DATA ANALYST LEVEL 4 APPRENTICESHIP PORTFOLIO

Learner Name: Sergio Antonelli

ULN: 5904656938

Contents

Table Of Figures

Data Practices at Merlin

Team Structures, Roles & Responsibilities, Data Protection & Legislation

Merlin Technology Team Overview

The data team at Merlin Entertainments works under Shan Liu, my line manager, who oversees Data & Integrations within the IT department of Merlin. She reports to the head of products & corporate portfolio, who in turn reports to the CTO of Merlin. The data team is made up of three data developers (including me as a trainee), who design, construct and implement new databases, manage existing databases, and develop BI reports. Two data architects and a designer within the team oversee the architecture of the data model for the company. The technical BA acts as a translator between the technical and non-technical sides of the business to manage the requirements of IT projects, mainly those that deal with data. Finally, the data analyst and data scientist utilise the data to gather relevant business insights to try and improve the customer experience. An overview of the data team within my organisation is shown in figure 1.

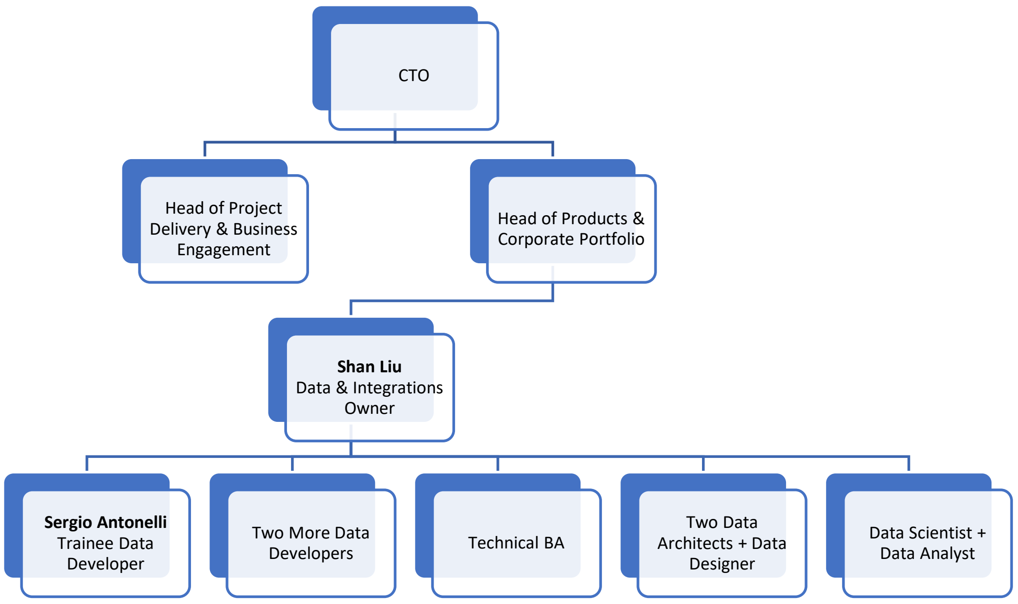
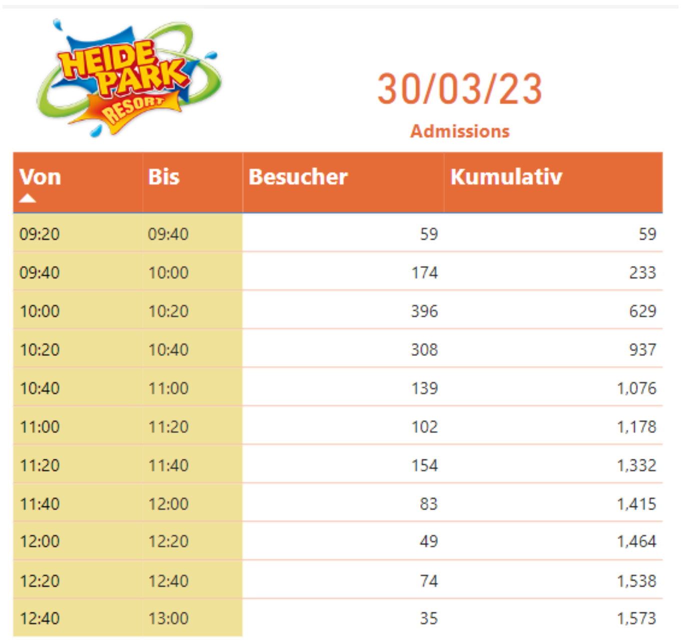

CTO
Head of Project Delivery & Business Engagement
Head of Products & Corporate Portfolio
Shan Liu
Data & Integrations Owner
Sergio Antonelli
Trainee Data Developer
Two More Data Developers
Technical BA
Two Data Architects + Data Designer
Data Scientist + Data Analyst

Figure 1: Overview of data team within Merlin Entertainments Technology Team

Data Developer Roles & Responsibilities

As a data developer working for Merlin Entertainments, I am involved in projects such as data integrations, to connect data coming from, for example, park admissions or retail sales, to the company data hub. I am also involved in reporting projects, such as live admissions or mobile app data reporting. In terms of data analytics, while working on reporting projects, I mainly use SQL server databases, as well as Google BigQuery databases, where I first transform and organise my data into new tables, therefore ensuring that I maximise efficiency when importing my data into Power BI (as importing millions of rows of data within Power BI is sometimes not feasible). Depending on the project, I may sometimes carry out all the transformation using SQL before importing, while sometimes most of it is done in Power Query. I then create my visualisations within Power BI desktop before publishing the report to the correct BI workspace. An AAD (Azure Active Directory) group is created including all the stakeholders which require access to the report. The BI workspace is then shared with this AAD, who can then view the report. An example of these sorts of projects that I have been working on can be seen in figure 2, which depicts the live admissions at one of our resort theme parks in Germany. This BI report is linked to a dataset in our SQL server which is in turn updated through SSIS (SQL Server Integration Services) every 5 minutes from a database in the park's server.


30/03/23

Admissions

Von	Bis	Besucher	Kumulativ
09:20	09:40	59	59
09:40	10:00	174	233
10:00	10:20	396	629
10:20	10:40	308	937
10:40	11:00	139	1,076
11:00	11:20	102	1,178
11:20	11:40	154	1,332
11:40	12:00	83	1,415
12:00	12:20	49	1,464
12:20	12:40	74	1,538
12:40	13:00	35	1,573

Figure 117: Live Admissions report for Heide Park Resort theme park in Germany.

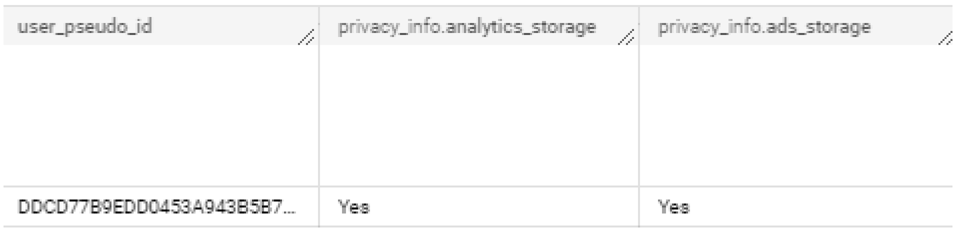
While I have not been involved in integration projects yet, I have used SSIS, as well as SQL Agent in reporting projects. Using SQL Agent, I can automate workflows, such as receiving real-time data from a database for one of our parks every 5 minutes. This data can then automatically updates some of our reports, making it possible to create rides and admissions real-time analytics for the parks. There is one more type of project, which we often call "Strategic Products". These are usually larger-scale projects, which may involve multiple data integrations. For example, one of these projects that the IT team is currently working on is a "Customer Data Platform". The aim of this project is to obtain a single customer profile by retrieving data from multiple sources, such as the mobile app, online ticketing purchases, or google ratings, to create a single customer profile which will make for a much better user experience on our systems as everything will be connected and searches, as well as log-ins, will all be far more efficient (this will also lead to better marketing strategies). More recently, I have been involved in a project within the data science team. The team is a product team and works in an agile environment, made up of sprints. Each sprint consists of two weeks, starting with a planning session, where everyone's responsibilities are laid out; a review, where everyone's work throughout the two weeks is summaries; and a retro, where we discuss the pros and cons of the past fortnight's work.

Internal VS External Customers

As a data developer, I mainly work with internal customers who require certain aspects of their parks/attractions reported. Working on data integrations will instead affect the work of the IT team, as they will have better access to certain data required to complete specific IT-related projects. Finally, some of the projects will instead be directed towards external customers. This could be indirectly, such as utilising data to drive better business decision-making, which can then improve the guest user experience as we understand their wishes to a larger extent. However, my work could impact external customers (such as guests) directly as well; for example, creating a single-customer data platform will directly improve the guest experience as guests will be able to receive better targeted suggestions on other attractions or rides they may enjoy, while also creating a well-connected Merlin environment for usual customers, such as Annual Pass holders, which will again improve the user experience.

Data Protection Practices

To maintain data protection standards throughout any data processing activity, the Merlin tech team has a data protection team that oversees any projects that involve the use of personal and non-personal data. At the start of every project, during the requirements stage, whoever is responsible for the processing of any sort of data is also responsible for data compliance. As a rule of thumb, data compliance is everyone's responsibility. If a project uses any personal data, a ROPA must be completed (Record of Processing Activity), where all the personal data processing is recorded. Following the RoPA is a DPIA (Data Protection Impact Assessment) which must be completed by the contract owner, with the support of a relevant IT team member or data protection team. According to Article 35(1) in the UK GDPR, a DPIA is only needed in projects where there is high risk personal data processing (for example, the processing of large volumes of employee payroll data or prior to implementing a new system for managing guest health data), which can risk rights and freedoms of individuals. Currently, I am working on a report for all the data coming from the mobile apps for many of our attractions. This data includes some level of personal data, such as the location that the phone's app store is set to, as well as a unique ID for each user. For now, I have only been working with the data collected on user interactions (that is, viewing different screens, applying filters, etc). This data is pseudonymised, where each interaction with the app has a record with a unique pseudo user



user_pseudo_id	privacy_info.analytics_storage	privacy_info.ads_storage
DDCD77B9EDD0453A943B5B7...	Yes	Yes

Figure 192: User pseudo-ID recorded, along with privacy settings. KSB(K1,K2)

ID (as shown in figure 3). Each user's privacy preferences are also recorded, and the appropriate actions are taken to further anonymise or remove data where consent for data processing was not given.

The reporting I am working requires features such as the number of guests at any time using the app, the most popular rides by their filter usage, and how often guests will use features such as queue times or wayfinding. None of these will display any level of personal detail within the report. The data I will soon receive will instead contain sign-up information, such as email addresses and dates of birth. This will require a higher standard of data protection practices compared to what I have worked on for now. To anonymise all this data, I will remove some of the information which I will not need for my report, such as the names of the guests and their email addresses. Some personal data that I will instead need to keep is age, nationality, and gender, as these can be useful in reporting on business insights. Due to the nature of the report requirements, which will involve a lot of aggregation and counts, there will be very low risk, if any, to personal data security breaches. A data breach could be extremely detrimental for the company, as we mainly cater to families. Maintaining the privacy and security of children on our systems is a top priority as they are usually more at risk. If anything of this sort were to happen, I would immediately report it to both Shan as my line manager, as well as Cameron, who works within the data protection team in my department, as he would know the most appropriate steps to take to manage these issues. As a data developer, I also manage our databases, which can again contain large amounts of customer personal data. Having access to this data makes me responsible for it, meaning I will have to make sure that there are no breaches to our servers and that the data being taken out of our data hub is necessary and meets data compliance practices. If I were to receive a request to delete a customer's personal data, as is their right under the UK GDPR legislation, the process would simply take a short command in SSMS to remove the relevant data from our server.

Merlin Foresight – Customer Focus & UX

Foresight POC, Customer Personas, Project Types & Communication

Introduction

Merlin Entertainments is currently working with Boston Consulting Group to introduce a new pricing model for one of our parks: Chessington World of Adventures. This new pricing model, which will be rolled out to other attractions in the future, will entail a dynamic pricing approach, meaning that ticket prices will be based on attendance tiers (Super Off-Peak, Off-Peak, Peak and Super Peak). Currently, these tiers are being set based on a park budget which is discussed at the start of each calendar year. These numbers have shown to often be far from accurate and can sometimes miss actual attendance numbers by the thousands. The project I have therefore been working on has been a machine learning enabled demand forecast to achieve more accurate tier predictions to maximise revenue at our parks when using dynamic pricing. This will positively impact the work of our stakeholders, and the financial team at Chessington, who will therefore be able to set these prices without having to manually calculate and predict the correct ticket prices at their park. This project will also involve the construction of a Power BI dashboard which will depict the attendance forecast numbers for the next two weeks, as well as show some recommendations on pricing and some analysis on previous forecasts to build trust in the model. When this trust is built, it will enable me to create an automatic system of dynamic pricing with little to no manual changes, which can be rolled out to other parks across the organization.

Project Outcomes

- Build a forecasting model with a ~90% tier prediction accuracy
- Create a dashboard depicting forecasted numbers and relevant tiers
- Build trust in machine learning technology within the department for future uses
- Plan company-wide roll-out.

Customer Persona

The consumer for the forecasting model will be internal stakeholders, such as the finance team within merlin using predicted attendance for revenue forecasting. The pricing team at Chessington will also be using the model, as the attendance tiers predicted by the model will help set prices based on attendance on any given day within the next two weeks.

Mark, 38: Pricing Team at Chessington World of Adventures

- Spends at least 4 hours weekly with his team on pricing tickets for upcoming days based on recent trends in attendance, weather, park events, prebooks and more.
- Mark hopes to reduce the time he spends weekly on these pricing predictions to focus on other important work such as setting promotional offers for customers.
- Every week, once he has set these prices, he must discuss this pricing with the wider team, and then raise a request with the IT team to make these prices official on the website, and communicate these numbers to the finance team for their revenue predictions, making the whole process very redundant.

Mathilde, 47: Financial Team at Chessington World of Adventures

- Her work includes predicting revenue for Chessington throughout the week.
- Her manager wishes she could predict revenue longer term, which would lead to less resources being needed each week for these predictions.
- The finance team she works with has also been struggling to accurately predict revenue, making stakeholders unhappy with the information they had gathered at previous stakeholder meetings.

Root Cause Analysis

Predicted attendance tiers are inaccurate and lead to diminished revenue. WHY?

Park budget usually set at the start of the year, and pricing then based on this budget and current attendance trends. This leads to less revenue throughout the year. WHY?

It is impossible to accurately predict attendance at the start of the year for revenue forecasting or pricing, this will therefore lead to incorrect attendance tier predictions and therefore will not maximise revenue and park attendance. It will also lead to more work for the pricing and finance team who will need to constantly update these numbers every 5 days based on mainly prebooks. WHY?

Current trends are often unknown at the start of the year. It therefore makes sense that numbers are forecasted closer to the date. Currently, these numbers are forecasted a maximum of 5 days in advance to be accurate. This is however inefficient as the business wants to have somewhat accurate numbers at least two weeks in advance. WHY?

Forecasting 5 days in advance will create a lot of work for the business who will have to change pricing every 5 days, and makes accurate revenue forecasting impossible in the long-term. WHY?

Currently, there is a lot of manual work to be done every week. The root cause of this problem is therefore that having to manually update these numbers each week may lead to more inaccurate predictions due to human error. Furthermore, more accurate predictions are currently only possible 5 days in advance using these methods.

Project Characteristics

This project is a movie; the stakeholders (the finance and pricing teams at Chessington) know exactly what they want: a time-series forecast to predict park attendance. While there are some variables that are still up to debate, such as the length of the time-series (two weeks, three weeks, or more), the requirements were set early in the project timeline. Since the stakeholders have been using manual methods for years, they do not know how to move from these methods to find more accurate ways of predicting attendance. I can help design a machine learning model to maximise revenue and minimise the manual labour that our stakeholders are currently accustomed to. The project will initially be a software development linear project, with a planned set of deliverables (with some adjustments throughout), a design (plan the algorithm, the data types, the features, etc.), the implementation in databricks/python, deployment of the dashboard with constant testing week after week to build confidence in the model, and maintenance (either biweekly if manual runs will be required, or whenever issues arise with the dashboard or model). The project may later become agile when the model has built enough confidence where other parks will request the same dynamic pricing plans with a forecasting model. Since each park is different, they will each require a separate set of deliverables and will need different features to accurately score attendance, creating cycles of smaller projects for each park.

MoSCoW

Must Have:

- More accurate predictions compared to current method (park budget)
- Dashboard for easy visualization of attendance number predictions
- Include weather, school holidays and park closures to model
- At least 4 tiers of attendance predictions (super off-peak, off-peak, peak, super peak).
- At least two weeks worth of predictions

Should Have:

- Automation based on mounting servers to model to avoid manual predictions every two weeks
- Model connected to power bi dashboard for updating numbers to prevent manually refreshing dashboard
- Include web prebooks and park events within model
- Pipeline easily transferrable to other parks in case the model is successful at Chessington

Could Have:

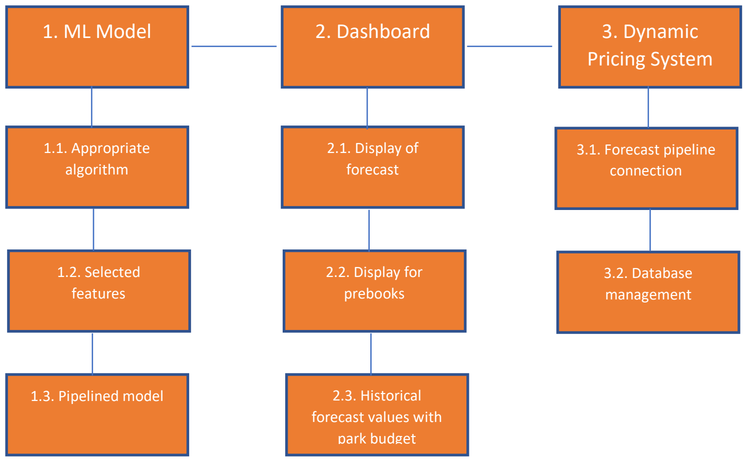
- Prebook numbers in dashboard to view adjacent to predictions for finance team to assess whether predictions will be accurate closer to dates
- Automated dynamic pricing based on predictions to completely cut off any manual labour when setting park pricing
- Up to 7 tiers of attendance predictions
- Up to three weeks of predictions

Won’t Have:

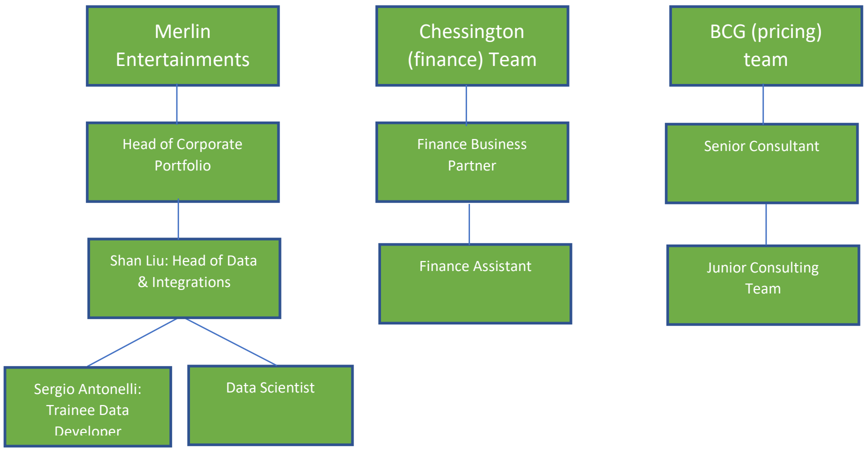
- Worst overall predictions compared to current method
- Prediction length of less than two weeks
- Less than 4 tiers of attendance predictions

Breakdown Structures

PBS


1. ML Model
1.1. Appropriate algorithm
1.2. Selected features
1.3. Pipelined model
2. Dashboard
2.1. Display of forecast
2.2. Display for prebooks
2.3. Historical forecast values with park budget
3. Dynamic Pricing System
3.1. Forecast pipeline connection
3.2. Database management

OBS



WBS

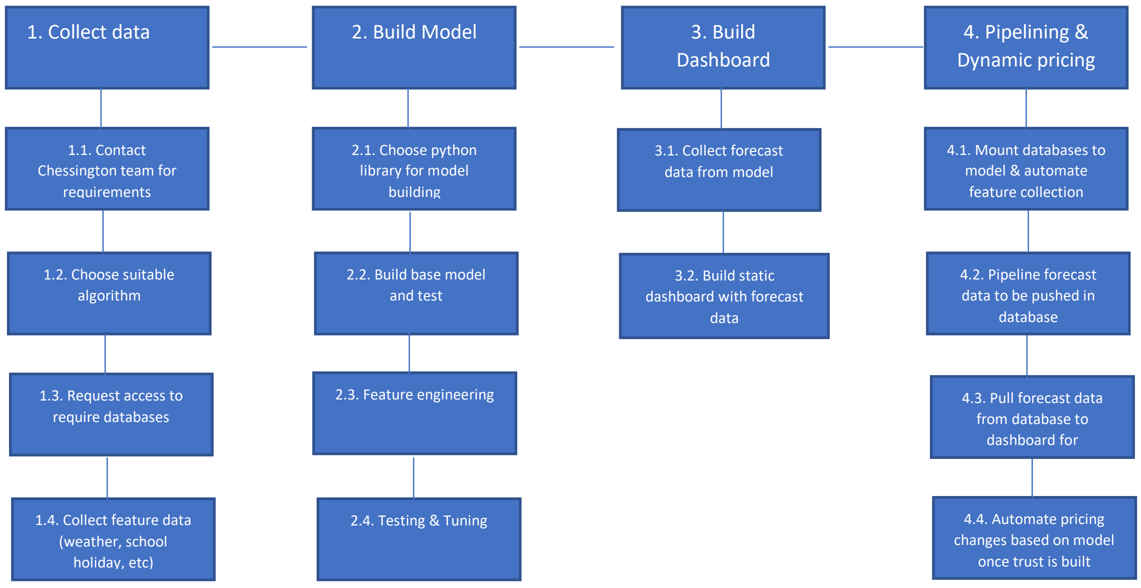

1. Collect data
1.1. Contact Chessington team for requirements
1.2. Choose suitable algorithm
1.3. Request access to require databases
1.4. Collect feature data (weather, school holiday, etc)
2. Build Model
2.1. Choose python library for model building
2.2. Build base model and test
2.3. Feature engineering
2.4. Testing & Tuning
3. Build Dashboard
3.1. Collect forecast data from model
3.2. Build static dashboard with forecast data
4. Pipelining & Dynamic pricing
4.1. Mount databases to model & automate feature collection
4.2. Pipeline forecast data to be pushed in database
4.3. Pull forecast data from database to dashboard for
4.4. Automate pricing changes based on model once trust is built

Figure 4: Product Breakdown Structure (PBS), Organizational Breakdown Structure (OBS) and Work Breakdown Structure (WBS) KSB(K7, S5, B5)

Identifying Risk

The risks associated with this project are all based on the model outcome. While I wish for the model to be as accurate as possible, there are always going to be limitations to its success also based on the data available. Table 1 is a grid of risks, with their likeliness to happen and their potential impact:

Risk	**Impact**	**Impact Level**	**Prob. level**	**Priority level**	**Mitigation notes**	**Owner**
Model scores worse than budget	This would be a waste of resources on a long project, as the product will be completely redundant and worse than current methods	5	1	5	While it is very unlikely that my model will score worse than park budget (as this is planned at the start of the year), I must also make sure that the model can score ***significantly*** better for it to be a good use of resources.	Merlin Team (improve model) Chessington Team (support merlin team with their previous forecasting methods)
Data missing from database	This will lead to issues in model building. Some features might have to be removed from the model due to this.	2	3	6	Strong communication with different departments to ensure I am aware of where the correct data is stored. Before planning a feature, be sure of the existence of relevant data.	Merlin Team (internal issues) Chessington Team (support merlin with experience in using similar data for their forecasts)
Automation impossible due to permission issues	If I cannot get access to certain databases, and therefore cannot mount them to my model, the forecast will be a manual job to be carried out each week.	2	2	4	While the manual job would only take two hours every two weeks for a single employee, it could lead to human error and will use more resources. It is therefore important to ensure I will have relevant access and request it ahead of time.	Merlin Team (internal requests)
BCG pricing plan leads to a significant decrease in model accuracy	If BCG, for example, decides to create 7 tiers of attendance and therefore 7 separate prices for the park, this may lead to my model struggling to accurately predict these, as it is easier for the attendance numbers to be in the correct pricing band when there are only 4 tiers rather than 7.	3	4	12	It is important that I stay in constant communication with BCG and provide information on why a strict pricing plan may lead to issues in the model accuracy. This is something that can be changed in the future, as the model will become more and more accurate going forward. If this were to happen, I could re-engineer the model to predict attendance tiers using a classifier rather than attendance numbers using a regressor, which statistically speaking should score much better, but will make the model redundant in terms of revenue forecasting.	Chessington Team (Decide whether a classifier model might work for revenue forecasting) BCG Team (Plan the dynamic pricing at Chessington)

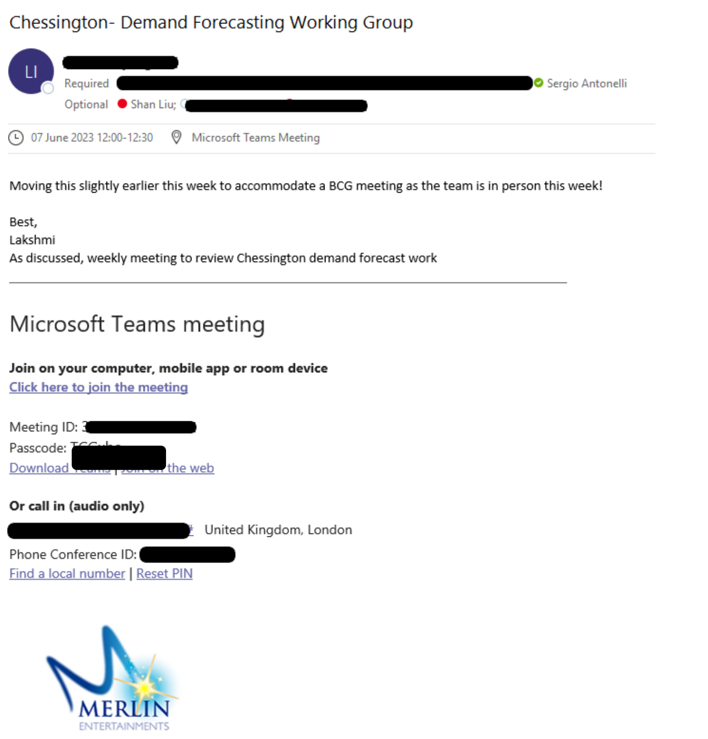
Table 1: Risks grid for Foresight In-Attraction forecasting project KSB(K7, S5, B5)

Research on End-User

To capture all the wishes of the end-user to my model, I used “interviews”. In other words, I had recurring teams calls with both dynamic pricing team (BCG), financial team (Chessington team) and the data team working within Merlin. To make the dashboard as efficient as possible, I kept only relevant information displayed. While there are a lot of factors which could influence attendance which cannot be captured on my model, these should be known by the end-user and are kept to only essential ones in the dashboard. The end-user only wishes to visualize the model scores and a visual on the historical accuracy of the model to build trust in it. I kept navigation very simple, by having two panes, one with the current forecast and one with historical accuracy. The dashboard is mostly be static, except for sliders for date to check model accuracy on any given period in the past. This simplicity ensures efficient use of the model to set dynamic pricing.

Communication

Throughout this project, there has been a lot of communication with both the Chessington team and BCG. Personally, I have been more involved with the Chessington team, while our Senior data scientist has been in more communication with BCG. Below (figure 5) is the meeting invite for the weekly conversations with the Chessington team.



Chessington- Demand Forecasting Working Group

LI [redacted]

Required [redacted] Sergio Antonelli

Optional Shan Liu; [redacted]

07 June 2023 12:00-12:30 Microsoft Teams Meeting

Moving this slightly earlier this week to accommodate a BCG meeting as the team is in person this week!

Best,
Lakshmi
As discussed, weekly meeting to review Chessington demand forecast work

Microsoft Teams meeting

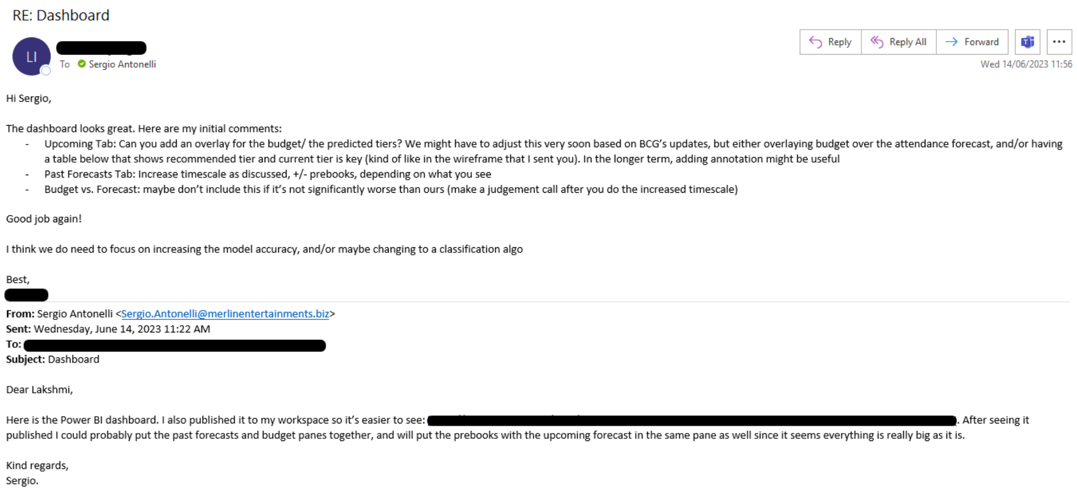
Join on your computer, mobile app or room device
Click here to join the meeting

Meeting ID: [redacted]
Passcode: [redacted]
Download [redacted] the web

Or call in (audio only)
[redacted] United Kingdom, London
Phone Conference ID: [redacted]
Find a local number | Reset PIN

Figure 5: Teams invite weekly meeting with Chessington Working Group
KSB(K7, S5, B1, B5)

There was a lot of communication between me and the senior data scientist in our team. This product was in fact built by us with the guidance of the other teams involved in this project. This meant a great deal of adjustments having to be communicated between us every day. Following is an email (figure 6) between us regarding the dashboard which would visualize my forecast. It mentions BCG updates, which refers to their pricing plan and the possibility of increasing attendance tiers from 4 to 7, as discussed in the next section.


