Team: 14
Professor: Yugyung Lee

Name: Sneha Mishra
Class ID: 19
Email: smccr@mail.umkc.edu
MyGitHub

Technical Partner:
Name: Raju Nekadi
Class ID: 20
Email: rn8mh@mail.umkc.edu
GitHub

Introduction:

In todays world most of the top institution around the world mostly depend upon the IT infrastructure availability, web site availability plays very important role.

In order to make sure their Web Infrastructure available 24/7 to their customer organization around the world are spending a lot money in various research and solutions.

The main proposal of this project address one such solution where we try to predict future traffics for website and helps plan for infrastructure accordingly.

Objective:

The main objective of this project is very simple analytical work of taking the historical data and predicting the future.

The objective of this project is to predict future web traffic for given web pages of their site that are popular and if there are any apparent trends, such as one specific page being viewed mostly by people in a particular country.

It mainly focuses on forecasting the traffic on multiple time series values.

Motivation:

Being in professional world we came across a situation for client where they were not ready to support large incoming web request and end up losing the business.

That is the main motive for this project to implement solution for one such situation.

Technologies Used:

- This project mainly involves use of Machine learning and Deep learning frameworks, models and tools.
- The first phase is to use Regression model from Machine Learning library and get our prediction.
- The second phase is to use Deep Learning model bases RNN (Recurrent neural network), LSTM, Keras.
- Facebook New Forecasting model prophet - fbprophet
- Matplotlib, Seaborn and Cross validation libraries.

Approach:

We divided the project in 2 parts: Model Analysis and Data Analysis, which will help us in the analysis and prediction.

- Model Analysis: Create a model (RNN, then LSTM), compile and fit the model, then calculate the loss and accuracy.

- Data Analysis: Get the different data forms from the available dataset, do the Feature Scale on the data, then analyze and visualize the different categories.

Workflow:

Model Analysis:

Load the model:

Load the model in csv format

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


train =pd.read_csv("/content/drive/My Drive/Deep_Learning_Project_Fall_2018/data/train.csv")
page = train['Page']
train.head()
```

Display the first 5 rows of the dataset

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	15.0	26.0	14.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	28.0	15.0	9.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	1.0	7.0	4.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	26.0	27.0	16.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	25.0	13.0	3.0

5 rows × 551 columns

Or Mount the model using content drive in Colaboratory

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6b

Enter your authorization code:
··········
Mounted at /content/drive/
```

Work on the dataset:

Filling the Nan Value with Zeros.
Our dataset had more than 150k rows, so we consider only 90k rows for our project.
Also defining the X and Y for the model.

```
# Filling the Nan Value with Zeros

train.fillna(0, inplace=True)


row = train.iloc[90000, :].values
X = row[1:549]
y = row[2:550]
```

Splitting the dataset into the Training set and Test set.
Feature Scaling.
And Reshaping the Array.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
import numpy as np
sc = MinMaxScaler()
X_train = np.reshape(X_train, (-1, 1))
y_train = np.reshape(y_train, (-1, 1))
X_train = sc.fit_transform(X_train)
y_train = sc.fit_transform(y_train)


#Reshaping Array
X_train = np.reshape(X_train, (383, 1, 1))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:475: DataConversionWarning: Data with input dtype object was converted to float64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)
```

Create the model:

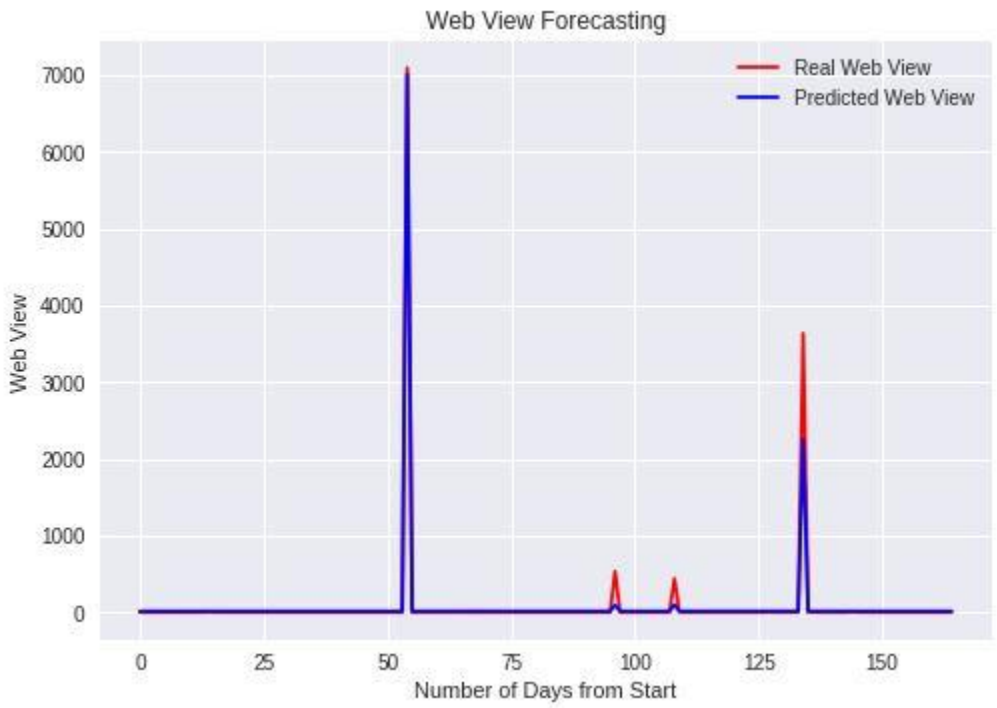
Importing the Keras libraries and packages for LSTM
Initialising the RNN
Adding the input layerand the LSTM layer
Adding the output layer
Compiling the RNN
Fitting the LSTM to the Training set

