STAFF MANAGEMENT TOOL

Exploring an approach towards optimising number of employees while maximizing customer satisfaction and minimizing cost to organization.

TEAM THINKQUE

ABHYUDAY PATIL | ANSH BHATT

PROBLEM STATEMENT

To devise a model which, given the historical volume of Calls and Chats received by the remote IT service desk, predicts the number of employees required to handle the incoming service requests on a given day in the future on an hourly basis.

WHY A PREDICTIVE MODEL?

INCOMING SERVICE REQUESTS

The incoming calls or chats are randomly distributed across each hour. Hence, we can not directly use the call hours to determine the optimal number of employees.

CUSTOMER SATISFACTION

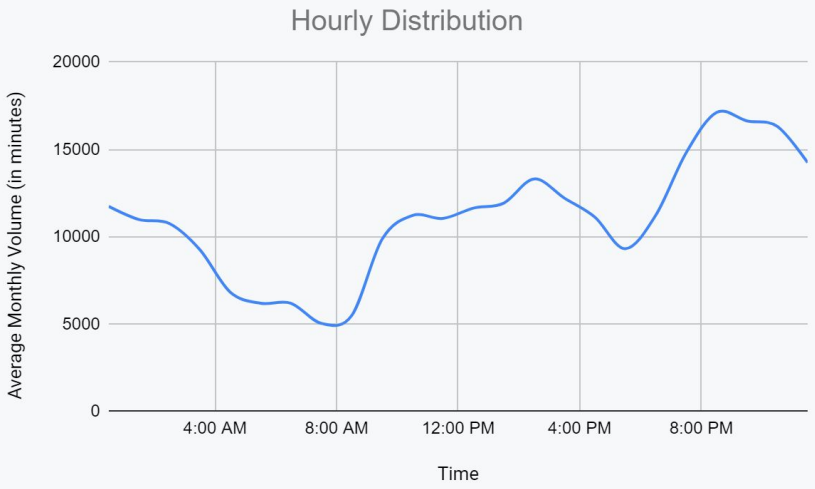
Customer Satisfaction is largely impacted by the wait time in the queue and by the frequency of call drops, which our model aims to minimize.

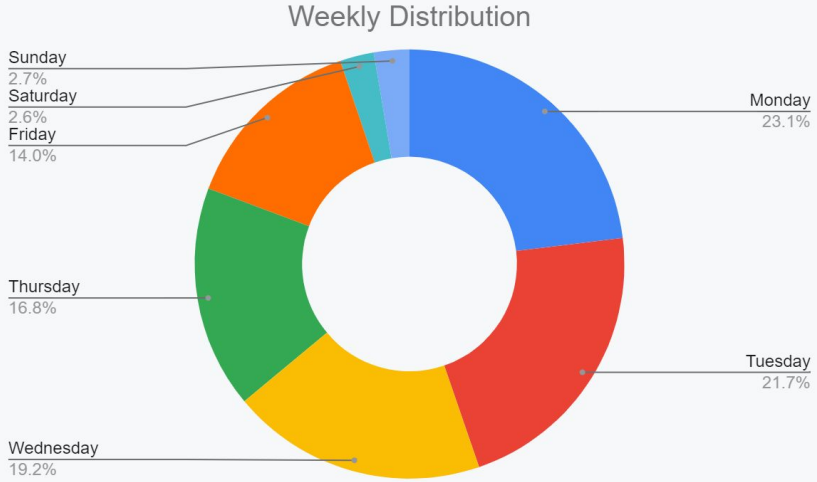
OPTIMISING RESOURCES

Increasing resources will steadily reduce the waiting time of customers. By fixing a target service level and properly planned shifts, we can optimise the resources required.

OUR CUSTOMER ANALYTICS

Visualizing the minutes spent on addressing the incoming service requests on an hourly and weekly basis.





With the hourly distribution, we notice that the service requests peak twice during the day. Once close to 3:00pm and the other close to 8:00 pm. This can give us valuable insight regarding shift rotations.

With the Weekly Distribution, we notice that the service requests peak on Monday and almost steadily decrease through the rest of the week, with weekends seeing very little traffic.

INPUT PARAMETERS

HISTORICAL VOLUME

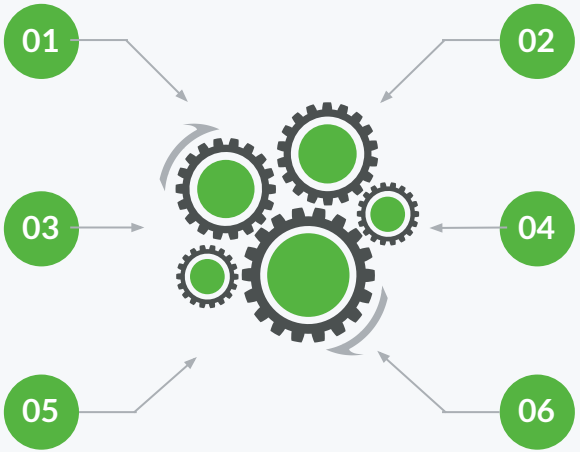
Two months of Historical Service Volume which includes both Calls and Chats.

