An Unconventional Unsupervised Learning Approaches on New York City Bus Data

by Kefei Mo

Introduction

This is the notebook of my capstone 3 unsupervised learning project.

In this project, I choose New Youk city bus dataset The chosen dataset has 26M rows and 13 columns with no pre-defined target. It captured New Your city bus operation data on 2017/06, 2017/08, 2017/10 and 2017/12.

The purpose of this project is to demonstrate several data analysis, feature engineering and machine learning techniques, especially to explore the concept and application of unsupervised learning. Since the chosen dataset has no explicit labels it fits in the unsupervised learning theme. But instead of using well-known unsupervised learning tools for clustering and/or dimenionality reduction, I try to facilitate unsupervised learning's pattern-finding capability and demonstrate its use to improve supervised learning performance, even though such pattern(s) is hard to identify.

Another focus of this project is to emphasize the importance of feature engineering, i.e. to enhance model interpretability and boosting machine learning.

During working on this project I was reading a great book about data visualization by Claus O. Wilke Fundamentals of Data Visualization. (https://serialmentor.com/dataviz/) I will be trying to apply many principles mentioned in the book.

By the way, I code-folded most of the shells. Unfold them as you wish. Here is the main content of this project.

Table of Contents

In [191]:
```
1 ▸ # load package↔
```

Data Cleaning (Data source: Kaggle (https://www.kaggle.com/stoney71/new-york-city-transport-statistics))

Load data and data info

back to main

In [4]:
```
#  Load data↔
```

In [5]:
```
# data_info↔
```

Out[5]:

g	VehicleRef	VehicleLocation.Latitude	VehicleLocation.Longitude	NextStopPointName	ArrivalProximi
9	NYCT_430	40.635170	-73.960803	FOSTER AV/E 18 ST	appro
6	NYCT_8263	40.590802	-74.158340	MERRYMOUNT ST/TRAVIS AV	appro
4	NYCT_4223	40.886010	-73.912647	HENRY HUDSON PKY E/W 235 ST	
9	NYCT_8422	40.668002	-73.729348	HOOK CREEK BL/SUNRISE HY	< 1 sto
0	NYCT_4710	40.868134	-73.893032	GRAND CONCOURSE/E 196 ST	

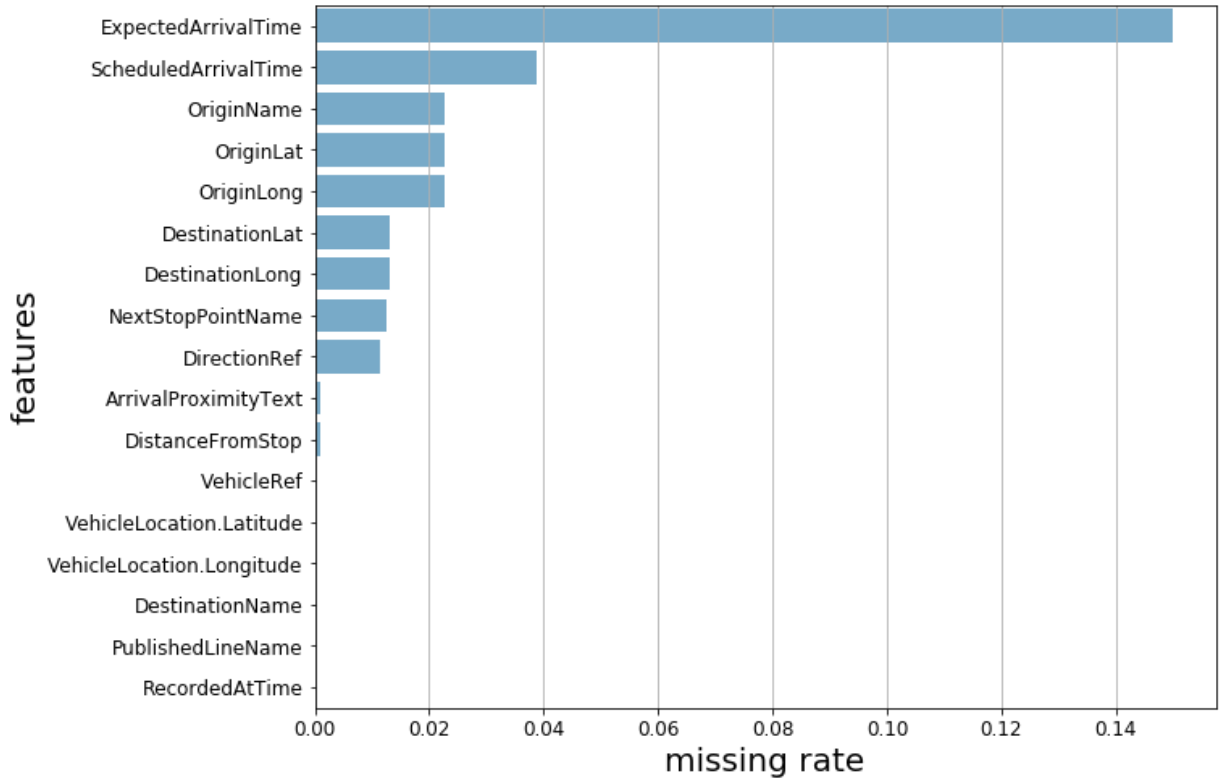
In [6]:
```
1 ▸ # data_info↔
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26520716 entries, 0 to 26520715
Data columns (total 17 columns):
 #   Column                     Dtype  
---  ------                     -----  
 0   RecordedAtTime             object 
 1   DirectionRef               float64
 2   PublishedLineName          object 
 3   OriginName                 object 
 4   OriginLat                  float64
 5   OriginLong                 float64
 6   DestinationName            object 
 7   DestinationLat             float64
 8   DestinationLong            float64
 9   VehicleRef                 object 
 10  VehicleLocation.Latitude   float64
 11  VehicleLocation.Longitude  float64
 12  NextStopPointName          object 
 13  ArrivalProximityText       object 
 14  DistanceFromStop           float64
 15  ExpectedArrivalTime        object 
 16  ScheduledArrivalTime       object 
dtypes: float64(8), object(9)
memory usage: 3.4+ GB
```

