Staff Management Tool

Presented by-Dionysus Anguis

HOURLY_BASIS_CALL

Data Pre-Processing in Excel

1. In the excel sheet we made 5 columns named Date, Days, Time, Data, Date_Extracted, Time_Span_Hour.
2. In TIME_SPAN_HOUR we extracted hour from Time.
3. We extracted only the date in the column DATE_EXTRACTED discarding the month and year.
4. Extracted data from Call Volume excel sheet and copied that data to Call_Volume_hourly.xlsx corresponding to respective Date and Time.

importing pandas and numpy

```
import pandas as pd
import numpy as np
```

1. Uploading Excel sheets

```
from google.colab import files
call_data_hourly=files.upload()
```

2. Reading files with Pandas

```
import io
df=pd.read_excel(io.BytesIO(call_data_hourly['Call_Volume_hourly.xlsx']))
df
```

dropping Date and Time column from datasheet

```
df.drop(['Date', 'Time'], axis=1, inplace=True)
df
```

marking weekends as 1 and weekdays as 0

```
df['Weekend'] = np.where(((df['Days'] == 'Saturday') | (df['Days'] == 'Sunday')), 1, 0)
Df
```

dropping Days column from datasheet

```
df.drop(['Days'], axis=1, inplace=True)
df
```

Now we should analyse our dataset.
We will import seaborn library to plot a boxplot which will describe to us about outliers.
Checking for the Outliers is an important step which will help us in deciding best evaluation metrics.
If there are outliers MAE is the best metric to evaluate errors.

import seaborn as sns
sns.boxplot(x = df["Data"])

We will split our dataset in 3 parts-

1. **Train 70%**
2. **Test 15%**
3. **Live/Validation 15%**

We will create a validation set for only large datasets.

from sklearn.model_selection import train_test_split as tts
X_train,X_test,y_train,y_test = tts(df.drop('Data',axis = 1),df['Data'],test_size = 0.3,random_state = 1)
X_test, X_live , y_test , y_live = tts(X_test,y_test,test_size = 0.5,random_state = 1)

Here we are importing Evaluation Metrics from sklearn library.
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

3. Applying RANDOM FOREST REGRESSION on the data

We will use Random Forest Regressor as our model to predict the data(number of calls per hour).
Also, we will measure our performance using R2 score and will measure Error by various methods such as Mean_Squared_Error , Root_Mean_Squared_Error, Mean_Absolute_Error.
Because our data has Outlier,   Mean Absolute Error will be the best metric to evaluate our result.
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators= 200 , max_depth=9 , n_jobs=-1 , random_state= 1)

FIT THE DATA using FIT METHOD
Calculate the R2_score, mse, mae for training data as well as testing data.

```
rfr.fit(X_train , y_train)

import numpy as np
mae_train = mean_absolute_error((y_train) ,(rfr.predict(X_train)))
mae_test = mean_absolute_error((y_test) , (rfr.predict(X_test)))
mse_train = mean_squared_error((y_train) ,(rfr.predict(X_train)))
mse_test = mean_squared_error((y_test) , (rfr.predict(X_test)))
rmse_train = np.sqrt(mean_squared_error((y_train) ,(rfr.predict(X_train))))
rmse_test = np.sqrt(mean_squared_error((y_test) , (rfr.predict(X_test))))
R_score_train = r2_score((y_train) , (rfr.predict(X_train)))
R_score_test = r2_score((y_test) , (rfr.predict(X_test)))

mae_live = mean_absolute_error((y_live) ,(rfr.predict(X_live)))
mse_live = mean_squared_error((y_live) ,(rfr.predict(X_live)))
rmse_live = np.sqrt(mean_squared_error((y_live) ,(rfr.predict(X_live))))
R_score_live = r2_score((y_live) , (rfr.predict(X_live)))

print("ACCURACY : ")
print("Because our dataset has outliers Mean Absolute Error is best method")
print("Mean Absolute Error Training Set")
print(mae_train)
print("Mean Absolute Error Test Set")
print(mae_test)
print("Mean Absolute Error Live Set")
print(mae_live)
print("")
print("Higher the value greater the accuracy")
print("R2 Score Training Set")
print(R_score_train)
print("R2 Score Test Set")
print(R_score_test)
print("R2 Score Live Set")
print(R_score_live)
print("")
print("")
print("Root Mean Squared Error Training Set")
print(rmse_train)
print("Root Mean Squared Error Test Set")
```

```
print(rmse_test)
print("Root Mean Squared Error Live Set")
print(rmse_live)
print("Mean Squared Error Training Set")
print(mse_train)
print("Mean Squared Error Test Set")
print(mse_test)
print("Mean Squared Error Live Set")
print(mse_live)
```