TARGET ANSWER TIME

The Average Time in which the service desk strives to Connect the Customer to one of the Employees.

MAXIMUM OCCUPANCY

The Maximum Amount of time agents spend on the phone over the course of an hour, expressed as a percentage.



AVERAGE HANDLING TIME

The Average Time that an employee must spend on a Service Request, as obtained from the historical data.

WORKING HOURS

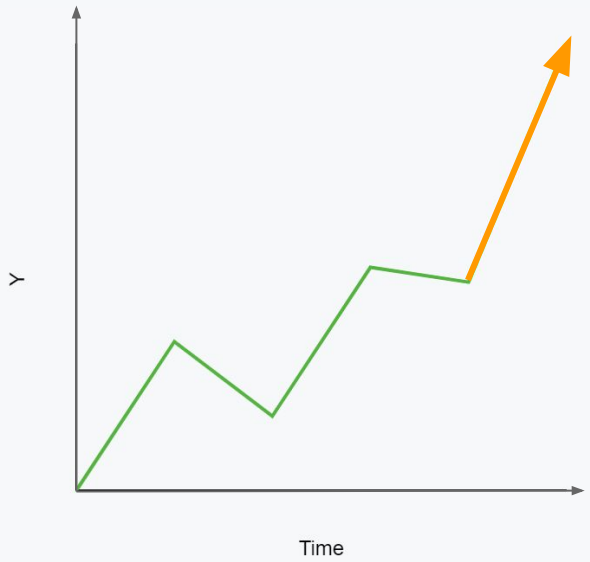
The amount of time that an Employee is expected to be available per day.

SHRINKAGE

The Amount of Paid Time that an employee is not available at work, despite being scheduled.

TWO PHASE PREDICTIVE MODEL

We have devised a two phase model which takes in the previously discussed input parameters and outputs the optimal number of employees for the organization



01

TIME SERIES FORECAST

A Time Series Forecast Model will be fit on Historical Call and Chat Volumes and will be used to predict the Service Volumes for the Future. This Model was preferred over Point Estimation and Averaging Models since it is able to capture seasonalities. This will be done by Preprocessing the raw data, estimating and eliminating Seasonality by Differencing, using seasonal ARIMA, MSE and RMSE will be used to evaluate the Forecast.

QUEUEING SIMULATION

02

In this model, we assume that the same set of employees cater to both the incoming calls and the incoming chats. This ensures optimum utilisation of resources and even distribution of workload among all employees.

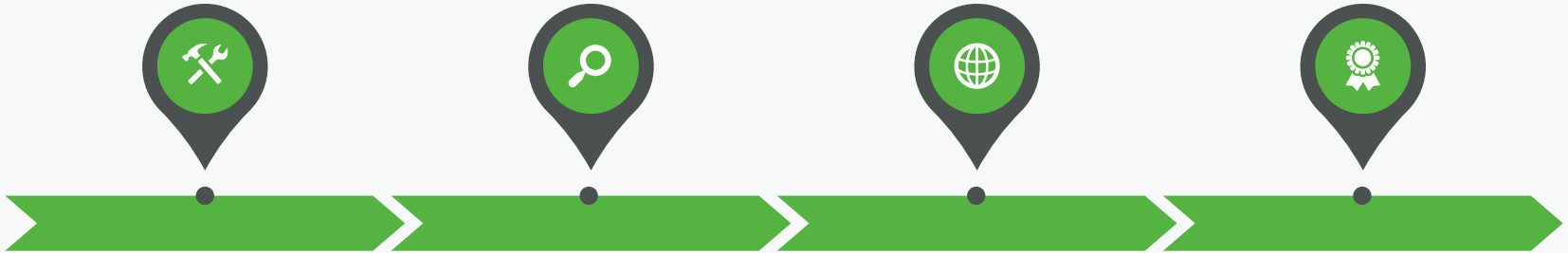
What separates this from regular queueing models is that there are two different queues. Requests in different queues need to be treated differently because the Average Handling Times for both types of requests are significantly different. To achieve this, our model simulates the queues as well as the request handling times. Thus, we can get a close to real life estimate of the waiting time for the customers.

Since we are using a simulation, we can also check for the optimum utilisation of the resources. If a server lies idle during a particular hour, we can remove it.

PHASE I - TIME SERIES FORECAST

MACHINE LEARNING PIPELINE

We look to develop a manual ML Pipeline since our goal is to produce a model to solve our staffing problem



PREPROCESSING

MODEL SELECTION AND TRAINING

MODEL EVALUATION AND VALIDATION

DEPLOYMENT AND MONITORING

PRE-PROCESSING DATA

Step 1 of the Pipeline

- The Preprocessing will include incorporating both the date and the time duration of the service time using the datetime library of Python, concatenating the Call and Chat Request data in a single dataframe and indexing it with the datetime feature. The final dataframe can be visualised as follows:

Date-Time	Call Requests	Chat Requests
2/1/2021 6:30:00	3	12
2/1/2021 7:30:00	5	4
2/1/2021 8:30:00	9	15

- We observe that the data contains double seasonality with an intraday pattern for hourly calls within a day and an inter-day pattern within a week. Therefore, the data has two seasonal cycles: a daily cycle of length 24 and a weekly cycle of length 168 (24 hours x 7 days).
- We define Calendar-related dummy variables for time of day (24 hours) and day of week (7 days) to use for the regression part of the models. This is similar to the One-Hot Encoding method which is widely used while working with pure-regression based models.

