Structured Data Modeling & Analysis (MSBC 5405)

Final Project

By:

Sam Alqarzi, Jack Beck, Isaac Everitt, Cody Rogers, Jordan Waldroop

Team 17

Project Scenario

The theme of our research project is the simulated organization of a realty office in a major metropolis that deals with rental apartments. This office holds many assets, most notably several apartment buildings, each with many tenants. Every building has several individual apartments, each blueprinted to have a certain number of bedrooms and bathrooms. Tenants of each apartment are charged according to individual rates and are to pay their monthly rent to the office for the length of their contract. It is assumed this company has been in business for a while, and as such, there is a history of maintenance and repairs to the apartments along with a history of furnishings, as each apartment comes with furniture. Over time, there have been a number of repairs for different reasons, and some of the furniture has aged and been replaced. The office is assumed to have many roles for staff, as well as staff at each apartment building.

On the business side, the office holds a database with many tables and entries. There are countless relationships that can be made within this data that will help the office prosper in many ways, such as: maintaining proper accounting, assisting efforts in marketing, and even analyzing potential business expansion or missed opportunities. Tables include information on the buildings, individual apartments, past maintenance, furniture, associated expenditures, staff, residents, leases, rental amounts, and payments. Each table holds more associated details to call upon. For example, the resident table not only has information like first name, last name, and email, but also credit score and an unique ID. The building table has its own ID, the street addresses, and binary indicators for having a parking lot or pool.

Our team is building this database for the simulated organization for many reasons. We felt this concept was easy to grasp, but hearty enough to allow for exercising out analytical and database building skills. Having a database centered around a single organization allows for many relationships to be made between several tables as well as displaying different data types. This database is easily expandable, which allowed us to load in thousands of rows of data to work with. We were able to create expansive tables that serve as realistic examples of ones that would be found in a similarly-sized business. Although this database is completely simulated, it still allows us to insert and normalize data, create visualizations, and otherwise ask many meaningful questions and analyze the database.

Our database consists of 10 tables, each serving as a record-keeping container for a distinctive type of information whether qualitative, quantitative, or nominal data. The first 5 of those 10 tables archive data of the acclaimed property, management team and appointed tenants. Each table majorly enfolds descriptive information enlisting personal, demographic and qualitative data about various staff members, residents, apartments and buildings. One of the core tables in the database is the ‘Building’ table. It includes 50 different buildings along with each property’s address, number of apartments, the date in which it was built, and two binary fields indicating whether a particular building has a pool or a parking lot. ‘Building’ table is connected to the whole database via one-many relationship with the ‘Apartment’ table which maps each apartment with its corresponding building.

Similar to the ‘Building’ table, the ‘Apartment’ table enfolds a set of diverse variables presenting descriptive data about each apartment such as apartment number, number of bedrooms and bathrooms and its space in square foot. Each apartment is linked to one or many tenants in the ‘Resident’ table through a one-many relationship. This is the only link of the ‘Resident’ table to the entire database. ‘Resident’ table has about 8 columns scripting both personal and demographic information of all residents in the database. The fourth table of those 5 tables is the ‘Office’ table which handles all staff members’ information such as, their names, gender, emails, and their role in the management team. Office table has a one-many relationship to the ‘Expenditure’ table which captures its only connection to the database represented by wage expenses with their related ‘StaffID’ in the ‘Expenditure’ table. The last table of those 5 tables is the ‘Lease’ table which is a descriptive table with only three qualitative fields offering information about each of three lease types whether the lease is 3, 6 or 12 -months lease and its corresponding start and end date.

The other 5 tables cultivate the financial aspect of the business ranging from rent payment to expense allocations. For instance, the ‘Rent’ table is the third most connected table after the ‘Apartment’ and ‘Expenditure’ tables, which incorporates rent information (e.g. apartment number and lease type) gathered from its one-many relationships with the ‘Lease’ and ‘Apartment’ tables. Then, it propagates all that information to the last stop of the rent journey in the ‘payment’ table through one-many relationship. This relationship is the only link between the ‘payment’ table and the whole database which explains the main purpose of this table which is to be the final stage of the business’ revenue source.

The other three tables are mainly concerned with expenses such as maintenance, wages and furniture. Both maintenance and furniture expenditures are forged in two separate tables in which each enlists the expense’s required information such as type mapping each to their corresponding ‘ApartmentID’ as foreign key in those tables via one-many relationship. The last table is the ‘Expenditure’ table which accumulates all types of expenses whether it is maintenance, furniture procurement or paid wages to office staff. This table has three foreign keys directed to ‘Office’, ‘Maintenance’ and ‘Furniture’ tables linking each type of expenditure to its correct allocation.

Of the data used for this database project, a small amount of the data is taken from public data libraries, specifically credit score data, which was obtained from www.kaggle.com, while the majority of the data is mock data created by hand or by using Mockaroo. The mock data allowed us to test the functionality and relationships of our database without having to find specific data that perfectly fits our mock business model. The most important factor when populating our tables with data was to ensure that the various ID’s in each table were consistent throughout the database (as these represent the keys between tables) allowing us to ensure that the DB’s relationships were preserved.

The ‘ApartmentID’ field is perhaps the most important ID used in the database, as it is the primary key of the database’s central table ‘Apartment’. Each apartment ID is intended to represent 1 unit in its respective building. The ID’s arbitrarily start at ID 100 and extend out for a

total of 5000 ID's, each corresponding to one unit, for a total of 5000 individual apartments. The ID's were grouped into groups of 100, and each group was assigned a 'buildingID', which is the primary key of the 'building' table. This comes out to a total of 50 buildings owned by our mock company, each with 100 individual apartments. By having the building ID linking each apartment unit to a specific building, we can effectively relate each unit and each building to specific residents, their furniture rentals, maintenance requests, as well as rent payments and lease information.

Because most of our data was mock data, few data transformations were required. The most frequent issues we encountered that required data transformations were improper data types being assigned to fields as a result of our top-down database design approach. For example, under the 'resident' table, the field 'CreditScore' was originally imputed as a varchar(45) data type, however a credit score is a whole number, and as a result the data type needed to be changed to an integer. Many of these errors were found while writing our SELECT statements, and the appropriate transformations were made as needed, using MySQL DDL commands.

One of the only other issues we ran into while implementing the data into the database came from trying to load the data into the 'expenditure' table. Prior to this table, we had used the LOAD DATA INFILE script to load our data. However, we ran into issues with the 'expenditure' table because of three fields in the table: 'maintenanceID', 'StaffID', and 'furnitureID'. These began throwing errors because some of the ID values were null, as each expenditure could only have one ID, corresponding to whether the expenditure was for maintenance, staff, or furniture. Anything written in these positions would be treated as a string by SQL and not a NULL, whether it was a blank space or if null was written out. To get around this issue, we used the built in MySQL data import wizard, which treated the blanks in the .csv as proper NULL values.

When building our database, we built it in a way where most tables were naturally in third natural form (3NF). There are a few areas where we could improve the database in the future -- namely separating the "role" column out of the "Office" table into its own table and restructuring the "Rent" table where "LeaseID" and "ApartmentID" are a composite primary key instead of "RentID".

While the database was built naturally in 3NF, theoretically it is possible for this database to be in Unnormalized Form (UNF) through 3NF (e.g., UNF, 1NF, 2NF, 3NF). The primary key that would allow this is "ApartmentID" and "BuildingID", which work together to be the key attributes of the unnormalized table. "ApartmentID" is the key to many tables, but there are multiple buildings with the same apartment numbers, which is why it is necessary for "BuildingID" to be included as a key attribute.

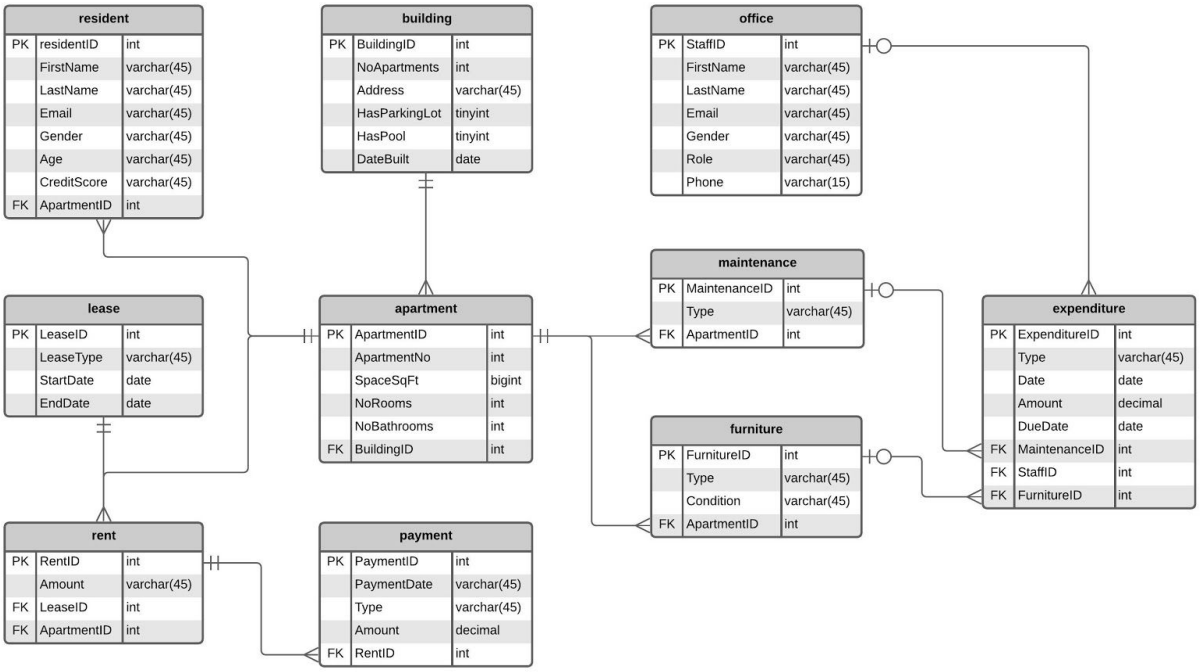
Creating a new "Role" table would allow us to transition the "Office" table from 2NF to 3NF. Currently, there are multiple staff that have the same role within the company. While not unusual from a business perspective, this means that the current "role" creates redundancies within the table. When creating a new "Role" table, each role would have its own ID and then each staff member in the "Office" table would be assigned a "RoleID". You could then also set

up restrictions for singular roles so that multiple people do not have the “CEO” or “COO” role. This would be important should future iterations of the database use this table to link to permissions within the company. Should someone within the company get a promotion, you could easily update their “RoleID” and they would gain any new permissions associated with that role.

We could also add a “salary” column to the “Role” table. While we are able to see the bi-weekly deposits that are sent to each of our employees in the ‘Expenditure’ table, it would also be helpful to see their overall base yearly salary. This would ensure that we are paying employees of the same title the same regardless of gender.

We could further normalize the database by restructuring the “Rent”, “Payment”, and “Lease” tables by removing the redundancy of having both “RentID” and “LeaseID”. These two fields are not different data or identifying different aspects of the lease, so they could all be consolidated under/changed to “LeaseID” and remove “RentID” from the tables. This would require a restructuring of the “Rent” and “Payment” tables to change the foreign key from “RentID” to “LeaseID”.

Database ER Diagram Screenshot



List of Business Questions

Jack Beck:

- What is the average SqFt for apartments in building 1? Customers would like this information to compare their units to the other units in the same building.

```
19    ## Jack Beck ##
20
21    -- Q1: What is the average SqFt for apartments in building 1? Customers would like this information to compare their units to the other units in the same building.
22    SELECT
23    AVG(SpaceSqFt) as Average_SqFt_All_Units
24    FROM apartment
25    WHERE BuildingID = 1;
26
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Average_SqFt_All_Units
1065.0700

- What is the average number of rooms in each unit where SqFt > 1000?
  Management would like this done to inform rent adjustments for the next leasing season.

```
27    -- Q2: What is the average number of rooms in each unit where SqFt > 1000? Management would like this done to inform rent adjustments for the next leasing season.
28    # The 'FLOOR' fucntion is used to round down the average, as in practice there is no fractional rooms.
29    SELECT
30    FLOOR(AVG(NoRooms)) as Average_No_Rooms
31    FROM apartment
32    WHERE SpaceSqFt > 1000;
33
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Average_No_Rooms
2

- Which apartments have the highest average Sq Ft per unit?
  Rank all apartment buildings by their average SqFt per unit. This information will be compared to other local apartment leasing options to inform rent adjustments for the next leasing season.

```
34    -- Q3: Which apartments have the highest average Sqft per unit? Rank all apartment buildings by their average SqFt per unit.
35    -- This information will be compared to other local apartment leasing options to inform rent adjustments for the next leasing season.
36
37 *  SELECT
38    RANK() OVER( ORDER BY AVG(SpaceSqFt) DESC) as 'Rank' , BuildingID, AVG(SpaceSqFt) as Avg_SqFt
39    FROM apartment
40    GROUP BY BuildingID;
41
42    -- Q4: What are the construction years of the buildings with the top 3 highest average Sqft per unit? The marketing team would like t
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Rank	BuildingID	Avg_SqFt
1	43	1118.4800
2	5	1114.8600
3	3	1107.8100
4	38	1080.7400
5	40	1078.6200
6	33	1066.3000
7	1	1065.0700
8	4	1063.7300
9	16	1038.2900
10	39	1038.2600
11	21	1036.9800
12	2	1031.3700
13	8	1029.7900
14	29	1025.4200
15	45	1023.7100
16	18	1022.3400
17	6	1020.8700
18	22	1017.6700

Result 3

- What are the construction years of the buildings with the top 3 highest average Sq Ft per unit?
  The marketing team would like this information to appropriately advertise the buildings.