```
# Importing the Keras libraries and packages for LSTM
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
#
# Initialising the RNN
lstm_model = Sequential()
#
# Adding the input layerand the LSTM layer
lstm_model.add(LSTM(units = 100, activation = 'relu', input_shape = (None, 1)))

# Adding the output layer
lstm_model.add(Dense(units = 1))


# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics=['accuracy'])

# Fitting the LSTM to the Training set
history = lstm_model.fit(X_train, y_train, batch_size = 10, epochs = 100, verbose = 0 )
```

Analyze and Predict on the Model:

Getting the predicted Web View
Visualizing Result

```
# Getting the predicted Web View
inputs = X_test
inputs = np.reshape(inputs,(-1,1))
inputs = sc.transform(inputs)
inputs = np.reshape(inputs, (165, 1, 1))
y_pred = lstm_model.predict(inputs)
y_pred = sc.inverse_transform(y_pred)

# Visualising Result

plt.figure
plt.plot(y_test, color = 'red', label = 'Real Web View')
plt.plot(y_pred, color = 'blue', label = 'Predicted Web View')
plt.title('Web View Forecasting')
plt.xlabel('Number of Days from Start')
plt.ylabel('Web View')
plt.legend()
plt.show()
```



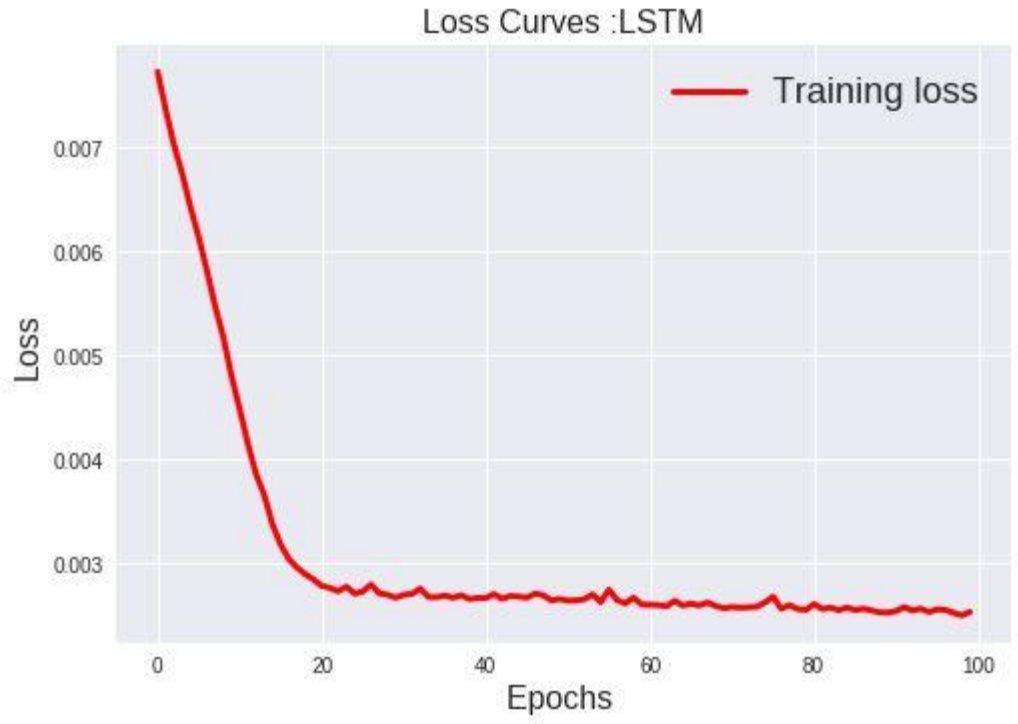
Evaluating and Plotting the Accuracy and Score