PRE-PROCESSING DATA - IMPLEMENTATION

Step 1 of the Pipeline

DateTime	CallVolume	ChatVolume
2021-02-01 06:30:00	3	12
2021-02-01 07:30:00	5	4
2021-02-01 08:30:00	9	15
2021-02-01 09:30:00	17	17
2021-02-01 10:30:00	9	36

This is a snippet from our code which depicts the Pre-Processes Data.

Post Pre-Processing the Data, we perform **ADF** and **KPSS** Tests on the Data to determine Stationarity of the TimeSeries.

ACF TEST

Before Differencing

After Differencing

Call Volumes

```
Results of Dickey-Fuller Test:
Test Statistic                  -4.271113
p-value                          0.000498
#Lags Used                      22.000000
Number of Observations Used    1393.000000
Critical Value (1%)              -3.435053
Critical Value (5%)              -2.863617
Critical Value (10%)             -2.567876
dtype: float64
None
```

```
Results of Dickey-Fuller Test:
Test Statistic                -8.008638e+00
p-value                        2.232205e-12
#Lags Used                     2.400000e+01
Number of Observations Used    1.367000e+03
Critical Value (1%)           -3.435143e+00
Critical Value (5%)           -2.863657e+00
Critical Value (10%)          -2.567897e+00
dtype: float64
None
```

Chat Volumes

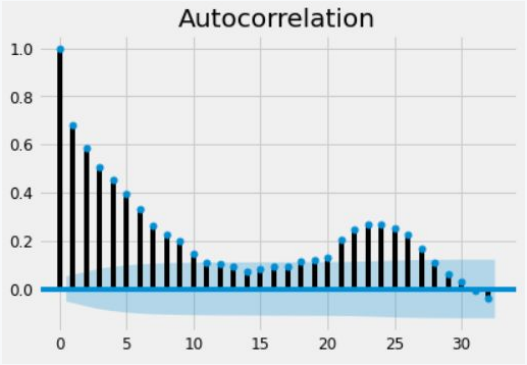
```
Results of Dickey-Fuller Test:
Test Statistic                  -3.425784
p-value                          0.010111
#Lags Used                      22.000000
Number of Observations Used    1393.000000
Critical Value (1%)              -3.435053
Critical Value (5%)              -2.863617
Critical Value (10%)             -2.567876
dtype: float64
None
```

```
Results of Dickey-Fuller Test:
Test Statistic                -6.813952e+00
p-value                        2.082037e-09
#Lags Used                     2.400000e+01
Number of Observations Used    1.367000e+03
Critical Value (1%)           -3.435143e+00
Critical Value (5%)           -2.863657e+00
Critical Value (10%)          -2.567897e+00
dtype: float64
None
```

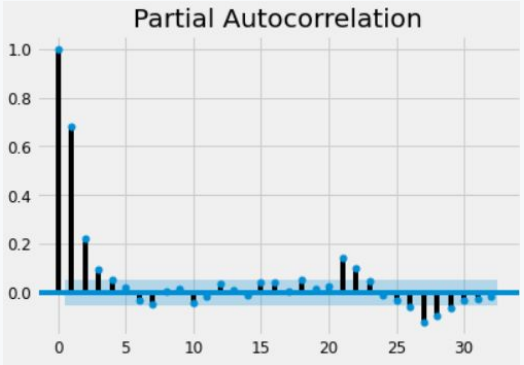
The Drastic Lowering of the p-value after Differencing indicates that it was able to make the Data Stationary.