```
42    -- Q4: What are the construction years of the buildings with the top 3 highest average Sqft per unit? The marketing team would like this information to appropri
43 *  SELECT
44    RANK() OVER( ORDER BY AVG(a.SpaceSqFt) DESC) as 'Rank', AVG(a.SpaceSqFt) as Avg_SqFt, a.BuildingID, YEAR(b.DateBuilt) as 'Year Built'
45    FROM apartment as a
46    INNER JOIN building as b ON a.BuildingID = b.BuildingID
47    GROUP BY BuildingID
48    LIMIT 3;
49
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Rank	Avg_SqFt	BuildingID	Year Built
1	1118.4800	43	2009
2	1114.8600	5	2004
3	1107.8100	3	2016

- List the length of time between each maintenance expenditure assigned date and due date. Show the ApartmentID associated with each maintenance expense. Management would like this information queried so that they can plan future expenses more efficiently.

```
50    -- Q5: List the length of time between each maintencance expendtiures assigned date and due date. Show the ApartmentID associated with each maintenance exper
51 •  SELECT
52    a.ApartmentID, e.Date as 'Maintenance Expense Date', e.DueDate, DATEDIFF(e.DueDate , e.Date) as 'Days Until Due'
53    FROM expenditure as e
54    INNER JOIN maintenance as m ON e.MaintenanceID = m.MaintenanceID
55    INNER JOIN apartment as a ON m.ApartmentID = a.ApartmentID
56    WHERE e.Type = 'Maintenance'
57    ORDER BY e.Date;
58
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

ApartmentID	Maintenance Expense Date	DueDate	Days Until Due
1354	2015-11-17	2016-01-09	53
4735	2015-11-20	2016-01-15	56
558	2015-11-24	2016-01-05	42
555	2015-11-24	2015-12-26	32
2821	2015-11-24	2016-01-17	54
1478	2015-11-25	2016-01-18	54
1639	2015-11-29	2016-01-21	53
3822	2015-11-29	2016-01-04	36
2147	2015-11-29	2015-12-23	24
1864	2015-11-30	2016-01-02	33
2912	2015-11-30	2015-12-21	21
3304	2015-12-01	2015-12-20	19
3292	2015-12-01	2015-12-16	15
447	2015-12-06	2015-12-17	11
534	2015-12-06	2016-02-02	58
712	2015-12-08	2016-01-09	32
4991	2015-12-08	2016-01-13	36

Result 5

Jordan Waldroop:

- Which buildings have apartments with both parking lots and pools and how many bedrooms? Potential residents may be looking to be close to both.

```
18    # which buildings have apartments with both parking lots and pools and how many bedrooms?
19    # potential residents may be looking to be close to both
20
21 •  select BuildingID, NoRooms, HasParkingLot, HasPool, count(distinct ApartmentID) as `# of Apartments`
22    from building
23    join apartment using (BuildingID)
24    where HasParkingLot = 1 AND HasPool = 1
25    group by BuildingID, NoRooms
26    order by BuildingID, NoRooms;
27
28
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

BuildingID	NoRooms	HasParkingLot	HasPool	# of Apartments
1	1	1	1	27
1	2	1	1	23
1	3	1	1	23
1	4	1	1	27
5	1	1	1	25
5	2	1	1	21
5	3	1	1	22
5	4	1	1	32
8	1	1	1	31
8	2	1	1	24
8	3	1	1	15
8	4	1	1	30
10	1	1	1	21
10	2	1	1	28
10	3	1	1	19
10	4	1	1	32
11	1	1	1	26
11	2	1	1	30

- What is the average credit score of the residents for each apartment size? This could be important to owners/property managers looking to screen applicants.

```
30
31      # what is the average credit score of the residents for each apartment size?
32      # this could be important to owners/property managers looking to screen applicants.
33
34 •    select NoRooms, FLOOR(avg(CreditScore)) as Average_Credit_Score
35      from resident
36      join apartment using (ApartmentID)
37      group by NoRooms;
38
39
40
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

NoRooms	Average_Credit_Score
1	650
3	649
4	651
2	649

- How many of each lease types (length) by the number of rooms in each apartment? This is important to management for advertising/lease management purposes.

```
42      # how many of each lease types (length) by number of rooms in each apartment?
43      # this is important to management for advertising/lease management purposes.
44
45 •    select NoRooms, LeaseType, count(distinct LeaseID) as `# of Leases`
46      from lease
47      join rent using (LeaseID)
48      join apartment using (ApartmentID)
49      group by NoRooms, LeaseType
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

NoRooms	LeaseType	# of Leases
1	12 Month	780
1	6 Month	270
1	3 Month	267
2	12 Month	761
2	6 Month	266
2	3 Month	244
3	12 Month	725
3	3 Month	235
3	6 Month	228
4	12 Month	736
4	3 Month	254
4	6 Month	234

- Which apartments have new furniture? used furniture? per room number count? This is important for audits of when new furniture will need to be purchased to replace the used furniture.

```
55    # which apartments have new furniture? used furniture? per room number count?
56    # this is important for audits of when new furniture will be need to be purchase to replace the used furniture
57
58 •  select furniture.Condition as `Condition`,
59        count(distinct ApartmentID) as `No of Apartments`,
60        (select round(avg(amount),2)
61        from furniture join expenditure using (FurnitureID)
62        where expenditure.type = 'Furniture') as `Average Cost`
63    from furniture
64    join expenditure using (FurnitureID)
65    group by `Condition`;
66
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Condition	No of Apartments	Average Cost
New	528	749.59
Used	522	749.59

- What is the average rent for each bedroom and bathroom size? This is important for leasing/advertising information.

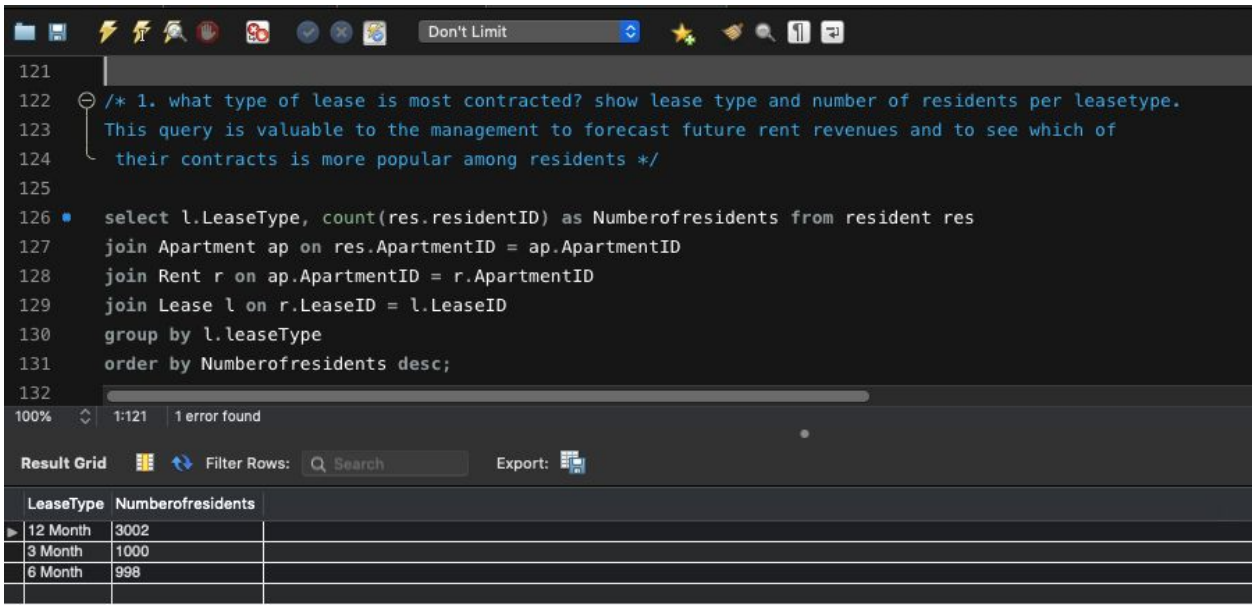
```
67
68    # what is the average rent for each bedroom and bathroom size?
69    # this is important for leasing information
70
71 •  select NoRooms, NoBathrooms, round(avg(amount),2) as `Avg Payment`
72    from rent
73    join apartment using (ApartmentID)
74    group by NoRooms, NoBathrooms
75    order by NoRooms, NoBathrooms;
76
77
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

NoRooms	NoBathrooms	Avg Payment
1	1	2382.67
1	2	2355.16
2	1	2369.08
2	2	2340.97
3	1	2321.42
3	2	2360.47
4	1	2301.03
4	2	2378.77

Sam Al Qarzi:

- What type of lease is most contracted? show lease type and number of residents per lease type. This query is valuable to the management to forecast future rent revenues and to see which of their contracts is more popular among residents.



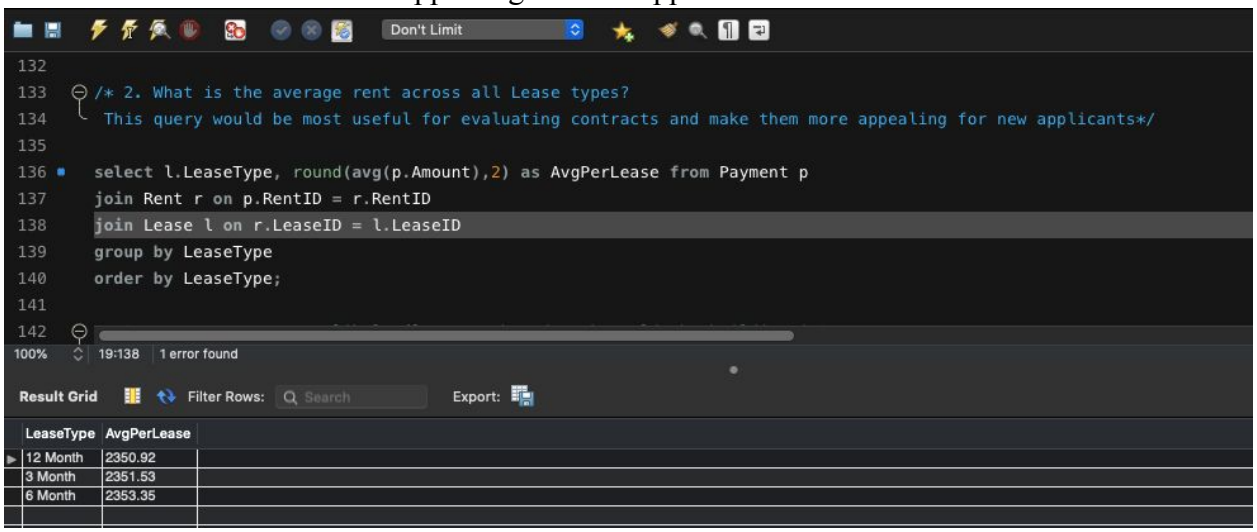
```
121
122  /* 1. what type of lease is most contracted? show lease type and number of residents per leasetype.
123  This query is valuable to the management to forecast future rent revenues and to see which of
124   their contracts is more popular among residents */
125
126  select l.LeaseType, count(res.residentID) as Numberofresidents from resident res
127  join Apartment ap on res.ApartmentID = ap.ApartmentID
128  join Rent r on ap.ApartmentID = r.ApartmentID
129  join Lease l on r.LeaseID = l.LeaseID
130  group by l.leaseType
131  order by Numberofresidents desc;
132
```

100% 1:121 1 error found

Result Grid Filter Rows: Search Export:

LeaseType	Numberofresidents
12 Month	3002
3 Month	1000
6 Month	998

- What is the average rent across all Lease types? This query would be most useful for evaluating contracts and make them more appealing for new applicants.



```
132
133  /* 2. What is the average rent across all Lease types?
134   This query would be most useful for evaluating contracts and make them more appealing for new applicants*/
135
136  select l.LeaseType, round(avg(p.Amount),2) as AvgPerLease from Payment p
137  join Rent r on p.RentID = r.RentID
138  join Lease l on r.LeaseID = l.LeaseID
139  group by LeaseType
140  order by LeaseType;
141
142
```

