FlixBus Ticket Sales Forecast

Overview

This project is to forecast the number of tickets that FlixBus can sell during the next 11 days, starting from 'Feb 27th 2018', at each location through specific channel.

Forecasting sales or any other measures with periodical data at hand is considered as a time-series problem in machine learning.

In a simple time-series problem, we will predict the trend of a measure for 't+n' time periods for a single attribute.

However, since we have to forecast the country-wise ticket sales through specific channels, this can be treated a multi-level time series problem.

Solution Approach

I have followed a four-step approach for the solution

- **Data Gathering** – Data was given in three separate csv files, that need to be carefully merged.
- **EDA** – Data Wrangling and EDA will be done to get a preliminary understanding of the data distribution and to identify any issues with the data. This would also help us in carrying out the required treatment to the data.
- **Data Pre-Processing**– We will address the issues in our dataset by treating nulls, outliers, adding or dropping rows/columns as required. In this stage, we will prepare the data set that can be fed to the machine learning algorithm.
- **Machine Learning** – This is a multi-level timeseries problem that we have at hand. We can approach this problem in multiple ways. We can either loop over an appropriate time-series algorithm to get forecasts for each combination of country and channel or we can extract more features from date column and normalize the data to do a simple linear regression analysis. However, we will decide on which approach to follow based on our data analysis and the assumptions that we make.

Data Gathering

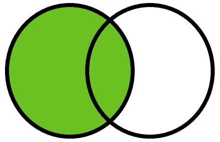
We have historical ticket sales data from 1st Jan 2017 till 26th Feb 2018, stored in three different tables.

- **orders_date** table contains order ***id***, ***date*** of sale and the ***channel_id*** of the channel through which the order was done.
- **orders_tickets** table contains order ***id***, number of tickets(***n_tickets***) in the order, and the ***type*** of product.
- **orders_country** table contains order ***id***, ***country_1*** and ***country_2*** for first and second country id details.

I have gathered the required data by merging these tables to get all attributes related to a ticket order by merging *order_tickets* table with other two tables by **left outer join** using Pandas module in python.

Orders_tickets

id	n_tickets	type
1586391	2	pax
438232	2	pax
270896	1	pax
1181593	2	pax
964842	1	pax
246783	1	pax
1693998	2	pax
1958576	1	other



Orders_date

id	date	channel_id
10173	6/12/2017	39
95062	9/11/2017	35
171081	7/5/2017	39
122867	8/18/2017	39