Calculating Number of Engineers Required

```
data = [[15 , 17 , 0] , ]
```

Predicting for Date = 15 , Time = 5:30 PM , Weekday(not Weekend)

```
rfr.predict(data)
```

Further Processing
The model predicts the number of calls.
Average Handling Time of call is given 14 minutes.
Engineers work for 7 hours 30 minutes (= 450 minutes) per day.
So, they will work for 18.75 minutes(= 450 minutes / 24 hours) per hour.
The number of minutes workers are required per hour = the number of call   Average Handling Time of call
Thus, the number of resources required = (The number of minutes workers are required) / (Number of minutes each workers works per hour)

```
number_of_chats=4.2447566

number_of_resourses_required=(number_of_chats) 14/18.75
number_of_resourses_required
```

4 Engineers required on 15th April , 5:30 PM with weekday.

HOURLY_BASIS_CHAT

Data Pre-Processing in Excel

1. In the excel sheet we made 5 columns named DATE, Day, Time, Data, DATE_EXTRACTED, TIME_SPAN_HOUR.
2. In TIME_SPAN_HOUR we extracted hour from Time.
3. We extracted only the date in the column DATE_EXTRACTED discarding the month and year.
4. Extracted data from Chat Volume excel sheet and copied that data to Hourly_dataset_chats.xlsx corresponding to respective Date and Time.

importing pandas and numpy

```
import pandas as pd
import numpy as np
```

1. Uploading Excel sheets

```
from google.colab import files
chat_data_hourly=files.upload()
```

2. Reading files with Pandas

```
import io
df=pd.read_excel(io.BytesIO(chat_data_hourly['Hourly_dataset_chats.xlsx']))
df
```

dropping DATE and Time column from datasheet

```
df.drop(['DATE', 'Time'], axis=1, inplace=True)
df
```

marking weekends as 1 and weekdays as 0

```
df['Weekend'] = np.where(((df['Day'] == 'Saturday') | (df['Day'] == 'Sunday')), 1, 0)
df
```

dropping Day column from datasheet
as it is of no use now

```
df.drop(['Day'], axis=1, inplace=True)
df
```

Now we should analyse our dataset.
We will import seaborn library to plot a boxplot which will describe to us about outliers.
Checking for the Outliers is an important step which will help us in deciding best evaluation metrics.
If there are outliers MAE is the best metric to evaluate errors.

import seaborn as sns
sns.boxplot(x = df["Data"])

We will split our dataset in 3 parts-

1. **Train 70%**
2. **Test 15%**
3. **Live/Validation 15%**

We will create a validation set for only large datasets.

from sklearn.model_selection import train_test_split as tts
X_train,X_test,y_train,y_test = tts(df.drop('Data',axis = 1),df['Data'],test_size = 0.3,random_state = 1)
X_test, X_live , y_test , y_live = tts(X_test,y_test,test_size = 0.5,random_state = 1)

Here we are importing Evaluation Metrics from sklearn library.
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

