# Traffic Management

# Contents

# 1) Data Preview

| | geohash6 | day | timestamp | demand |
|---|---|---|---|---|
| 0 | qp03wc | 18 | 20:0 | 0.020072 |
| 1 | qp03pn | 10 | 14:30 | 0.024721 |
| 2 | qp09sw | 9 | 6:15 | 0.102821 |
| 3 | qp0991 | 32 | 5:0 | 0.088755 |
| 4 | qp090q | 15 | 4:0 | 0.074468 |

geohash6 = (str/object) location

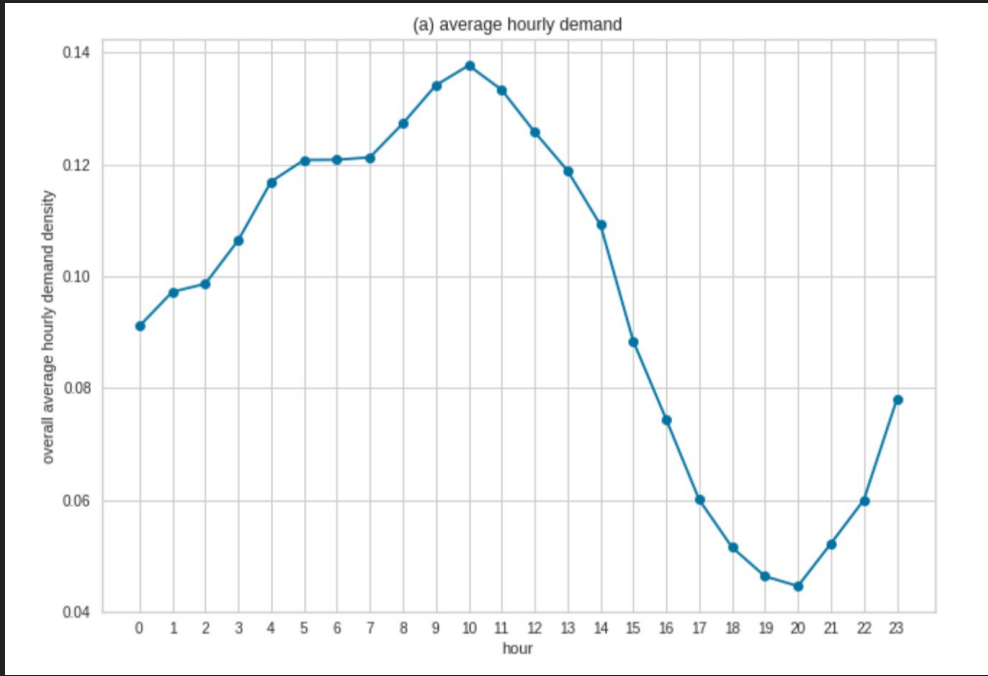
day = (int) day 1 to 60 inclusive

timestamp = (str/object) hh/mm

demand = (float) target variable - 0 - 1 normalised

# 1) Data Visualisation - Average Hourly Demand

by plotting the `average hourly demand` from overall dataset. The `demand` doesn't have the linear correlation to the `hour`. Therefore, we can assume that `hour` is categorical `ordinal` feature