Orders_country

id	country_1	country_2
10503	24	
79360	24	0
81388	24	15
82828	24	9
38756	23	

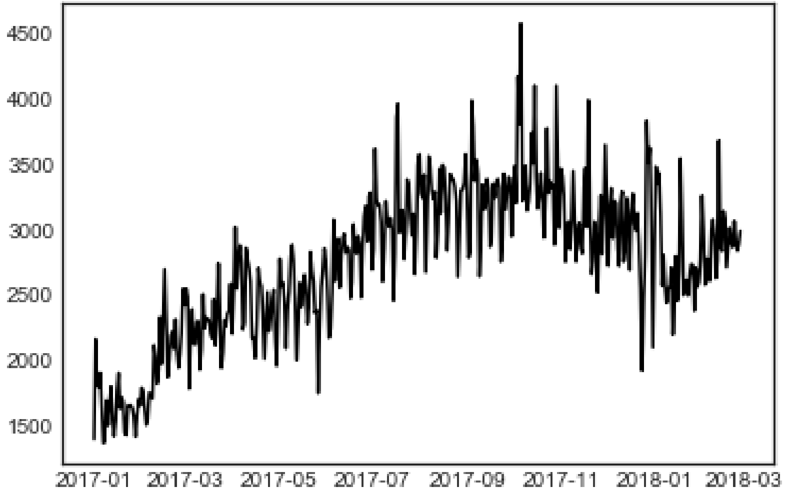
Merged Table

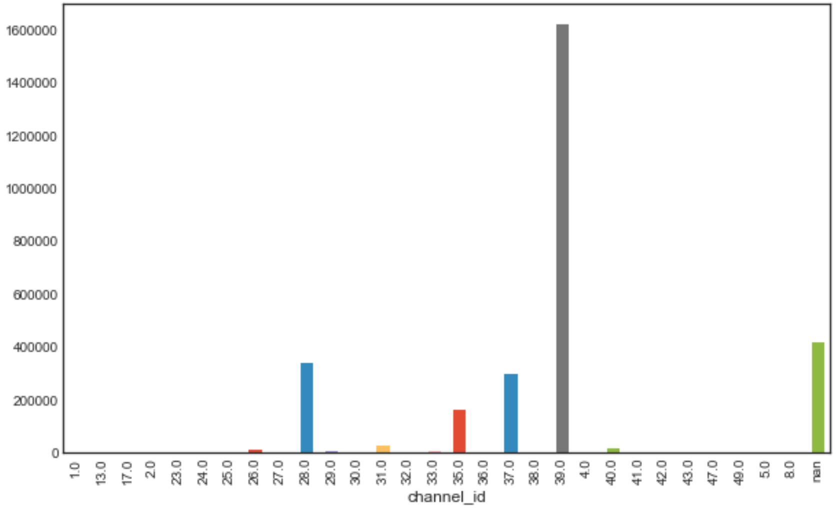
id	n_tickets	type	date	channel_id	country_1	country_2
1586391	2	pax	6/12/2017	39	24	
438232	2	pax	9/11/2017	35	24	0
270896	1	pax	7/5/2017	39	24	15
1181593	2	pax	8/18/2017	39	24	9
964842	1	pax	11/23/2017		23	
246783	1	pax	9/2/2017		19	13

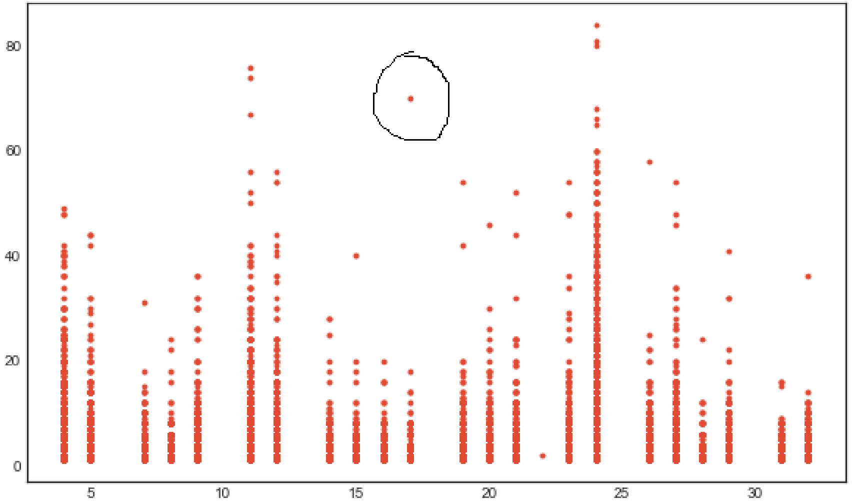
Exploratory Data Analysis

In this stage, I have performed exploratory data analysis and identified

- the target variable’s(*n_tickets*) data distribution for various country and channel combinations
- **duplicates** in orders_date table
- special country codes **xx** and **0** in country_2 column
- country 24 and channel 39 has more ticket sales
- **null** values in the dataset
- country_1, 22 has only two records in the data
- for many country and channel combinations have no data points, i.e., no ticket sales
- plotted to identify seasonality or any explainable spikes in the trends



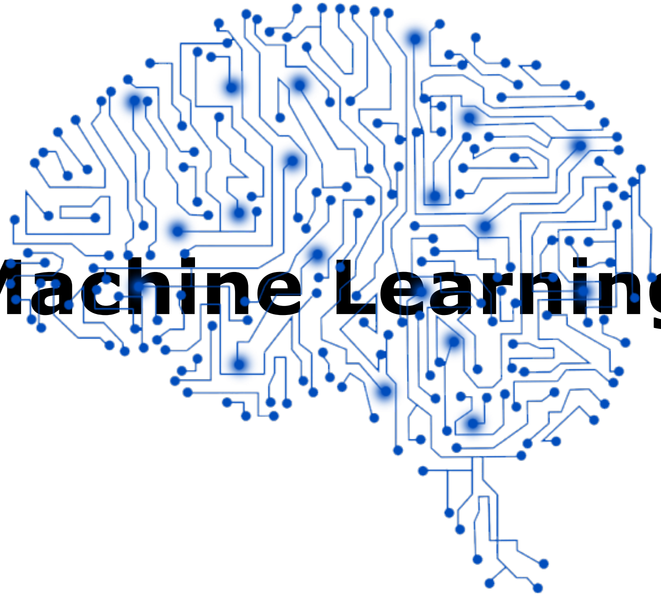




Data Pre-processing

Preparing and transforming the available raw data to algorithm-ready format is one of the biggest challenges of any machine learning project. It applies in this case too.