3. Applying RANDOM FOREST REGRESSION on the data
We will use Random Forest Regressor as our model to predict the data(number of calls per hour).
Also, we will measure our performance using R2 score and will measure Error by various methods such as Mean_Squared_Error , Root_Mean_Squared_Error, Mean_Absolute_Error.
Because our data has Outlier , Mean Absolute Error will be the best metric to evaluate our result.
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators= 200 , max_depth=9 , n_jobs=-1 , random_state= 1)
FIT THE DATA using FIT METHOD
Calculate the R2_score, mse, mae for training data as well as testing data.
rfr.fit(X_train , y_train)
import numpy as np
mae_train = mean_absolute_error((y_train) ,(rfr.predict(X_train)))

```
mae_test = mean_absolute_error((y_test) , (rfr.predict(X_test)))
mse_train = mean_squared_error((y_train) ,(rfr.predict(X_train)))
mse_test = mean_squared_error((y_test) , (rfr.predict(X_test)))
rmse_train = np.sqrt(mean_squared_error((y_train) ,(rfr.predict(X_train))))
rmse_test = np.sqrt(mean_squared_error((y_test) , (rfr.predict(X_test))))
R_score_train = r2_score((y_train) , (rfr.predict(X_train)))
R_score_test = r2_score((y_test) , (rfr.predict(X_test)))
mae_live = mean_absolute_error((y_live) ,(rfr.predict(X_live)))
mse_live = mean_squared_error((y_live) ,(rfr.predict(X_live)))
rmse_live = np.sqrt(mean_squared_error((y_live) ,(rfr.predict(X_live))))
R_score_live = r2_score((y_live) , (rfr.predict(X_live)))
print("ACCURACY : ")
print("Because our dataset has outliers Mean Absolute Error is best method")
print("Mean Absolute Error Training Set")
print(mae_train)
print("Mean Absolute Error Test Set")
print(mae_test)
print("Mean Absolute Error Live Set")
print(mae_live)
print("")
print("Higher the value greater the accuracy")
print("R2 Score Training Set")
print(R_score_train)
print("R2 Score Test Set")
print(R_score_test)
print("R2 Score Live Set")
print(R_score_live)
print("")
print("")
print("")
print("Root Mean Squared Error Training Set")
print(rmse_train)
print("Root Mean Squared Error Test Set")
print(rmse_test)
print("Root Mean Squared Error Live Set")
print(rmse_live)
print("Mean Squared Error Training Set")
print(mse_train)
print("Mean Squared Error Test Set")
```

print(mse_test)
print("Mean Squared Error Live Set")
print(mse_live)

Calculating Number of Engineers required
data = [[15 , 17 , 0]]

Predicting for   Date = 15 , Time = 5:30 PM , WeekDay (no weekend)
rfr.predict(data)

Further Processing
The model predicts the number of chats.
Average Handling Time of chat is given 23 minutes.
Engineers works for 7 hours 30 minutes (= 450 minutes) per day.
Engineers works for 7 hours 30 minutes (= 450 minutes) per day.
So, they will work for 18.75 minutes(= 450 minutes / 24 hours) per hour.
The number of minutes workers are required per hour = the number of chat   Average Handling Time of chat
Thus, the number of resources required = (The number of minutes workers are required) / (Number of minutes each workers works per hour)
number_of_chats=16.11744694
number_of_resourses_required=(number_of_chats) 23/18.75
number_of_resourses_required

20 Engineers required on 15th April , 5:30 PM with weekday.

DAILY_BASIS_CALL

Data Pre-Processing in Excel
Extracted DAY, DATE and DAILY_TOTAL from original dataset.

importing pandas and numpy
import pandas as pd
import numpy as np

1. Uploading Excel sheets
from google.colab import files
uploaded1=files.upload()
uploaded2 = files.upload()

2. Reading files with Pandas
import io
df1=pd.read_excel(io.BytesIO(uploaded1['Call_Volume_Feb_cleaned.xlsx']))
df2=pd.read_excel(io.BytesIO(uploaded2['Call_Volume_March_cleaned.xlsx']))
df1

df2

extracting dates from index by adding 1 because data provided is monthly
df1['DATE_EXTRACTED'] = df1.index + 1
df2['DATE_EXTRACTED'] = df2.index + 1
df1

df2

Making Weekend Column
marking weekends as 1 and weekdays as 0
df1['Weekend'] = np.where(((df1['DAY'] == 'Saturday') | (df1['DAY'] == 'Sunday')), 1, 0)
df2['Weekend'] = np.where(((df2['DAY'] == 'Saturday') | (df2['DAY'] == 'Sunday')), 1, 0)
df1

df2

Dropping insignificant column
dropping DATE and DAY column from datasheet
df1.drop(['DATE' , 'DAY'], axis=1, inplace=True)
df2.drop(['DATE' , 'DAY'], axis=1, inplace=True)
df1

df2

concatenating two datasets together and resetting index because the data of both the months have same index

