        employer,
        case when
            lag(end_date) over(
                order by user_id, start_date
            ) = start_date then 1 else 0 end as consecutive,
        lag(employer) over(order by user_id, start_date) as prev_empl
    from linkedin_users
)
select
    count(distinct(user_id))
from lags
where consecutive = 1 and employer = 'Google' and prev_empl = 'Microsoft'
```

Sorting: Sort rental popularity in specific buckets.

```
select
    case
        when number_of_reviews = 0 then 'New'
        when number_of_reviews >= 1 and number_of_reviews <= 5 then 'Rising
        when number_of_reviews >= 6 and number_of_reviews <= 15 then 'Trend
        when number_of_reviews >= 16 and number_of_reviews <= 40 then 'Popu
        else 'Hot'
            end as popularity_rating,
    min(price) as minimum_price,
    avg(price) as average_price,
    max(price) as maximum_price
from (
    select
        price,
        room_type,
        host_since,
        zipcode,
        number_of_reviews
    from airbnb_host_searches
    group by 1,2,3,4,5
) as temp
group by 1;
```

R

Cleaning, Processing, and Reporting

A set of functions that ingests, cleans, and processes data, then outputs it as a KPI report in Excel for my manager to view.

Clean data

```
#' Removes headers and footers from Excel exports from CUIC
#'
#' @param x A dataframe
#' @return A dataframe without the CUIC header and footer
clean_report_from_cuic <- function(report_from_cuic) {
    if (grepl(
    "Generated",
    report_from_cuic[nrow(report_from_cuic)-1,
    ])[1] == TRUE) {
        footer_row_numbers <- c(
            nrow(report_from_cuic),
            nrow(report_from_cuic) - 1,
            nrow(report_from_cuic) -2
        )
        report_from_cuic_no_footer <-
            report_from_cuic[-footer_row_numbers,]
        return(report_from_cuic_no_footer)
    } else {
        return(report_from_cuic)
    }
}

#' Takes an ABA report from CUIC, removes headers and footers, renames colu
#' converts formats where necessary, and returns a dataframe
#'
#' @param x A dataframe
#' @return A dataframe without the CUIC header and footer, with columns ren
#' formats converted
clean_aba_report_from_cuic <- function(
    aba_report_from_cuic,
    new_date_type = FALSE
    ) {
    if (class(aba_report_from_cuic) == "data.frame") {
        report_from_cuic_clean <-
            clean_report_from_cuic(aba_report_from_cuic)
    } else if (class(aba_report_from_cuic) == "list") {
        report_from_cuic_clean <-
            lapply(aba_report_from_cuic,
                    clean_report_from_cuic) %>%
                bind_rows
    }
    if (new_date_type == TRUE) {
        report_from_cuic_clean <- report_from_cuic_clean %>%
            mutate(
                date_time = parse_new_cuic_date_format(DateTime),
                date = parse_new_cuic_date_format(DATE, "date")
            )
    } else {
        report_from_cuic_clean <- report_from_cuic_clean %>%
            mutate(
                date_time = convertToDateTime(DateTime),
                date = convertToDate(DATE)
            )
    }
    report_from_cuic_clean <-
        report_from_cuic_clean %>%
            mutate(precision_queue = tolower(Precision.Queue),
                    weekday = weekdays(date),
                    time = date_time %>%
                        format("%H:%M:%S") %>%
                        hms %>%
                        as_datetime,
                    hour = hour(time),
                    service_level = Service.Level,
                    hold_time = convertToDateTime(
                        Hold.Time,
                        origin = "1970-01-01"
                    ),
                    speed_of_answer_mean = convertToDateTime(
                        Avg.Speed.of.Answer,
                        origin = "1970-01-01"
                    ),
                    calls_offered = Total,
                    calls_handled = Handled,
                    calls_abandoned = Aban,
                    calls_answered = Answered,
                    handle_time_mean = convertToDateTime(
                        Avg.Handle.Time,
                        origin = "1970-01-01"
                    ),
                    longest_queued = convertToDateTime(
                        Longest.Queued,
                        origin = "1970-01-01"
                    ),
                    max_queued = Max.Queued,
                    service_level_abandoned = Service.Level.Abandon
            ) %>%
            select(precision_queue,
                    date,
                    weekday,
                    date_time,
                    time,
                    hour,
                    service_level,
                    hold_time,
                    speed_of_answer_mean,
                    calls_offered,
                    calls_handled,
                    calls_abandoned,
                    calls_answered,
                    handle_time_mean,
                    longest_queued,
                    max_queued,
                    service_level_abandoned)
    queue_names <-
        c("clsma",
          "corp",
          "derm",
          "gim",
          "intmed",
          "cim",
          "ped",
          "student",
          "travel")
    #### rename queue names to readable English
    for (queue_name in queue_names) {
        report_from_cuic_clean$precision_queue[
            grepl(queue_name, report_from_cuic_clean$precision_queue)
            ] <- toupper(queue_name)
    }
    return(report_from_cuic_clean)
}
```

Process data