- **Nulls** in country_1 and channel_id were imputed using ***mode*** method. Country_2 value was first preferred for imputing country_1 nulls.
- **Dropped** id(insignificant column), type and country_2 columns.
- **Grouped** sum(n_tickets) by date, country and channel for maintaining single transactional data point per day per country by channel.
- Removed **duplicate** records in order_date table.
- Few **outliers**(n_tickets) were capped at grouped max values for channel_id and country_1 using percentile method. **Extensive outlier treatment** can be done with respect to **business context**.
- **Added records** for each day for country and channel combinations that do not have any sales. For these records, n_tickets are 0. This resulted in a perfect data set with continuous data points for all country-channel combinations.
- **Train/Test sets:** Cleaned dataset was then sliced in to two parts with records from 1^{st} Jan 2017 till 4^{th} Feb 2018 as ***train set*** and records from 5^{th} Feb 2018 to 26^{th} Feb 2018 as ***test set.***



Machine Learning

Assumptions

Before starting with modelling, the following assumptions were made based on the data analysis.

- Order at locations are uncorrelated i.e., orders at a particular country through a specific channel has no association to an order at a different country made through same or different channels. This assumption allows me to pick time-series approach over regular regression approach.
- No order record for a given country through a channel on a specific day is assumed as 0 sales and not a missing event.
- Shocks are randomly distributed with constant variance and has no seasonality in the trend.

ML Approach

As mentioned earlier, we can try two different machine learning approaches in solving this problem.

1. **Regression problem with cross-sectional data:** By extracting features like month, day and dayofweek and then creating more features like moving average, std, differenced mean etc, we can solve this in a cross sectional linear regression approach using linear regression models.
2. **Multi-level time series Analysis:** We can do regular time-series analysis and perform statistical tests to use time-series models. We can loop these models for each combination of country and channel.

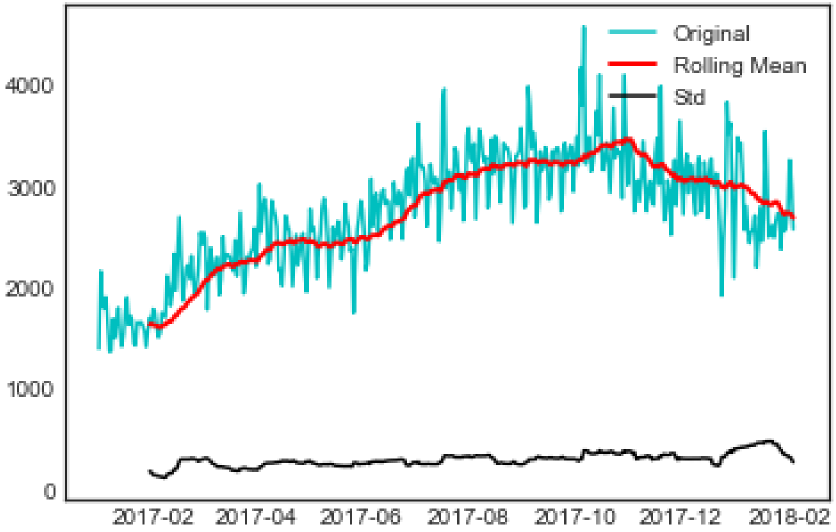
Based on my assumptions and data distribution for multiple country-channel combinations, I have decided to follow the **second approach**.

Statistical Analysis

The first assumption/criteria to perform the time series analysis is that my data should be stationary. We can check the stationarity of data by using following tests:

- **Rolling Stats Test**
  By plotting rolling mean and std for 30days interval, we can say whether the data looks stationary or not.
- **Dickey-Fuller Test**
  This is more advanced test that gives exact results such a p-value, test statistic and critical values. With the help of these values, we can confidently say whether a time-series is stationary or not.