RE: Dashboard

LI [redacted]
To Sergio Antonelli

Hi Sergio,

The dashboard looks great. Here are my initial comments:

- Upcoming Tab: Can you add an overlay for the budget/ the predicted tiers? We might have to adjust this very soon based on BCG's updates, but either overlaying budget over the attendance forecast, and/or having a table below that shows recommended tier and current tier is key (kind of like in the wireframe that I sent you). In the longer term, adding annotation might be useful
- Past Forecasts Tab: Increase timescale as discussed, +/- prebooks, depending on what you see
- Budget vs. Forecast: maybe don't include this if it's not significantly worse than ours (make a judgement call after you do the increased timescale)

Good job again!

I think we do need to focus on increasing the model accuracy, and/or maybe changing to a classification algo

Best,
[redacted]

From: Sergio Antonelli <Sergio.Antonelli@merlinentertainments.biz>
Sent: Wednesday, June 14, 2023 11:22 AM
To: [redacted]
Subject: Dashboard

Dear Lakshmi,

Here is the Power BI dashboard. I also published it to my workspace so it's easier to see: [redacted]. After seeing it published I could probably put the past forecasts and budget panes together, and will put the prebooks with the upcoming forecast in the same pane as well since it seems everything is really big as it is.

Kind regards,
Sergio.

Figure 6: Example email exchange with supervisor Senior Data Scientst KSB(K7, S5, B1, B5)

Conflict Management

Boston consulting group (BCG) is currently dealing with some of the model requirements to make their dynamic pricing model efficient. They have requested in the past 7 tiers of attendance to be predicted. However, this heavily impacted the accuracy of the model. I currently reached 83% accuracy for the model with 4 tiers. Increasing the tiers to 7 would drop this accuracy to about 60%, which would not be efficient when used with dynamic pricing. To manage these issues, I had to stay in constant communication with BCG and acquired evidence that the model cannot be more accurate than it currently is due to some missing data which is impossible to retrieve. This data is slowly being collected, and the model will continue learning every week. It is therefore important to note that 7 tiers will most probably be possible and a useful requirement further down the line, but an impossible request with the current resources. Following the Thomas Kilmann Model, I adopted a collaborating style of conflict management, meaning I had high cooperation with BCG and their requests as they are a very important part of many of our data-driven projects, and are therefore a relationship I must preserve. I also had high assertiveness as their request can prove to be a big issue, as it could make my model increasingly worse.

Merlin Foresight – Development, Deployment & Support

Foresight Product Workstream, Python & time-series forecasting, Power BI & analytics

Introduction

Assignment Background

Merlin Foresight is a demand forecasting product which first went live for one of Merlin Entertainments' parks: Chessington World of Adventures. It is the workstream product born from an initial POC, which I discussed in the Merlin Foresight - Customer Focus & UX section. It is now rolling out to multiple other parks, such as the north American Legolands, as well as other UK parks such as Alton Towers, Thorpe Park and Legoland Windsor, with the plan to roll out to all 200 Merlin attractions worldwide in the next two years. This product is an attendance-based forecasting tool, to produce highly accurate predictions on park volume/attendance for the upcoming 14 days. It was initially a very abstract concept, with little to no data science environment built to produce something of value efficiently. After having to slowly build up and environment and data engineering pipelines around this project, it has since grown to be one of the most exciting projects within Merlin IT, gaining attention from the wider business, with the interest of the CTO and CEO. Due to this, we have been building a data science team which will initially focus on just Merlin Foresight, to speed up the roll out of the model and improve the underlying processes. Since I built the initial model, as well as being responsible for the feature engineering and testing of the first instances of this product, I have since been moved from the general data team to this newly built data science team.

The key stakeholders for this project are park revenue managers, as this product should mostly be used as a pricing adjustment tool, to maximise revenue based on predicted attendance. Initially, there was no budget for this project, but was rather a trial to see the potential for it. As I mentioned, a team of data scientists has since been hired and a budget for this product specifically will soon be set, which will mostly be computing costs, as most of my data inputs are either inexpensive or 1st party data. The project has also been set as a worksteam in the company's current data transformation, with the key requirements of the workstream being, simply put, to deliver a highly accurate attendance forecasting model.

This project covers key concepts about data visualization using power bi and plotly, data modelling using python and azure services, and data analytics/data science work using time-series models and linear regressions for predictive analysis.

Requirements & Dashboard Design

The requirements for the model & dashboard are broad, as this is a somewhat experimental project. However, the overall expectations are as follows:

- At least two weeks of attendance predictions.

- Predictions of tickets sold split by channel (annual passes, online purchases, promotions, etc.)
- A dashboard depicting said predictions and the model's historical accuracy.
- Daily predictions and dashboard refresh.

The dashboard requirements are as follows:

- Page 1: forecasted attendance for the next two weeks, as well as a YTD mean average error of the model.
    - On the same pane, a table showing all forecasted attendance tiers (these differ from park to park, but are usually from 4 to 8 pricing tiers), as well as recommendations based on initially set attendance tiers.
- Page 2: forecasted prebooks split by channel.
- Page 3: historical forecasts for attendance for at least a year, with a mean average error and a percentage of correct tier predictions
- Page 4: same as pane 3, but for prebooking forecasts.

In order to connect to the data, I simply connect to the correct container on an Azure storage account. Each region has it's own container, each containing a folder for each park in that region.

Data Sources & Software

To build this time-series forecast, I used python as per industry standards in data science. The main packages I used for the transformation and formatting of the training data were pandas, numpy and datetime, which are all standard for this use-case, as well as some use of pyspark to ingest attendance data from a large industry database. To build out the actual model, I used Greykite, a forecasting library for explanatory data analysis, forecast pipeline, model tuning, etc. It includes a Silverkite model, which is a forecasting model developed by LinkedIn, which allows for feature engineering, automatic changepoint detection, holiday effects, and works very well with picking up on seasonality, also handling things like overfitting very well. The industry standard for time-series forecasts is usually Prophet, a forecasting model developed by Facebook, which Greykite also uses in part. I opted to use Greykite due to the ease of use, as well as it's great handling of seasonal trends, and built in hyperparameter tuning (Grid search). Having to build 200+ models for separate parks, with slightly different feature importance and seasonality trends, I opted for this package to speed up the roll out process.

Most of my code, both for data transformation / modelling and the actual model build, is stored in Databricks notebooks. I currently have one .py file containing all the functions I built for the data modelling and model run, and then one notebook for each park I have rolled out as of now. Merlin IT uses Microsoft services, therefore making Databricks an obvious choice, due to it's compatibility with other products such as Azure Services and Power BI. Once the models are run, the outputs are pushed into an Azure Blob Storage. I have various Azure Functions which trigger once these files are output, built to update historical files and do some final transformation before pushing to Power BI dashboards.

As in any data project, to collect all the relevant data, there was a lot of communication with both the project stakeholders and the business. The main data sources required involved the following:

- Historical park attendance & prebooking data (from 2019 onward):
  - This data is taken from an Azure storage account, where data is ingested from accesso (ticketing system) servers.
  - The prebooking data is split by channel. The included channels, some of which are separated in sub-channels, are: Web, Annual Passes, Promotions, Hotels, Trade-in, Schools, Corporate and Walk up. The last two are other types of promotional ticket types: Consumer Frees & Sun Frees.
- Historical and current advanced prebooking data (therefore, the total amount of prebooks x days in advance from actual date):
  - This is a more complex data source. Currently, I do not have access to this data directly on our servers. I instead receive an email every day with a hyperlink containing the advanced prebooking data for the upcoming year. I have pipelined this to extract the next 14 days' worth of data.
- Historical and forecasted weather data:
  - This is collected using an API from a weather forecasting website.
- Other data sources to be updated yearly:
  - School holidays: This can either be provided by parks, or may have to be internal research using google or other third party providers.
  - Park events, closures, and hours: All of these regressors are collected from the park calendar.

Model Pipeline

The general pipeline for the foresight product can be visualized below. This section will focus on each detail within the diagram in figure 7.

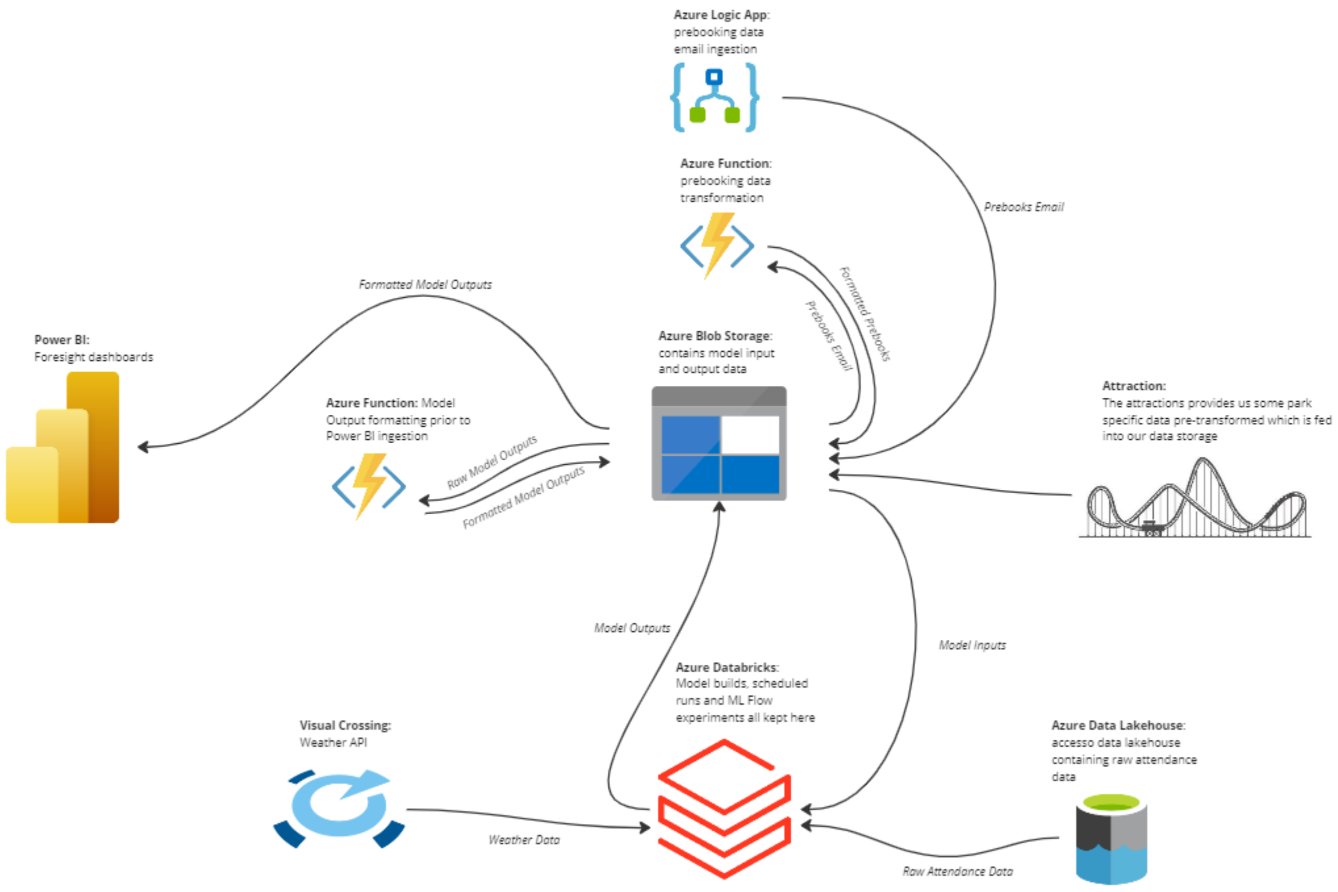
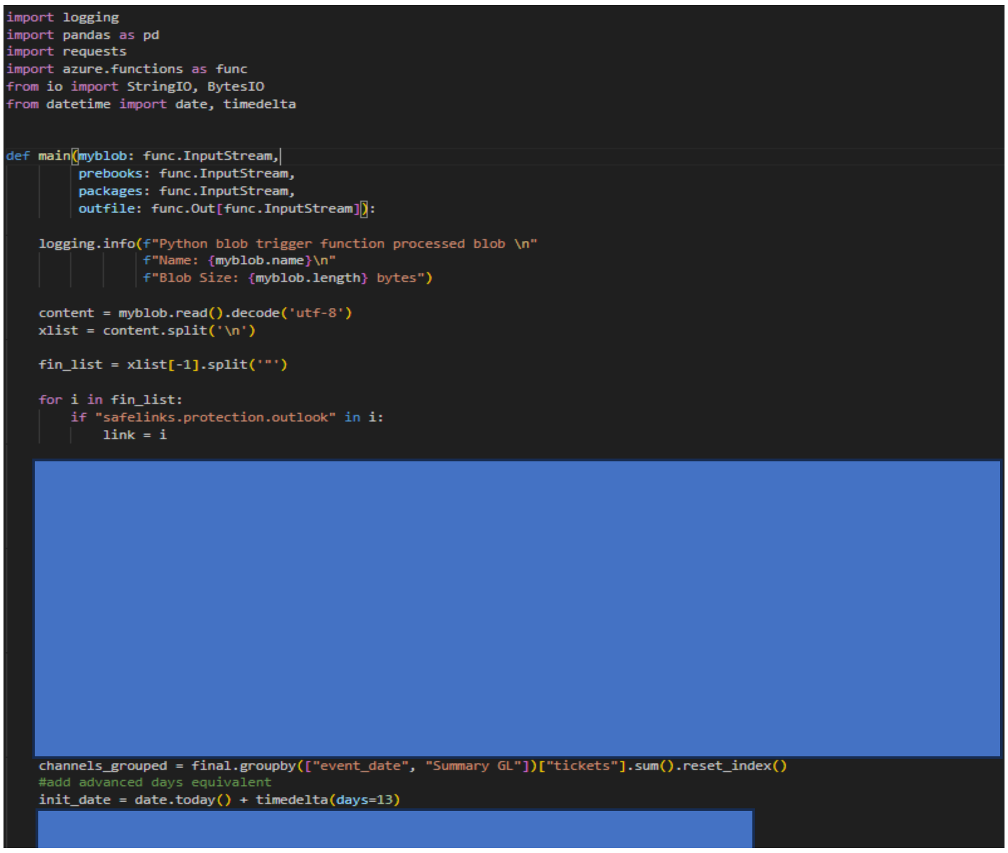

Azure Logic App: prebooking data email ingestion
Azure Function: prebooking data transformation
Prebooks Email
Formatted Prebooks
Prebooks Email
Formatted Model Outputs
Power BI: Foresight dashboards
Azure Blob Storage: contains model input and output data
Azure Function: Model Output formatting prior to Power BI ingestion
Raw Model Outputs
Formatted Model Outputs
Attraction: The attractions provides us some park specific data pre-transformed which is fed into our data storage
Model Outputs
Model Inputs
Azure Databricks: Model builds, scheduled runs and ML Flow experiments all kept here
Visual Crossing: Weather API
Azure Data Lakehouse: accesso data lakehouse containing raw attendance data
Weather Data
Raw Attendance Data

Figure 7: Foresight Product Pipeline Diagram. KSB(K10)

HTTP Trigger

The model runs each night thanks to a chain of events which lead to downstream reactions. The first event to trigger the pipeline is an “accesso insights” email containing the advanced prebooking data. An Azure Logic App triggers once these types of emails are received. The Logic App (figure 25, see appendix) counts the files in the destination folder and save the email in a .txt file within the folder “EmailTXT” adding the number of current files to the name. This is done to prevent the logic app overwriting the old files rather than adding a new one. Overwriting causes the downstream function app to not recognize that a new file has been added, and therefore won’t run. Once the new .txt file is added to the “EmailTXT” folder, an Azure Function triggers thanks to its connection to the folder.

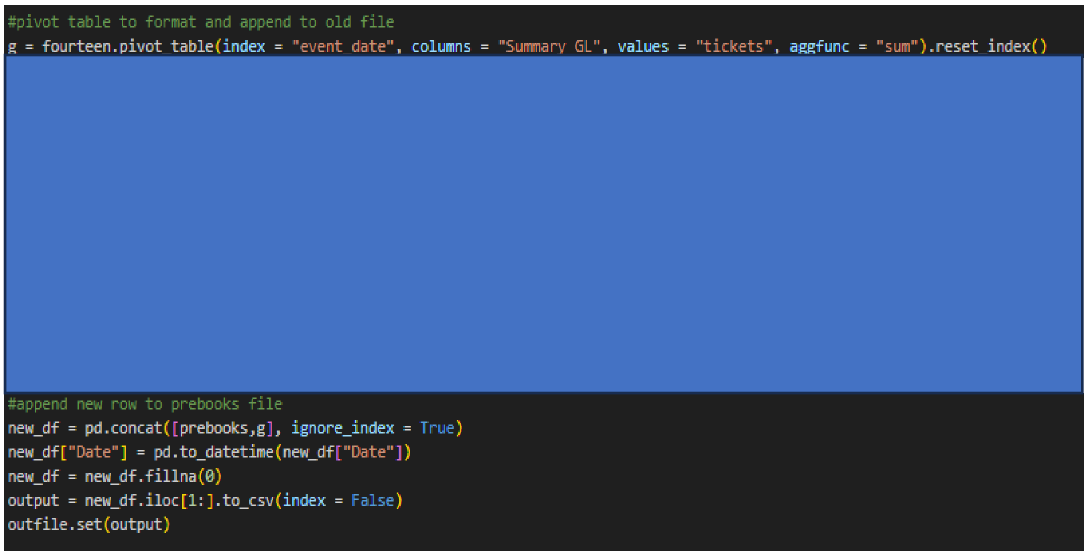
The Function's job is to extract the hyperlink from the .txt file and send an HTTP request to extract the contents of the hyperlink as a pandas data frame in python. This file contains the current prebooking data for the next years' worth of dates. My model only requires data for prebooks 14 days in advance. Therefore, from this data frame, I only extract a single row of data, corresponding to prebooks for a date 14 days in the future. The Azure Function then imports another .csv file containing historical prebooking data and appends this new row to it and stores it back in the storage account. The first part of the function is dedicated to extracting the hyperlink ("link" variable) by splitting the .txt file contents and finding the correct text row identifier. The function then uses the requests package to get the contents of the .csv file within the hyperlink. After some formatting of both the imported data, as well as our own data, I can filter for the data relating to 14 days from the current date. Due to this code being Merlin IP, most of this code had to be censored. Below are snippets of this code (figure 8).



```python
import logging
import pandas as pd
import requests
import azure.functions as func
from io import StringIO, BytesIO
from datetime import date, timedelta


def main(myblob: func.InputStream,
         prebooks: func.InputStream,
         packages: func.InputStream,
         outfile: func.Out[func.InputStream]):

    logging.info(f"Python blob trigger function processed blob \n"
                 f"Name: {myblob.name}\n"
                 f"Blob Size: {myblob.length} bytes")

    content = myblob.read().decode('utf-8')
    xlist = content.split('\n')

    fin_list = xlist[-1].split('"')

    for i in fin_list:
        if "safelinks.protection.outlook" in i:
            link = i

    [redacted]
    channels_grouped = final.groupby(["event_date", "Summary GL"])["tickets"].sum().reset_index()
    #add advanced days equivalent
    init_date = date.today() + timedelta(days=13)
    [redacted]
```

Figure 8: Azure Function App to tranform prebooking data from email txt file. Censored for Merlin IP. KSB(S10)

The second part of the function deals with some more formatting and pivots the table to get one row. This row is then appended to our historical prebooking file which is then saved back into our Azure container. Below are snippets of this code (figure 9).



```
#pivot table to format and append to old file
g = fourteen.pivot_table(index = "event date", columns = "Summary GL", values = "tickets", aggfunc = "sum").reset_index()
[redacted]
#append new row to prebooks file
new_df = pd.concat([prebooks,g], ignore_index = True)
new_df["Date"] = pd.to_datetime(new_df["Date"])
new_df = new_df.fillna(0)
output = new_df.iloc[1:].to_csv(index = False)
outfile.set(output)
```

Figure 9: Azure Function App to tranform prebooking data, pivoting and appending to existing file. KSB(S10)

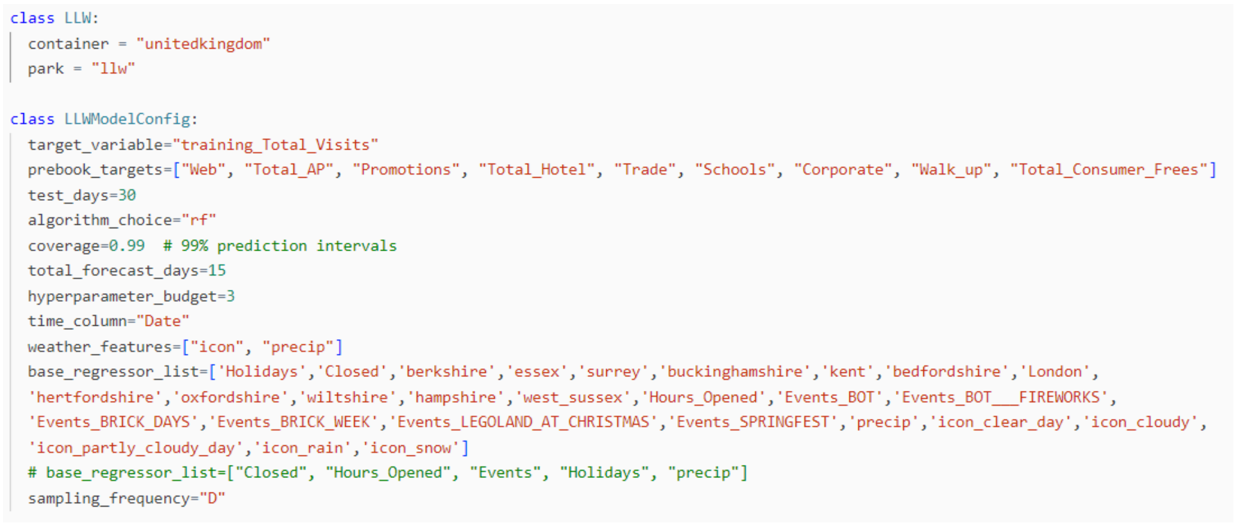
Random Forest

To build my time-series forecast, I opted to use a random forest algorithm. This type of algorithm helps me model a complex, non-linear relationship between model inputs and outputs. The reason for this choice is that random forests classifiers are less influenced by outliers, which will occur often given the nature of my data; it handles collinearity very well, which is important when having to roll-out to multiple parks, giving me the opportunity to use, for example, rain and wind speed as two separate features, although they are often correlated; and is very robust, which is great in my case due to some features having a much larger range than others.

Using a random forest, N random records and M features are taken from my dataset having K number of records. Individual decision trees are then constructed from each sample. Each of these decision trees generate an output, which is then compared and considered based on averaging to pick the best possible outcome.

Configuration

The models for each park all use very similar inputs and therefore require similar code. I therefore created a functions python file containing all the formatting code. Within each park model, I pull in the required data all from Azure. This includes attendance data, weather data, model features (school holidays, events and park hours), and prebooking data. All of these are merged together and most of the data types are all changed to float64 to account for the missing data for some years (int does not support NaN values) and to work well with greykite. I have a .config file, where I can configure each park individually. Some may require different variables and may have slightly different data inputs, and these are all specified here. Below is an example of a configuration, for Legoland Windsor (figure 10).

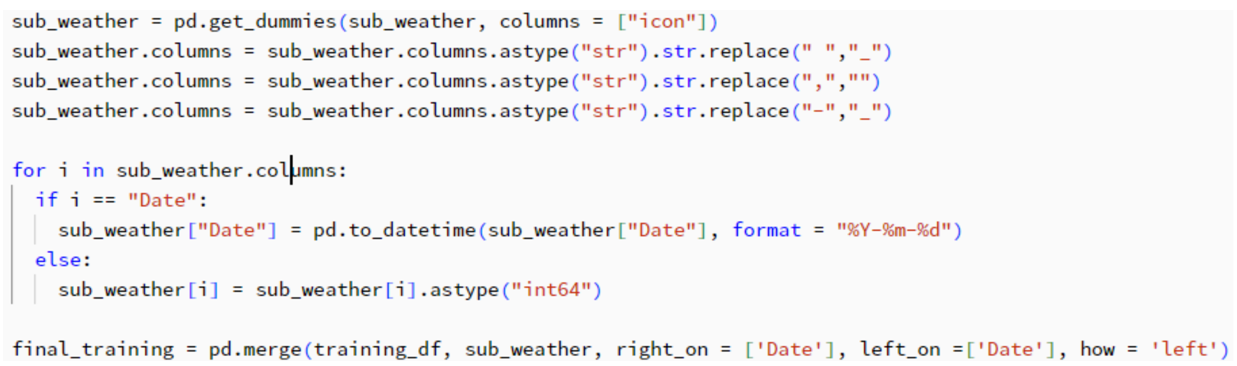


```
class LLW:
    container = "unitedkingdom"
    park = "llw"

class LLWModelConfig:
    target_variable="training_Total_Visits"
    prebook_targets=["Web", "Total_AP", "Promotions", "Total_Hotel", "Trade", "Schools", "Corporate", "Walk_up", "Total_Consumer_Frees"]
    test_days=30
    algorithm_choice="rf"
    coverage=0.99  # 99% prediction intervals
    total_forecast_days=15
    hyperparameter_budget=3
    time_column="Date"
    weather_features=["icon", "precip"]
    base_regressor_list=['Holidays','Closed','berkshire','essex','surrey','buckinghamshire','kent','bedfordshire','London',
    'hertfordshire','oxfordshire','wiltshire','hampshire','west_sussex','Hours_Opened','Events_BOT','Events_BOT___FIREWORKS',
    'Events_BRICK_DAYS','Events_BRICK_WEEK','Events_LEGOLAND_AT_CHRISTMAS','Events_SPRINGFEST','precip','icon_clear_day','icon_cloudy',
    'icon_partly_cloudy_day','icon_rain','icon_snow']
    # base_regressor_list=["Closed", "Hours_Opened", "Events", "Holidays", "precip"]
    sampling_frequency="D"
```

Figure 10: LLW Example Model Config used in model pipeline KSB(S10, S11, S13)

The python script in charge of running the model is set to run at 5AM every day. The weather API runs at 2AM, while the ingestion of attendance data runs at 10PM. Before training my model, there is some transformation that needs to be done. There are 7 general steps I take to do this:

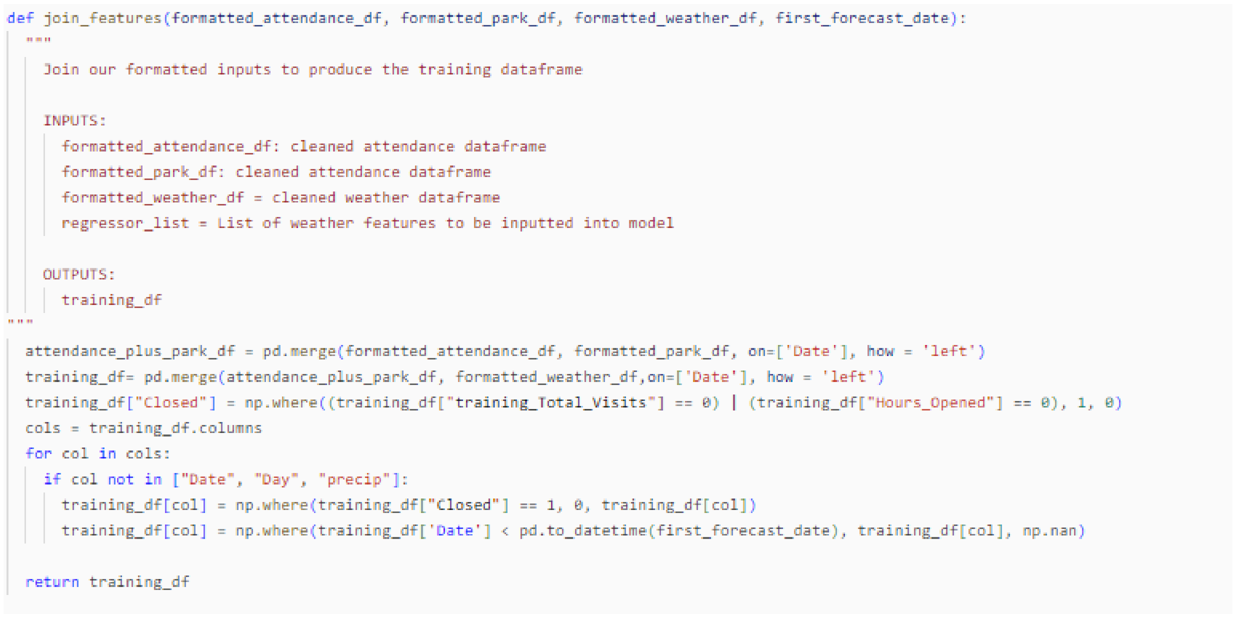
1. Import relevant packages (pandas, numpy, datetime, greykite, etc.)
2. Mount Azure Blob Storage (using storage account name & key)
3. Import historical attendance values from Blob Storage
4. Import historical weather values from Blob Storage
5. Import feature data from Blob Storage (Holidays, Events, etc.)
6. Create data dummies (multiple columns of binary regressors for one feature. Separate columns for rainy, clear, cloudy, etc. 1 for true, 0 for false) and merge to attendance dataframe (figure 11).



```
sub_weather = pd.get_dummies(sub_weather, columns = ["icon"])
sub_weather.columns = sub_weather.columns.astype("str").str.replace(" ","_")
sub_weather.columns = sub_weather.columns.astype("str").str.replace(",","")
sub_weather.columns = sub_weather.columns.astype("str").str.replace("-","_")

for i in sub_weather.columns:
    if i == "Date":
        sub_weather["Date"] = pd.to_datetime(sub_weather["Date"], format = "%Y-%m-%d")
    else:
        sub_weather[i] = sub_weather[i].astype("int64")

final_training = pd.merge(training_df, sub_weather, right_on = ['Date'], left_on =['Date'], how = 'left')
```