100% 19:138 1 error found

Result Grid Filter Rows: Search Export:

LeaseType	AvgPerLease
12 Month	2350.92
3 Month	2351.53
6 Month	2353.35

- Does average rent likely fluctuate based on how old the building is? This query would expose any changes in rent across all buildings. Also, it will help the management adjust the rent based on the building's condition.

```
/* 3. Does average rent likely fluctuate based on how old the building is?
   This query would expose any changes in rent across all buildings.
   Also, it will help the management adjust the rent based on the building's condition*/

select b.BuildingID,
       b.DateBuilt,
       round((DATEDIFF(curDate(),b.DateBuilt)/365),0) as BuildingAgeYears,
       round(avg(r.Amount),2) as AvgRent
       from building b
 join Apartment p on b.BuildingID = p.BuildingID
 join Rent r on p.ApartmentID = r.ApartmentID
 group by b.BuildingID
 order by DateBuilt desc;
```

100% 25:151 1 error found

Result Grid Filter Rows: Search Export:

BuildingID	DateBuilt	BuildingAgeYears	AvgRent
22	2020-05-24	1	2353.61
46	2019-10-15	1	2399.20
34	2019-07-18	1	2302.91
28	2019-01-16	2	2347.32
32	2018-06-18	2	2353.61
50	2018-06-03	3	2407.37
48	2016-10-18	4	2347.32
3	2016-01-10	5	2212.74
23	2015-12-04	5	2212.74
15	2015-04-08	6	2369.53
19	2015-02-20	6	2455.35
44	2014-11-08	6	2302.91
8	2014-08-29	6	2347.32

- How many Maintenance Jobs were carried each Year per apartment? and what are their total costs? What is the average cost across all jobs? This will help the management monitor the cost of maintenance jobs across apartments and try to implement permanent solutions.

```
/* 4. How many Maintenance Jobs were carried each Year per apartment? and what are their total costs?
    what is the average cost across all jobs?
    This will help the manaement monitor cost of maintenance jobs across apartments and try to implement permenant solutions. */

select  m.ApartmentID,
        year(E.Date) as JobYear,
        count(m.MaintenanceID) as MatJobs,
        sum(E.amount) as totalcost,
        round((select sum(E.amount)/count(m.MaintenanceID) from Maintenance m
         join Expenditure E on m.MaintenanceID = E.MaintenanceID),2) as AvgCost
from Maintenance m
join Expenditure E on m.MaintenanceID = E.MaintenanceID
group by m.ApartmentID, JobYear
order by m.ApartmentID, JobYear;
```

100% 19:165 1 error found

Result Grid Filter Rows: Search Export: Fetch rows:

ApartmentID	JobYear	MatJobs	totalcost	AvgCost
111	2016	1	731.98	1049.50
119	2018	1	502.04	1049.50
120	2016	1	467.14	1049.50
120	2018	1	584.88	1049.50
124	2020	1	1428.76	1049.50
140	2016	1	1985.76	1049.50
140	2018	1	735.01	1049.50

- Were most of those Maintenance Jobs done in new or old buildings? What are the total number and cost of maintenance jobs in each building? This will help the management allocate maintenance cost on a building scale and determine if those expenses are most likely associated with a building's condition.

```
/* 5. Were most of those Maintenance Jobs done in new or old buildings?
  what are the total number and cost of maintenance jobs in each buliding?
  This will help the management allocate maintenance cost on a building scale and
  determine if those expense are most likely associated with a building's condition. */

select b.BuildingID,
       b.DateBuilt,
       round((DATEDIFF(curDate(),b.DateBuilt)/365),0) as BuildingAgeYears,
       count(m.MaintenanceID) as NoMatJobs,
       sum(E.amount) as totalcost
from  building b
join apartment p on b.BuildingID = p.BuildingID
join Maintenance m on p.ApartmentID = m.ApartmentID
join Expenditure E  on m.MaintenanceID = E.MaintenanceID
group by b.BuildingID
order by b.DateBuilt desc;
```

100% 3:175 1 error found

Result Grid Filter Rows: Search Export: Fetch rows:

ApartmentID	JobYear	MatJobs	totalcost	AvgCost
101	2016	1	1802.46	1049.50
104	2019	1	1581.90	1049.50
111	2016	1	731.98	1049.50
119	2018	1	502.04	1049.50
120	2016	1	467.14	1049.50
120	2018	1	584.88	1049.50
124	2020	1	1428.76	1049.50
140	2016	1	1985.76	1049.50
140	2018	1	735.01	1049.50

Cody Rogers:

- There has been a manufacturer recall on the new lamps that you have been installing, which states that the lamps are prone to overheating leading to fire. Obtain a list of all apartments with new lamps, so they can be replaced, and so current residents can be warned.

```
#Q1
#there has been a manufacturer recall on the new lamps that you have been installing,
    #which states that the lamps are prone to overheating leading to fire.
#obtain a list of all apartments with new lamps, so they can be replaced, and so current residents can be warned
select residentID, FirstName, LastName, Email, f.ApartmentId, f.type, f.condition
from furniture f
join apartment a on a.apartmentID = f.ApartmentID
join Resident r on  r.ApartmentID = a.ApartmentID
where f.type = 'lamps' and f.condition = 'New';
```

Result Grid Filter Rows: Export: Wrap Cell Content:

residentID	FirstName	LastName	Email	ApartmentId	type	condition
2690	Sally	Butler	sally.butler@team17rental.com	2789	lamps	New
3872	Nicola	Rees	nicola.rees@team17rental.com	3971	lamps	New
3641	Julia	Underwood	julia.underwood@team17rental.com	3740	lamps	New
2993	Ella	Wilson	ella.wilson@team17rental.com	3092	lamps	New
2226	Edward	Cameron	edward.cameron@team17rental.com	2325	lamps	New
4511	Carolyn	Grant	carolyn.grant@team17rental.com	4610	lamps	New
1417	Joshua	Wright	joshua.wright@team17rental.com	1516	lamps	New
1568	Christopher	Sanderson	christopher.sanderson@team17rental.com	1667	lamps	New
2924	Ruth	Hughes	ruth.hughes@team17rental.com	3023	lamps	New
2470	Robert	Hudson	robert.hudson@team17rental.com	2569	lamps	New
2835	Stephanie	Marshall	stephanie.marshall@team17rental.com	2934	lamps	New
3434	Alexandra	Davies	alexandra.davies@team17rental.com	3533	lamps	New
1858	Joseph	McLean	joseph.mclean@team17rental.com	1957	lamps	New
3235	Samantha	Edmunds	samantha.edmunds@team17rental.com	3334	lamps	New
645	Max	Forsyth	max.forsyth@team17rental.com	744	lamps	New
1078	Julian	Burgess	julian.burgess@team17rental.com	1177	lamps	New
1487	Sam	Davidson	sam.davidson@team17rental.com	1586	lamps	New
3296	Felicity	Hardacre	felicity.hardacre@team17rental.com	3395	lamps	New
2104	Max	Glover	max.glover@team17rental.com	2203	lamps	New

Result 1

- Accounting wants to know if there have been any underpayments by residents, to make sure their collection books are correct and current.

```
#accounting wants to know if there have been any underpayments by residents,
    #to make sure their collection books are correct and current
select r.rentID, apartmentID, r.amount as rent_amount, p.amount as payment,
    r.amount-p.amount as amount_owed, paymentdate, type  from rent r
join payment p
on r.rentID = p.rentID
where r.amount-p.amount > 0;
#this should return no rows, as all accounts are fully paid
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

rentID	apartmentID	rent_amount	payment	amount_owed	paymentdate	type

- The office staff need to know which apartment's leases will be ending next month, so they can prepare renewal paperwork or advertise the units for new tenants. Assume current date is May 2019 to find apartments that will be renewing in the current month (resign, find new tenants)

```
#Q3
#the office staff need to know which apartments leases will be ending next month,
    #so they can prepare renewal paperwork or advertize the units for new tenants
    #assume current date is may 2019
    #to find apartments that will be renewing in the current month (resign, find new tenants)
select l.LeaseID, l.EndDate, res.ResidentID, res.ApartmentID from Lease l
join rent r on r.LeaseID = l.LeaseID
join Apartment a on a.ApartmentID = r.ApartmentID
join Resident res on res.ApartmentID = a.apartmentID
where MONTH(EndDate) = 6 and YEAR(EndDate) = 2019;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

LeaseID	EndDate	ResidentID	ApartmentID
400	2019-06-14	1	100
404	2019-06-14	2	101
408	2019-06-14	3	102
412	2019-06-14	4	103
416	2019-06-14	5	104
420	2019-06-14	6	105
424	2019-06-14	7	106
428	2019-06-14	8	107
432	2019-06-14	9	108
436	2019-06-14	10	109
440	2019-06-14	11	110
444	2019-06-14	12	111
448	2019-06-14	13	112
452	2019-06-14	14	113
456	2019-06-14	15	114
460	2019-06-14	16	115
464	2019-06-14	17	116
468	2019-06-14	18	117
472	2019-06-14	19	118

Result 3

- The maintenance operating manager wants to know which apartments had water damage repaired, and when, so the apartments can be tested for standing mold/failed repairs.

```
#Q4
#the maintanance operating manager wants to know which apartments had water damage repaired, and when,
    #so the apartments can be tested for standing mold/failed repairs

select m.MaintenanceID, m.type as maintenance_type, m.ApartmentID, ex.Date, ex.Amount from maintenance m
join expenditure ex on ex.MaintenanceID = m.MaintenanceID
where m.type like 'leaks%'
order by ApartmentID, date desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MaintenanceID	maintenance_type	ApartmentID	Date	Amount
2938	Leaks and Water Damage	119	2018-04-26	502.04
1257	leaks and water damage	149	2020-05-30	196.12
1257	leaks and water damage	149	2019-11-15	1701.63
1257	leaks and water damage	149	2017-05-31	1801.92
1581	leaks and water damage	247	2020-08-31	666.63
1581	leaks and water damage	247	2017-12-06	337.84
1581	leaks and water damage	247	2016-10-24	842.05
2861	Leaks and Water Damage	325	2016-04-18	1798.41
2202	Leaks and Water Damage	372	2018-10-15	1495.87
2202	Leaks and Water Damage	372	2017-09-20	274.28
2202	Leaks and Water Damage	372	2017-04-04	1751.09
1605	leaks and water damage	385	2018-04-22	1019.46
1456	leaks and water damage	403	2017-02-28	447.55
2689	Leaks and Water Damage	421	2017-03-28	186.93
1011	leaks and water damage	439	2019-06-20	372.31
2270	Leaks and Water Damage	470	2019-09-05	1644.80
2490	Leaks and Water Damage	472	2017-06-09	1544.25
2490	Leaks and Water Damage	472	2017-02-22	541.70
1778	leaks and water damage	504	2019-05-28	1085.61
1724	leaks and water damage	526	2020-01-12	771.19
2382	Leaks and Water Damage	555	2015-11-24	1104.43

Result 4

- Find all the residents who have a lower-than-average credit score, so office staff and accounting can keep an eye on risky individuals (late payments, etc).

```
#Q5
#find all the residents who have a lower than average credit score,
    #so office staff and accounting can keep an eye on risky individuals (late payments, etc)
select residentID, firstname, lastname, creditscore
from resident
where creditscore < (select avg(CreditScore) from resident);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

Import records from an external file

residentID	firstname	lastname	creditscore
1	Connor	Martin	619
2	Kevin	Sanderson	608
3	Ryan	Manning	502
6	Stewart	Stewart	645
8	Harry	Hughes	376
9	Dylan	Davies	501
11	Neil	Knox	528
12	Sebastian	Burgess	497
13	Charles	Ferguson	476
14	Leonard	Graham	549
15	Dylan	MacDonald	635
16	Piers	Oliver	616
18	Jack	Simpson	549
19	Cameron	Dowd	587
22	Piers	Berry	636
23	Julian	Harris	510
26	Austin	Hughes	577
28	Joseph	Kelly	571
29	Liam	Marshall	574
30	Lucas	Churchill	411
31	Max	McLean	591

resident 5

Isaac Everitt:

- What is the count of each type of role within the company? Our HR team would like to know if we are understaffed for a company of our size. This could dictate if members of that department deserve a raise or if more people need to be hired.

```
9   ## Q2: Which apartment buildings have the highest average square footage?  Return in descending order.
10  SELECT BuildingID, ROUND(avg(SpaceSqFt), 2) AS 'Apt Avg Square Footage'
11  FROM apartment
12  GROUP BY BuildingID
13  ORDER BY avg(SpaceSqFt) DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

role	COUNT(role)
CEO	1
Chief Operating Officer	1
Financial Manager	1
Marketing Manager	1
Operations Manager	1
Office Manager	1
Accounting Manager	1
Executive assistant	4
Human Resource Manger	1
Residential Service Specialist	1
Data Chief Officer	1
Data Analyst	8
Accountant Staff	6
Reciptionist	1
Maitenance Operating Man...	1
Maintenance Technician	5
Utilty Worker	4
Office Staff	79

- Which apartment buildings have the highest average square footage? Return in descending order.