```
# Evaluating and Accuracy and Score

train_score = lstm_model.evaluate(X_train, y_train, verbose=2)

print('Train MAE: ', round(train_score[1], 4), ', Train Loss: ', round(train_score[0], 4))

score, accuracy = lstm_model.evaluate(X_train, y_train, verbose=2)

print('Validation Accuracy: %.2f' % (accuracy))
```

```
Train MAE:  0.9373 , Train Loss:  0.0026
Validation Accuracy: 0.94
```

```
fig1 = plt.figure()
plt.plot(history.history['loss'], 'r', linewidth=3.0)

plt.legend(['Training loss', 'Validation Loss'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Loss', fontsize=16)
plt.title('Loss Curves :LSTM', fontsize=16)
```

```
Text(0.5,1,'Loss Curves :LSTM')
```



Data Analysis:

Load the data and print the first 5 rows

```
import pandas as pd

train = pd.read_csv("/content/drive/My Drive/Deep_Learning_Project_Fall_2018/data/train.csv")
train.head()
```

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0

5 rows × 551 columns

Map the languages of each page and display it

```
import re

def get_language(page):
    res = re.search('[a-z][a-z].wikipedia.org', page)
    if res:
        return res[0][0:2]
    return 'na'


train['lang'] = train.Page.map(get_language)
train.head()
```

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	lang
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	63.0	15.0	26.0	14.0	20.0	22.0	19.0	18.0	20.0	zh
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	42.0	28.0	15.0	9.0	30.0	52.0	45.0	26.0	20.0	zh
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	1.0	1.0	7.0	4.0	4.0	6.0	3.0	4.0	17.0	zh
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	10.0	26.0	27.0	16.0	11.0	17.0	19.0	10.0	11.0	zh
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	9.0	25.0	13.0	3.0	11.0	27.0	13.0	36.0	10.0	zh

5 rows × 552 columns

Group the pages and print the language, visits and date

```
train_flattened = pd.melt(train[list(train.columns[-91:])+['Page']], id_vars=['Page', 'lang'], var_name='date',
                                                                    value_name='Visits')
train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
train_flattened['weekend'] = ((train_flattened.date.dt.dayofweek) // 5 == 1).astype(float)

train_flattened.head()
```

	Page	lang	date	Visits	weekend
0	2NE1_zh.wikipedia.org_all-access_spider	zh	2016-10-03	12.0	0.0
1	2PM_zh.wikipedia.org_all-access_spider	zh	2016-10-03	20.0	0.0
2	3C_zh.wikipedia.org_all-access_spider	zh	2016-10-03	6.0	0.0
3	4minute_zh.wikipedia.org_all-access_spider	zh	2016-10-03	23.0	0.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	zh	2016-10-03	0.0	0.0

Get the median

```
df_median = pd.DataFrame(train_flattened.groupby(['Page'])['Visits'].median())
df_median.columns = ['median']
df_median.head()
```

Page	median
!vote_en.wikipedia.org_all-access_all-agents	3.0
!vote_en.wikipedia.org_all-access_spider	2.0
!vote_en.wikipedia.org_desktop_all-agents	3.0
"Awaken,_My_Love!"_en.wikipedia.org_all-access_all-agents	6213.0
"Awaken,_My_Love!"_en.wikipedia.org_all-access_spider	61.0

Get the mean

```
df_mean = pd.DataFrame(train_flattened.groupby(['Page'])['Visits'].mean())
df_mean.columns = ['mean']
df_mean.head()
```

Page	mean
!vote_en.wikipedia.org_all-access_all-agents	3.348315
!vote_en.wikipedia.org_all-access_spider	1.775281
!vote_en.wikipedia.org_desktop_all-agents	3.022472
"Awaken,_My_Love!"_en.wikipedia.org_all-access_all-agents	8959.833333
"Awaken,_My_Love!"_en.wikipedia.org_all-access_spider	107.750000

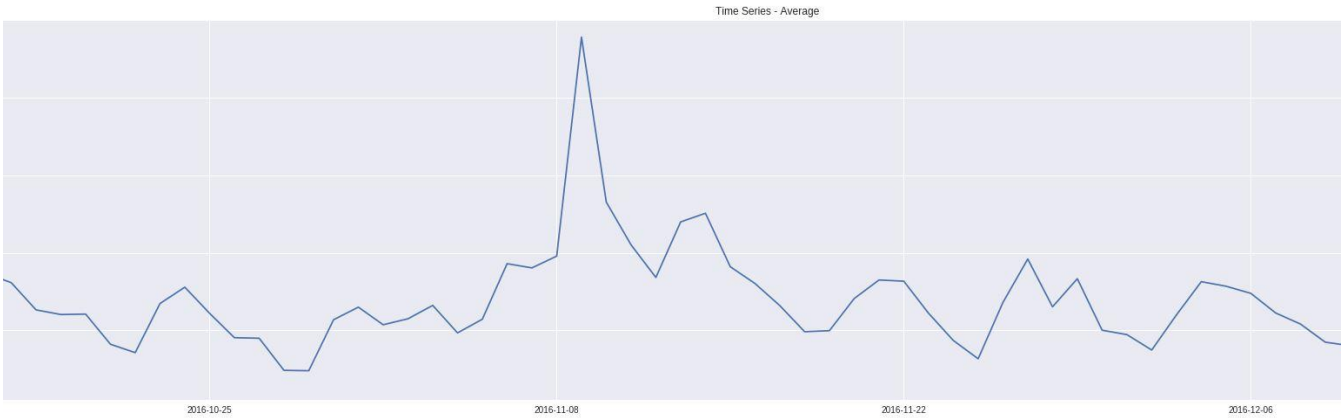
Get the weekdays of the visited site, to visualize the same