Dropping index column from the dataset
concatenating two datasets together
DAILY_CALL_DATASET = pd.concat([df1 , df2] , axis = 0)
resetting index because the data of both the months have same index
DAILY_CALL_DATASET = DAILY_CALL_DATASET.reset_index()
Dropping index column from the dataset
DAILY_CALL_DATASET.drop(['index'], axis=1, inplace=True)
DAILY_CALL_DATASET

Now we should analyse our dataset.
We will import seaborn library to plot a boxplot which will describe to us about outliers.
Checking for the Outliers is an important step which will help us in deciding best evaluation metrics.

import seaborn as sns
sns.boxplot(x = DAILY_CALL_DATASET["DAILY_TOTAL"])

Our data has No Outliers
Because Our dataset is not large we will only split in train/test set
No validation set will be used for evaluation of performance

HERE WE IMPORT IT FROM SKLEARN
from sklearn.model_selection import train_test_split as tts
HERE WE SPLIT DATA INTO TRAIN TEST SPLIT
X_train,X_test,y_train,y_test = tts(DAILY_CALL_DATASET.drop('DAILY_TOTAL',axis = 1),DAILY_CALL_DATASET['DAILY_TOTAL'],test_size = 0.3,random_state = 1)

from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from sklearn.ensemble import RandomForestRegressor

WE have INITIALIZE THE Random Forest Regressor with no: of trees as 200, max_depth as 9, n_jobs as -1(read documentation for more details) and random state as 1
rfr = RandomForestRegressor(n_estimators= 200 , max_depth=9 , n_jobs=-1 , random_state= 1)

FIT THE DATA using FIT METHOD
Calculate the R2_score, mse, mae for training data as well as testing data.

rfr.fit(X_train , y_train)

mae_train = mean_absolute_error((y_train) ,(rfr.predict(X_train)))
mae_test = mean_absolute_error((y_test) , (rfr.predict(X_test)))
mse_train = mean_squared_error((y_train) ,(rfr.predict(X_train)))
mse_test = mean_squared_error((y_test) , (rfr.predict(X_test)))
rmse_train = np.sqrt(mean_squared_error((y_train) ,(rfr.predict(X_train))))
rmse_test = np.sqrt(mean_squared_error((y_test) , (rfr.predict(X_test))))
R_score_train = r2_score((y_train) , (rfr.predict(X_train)))
R_score_test = r2_score((y_test) , (rfr.predict(X_test)))

print("ACCURACY : ")
print("Mean Absolute Error Training Set")
print(mae_train)
print("Mean Absolute Error Test Set")
print(mae_test)

print("")
print("Higher the value greater the accuracy")
print("R2 Score Training Set")
print(R_score_train)
print("R2 Score Test Set")
print(R_score_test)

```
print("")
print("")
print("")

print("Root Mean Squared Error Training Set")
print(rmse_train)
print("Root Mean Squared Error Test Set")
print(rmse_test)

print("Mean Squared Error Training Set")
print(mse_train)
print("Mean Squared Error Test Set")
print(mse_test)
```

Calculating Number of Engineers required

```
data = [[15 , 0]]
```

Predicting forDate = 15   and a   Weekday (not Weekend)

```
rfr.predict(data)
```

Further Processing
The model predicts the number of calls.
Average Handling Time of call is given 14 minutes.
Engineers works for 7 hours 30 minutes (= 450 minutes) per day.
The number of minutes workers are required per day = the number of call   Average Handling Time of call
Thus, the number of resources required = (The number of minutes workers are required) / (Number of minutes each workers works per day)

```
number_of_calls=181.75466667
number_of_resourses_required=(number_of_calls) 14/450
number_of_resourses_required
```

6 Engineers required on Date 15 with Weekday for handling Calls

DAILY_BASIS_CHAT

Data Pre-Processing in Excel
Extracted DAY, DATE and DAILY_TOTAL from Chats Volume Excel sheet.

importing pandas and numpy
import pandas as pd
import numpy as np

1. Uploading Excel sheets

from google.colab import files

uploaded1=files.upload()
uploaded2 = files.upload()