Handel missing data: A non-trivial fillna strategy

back to main

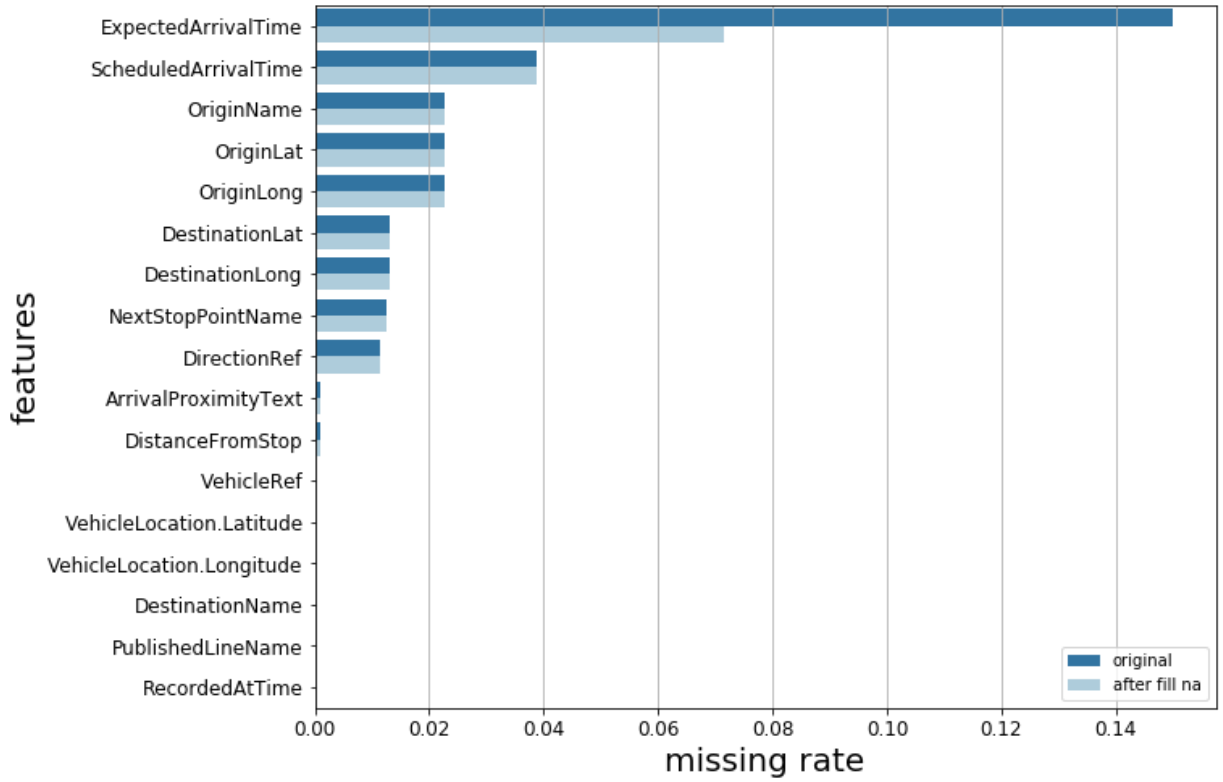
In [7]:
```
# null value percentage↔
```