```
train_flattened = train_flattened.set_index('Page').join(df_mean, how='left', lsuffix='_left', rsuffix='_right').join(df_median, how='left',lsuffix='_left', rsuffix='_right')
train_flattened.reset_index(drop=False, inplace=True)
train_flattened['weekday'] = train_flattened['date'].apply(lambda x: x.weekday())

train_flattened['year']=train_flattened.date.dt.year
train_flattened['month']=train_flattened.date.dt.month
train_flattened['day']=train_flattened.date.dt.day

train_flattened.head()
```

	Page	lang	date	Visits	weekend	mean	median	weekday	year	month	day
0	!vote_en.wikipedia.org_all-access_all-agents	en	2016-10-03	7.0	0.0	3.348315	3.0	0	2016	10	3
1	!vote_en.wikipedia.org_all-access_all-agents	en	2016-10-04	2.0	0.0	3.348315	3.0	1	2016	10	4
2	!vote_en.wikipedia.org_all-access_all-agents	en	2016-10-05	5.0	0.0	3.348315	3.0	2	2016	10	5
3	!vote_en.wikipedia.org_all-access_all-agents	en	2016-10-06	3.0	0.0	3.348315	3.0	3	2016	10	6
4	!vote_en.wikipedia.org_all-access_all-agents	en	2016-10-07	1.0	0.0	3.348315	3.0	4	2016	10	7

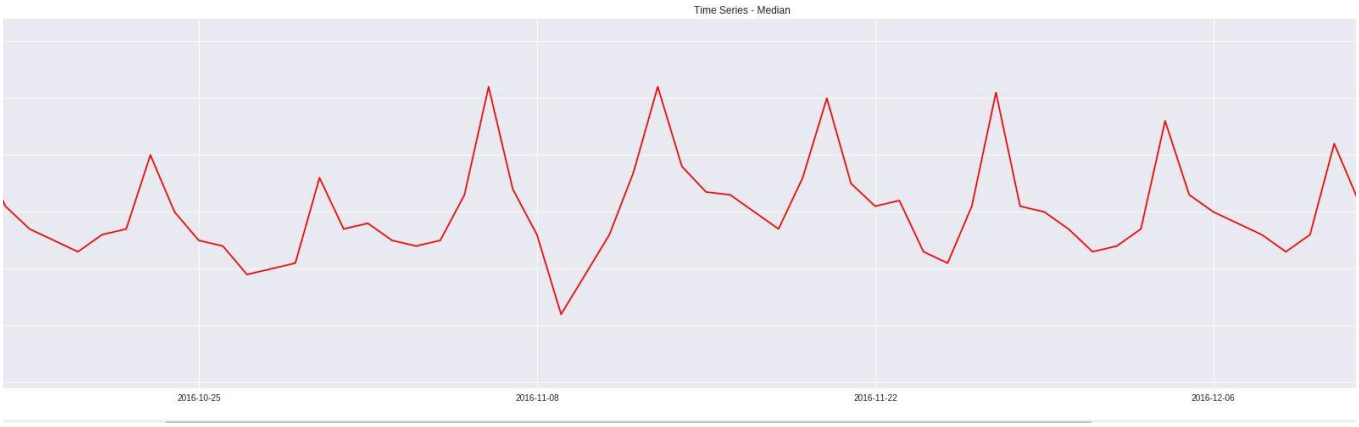
View the mean pattern (need to be linear)

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(50, 8))
mean_group = train_flattened[['Page', 'date', 'Visits']].groupby(['date'])['Visits'].mean()
plt.plot(mean_group)
plt.title('Time Series - Average')
```