2. Reading files with Pandas

import io
df1=pd.read_excel(io.BytesIO(uploaded1['Chat_Volume_Feb_cleaned.xlsx']))
df2=pd.read_excel(io.BytesIO(uploaded2['Chat_Volume_March_cleaned.xlsx']))

extracting dates from index by adding 1 because data provided is monthly day
df1['DATE_EXTRACTED'] = df1.index + 1
df2['DATE_EXTRACTED'] = df2.index + 1

marking   weekends as 1 and weekdays as 0
df1['Weekend'] = np.where(((df1['DAY'] == 'Saturday') | (df1['DAY'] == 'Sunday')), 1, 0)
df2['Weekend'] = np.where(((df2['DAY'] == 'Saturday') | (df2['DAY'] == 'Sunday')), 1, 0)

dropping DATE and DAY column from datasheet
df1.drop(['DATE' , 'DAY'], axis=1, inplace=True)
df2.drop(['DATE' , 'DAY'], axis=1, inplace=True)
df1

df2

concatenating two datasets together
DAILY_CHAT_DATASET = pd.concat([df1 , df2] , axis = 0)
resetting index because the data of both the months have same index
DAILY_CHAT_DATASET = DAILY_CHAT_DATASET.reset_index()
Dropping index column from the dataset
DAILY_CHAT_DATASET.drop(['index'], axis=1, inplace=True)

DAILY_CHAT_DATASET

Now we should analyse our dataset.
We will import seaborn library to plot a boxplot which will describe to us about outliers.
Checking for the Outliers is an important step which will help us in deciding the best evaluation metrics.

import seaborn as sns
sns.boxplot(x = DAILY_CHAT_DATASET["DAILY_TOTAL"])

Our dataset has No Outliers

3. Applying RANDOM FOREST REGRESSION on the data

We will use Random Forest Regressor as our model to predict the data(number of chats per hour).
Also, we will measure our performance using R2 score and will measuring Error by various methods such as Mean_Squared_Error , Root_Mean_Squared_Error, Mean_Absolute_Error.
Because our data has Outlier, Mean Absolute Error will be the best metric to evaluate our result.
No Validation Set , because dataset is not large

HERE WE IMPORT IT FROM SKLEARN
from sklearn.model_selection import train_test_split as tts

HERE WE SPLIT DATA INTO TRAIN TEST SPLIT
X_train,X_test,y_train,y_test = tts(DAILY_CHAT_DATASET.drop('DAILY_TOTAL',axis = 1),DAILY_CHAT_DATASET['DAILY_TOTAL'],test_size = 0.3,random_state = 1)

from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error *from sklearn.ensemble import RandomForestRegressor*

WE have INITIALIZE THE Random Forest Regressor with no: of trees as 200, max_depth as 9, n_jobs as -1(read documentation for more details) and random state as 1

rfr = RandomForestRegressor(n_estimators= 200 , max_depth=9 , n_jobs=-1 , random_state= 1)

FIT THE DATA using FIT METHOD

Calculate the R2_score, mse, mae for training data as well as testing data.

```
rfr.fit(X_train , y_train)

mae_train = mean_absolute_error((y_train) ,(rfr.predict(X_train)))
mae_test = mean_absolute_error((y_test) , (rfr.predict(X_test)))
mse_train = mean_squared_error((y_train) ,(rfr.predict(X_train)))
mse_test = mean_squared_error((y_test) , (rfr.predict(X_test)))
rmse_train = np.sqrt(mean_squared_error((y_train) ,(rfr.predict(X_train))))
rmse_test = np.sqrt(mean_squared_error((y_test) , (rfr.predict(X_test))))
R_score_train = r2_score((y_train) , (rfr.predict(X_train)))
R_score_test = r2_score((y_test) , (rfr.predict(X_test)))

print("ACCURACY : ")
print("Mean Absolute Error Training Set")
print(mae_train)
print("Mean Absolute Error Test Set")
print(mae_test)

print("")
print("Higher the value greater the accuracy")
print("R2 Score Training Set")
print(R_score_train)
print("R2 Score Test Set")
print(R_score_test)

print("")
print("")
print("")
print("Mean Squared Error Training Set")
```

print(mse_train)
print("Mean Squared Error Test Set")
print(mse_test)