Figure 11: Weather Data transformation into weather dummies as model features. KSB(S10)

7. Format regressors and merge to training data (figure 12).

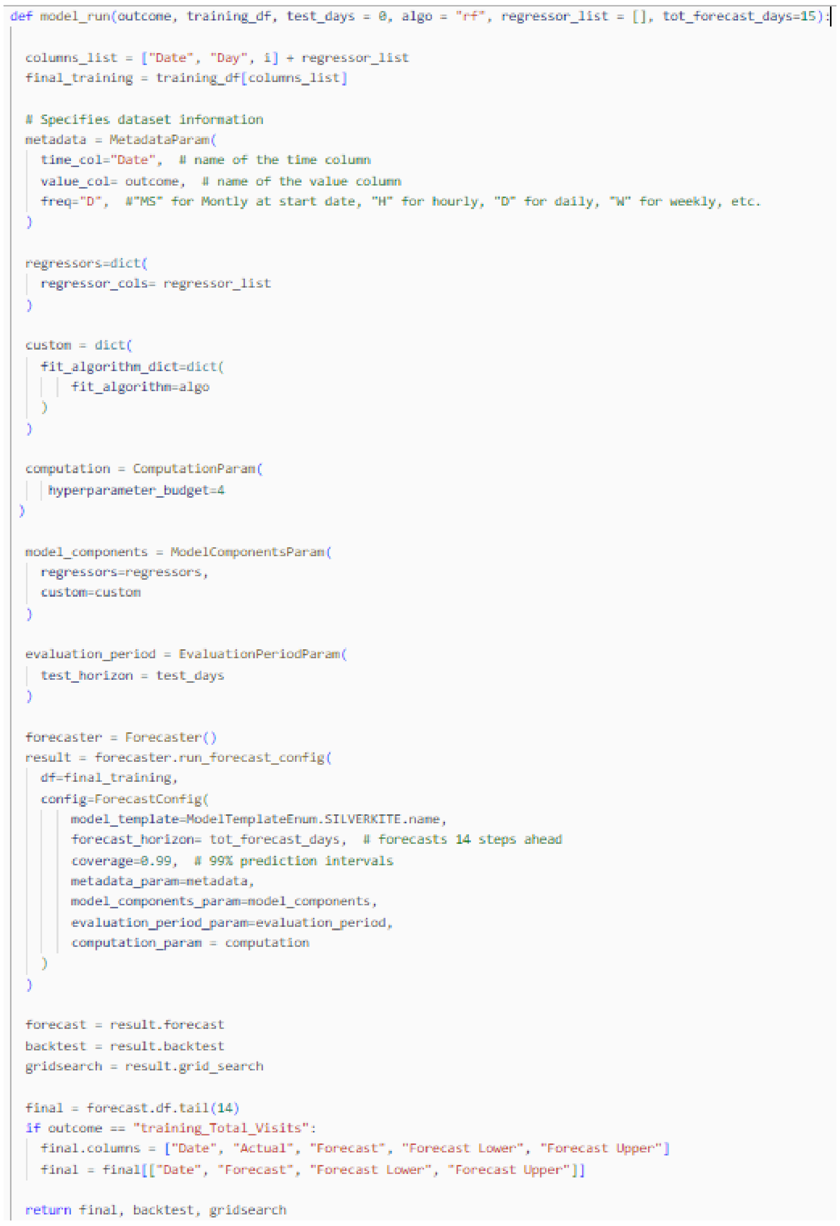


```
def join_features(formatted_attendance_df, formatted_park_df, formatted_weather_df, first_forecast_date):
    """
    Join our formatted inputs to produce the training dataframe

    INPUTS:
        formatted_attendance_df: cleaned attendance dataframe
        formatted_park_df: cleaned attendance dataframe
        formatted_weather_df = cleaned weather dataframe
        regressor_list = List of weather features to be inputted into model

    OUTPUTS:
        training_df
"""
    attendance_plus_park_df = pd.merge(formatted_attendance_df, formatted_park_df, on=['Date'], how = 'left')
    training_df= pd.merge(attendance_plus_park_df, formatted_weather_df,on=['Date'], how = 'left')
    training_df["Closed"] = np.where((training_df["training_Total_Visits"] == 0) | (training_df["Hours_Opened"] == 0), 1, 0)
    cols = training_df.columns
    for col in cols:
        if col not in ["Date", "Day", "precip"]:
            training_df[col] = np.where(training_df["Closed"] == 1, 0, training_df[col])
            training_df[col] = np.where(training_df['Date'] < pd.to_datetime(first_forecast_date), training_df[col], np.nan)

    return training_df
```

Figure 12: Code to join all features and training data together. KSB(S10)

I now have a data frame with 31 columns. 10 of these represent 10 separate channels of prebooks, and another 10 represent the prebooks for those channels 14 days in advance. The other 11 include the following columns: Date, Closed, Total Attendance, Events, Park Hours, School Holidays, Rainy, Clear, Cloudy, Partly Cloudy, Snow. Depending on the park, I may have precipitation or a different weather feature rather than these binary weather dummies. In order to make the model as quick as possible, I then separate the dataframe after each model run with the relevant columns. Starting with total attendance, I remove the other channels of historical attendance data. Each run after this is for predicting separate channels (Web Bookings, Annual Passes, School Bookings, etc.). There are a total of 11 runs, one for total attendance and 10 after that, one for each channel. Each run is stored in a dictionary with a specific name as the key and the data frame as the value. I then merge all the split channel runs into one to get a Total Prebooks forecast. This is another method to predict attendance, separate to the single model I ran at the start. I also take each separate run which I save within one big .csv file.

To run my model, I created a separate function (figure 13). The variables for the function are the outcome variable, which is the daily attendance; the training data frame, which I formatted earlier with all my regressors; the number of test days, which I set to 0 to ignore as I don't need any testing; the algorithm, which usually is set to be random forest; a regressor list, containing all relevant features; and the number of days to forecast. I currently run a 15 day forecast and in my output dashboards do not display the first day of the forecast. The reason for this is that my training data gets updated with one day of delay due to separate issues with the ticketing system. I therefore have to forecast one day early and then simply ignore the first day. Greykite has built in gridsearch which automatically tests multiple hyperparameters for the random forest. These hyperparameters can be overridden using the function hyperparameter_override. However, this isn't essential as my dataset is quite small and the model can therefore perform well even when testing many hyperparameters. The function returns a dataframe containing just the 14 days of forecast (using .tail()), as well as the results from the testing split and the gridsearch. These last two are for testing and are not required when I run my model. I currently retrain my model after every run.



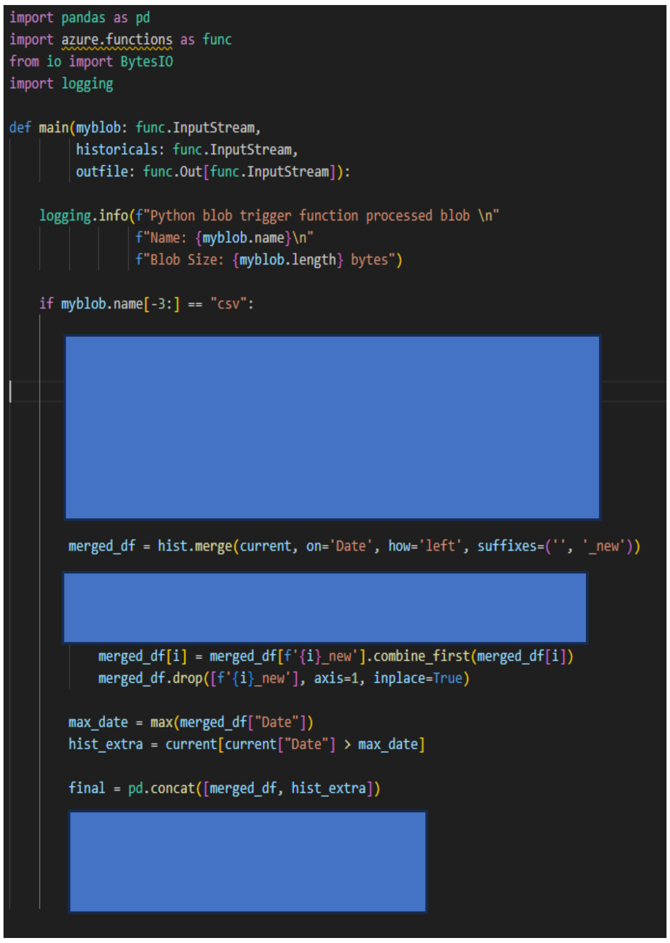
```python
def model_run(outcome, training_df, test_days = 0, algo = "rf", regressor_list = [], tot_forecast_days=15):

  columns_list = ["Date", "Day", i] + regressor_list
  final_training = training_df[columns_list]

  # Specifies dataset information
  metadata = MetadataParam(
    time_col="Date",  # name of the time column
    value_col= outcome,  # name of the value column
    freq="D",  #"MS" for Montly at start date, "H" for hourly, "D" for daily, "W" for weekly, etc.
  )

  regressors=dict(
    regressor_cols= regressor_list
  )

  custom = dict(
    fit_algorithm_dict=dict(
        fit_algorithm=algo
    )
  )

  computation = ComputationParam(
     hyperparameter_budget=4
 )

  model_components = ModelComponentsParam(
    regressors=regressors,
    custom=custom
  )

  evaluation_period = EvaluationPeriodParam(
    test_horizon = test_days
  )

  forecaster = Forecaster()
  result = forecaster.run_forecast_config(
    df=final_training,
    config=ForecastConfig(
        model_template=ModelTemplateEnum.SILVERKITE.name,
        forecast_horizon= tot_forecast_days,  # forecasts 14 steps ahead
        coverage=0.99,  # 99% prediction intervals
        metadata_param=metadata,
        model_components_param=model_components,
        evaluation_period_param=evaluation_period,
        computation_param = computation
    )
  )

  forecast = result.forecast
  backtest = result.backtest
  gridsearch = result.grid_search

  final = forecast.df.tail(14)
  if outcome == "training_Total_Visits":
    final.columns = ["Date", "Actual", "Forecast", "Forecast Lower", "Forecast Upper"]
    final = final[["Date", "Forecast", "Forecast Lower", "Forecast Upper"]]

  return final, backtest, gridsearch
```

Figure 13: Model Configuration using Greykite. KSB(K13, K14, S10, S11, S13)

The first method of prediction I use is a simple model using total attendance data at the parks to predict the next two weeks of attendance. The second is instead an aggregate of multiple models, ran one after the other. Each model predicts the total prebooks for a specific channel. These channels include Web, Annual Pass, Trade, Promotions, Corporate, Schools, etc. prebooks. By predicting these channels separately, I pick up on trends from each one which may be different from others. For example, annual pass bookings may stay more stable throughout the year, while school pre-bookings are much more prevalent right before summer break. Both methods use the same features and are run the same way. I currently display both methods in my dashboard and let the parks decide which one they prefer to use and trust more. I then save three files to Azure following these runs: TotalAttendance_Forecast (the first run), TotalPrebooks_Forecast (the sum of the last 10 runs) and SplitChannels_Forecast (the last 10 runs separated but kept in one file).

Azure Storage Trigger

Once the three forecast files are uploaded to the Blob storage, they trigger another Azure Function. This function takes in these files and update historical forecast files. Thanks to the naming conventions of both the current and historical forecast files, one function processes all three types contemporarily. Since I want the newest forecasted values to overwrite the older historical values, I couldn't simply use the .update() function but instead merged all the new forecasted dates, and then used the combine_first function to keep only the newest values and remove the old forecasts. Since this only keeps forecasts that had matching dates, I also appended the rows with new dates (hist_extra). I then save these historical .csv forecast files back to the Azure storage account. Snippets of this function are shown in figure 14.



```
import pandas as pd
import azure.functions as func
from io import BytesIO
import logging

def main(myblob: func.InputStream,
         historicals: func.InputStream,
         outfile: func.Out[func.InputStream]):

    logging.info(f"Python blob trigger function processed blob \n"
                 f"Name: {myblob.name}\n"
                 f"Blob Size: {myblob.length} bytes")

    if myblob.name[-3:] == "csv":

        [redacted]

        merged_df = hist.merge(current, on='Date', how='left', suffixes=('', '_new'))

        [redacted]
            merged_df[i] = merged_df[f'{i}_new'].combine_first(merged_df[i])
            merged_df.drop([f'{i}_new'], axis=1, inplace=True)

        max_date = max(merged_df["Date"])
        hist_extra = current[current["Date"] > max_date]

        final = pd.concat([merged_df, hist_extra])

        [redacted]
```

Figure 14: Azure function to add forecasts to historicals file using a storage trigger. KSB(S10)

Testing

All my time-series forecasts, before going into production, are tested for accuracy. Below is a diagram showing how my training data fits into one of my models. The data clearly fits very well (figure 15), but this may be misleading as I am only fitting training and not testing data, which may lead to target leakage.

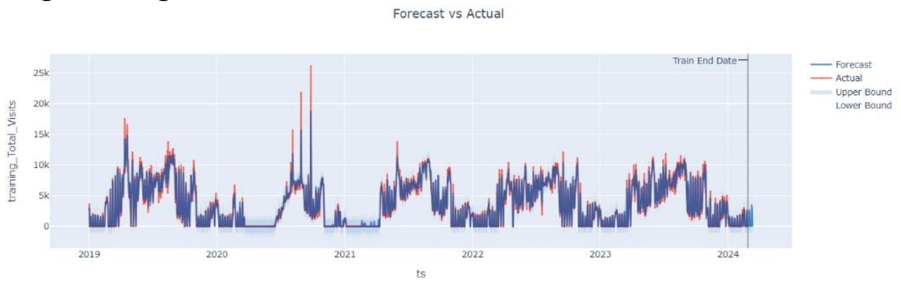


Figure 15: Model fit to training dataset. KSB(K13, K14, S10, S11, S13)

To get a more accurate depiction of the performance of the model, I then perform a backtest. Below is a 30 day backtest performed for Chessington World of Adventures (figure 16).

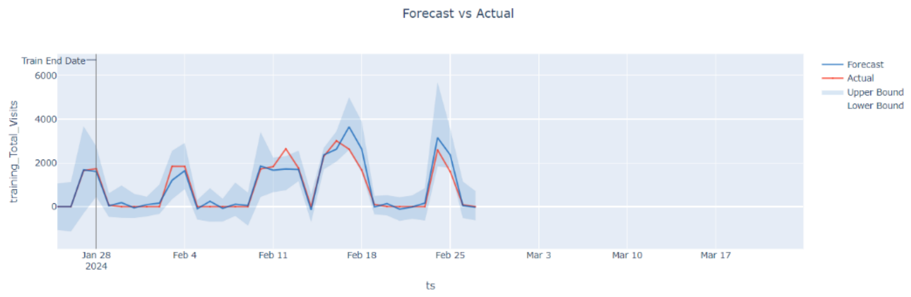
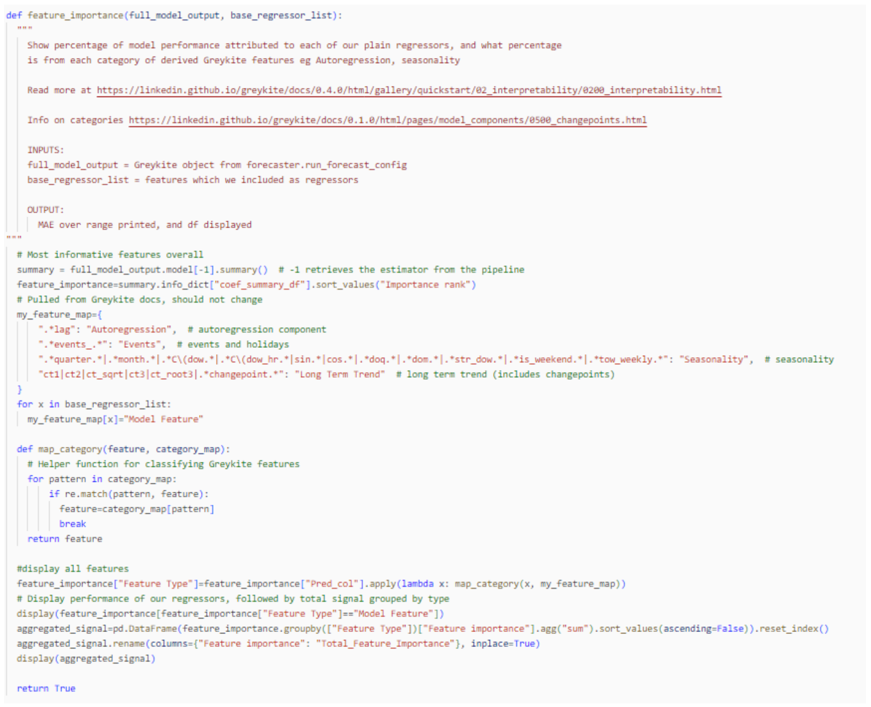


Figure 16: Model fit to testing dataset. KSB(K13, K14, S10, S11, S13)

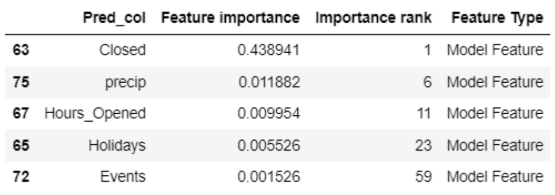
Once again, the forecasted attendance seems to follow the actual attendance quite well. I may however want to improve this accuracy further. To do this, I can rank the importance of each feature using the below function (figure 17).



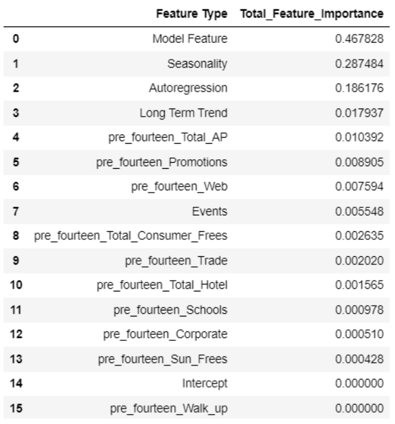
```python
def feature_importance(full_model_output, base_regressor_list):
  """
    Show percentage of model performance attributed to each of our plain regressors, and what percentage
    is from each category of derived Greykite features eg Autoregression, seasonality

    Read more at https://linkedin.github.io/greykite/docs/0.4.0/html/gallery/quickstart/02_interpretability/0200_interpretability.html

    Info on categories https://linkedin.github.io/greykite/docs/0.1.0/html/pages/model_components/0500_changepoints.html

    INPUTS:
    full_model_output = Greykite object from forecaster.run_forecast_config
    base_regressor_list = features which we included as regressors

    OUTPUT:
      MAE over range printed, and df displayed
"""
  # Most informative features overall
  summary = full_model_output.model[-1].summary()  # -1 retrieves the estimator from the pipeline
  feature_importance=summary.info_dict["coef_summary_df"].sort_values("Importance rank")
  # Pulled from Greykite docs, should not change
  my_feature_map={
      ".*lag": "Autoregression",  # autoregression component
      ".*events_.*": "Events",  # events and holidays
      ".*quarter.*|.*month.*|.*C\(dow.*|.*C\(dow_hr.*|sin.*|cos.*|.*doq.*|.*dom.*|.*str_dow.*|.*is_weekend.*|.*tow_weekly.*": "Seasonality",  # seasonality
      "ct1|ct2|ct_sqrt|ct3|ct_root3|.*changepoint.*": "Long Term Trend"  # long term trend (includes changepoints)
  }
  for x in base_regressor_list:
    my_feature_map[x]="Model Feature"

  def map_category(feature, category_map):
    # Helper function for classifying Greykite features
    for pattern in category_map:
        if re.match(pattern, feature):
          feature=category_map[pattern]
          break
    return feature

  #display all features
  feature_importance["Feature Type"]=feature_importance["Pred_col"].apply(lambda x: map_category(x, my_feature_map))
  # Display performance of our regressors, followed by total signal grouped by type
  display(feature_importance[feature_importance["Feature Type"]=="Model Feature"])
  aggregated_signal=pd.DataFrame(feature_importance.groupby(["Feature Type"])["Feature importance"].agg("sum").sort_values(ascending=False)).reset_index()
  aggregated_signal.rename(columns={"Feature importance": "Total_Feature_Importance"}, inplace=True)
  display(aggregated_signal)

  return True
```

Figure 17: Python function to create and display model feature importance. KSB(K13, K14, S10, S11)

For this specific model (Chessington world of adventures), the output from this test produced the following (figure 18):



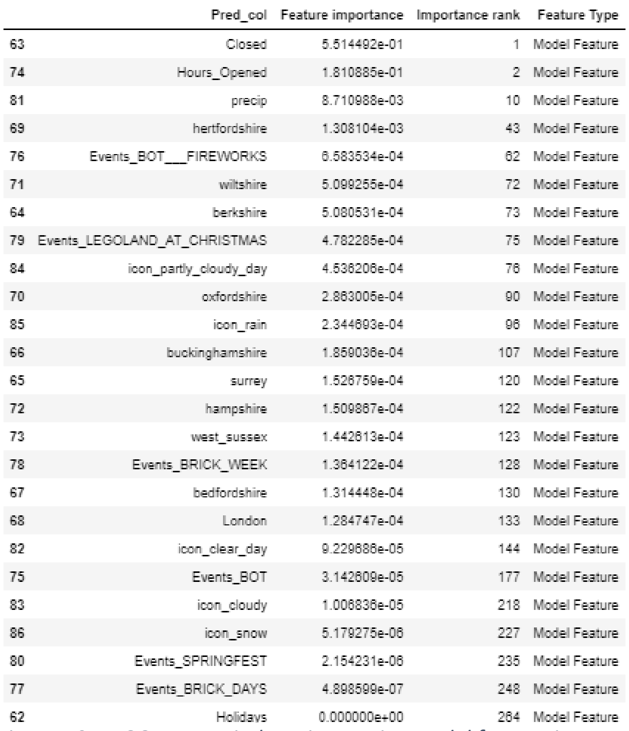
	Pred_col	Feature importance	Importance rank	Feature Type
63	Closed	0.438941	1	Model Feature
75	precip	0.011882	6	Model Feature
67	Hours_Opened	0.009954	11	Model Feature
65	Holidays	0.005526	23	Model Feature
72	Events	0.001526	59	Model Feature



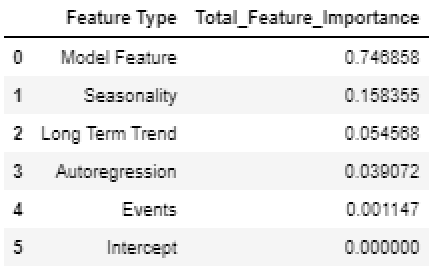
	Feature Type	Total_Feature_Importance
0	Model Feature	0.467828
1	Seasonality	0.287484
2	Autoregression	0.186176
3	Long Term Trend	0.017937
4	pre_fourteen_Total_AP	0.010392
5	pre_fourteen_Promotions	0.008905
6	pre_fourteen_Web	0.007594
7	Events	0.005548
8	pre_fourteen_Total_Consumer_Frees	0.002635
9	pre_fourteen_Trade	0.002020
10	pre_fourteen_Total_Hotel	0.001565
11	pre_fourteen_Schools	0.000978
12	pre_fourteen_Corporate	0.000510
13	pre_fourteen_Sun_Frees	0.000428
14	Intercept	0.000000
15	pre_fourteen_Walk_up	0.000000

Figure 18: Chessington World of Adventures time-series model feature importance. Overall feature rank (Right) Model feature rank (Left). KSB(K13, K14, S11, S13)

As we can see here, the model features account for 47% of the model performance. These are features I added to the model, rather than in built seasonality and autoregression that greykite handles. When I look at the feature importance within the model features, however, I can see that a big chunk of this is the "Closed" regressor (44%), which simply tells me whether the park is closed on any given day. Weather does not affect the model as much as I had hoped and have been working on improving my feature engineering. For now, however, the model works well and meets the accuracy requirements. Another example of feature importance can be seen below, for Legoland Windsor (figure 19):



	Pred_col	Feature importance	Importance rank	Feature Type
63	Closed	5.514492e-01	1	Model Feature
74	Hours_Opened	1.810885e-01	2	Model Feature
81	precip	8.710988e-03	10	Model Feature
69	hertfordshire	1.308104e-03	43	Model Feature
76	Events_BOT___FIREWORKS	6.583534e-04	62	Model Feature
71	wiltshire	5.099255e-04	72	Model Feature
64	berkshire	5.080531e-04	73	Model Feature
79	Events_LEGOLAND_AT_CHRISTMAS	4.782285e-04	75	Model Feature
84	icon_partly_cloudy_day	4.536206e-04	76	Model Feature
70	oxfordshire	2.863005e-04	90	Model Feature
85	icon_rain	2.344693e-04	96	Model Feature
66	buckinghamshire	1.859036e-04	107	Model Feature
65	surrey	1.526759e-04	120	Model Feature
72	hampshire	1.509867e-04	122	Model Feature
73	west_sussex	1.442613e-04	123	Model Feature
78	Events_BRICK_WEEK	1.384122e-04	128	Model Feature
67	bedfordshire	1.314448e-04	130	Model Feature
68	London	1.284747e-04	133	Model Feature
82	icon_clear_day	9.229686e-05	144	Model Feature
75	Events_BOT	3.142609e-05	177	Model Feature
83	icon_cloudy	1.006836e-05	218	Model Feature
86	icon_snow	5.179275e-06	227	Model Feature
80	Events_SPRINGFEST	2.154231e-06	235	Model Feature
77	Events_BRICK_DAYS	4.898599e-07	248	Model Feature
62	Holidays	0.000000e+00	264	Model Feature



	Feature Type	Total_Feature_Importance
0	Model Feature	0.746858
1	Seasonality	0.158355
2	Long Term Trend	0.054568
3	Autoregression	0.039072
4	Events	0.001147
5	Intercept	0.000000

Figure 19: LEGOLAND Windsor time-series model feature importance. Overall feature rank (Left) Model feature rank (Right). KSB(K13, K14, S11, S13)

You can see here that 75% of the model accuracy can be attributed by the regressors I feed it. The features I have picked are quite different to those in the previous model. In this case, seasonality is less important and the operating hours of the park (hours_opened) accounts for 18% of the model accuracy, which is much higher compared to the previous model (0.9%). However, the first model still outperforms the second, therefore proving that Chessington may be easier to predict on seasonality alone. I can take a look at this using some visualization for other parks, such as LEGOLAND New York (figure 20).