I have performed both the tests on multiple country-channel combinations and inferred that my time-series is **not stationary. (P-value>alpha** & **critcal values** > **Test statistic)** and hence failed to reject null hypothesis that the data is not stationary.




```
Results of Dickey-Fuller Test:
Test Statistic                 -2.284919
p-value                         0.176878
#Lags Used                     14.000000
No. of Observations Used      384.000000
Critical value (1%)            -3.447495
Critical value (5%)            -2.869096
Critical value (10%)           -2.570795
dtype: float64
-
```

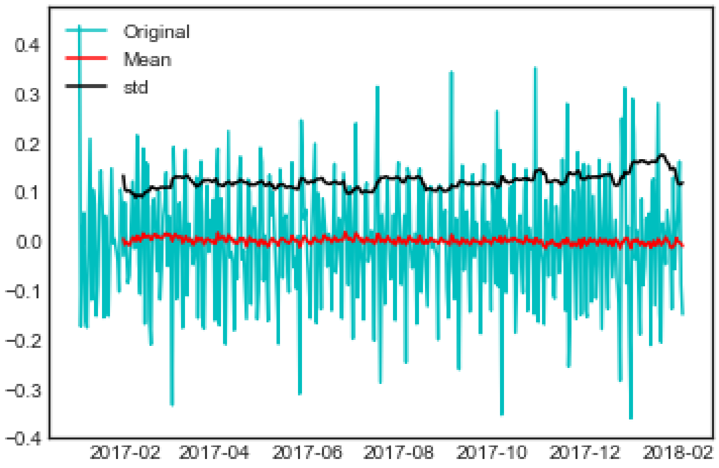
Statistical Analysis

Making time Series Stationary:

I tried the following methods to make the series stationary.

- **Log Transformation**
- **Log minus Moving Average**
- **Log Shift**

Of all, '**log shift**' has made the data stationary at a statistically significant level.




```
Results of Dickey-Fuller Test:
Test Statistic                -7.628246e+00
p-value                        2.041287e-11
#Lags Used                     1.300000e+01
No. of Observations Used       3.840000e+02
Critical value (1%)           -3.447495e+00
Critical value (5%)           -2.869096e+00
Critical value (10%)          -2.570795e+00
```

Then, I have induced noise using residuals to test stationary and the log shifted data remained stationary

AR-I-MA

Used ARIMA model to perform time series forecast on the stationary data.

AR value: Plotted Auto Correlation Function(ACF) to get optimal AR value (p)

I value: Since our data was not stationary and had to do one differencing, I value remained at 1. This was the case for most of the combinations and overall data. Hence I value remained at 1 forecasting all combinations.

MA value: Plotted Partial Auto Correlation Function(PACF) to get optimal MA values (q).

However, optimal p and q values have not performed well in all cases and hence tuned the values appropriately to get lower Residual Sum of Squares.

Looped Forecasting

For each combination of country-channel, I have looped the steps of log transformation, log shift and ARIMA model fit to get the forecasted predictions.

Since the p and q values vary for each data set, I have iterated them for 0-8 to get the best_model with lower RSS.

Out of 812 possible combinations of country-channel, 605 combinations did not have sufficient data points (at least 25) and hence these we ignored and predicted as zero sales. Iterated the loop for the other 207 combinations.

Model Evaluation

Forecasted ***n_tickets*** for each country through each channel from 4th Feb 2018 till 26th Feb 2018 (**test data dates**).

Evaluation Metric:

SMAPE: Symmetric Mean Absolute Percentage Error

$$\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|\text{Forcast}_t - \text{Actual}_t|}{|\text{Actual}_t| + |\text{Forcast}_t|}$$

Code in Python:

```
def smape(y, yhat):
        return 100 * np.mean(2 * np.abs(y - yhat)/(np.abs(y) + np.abs(yhat)))
```

Best model selected based on lowered **RSS** and the selected best model was evaluated using **SMAPE** metric.

In this case, actuals are the test data tickets, and the forecasted values were the model predictions.

Final Forecast: Same looped forecast technique was used to forecast the ticket sales from current date (27th Feb 2018) and the next 10 days i.e., till 9th March 2018.