Calculating Number of Engineers required
data = [[15 , 0] ,]

Predicting for Date = 15 and Weekday (not weekend)
rfr.predict(data)

Further Processing
The model predicts the number of chats.
Average Handling Time of chat is given 23 minutes.
Engineers works for 7 hours 30 minutes (= 450 minutes) per day.
The number of minutes workers are required per day = the number of chat Average Handling Time of chat.
Thus, the number of resources required = (The number of minutes workers are required) / (Number of minutes each workers works per day)

number_of_chats=395.228875
number_of_resourses_required=(number_of_chats) 23/450
number_of_resourses_required

21 Engineers required on Date 15 with Weekday for handling Chat

<u>Weekly_Basis_Chat</u>

Data Pre-Processing in Excel
Extracted Month_Number,Week_Number and corresponding Weekly_total from the extracted excel sheet of Daily basis chat

importing pandas and numpy
import pandas as pd
import numpy as np

1. Uploading Excel sheets
from google.colab import files
chat_data_weekly=files.upload()

2. Reading files with Pandas
import io
df=pd.read_excel(io.BytesIO(chat_data_weekly['Chat_Weekly_Data.xlsx']))
df

Now we should analyse our dataset.
We will import seaborn library to plot a boxplot which will describe to us about outliers .
Checking for the Outliers is an important step which will help us in deciding the best evaluation metrics.
If there are outliers MAE is the best metric to evaluate errors .

from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

HERE WE SPLIT DATA INTO TRAIN TEST SPLIT
X_train,X_test,y_train,y_test = tts(df.drop('Weekly_Total',axis = 1),df['Weekly_Total'],test_size = 0.5,random_state = 1)

Because our dataset is very small it is prone to OVERFITTING, therefore we will not use R-squared
Because of outliers Mean Absolute Error is best method

from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(n_estimators= 1000 , max_depth=20 , n_jobs=-1 , random_state= 1)

FIT THE DATA using FIT METHOD
Calculate the R2_score, mse, mae for training data as well as testing data.

```
rfr.fit(X_train , y_train)
import numpy as np
mae_train = mean_absolute_error((y_train) ,(rfr.predict(X_train)))
mae_test = mean_absolute_error((y_test) , (rfr.predict(X_test)))
mse_train = mean_squared_error((y_train) ,(rfr.predict(X_train)))
mse_test = mean_squared_error((y_test) , (rfr.predict(X_test)))
rmse_train = np.sqrt(mean_squared_error((y_train) ,(rfr.predict(X_train))))
rmse_test = np.sqrt(mean_squared_error((y_test) , (rfr.predict(X_test))))
R_score_train = r2_score((y_train) , (rfr.predict(X_train)))
R_score_test = r2_score((y_test) , (rfr.predict(X_test)))

print("ACCURACY : ")
print("Because our dataset is very small it is prone to OVERFITTING, therefore we will not use R-squared")
print("because of outliers Mean Absolute Error is best method")
print("Mean Absolute Error Training Set")
print(mae_train)
print("Mean Absolute Error Test Set")
print(mae_test)

print("Higher the value greater the accuracy")
print("R2 Score Training Set")
print(R_score_train)
print("R2 Score Test Set")
print(R_score_test)
print("")
print("")
print("")

print("Root Mean Squared Error Training Set")
print(rmse_train)
```

print("Root Mean Squared Error Test Set")
print(rmse_test)

print("Mean Squared Error Training Set")
print(mse_train)
print("Mean Squared Error Test Set")
print(mse_test)

Calculating Number of Engineers required

Lets Predict data for Month_Number 4 and Week_Number 2

rfr.predict([[4 , 2] ,])

Further Processing
The model predicts the number of chats.

number_of_chats=2369.848

Average Handling Time of chat is given 23 minutes.
Engineers works for 7 hours 30 minutes (= 450 minutes) per day.
So, they will work for 2250 minutes(= 450 minutes 5days) per week.
The number of minutes workers are required per week = the number of chats Average Handling Time of chat
Thus, the number of resources required = (The number of minutes workers are required) / (Number of minutes each workers works per week)
number_of_resourses_required=(number_of_chats) 23/2250
number_of_resourses_required