```
#' Helper function that generates an ABA report summarized by queue and dat
#' including total/overall
#'
#' @param x A dataframe from CUIC, cleaned
#' @return A dataframe summarizing by queue and date, including total/overa
generate_aba_summary_by_date <- function(report_from_cuic_clean) {
    aba_summary_by_date_by_queue <-
        report_from_cuic_clean %>%
            group_by(precision_queue, date) %>%
            summarise(
                service_level_mean = mean(na.omit(service_level)),
                calls_offered_total = sum(na.omit(calls_offered)),
                calls_answered_total = sum(na.omit(calls_answered)),
                calls_abandoned_total = sum(na.omit(calls_abandoned)),
                handle_time_mean = mean(na.omit(handle_time_mean)),
                speed_of_answer_mean = mean(na.omit(speed_of_answer_mean)),
                hold_time_mean = mean(na.omit(hold_time))
            ) %>%
            ungroup() %>%
            mutate(aba = calls_abandoned_total/calls_offered_total)
    aba_summary_by_date_overall <-
        report_from_cuic_clean %>%
            group_by(date) %>%
            summarise(
                service_level_mean = mean(na.omit(service_level)),
                calls_offered_total = sum(na.omit(calls_offered)),
                calls_answered_total = sum(na.omit(calls_answered)),
                calls_abandoned_total = sum(na.omit(calls_abandoned)),
                handle_time_mean = mean(na.omit(handle_time_mean)),
                speed_of_answer_mean = mean(na.omit(speed_of_answer_mean)),
                hold_time_mean = mean(na.omit(hold_time))
            ) %>%
            ungroup() %>%
            mutate(aba = calls_abandoned_total/calls_offered_total,
                    precision_queue = "total_overall")
    aba_summary_by_date_by_queue <-
        aba_summary_by_date_by_queue[c(1,2,10,3,4,5,6,7,8,9)]
    aba_summary_by_date_overall <-
        aba_summary_by_date_overall[c(10,1,2,9,3,4,5,6,7,8)]
    aba_summary_by_date <-
        rbind(aba_summary_by_date_by_queue,
                aba_summary_by_date_overall) %>%
        na.omit
    return(aba_summary_by_date)
}
```

There are additional helper functions I will omit for brevity.

```
#' Generates an ABA report summarized by queue and a set of options by choi
#'
#' @param x A dataframe from CUIC, uncleaned
#' @param type The type of report to generate, options are "standard" (defa
#' @param queue The queue to generate for
#' (CLSMA,INTMED,GIM,CIM,STUDENT,TRAVEL,CORP,PED,DERM).
#' Use "all" to show all queues and the overall numbers (the default) and
#' "overall" to only show the overall numbers.
#' @return A dataframe
#' @example generate_aba_summary(aba_report_2021-07-01)
#' @example generate_aba_summary(aba_report_2021-07-01, type = "by_date")
#' @example generate_aba_summary(
    #' aba_report_2021-07-01,
    #' type = "by_date_and_hour",
    #' queue = "CLSMA"
    )
```

```
generate_aba_summary <- function(
    data,
    type = "standard",
    queue = "all",
    new_date_type = FALSE,
    exclude_saturday = FALSE,
    presentation_view = FALSE
    ) {
    cleaned_report <- clean_aba_report_from_cuic(
        data,
        new_date_type = new_date_type
    )
    if (exclude_saturday == TRUE) {
        cleaned_report <- cleaned_report %>% filter(weekday != "Saturday")
    } else {}
    if (type == "standard" & presentation_view == FALSE) {
        aba_summary <-
            generate_aba_summary_standard(cleaned_report)
    } else if (type == "standard" & presentation_view == TRUE) {
        aba_summary <-
            generate_aba_summary_standard(cleaned_report) %>%
                prettify_aba_standard_summary()
    } else if (type == "by_date") {
        aba_summary <-
            generate_aba_summary_by_date(cleaned_report)
    } else if (type == "by_hour") {
        aba_summary <-
            generate_aba_summary_by_hour(cleaned_report)
    } else if (type == "by_date_and_hour") {
        aba_summary <-
            generate_aba_summary_by_date_by_hour(cleaned_report)
    } else if (type == "by_weekday") {
        aba_summary <-
            generate_aba_summary_by_weekday(cleaned_report)
    } else if (type == "by_15_min") {
        aba_summary <-
            generate_aba_summary_by_15_min(cleaned_report)
    } else if (type == "by_date_and_15_min") {
        aba_summary <-
            generate_aba_summary_by_date_by_15_min(cleaned_report)
    }
    if (queue == "all") {
        summary_filtered_by_queue <-
            aba_summary
    } else {
        summary_filtered_by_queue <-
            aba_summary %>%
                filter(grepl(collapse_vector(queue), precision_queue))
    }
    return(summary_filtered_by_queue)
}
```

Report Generation