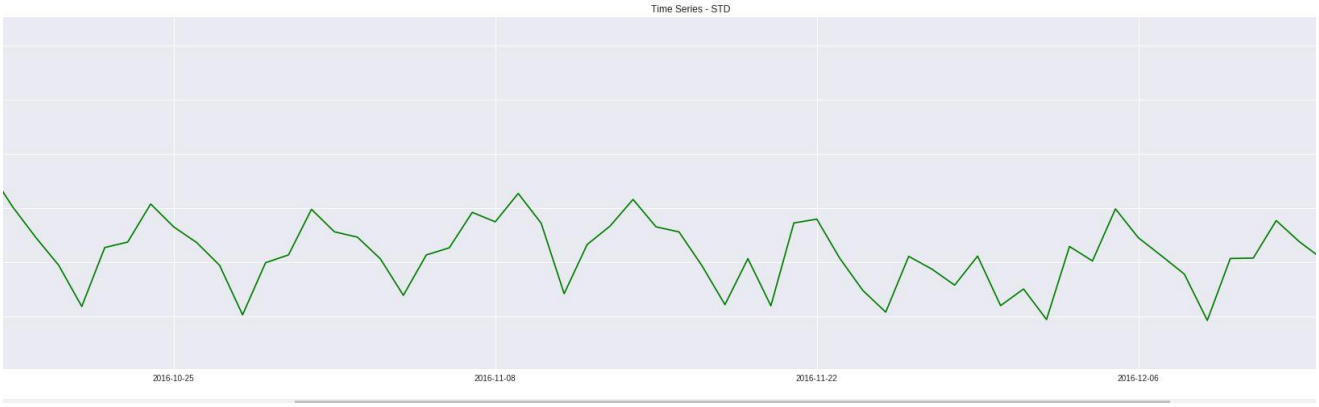
View the median pattern (need to be linear)

```
plt.figure(figsize=(50, 8))
median_group = train_flattened[['Page','date','Visits']].groupby(['date'])['Visits'].median()
plt.plot(median_group, color = 'r')
plt.title('Time Series - Median')
```



View the standard pattern (need to be linear)

```
plt.figure(figsize=(50, 8))
std_group = train_flattened[['Page','date','Visits']].groupby(['date'])['Visits'].std()
plt.plot(std_group, color = 'g')
plt.title('Time Series - STD')
```