```
10  SELECT BuildingID, ROUND(avg(SpaceSqFt), 2) AS 'Apt Avg Square Footage'
11  FROM apartment
12  GROUP BY BuildingID
13  ORDER BY avg(SpaceSqFt) DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

BuildingID	Apt Avg Square Footage
43	1118.48
5	1114.86
3	1107.81
38	1080.74
40	1078.62
33	1066.30
1	1065.07

- What are the residents' names, their address, and rent amount that have a credit score below 600? These tenants should be watched to ensure they are not late on consecutive months of rent.

```
16 •  SELECT res.FirstName, res.LastName, b.Address, res.CreditScore, rent.Amount
17    FROM resident AS res
18    JOIN rent
19        ON res.ApartmentID = rent.ApartmentID
20    JOIN apartment AS a
21        ON res.ApartmentID = a.ApartmentID
22    JOIN building AS b
23        ON a.BuildingID = b.BuildingID
24    WHERE res.CreditScore < 600
25    ORDER BY res.CreditScore DESC;
26
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

FirstName	LastName	Address	CreditScore	Amount
Owen	Brown	9157 Ridgeview Place	599	1522.38
Dorothy	Springer	50 Cambridge Drive	599	1155.67
Ella	Parr	50 Cambridge Drive	599	3139.82
Jan	Poole	6209 Main Circle	599	3464.5
Alexander	Morrison	79567 Alpine Crossing	599	1011.88
Pippa	Mitchell	99591 Novick Place	599	2372.48
Amelia	Piper	6092 1st Drive	599	1285.46
Oliver	Piper	79567 Alpine Crossing	599	1381.47
Lily	McLean	8 Loeprich Terrace	599	2643.99

- Identify the leases that are 12 months in length, the tenants in those leases, and when the lease ends. The marketing team would like to know these customers so they can target them to resign.

```
28 •  SELECT l.LeaseType, r.FirstName, r.LastName, l.EndDate
29    FROM lease AS l
30    JOIN rent USING (LeaseID)
31    JOIN resident AS r
32        ON rent.ApartmentID = r.ApartmentID
33    WHERE LeaseType = '12 Month';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

LeaseType	FirstName	LastName	EndDate
12 Month	Carl	Terry	2020-11-16
12 Month	Benjamin	Glover	2020-11-16
12 Month	Nathan	Abraham	2020-11-16
12 Month	Ian	Buckland	2020-11-16
12 Month	Charles	Allan	2020-11-16
12 Month	Brandon	Gibson	2020-11-16
12 Month	James	Sharp	2020-11-16
12 Month	Piers	Quinn	2020-11-16
12 Month	Austin	Nash	2020-11-16
12 Month	Thomas	Wilson	2020-11-16

- What building year requires the most maintenance? This could help our maintenance team identify if there was a construction defect in that year of building.

```
35    ## Q5: What building year requires the most maintenance?
36 •  SELECT YEAR(b.DateBuilt), SUM(e.amount)
37    FROM expenditure AS e
38    JOIN maintenance m
39        ON e.MaintenanceID = m.MaintenanceID
40    JOIN apartment AS a
41        ON m.ApartmentID = a.ApartmentID
42    JOIN building AS b
43        ON a.BuildingID = b.BuildingID
44    GROUP BY YEAR(b.DateBuilt)
45    ORDER BY SUM(e.amount) DESC;
46
```

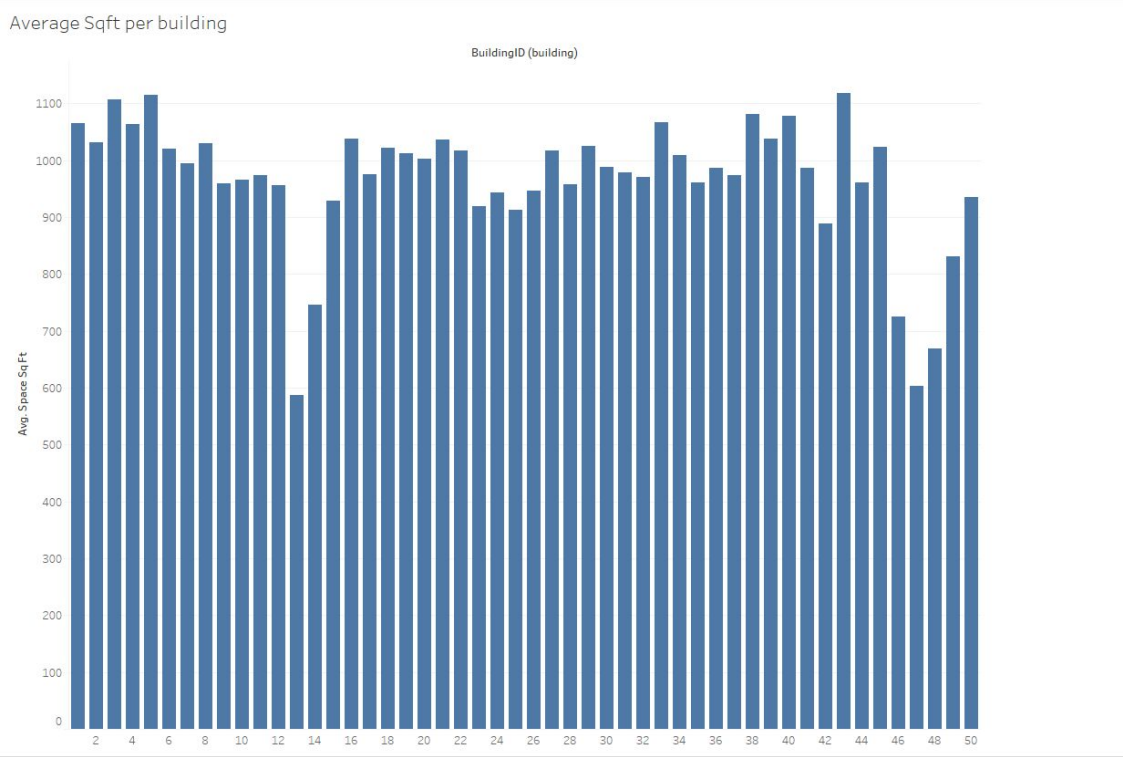
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

YEAR(b.DateBuilt)	SUM(e.amount)
2006	108706.51
1994	92501.54
2009	82697.52
2010	82638.48
2015	81151.09
2001	72842.22
2019	69522.30

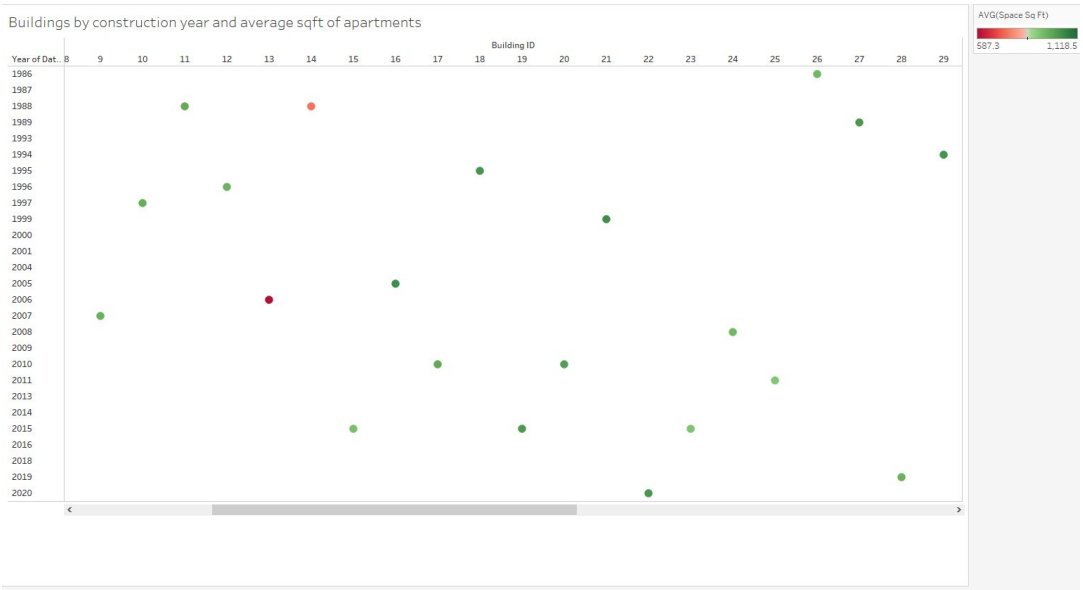
Final Project Visualizations

Jack Beck:

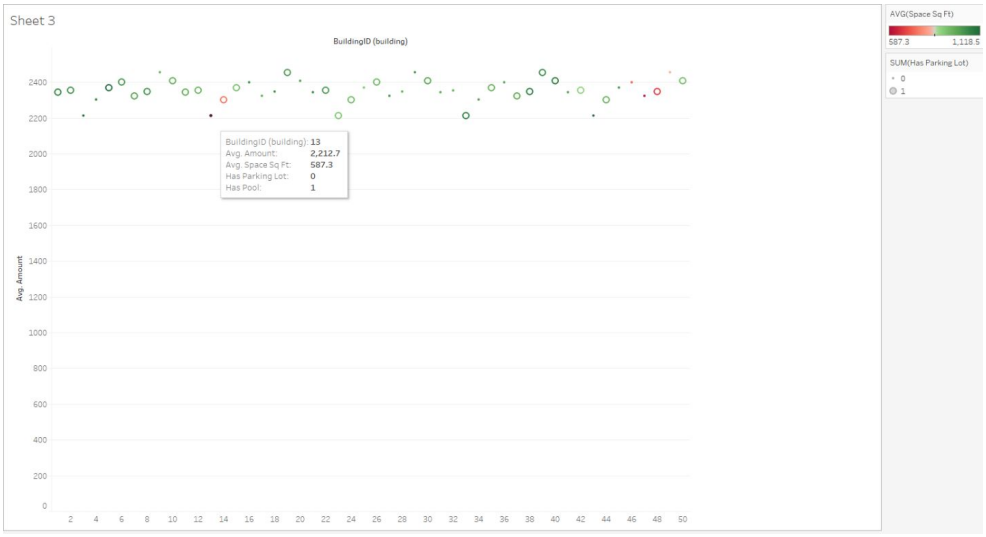
- This first bar graph aims to answer a similar question to that of my first select statement query. This graph shows the average square feet of all units in each of the 50 apartment buildings. By joining the apartment and building tables using the key ‘BuildingID’, this chart provides easy to digest summary information which could be used for advertising purposes.



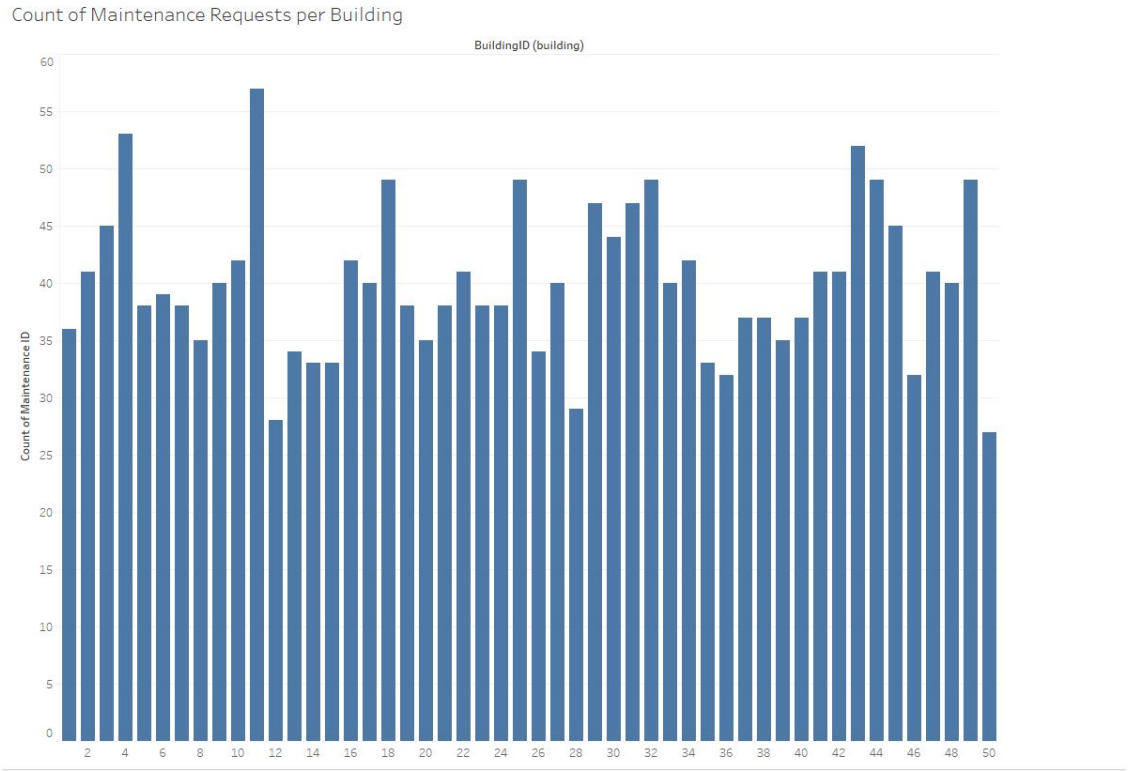
- This second plot attempts answering a similar question to that of my fourth select statement query: “What are the construction years of the buildings with the top 3 highest average square feet per unit?” In this case, instead of restricting it to the top 3 buildings, I choose to show all the buildings, as it is easier to process all this info from a visual perspective. The columns identify the buildings while the rows identify the construction year. The color of each point relates to the average square feet of the building’s units.