Further Scope for Improvement

There could be **many** other possible and **innovative ways** to make this model better. Few of the options that could be tried to improve the model are listed below:

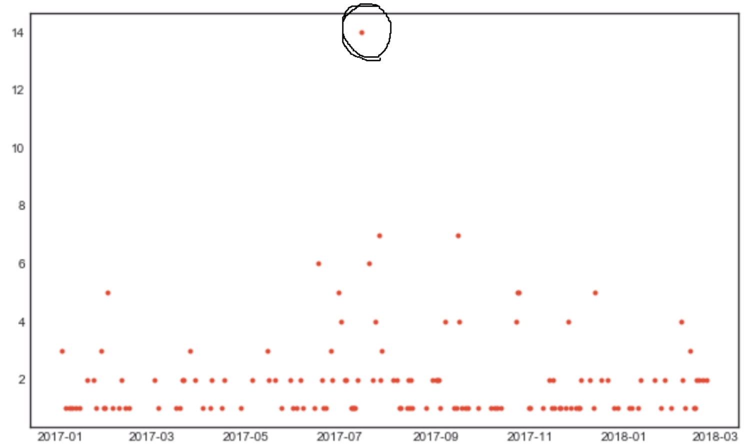
1. Outliers/unexplainable spikes in the time series should be addressed. This can be done by using standardization, outlier detection methods or reasonable treatment based on the business context.
2. I have generated a table to capture the best RSS for each of the country-channel combination. By carefully examining each of those trends with unreasonably high RSS, we can minimize the noise to get better predictions. I have tried doing the same for few observations. For eg: below table shows a high RSS for country 12 and channel 29 – when plotted, it seems to have an **outlier.**

country	channel	best_p-d-q	best_RSS
20	26	1-1-4	43.6495
20	29	4-1-5	19.8677
20	40	5-1-5	50.0924
24	25	6-1-5	49.5584
24	41	4-1-6	22.1308
12	**29**	**6-1-7**	**104.2427**
12	49	6-1-6	31.0183
12	33	6-1-7	35.9361

```
pch = df_3.loc[((df_3.country_1 == '12')&(df_3.channel_id=='29')), ['date', 'tickets']]
```

```
plt.plot(pch.date, pch.tickets, '.')
```

```
[<matplotlib.lines.Line2D at 0x180d2b64748>]
```



3. Furthermore, we can try other time series models like ARCH, GARCH etc to verify which model fares well in the long-run.

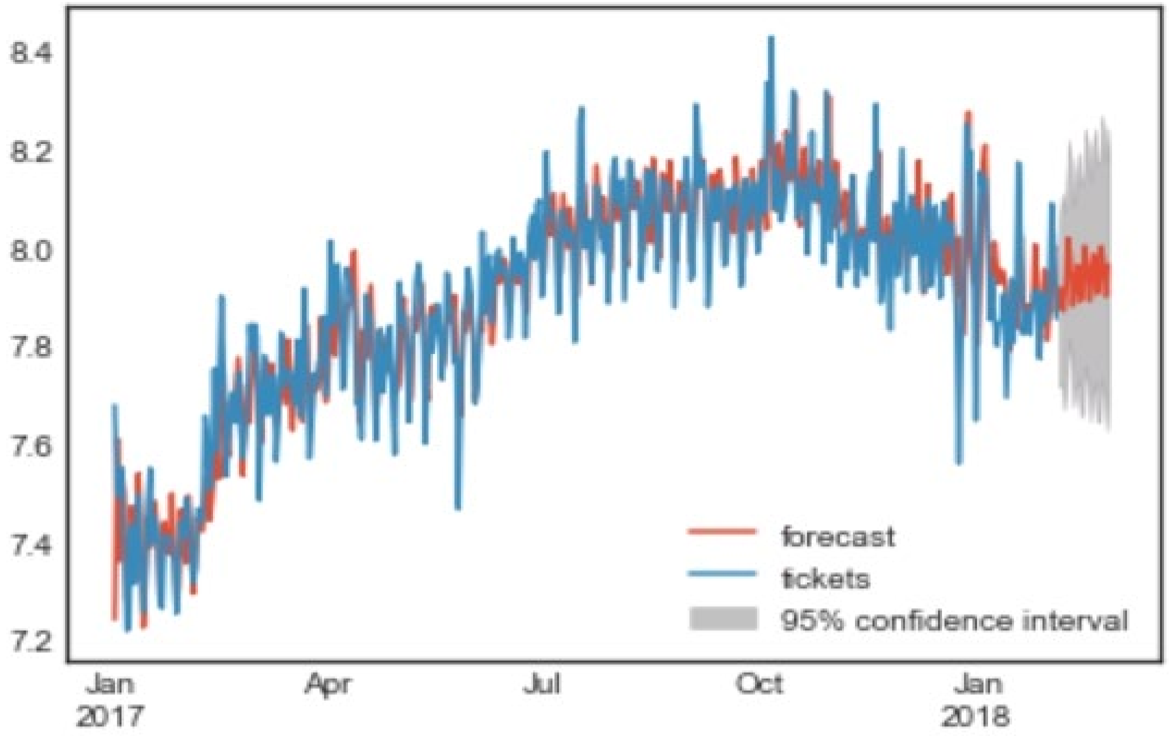
Code Snippet – Time Series Analysis

```
#Dickey-fuller test
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #determine rolling stats
    movavg = timeseries.rolling(30).mean()
    movstd = timeseries.rolling(30).std()

    #plot rolling stats
    orig = plt.plot(timeseries, color = 'c', label = 'Original')
    mean = plt.plot(movavg, color = 'r', label = 'Mean')
    std = plt.plot(movstd, color = 'black', label = 'std')
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Std Dev')
    plt.show(block = False)

    #perform Dickey-fuller test
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries['tickets'], autolag = 'AIC')
    dfout = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lags Used', 'No. of Observations Used'])
    for key, value in dftest[4].items():
        dfout['Critical value (%s)'%key] = value
    print(dfout)
```

```
#ACF and PACF
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(dflogshift, nlags = 20)
lag_pacf = pacf(dflogshift, nlags = 20, method = 'ols')
```