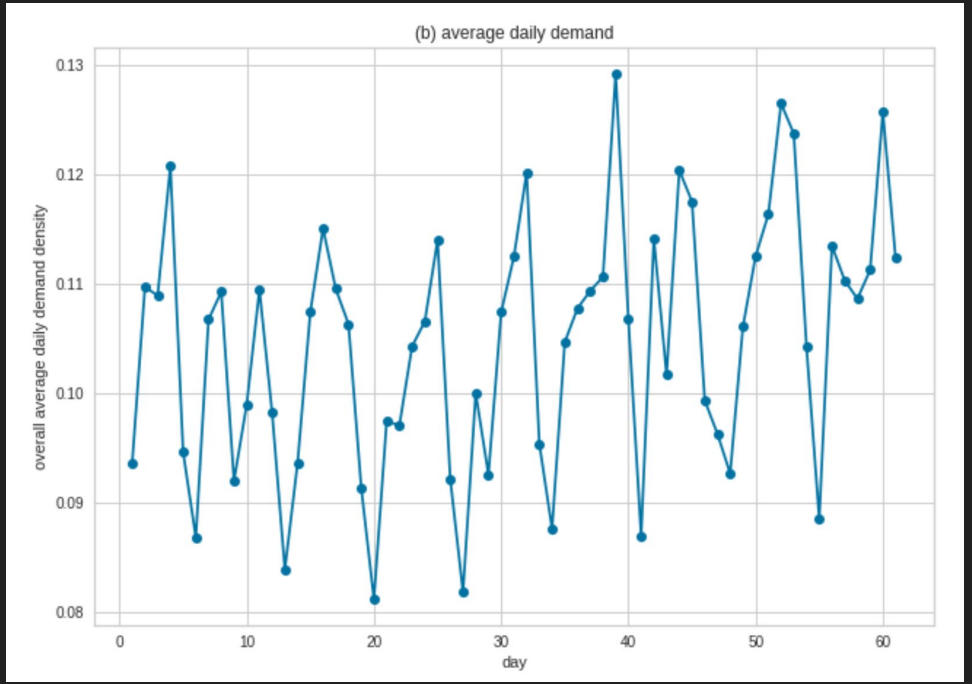


# 1) Data Visualisation - Average Daily Demand

We plot visualisation of average daily demand by grouping through day, and averaging the demand.

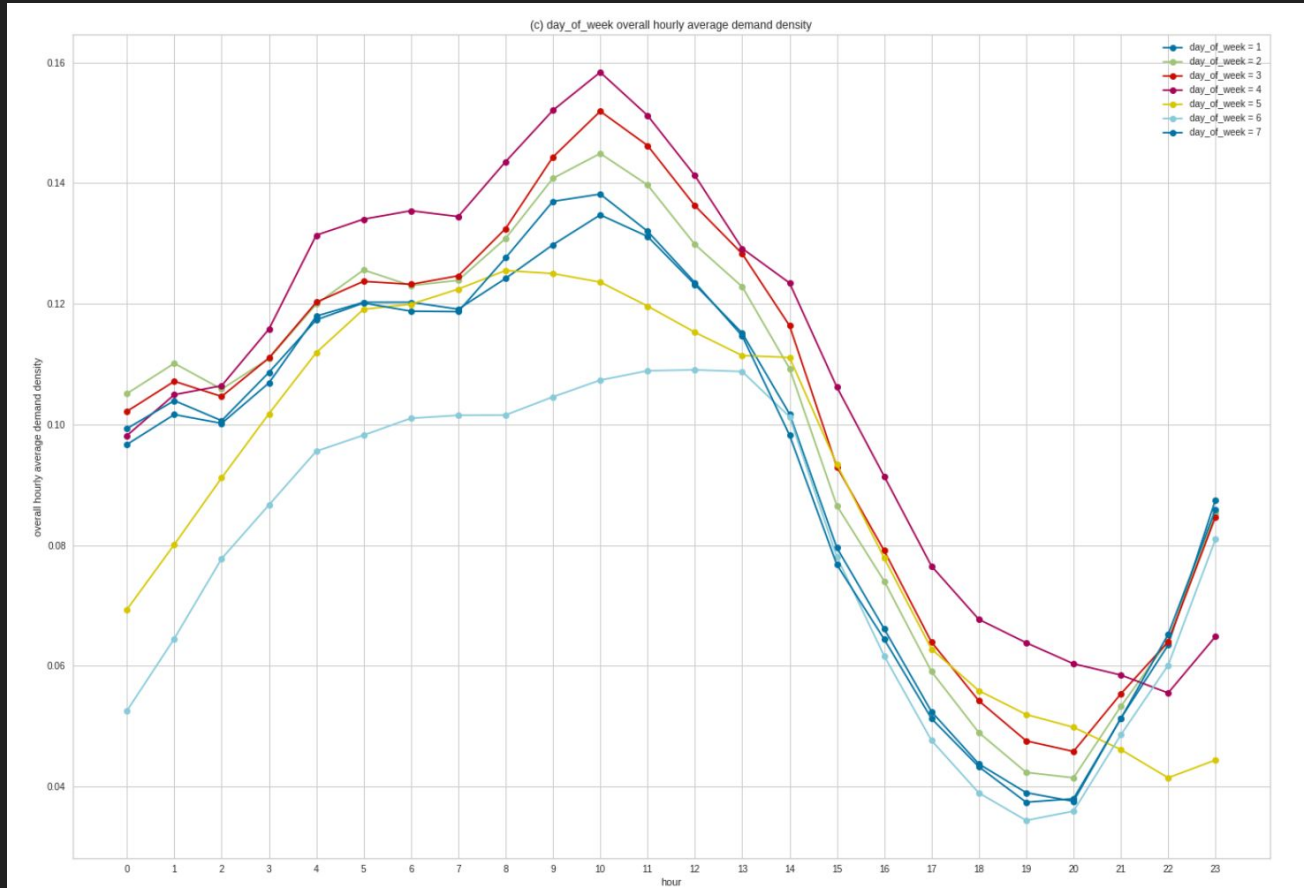
```
14 ax2 = fig.add_axes([1.2,0,1,1])
15 mean_demand_daily = dataset_plotter.groupby("day")["demand"].mean()
16 day = mean_demand_daily.index
17 average_demand_daily = mean_demand_daily.values
18
19 ax2.plot(day, average_demand_daily, "bo-")
20 ax2.set_title("(b) average daily demand")
21 ax2.set_xlabel("day")
22 ax2.set_ylabel("overall average daily demand density")
23 # ax2.set_xticks(day)
```