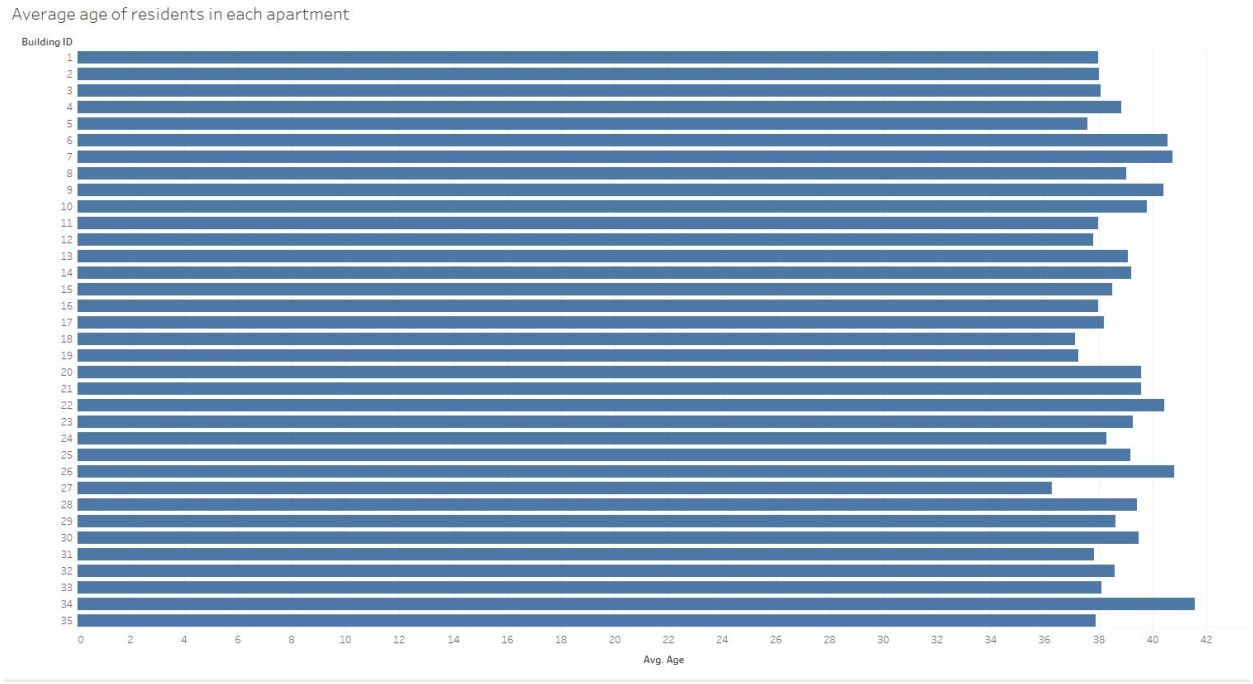
- This third scatter plot illustrates the average rent per apartment unit in each building. The average square feet of the units in each building is again added as the color of each point. This plot also features size as a dimension, which represents whether the building has a private parking lot. The details of each point also indicate whether each building has a pool or not. This graph or something similar could be deployed to the company website for consumer use so they can easily get more information about their choices.



- This bar chart joins the building and maintenance tables to provide a visualization of the amount of maintenance requests per each building. This information could be used to inform future building renovation decisions, as buildings with a high number of maintenance requests will likely need renovation sooner.

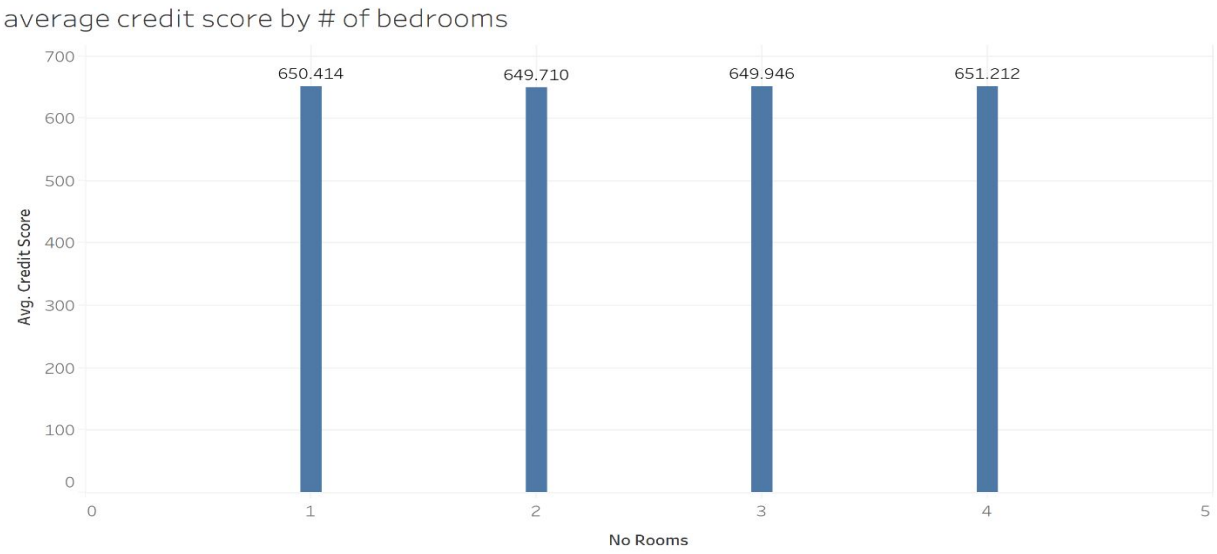


- This final bar chart shows the average age of the residents in each apartment building. This was created by joining the resident table with the apartment table, which was then joined with the building table to identify which building each resident belongs to. This information will help to inform future marketing strategies for the company.

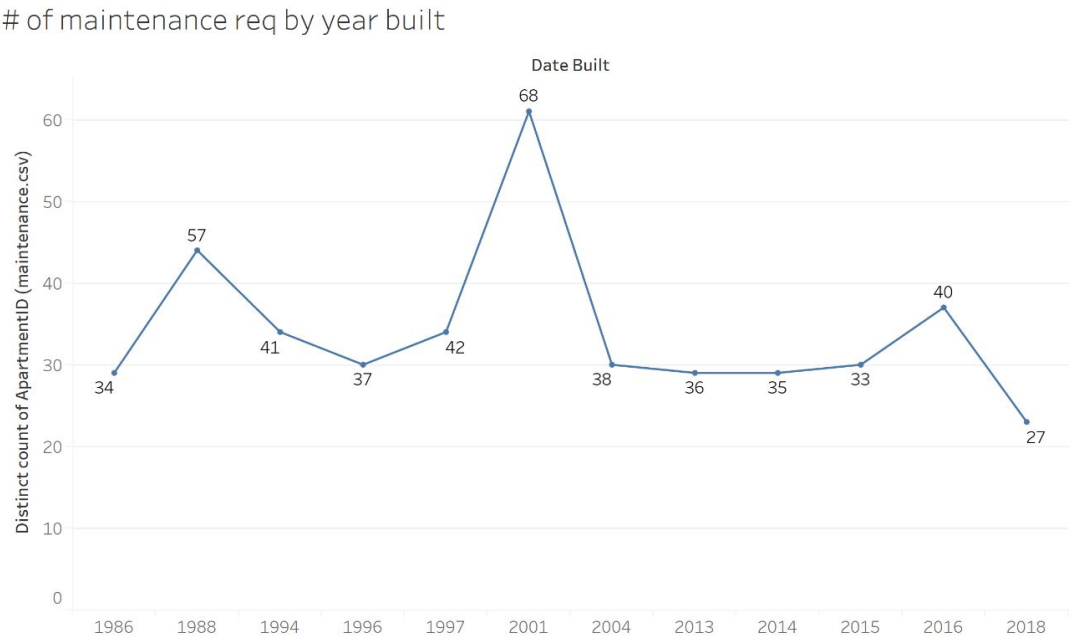


Jordan Waldroop:

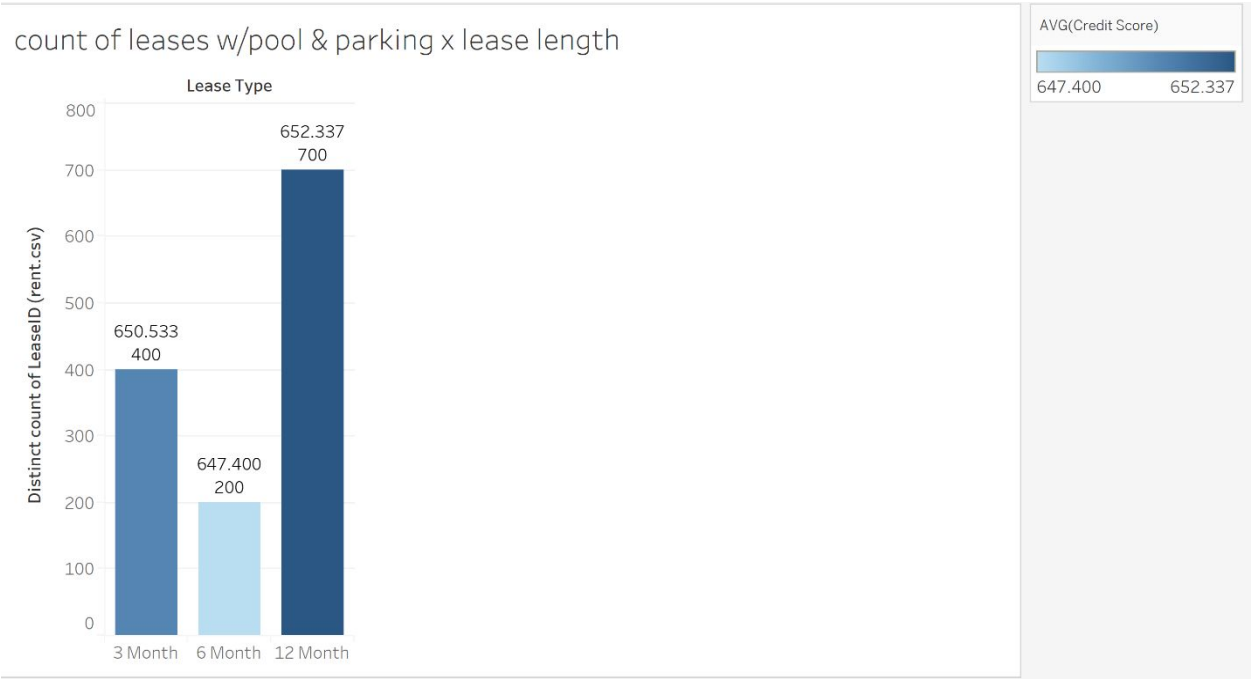
- This bar chart is based on my second select statement. This information is important to the property owners/managers when trying to identify their target resident group and for determining leasing guidelines. It identifies the average credit score of current residents, broken up by the number of bedrooms in their respective apartment.



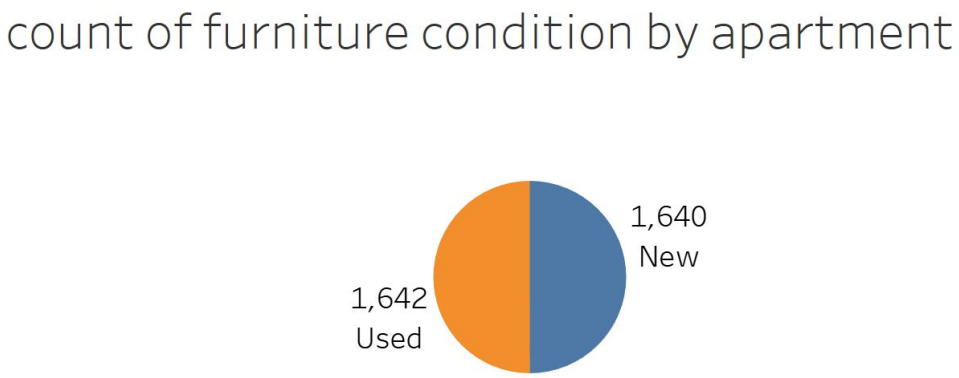
- This line graph identifies the number of maintenance requests separated by the year the apartment building was constructed. This is important because it can help the property owners/managers identify which buildings are requiring the most maintenance work (i.e., buildings constructed in 2001 may want to be investigated further), which may identify further problems with the building structure/integrity.