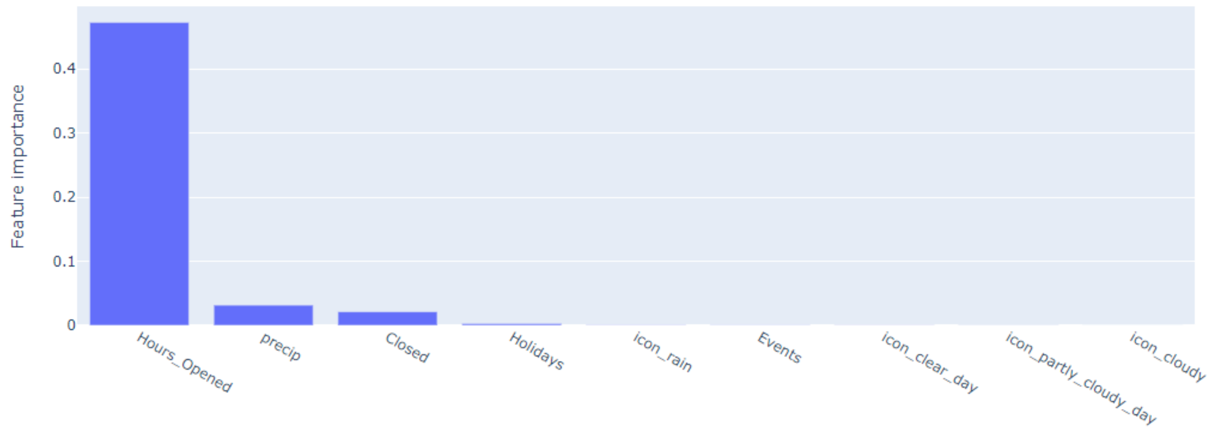
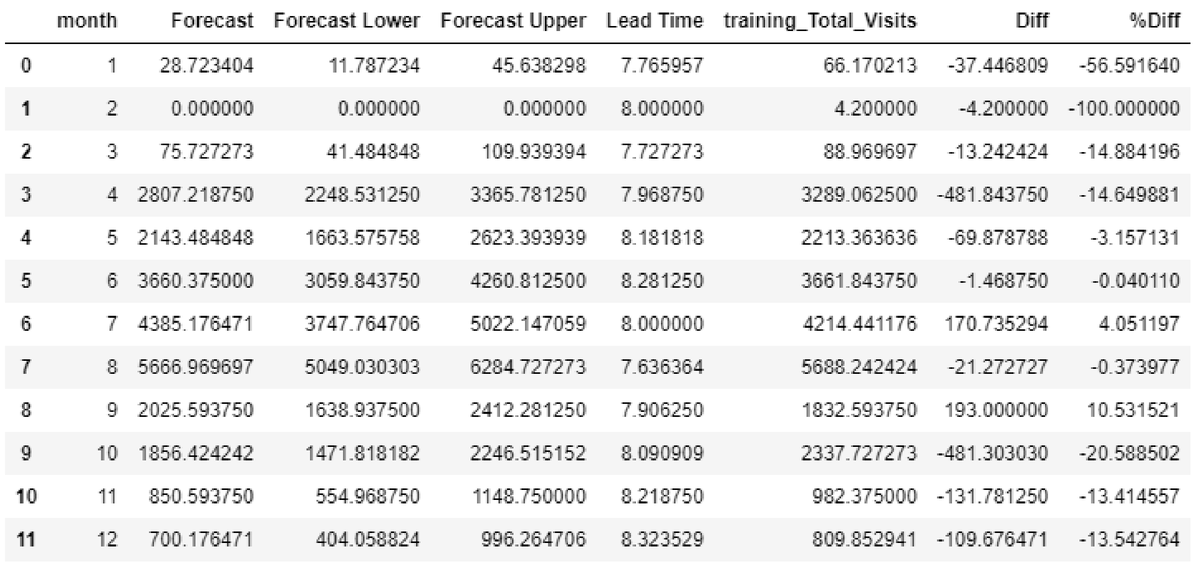


Figure 20: LEGOLAND New York time-series model feature importance, visualized. KSB(K13, K14, S11, S13)

Here, I can see that the operating hours of the park have a massive impact on the model, moreso than any other added feature in the model. The other 40% is related to greykite built in features, such as seasonality and autoregression. I can see from this visuals that the park events are not affecting the model at all. I therefore decided to get rid of them altogether, which didn't affect my accuracy.

To build a dashboard, I also needed to show the model accuracy throughout a given time period. I was asked to produce one year of backtesting and show the accuracy of the model since then. To do this, I created a loop to forecast two weeks at a time. I could also run a whole year forecast in one run, but this would not accurately depict how well the model performs, as accuracy will fall the longer a forecast is set to run (due to data availability, seasonality trends and autoregression). Below is the backtest for a whole year, split by month, with MAE and % accuracy for each month for LEGOLAND New York (table 2):



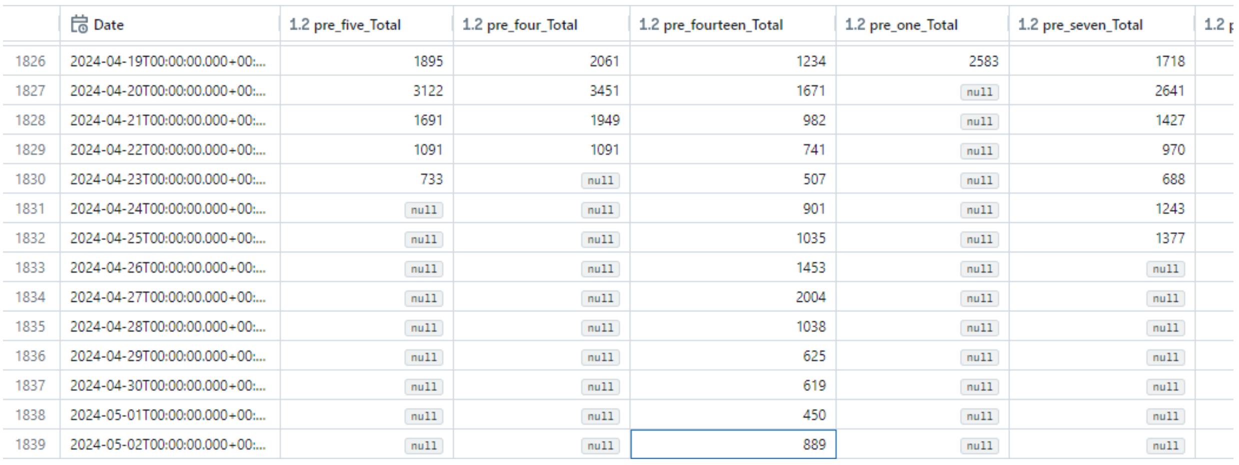
	month	Forecast	Forecast Lower	Forecast Upper	Lead Time	training_Total_Visits	Diff	%Diff
0	1	28.723404	11.787234	45.638298	7.765957	66.170213	-37.446809	-56.591640
1	2	0.000000	0.000000	0.000000	8.000000	4.200000	-4.200000	-100.000000
2	3	75.727273	41.484848	109.939394	7.727273	88.969697	-13.242424	-14.884196
3	4	2807.218750	2248.531250	3365.781250	7.968750	3289.062500	-481.843750	-14.649881
4	5	2143.484848	1663.575758	2623.393939	8.181818	2213.363636	-69.878788	-3.157131
5	6	3660.375000	3059.843750	4260.812500	8.281250	3661.843750	-1.468750	-0.040110
6	7	4385.176471	3747.764706	5022.147059	8.000000	4214.441176	170.735294	4.051197
7	8	5666.969697	5049.030303	6284.727273	7.636364	5688.242424	-21.272727	-0.373977
8	9	2025.593750	1638.937500	2412.281250	7.906250	1832.593750	193.000000	10.531521
9	10	1856.424242	1471.818182	2246.515152	8.090909	2337.727273	-481.303030	-20.588502
10	11	850.593750	554.968750	1148.750000	8.218750	982.375000	-131.781250	-13.414557
11	12	700.176471	404.058824	996.264706	8.323529	809.852941	-109.676471	-13.542764

Table 2: LEGOLAND New York accuracy per month, MAE (Diff column) and MAPE (%Diff column). KSB(K13, K14, S11, S13)

The average forecast for the whole year is 1962, while average actual attendance is 2042, giving me a 3.9% error rate. This is a great error rate, as I usually aim for sub 10% to consider a model ready for production. The model is continuously improving thanks to the increase in data availability, meaning what I have produced now is highly appreciated. Because of these results, improving feature engineering (such as the Events regressor) for this park is of very low priority compared to improving other parks, such as Chessington World of Adventures.

Prebooking Curves

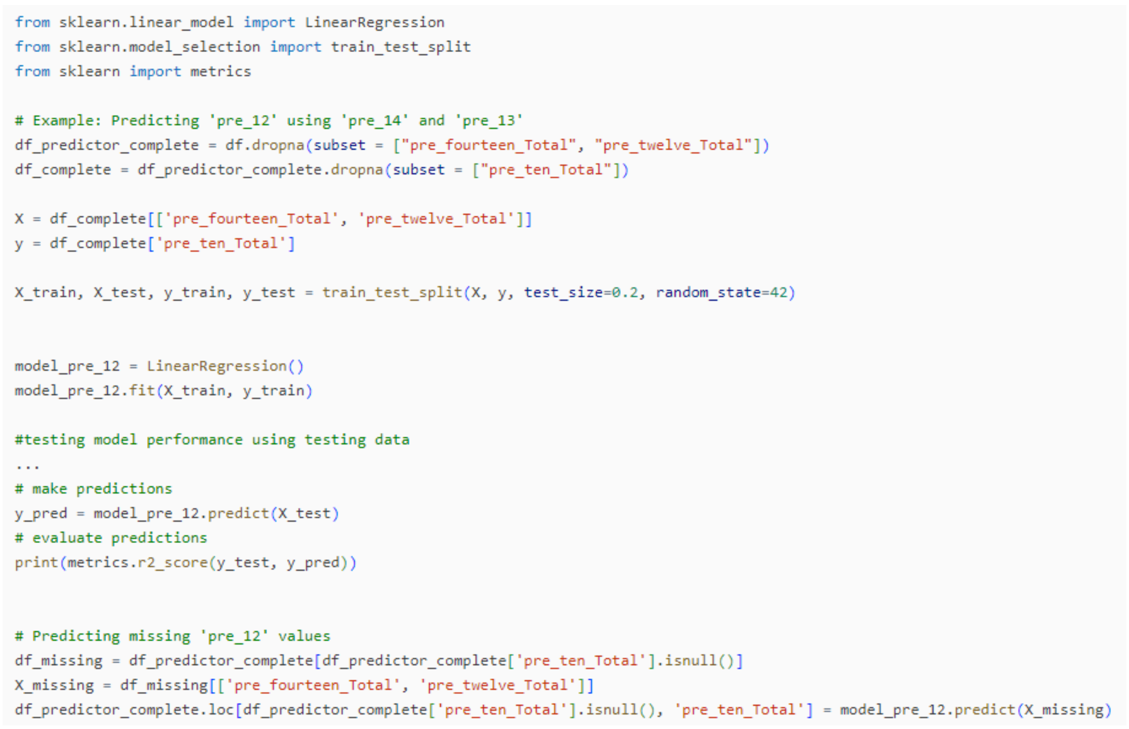
My raw dataset contains prebooking data x days in advance. Since I forecast 14 days ahead, to get a complete dataset, I would only be able to use prebooks 14 days ahead of an event date. However, this is leaving a lot of relevant data out which could significantly improve the model. This is what my prebooking data looks like when I import prebooking data 14,12, 10 etc days in advance, rather than just 14 (table 3):



	Date	1.2 pre_five_Total	1.2 pre_four_Total	1.2 pre_fourteen_Total	1.2 pre_one_Total	1.2 pre_seven_Total	1.2 p
1826	2024-04-19T00:00:00.000+00:...	1895	2061	1234	2583	1718	
1827	2024-04-20T00:00:00.000+00:...	3122	3451	1671	null	2641	
1828	2024-04-21T00:00:00.000+00:...	1691	1949	982	null	1427	
1829	2024-04-22T00:00:00.000+00:...	1091	1091	741	null	970	
1830	2024-04-23T00:00:00.000+00:...	733	null	507	null	688	
1831	2024-04-24T00:00:00.000+00:...	null	null	901	null	1243	
1832	2024-04-25T00:00:00.000+00:...	null	null	1035	null	1377	
1833	2024-04-26T00:00:00.000+00:...	null	null	1453	null	null	
1834	2024-04-27T00:00:00.000+00:...	null	null	2004	null	null	
1835	2024-04-28T00:00:00.000+00:...	null	null	1038	null	null	
1836	2024-04-29T00:00:00.000+00:...	null	null	625	null	null	
1837	2024-04-30T00:00:00.000+00:...	null	null	619	null	null	
1838	2024-05-01T00:00:00.000+00:...	null	null	450	null	null	
1839	2024-05-02T00:00:00.000+00:...	null	null	889	null	null	

Table 3: Prebooking data extract for LEGOLAND New York. KSB(K13, K14, S10)

In this case, the first day of prediction is the 19th of April 2024. For the 14 day forecast, I have available all the 14 day advanced prebooks (pre_fourteen_Total), but will have a lot of missing values for all other days. To make this data usable, I created a linear regression which takes the historical prebooking data and predicts the next day. For example, in this case, I trained a model on all the data prior to the 19th of April. I then created a linear regression for each column of advanced prebooks. For pre_twelve_Total, I can only use one predictor (pre_fourteen). For pre_one_Total, I can instead use 14,12,10,7,6,5,4,3 and 2 days out as predictor data. Below is an example of a linear regression to predict pre_ten_Total using pre_twelve and pre_fourteen (figure 21):

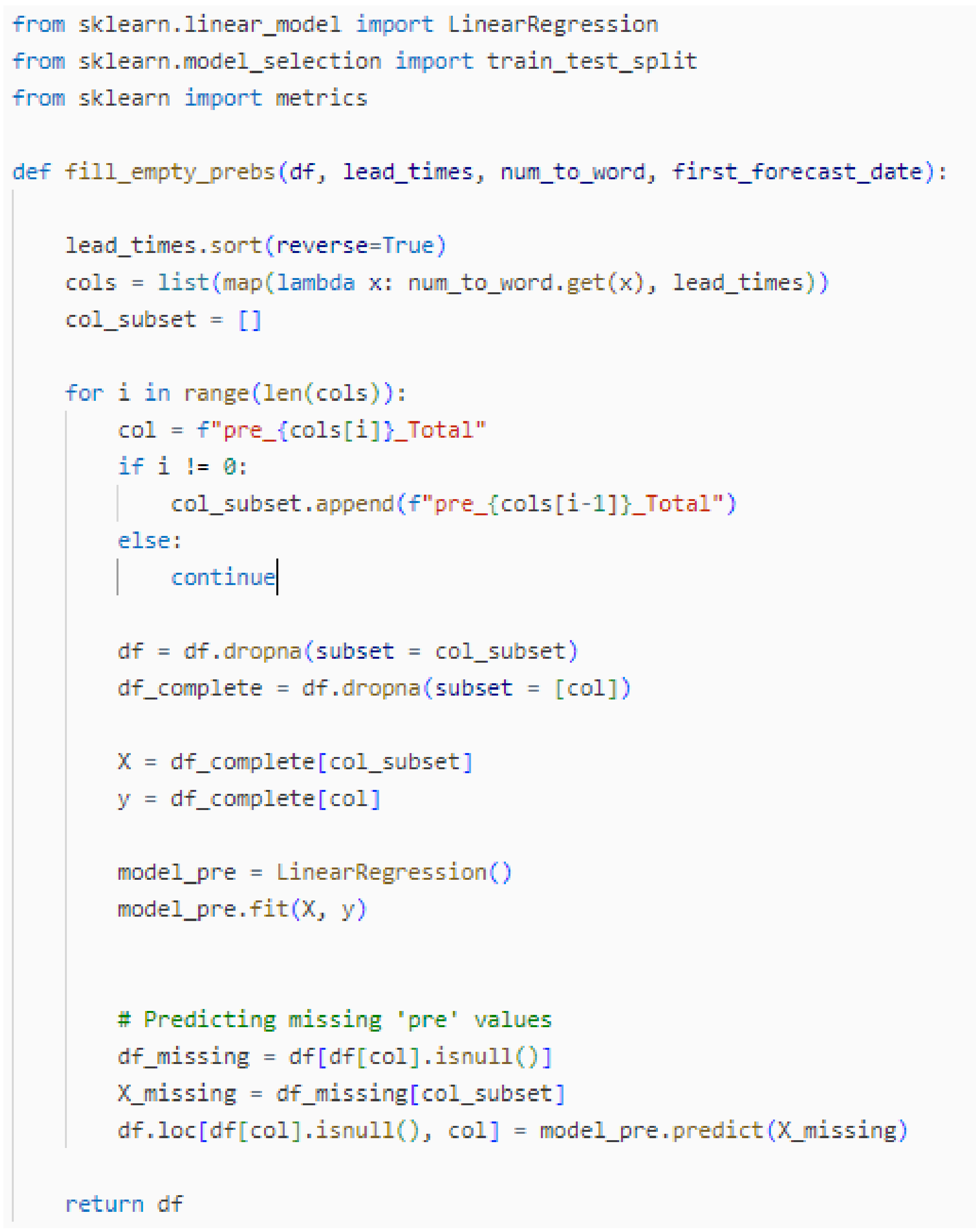


```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Example: Predicting 'pre_12' using 'pre_14' and 'pre_13'
df_predictor_complete = df.dropna(subset = ["pre_fourteen_Total", "pre_twelve_Total"])
df_complete = df_predictor_complete.dropna(subset = ["pre_ten_Total"])

X = df_complete[['pre_fourteen_Total', 'pre_twelve_Total']]
y = df_complete['pre_ten_Total']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


model_pre_12 = LinearRegression()
model_pre_12.fit(X_train, y_train)

#testing model performance using testing data
...
# make predictions
y_pred = model_pre_12.predict(X_test)
# evaluate predictions
print(metrics.r2_score(y_test, y_pred))


# Predicting missing 'pre_12' values
df_missing = df_predictor_complete[df_predictor_complete['pre_ten_Total'].isnull()]
X_missing = df_missing[['pre_fourteen_Total', 'pre_twelve_Total']]
df_predictor_complete.loc[df_predictor_complete['pre_ten_Total'].isnull(), 'pre_ten_Total'] = model_pre_12.predict(X_missing)
```

Figure 21: Linear regression model to predict prebooks and fill in missing data for time-series model. KSB(K13, K14, S11, S13, B2, B6)

This code is taking all the clean complete training and testing data in my dataset and using it to train a linear regression. After fitting the model, I used the testing split to evaluate the r2 of the model, which comes out to be 0.998 with a MAPE of ~5%. This is extremely promising, and tells me that this methodology will give very accurate predictions. This score will get slightly worse as we get closer to pre_one_Total. This is because some values were predicted using 8 prior predictions, and each consecutive prediction introduces some additional error. Below is the complete function used to predict all advanced prebook columns (figure 22).

This linear regression, used as an imputation method for missing prebooking numbers, was able to improve the model, in some cases, nearly two fold. Figure 23 shows an example of a backtest for Alton Towers prior to the addition of the imputed prebooks.



```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

def fill_empty_prebs(df, lead_times, num_to_word, first_forecast_date):

    lead_times.sort(reverse=True)
    cols = list(map(lambda x: num_to_word.get(x), lead_times))
    col_subset = []

    for i in range(len(cols)):
        col = f"pre_{cols[i]}_Total"
        if i != 0:
            col_subset.append(f"pre_{cols[i-1]}_Total")
        else:
            continue

        df = df.dropna(subset = col_subset)
        df_complete = df.dropna(subset = [col])

        X = df_complete[col_subset]
        y = df_complete[col]

        model_pre = LinearRegression()
        model_pre.fit(X, y)


        # Predicting missing 'pre' values
        df_missing = df[df[col].isnull()]
        X_missing = df_missing[col_subset]
        df.loc[df[col].isnull(), col] = model_pre.predict(X_missing)

    return df
```

Figure 22: Linear regression model functionalized to fill training data for foresight model. KSB(K13, K14, S11, S13, B2, B6)

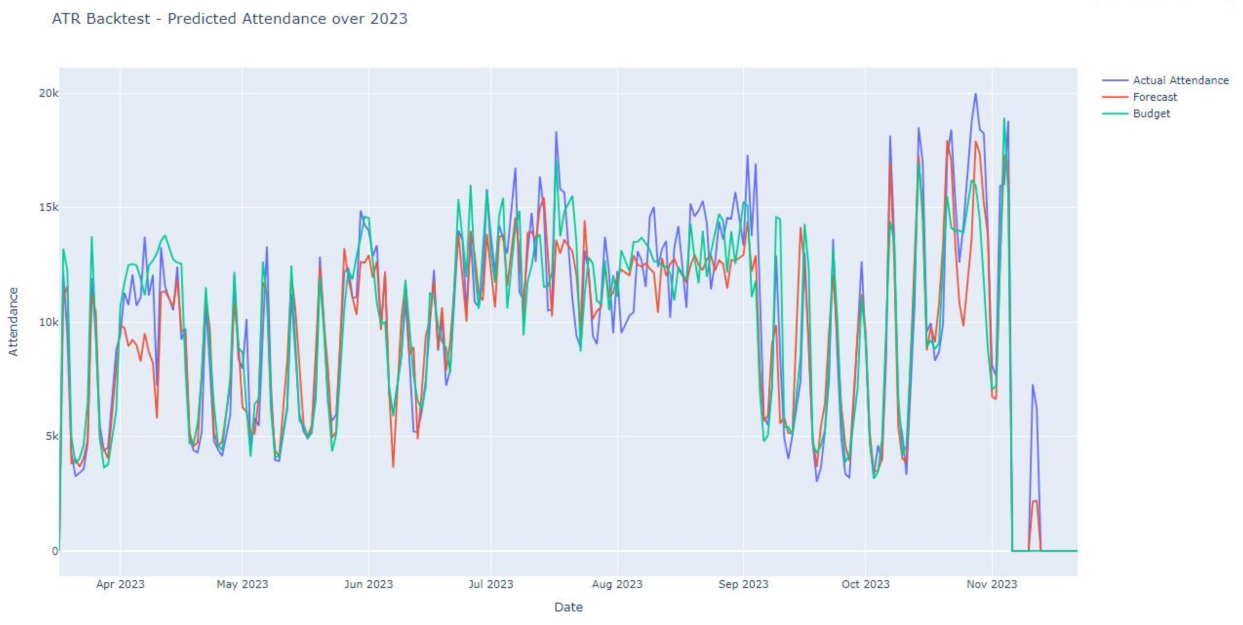


Figure 23: Predicted Attendance for Alton Towers, plotted against time with park budget and actual attendance, without imputed prebooks. KSB(K13, K14, S10, S11, S13)

Here, the model stands at an accuracy of 13.47% MAPE from the actual values for 2023. After I feed in the imputed numbers, this is the result (figure 24):

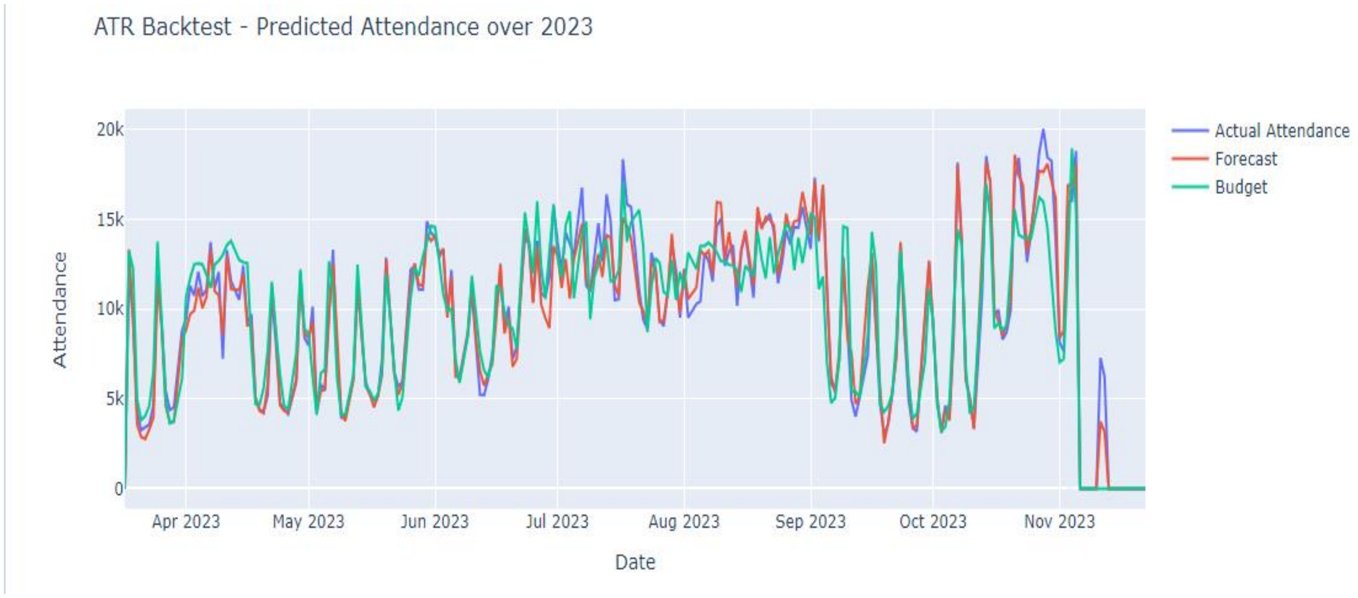


Figure 24: Predicted Attendance for Alton Towers, plotted against time with park budget and actual attendance, with imputed prebooks. KSB(K13, K14, S10, S11, S13, B2, B6)

Here, the MAPE from the actual values for 2023 is 6.06%, an improvement of over 100% compared to the previous model.

Dashboard Automation

Output Data

The data for the dashboard is a combination of exports pulled from the model outputs in Databricks, as well as historical attendance data that is saved after each run of the model (this includes prebook values and total attendance values) which I then used to assess the accuracy of the model. All the tables referenced as model outputs are saved as .csv files on an Azure storage account. Table 4 has a comprehensive review of each piece of raw data that will later get imported into a dashboard.

DATA SOURCE	DESCRIPTION	SCHEMA
TOTATTENDANCE_FORECAST	Total forecasted attendance (model output)	Columns: “Date”, “forecast”, “forecast_lower”, “forecast_upper”
TOTPREBOOKS_FORECAST	Total forecasted prebook purchases (model output)	Same as above
SPLITCHANNELS_FORECAST	Forecasted prebook purchases split by channel (model output)	Same as above with repeating last three forecasts columns for each channel.
HISTORICAL_TOTATTENDANCE_FORECAST	Historical attendance forecasts (post-transformation)	Columns: “Date”, “Actuals”, “Forecast”, “Forecast Lower”, “Forecast Upper”
HISTORICAL_TOTPREBOOKS_FORECAST	Historical prebook forecasts (post_transformation)	Same as above
HISTORICAL_SPLITCHANNELS_FORECAST	Historical prebook forecasts split by channel (post_transformation)	Same as above with repeating last three forecasts columns for each channel.
HISTORICAL_ATTENDANCE	Contains historical values for prebooks split by channel and totals (with no forecasts)	Columns: “Date”, “Total”, “Web”, “AP”, “Promotions”, “Hotel”, “Trade”, “Schools”, “Corporate”, “Walk up”, “Consumer Frees”, “Sun Frees”
FISCAL_YEAR	A dictionary to convert dates in fiscal year, period and week for financial team.	Columns: “Date”, “FiscalPeriod”, “FiscalWeek”, “FiscalYear”
PARK_BUDGET	Range of dates from 2022 to 2023 containing set budget for park attendance. Used to compare my model vs current way of working at Chessington	Columns: “Date”, “Budget”

Table 4: Description of each dashboard inputs, along with their schema. All park dashboards follow the same data model, KSB(S9)

Connecting to data

Connecting to the .csv files was a simple process, using the built in “Get Data” function in Power BI (figure 25A, see appendix). I connected directly to the Azure storage account holding all the data. The storage is split into containers, one for each country. Within each container are separate folders for each park, and within each park folder are four sub-folders:

- “Automation Files”: contains advanced prebooks, park budget, fiscal year, historical attendance, package codes & weather conditions.
- “Current Forecasts”: contains TotAttendance, TotPrebooks & SplitChannels forecasts.
- “Historicals”: contains one historical data file per forecasting type.

- "EmailTXT": mentioned earlier, only used as a storage for .txt files created by the logic app in order for the Azure Function trigger to work properly.

Using Power Query, I can import a whole park folder from the specific country's container in my azure storage account (figure 26). By doing this, I can easily work on all the files I need that are within this folder. These were transformed to properly fit the dashboard requirements.

Data Transformation in Power Query

The first step in transforming my data was making sure that all my tables had proper headers and the correct data types. This model forecasts attendance with decimal numbers, so all the forecasted numbers were converted from decimal numbers to whole numbers (since you can't have a decimal number as attendance). Next, some of the date columns were not automatically detected due to their long format. I therefore changed the data type to datetime, and then date. Most other data types were correctly predicted. Most column headers were then renamed to something more sensible, such as "ts" to "Date" or "forecast_lower" to "Forecast Lower". park_budget and fiscal_year were already formatted correctly after these adjustments. The processes used to transform the rest of the data sources are listed below:

TotAttendance & TotPrebook Forecasts

For these data sources (figure 28A, see appendix), I merged the park_budget table onto "Date" to get budget values for each of the upcoming forecasted dates using a left-outer join. I then removed the "actual" column (which is automatically generated as a blank column in my model by the python library I use). The result is a table with 5 columns, as seen in figure 28B (see appendix).

Historical Attendance

This data source was by far the least formatted, as it was still in its raw database form and therefore required a lot of transformation. After filtering through files and expanding tables, I am left with 12 columns of data (figure 27A). The columns "Flash_Data_Source", "Currency", "AA1750" and "AA7000" were a;; removed as they are not of interest. I then filtered out all attraction codes except "12025" (which is the code for Chessington World of Adventures"). In doing this, an error was raised that one of the codes was not a whole number and therefore the filter would not work, so I simply used the "Replace Errors" built-in function after changing data type to whole number. I can also filter out the "Budget" rows since we only need actual values.

Now that the rows are properly filtered, I can remove columns “Attraction” and “Scenario” as well. The first three columns (“year”, “period”, “week”) represent a unique date in a fiscal format, since this data is pulled from the financial database. Using the fiscal_year data source, I merged calendar dates to the table using all three columns matching with a left-outer join and removed the fiscal format columns. “Sales Channel” represents codes for how the ticket was sold (this may be through a promotion, through a school booking, etc). Before merging channel_types to get the actual channel names from these codes, I had to trim the codes column in both tables as some codes had some whitespace which was preventing a successful merge. After merging the channel names and removing the channel codes, I could now merge the channel IDs from the newly created table from the last step, and then removed the channel names column. I then created a new table referencing the table we have just formatted. Here, I grouped columns “Actuals” and “Revenue” (formerly “AA1000” and “AA2000”) by “Date” and “Sales Channel”, therefore giving me total attendance and revenue per date split by channel (figure 27B). Finally, I referenced this table to create a new one with only total attendance values (therefore grouping only by date and summing up all channels together, figure 27C).