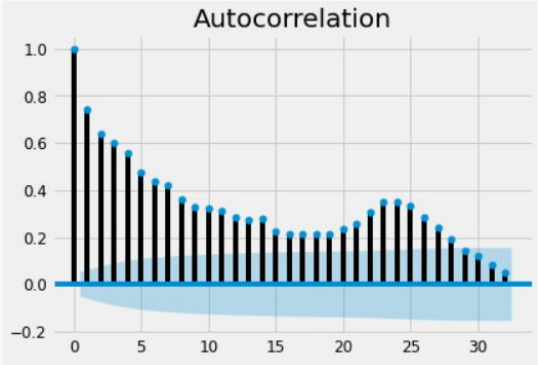
ACF AND PACF PLOTS

To estimate the AR and MA parameters for the SARIMAX Model

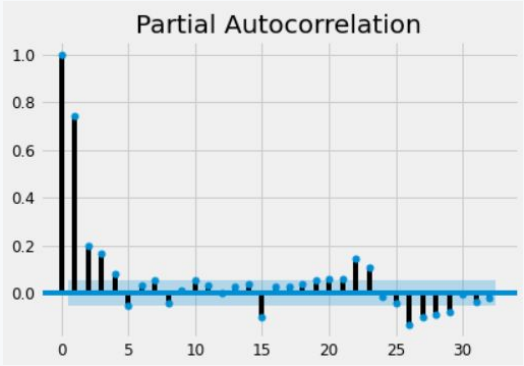


Call Volumes



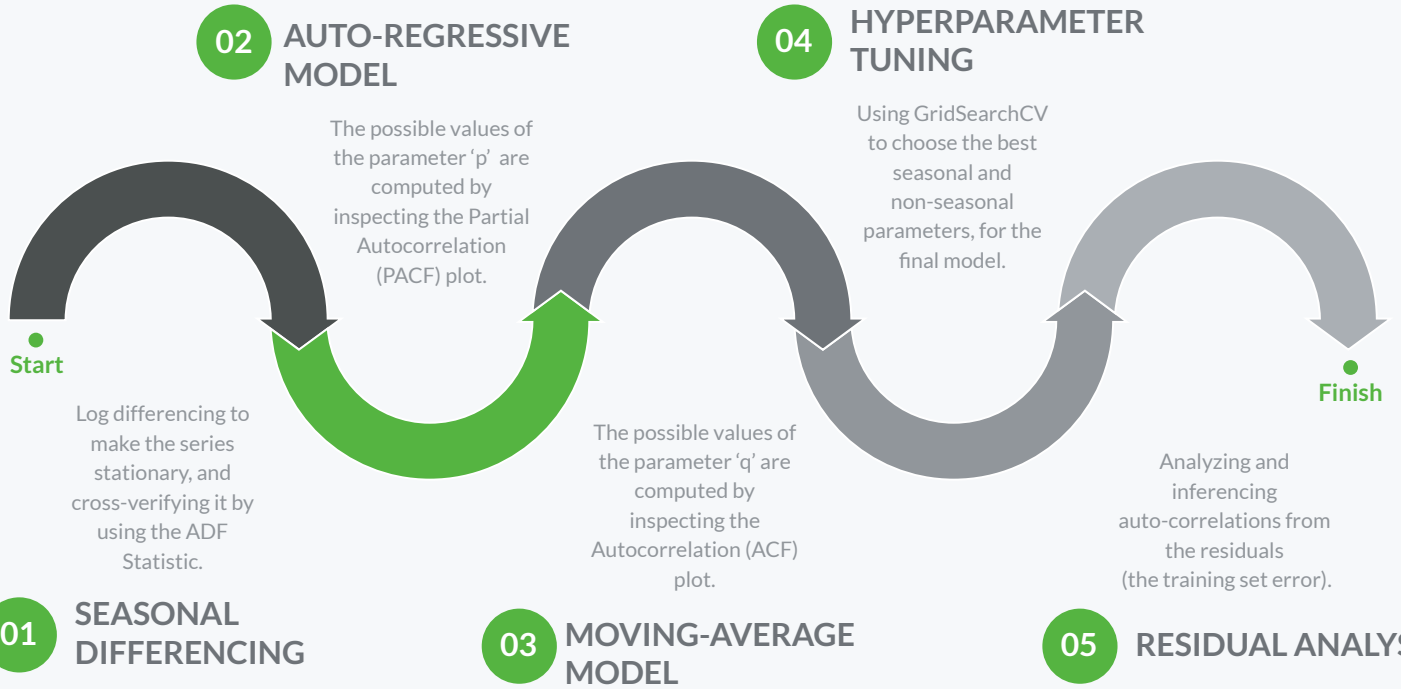


Chat Volumes



MODEL TRAINING

The Training Process of SARIMAX Model



MODEL SELECTION FOR CALLS

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                             AdjVol   No. Observations:                 1392
Model:             SARIMAX(1, 0, 2)x(0, 0, [1], 24)   Log Likelihood               -3505.347
Date:                              Sat, 05 Jun 2021   AIC                           7020.695
Time:                                      19:44:16   BIC                           7046.887
Sample:                                           0   HQIC                          7030.489
                                                - 1392
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9363      0.015     63.782      0.000       0.908       0.965
ma.L1         -0.5446      0.029    -19.048      0.000      -0.601      -0.489
ma.L2         -0.0587      0.027     -2.187      0.029      -0.111      -0.006
ma.S.L24      -0.9795      0.022    -45.489      0.000      -1.022      -0.937
sigma2         8.5352      0.321     26.554      0.000       7.905       9.165
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                97.30
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               1.11   Skew:                             0.54
Prob(H) (two-sided):                  0.27   Kurtosis:                         3.70
===================================================================================
```

MODEL SELECTION FOR CHATS

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                             AdjVol   No. Observations:                 1392
Model:            SARIMAX(1, 0, 2)x(0, 0, [1], 24)   Log Likelihood               -4613.873
Date:                             Sat, 05 Jun 2021   AIC                           9237.746
Time:                                     19:44:22   BIC                           9263.939
Sample:                                          0   HQIC                          9247.540
                                               - 1392
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9558      0.010     96.385      0.000       0.936       0.975
ma.L1         -0.5135      0.019    -26.964      0.000      -0.551      -0.476
ma.L2         -0.0947      0.023     -4.060      0.000      -0.140      -0.049
ma.S.L24      -0.9914      0.059    -16.913      0.000      -1.106      -0.877
sigma2        41.5626      2.140     19.422      0.000      37.368      45.757
==========================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):            116420.72
Prob(Q):                              0.87   Prob(JB):                         0.00
Heteroskedasticity (H):               0.90   Skew:                             3.56
Prob(H) (two-sided):                  0.24   Kurtosis:                        47.23
==========================================================================================
```

MODEL EVALUATION

To determine the degree of fit of the model on the data

DATA	METRIC	VALUE
Call Volumes	MSE	8.87
	RMSE	2.98
Chat Volumes	MSE	21.55
	RMSE	4.64

RMSE will lead to forecasts of the mean, which implies that minimizing RMSE will take output the forecast closer to the mean of the TimeSeries.