```
#' Generates and saves an Excel workbook containing all five types of ABA r
#' to the current working directory
#'
#' @param x A dataframe from CUIC, uncleaned
#' @param report_date_or_month The date of the report, or the month of the
#' (format Jun 21), defaults to the current date
#' @param report_name Filename suffix, defaults to "export"
#' @return Workbook saved and success message printed
generate_and_save_aba_summary_combined <-
    function(
        report_from_cuic,
       new_date_type = FALSE,
       report_date_or_month = Sys.Date(),
       report_name = "export"
       ) {
    summary_standard <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "standard",
            "all"
        )
    summary_by_date <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "by_date",
            "all"
        ) %>% na.omit
    summary_by_weekday <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "by_weekday",
            "all"
        ) %>% na.omit
    summary_by_hour <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "by_hour",
            "all"
        ) %>% na.omit
    summary_by_date_and_hour <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "by_date_and_hour",
            "all"
        ) %>% na.omit
    summary_by_15_min <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "by_15_min",
            "all"
        ) %>% na.omit
    summary_by_date_and_15_min <-
        generate_aba_summary(
            report_from_cuic,
            new_date_type = new_date_type,
            "by_date_and_15_min",
            "all"
        ) %>% na.omit
    report_sheet_list <-
        lst(
            summary_standard,
            summary_by_date,
            summary_by_weekday,
            summary_by_hour,
            summary_by_date_and_hour,
            summary_by_15_min,
            summary_by_date_and_15_min
        )
    write.xlsx(
        report_sheet_list,
        paste0(
            "aba_combined_report",
            "_",
            report_name,
            ".xlsx"
        ),
        overwrite = TRUE
    )
    return(paste0("Success! Data written to file."))
}
```

Plotting and Visualization Functions

Functions to automatically generate interactive plots for dashboards using R and Plotly:

```
plot_aba_summary <- function(aba_data,
                                type,
                                queue_search = NA,
                                exclude_queues = NA) {
    aba_data <- aba_data %>%
        pivot_longer(!c(Queue, Month), names_to = "variable")
    if (!is.na(queue_search[1])) {
        queue_search <- collapse_vector(queue_search)
        aba_data <- aba_data %>%
            filter(grepl(queue_search, Queue))
    } else {}
    if (!is.na(exclude_queues[1])) {
        exclude_queues <- collapse_vector(exclude_queues)
        aba_data <- aba_data %>%
            filter(!grepl(exclude_queues, Queue))
    } else {}
    if (type == "calls") {
        plot <- aba_data %>%
            filter(grepl("Calls", variable)) %>%
            ggplot(aes(x = Month, y = value)) +
                geom_path(aes(color = Queue)) +
                scale_y_continuous(
                    # breaks = function(x) {seq(0, max(x), by = 2500)},
                    labels = function(x) {round(x, digits = 0)}
                    ) +
                facet_wrap(. ~ variable, scales = "free_y")
    } else if (type == "service_level") {
        plot <- aba_data %>%
            filter(variable == "Service Level (Avg)") %>%
            ggplot(aes(x = Month, y = value)) +
                geom_path(aes(color = Queue)) +
                scale_y_continuous(
                    labels = function(x) {scales::percent(x, accuracy = 1)}
                    )
    } else if (type == "times") {
        plot <- aba_data %>%
            filter(variable == "Answer Speed" | variable == "Handle Time")
            ggplot(aes(x = Month, y = value)) +
                geom_line(aes(color = Queue)) +
                scale_y_continuous(
                    labels = function(x) {x %>%
                        seconds_to_period %>%
                        as_datetime %>%
                        format("%M:%S")},
                    breaks = function(x) {seq(0, max(x+60), by = 60)}
                    ) +
                facet_wrap(. ~ variable, scales = "free_y")
    } else if (type == "aba") {
        plot <- aba_data %>%
            filter(variable == "ABA") %>%
            ggplot(aes(x = Month, y = value)) +
                geom_line(aes(color = Queue)) +
                scale_y_continuous(
                    labels = function(x) {scales::percent(x, accuracy = 1)}
                    breaks = function(x) {seq(0, max(x), by = 0.05)}
                    )
    } else if (type == "service_level_aba") {
        plot <- aba_data %>%
            filter(variable == "ABA" | variable == "Service Level (Avg)") %
            ggplot(aes(x = Month, y = value)) +
                geom_line(aes(color = Queue)) +
                facet_wrap(variable ~ ., scales = "free_y") +
                scale_y_continuous(
                    labels = function(x) {scales::percent(x)}
                    )
    }
    plot <- plot +
        theme_bw() +
        expand_limits(y = 0) +
        scale_x_date(date_breaks = "1 month", date_labels = "%b %y") +
        geom_smooth(method = "lm", se = F, linetype = "dotted", size = 0.25
    return(plot)
}