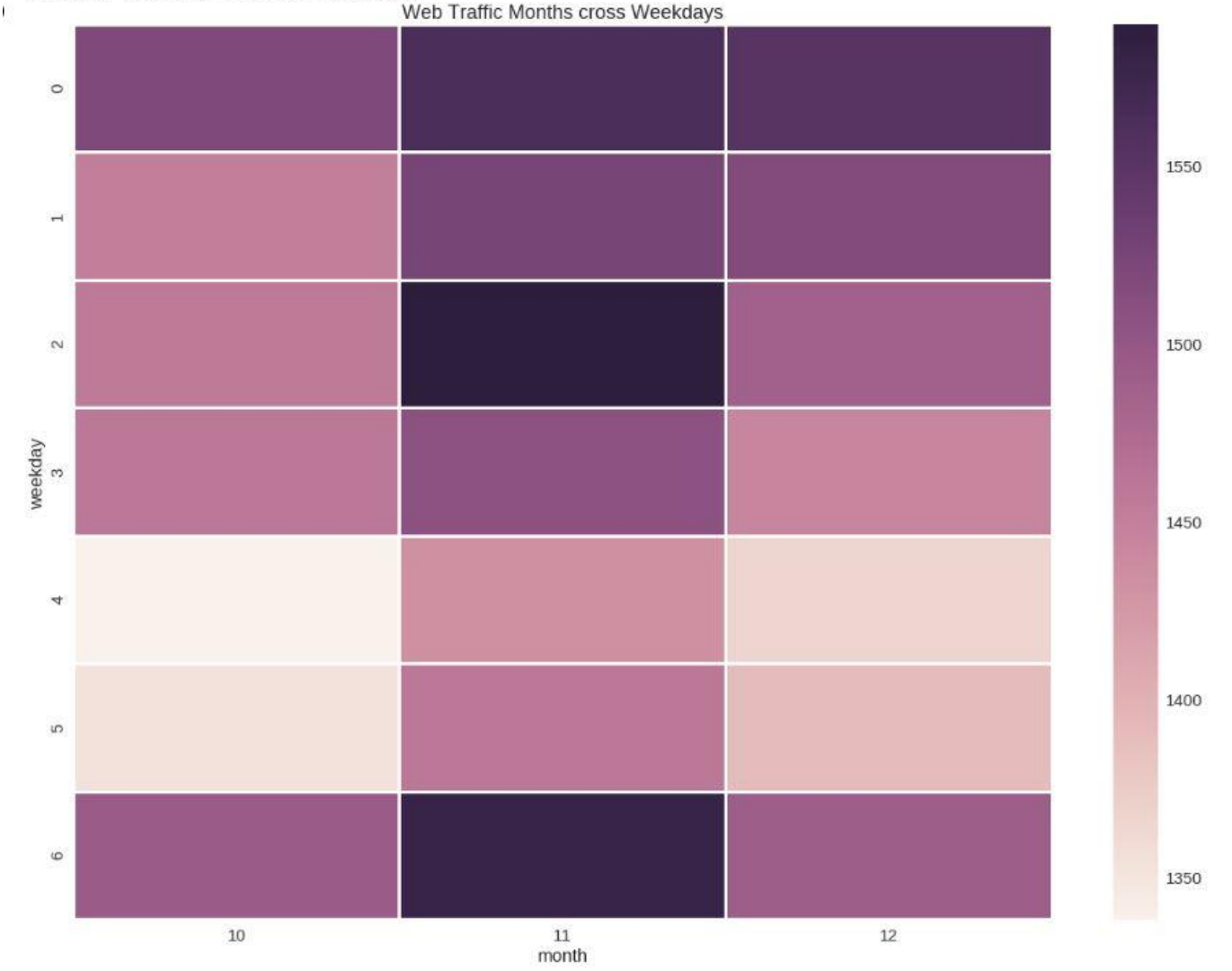


Assigning the months and days, also view the Web Traffic Months cross Weekdays in heatmaps

```
train_flattened['month_num'] = train_flattened['month']
train_flattened['month'].replace('11', '11 - Nov', inplace=True)
train_flattened['month'].replace('12', '12 - Dec', inplace=True)
train_flattened['month'].replace('10', '10 - Oct', inplace=True)
train_flattened['month'].replace('09', '12 - Sep', inplace=True)
train_flattened['month'].replace('08', '12 - Aug', inplace=True)
train_flattened['month'].replace('07', '12 - Jul', inplace=True)
train_flattened['month'].replace('06', '12 - Jun', inplace=True)
train_flattened['month'].replace('05', '12 - May', inplace=True)
train_flattened['month'].replace('04', '12 - Apr', inplace=True)
train_flattened['month'].replace('03', '12 - Mar', inplace=True)
train_flattened['month'].replace('02', '12 - Feb', inplace=True)
train_flattened['month'].replace('01', '12 - Jan', inplace=True)


train_flattened['weekday_num'] = train_flattened['weekday']
train_flattened['weekday'].replace(0, '01 - Monday', inplace=True)
train_flattened['weekday'].replace(1, '02 - Tuesday', inplace=True)
train_flattened['weekday'].replace(2, '03 - Wednesday', inplace=True)
train_flattened['weekday'].replace(3, '04 - Thursday', inplace=True)
train_flattened['weekday'].replace(4, '05 - Friday', inplace=True)
train_flattened['weekday'].replace(5, '06 - Saturday', inplace=True)
train_flattened['weekday'].replace(6, '07 - Sunday', inplace=True)
```

```
train_group = train_flattened.groupby(["month", "weekday"])['Visits'].mean().reset_index()
train_group = train_group.pivot('weekday', 'month', 'Visits')
train_group.sort_index(inplace=True)

sns.set(font_scale=1.5)

# Draw a heatmap with the numeric values in each cell
f, ax = plt.subplots(figsize=(20, 15))
sns.heatmap(train_group, annot=False, ax=ax, fmt="d", linewidths=2)
plt.title('Web Traffic Months cross Weekdays')
```