Code Snippet – Time Series Analysis

```
results_AR2.plot_predict(1,420)
x = results_AR2.forecast(steps = 21)
```




```
preds_array = results_AR2.forecast(steps = 21, alpha = 0.05)[0]
```

Code Snippet – Looped Forecast

```
: def blackmamba(cc, train):
      tf = train.loc[train.cc == cc, ['date','tickets']]
      tf.sort_values('date', inplace=True)
      tf.set_index('date', inplace=True)

      df_logscale = np.log1p(tf) #log transformation for stationarity

      dflogshift = df_logscale - df_logscale.shift() #one shift of logvalues for better statiotionarity
      dflogshift.dropna(inplace=True)

      results = arima_tune(df_logscale, dflogshift.tickets) #call timeseries model to get the best model after tuning

      best_pqd = min(results, key = results.get)
      best_rss, best_model = results.get(best_pqd)

      preds_array = best_model.forecast(steps = 22, alpha = 0.05)[0]
      final_preds = np.round(np.expm1(preds_array))

      return final_preds
```

```
: def arima_tune(logscale, logshift):
      results = {}
      for AR in range(0,8):
          for MA in range(0,8):
              model = ARIMA(logscale, order = (AR,1,MA))
              fit_is_available = False
              results_ARIMA = None
              try:
                  results_ARIMA = model.fit(disp = -1, method = 'css')
                  fit_is_available = True
              except:
                  continue
              if fit_is_available:
                  RSS = get_rss(logshift, results_ARIMA.fittedvalues)
                  results['%d-1-%d' % (AR,MA)]=[RSS, results_ARIMA]
      return results
```

Contd…

Code Snippet – Looped Forecast

```
def get_rss(series, fits):
    fits_new = fits
    missing_idx = list(set(series.index).difference(set(fits_new.index)))
    if missing_idx:
        nan_series = pd.Series(index = pd.to_datetime(missing_idx))
        fits_new = fits_new.append(nan_series)
        fits_new.sort_index(inplace = True)
        fits_new.fillna(method = 'bfill', inplace = True)
        fits_new.fillna(method = 'ffill', inplace = True)
    return sum((fits_new - series)**2)
```

```
counter = 207
for cc in cc_major.values:
    print('Forecasting cc: {}'.format(cc))
    print(counter, ' left')
    preds = blackmamba(cc,new_train)

    i = 0
    for d in forecast_dates:
        new_test.loc[((new_test.date == d)&(new_test.cc==cc)), 'tickets'] = preds[i]
        i+=1
    counter -=1
```

Technologies/Tools Involved

jupyter

python

matplotlib

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Thank You