need to switch the “day” feature into “day_of_week” due to periodicity every 7 day in average - as categorical ordinal feature



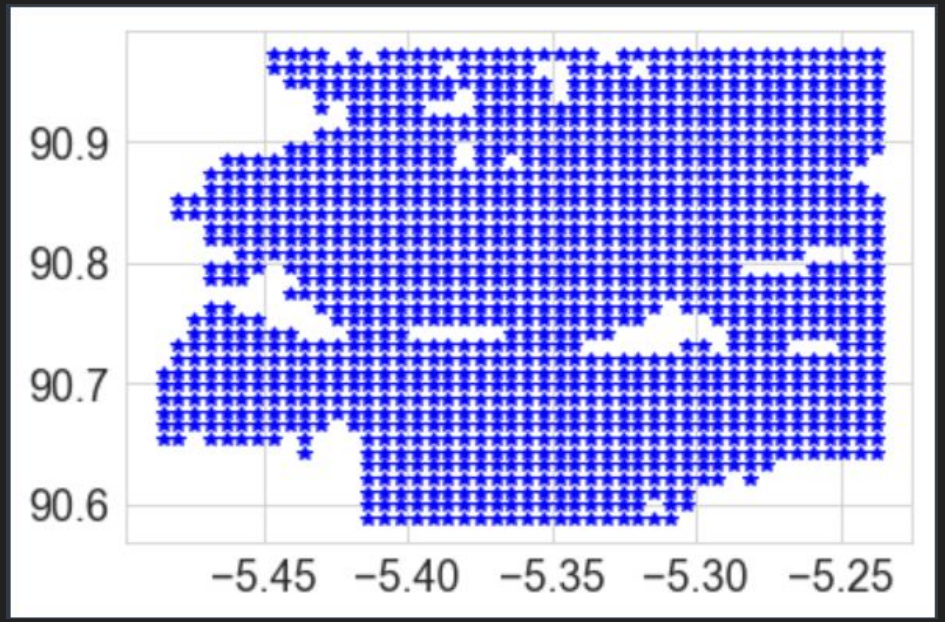
# 1) Data Visualisation - Average Daily Demand grouped through day_of_week

From this figure, we can see that peak hours occur at around 5 am to 2 pm for every day_of_week



# 1) Data Visualisation - Geohash6

The geohash plot (by decoding it and plotting it).