PHASE I COMPLETE

The model trained as above shall provide us with the Call and Chat volumes for each hour of each day of the future, which will be utilized in Phase II of our Predictive Model.

The model is re-trainable on new datasets which can be created by appending the new data to the old dataset and by monitoring the results, the hyperparameters of the model can be tuned to accommodate new discernable patterns, if any.

PHASE II - QUEUEING SIMULATION

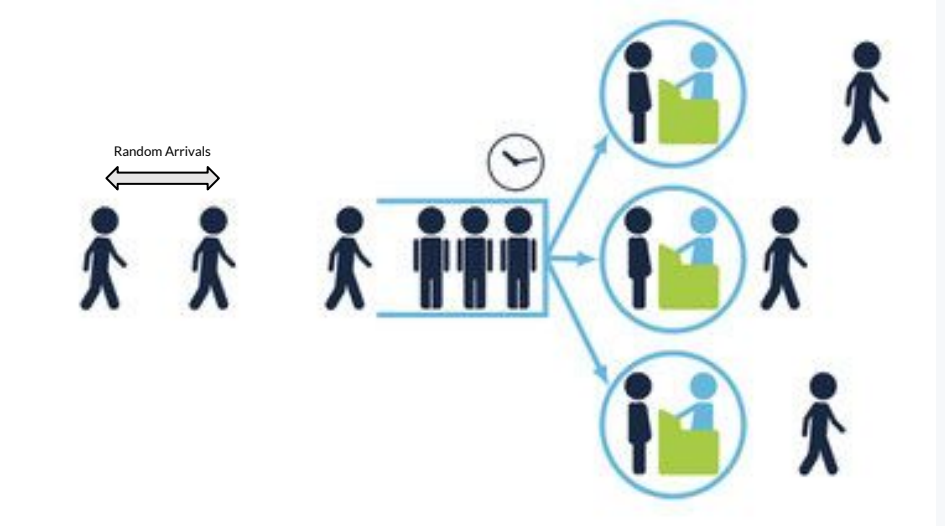
INDEPENDENT ARRIVALS

The receipt of each service request is independent of all the previous requests

Since all the service requests can, in general, be assumed to be independent of each other, we expect that the inter-arrival times between subsequent requests are also all independent. Thus we can model the arrivals as a Counting Process.

The POISSON DISTRIBUTION is the easiest representation of a Counting Process. While there can be other representations, most queueing models utilize the PD for convenience.

Given a Poisson Distribution, the inter-arrival times are Exponentially Distributed. Thus, we can simulate the time points for all incoming requests using the Exponential Distribution. What remains, is to simulate service by the employees and check for proper workload distribution. No employee should be over-burdened.



GENERATING TIMEPOINTS

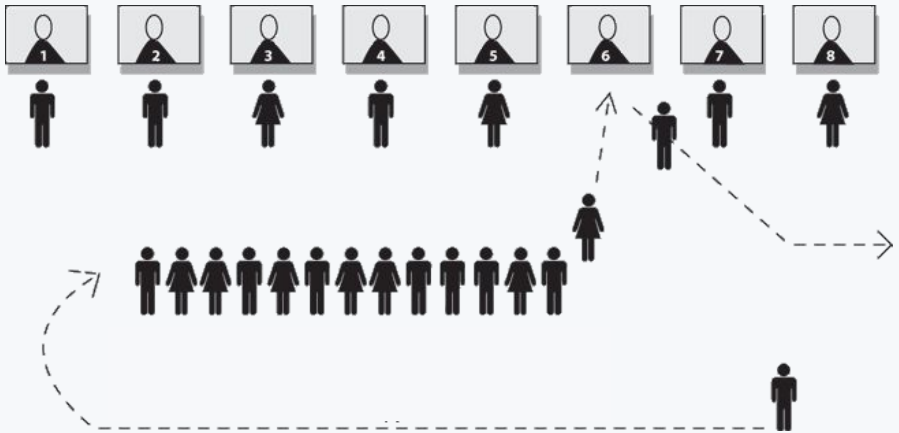
The attached code snippet has been used for generating the time points of the incoming service requests. Here, **volume** denotes the hourly call or chat volume as obtained from the previous phase.

The retExponential() is a function which makes use of the 'random' library of python to generate and return exponential random variables.