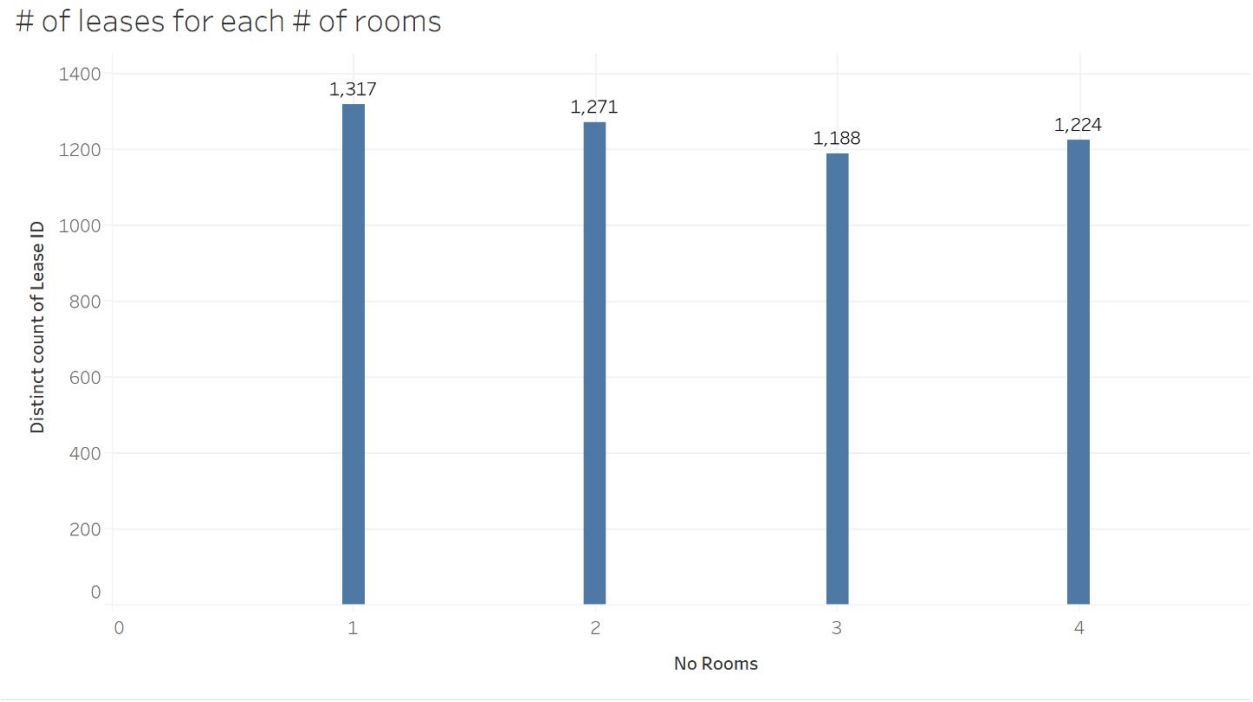
- This bar graph is a count (total number of) leases for apartments with pools and parking lots, grouped by lease length, along with the average credit score of residents in the group. This is important for lease management and identifying target/future residents.



- This pie chart relates to one of my select statements, which aims to get a count of how many apartments have each type of furniture condition. This is important for management so they can be aware of what their furniture allocation looks like.

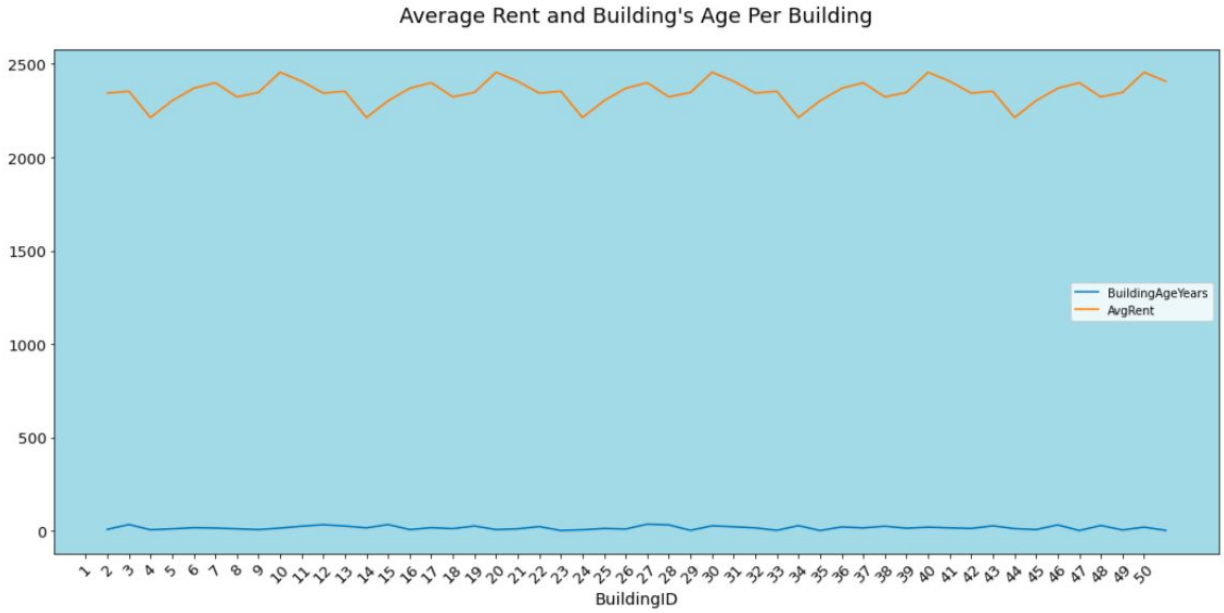


- This bar chart shows a count of the number of leases total for each apartment type. This is important to management because it can help identify if there is a certain apartment style that is in higher demand, especially if they were to be considering additional building(s) to be added to the properties.

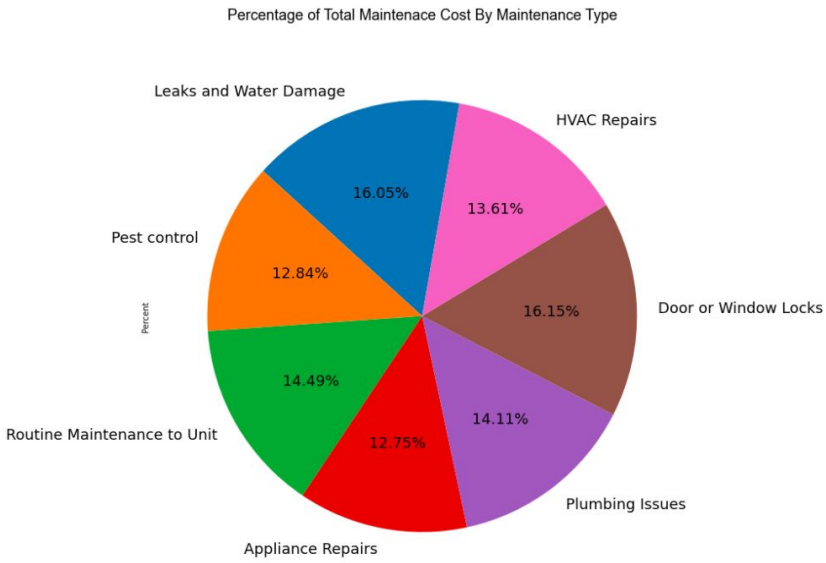


Sam Al Qarzi :

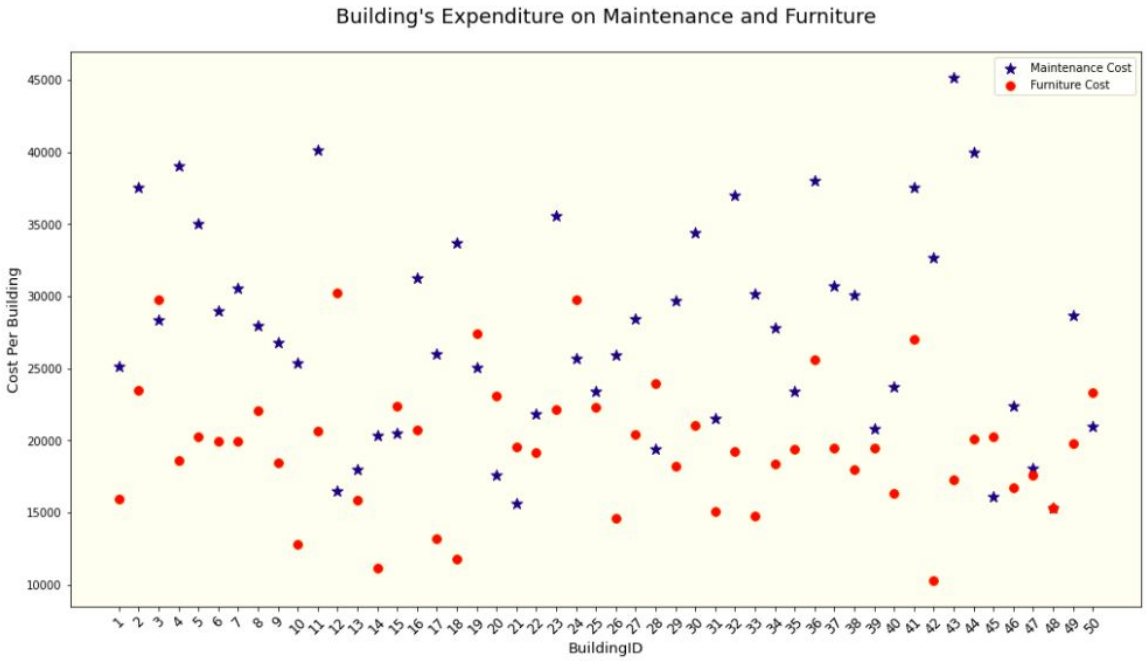
- This is a two-dimensional line chart mapping building's age and the average rent per each building. It was constructed on the result of a SQL Query of one of the select statements. This chart compares the rent average per a building to its age, which will convey any valuable information for the office on how a building's age influences the average rent in that particular building.



- This two- dimensional pie chart shows the percentage of total maintenance cost by each type of maintenance that had been expensed in the expenditure table. To obtain this data, both Maintenance and Expenditure tables had been joined which helped to create a new calculated field for the computed percentage. Such a chart will help the management gain more perspective on which maintenance issues that have most recurring records and their expenses.

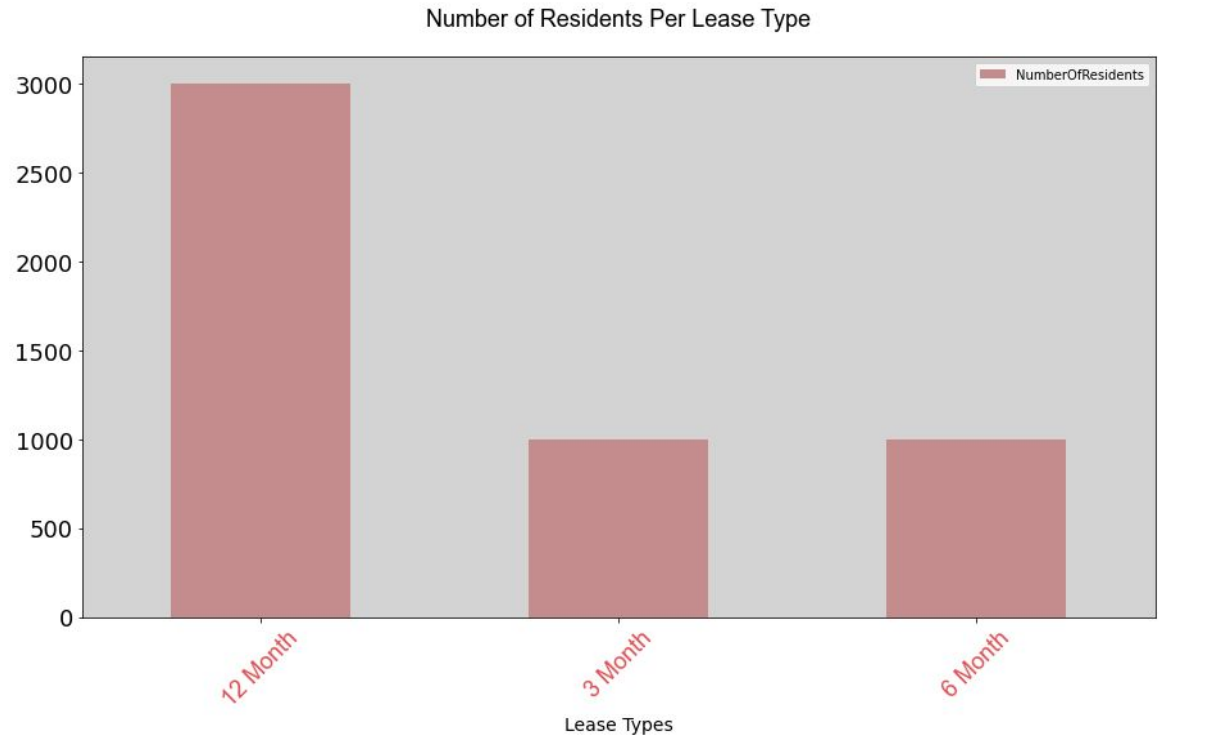


- This scatter-plot chart shows the total expenditure for both the maintenance and furniture for every single building. It is a two-dimensional chart used to compare both expenses for each building.