Historical TotAttendance & Historical TotPrebooks

These data sources share an identical format but use two different methods to obtain values (figure 29A). I therefore had to add a custom column to each table called “Method”, which represents whether the table used the “attendance” method or the “prebook” method. I then appended the latter to the former to create a big table containing all values. The reasoning for doing this rather than creating a relationship is to later create visuals with both methods which can be selected using slicers. I then merged the actual total attendance values from the transformed and cleaned blob storage data table using a left-outer join (final result in figure 29B).

SplitChannels & Historical SplitChannels

Before working on the channels forecast table, I needed a dictionary for channels. I therefore manually created a table with two columns: channel ID and channel name (figure 30).

For the channels forecast, I started with a wide data format, where I had a row with unique dates, and then three columns per channel (forecast, forecast lower bound and forecast upper bound). I first unpivoted all the columns other than date. I then split the column formed from column names in order to get a separate column with values: forecast, forecast_lower and forecast_upper. I then repivoted this new column to get new values separated by forecast boundaries in three columns, and separated by channel using a channels column. Finally, I merged the dictionary table for channels I had created earlier, kept the channel IDs and removed the names for a cleaner dataset (figure 31, see appendix).

Relationships

Before applying all these changes, some tables are disabled from loading, as they won't be needed in the visualization and thus allowed for quicker loading times. The tables I kept in this model, which have now been renamed to something more sensible, are as follow:

- TotPrebooks: "Prebooks Forecast"
- TotAttendance: "Attendance Forecast"
- Historical SplitChannels: "Historical Channel Forecasts"
- Historical TotAttendance & TotPrebooks: "Historical Totals Forecast"
- SplitChannels: "Split Channels Forecast"
- Codes: dictionary to retrieve channel name from channel ID in other tables
- Dates: table of dates created with DAX (figure 32, see appendix)
- Weather: weather description per day
- Fiscal Calendar: fiscal calendar for reference

After applying all the changes to the dataset, relationships are created between tables to get the desired output in the visuals. I used a sort of "Kimball style" relationship between the data tables. To do this, a "Dates" table is created using DAX expressions. All the tables are then connected to the dates table with one to one or many to one relationships. The channels dictionary table was also connected using one to many relationships to all tables containing channel IDs. The final output of these relationships can be visualized in figure 33 (see appendix).

Measures and DAX Columns

There are various other calculated columns created in some of the tables for better calculations, as well as some measures to display on cards throughout the dashboard. These are listed below:

- Dates Table:

- Past Dates (column): This is a Boolean column showing whether each date is part of previous forecasts or the upcoming forecast. This is used to filter out upcoming dates in the historical forecast visuals and other measures.

```
Past Dates =

VAR last_day = min(Forecast[Date]) - 1

VAR Prev = IF(Dates[Date] <= last_day, TRUE, FALSE) return Prev
```

- Forecast & Prebook Forecast Table (figure 34A, see appendix):
    - Forecast Tier (column): This is the attendance tier calculated using the forecasted attendance numbers.

```
ForecastTier = IF('Forecast'[Forecast]<4800,"Super Off Peak", IF('Forecast'[Forecast]<8400,"Off Peak",IF('Forecast'[Forecast]<10800,
"Peak","Super Peak")))
```

- Budget Tier (column): Same as forecast tier but for the pre-set park budget

```
Recommendations =
IF(Forecast[ForecastTier] = Forecast[BudgetTier], "--", "Adjust to "&Forecast[ForecastTier])
```

- Super off-peak, off-peak, peak, super peak (measures): Represent each attendance tier as their maximum values, used in visual as lines (e.g. for Chessington: super peak = 12000, peak = 10400, off peak = 8400, super off peak = 4800).
- Recommendations (column): A string showing recommendations based on set tier and forecasted tier

```
Recommendations =
IF(Forecast[ForecastTier] = Forecast[BudgetTier], "--", "Adjust to "&Forecast[ForecastTier])
```

- Past Channel Forecast:
    - Difference (column): Absolute difference between forecasted attendance and actual attendance

```
Difference = ABS('Past Channel Forecasts'[Actuals] - 'Past Channel Forecasts'[forecast])
```

- Past Forecasts (figure 34B, see appendix):
    - Actual Tier & Forecast Tier (columns): Same as previous tier columns, based on attendance
    - Difference (column): Same as previous difference column
    - Accuracy (column): Boolean column. True if actual and forecast tier match, otherwise false.

```
Accuracy = IF('Past Forecasts'[Actual Tier] = 'Past Forecasts'[Forecast Tier], TRUE, FALSE)
```

- Tier Accuracy % (measure): measures percentage of accurate dates using “Accuracy” column.

```
Tier Accuracy % = DIVIDE(CALCULATE(COUNTROWS('Past Forecasts'), 'Past Forecasts'[Accuracy] = TRUE), COUNTROWS('Past Forecasts'))
```

This built dataset was then published to a workspace on our Power BI platform prior to any visualization, and the empty report that gets generated was removed. This is common practice at Merlin to allow for others to connect to the same dataset and create their own dashboards if needed. I therefore created a new .pbix file and connected to the dataset I created earlier. The visualization is described in detail in the next section.

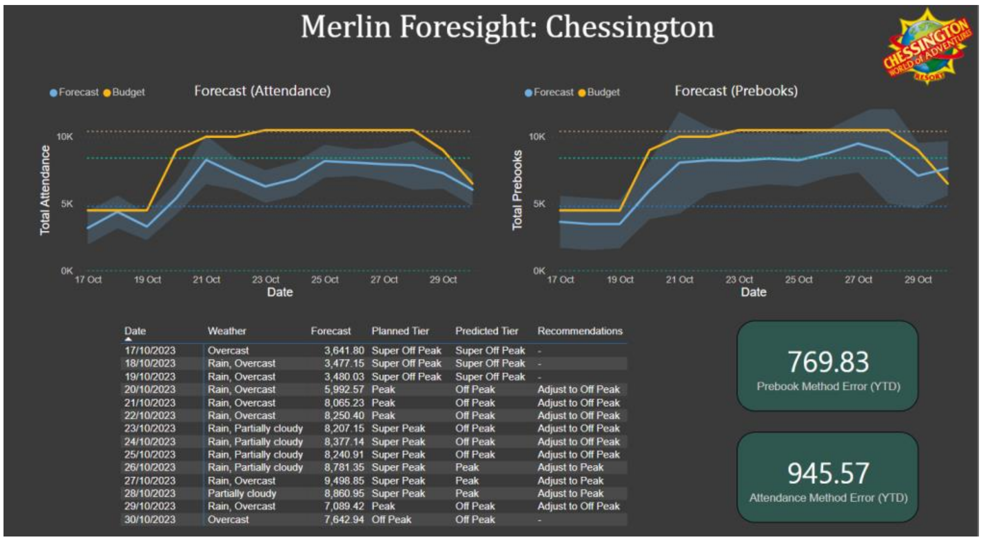
Data Visualization

In line with the requirements of this dashboard, I created four pages within the dashboard showing the upcoming totals forecast, the upcoming channels forecast, the historical forecast for totals and the historical forecast for channels. Finally, I added a fifth page showing just the actual totals with various filters as per a request from the financial team. The five pages, along with descriptions for each, are listed in the next section.

Dashboard Design

All dashboard screenshots below are using a random dataset for data privacy purposes, and do not represent any real-life data.

Page 1: Upcoming attendance forecast.



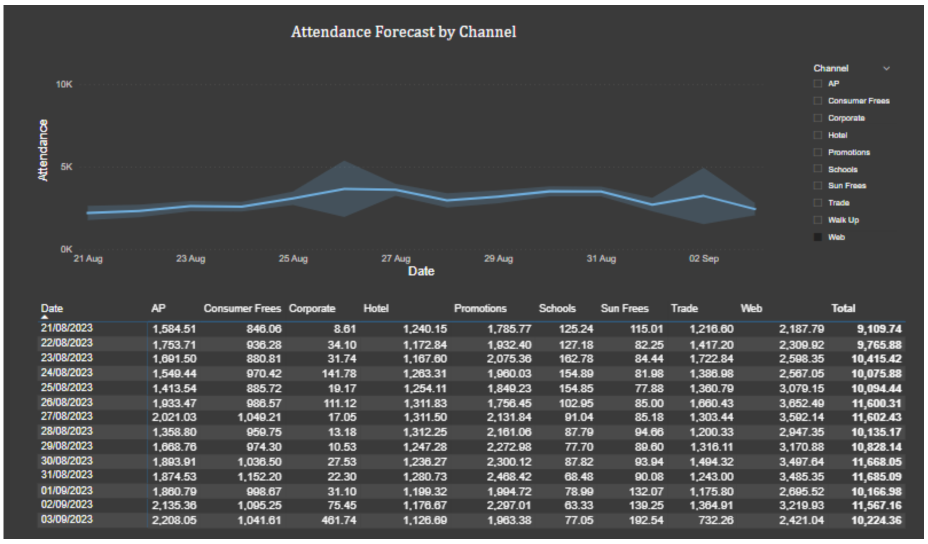
Date	Weather	Forecast	Planned Tier	Predicted Tier	Recommendations
17/10/2023	Overcast	3,641.80	Super Off Peak	Super Off Peak	-
18/10/2023	Rain, Overcast	3,477.15	Super Off Peak	Super Off Peak	-
19/10/2023	Rain, Overcast	3,480.03	Super Off Peak	Super Off Peak	-
20/10/2023	Rain, Overcast	5,992.57	Peak	Off Peak	Adjust to Off Peak
21/10/2023	Rain, Overcast	8,065.23	Peak	Off Peak	Adjust to Off Peak
22/10/2023	Rain, Overcast	8,250.40	Peak	Off Peak	Adjust to Off Peak
23/10/2023	Rain, Partially cloudy	8,207.15	Super Peak	Off Peak	Adjust to Off Peak
24/10/2023	Rain, Partially cloudy	8,377.14	Super Peak	Off Peak	Adjust to Off Peak
25/10/2023	Rain, Partially cloudy	8,240.91	Super Peak	Off Peak	Adjust to Off Peak
26/10/2023	Rain, Partially cloudy	8,781.35	Super Peak	Peak	Adjust to Peak
27/10/2023	Rain, Overcast	9,498.85	Super Peak	Peak	Adjust to Peak
28/10/2023	Partially cloudy	8,860.95	Super Peak	Peak	Adjust to Peak
29/10/2023	Rain, Overcast	7,089.42	Peak	Off Peak	Adjust to Off Peak
30/10/2023	Overcast	7,642.94	Off Peak	Off Peak	-

Figure 35: Page 1 Chessington World of Adventures Dashboard in Power BI (Upcoming Tab). KSB(S14)

On this first page (figure 35), I created visuals for the upcoming two weeks of forecasted attendance.

1. Line Graph Visual: This pane shows both the total attendance forecast method and the prebook method. These are sjpwm in two separate line graphs, both showing error bars and lines representing each attendance tier, as well as another line showing the set budget for each date. The X-axis is therefore date, while the Y-axis are budget, forecast, and the four tiers. To set the error bars on the forecast line, I went on Analytics -> Error Bars -> input lower and upper boundaries for forecast. I used a line chart as it clearly depicts our forecasts throughout time and is easy to compare to budget. The shaded blue area represents the confidence interval for each prediction, which looks great using a line chart as it clearly depicts the confidence changes over time depending on the day forecasted.
2. Table with recommendations: This table shows the planned and predicted tier for each day, along with the recommendations for that day and a column for weather, to give guidance to the pricing team on what price to set at each given day. A table here is useful as a means of giving clear guidance on pricing recommendations with no added effort by the revenue managers to interpret charts.
3. Cards: This page has three cards. Two of these show the average error of my model for each method from the start of the year. I achieved this by using a relative date filter and select the last year. To then filter out future dates, I used the dates column "Past Dates" and placed it in the filters pane, selecting "True". These cards give a quick indication of model performance without the need to look through past model performance.

Page 2: Upcoming Channels



Date	AP	Consumer Frees	Corporate	Hotel	Promotions	Schools	Sun Frees	Trade	Web	Total
21/08/2023	1,584.51	846.06	8.61	1,240.15	1,785.77	125.24	115.01	1,216.60	2,187.79	9,109.74
22/08/2023	1,753.71	936.28	34.10	1,172.84	1,932.40	127.18	82.25	1,417.20	2,309.92	9,765.88
23/08/2023	1,691.50	880.81	31.74	1,167.60	2,075.36	162.78	84.44	1,722.84	2,598.35	10,415.42
24/08/2023	1,549.44	970.42	141.78	1,263.31	1,960.03	154.89	81.98	1,386.98	2,567.05	10,075.88
25/08/2023	1,413.54	885.72	19.17	1,254.11	1,849.23	154.85	77.88	1,360.79	3,079.15	10,094.44
26/08/2023	1,933.47	986.57	111.12	1,311.83	1,756.45	102.95	85.00	1,660.43	3,652.49	11,600.31
27/08/2023	2,021.03	1,049.21	17.05	1,311.50	2,131.84	91.04	85.18	1,303.44	3,592.14	11,602.43
28/08/2023	1,358.80	959.75	13.18	1,312.25	2,161.06	87.79	94.66	1,200.33	2,947.35	10,135.17
29/08/2023	1,668.76	974.30	10.53	1,247.28	2,272.98	77.70	89.60	1,316.11	3,170.88	10,828.14
30/08/2023	1,893.91	1,036.50	27.53	1,236.27	2,300.12	87.82	93.94	1,494.32	3,497.64	11,668.05
31/08/2023	1,874.53	1,152.20	22.30	1,280.73	2,468.42	68.48	90.08	1,243.00	3,485.35	11,685.09
01/09/2023	1,860.79	998.67	31.10	1,199.32	1,994.72	78.99	132.07	1,175.80	2,695.52	10,166.98
02/09/2023	2,135.36	1,095.25	75.45	1,176.67	2,297.01	63.33	139.25	1,364.91	3,219.93	11,567.16
03/09/2023	2,208.05	1,041.61	461.74	1,126.69	1,963.38	77.05	192.54	732.26	2,421.04	10,224.36

Figure 36: Page 2 Chessington World of Adventures Dashboard in Power BI (Upcoming Channels Tab) KSB(S14)

On this page (figure 36), I show the forecasted values split by channel for the next two weeks.

1. Line Graph Visual: Another line graph visual here with less complexity to the previous ones. This only has the forecasted numbers on the Y-axis with an error band, with dates on the X-axis. This is the same to the upcoming page line chart but has the functionality to split by channel.
2. Slicer: A slicer for channel type to check each separate channel's forecasted numbers for the next two weeks. A clear way to switch between channels on a single tab.
3. Table: A table showing total attendance values split by channel per date for easier access to specific numbers. This visual has been disconnected from the filter on the same page as I want it shown at all times, and is a useful tool to gather quick insights on forecasted prebooks.

Page 3: Past Forecasts

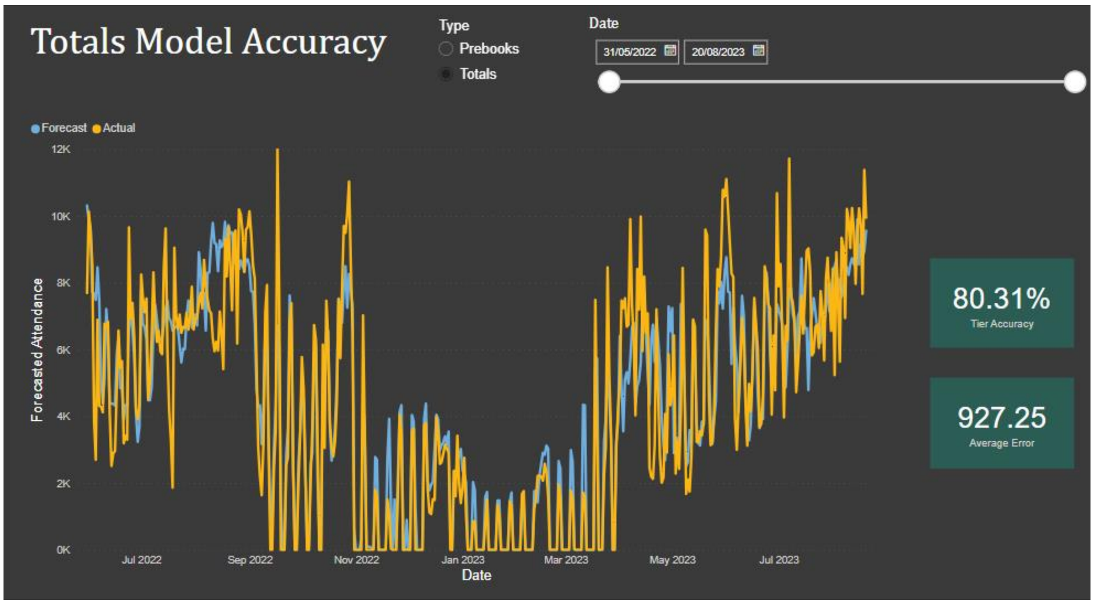


Figure 37: Page 3 Chessington World of Adventures Dashboard in Power BI (Past Forecasts Tab). KSB(S14)

Here (figure 37), I display all values forecasted in the past. These range from may 2022 to the day prior of the upcoming forecast.

1. Line graph visual: this visual shows the forecasted values and the actual attendance values throughout time. A line chart keeps the dashboard consistent to the first tab.
2. Cards: two cards, one for tier accuracy (percentage of dates where the forecast correctly predicted the attendance tier), and one for mean absolute error (measured by using the "Difference" column calculated earlier, and getting the average of this column). Again, these cards give stakeholders a quick overview of accuracy.

3. Slicers: two slicers on this page, one for method of prediction (totals or prebooks) and one for date, using a date slider. These enable the user to switch between methods and dates quickly and efficiently.

Page 4: Past Channel Forecasts

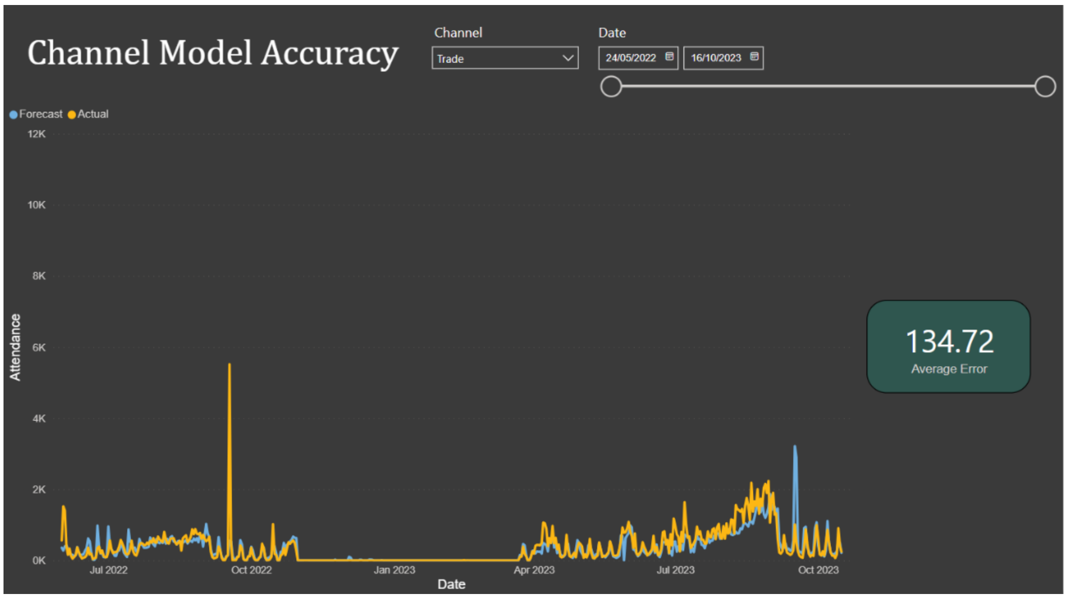
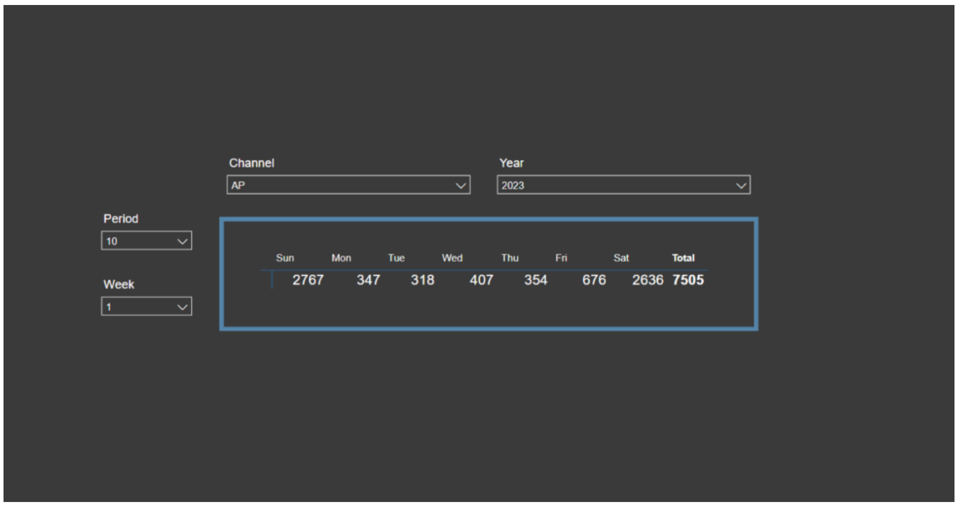


Figure 38: Page 4 Chessington World of Adventures Dashboard in Power BI (Past Channels Forecasts Tab). KSB(S14)

This page (figure 38) is somewhat similar to the previous, where the only difference is that there is no tier accuracy, as it cannot be measured when split by channel. Furthermore, instead of a filter for prediction method, there will be a filter for Sales Channel, enabling the user to select which channel they want to view historical forecasts for. The Average error card remains, but measures the average error for each sales channel and then averages these values out.

Page 5: Historical Attendance



Sun	Mon	Tue	Wed	Thu	Fri	Sat	Total
2767	347	318	407	354	676	2636	7505

Figure 39: Page 5 Chessington World of Adventures Dashboard in Power BI (Historical Attendance Tab). KSB(S14)

This page (figure 39) shows the historical attendance in a format more legible and usable for the finance team at the parks. This is because attendance values will be split by financial periods.

1. Table: the table shows actual attendance values from Sunday to Monday. This gives the user a quick overview of park performance and attendance for any given week in the financial year.
2. Slicers: There are various easy-to-use slicers here, including sales channel, year, financial period and financial week.

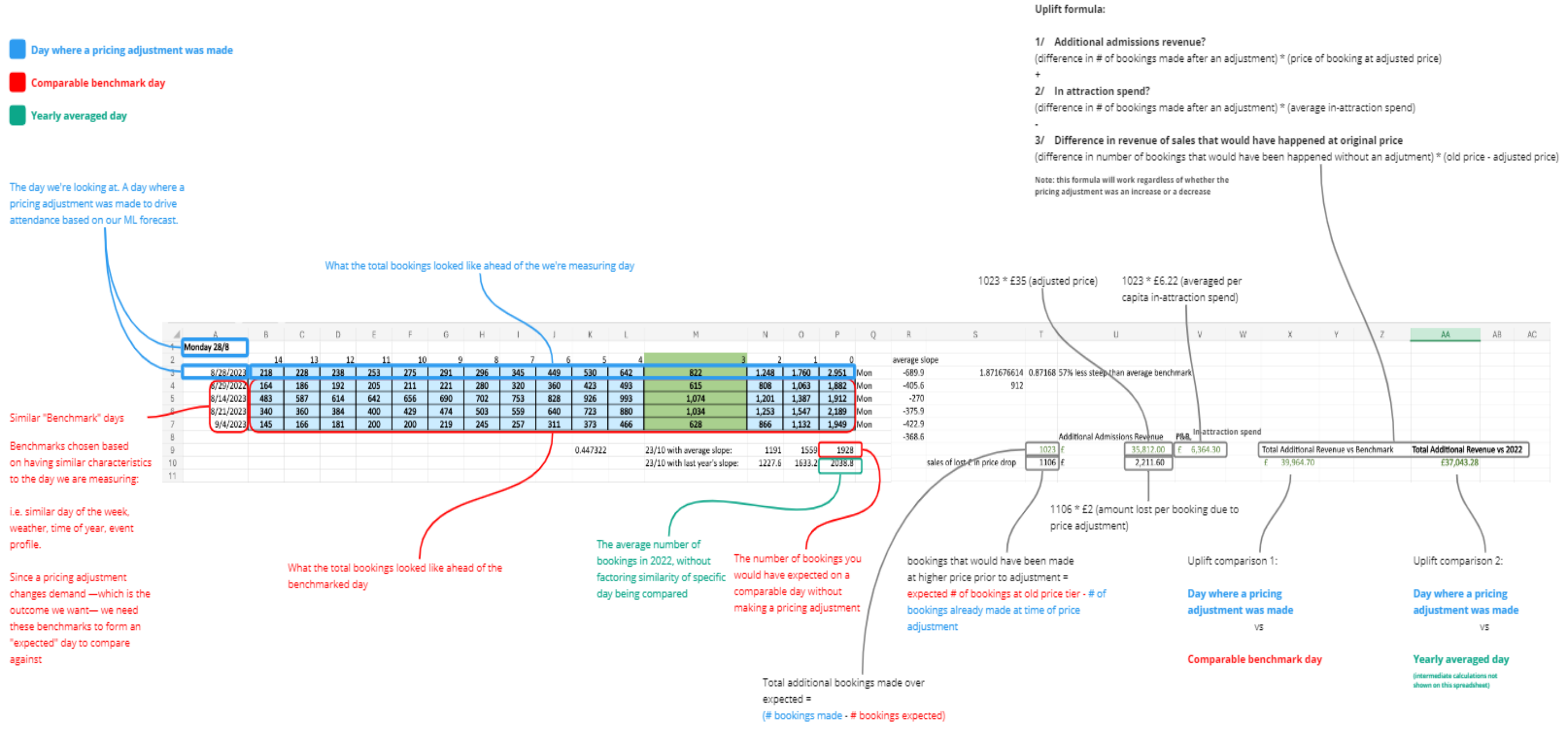
Implementation & Prescriptive Analytics

After deploying this dashboard, I have been receiving feedback and questions from both the financial teams at each park and the pricing team at BCG. Since these departments are using the model for separate use-cases, they often request added visuals or features which sometimes must be pushed back. For example, the financial team has requested to predict revenue by using the channels forecast, as I know the pricing for each channel ticket. However, this has been pushed back, as my model should not be used specifically for revenue prediction purposes but should instead be an all-encompassing tool which separate departments can use. Throughout the testing, we've also been able to notice which of the two methods is more accurate, and can therefore make a decision later, with user feedback included, on which method to use going forward.

There have been several pricing adjustment decisions made from out POC park: Chessington World of Adventures. I usually recommend parks to take pricing adjustments when our model predicts lower attendance compared to what the park had planned. This is in order to sell more tickets which will bring an overall revenue uplift. In my case, prescriptive analysis would involve taking the forecasted days, and calculate the pricing tier band based on these forecasts. If the pricing tier does not match the initial pricing tier set by the park, we make a recommendation, as shown in the dashboards. This prescriptive analysis was done directly in Power BI, as seen in the dashboards using the calculated columns "Recommendations", calculated using "Actual Tier" and "Predicted Tier", which in turn are calculated using specified pricing tier bands on the "Forecast" column.

To show the power of these pricing adjustments, I carried out some analysis to show the potential of a pricing adjustment. A few trial days were chosen and a pricing adjustment was made based on my model. The diagram in figure 40 shows how we measured the uplift based on these recommendations made using prescriptive analysis.