25 Engineers required on Week 2 and month 4 to handle Chats.

Weekly_Basis_Call

Data Pre-Processing in Excel
Extracted Month_Number,Week_Number and corresponding Weekly_total from the extracted excel sheet of Daily basis calls

importing pandas and numpy
import pandas as pd
import numpy as np

1. Uploading Excel sheets
from google.colab import files
call_data_weekly=files.upload()

2. Reading files with Pandas
import io
df=pd.read_excel(io.BytesIO(call_data_weekly['Call_weekly_data.xlsx']))
df

Now we should analyse our dataset.
We will import seaborn library to plot a boxplot which will describe to us about outliers.
Checking for the Outliers is an important step which will help us in deciding best evaluation metrics.

import seaborn as sns
sns.boxplot(x = df["Weekly_Total"])

Our dataset has No Outliers

from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

No validation dataset because dataset is too small
Only Train/test set.
Test Set size is 50% to avoid overfitting caused due to a very small dataset.

HERE WE SPLIT DATA INTO TRAIN TEST SPLIT

X_train,X_test,y_train,y_test = tts(df.drop('Weekly_Total',axis = 1),df['Weekly_Total'],test_size = 0.5,random_state = 1)

Because our dataset is very small it is prone to OVERFITTING, therefore we will not use R-squared
because of outliers Mean Absolute Error is best method
We will use Random Forest Regressor as our model to predict the data(number of calls per hour).
Also, we will avoid using R2 score because of overfitting caused due to small dataset
Because our data has Outlier, Mean Absolute Error will be the best metric to evaluate our result.

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators= 200 , max_depth=9 , n_jobs=-1 ,
random_state= 1)
```

FIT THE DATA using FIT METHOD
Calculate the R2_score, mse, mae for training data as well as testing data.

```
rfr.fit(X_train , y_train)

import numpy as np
mae_train = mean_absolute_error((y_train) ,(rfr.predict(X_train)))
mae_test = mean_absolute_error((y_test) , (rfr.predict(X_test)))
mse_train = mean_squared_error((y_train) ,(rfr.predict(X_train)))
mse_test = mean_squared_error((y_test) , (rfr.predict(X_test)))
rmse_train = np.sqrt(mean_squared_error((y_train) ,(rfr.predict(X_train))))
rmse_test = np.sqrt(mean_squared_error((y_test) , (rfr.predict(X_test))))
R_score_train = r2_score((y_train) , (rfr.predict(X_train)))
R_score_test = r2_score((y_test) , (rfr.predict(X_test)))

print("ACCURACY : ")
print("Because our dataset is very small it is prone to OVERFITTING, therefore we will not
use R-squared")
print("Mean Absolute Error Training Set")
print(mae_train)
print("Mean Absolute Error Test Set")
print(mae_test)
```

```
print("Higher the value greater the accuracy")
print("R2 Score Training Set")
print(R_score_train)
print("R2 Score Test Set")
print(R_score_test)

print("Root Mean Squared Error Training Set")
print(rmse_train)
print("Root Mean Squared Error Test Set")
print(rmse_test)
print("Mean Squared Error Training Set")
print(mse_train)
print("Mean Squared Error Test Set")
print(mse_test)
```

Calculating Number of Engineers required

Predicting for Month_Number 4 and Week_Number 2

```
data = [[4 , 2],]

rfr.predict(data)
```

Further Processing

The model predicts the number of calls.

```
number_of_calls=828.605
```

Average Handling Time of call is given 14 minutes.
Engineers work for 7 hours 30 minutes (= 450 minutes) per day.
So, they will work for 2250 minutes(= 450 minutes 5days) per week.
The number of minutes workers are required per week = the number of calls Average Handling Time of calls
Thus, the number of resources required = (The number of minutes workers are required) / (Number of minutes each workers works per week)

```
number_of_resourses_required=(number_of_calls) 14/2250
number_of_resourses_required
```

6 Engineers required on Week 2 and month 4 to handle Calls.