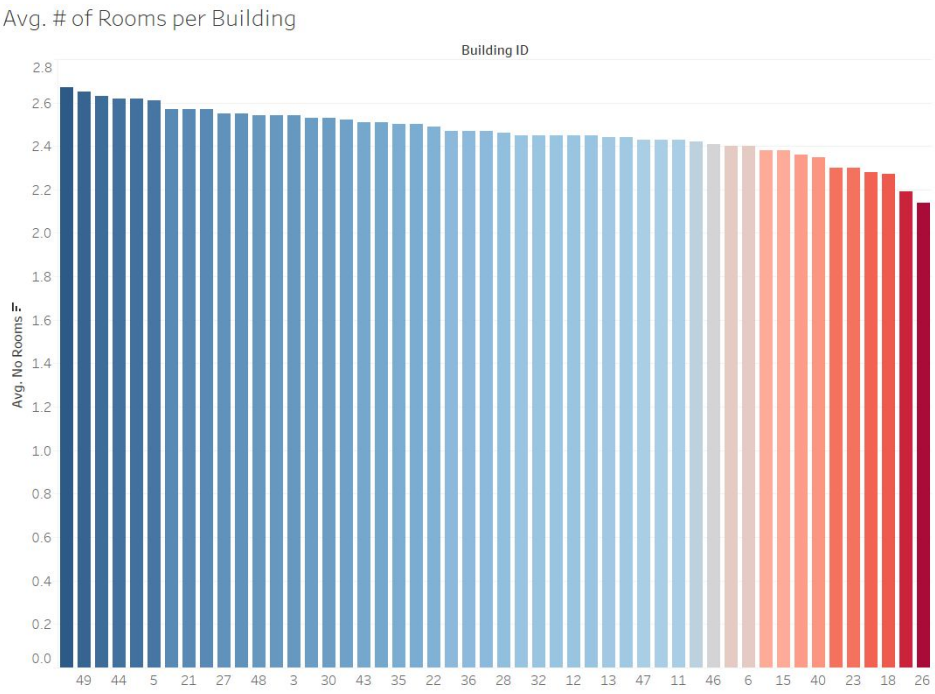
- This is a two-dimensional bar chart presenting the count of tenants by gender per apartment type. It was used to account for apartment types' popularity among residents based on gender.

- This is a two-dimensional bar chart showing tenants' count per lease type, and testing the lease type's preference among residents. It was used to visualize the result of one of the SQL queries.

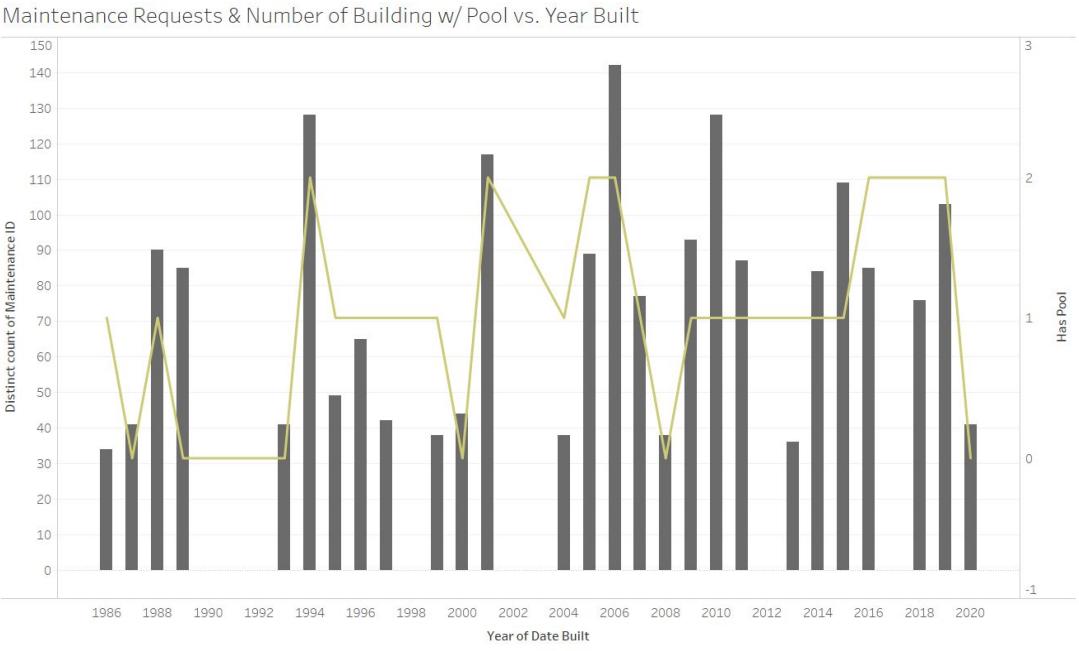


Isaac Everitt:

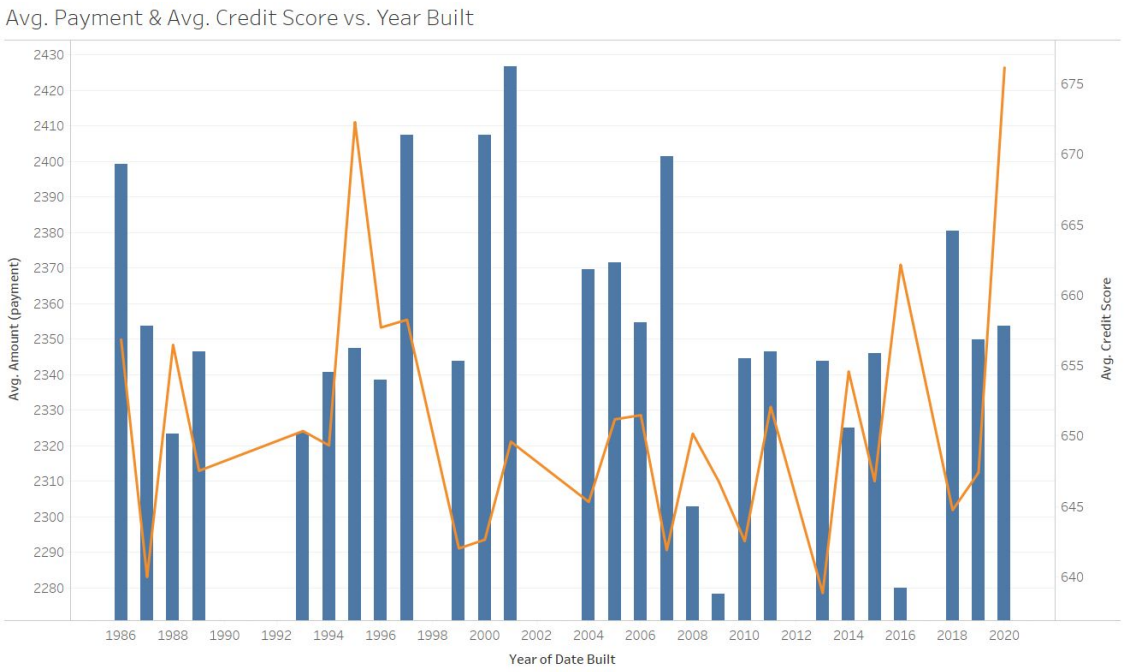
- This mirrors my second select statement where we are determining how many rooms on average each building has so prospective tenants can know how many neighbors they are going to have.



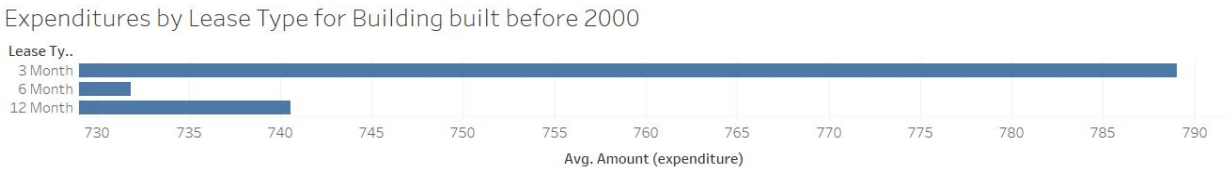
- Here we are looking at the number of maintenance requests and the total number of buildings with a pool grouped by year built to determine if buildings with pools are more likely to have more maintenance requests.



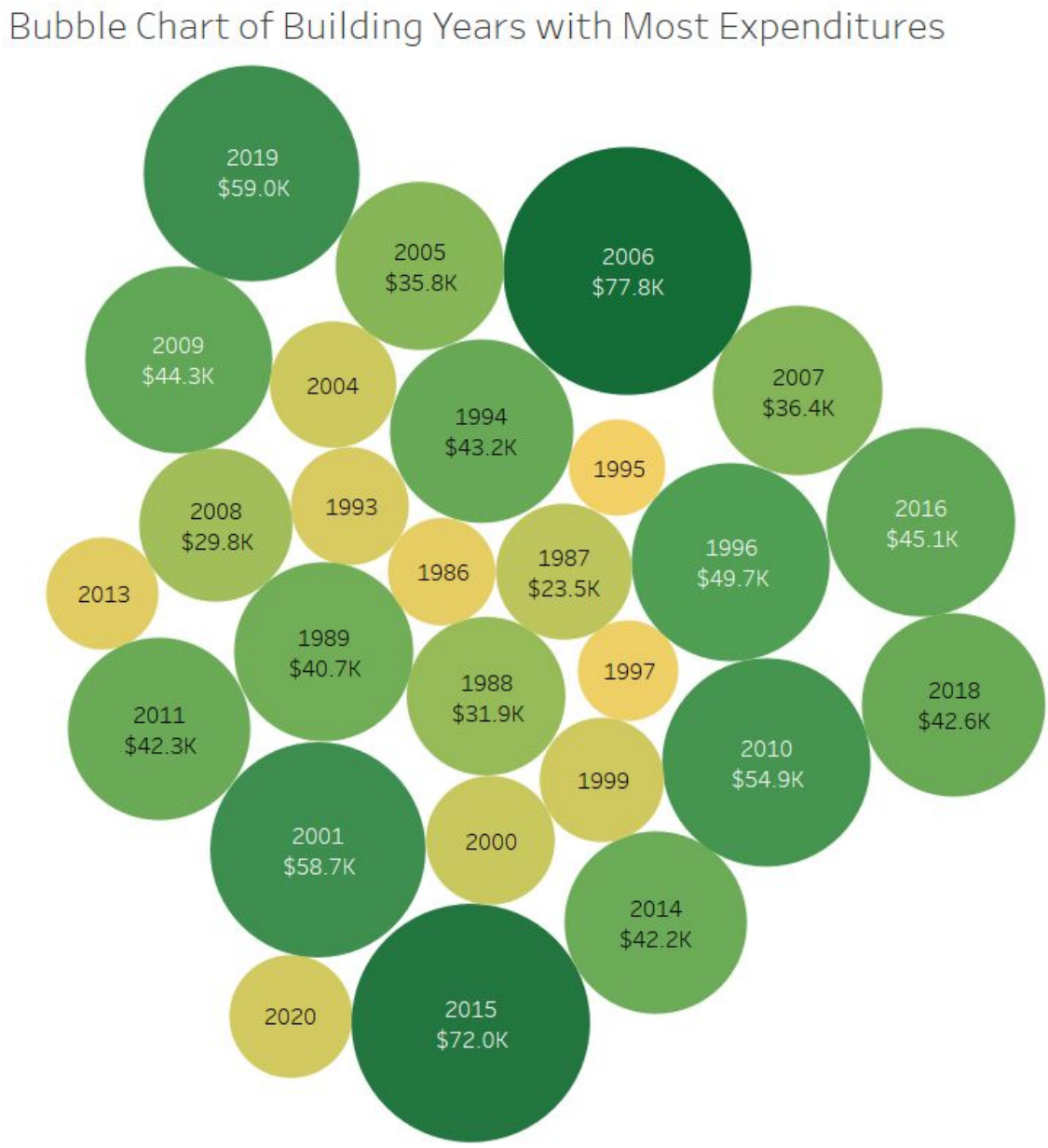
- Here we are examining the average amount paid in rent to us each month with the average credit score to see if tenants with lower credit scores live in older buildings or buildings with lower rent.