Pricing Action Uplift Analysis


🟦 Day where a pricing adjustment was made
🟥 Comparable benchmark day
🟩 Yearly averaged day
The day we're looking at. A day where a pricing adjustment was made to drive attendance based on our ML forecast.
What the total bookings looked like ahead of the we're measuring day
Similar "Benchmark" days
Benchmarks chosen based on having similar characteristics to the day we are measuring:
i.e. similar day of the week, weather, time of year, event profile.
Since a pricing adjustment changes demand —which is the outcome we want— we need these benchmarks to form an "expected" day to compare against
What the total bookings looked like ahead of the benchmarked day
The average number of bookings in 2022, without factoring similarity of specific day being compared
The number of bookings you would have expected on a comparable day without making a pricing adjustment
Total additional bookings made over expected =
(# bookings made - # bookings expected)
bookings that would have been made at higher price prior to adjustment =
expected # of bookings at old price tier - # of bookings already made at time of price adjustment
1023 * £35 (adjusted price)
1023 * £6.22 (averaged per capita in-attraction spend)
1106 * £2 (amount lost per booking due to price adjustment)
Uplift comparison 1:
Day where a pricing adjustment was made
vs
Comparable benchmark day
Uplift comparison 2:
Day where a pricing adjustment was made
vs
Yearly averaged day
(intermediate calculations not shown on this spreadsheet)
Uplift formula:
1/ Additional admissions revenue?
(difference in # of bookings made after an adjustment) * (price of booking at adjusted price)
+
2/ In attraction spend?
(difference in # of bookings made after an adjustment) * (average in-attraction spend)
-
3/ Difference in revenue of sales that would have happened at original price
(difference in number of bookings that would have been happened without an adjutment) * (old price - adjusted price)
Note: this formula will work regardless of whether the pricing adjustment was an increase or a decrease

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Monday 28/8																
2		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
3	8/28/2023	218	228	238	253	275	291	296	345	449	530	642	822	1,248	1,760	2,951	Mon
4	8/29/2022	164	186	192	205	211	221	280	320	360	423	493	615	808	1,063	1,882	Mon
5	8/14/2023	483	587	614	642	656	690	702	753	828	926	993	1,074	1,201	1,387	1,912	Mon
6	8/21/2023	340	360	384	400	429	474	503	559	640	723	880	1,034	1,253	1,547	2,189	Mon
7	9/4/2023	145	166	181	200	200	219	245	257	311	373	466	628	866	1,132	1,949	Mon
8																	
9											0.447322		23/10 with average slope:	1191	1559	1928	
10													23/10 with last year's slope:	1227.6	1633.2	2038.8	
11																	

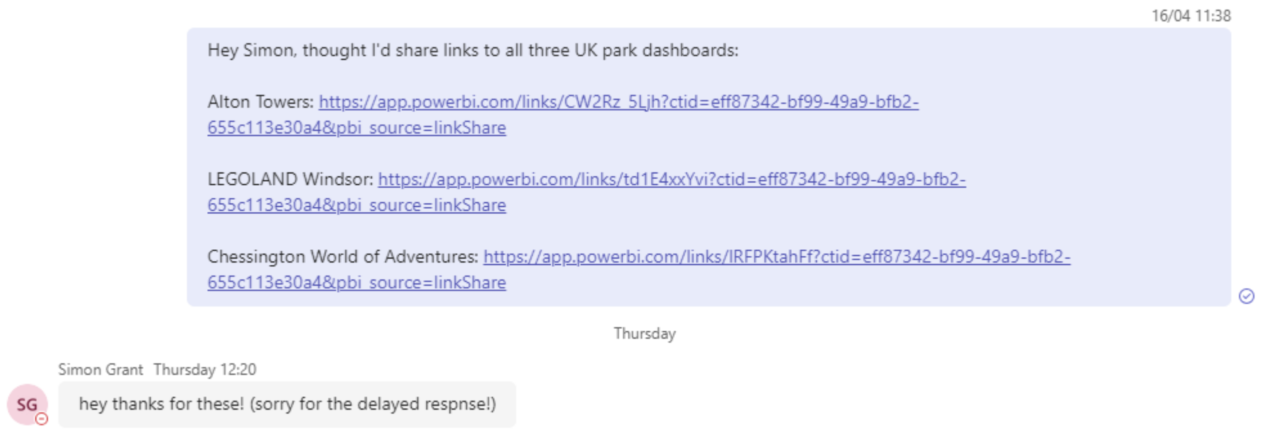
	R	S	T	U	V	W	X	Y	Z	AA
1										
2	average slope									
3	-689.9	1.871676614	0.87168	57% less steep than average benchmark						
4	-405.6	912								
5	-270									
6	-375.9									
7	-422.9									
8	-368.6			Additional Admissions Revenue	P&B,	In-attraction spend				
9			1023 £	35,812.00	£ 6,364.30		Total Additional Revenue vs Benchmark			Total Additional Revenue vs 2022
10		sales of lost £ in price drop	1106 £	2,211.60			£ 39,964.70			£37,043.28
11										

Figure 40: Pricing Action Analysis diagram KSB(K13,K14, S10, B1, B2)

I took our pricing action day and a few similar benchmark days from that same year, as well as the same day the year prior. I gathered the prebooking numbers for all of these days and highlighted the day where the pricing action took place (in green). I then measured the slope of the curve for each day from the pricing action date to the event date. I then averaged out all of the slopes (not including the pricing action day slope) and, in conjunction with the total prebooks for the day of interest prior to the pricing action (up to green highlighted cell), I measured how many tickets would have been sold on a normal day. I then found the difference between the actual sold tickets and the predicted sold tickets if the pricing action did not occur. I multiplied these tickets by their total price, and subtracted the loss in revenue due to the pricing difference (as prices would have been lowered). I also multiplied the total extra tickets sold by the average spend per guest (about 6.22£). For this particular day, I estimated the uplift to be in the region of 39,000£. Across the 9 days of pricing action from last year, I estimated an uplift of 273,120£ in revenue.

Stakeholder Feedback

A lot of communication with our stakeholders was carried out on teams and via email. Most of my conversations were with the head of revenue management for UK parks (Simon Grant), as well as with the head of revenue management for LEGOLANDs in North America (Matt Atkinson). Below show some of the conversations with them on teams where I shared model dashboards and model improvements (figures 41 & 42).



16/04 11:38

Hey Simon, thought I'd share links to all three UK park dashboards:

Alton Towers: https://app.powerbi.com/links/CW2Rz_5Ljh?ctid=eff87342-bf99-49a9-bfb2-655c113e30a4&pbi_source=linkShare

LEGOLAND Windsor: https://app.powerbi.com/links/td1E4xxYvi?ctid=eff87342-bf99-49a9-bfb2-655c113e30a4&pbi_source=linkShare

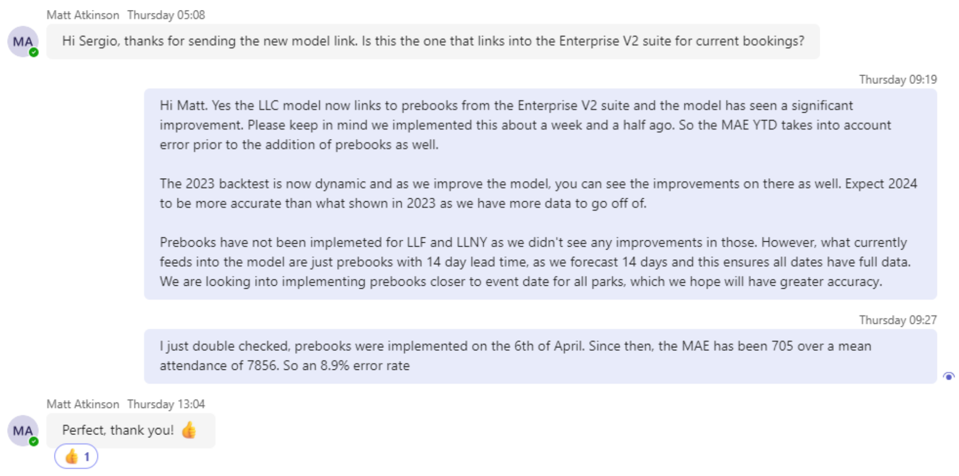
Chessington World of Adventures: https://app.powerbi.com/links/lRFPKtahFf?ctid=eff87342-bf99-49a9-bfb2-655c113e30a4&pbi_source=linkShare

Thursday

Simon Grant Thursday 12:20

SG hey thanks for these! (sorry for the delayed respnse!)

Figure 41: Model Dashboards shared using Power BI link to UK Head of Revenue Management KSB(K7, S5, B1, B5)



Matt Atkinson Thursday 05:08

MA

Hi Sergio, thanks for sending the new model link. Is this the one that links into the Enterprise V2 suite for current bookings?

Thursday 09:19

Hi Matt. Yes the LLC model now links to prebooks from the Enterprise V2 suite and the model has seen a significant improvement. Please keep in mind we implemented this about a week and a half ago. So the MAE YTD takes into account error prior to the addition of prebooks as well.

The 2023 backtest is now dynamic and as we improve the model, you can see the improvements on there as well. Expect 2024 to be more accurate than what shown in 2023 as we have more data to go off of.

Prebooks have not been implemeted for LLF and LLNY as we didn't see any improvements in those. However, what currently feeds into the model are just prebooks with 14 day lead time, as we forecast 14 days and this ensures all dates have full data. We are looking into implementing prebooks closer to event date for all parks, which we hope will have greater accuracy.

Thursday 09:27

I just double checked, prebooks were implemented on the 6th of April. Since then, the MAE has been 705 over a mean attendance of 7856. So an 8.9% error rate

Matt Atkinson Thursday 13:04

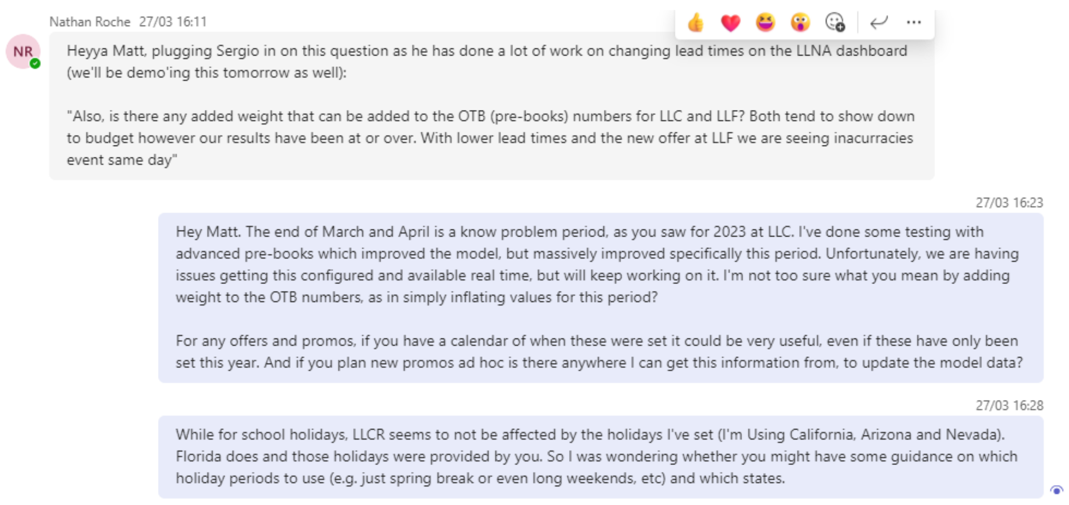
MA

Perfect, thank you! 👍

👍 1

Figure 42: Example of communication with Head of Revenue Management for North American LEGOLANDs KSB(K7, S5, B1, B5, B6)

All our stakeholders, including the two heads of revenue management, are very excited about the product, but some can show some concern at times. These conversations usually go through our product owner, who then includes us when needed. Here is an example of how I managed one of these discussions (figure 43).



Nathan Roche 27/03 16:11

NR

Heyya Matt, plugging Sergio in on this question as he has done a lot of work on changing lead times on the LLNA dashboard (we'll be demo'ing this tomorrow as well):

"Also, is there any added weight that can be added to the OTB (pre-books) numbers for LLC and LLF? Both tend to show down to budget however our results have been at or over. With lower lead times and the new offer at LLF we are seeing inacurracies event same day"

27/03 16:23

Hey Matt. The end of March and April is a know problem period, as you saw for 2023 at LLC. I've done some testing with advanced pre-books which improved the model, but massively improved specifically this period. Unfortunately, we are having issues getting this configured and available real time, but will keep working on it. I'm not too sure what you mean by adding weight to the OTB numbers, as in simply inflating values for this period?

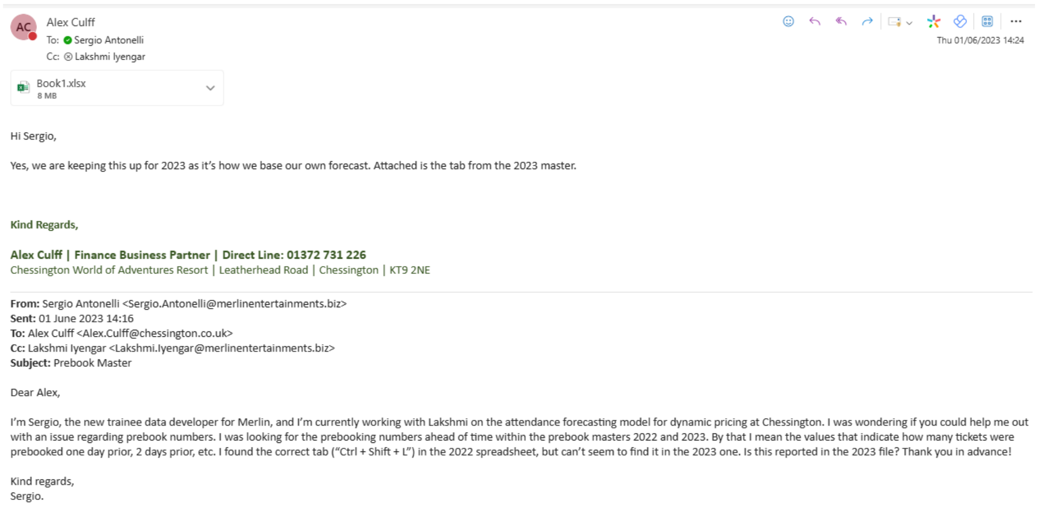
For any offers and promos, if you have a calendar of when these were set it could be very useful, even if these have only been set this year. And if you plan new promos ad hoc is there anywhere I can get this information from, to update the model data?

27/03 16:28

While for school holidays, LLCR seems to not be affected by the holidays I've set (I'm Using California, Arizona and Nevada). Florida does and those holidays were provided by you. So I was wondering whether you might have some guidance on which holiday periods to use (e.g. just spring break or even long weekends, etc) and which states.

Figure 43: Answering concerns about model accuracy and managing expectations KSB(K7, S5, B1, B5, B6)

In other cases, we may need additional information from other stakeholders, such as park operators, to improve our models. Below is the initial email I sent to one of our stakeholders at our POC park Chessington World of Adventures, as well as their response (figure 44).



Alex Culff
To: Sergio Antonelli
Cc: Lakshmi Iyengar
Thu 01/06/2023 14:24

Book1.xlsx
8 MB

Hi Sergio,

Yes, we are keeping this up for 2023 as it's how we base our own forecast. Attached is the tab from the 2023 master.

Kind Regards,

Alex Culff | Finance Business Partner | Direct Line: 01372 731 226
Chessington World of Adventures Resort | Leatherhead Road | Chessington | KT9 2NE

From: Sergio Antonelli <Sergio.Antonelli@merlinentertainments.biz>
Sent: 01 June 2023 14:16
To: Alex Culff <Alex.Culff@chessington.co.uk>
Cc: Lakshmi Iyengar <Lakshmi.Iyengar@merlinentertainments.biz>
Subject: Prebook Master

Dear Alex,

I'm Sergio, the new trainee data developer for Merlin, and I'm currently working with Lakshmi on the attendance forecasting model for dynamic pricing at Chessington. I was wondering if you could help me out with an issue regarding prebook numbers. I was looking for the prebooking numbers ahead of time within the prebook masters 2022 and 2023. By that I mean the values that indicate how many tickets were prebooked one day prior, 2 days prior, etc. I found the correct tab ("Ctrl + Shift + L") in the 2022 spreadsheet, but can't seem to find it in the 2023 one. Is this reported in the 2023 file? Thank you in advance!

Kind regards,
Sergio.

Figure 44: Requesting additional data from park operators KSB(K7, S5, B1, B5)

Ethical Considerations for Dynamic Pricing

It is important to keep in consideration the ethics of using this model as a dynamic pricing tool. There is a lot of controversy on surge pricing, which is why we tend to only recommend lowering pricing, although a surge in pricing can be optimal in very high demand days to maintain a manageable number of guests at the parks. Dynamic pricing has been used by many businesses such as airlines for years. In our case, dynamic pricing will increase revenue for the company while always keeping into consideration customer experience and maintain our standards at the parks. However, there are 5 main ethical considerations when it comes to dynamic pricing: Discrimination & Fairness, Transparency & Trust, Price Gouging, Data Privacy & Security, and Customer Perception & Loyalty. These are explained in more detail in figure 45.

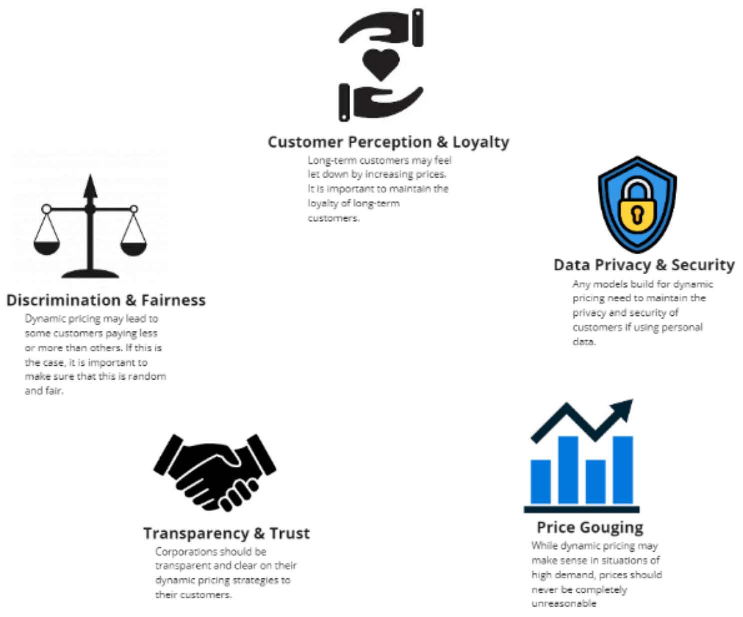
Discrimination & Fairness: Pricing adjustments are made for all customers ad-hoc a selected number of days prior to the event date. This may lead some customers who bought tickets earlier on to pay more, which may not seem fair. However, this may also lead to some customers to pay less depending on whether pricing has increased or decreased, therefore balancing the scales.

Transparency & Trust: We have already gone public about our plans for dynamic pricing, and customers are aware of pricing adjustments being made at parks. The specifics of the model have not been made public as of yet, as we have only used it to lower prices.

Price gouging: Pricing adjustments are typically of about +- 2£, therefore not creating a big burden on customers, but lead to a big uplift for us, whether the price has decreased or increased.

Data Privacy & Security: We use no personal data, and therefore do not have to worry about GDPR considerations.

Customer Perception & Loyalty: Most of our long-term customers have annual passes and therefore do not feel the effects of dynamic pricing. For those who do not own annual passes, we see more loyal consumers in winter season, as summer seasons, more often than not, have larger numbers tourists. Most of the pricing actions in the winter seasons are to reduce price, therefore rewarding loyal customers who still enjoy roller coasters on grey days.



Customer Perception & Loyalty
Long-term customers may feel let down by increasing prices. It is important to maintain the loyalty of long-term customers.

Discrimination & Fairness
Dynamic pricing may lead to some customers paying less or more than others. If this is the case, it is important to make sure that this is random and fair.

Data Privacy & Security
Any models build for dynamic pricing need to maintain the privacy and security of customers if using personal data.

Transparency & Trust
Corporations should be transparent and clear on their dynamic pricing strategies to their customers.

Price Gouging
While dynamic pricing may make sense in situations of high demand, prices should never be completely unreasonable

Figure 45: Ethical considerations for dynamic pricing models KSB(K15)

Foresight Model Conclusions

The main results I see from my Chessington model are an 85% accuracy in pricing tier predictions, with a mean average error of 760 guests with an average attendance of 6435 using the Prebooks method (meaning a 11.8% error rate in park attendance). I plan to reduce this error down to below 10% as the model learns from new data every day. The North American legolands are all operating at a sub 10% error rate (7% for Florida, 9% for California, and 4% for New York). The dashboard is currently the 35th most used Power BI dashboard out of 417 within the organization with 11 unique users. The outcome of the Merlin Foresight model is a widespread interest in the power of machine learning for business insights throughout Merlin. The new Chief Digital & Data Officer has in fact asked for a worldwide rollout in the next two years, with the support of a newly built data science team and a team of consultants. This dashboard can be used as a template for other parks as well but will still require a few minor adjustments. While rolling out new and bigger parks, the unique users of the product have increased exponentially.

To date, the model has increased the revenue by an average of 56% on days where pricing action was enacted from recommendations given in the dashboard. This model has also worked as a catalyst in the business to bring in more in house development of machine learning tools and other software which we have been outsourcing for a long time, costing the business a fortune. This has also led to new data scientists being hired to carry on the work for these models and more. Next steps also include the productionalizing the code that runs the model, to ease the roll out of future products. Furthermore, thorough documentation will have to be put in place to ease the handover to future owners of the product.

I have built a model for 6 parks and plan to roll out to midways (which include smaller attractions such as Madame Tussauds) by the second quarter. We have also since started building a repository for all the data pipelines and dashboards for the model, as the team has increased and this project has now become a workstream. I am incredibly proud of the journey that this project has gone through, from being a personal project for myself to being one of the most interesting pieces of work happening at Merlin Technology, who have since started building a team of data scientists due to the interest the model has brought to the field.

Data Architecture at Merlin

Data Models, Database Design & Implementation, Data Structures,

Merlin Data

Merlin’s data strategy is a complex, hybrid system of both on-premise databases and Azure cloud services. The volume of data at Merlin is very large, as we have over 20,000 seasonal employees and over 60 million guests every year. Furthermore, Merlin doesn’t just collect ticketing data, but we also own various hotels, as well as food and beverage at each park. This means we have data regarding industry sectors such as entertainment, hospitality, catering, etc. There is therefore not a simple data model to be described. Here, I will focus on some of the data projects at Merlin and the data models associated with them.

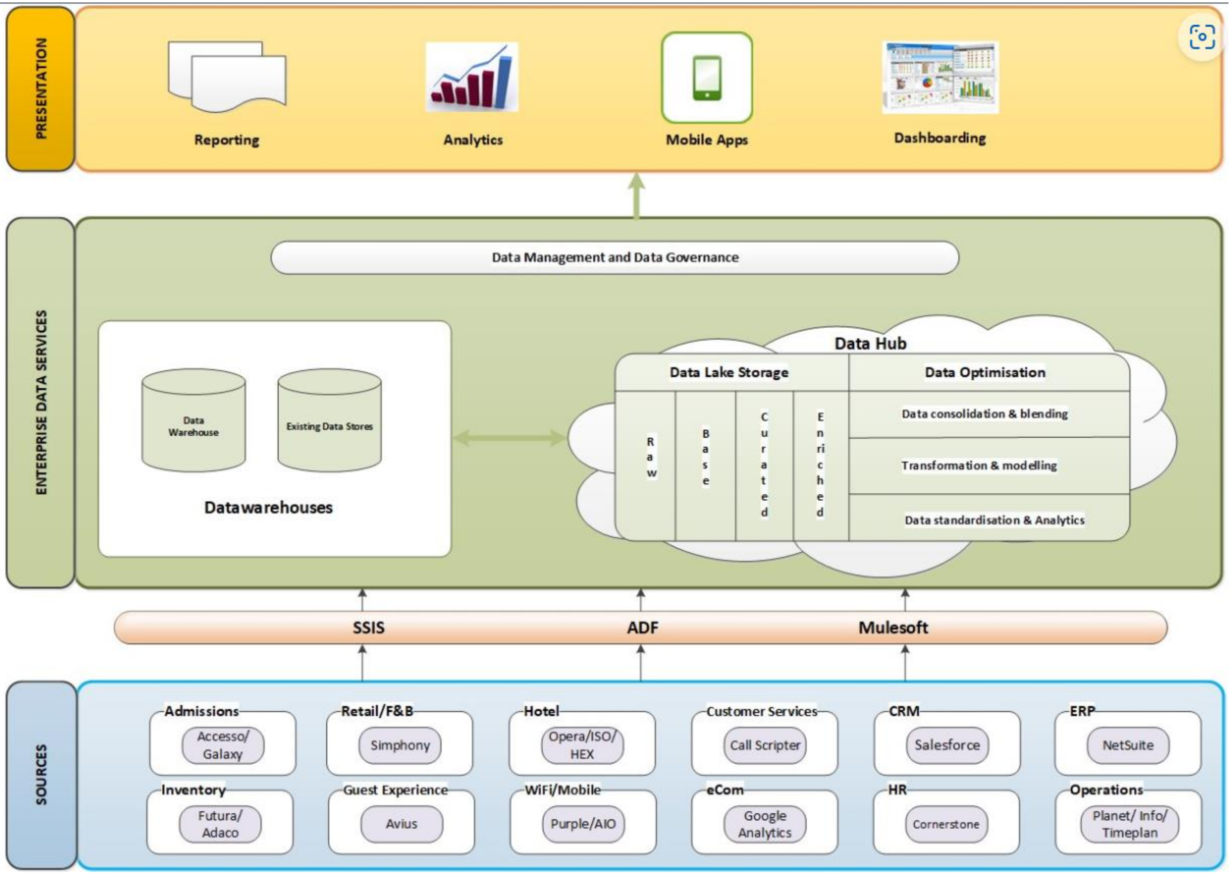


Figure 46: Merlin data strategy overview. Data collected from sources (blue), integrated to company data hub (green) using various integreation techniques and services (orange). ETL or ELT used depending on data source. Data used for reporting, analytics, dashboarding and data science (yellow) using systems such as Power BI, Databricks or Azure services. KSB(K5)

Due to all these systems and sources, the data architecture at Merlin relies on heavy integration work. Figure 46 shows our data strategy. We receive data from a vast amount of systems depending on the service (hotels, food and beverage, HR, etc.). We then use different integration services (SSIS, ADF or Mulesoft) to pull this data into our datahub (built on Azure services since 2019) or other data warehouses on-prem, depending on the needs for each data source. The more unstructured data is then curated and enriched for better dashboarding, reporting and analytics (therefore using an ELT approach, mostly used for data stored in Azure).

Some of the more pre-structured and logical data is instead stored as is past the integration step (using an ETL approach, mostly used for data stored on on-prem servers). As we progress in our data journey, the business will start using more cloud services for more advanced analytics models to transform the organization and drive better with a speed layer, as well as AI and ML.

Drive Time Calculation

Summary

The drive time calculation project was requested by a senior CRM lead who needed to see drive times in their customer data to all parks in their region, to produce some useful analysis on the likelihood of ticket or annual pass purchases, given drive times. To carry this out, I needed to use CRM data, which is not owned by the data team, but rather by Salesforce. This is stored in a server which the data team has read access on. After connecting to this customer data, I could use postcode data to find coordinates, and by using these coordinates I could measure their distance to park coordinates. I then just needed to use the average driving speeds in each country to find drive times. While I could have used the Google Maps API to get more accurate drive times, the cost for this was not worth the level of accuracy it had over the previously discussed method. After measuring drive times, I set a band (1-5) based on how long the drive time was for each customer and each park. These were then sent back to Salesforce who added a new table containing a Customer Key and one column per park. Each row value is therefore a drivetime band for a specific park from a specific customer.

CRM Data

This project therefore uses customer data from Salesforce to create new data, useful for marketing analysis going forward. The data is stored in a STAR schema, with multiple fact tables all linked to various dimension tables. For example, if I needed ticket usage data, I would use the FACTUsage table and whichever dimensions I am interested in (such as customer, attraction, sales merchant, etc.) Below (figure 47) is a simplified view of the conceptual data model for this data source.

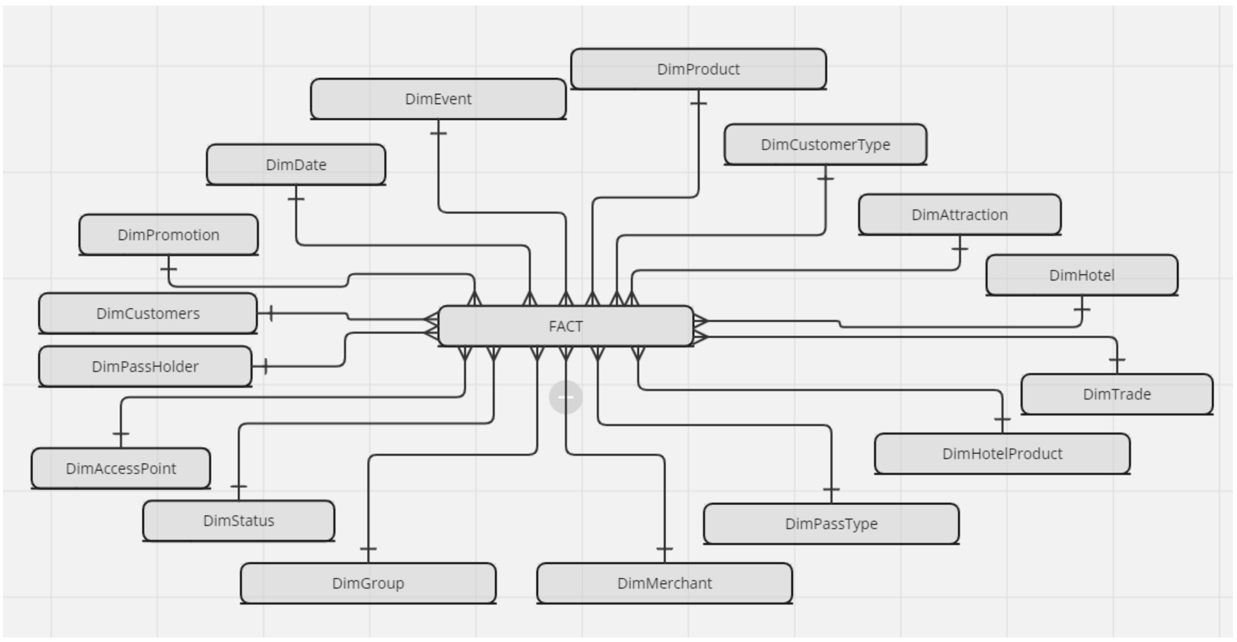


Figure 47: CRM data model. FACT table in the middle of the STAR schema represents all fact tables within CRM (Order, OrderLine, Usage, HotelBookings, etc.). Each of these fact tables will use various dimension tables as shown above. Not all dimension tables will be useful for all the fact tables, requires some investigation into the fact tables before joining. KSB(K6)

Prebooking Portal Reporting Project

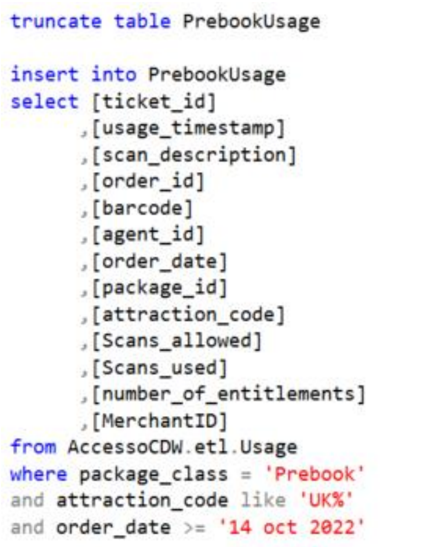
Summary

The prebooking portal is a product, accessible via the merlin website, where merlin annual pass holders can book their visits ahead of time, as well as purchase or add their annual passes. The issue with annual pass pre-bookings, is that some people may simply book every day for the next year and pick which day they wish to go to the park. This can cause issues with budgeting or reporting as we will overestimate the number of visitors on each given day. The business has therefore requested a report to check these numbers, as well as to have a good overview of the prebooking portal usage metrics and how efficient it is. For this project, I therefore needed to create a Power BI dataset using data from a Cosmos DB, containing all the data from the prebooking portal, as well as usage data from the internal data hub. I then joined these together through ticket barcodes and check edhow many prebooked tickets have actually been scanned.