We keep a track of the time elapsing and hence generate just enough time points so that the total time does not exceed ONE HOUR. Once edge case which has been taken care of in this code snippet is when volume is zero. In that case, no service requests would be made and hence we just return an empty list.

```
def generateTimePoints(volume):
    l = []
    if volume == 0:
      return l
    t = 0
    while t <= 60:
        t1 = retExponential(volume)
        if t+t1 > 60:
            break
        t += t1
        l.append(t)
    return l
```

SIMULATING SERVICE



01

ASSUMING DISTRIBUTION

While we have the AHT for both calls and chats, the actual time taken over handling a request may vary. We have assumed the Service Times to be Exponentially Distributed.

02

IDENTIFYING SERVICE

Since we are dealing with both calls and chats simultaneously, before simulating the service time, we must identify whether the request is a call or a chat.

03

INCREMENTING AGENTS

We begin each simulation with a single agent attending all requests. We increase the agents till we finally achieve our desired Target Service Level.

04

ENSURING SIMULTANEITY

While one agent serves one request, another agent should be able to serve another request. We ensure this by keeping a track of each employee's status (i.e. 'BUSY' or 'FREE')

ALLOCATING SERVICE

```
def allocateEmployee(employeeClocks, inTime):
    employeeItr = 0
    minClock = employeeClocks[0]
    for i in range(len(employeeClocks)):
        if employeeClocks[i] < minClock:
            minClock = employeeClocks[i]
            employeeItr = i
    wait = max(minClock - inTime, 0)
    return wait, employeeItr
```

```
wait, itr = allocateEmployee(employeeClocks, callList[i])
callWaitTimeList.append(wait)
handlingTime = retExponential(60.0/avgCallHandlingTime)
handlingTimeList.append(handlingTime)
employeeClocks[itr] = callList[i] + wait + handlingTime
```

The most important thing to be ensured while allocating employees to each incoming service request is that simultaneity of action of two different employees should be possible. Two employees should be able to serve a service request each at the same time.

How we have tried to capture that is by assigning to each employee their own clock. While an employee serves a request, the clock ticks. So, at any point of time, the clock shows the time at which an employee would be available to take the next request.

When a new service request is received, we iterate over the clocks of all the employees and select the employee who would be available the soonest (or the employee who has been sitting idle the longest). The corresponding waiting time for the service request and the employee id are returned by this function. The waiting times are stored for future reference when we shall judge our simulation. The employee id is used to update the appropriate clock to the next point in time when the employee shall be free.

The service time required for the request is assumed to be exponentially distributed and is calculated depending on whether it is a call or a chat.

JUDGING SIMULATION

To accept or reject whether a particular number of Employees would be enough

TARGET ANSWER TIME

To always answer all calls or chats under a fixed pre-decided time might require a large number of employees under certain scenarios. Thus, we define the Target Answer Time with respect to a predefined Service Level. The Industry Standard for these is taken to be 20 seconds and 80% respectively. Put together, this means that we strive to answer 80% of all requests within 20 seconds of waiting time.

We can define a different Target for both types of requests. To check whether a certain number of employees meets the set target, we store the waiting times for each call and chat in two separate arrays. From there we can easily check if SL% of the waiting times are less than the TAT.

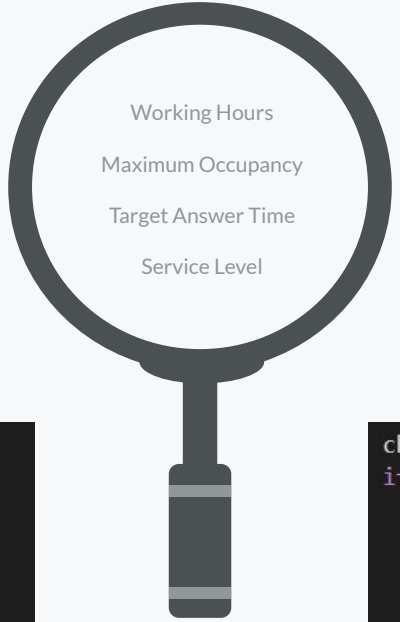
MAXIMUM OCCUPANCY

We might come across scenarios where despite the model meeting the Target Answer Time and the Service Level, the employees may be expected to work over 70 minutes every hour!

This above scenario should obviously not arise. And we must reject that particular simulation if it does. Therefore, we must define what is the maximum amount of time an employee may be expected to work per hour. We can in principle take this to be any amount less than 60 minutes, however, the employees must get rest between calls to perform well on the future Service Requests. As an Industry Standard we can take the Maximum Occupancy to be 80% - 90%.

```
totalHandlingTime = 0
for handlingTime in handlingTimeList:
    totalHandlingTime += handlingTime
avgOccupancy = totalHandlingTime/employees
if avgOccupancy > maxOccupancy*60:
    employees += 1
    continue
```

```
totalWorkingTime = 0
for clockTime in employeeClocks:
    totalWorkingTime += clockTime
NetWorkingTime = totalWorkingTime/employees
if NetWorkingTime > 60:
    employees += 1
    continue
```




```
checkCallServiceLevel = 0
if len(callWaitTimeList):
    for waitTime in callWaitTimeList:
        if waitTime <= targetCallAnsweringTime:
            checkCallServiceLevel += 1
    checkCallServiceLevel /= len(callWaitTimeList)
else:
    checkCallServiceLevel = 1
```

```
checkChatServiceLevel = 0
if len(chatWaitTimeList):
    for waitTime in chatWaitTimeList:
        if waitTime <= targetChatAnsweringTime:
            checkChatServiceLevel += 1
    checkChatServiceLevel /= len(chatWaitTimeList)
else:
    checkChatServiceLevel = 1
```

UNDERSTANDING SHRINKAGE

20% - 35%

of the employees at any point of time are assumed to be not available at work despite being paid to work

We must take into account, that despite employees being paid to work, employees could be absent due to a variety of reasons. Employees could be on paid or sick leaves. They could even be on a vacation.