- We are examining the average amount of expenditures (in $) that we expect to spend per apartment in buildings built before 2000 grouped on length of lease. This will tell us if we need to increase or decrease our rent prices based on the length of the lease.



- We are examining buildings grouped by year to see what year building has the highest summed expenditures. This way we will know where to allocate future resources.



Cody Rogers:

- Table of Residents with Lower than Average Credit Scores

Resident ID	Credit Sc..	
268	651	Abc
572	651	Abc
683	651	Abc
1094	651	Abc
1182	651	Abc
1570	651	Abc
1811	651	Abc
2124	651	Abc
2301	651	Abc
2340	651	Abc
2352	651	Abc
2536	651	Abc
2917	651	Abc
3166	651	Abc
4137	651	Abc
4227	651	Abc
4322	651	Abc
4386	651	Abc
4552	651	Abc
4742	651	Abc
4806	651	Abc
4924	651	Abc
86	652	Abc
311	652	Abc
650	652	Abc
878	652	Abc
979	652	Abc
1153	652	Abc
1184	652	Abc
1343	652	Abc
1370	652	Abc
1550	652	Abc
1674	652	Abc

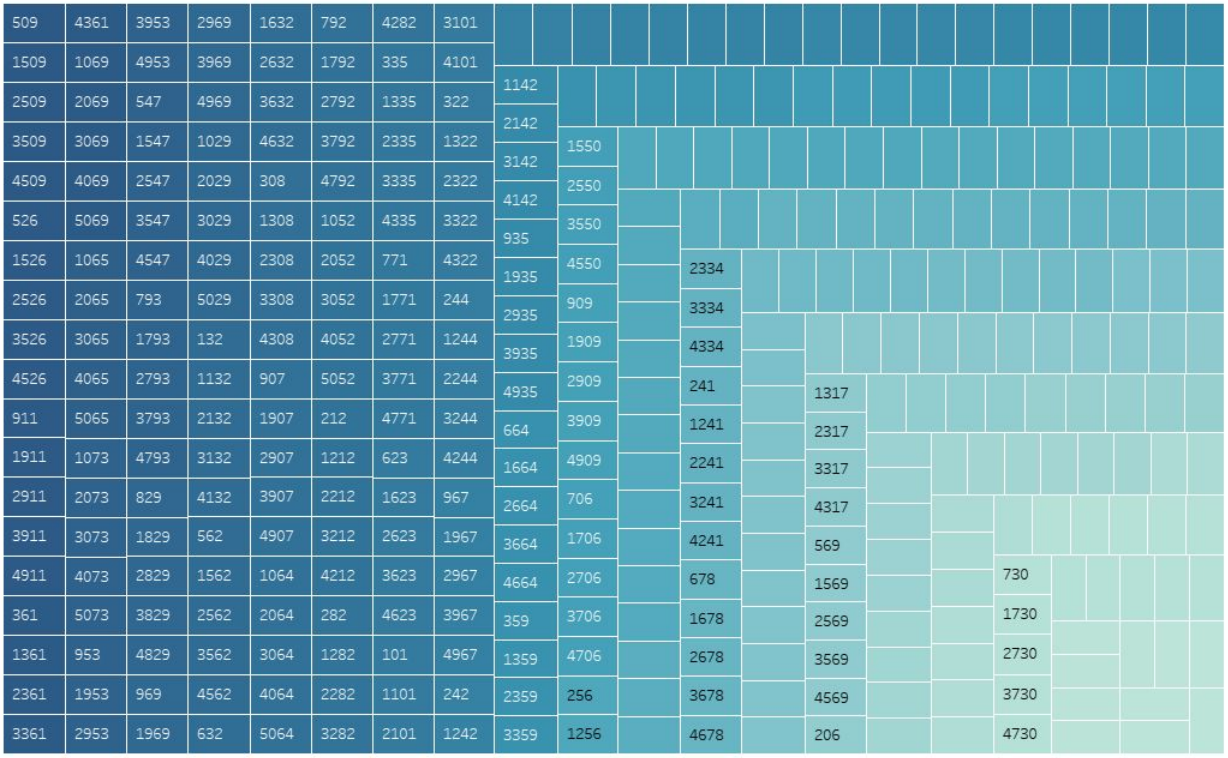
Similar to my fifth select statement, this text table shows a list of every residentID who has a lower than average credit score. This text table was used to create a visual list to identify individuals and connect them to their credit score. The dimensions only include residentID,as an identifier, and credit score, filtered to only show those below the average credit score of all residents (currently sorted lowest to highest by credit score).

- Table of Apartments and Residents with New Lamps

Apartment ..	First Name	Last Name	Email	
143	Simon	Fisher	simon.fisher@team17ren..	Abc
148	Joe	Chapman	joe.chapman@team17ren..	Abc
168	Tim	Davies	tim.davies@team17renta..	Abc
184	Austin	Dickens	austin.dickens@team17r..	Abc
205	Austin	Bell	austin.bell@team17renta..	Abc
286	Stewart	MacDonald	stewart.macdonald@tea..	Abc
432	Richard	Johnston	richard.johnston@team1..	Abc
473	Steven	Anderson	steven.anderson@team1..	Abc
489	Brian	Berry	brian.berry@team17rent..	Abc
534	Leonard	Wilkins	leonard.wilkins@team17..	Abc
567	Kevin	Kelly	kevin.kelly@team17renta..	Abc
578	Jake	Duncan	jake.duncan@team17rent..	Abc
591	Blake	Short	blake.short@team17rent..	Abc
604	Sam	Short	sam.short@team17rental..	Abc
611	Warren	Brown	warren.brown@team17r..	Abc
660	Thomas	Watson	thomas.watson@team17..	Abc
685	Charles	May	charles.may@team17ren..	Abc
726	Sam	Wilson	sam.wilson@team17rent..	Abc
733	Brian	Davies	brian.davies@team17ren..	Abc
763	Christian	Dickens	christian.dickens@team1..	Abc
809	Christian	Brown	christian.brown@team17..	Abc
920	Carl	McDonald	carl.mcdonald@team17re..	Abc
1120	Sam	King	sam.king@team17rental...	Abc
1139	Gordon	Gibson	gordon.gibson@team17r..	Abc
1152	Phil	Scott	phil.scott@team17rental...	Abc
1225	Christian	Young	christian.young@team17..	Abc
1235	David	McGrath	david.mcgrath@team17r..	Abc
1268	Edward	Campbell	edward.campbell@team1..	Abc
1276	Joseph	Lewis	joseph.lewis@team17ren..	Abc
1317	Lucas	Johnston	lucas.johnston@team17r..	Abc
1334	Austin	Reid	austin.reid@team17rent..	Abc
1406	Trevor	Gibson	trevor.gibson@team17re..	Abc
1410	Tim	Glover	tim.glover@team17renta..	Abc

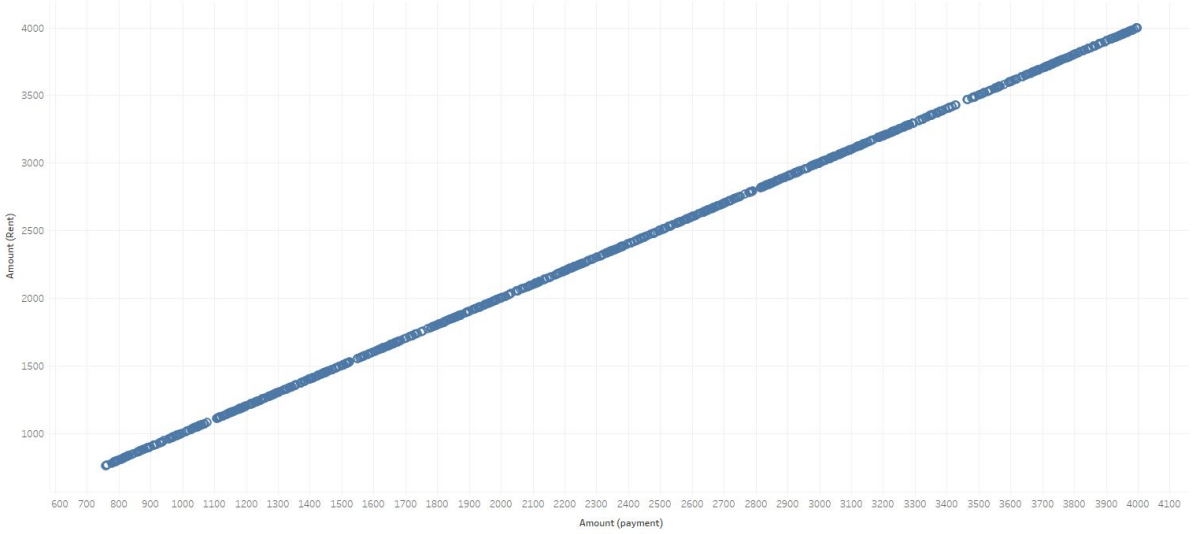
Similar to my first select statement, this text table was used to create a list that shows the apartment number, resident first name, resident last name, and resident email (dimensions) of apartments which have been associated with the "new lamp" recall. The filter for this table is A: to show only furniture that are lamps (no beds, couches, etc), and B: to show only new, not used lamps.

- Apartments Which Gross Over $3,750 in Rent per Month



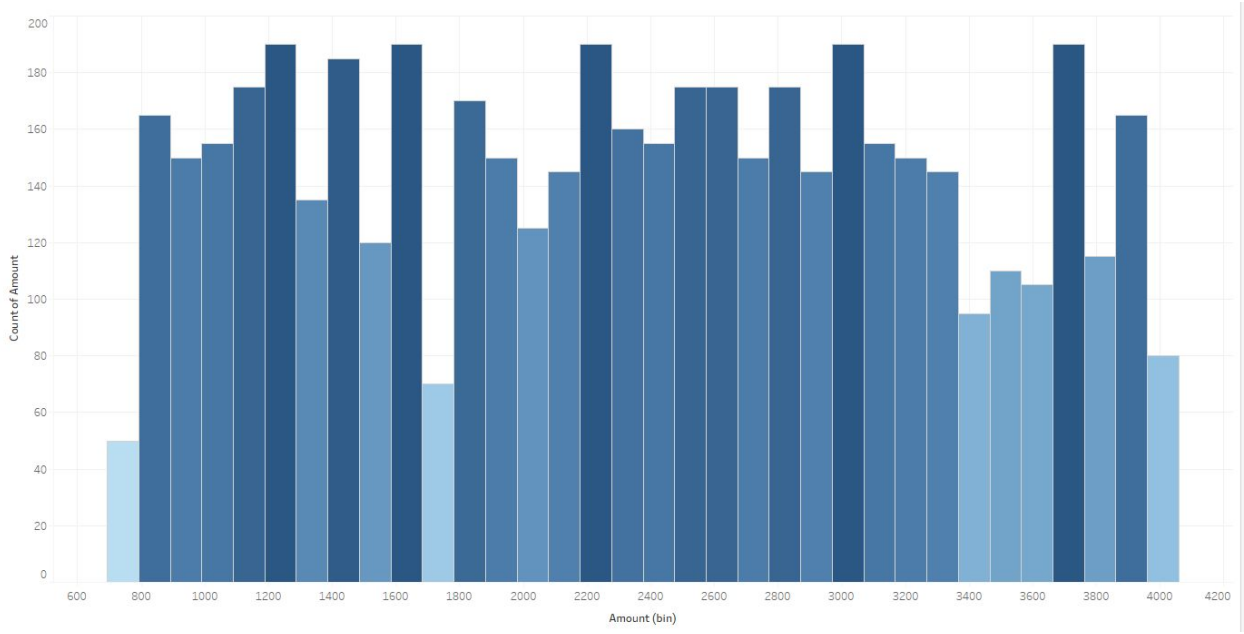
This treemap shows all the apartments which gross over 3,750 in rent each month. This visually shows which apartments gross more by having bigger, darker boxes than the others. This could be used by the business owner to see which of the apartments are more valuable by grossing the most income. The numbers shown in each box reflect the apartment ID (dimension) and the data is filtered by apartments that gross over $3,750 per month (measures).

- Variance in Amount Paid and Amount Due in Rent



This scatterplot shows the variance between the amount of rent charged versus the amount of rent paid. This straight line shows that all payments have been correct and full, with no under or overpayments by residents. The measures are simply the amount charged and the amount paid for rent, by each apartment, which are represented by each mark in the plot.

- How Many Apartments Pay Per Each Bin of Rent



This Bar chart shows how many apartment IDs are charged each tier of rent. Taller, darker bars show more apartments per bin. The dimensions are the amount of rent (binned), and the count of how many apartment IDs are within each bin. This bar chart will show how even (or uneven) the pricing is among all owned apartments. Perhaps there is a segment of budget that could be expanded to in the future.

References

- https://www.mockaroo.com/ used to generate simulated data
- https://www.kaggle.com/shubh0799/churn-modelling?select=Churn_Modelling.csv used for credit score data