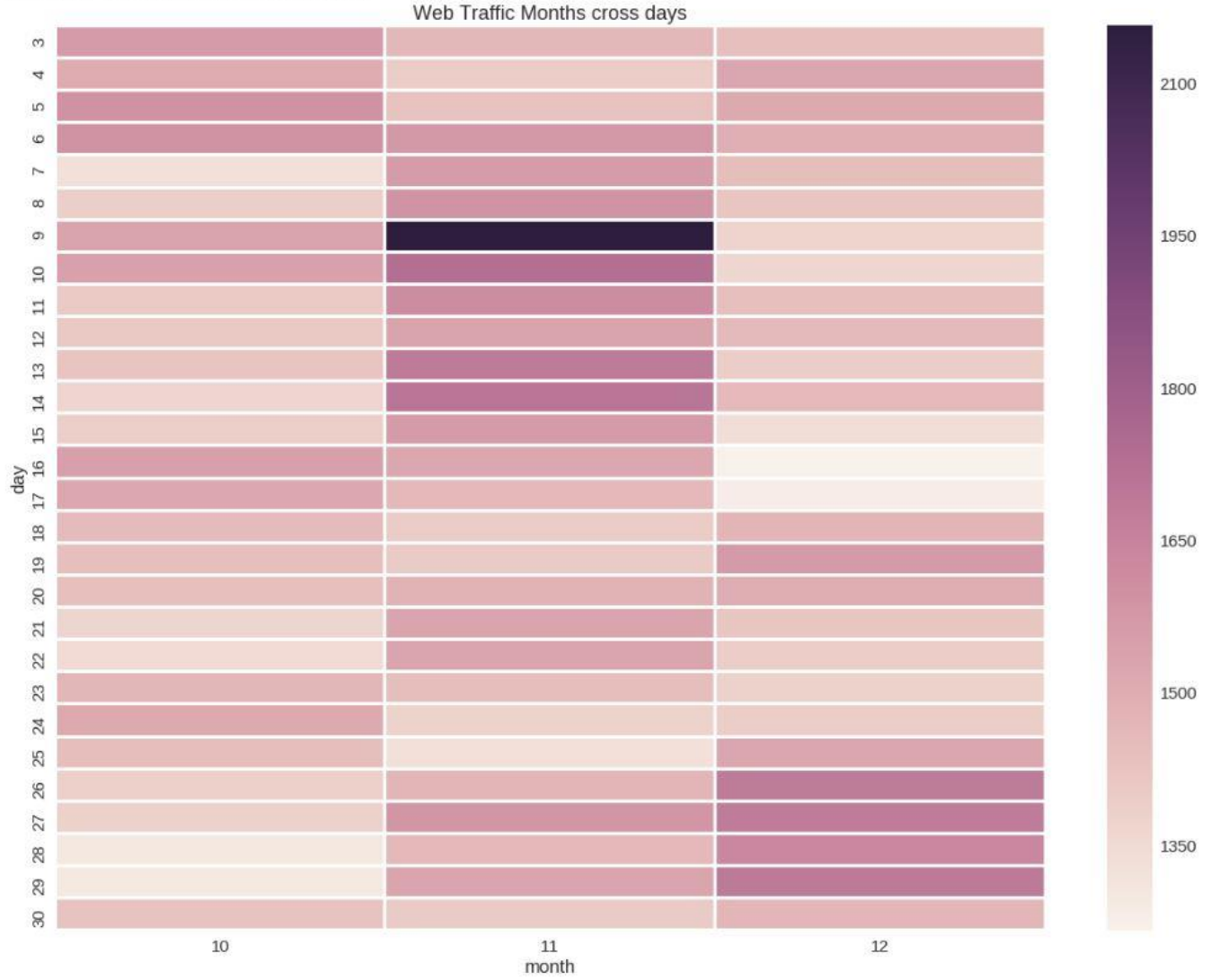
```
Text(0.5,1,'Web Traffic Months cross Weekdays')
```



View the Web Traffic Months cross days in heatmaps

```
train_day = train_flattened.groupby(["month", "day"])['Visits'].mean().reset_index()
train_day = train_day.pivot('day', 'month', 'Visits')
train_day.sort_index(inplace=True)
train_day.dropna(inplace=True)

# Draw a heatmap with the numeric values in each cell
f, ax = plt.subplots(figsize=(20, 15))
sns.heatmap(train_day, annot=False, ax=ax, fmt="d", linewidths=2)
plt.title('Web Traffic Months cross days')
```

```
Text(0.5,1,'Web Traffic Months cross days')
```