# 2)Preprocessing - Original Features

**Original Dataset Features:**

- **geohash6** = (object / str) location encoded as geohash6
- **day** = (int) day ranges from 1 to 60.
- **timestamp** = (object / str) hh/mm format (hh = hour, mm = minute), hour ranges from 0 to 23 and minute is within [0, 15, 30, 45]

replace **timestamp** into **hour** and **minute** and consider periodicity to convert the **day into 1-7 labels:**

- **timestamp** (object) → **hour**(int) and **minute**(int)

```
# step 1) converting timestamp values with new columns hour and minute in both training

self.training_df["hour"] = self.training_df["timestamp"].apply(lambda x: int(x.split(":")[0]))
self.training_df["minute"] = self.training_df["timestamp"].apply(lambda x: int(x.split(":")[1]))
self.training_df = self.training_df[["geohash6", "day", "hour", "minute", "demand"]]

self.test_df["hour"] = self.test_df["timestamp"].apply(lambda x: int(x.split(":")[0]))
self.test_df["minute"] = self.test_df["timestamp"].apply(lambda x: int(x.split(":")[1]))
self.test_df = self.test_df[["geohash6", "day", "hour", "minute", "demand"]]
```

- **day** (int) → convert this into 1 - 7 period integer into **day_of_week**

```
self.training_df["day_of_week"] = self.training_df["day"].apply(lambda x: 7 if x%7 == 0 else x%7)
self.test_df["day_of_week"] = self.test_df["day"].apply(lambda x: 7 if x%7 == 0 else x%7)
```

We have 4 features: geohash6, day_of_week, hour, minute

# 2) Preprocessing - Filling missing demand (in which = 0)

Using consecutive mergings (SQL like processing)

```
65 # step 2) filling the missing demand data that is actually 0 in value
66
67 self.training_df['key'] = 0
68 self.training_df = \
69 self.training_df[["key","geohash6"]].drop_duplicates().merge(self.training_df[["key","day"]].drop_duplicates(), how = "o
70 merge(self.training_df[["key", "hour"]].drop_duplicates(), how = "outer").\
71 merge(self.training_df[["key", "minute"]].drop_duplicates(), how = "outer").drop(columns = ["key"]).\
72 merge(self.training_df.drop(columns = ["key"]), how = "left").fillna(0)
73
74 self.test_df['key'] = 0
75 self.test_df = \
76 self.test_df[["key","geohash6"]].drop_duplicates().merge(self.test_df[["key","day"]].drop_duplicates(), how = "outer").\
77 merge(self.test_df[["key", "hour"]].drop_duplicates(), how = "outer").\
78 merge(self.test_df[["key", "minute"]].drop_duplicates(), how = "outer").drop(columns = ["key"]).\
79 merge(self.test_df.drop(columns = ["key"]), how = "left").fillna(0)
80
```

# 2) Preprocessing - Coupling Features

Adding more features by coupling existing features. Since all we have are categorical features (for time its ordinal), we will create **coupling categorical-categorical** variables and converting them using **mean encoding**

```
In [74]:  1 X_test.head()
```