Shrinkage also takes into account the employees who might have to leave their desks for bio-breaks or for lunch. During this time, they wouldn't be available to attend a Service Request. All of this absenteeism amounts to roughly 20% - 35% of the total work force. It is very important to take this into account. If we do not, then the rest of the employees may be over burdened and the service will deteriorate.

Thus, if our model gives us that the number of employees required to achieve the Service Level is, say, x, we must remember that x should constitute just about 70% of the workforce. The total workforce should be calculated accordingly.

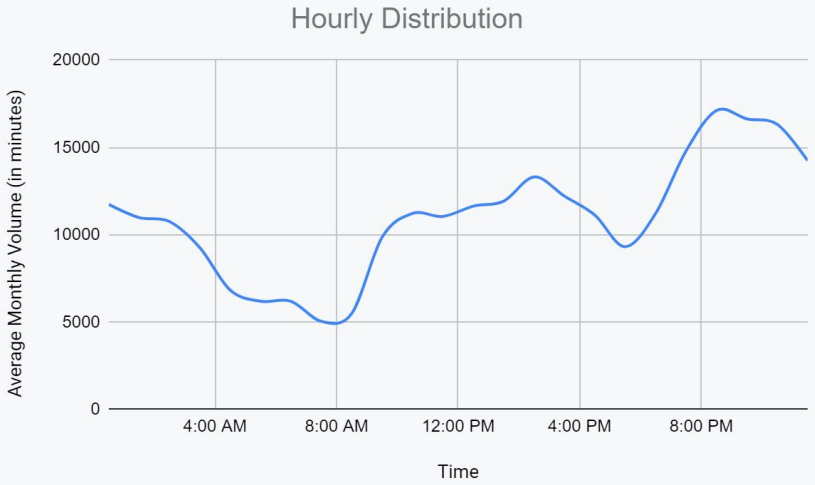
PHASE II COMPLETE

The two-phase model, as described in the previous slides, will provide us with the number of employees required to attend to all the incoming service requests, while maintaining the desired service level. This shall also take into account factors like Shrinkage and Maximum Occupancy.

The model provides us with this answer for each hour of every day of the upcoming month(s). So, what remains is to understand this output and to estimate the employees required for the month.

SHIFTS

DAILY EMPLOYEE REQUIREMENT



Each day is divided into three shifts. The shift timings are given to be:

- 06:00 AM - 03:00 PM
- 03:00 PM - 11:00 PM
- 11:00 PM - 06:00 AM

Since an employee can work only one shift in a day, the daily employee requirement is equal to the sum of the employee requirement in all the three shifts.

The same set of employees work the whole shift. So the employee requirement of the shift can be inferred from the busiest hour in the shift. For this graph, we can see that for the first shift the busiest hour is the final hour. So, the number of employees required from 02:00 PM to 03:00 PM, as obtained by the model, should be taken as the employee requirement of the first shift. The same can be done for each shift.

If we have the employee requirement for each shift, the employee requirement for that day can be obtained by adding the three up.

It should be noted, that this graph is based on historical data. The future data might differ and the analysis for that should be done using the same.

SHIFTWISE EMPLOYEE REQUIREMENT

```
shiftRequirement = pd.DataFrame(columns = ['Date', 'Shift', 'Employees'])
itr = 0
numOfRows = predEmployees['Employees'].size
while itr < numOfRows-1:
    temp = predEmployees.iloc[itr:itr+9, :]
    shiftRequirement = shiftRequirement.append([{
        'Date': temp.iloc[0, 0].date(),
        'Shift': 'Morning',
        'Employees': temp['Employees'].max()
    }])
    itr += 9

    temp = predEmployees.iloc[itr:itr+8, :]
    shiftRequirement = shiftRequirement.append([{
        'Date': temp.iloc[0, 0].date(),
        'Shift': 'Afternoon',
        'Employees': temp['Employees'].max()
    }])
    itr += 8

    temp = predEmployees.iloc[itr:itr+7, :]
    shiftRequirement = shiftRequirement.append([{
        'Date': temp.iloc[0, 0].date(),
        'Shift': 'Night',
        'Employees': temp['Employees'].max()
    }])
    itr += 7
```

For finding the shiftwise employee requirement, we iterate over the hourly requirement of each shift and find the maximum of that particular shift. This maximum is assigned as the employee requirement for the shift.

DAILY EMPLOYEE REQUIREMENT

```
dailyRequirement = pd.DataFrame(columns = ['Date', 'Employees'])
numOfRows = shiftRequirement['Employees'].size
itr = 0
while itr < numOfRows:
    dailyRequirement = dailyRequirement.append([{
        'Date': shiftRequirement.iloc[itr, 0],
        'Employees': shiftRequirement.iloc[itr, 2] + shiftRequirement.iloc[itr+1, 2] + shiftRequirement.iloc[itr+2, 2]
    }])
    itr += 3
```

For finding the employee requirement for a particular day, we simply add the employee requirements of each of the three shifts for that day. This is because no employee shall cover two shifts in a day.