Print the count of languages visited, then view it

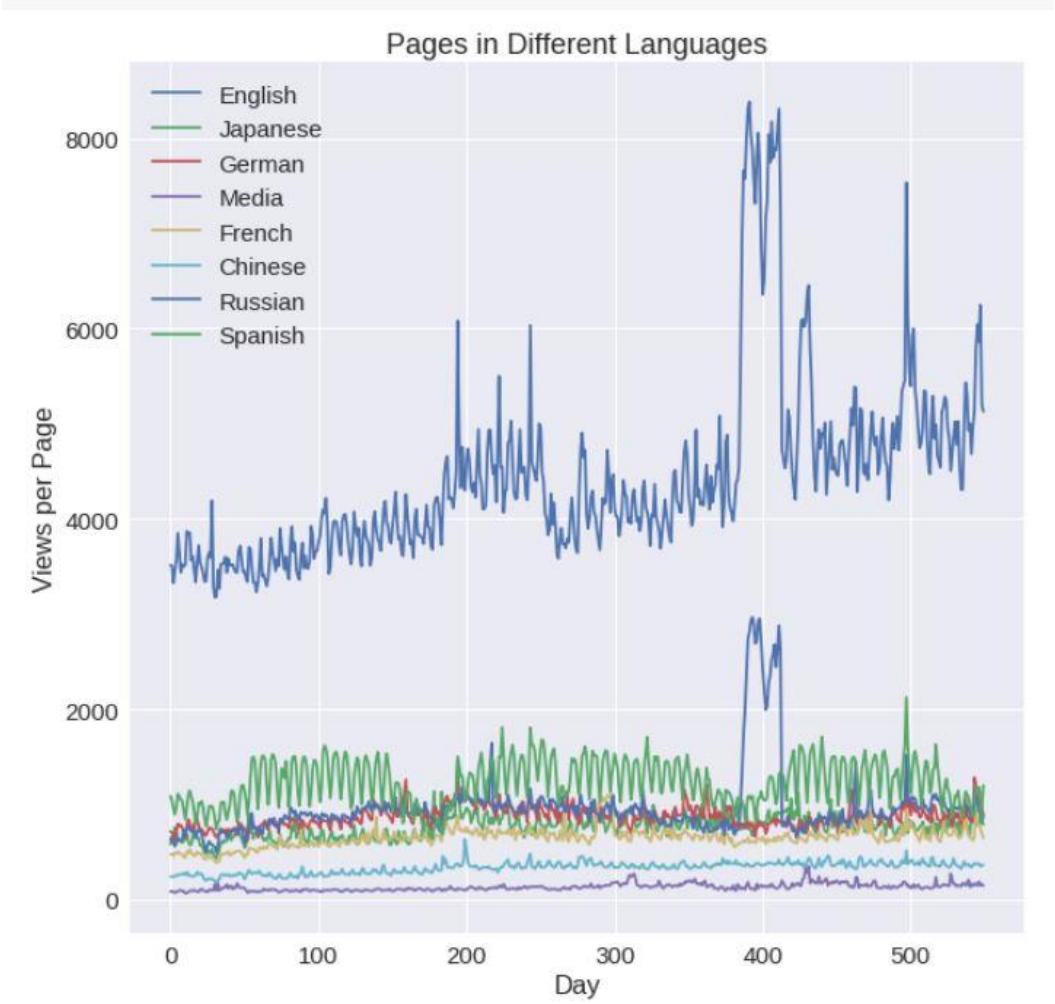
```
from collections import Counter
print(Counter(train.lang))
```

```
Counter({'en': 24108, 'ja': 20431, 'de': 18547, 'na': 17855, 'fr': 17802, 'zh': 17229, 'ru': 15022, 'es': 14069})
```

View pages visited in different language

```
lang_sets = {}
lang_sets['en'] = train[train.lang=='en'].iloc[:,0:-1]
lang_sets['ja'] = train[train.lang=='ja'].iloc[:,0:-1]
lang_sets['de'] = train[train.lang=='de'].iloc[:,0:-1]
lang_sets['na'] = train[train.lang=='na'].iloc[:,0:-1]
lang_sets['fr'] = train[train.lang=='fr'].iloc[:,0:-1]
lang_sets['zh'] = train[train.lang=='zh'].iloc[:,0:-1]
lang_sets['ru'] = train[train.lang=='ru'].iloc[:,0:-1]
lang_sets['es'] = train[train.lang=='es'].iloc[:,0:-1]

sums = {}
for key in lang_sets:
    sums[key] = lang_sets[key].iloc[:,1:].sum(axis=0) / lang_sets[key].shape[0]
```

```
days = [r for r in range(sums['en'].shape[0])]

fig = plt.figure(1,figsize=[10,10])
plt.ylabel('Views per Page')
plt.xlabel('Day')
plt.title('Pages in Different Languages')
labels={'en':'English','ja':'Japanese','de':'German',
        'na':'Media','fr':'French','zh':'Chinese',
        'ru':'Russian','es':'Spanish'
       }

for key in sums:
    plt.plot(days,sums[key],label = labels[key] )

plt.legend()
plt.show()
```



Dataset:

This Web Traffic Time Series Forecasting (Forecast future traffic to Wikipedia pages).
The file will look something like screenshot included
below:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	15.0	26.0	14.0	20.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	28.0	15.0	9.0	30.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	1.0	7.0	4.0	4.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	26.0	27.0	16.0	11.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	25.0	13.0	3.0	11.0

5 rows × 551 columns

Parameters:

- batch_size = 10
- epochs = 100
- activation = 'relu'
- test_size = 0.3
- row size = 90000

Evaluation:

Train MAE: 0.9373 , Train Loss: 0.0025
Validation Accuracy: 0.94

Conclusion:

LSTM is better for time series model and gives best solution, compared to other models in our course.

References:

1. https://en.wikipedia.org/wiki/Web_traffic
2. https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/