CDW

- Customer Data Warehouse DB
- Data Layers: Raw
- Format: SQL Server Tables

The Accesso CDW is an on-prem data team owned database stored within the company's data hub. This specific database contains accesso data, which is the third party ticketing system used by Merlin. This database contains orders, customers, attractions, annual passes, and much more data. What I need for this project is the Usage table. This table contains all the ticket scans from all Merlin parks which use Accesso (as there are some parks that use other legacy systems such as Galaxy). This is equivalent to about 166 million rows and 61 rows. In order to import this into power BI without creating performance issues, a filtering of this data is essential. To do this efficiently, I created a staging database for this project within our on-prem database. I then built a stored procedure in SSMS to filter for the specific columns and rows relevant to prebooking ticket usage since the prebooking portal has been live. This resulted in a table of 13 columns and 238,000 rows, which is much more manageable. Figure 48 displays the SQL code for the stored procedure.



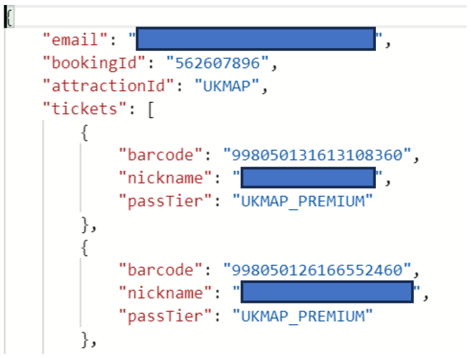
```sql
truncate table PrebookUsage

insert into PrebookUsage
select [ticket_id]
      ,[usage_timestamp]
      ,[scan_description]
      ,[order_id]
      ,[barcode]
      ,[agent_id]
      ,[order_date]
      ,[package_id]
      ,[attraction_code]
      ,[Scans_allowed]
      ,[Scans_used]
      ,[number_of_entitlements]
      ,[MerchantID]
from AccessoCDW.etl.Usage
where package_class = 'Prebook'
and attraction_code like 'UK%'
and order_date >= '14 oct 2022'
```

Figure 48: Stored procedure to filter usage data and store into Prebook Usage table within staging database. KSB(K6)

What this stored procedure does, is take the current PrebookUsage table, truncate it (therefore removing all the rows from the current table), and then use an insert command to filter and retrieve the correct data for the report. Since this report won't have to be real-time and the refresh can be done once a week, this is a more efficient way of getting this data, rather than using a view. If any of the requirements changed, I simply altered this stored procedure to retrieve the correct data. There is some further transformation that goes into this before the final data model, which is then carried out in Power BI in the next step.

Azure Cosmos DB

- Prebooking Portal DB
- Data Layers: Raw (Schemaless)
- Format: JSON



```
{
    "email": "[redacted]",
    "bookingId": "562607896",
    "attractionId": "UKMAP",
    "tickets": [
        {
            "barcode": "998050131613108360",
            "nickname": "[redacted]",
            "passTier": "UKMAP_PREMIUM"
        },
        {
            "barcode": "998050126166552460",
            "nickname": "[redacted]",
            "passTier": "UKMAP_PREMIUM"
        },
```

Figure 49: Example data from Cosmos DB Bookings Data as a collection of JSON files. KSB(K6)

The Cosmos DB is a schemaless database, meaning the data is not stored as tables and columns, but rather in a more unstructured JSON format, with different nested layers of data for different cosmos db containers. This type of data is known as semi-structured, and therefore isn't as complex as other unstructured sources (such as videos or images), as it contains some sort of hierarchy to the data. However, it is also not as structured as something like a relational database (tables with relationships) would be, therefore requiring me to create a clearer structure using some data software, such as Power BI in this case. Most of the data in the Cosmos DB is required within the dashboard, and is small enough to not cause performance issues, therefore the transformation was all carried out in Power BI. Below if the structure of the data within the Cosmos DB (figure 49), and the structure I built in Power BI including the usage data (figure 50).

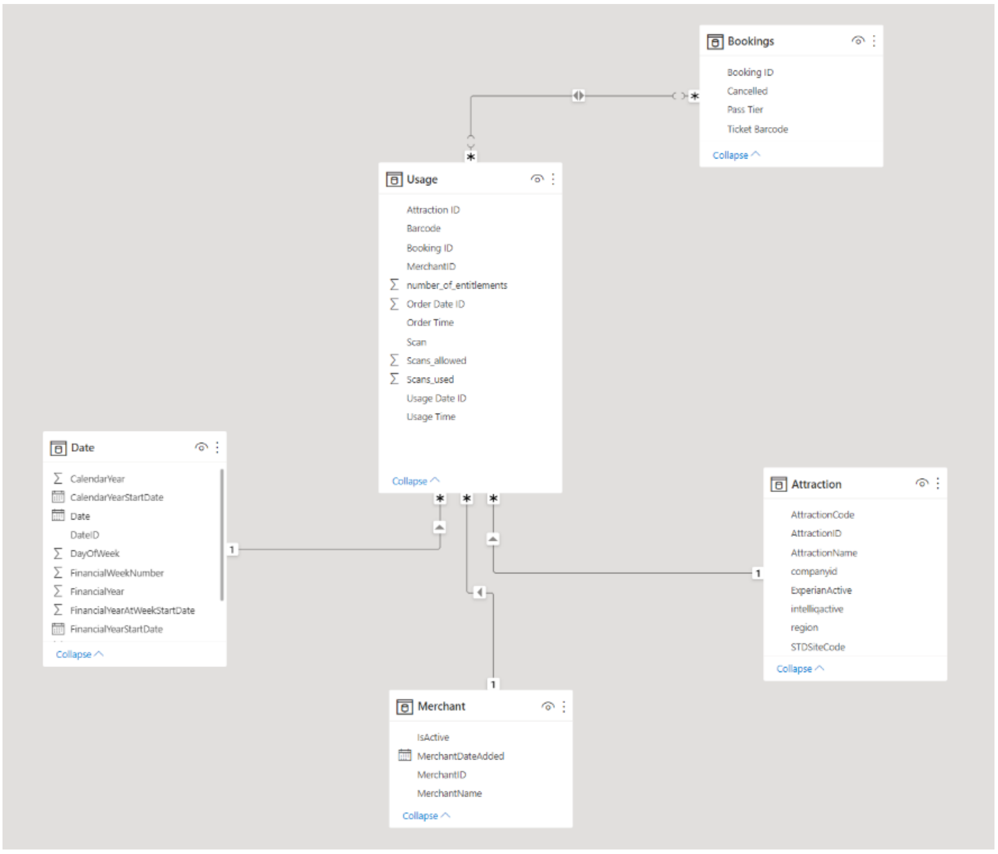


Figure 50: Data model in Power BI for prebooking ticket usage report. STAR schema approach to model, with four dimension tables (Attraction, Date, Bookings & Merchant), one fact table (Usage). All one to many relationships except Bookings (one booking may have multiple annual passes, and multiple bookings can have the same annual pass barcode). KSB(K6, S9)

Power BI Approval Data Model

Summary

A member of the data team has recently built a power BI dataset with a wide range of data useful for reporting by citizen developers. This was built as a measure to reduce the number of users requesting access to servers with very limited knowledge of data structures or SQL. To gain access to this new dataset, a member of staff is currently receiving requests via email. He then must check the requester's email or location and make an informed decision on which row level access to grant to the requester. This is very time consuming, and not scalable as more and more users are wanting access to this dataset, or once more datasets are rolled out to citizen developers. I therefore was tasked to create a solution to this issue, by creating an app where users could fill in a form requesting access. The app can detect the user's job description and location and pick the correct row level access based on a series of rules given to me by the project manager.

Business Entities & Logical Data Model

The row level access is split by business entities. These include things like operating group, region, country, division, consumer brand, management group, etc. I decided to create a data model with a Requests table and a Group table, with a Subdivision table to normalize the group table further. Each group in the "Group" table represents an AD group with a certain level of row level access. Each group is part of a subdivision (for example, the group "Austria" is part of the country subdivision, while "Peppa Pig" is under management brand). I therefore have another column in the "Group" table with a subdivision ID. These IDs and their respective subdivisions are stored in the table "Subdivision". The "Requests" table instead will have 5 columns, one unique Request ID, one for Group ID, and then three columns with information relative to the specific request. These rows are only populated once the app has picked a group using a separate python script based on the requester information. Figure 51 depicts the logical data model for this app.

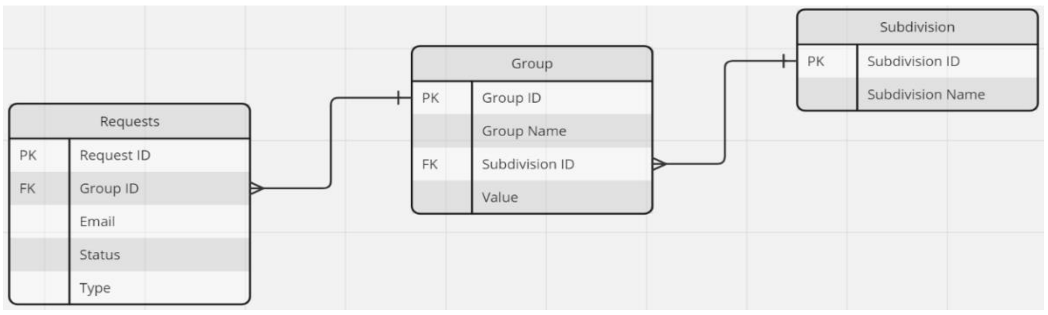
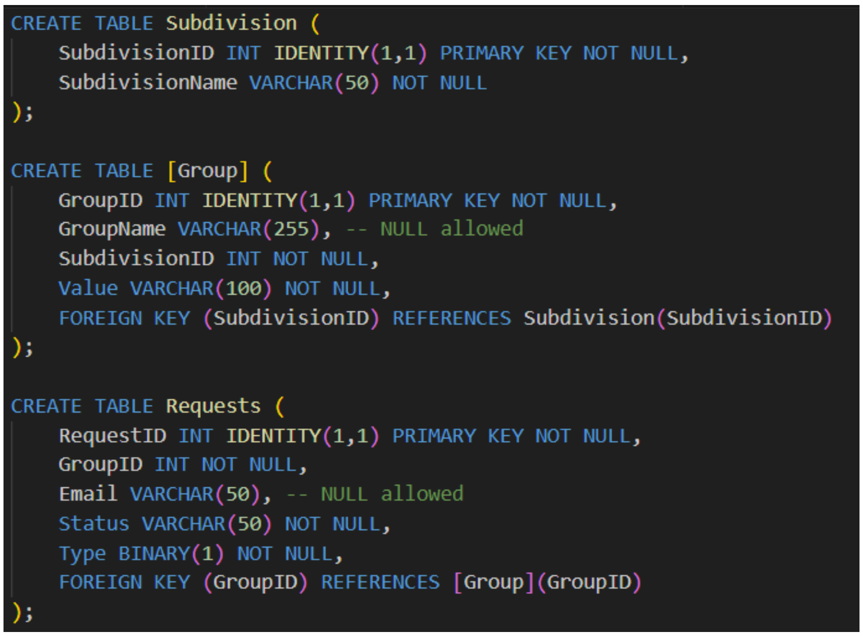


Figure 51: Logical data model for Enterprise dataset approval solution. Multiple requests per group (many to one) and Multiple groups in each subdivision (many to one). Subdivision used as a lookup table. KSB(K6, S9)

This data model was then built on a SQL server in azure. Below is the code used to create the model (figure 52):



```sql
CREATE TABLE Subdivision (
    SubdivisionID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
    SubdivisionName VARCHAR(50) NOT NULL
);

CREATE TABLE [Group] (
    GroupID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
    GroupName VARCHAR(255), -- NULL allowed
    SubdivisionID INT NOT NULL,
    Value VARCHAR(100) NOT NULL,
    FOREIGN KEY (SubdivisionID) REFERENCES Subdivision(SubdivisionID)
);

CREATE TABLE Requests (
    RequestID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
    GroupID INT NOT NULL,
    Email VARCHAR(50), -- NULL allowed
    Status VARCHAR(50) NOT NULL,
    Type BINARY(1) NOT NULL,
    FOREIGN KEY (GroupID) REFERENCES [Group](GroupID)
);
```

Figure 52: SQL Logic to build data model in SQL Warehouse for park subdivisions and regions KSB(K6, S9)

Third Party Data

Data Science Model Project

Our time-series forecasting ML model for predicting attendance at some of our parks utilizes both third party data and Merlin data. The third party data includes school holidays and weather conditions, as well as some accesso data which is not integrated or stored within our systems yet, but is needed for my model to function. All the data needed for this model is stored in an Azure Blob Storage in the form of .csv files. This is a temporary measure while I set up a more governed approach to rolling these models out. Due to the nature of these files, there is no data model to how my forecasted and historical data is stored. The pipelines in charge of this data updates these files as the model learns and the historical values accumulate. This data pipeline is described in the "Merlin Foresight – Build" section.

Google BigQuery Mobile App

This is a database containing interaction data from our mobile apps. These are built from a third party and therefore have the rights to the data. There is a data integration project underway to bring this data into our data hub, therefore allowing easier real-time reporting and analytics on mobile app use. For now, I only have this google bigquery data from google analytics. This data is completely raw, therefore having no data model in place either. It is very hard to manipulate and using it for analytics is inefficient, which is why a project to ingest mobile app data was started.

Conclusion & Reflections

Data at Merlin is complex, and the data team has only recently started expanding. Since I joined the company, the data team grew from 6 of us to 13, therefore more than doubling. While some of our systems are still inefficient and require some work, the data strategy at Merlin has significantly improved in the past 6 years, and just this year has moved to introducing machine learning concepts, gathering a lot of interest through the business. As shown from some of these projects, the vast amount of data collected by Merlin requires a large amount of effort to make some use out of it. There is not just a simple data model to be explained for Merlin, but hundreds of different models and techniques required for separate uses. Some of the new technologies which have caught the attention of my team include the power platform, which are a series of services from Microsoft linked to Power BI, such as power apps, power automate and much more. These services will create an easy development environment for less technical people within the business, which may drive quicker development, as it will free up some time for more technical and senior developers to work on more complex tasks. While these technologies are exciting, I think the business should be careful in letting citizen developers have too much access, as a lot of governance will be ignored and may lead to data security issues. Our data strategy is fairly messy, which is why most of the data projects within our department are more integration and database specific, to be able to collate all our systems data in an easy to access data hub within the cloud, with some exceptions. Once this is done, I think the data science work will be much more efficient and will revolutionize how we use data at Merlin. We are working hard to refine our standards for easier onboarding of new hires and working more efficiently. There is a lot of work to be done, but Merlin is on a clear path for better use of our data and a clearer architecture.

Data Analyst Apprenticeship Conclusion

When I started this apprenticeship, as well as my career, I had a general understand of data analysis and data science. I was very keen to learn about data science as a potential career choice. I started off as a data developer, and quickly progressed within the team by showcasing my acquired skills in my day-to-day job. Due to the success of some of the small-scale projects I was working on, I asked to be put on a new data science project with a data scientist from Blackstone, on loan at our company to prove the use-cases and benefits of data science at Merlin. I quickly picked up a lot of responsibility and started working on Merlin Foresight, the first business product I have built at a professional level. Thanks to the success of this product, I was moved to a newly built data science team, where I picked up even more responsibility working on this product. I adapted quickly to an agile environment and new ways of working, including working with source control and ML Flow. All the skills I acquired throughout this apprenticeship have aided me in achieving a highly accurate demand forecasting product, and will soon be working on many more data science projects. The apprenticeship has also aided me in gaining a diversified skill-set, including data engineering practices such as database management and cloud computing. I have been in communication with various different people throughout the business, including revenue managers, chief executives, central planning managers, and fellow data scientists, which has helped me in learning the key skills in stakeholder communication and professional discussion. This apprenticeship, and my time at Merlin, was a great source of professional development which will help me long-term in achieving my goals throughout my career.

KSB Summary

Knowledge

- K1: Current relevant legislation and its application to the safe use of data.
    - Covered in the Data Practices at Merlin – Data Protection Practices section,
        - Pages 6-8, where I describe the processes in place at Merlin for data projects using any level of personal data.
- K2: Organisational data and information security standards, policies, and procedures relevant to data management activities.
    - Covered in the Data Practices at Merlin – Data Protection Practices section
        - Pages 6-8, where I talk about the process of a DPIA or a ROPA based on the type of projects and type of data we use.
- K5: The differences between structured and unstructured data.
    - Covered in the Data Architecture at Merlin – Merlin Data section
        - pages 54-55, where I talk about the different types of data within our company datahub
- K6: The fundamentals of data structures, database system design, implementation, and maintenance.
    - Covered throughout the Data Architecture at Merlin section, in the Data Visualization, Data Modelling & Data Science – Data Visualization & Implementation & Prescriptive Analytics sections
        - Pages 54-61, where I talk about different data structures and databases within our data hub, as well as the implementation of new data warehouses
        - Pages 35-41, where I talk about the data structures of all dashboard inputs
- K7: Principles of user experience and domain context for data analytics.
    - Covered in the Merlin Foresight - Customer Focus & UX – Customer Persona, Research on End-User, Communication and MoSCoW sections.
        - Pages 10-11, where I discuss our customer persona and the root cause of their issues in order to find a useful solution
        - Pages 12-13, where I list the Must, Should, Could and Would haves for our users

        - Page 16, where I research our end-user needs and our communication with end-user stakeholders
- K10: Approaches to combining data from different sources.
    - Covered in the Merlin Foresight - Development, Deployment & Support – Model Pipeline & Data Visualization sections
        - Pages 22-24 where I discuss combining different sources of input data to our model (data from logic app emails, azure blob storage, and csv)
        - Pages 35-41 where I discuss combining input data for our dashboards
- K13: Principles of statistics for analysing datasets.
    - Covered in the Merlin Foresight - Development, Deployment & Support – Model Pipeline, Testing, Implementation & Prescriptive Analytics & Foresight Model Conclusions sections
        - Pages 29-35, 45-47, 51-52. Used machine learning and statistical analysis and statistical measures for model builds and accuracy
- K14: The principles of descriptive, predictive, and prescriptive analytics.
    - Covered in the Merlin Foresight - Development, Deployment & Support – Testing, Implementation & Prescriptive Analytics sections
        - Descriptive (pages 29-32): I discuss the process of ranking feature importance and the drivers to actual attendance at our parks
        - Predictive (pages 24-35): I discuss the time-series forecasting configuration and its main regressors to predict future park attendance
        - Prescriptive (pages 45-47); I discuss the analysis of our forecasted results and pricing tiers and the proceeding recommendations.
- K15: The ethical aspects associated with the use of and collation of data.
    - Merlin Foresight - Development, Deployment & Support – Ethical Considerations for Dynamic Pricing
        - Pages 50-51, where I discuss the ethical implications of a dynamic pricing model

Skills

- S5: Assess the impact on user experience and domain context on the data analysis activity.

  - Covered in the Merlin Foresight - Customer Focus & UX – Research on End-User section, Customer Persona & Root Cause Analysis sections
    - Pages 10-11, 16, where I discuss the customer persona for this project, as well as their needs and the root cause of the problem
- S9: Apply organizational architecture requirements to data analysis activities.
  - Covered throughout the Data Architecture at Merlin sections
    - Pages 54-61, where I mention various of the projects at my company where I worked with different Data Architecture at Merlins, both building or modifying different types of databases
- S10: Apply statistical methodologies to data analysis tasks.
  - Covered in the Merlin Foresight - Development, Deployment & Support – Testing and Implementation & Prescriptive Analysis sections
    - Pages 29-35, where I apply statistical predictive methods using two different types of ML models and algorithms, as well as some simple statistics and statistical calculations
- S11: Apply predictive analytics in the collation and use of data.
  - Covered in the Merlin Foresight - Development, Deployment & Support – Testing section
    - Pages 29-35, 45-47. Used a time-series forecast to predict future attendance at some of our parks
- S13: Use a range of analytical techniques such as data mining, time series forecasting, and modelling techniques to identify and predict trends and patterns in data.
  - Covered throughout the Merlin Foresight - Development, Deployment & Support
    - Pages 19-52, where I go into details about some of the data modelling, time-series forecasting and prescriptive analytics for our foresight model
- S14: To collate and interpret qualitative and quantitative data and convert into infographics, reports, tables, dashboards, and graphs.
  - Covered in the Merlin Foresight - Development, Deployment & Support – Dashboard automation & Data Visualization sections

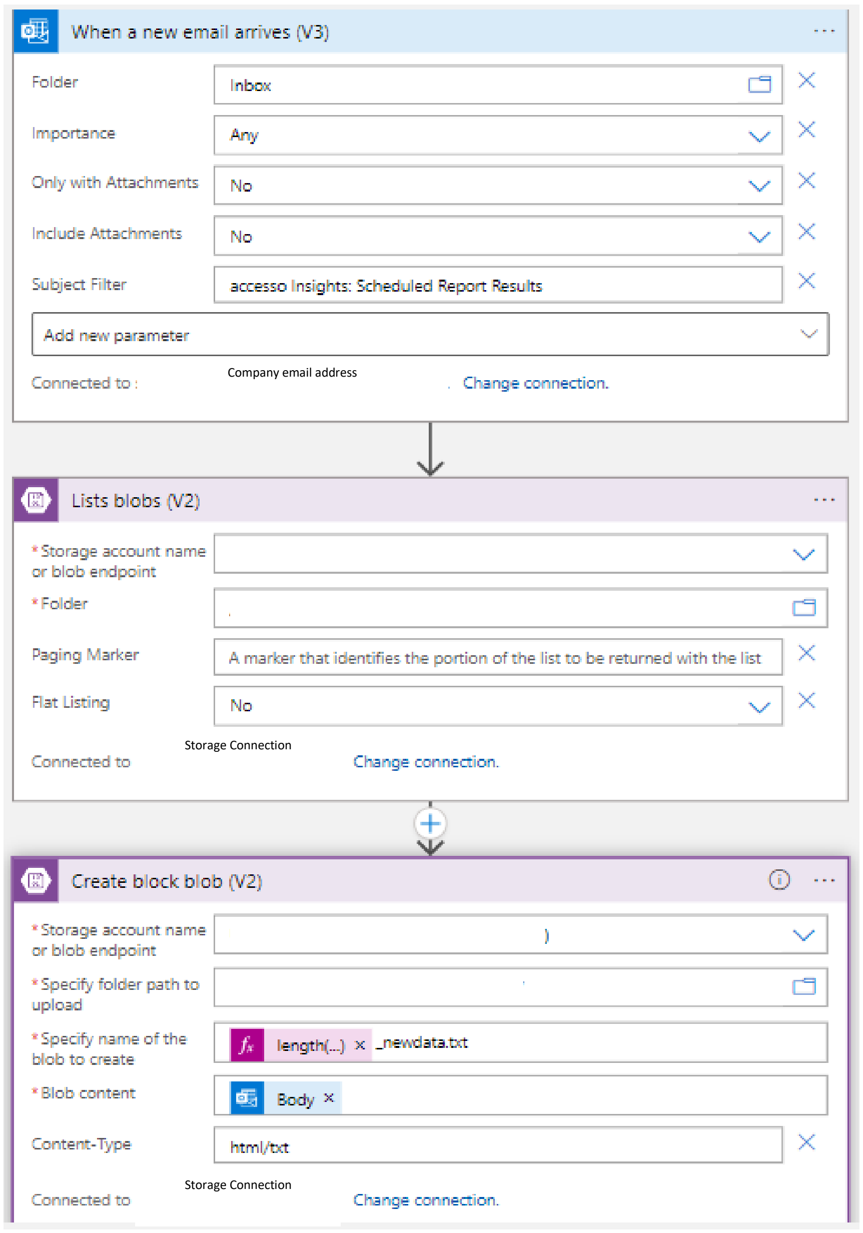
        - Pages 35-45, where I join and transform multiple data sources together within Power BI, before producing various visuals for stakeholder use

Behaviours

- B1: Maintain a productive, professional, and secure working environment.
    - Covered in the Merlin Foresight - Customer Focus & UX – Communication section & Merlin Foresight - Development, Deployment & Support – Stakeholder Feedback
        - Pages 16-17, 48-50, where I talk about my work with both stakeholders and colleagues and add evidence of professional and productive communication
- B2: Shows initiative, being resourceful when faced with a problem and taking responsibility for solving problems within their own remit.
    - Covered in the Merlin Foresight - Development, Deployment & Support – Testing & Foresight Model Conclusions section
        - Pages 28-31, 51-52, where I describe how my POC for Chessington world of adventures has led to a lot of investment in data science at Merlin.
- B5: Identifies issues quickly, enjoys investigating and solving complex problems and applies appropriate solutions. Has a strong desire to push to ensure the true root cause of any problem is found and a solution is identified which prevents recurrence.
    - Merlin Foresight - Development, Deployment & Support – Testing
        - Pages 29-35, where I show how I acted to improve accuracy for some of my models
    - Merlin Foresight - Customer Focus & UX – Root Cause Analysis
        - Page 11-12, were I track down the root cause and find a solution to the stakeholder's problems.
    - Data Architecture at Merlin
        - Pages 54-61, where I find appropriate architectural solutions to each project requirements
- B6: Demonstrates resilience by viewing obstacles as challenges and learning from failure.

  - Covered in the Merlin Foresight - Development, Deployment & Support – Model Pipeline section
    - Pages 19-25, where I describe the issues relating to the lack of a data science environment, and how I built around it using logic apps, email triggers, azure functions and other resources
- B7: Demonstrates an ability to adapt to changing contexts within the scope of a project, direction of the organisation or Data Analyst role.
  - Covered throughout portfolio, as I started as a Trainee Data Developer and have now been promoted to Junior Data Scientist, changing teams and onboarding multiple seniors within my current team.

Appendix



When a new email arrives (V3)

Folder: Inbox
Importance: Any
Only with Attachments: No
Include Attachments: No
Subject Filter: accesso Insights: Scheduled Report Results
Add new parameter
Connected to : Company email address . Change connection.

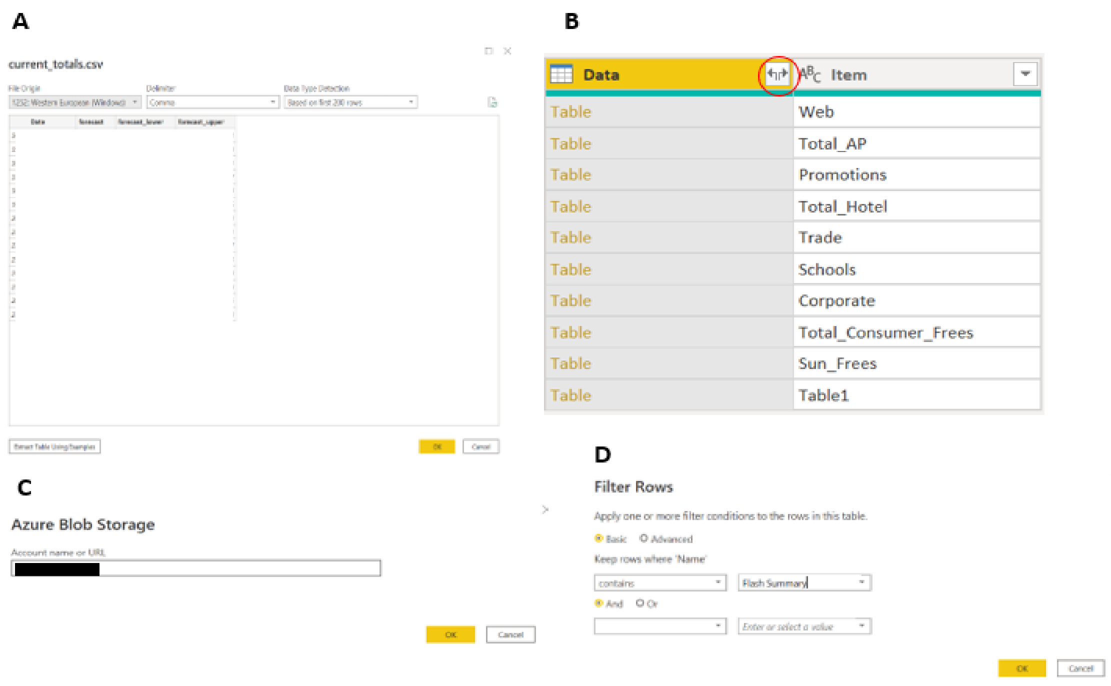
Lists blobs (V2)

*Storage account name or blob endpoint:
*Folder:
Paging Marker: A marker that identifies the portion of the list to be returned with the list
Flat Listing: No
Connected to Storage Connection Change connection.

Create block blob (V2)

*Storage account name or blob endpoint:)
*Specify folder path to upload:
*Specify name of the blob to create: length(...) _newdata.txt
*Blob content: Body
Content-Type: html/txt
Connected to Storage Connection Change connection.