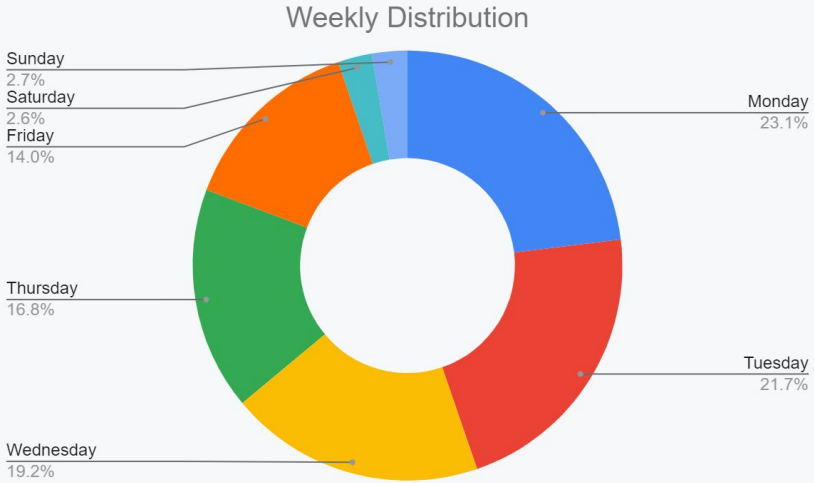
WEEKLY EMPLOYEE REQUIREMENT

While calculating the Weekly Requirement, there are a few things which need to be taken care of. If we are forecasting the Employee Requirement of the month of February, we will have four data points corresponding to the Employee Requirement of each day of the week. While these data points are not expected to vary greatly, for each day of the week the Employee Requirement can be taken to be the Median of these data points. This shall eliminate large deviations from the trend, if any.

Once we have determined the Employee Requirement for each day of the week, the Weekly Employee Requirement can be determined as below.

Since each employee works five days a week, if we take the sum of the Employee Requirements of each day of the week, we should end up having counted each employee five times. Therefore, dividing this sum by five should give us the Weekly Employee Requirement.

$$WR = \frac{\sum_{i=1}^{7} DR_i}{5}$$



MONTHLY EMPLOYEE REQUIREMENT

```
itr_list = [0, 1, 2, 3, 4, 5, 6]
weeklyData = []
numOfRows = dailyRequirement['Employees'].size
for itr in itr_list:
    temp = []
    while itr < numOfRows:
        temp.append(dailyRequirement.iloc[itr, 1])
        itr += 7
    weeklyData.append(statistics.median(temp))
monthlyRequirement = 0
for data in weeklyData:
    monthlyRequirement += data
monthlyRequirement /= 5
print('Number of employees reuired on Monthly/Weekly Basis : ', np.ceil(monthlyRequirement))
```

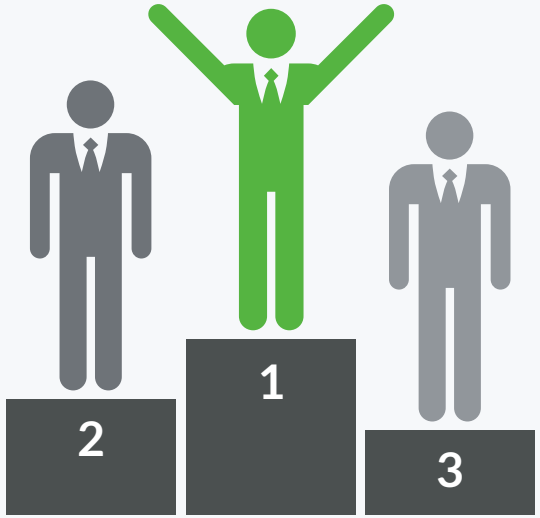
For finding the monthly employee requirement, we isolate the predicted employee requirements for each day of the week (Monday to Sunday) in that month. So for each day of the week, we have multiple data points corresponding to the employee requirement. We associate the median of these data points as the final value for the employee requirement on that day of the week. Further, as described on the previous slide, the weekly requirement is calculated. We take the weekly and the monthly requirement to be equal since each week the shift patterns shall repeat.

CONCLUDING NOTE

We have discussed a predictive model which incorporates Time Series Forecasting and Simulation Techniques to predict the optimum number of Employees required at a remote IT Service Desk to achieve the desired service level.

The model attempts to determine a close to real life estimate of the **WAITING TIME** as well as the **SERVER OCCUPANCY** both of which are very important factors for predicting the number of employees.

- With new data available every passing month, the model can be retrained, and the hyperparameters can be recalibrated for more accurate results.
- This shall ensure that the organisation is able to maximise customer satisfaction while minimising cost to the organisation. Thus, it is a win-win situation for both parties.



“IN GOD WE TRUST,
ALL OTHERS MUST BRING DATA.”

THANK YOU!