Out[74]:

| | geohash6(encoded) | day_of_week(encoded) | hour(encoded) | minute(encoded) | geohash6-day_of_week(encoded) | geohash6-hour(encoded) | geohash6-minute(encoded) | day_of_week-hour(encoded) |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.00342 | 0.062486 | 0.091062 | 0.061077 | 0.004275 | 0.003594 | 0.003277 | 0.089818 |
| 1 | 0.00342 | 0.062486 | 0.091062 | 0.061241 | 0.004275 | 0.003594 | 0.003528 | 0.089818 |
| 2 | 0.00342 | 0.062486 | 0.091062 | 0.061660 | 0.004275 | 0.003594 | 0.003309 | 0.089818 |
| 3 | 0.00342 | 0.062486 | 0.091062 | 0.062211 | 0.004275 | 0.003594 | 0.003566 | 0.089818 |
| 4 | 0.00342 | 0.062486 | 0.087912 | 0.061077 | 0.004275 | 0.005220 | 0.003277 | 0.094323 |

**Best Model (recorded in main.ipynb)**

using features below (including the coupled features)

```
In [572]:  1 features = ['geohash6', 'day_of_week', 'hour', 'minute']
```

```
In [573]:  1 j = 0
           2 for i in range(1, len(features)): # +1
           3     for comb in combinations(features, i):
           4         j += 1
           5         print("feature {} : {}".format(j, "-".join(list(comb)) + "(encoded)"))
```

```
feature 1 : geohash6(encoded)
feature 2 : day_of_week(encoded)
feature 3 : hour(encoded)
feature 4 : minute(encoded)
feature 5 : geohash6-day_of_week(encoded)
feature 6 : geohash6-hour(encoded)
feature 7 : geohash6-minute(encoded)
feature 8 : day_of_week-hour(encoded)
feature 9 : day_of_week-minute(encoded)
feature 10 : hour-minute(encoded)
feature 11 : geohash6-day_of_week-hour(encoded)
feature 12 : geohash6-day_of_week-minute(encoded)
feature 13 : geohash6-hour-minute(encoded)
feature 14 : day_of_week-hour-minute(encoded)
```

We ditched the `clustered_hour` feature due to lower in performance

# 2) Coupling Features Detail

| geohash6 | day_of_week | demand |
|---|---|---|
| a1 | 1 | 0.2 |
| a1 | 1 | 0.8 |
| a2 | 2 | 0.6 |
| a3 | 2 | 0.2 |
| a1 | 2 | 0.3 |

| geohash6 | day_of_week | geohash6 - day_of_week | demand |
|---|---|---|---|
| a1 | 1 | a1 - 1 | 0.2 |
| a1 | 1 | a1 - 1 | 0.8 |
| a2 | 2 | a2 - 2 | 0.6 |
| a3 | 2 | a3 - 2 | 0.2 |
| a1 | 2 | a1 - 2 | 0.3 |

Left: before coupling, Right: after coupling by adding geohash6 - day_of_week
(all features are dummy for visualisation)

# 2) Mean Encoding Detail

| geohash6 | day_of_week | geohash6 - day_of_week | demand |
|---|---|---|---|
| a1 | 1 | a1 - 1 | 0.2 |
| a1 | 1 | a1 - 1 | 0.8 |
| a2 | 2 | a2 - 2 | 0.6 |
| a3 | 2 | a3 - 2 | 0.2 |
| a1 | 2 | a1 - 2 | 0.3 |

| geohash6 | day_of_week | geohash6 - day_of_week | demand |
|---|---|---|---|
| 0.43 | 0.5 | 0.5 | 0.2 |
| 0.43 | 0.5 | 0.5 | 0.8 |
| 0.6 | 0.37 | 0.6 | 0.6 |
| 0.2 | 0.37 | 0.2 | 0.2 |
| 0.43 | 0.37 | 0.3 | 0.3 |