Figure 25: Azure Logic App. Step 1: New email arrives with subject “accesso Insights: Scheduled Report Results”. Step 2: List number of files within output folder. Step 3: convert email to .txt and change name using length of list from previous step (to avoid overwrites). Connection strings and keys have been covered for data privacy. KSB(K6, K10, S14)



current_totals.csv

File Origin: 1252: Western European (Windows)
Delimiter: Comma
Data Type Detection: Based on first 200 rows

Date	forecast	forecast_lower	forecast_upper

Data	Item
Table	Web
Table	Total_AP
Table	Promotions
Table	Total_Hotel
Table	Trade
Table	Schools
Table	Corporate
Table	Total_Consumer_Frees
Table	Sun_Frees
Table	Table1

Azure Blob Storage

Account name or URL: [redacted]

Filter Rows

Apply one or more filter conditions to the rows in this table.

[✓ Basic] Advanced

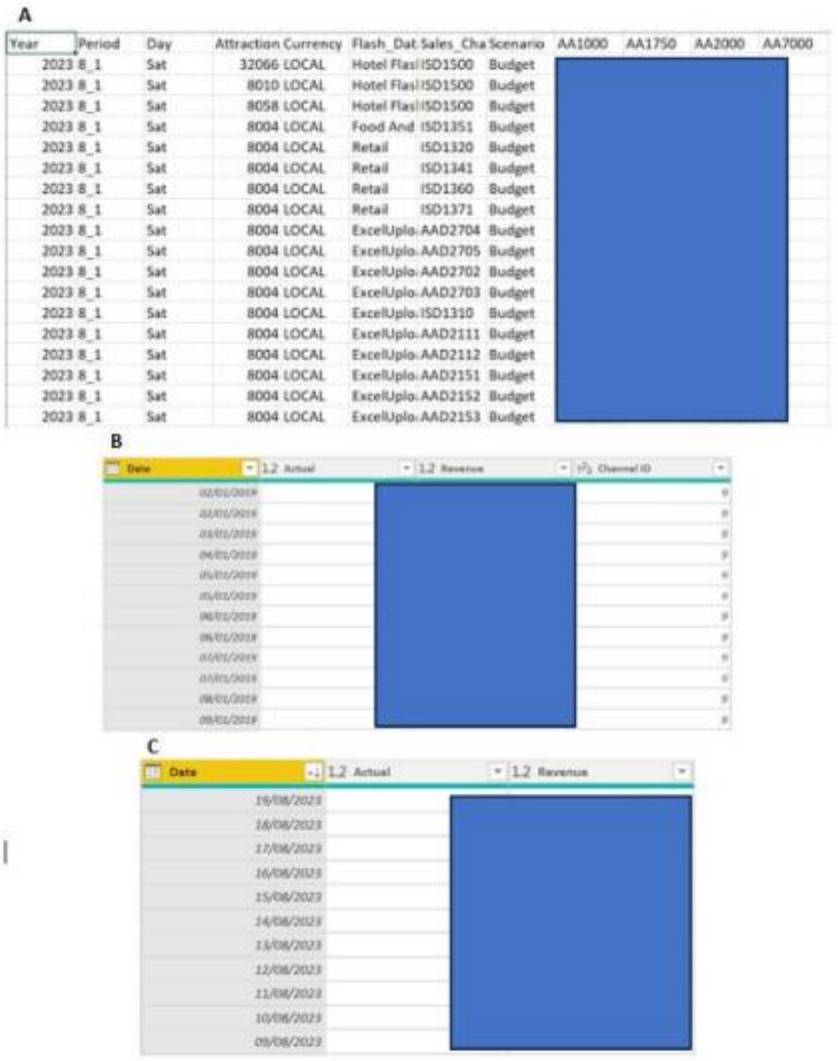
Keep rows where 'Name'

contains: Flash Summary

[✓ And] Or

Enter or select a value

Figure 26: Connecting to data through different sources. A: connecting to .csv files and loading them into Power Query. B: Combining multiple excel sheets into one table. C: Connecting to Azure Blob Storage using account name. D: Filtering out unwanted files from blob storage. KSB(K6, K10, S14)



A

Year	Period	Day	Attraction	Currency	Flash_Dat	Sales_Cha	Scenario	AA1000	AA1750	AA2000	AA7000
2023	8_1	Sat	32066	LOCAL	Hotel Flas	ISD1500	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8010	LOCAL	Hotel Flas	ISD1500	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8058	LOCAL	Hotel Flas	ISD1500	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	Food And	ISD1351	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	Retail	ISD1320	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	Retail	ISD1341	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	Retail	ISD1360	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	Retail	ISD1371	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2704	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2705	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2702	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2703	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	ISD1310	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2111	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2112	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2151	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2152	Budget	[redacted]	[redacted]	[redacted]	[redacted]
2023	8_1	Sat	8004	LOCAL	ExcelUplo	AAD2153	Budget	[redacted]	[redacted]	[redacted]	[redacted]

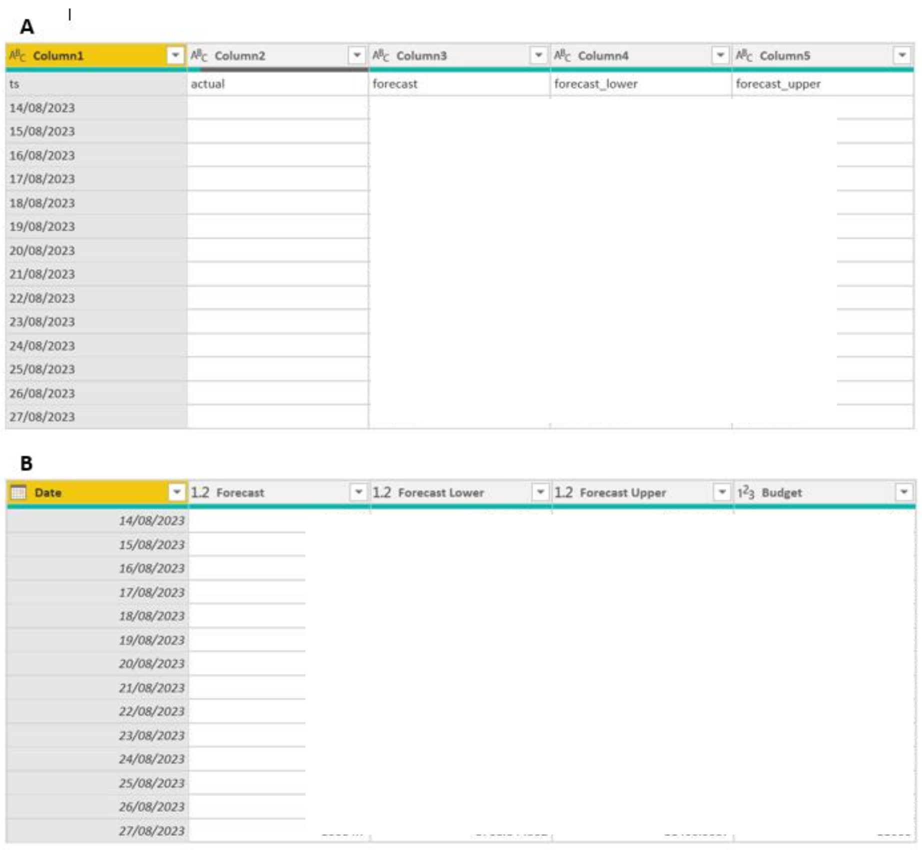
B

Date	1.2 Actual	1.2 Revenue	Channel ID
02/01/2019		[redacted]	9
02/01/2019		[redacted]	9
03/01/2019		[redacted]	9
04/01/2019		[redacted]	9
05/01/2019		[redacted]	9
05/01/2019		[redacted]	9
06/01/2019		[redacted]	9
06/01/2019		[redacted]	9
07/01/2019		[redacted]	9
07/01/2019		[redacted]	9
08/01/2019		[redacted]	9
09/01/2019		[redacted]	9

C

Date	1.2 Actual	1.2 Revenue
19/08/2023	[redacted]	[redacted]
18/08/2023	[redacted]	[redacted]
17/08/2023	[redacted]	[redacted]
16/08/2023	[redacted]	[redacted]
15/08/2023	[redacted]	[redacted]
14/08/2023	[redacted]	[redacted]
13/08/2023	[redacted]	[redacted]
12/08/2023	[redacted]	[redacted]
11/08/2023	[redacted]	[redacted]
10/08/2023	[redacted]	[redacted]
09/08/2023	[redacted]	[redacted]

Figure 27: (A) Example of a flash summary in raw form taken from Blob Storage. Result after filtering out unwanted files within storage. (B) Final table format after transformation of Flash Summaries. Attendance and Revenue split by channel. (C) Grouped attendance by date, no channel split. KSB(K6, K10, S14)

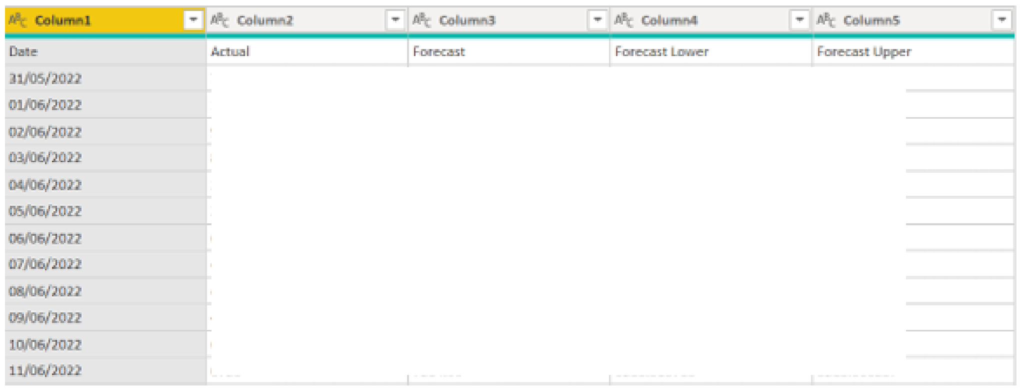


Column1	Column2	Column3	Column4	Column5
ts	actual	forecast	forecast_lower	forecast_upper
14/08/2023				
15/08/2023				
16/08/2023				
17/08/2023				
18/08/2023				
19/08/2023				
20/08/2023				
21/08/2023				
22/08/2023				
23/08/2023				
24/08/2023				
25/08/2023				
26/08/2023				
27/08/2023				

Date	Forecast	Forecast Lower	Forecast Upper	Budget
14/08/2023				
15/08/2023				
16/08/2023				
17/08/2023				
18/08/2023				
19/08/2023				
20/08/2023				
21/08/2023				
22/08/2023				
23/08/2023				
24/08/2023				
25/08/2023				
26/08/2023				
27/08/2023				

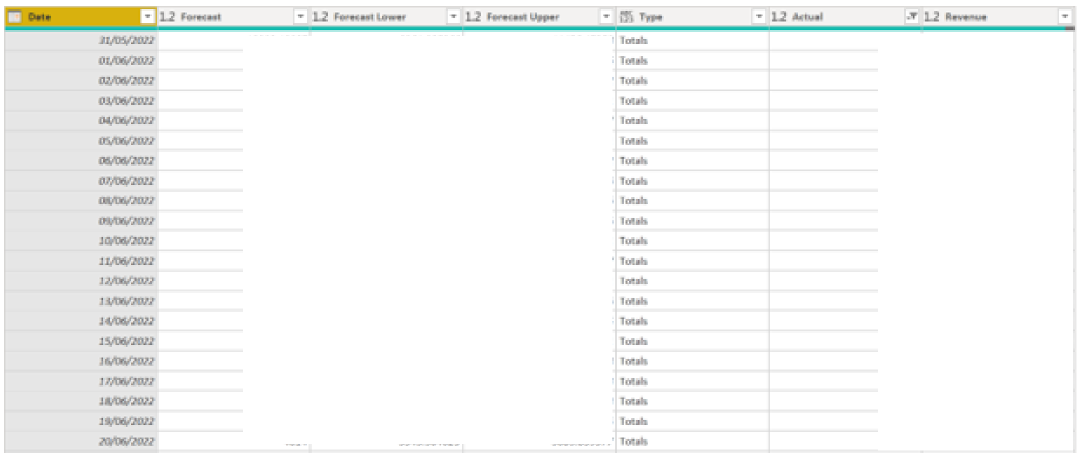
Figure 28: TotAttendance_forecast & TotPrebooks_forecast.csv pre (A) and post (B) transformation. Budget column merged on left. KSB(K6, S14)

A



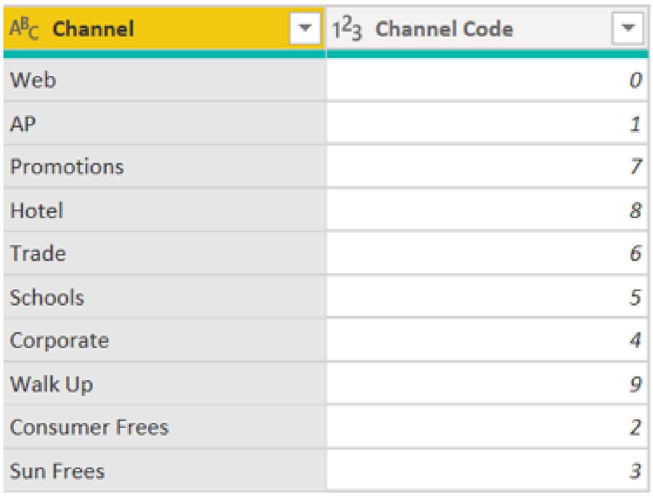
Column1	Column2	Column3	Column4	Column5
Date	Actual	Forecast	Forecast Lower	Forecast Upper
31/05/2022				
01/06/2022				
02/06/2022				
03/06/2022				
04/06/2022				
05/06/2022				
06/06/2022				
07/06/2022				
08/06/2022				
09/06/2022				
10/06/2022				
11/06/2022				

B



Date	1.2 Forecast	1.2 Forecast Lower	1.2 Forecast Upper	Type	1.2 Actual	1.2 Revenue
31/05/2022				Totals		
01/06/2022				Totals		
02/06/2022				Totals		
03/06/2022				Totals		
04/06/2022				Totals		
05/06/2022				Totals		
06/06/2022				Totals		
07/06/2022				Totals		
08/06/2022				Totals		
09/06/2022				Totals		
10/06/2022				Totals		
11/06/2022				Totals		
12/06/2022				Totals		
13/06/2022				Totals		
14/06/2022				Totals		
15/06/2022				Totals		
16/06/2022				Totals		
17/06/2022				Totals		
18/06/2022				Totals		
19/06/2022				Totals		
20/06/2022				Totals		

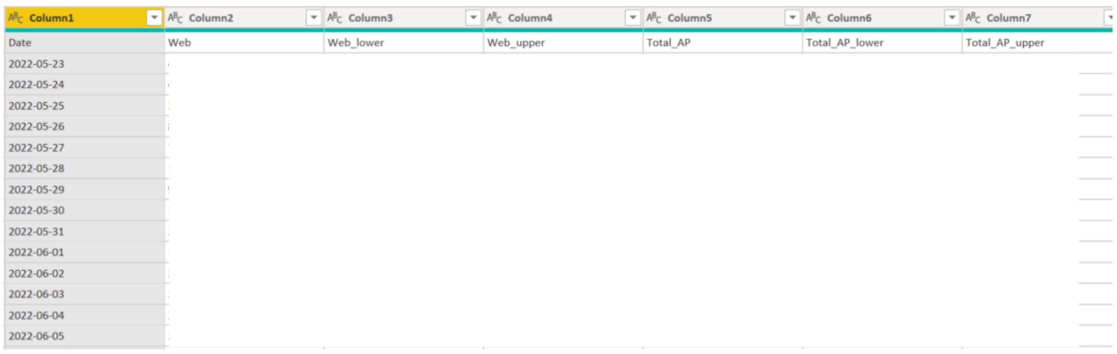
Figure 29: A) Initial table format for historical forecasts (prebooking and attendance). B) Final result after transforming both tables, adding a custom column "Type" and appending one to another. KSB(K6, S14)



Channel	Channel Code
Web	0
AP	1
Promotions	7
Hotel	8
Trade	6
Schools	5
Corporate	4
Walk Up	9
Consumer Frees	2
Sun Frees	3

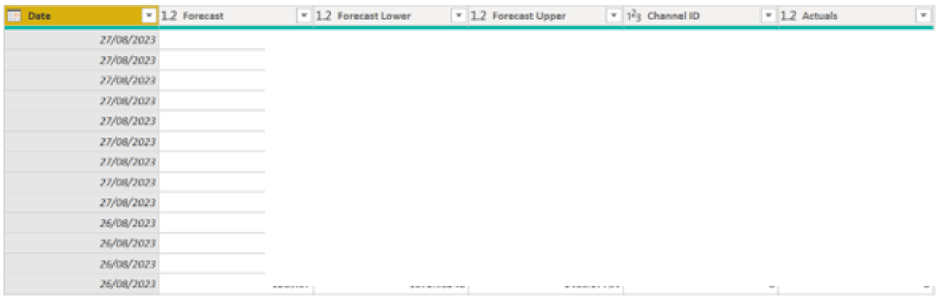
Figure 30: Creating a dictionary of channel types. Referenced initial table (A) and removed duplicates from new table, as well as code column. Added a numbered channel code column (B). KSB(K6, S14)

A



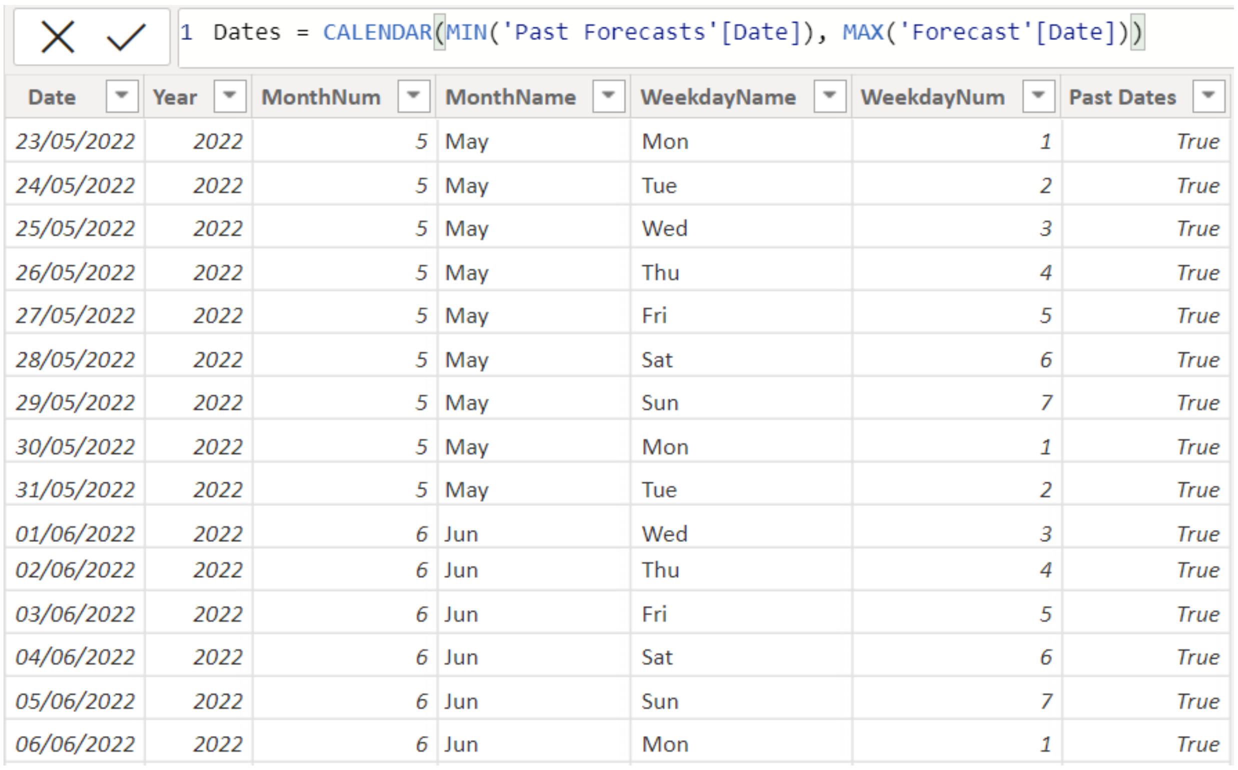
Column1	Column2	Column3	Column4	Column5	Column6	Column7
Date	Web	Web_lower	Web_upper	Total_AP	Total_AP_lower	Total_AP_upper
2022-05-23						
2022-05-24						
2022-05-25						
2022-05-26						
2022-05-27						
2022-05-28						
2022-05-29						
2022-05-30						
2022-05-31						
2022-06-01						
2022-06-02						
2022-06-03						
2022-06-04						
2022-06-05						

B



Date	Forecast	Forecast Lower	Forecast Upper	Channel ID	Actuals
27/08/2023					
27/08/2023					
27/08/2023					
27/08/2023					
27/08/2023					
27/08/2023					
27/08/2023					
27/08/2023					
27/08/2023					
26/08/2023					
26/08/2023					
26/08/2023					
26/08/2023					

Figure 31: Initial table format for channels.xlsx (A). Final result after transformation and merging channel ID onto type (B). KSB(K6, S14)



```
Dates = CALENDAR(MIN('Past Forecasts'[Date]), MAX('Forecast'[Date]))
```

Date	Year	MonthNum	MonthName	WeekdayName	WeekdayNum	Past Dates
23/05/2022	2022	5	May	Mon	1	True
24/05/2022	2022	5	May	Tue	2	True
25/05/2022	2022	5	May	Wed	3	True
26/05/2022	2022	5	May	Thu	4	True
27/05/2022	2022	5	May	Fri	5	True
28/05/2022	2022	5	May	Sat	6	True
29/05/2022	2022	5	May	Sun	7	True
30/05/2022	2022	5	May	Mon	1	True
31/05/2022	2022	5	May	Tue	2	True
01/06/2022	2022	6	Jun	Wed	3	True
02/06/2022	2022	6	Jun	Thu	4	True
03/06/2022	2022	6	Jun	Fri	5	True
04/06/2022	2022	6	Jun	Sat	6	True
05/06/2022	2022	6	Jun	Sun	7	True
06/06/2022	2022	6	Jun	Mon	1	True

Figure 32: Dates table with DAX formula. Created calendar table using minimum and maximum dates of forecast tables as start and end dates. Past forecast dates start in may 2022, while upcoming forecasts end in august 2023. KSB(K6, S14)

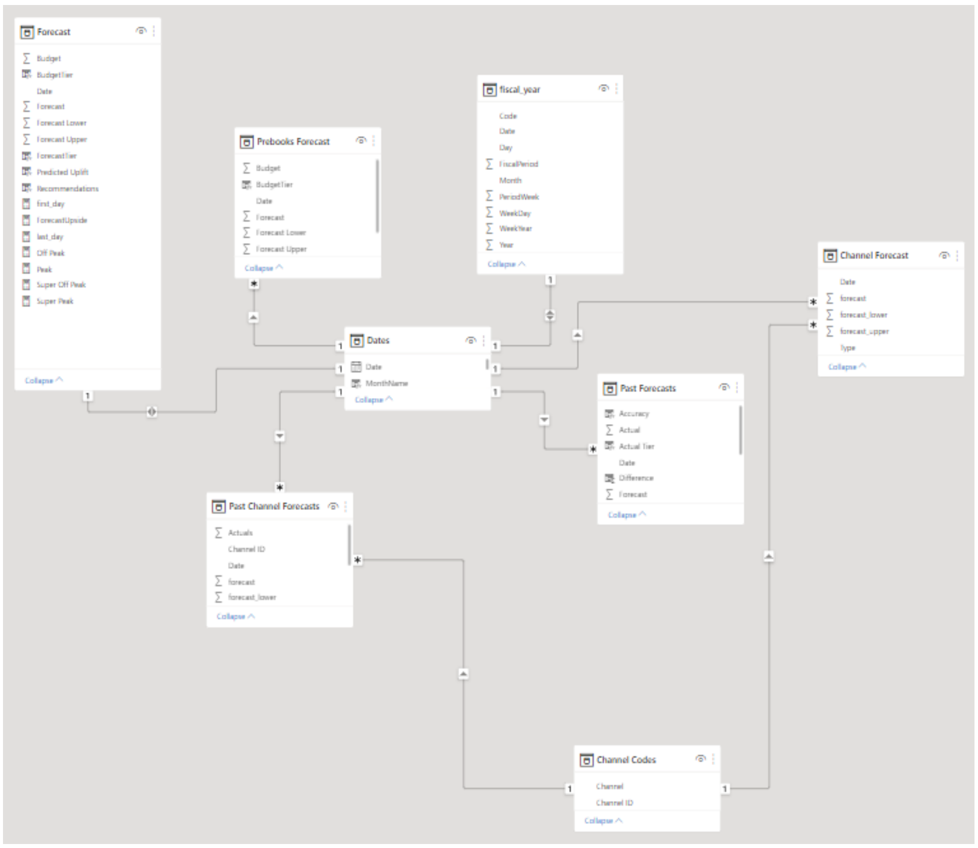
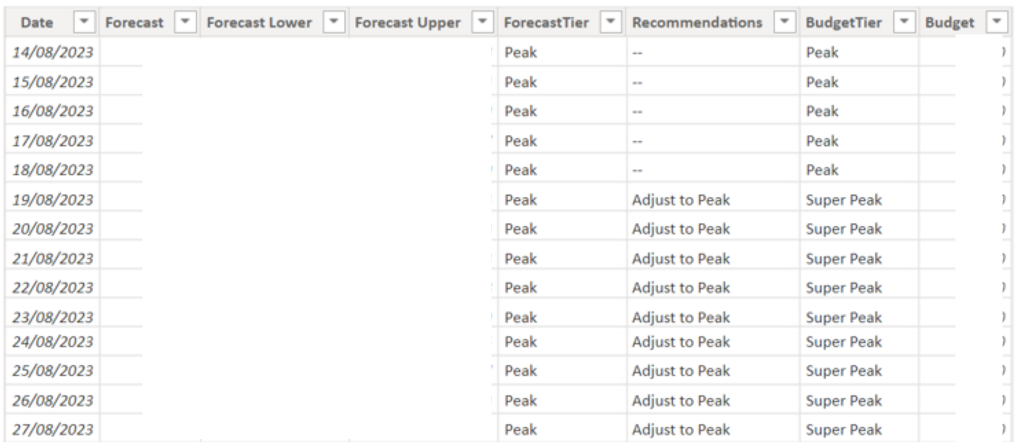


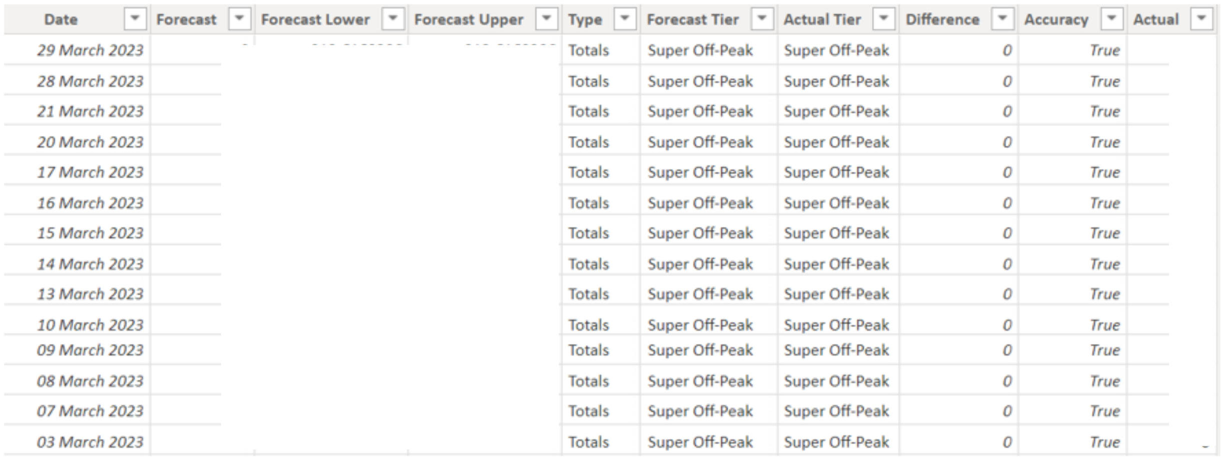
Figure 33: Relationships model. Based on Kimball-style organization, all connected to central dates column. Channel codes used as a dictionary table and therefore does not contain dates. KSB(K6, S9, S14)

A



Date	Forecast	Forecast Lower	Forecast Upper	ForecastTier	Recommendations	BudgetTier	Budget
14/08/2023				Peak	--	Peak	
15/08/2023				Peak	--	Peak	
16/08/2023				Peak	--	Peak	
17/08/2023				Peak	--	Peak	
18/08/2023				Peak	--	Peak	
19/08/2023				Peak	Adjust to Peak	Super Peak	
20/08/2023				Peak	Adjust to Peak	Super Peak	
21/08/2023				Peak	Adjust to Peak	Super Peak	
22/08/2023				Peak	Adjust to Peak	Super Peak	
23/08/2023				Peak	Adjust to Peak	Super Peak	
24/08/2023				Peak	Adjust to Peak	Super Peak	
25/08/2023				Peak	Adjust to Peak	Super Peak	
26/08/2023				Peak	Adjust to Peak	Super Peak	
27/08/2023				Peak	Adjust to Peak	Super Peak	

B



Date	Forecast	Forecast Lower	Forecast Upper	Type	Forecast Tier	Actual Tier	Difference	Accuracy	Actual
29 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
28 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
21 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
20 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
17 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
16 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
15 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
14 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
13 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
10 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
09 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
08 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
07 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	
03 March 2023				Totals	Super Off-Peak	Super Off-Peak	0	True	

Figure 34: Upcoming (A) and historical (B) forecasts. Identical tables split by channels also exist. Added tier, recommendations, difference, and accuracy columns. KSB(K6, S14)