customize_hovertext <- function(plotly_object, plot_type) {
  chart_font <- list(family = "arial", size = 12)
  # plotly_object$x$layout$showlegend <- FALSE
  get_hovertext_for_chart_number <- function(chart_number, type = plot_type
    if (class(plotly_object$x$data[[1]]$x) != "Date") {
        hovertext_x <-
      format(convertToDate(
            plotly_object$x$data[[chart_number]]$x,
            origin = "1970-01-01"
            ), "%b %Y")
    } else {
        hovertext_x <-
      format(plotly_object$x$data[[chart_number]]$x, "%b %Y")
    }
    if (type == "aba") {
      hovertext_y <- plotly_object$x$data[[chart_number]]$y %>%
            as.numeric %>%
            scales::percent(accuracy = 1)
    #   hovertext_perc <- "%"
    } else if (type == "calls") {
      hovertext_y <- round(plotly_object$x$data[[chart_number]]$y, digits =
    #   hovertext_perc <- NA
    } else if (type == "times") {
      hovertext_y <-
        round(
            seconds_to_period(plotly_object$x$data[[chart_number]]$y),
            digits = 0
        )
    #   hovertext_perc <- NA
    }
    hovertext <- paste0(hovertext_x, ": ", hovertext_y)
    return(hovertext)
  }
  x_axis_format <- list(tickfont = chart_font)
  if (grepl("aba", plot_type)[1] == TRUE) {
    for (i in c(1,3)) {
      plotly_object$x$data[[i]]$text <- get_hovertext_for_chart_number(i)
    }
    plotly_object$x$data[[3]]$showlegend <- TRUE
    plotly_object$x$data[[3]]$name <- "Linear Trend"
    plotly_object$x$layout$annotations[[1]]$font$family <- "arial"
    plotly_object <- plotly_object %>%
        layout(
            xaxis = x_axis_format,
            yaxis = list(
                tickfont = chart_font,
                rangemode = "tozero",
                tickformat = "%"
                    )
                )
  } else if (grepl("calls", plot_type)[1] == TRUE) {
    for (i in c(1:3, 7:9)) {
      plotly_object$x$data[[i]]$text <- get_hovertext_for_chart_number(i)
    }
    for (i in 7:9) {
      names <- c(1,2,3,4,5,6,"Offered", "Abandoned", "Answered")
      plotly_object$x$data[[i]]$showlegend <- TRUE
      plotly_object$x$data[[i]]$name <- paste0(
        "Linear Trend",
        " (", names[i],
        ")"
        )
    }
    for (i in 1:3) {
      plotly_object$x$layout$annotations[[i]]$font$family <- "arial"
    }
    y_axis_format_calls <- list(
        tickfont = chart_font,
        rangemode = "tozero",
        tickformat = ",d"
        )
    plotly_object <- plotly_object %>% layout(
      xaxis = x_axis_format,
      yaxis = y_axis_format_calls,
      xaxis2 = x_axis_format,
      yaxis2 = y_axis_format_calls,
      xaxis3 = x_axis_format,
      yaxis3 = y_axis_format_calls
    )
  } else if (grepl("times", plot_type)[1] == TRUE) {
    for (i in 1:6) {
      plotly_object$x$data[[i]]$text <- get_hovertext_for_chart_number(i)
    }
    for (i in 4:6) {
      names <- c(1,2,3,"", "Answer Speed", "Handle Time")
      plotly_object$x$data[[i]]$showlegend <- TRUE
      plotly_object$x$data[[i]]$name <- paste0(
        "Linear Trend",
        " (", names[i], ")"
        )
    }
    for (i in 1:2) {
      plotly_object$x$layout$annotations[[i]]$font$family <- "arial"
    }
    y_axis_format_times <-
      list(
          tickfont = chart_font
           )
    plotly_object <- plotly_object %>% layout(
      xaxis = x_axis_format,
      yaxis = y_axis_format_times,
      xaxis2 = x_axis_format,
      yaxis2 = y_axis_format_times
    )
  }
  plotly_object <- plotly_object %>%
    layout(hovermode = "x",
           legend = list(font = chart_font))
  return(plotly_object)
}

plotly_aba_summary <- function(aba_data,
                               type,
                               queue_search = NA,
                               exclude_queues = NA) {
  plot_aba <- plot_aba_summary(aba_data, type, queue_search, exclude_queues
  plotly_aba <- customize_hovertext(
    ggplotly(plot_aba, dynamicTicks = FALSE),
    type
    )
  return(plotly_aba)
}
```