Left: before mean encoding, Right: after mean encoding

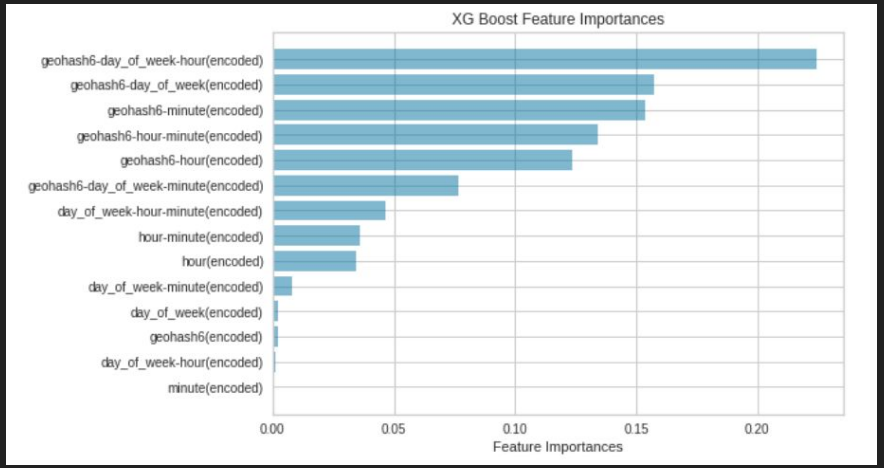
source: https://www.coursera.org/lecture/competitive-data-science/concept-of-mean-encoding-b5Gxv
https://towardsdatascience.com/why-you-should-try-mean-encoding-17057262cd0

# 3) Bayesian Optimisation for Hyperparameter Tuning and Feature Importances

We get XGBoost with parameter

- colsample_bytree= 0.4
- gamma= 0.21756976616440335
- min_child_weight= 10.0
- learning_rate= 0.05358746065589267
- max_depth= 4
- reg_alpha= 0.6870315939145919
- reg_lambda= 0.24937700167665464
- subsample= 0.845934912033431

Feature Importances:




coupled features have higher performance

# 4) Time Series Validation - Sliding Window of Time Series

- Training data : 14 days
- Test data: 5 days

Sliding windows of Time Series

performance metric =
root_mean_square(y_test, y_pred)

averaged over all train-test-ings

# 5) Model Performance - after Averaging all Time Series Windows

average rmse for test datasets = 0.0426

average rmse for train datasets = 0.0302

In [67]:

```
average_rmse_train = np.mean([np.sqrt(ms_error) for ms_error in result_performance["mse_train"]])
average_rmse_test = np.mean([np.sqrt(ms_error) for ms_error in result_performance["mse_test"]])
print("the train dataset's average root mean square for all of the time series train-testing is {}".format(average_rmse_
print("the test dataset's average root mean square for all of the time series train-testing is {}".format(average_rmse_t
```

```
the train dataset's average root mean square for all of the time series train-testing is 0.030236219003556015
the test dataset's average root mean square for all of the time series train-testing is 0.0426028202358991
```

# 6) Conclusion - Attempts with Positive Results

Several Attempts have been done:

1. Feature Engineering - by coupling features, we produce better performance metric (RMSE). We got to know that several `coupled features` have higher correlation in comparison to original features.

   Example: (geohash6 - day_of_week - hour) has the highest feature importances

2. Feature Engineering - `mean encoding` provide faster training process compared to `one hot encoding`. By applying Bayesian Optimisation, we can avoid overfitting.
3. XGBRegressor + Bayesian Optimisation combination has reduced the possibility of overfitting and underfitting. XGBRegressor gives better performance compared to Linear Regression and Lasso Regression combined with Bayesian Optimisation (I tried using these algorithms in the beginning)

# 6) Conclusion - Attempts with Worse Results

1. Adding `clustered hour` as a new feature to define 3 conditions = “high demand hour”, “low demand hour”, and “moderate demand hour” using K-Means Clustering. The recorded rmse score is worse.
2. Filling the possible missing values due to bugs or system failure. These missing can be identified by knowing that there is no demand in all locations at certain “day”, “hour”, and “minute”. By filling it using `nearest neighbors / nearest Minkowski distance`, (Example: Filling the demand at day = 18, hour = 14, and minute = 0 with demand at day = 17, hour = 14, and minute = 0), I got worse rmse scores.
3. Clustering geohash6 to generate new feature. Not good rmse.