In [8]:
```
# a simple logic to fill nan
def get_df_fillna():↔
# df_fillna = get_df_fillna()
```

In [9]:
```
df_fillna = get_df_fillna()
```

In [10]:
```
1 ▸ # missing data compare↔
```



Just by a simple fill na logic, we have dramatically saved half of the missing data on the largest missing value feature. And since it is not a brute force imputation fill na method, we can reasonablly feel confident about the data quality.

In [ ]:
```
1 ▸ # detail alart: how I fill na↔
```

Exploratory Data Analysis (EDA)

Brief stories of

- Operation features
- Geometric features
- DateTime features

back to main

EDA

The logic underlying. Let me tell a story about the data.

First of all, let's try to understand the logics behind the data.

The data was captured automatically from the API. We can think about at certain moment (recorded as 'RecordedAtTime') we query data from certain bus (assocaited with the 'VehicleRef' of certain 'PublishedLineName') about its location ('VehicleLocation.Latitude', 'VehicleLocation.Longitude'), what is the next bus stop ('NextStopPointName') and when it is expected to arrive the next stop ('ExpectedArrivalTime').

The feature names should be very self-explained. And I would like to seperate the features into three groups based on their names, namely, operation info, time info, spatial info.

Operation info

- PublishedLineName (static)
- DirectionRef (static)
- DestinationName (static)
- NextStopPointName (static/dynamic)
- VehicleRef (static)

Time info

- RecordedAtTime (dynamic)
- ExpectedArrivalTime (dynamic)
- ScheduledArrivalTime (static)

Spatial info

gps coordinates

- OriginLat (static)
- OriginLong (static)
- DestinationLat (static)
- DestinationLong (static)
- VehicleLocation.Latitude (dynamic)
- VehicleLocation.Longitude (dynamic)

relative location

- ArrivalProximityText (dynamic)
- DistanceFromStop (dynamic)

As you can see, I also consider the features as either dynamic or static, i.e. 'PublishedLineName' like "b6" indicate the name of the service bus line, and it is static because the name of the bus line would not change. On the other hand, 'VehicleLocation.Latitude' feature is a dynamic variable

since the location of individual bus is changing all the time when runnig. You might wonder why I labled 'NextStopPointName' as both static and dynamic. The reason behind is, for certain service line (i.e. b6) the bus stop would not move locations or change names over the period we are studying. But for individual bus that is running, the next bus stop is changing all the time. While the data is captured from individual bus, I would like to put a reminder here, we might be able to extract bus stop GPS coordinates, which is not given by the original featues.

Another thing we should pay attention is the hierarchy of the underlying logics. i.e. individual service line might operate several buses, but individual bus is not likely to run at several different service lines. That being said, 'VehicleRef' should be underneath 'PublishedLineName' but not the other way around. To justify this claim, you can simply check the number of unique 'PublishedLineName' values under single element under 'VehicleRef' features.

In [ ]:
```
1 ▸ # detail hidden here:↔
```

Operation features

back to main

recall the followings are our operation info

- PublishedLineName (static)
- DirectionRef (static)
- DestinationName (static)
- NextStopPointName (static/dynamic)
- VehicleRef (static)

All of them are categorical variables and as mentioned before, they are nested, it might be ideal if we can visualize their relationship using, say, sankey diagram. But sankey is not very straightforward to plot in Python, so let's dig in one layer at a time, starting from PublishedLineName.

In [243]:
```
# take a look at all the line services
df_eda.PublishedLineName.unique()
```

Out[243]:
```
array(['B8', 'S61', 'Bx10', 'Q5', 'Bx1', 'M1', 'B31', 'B83', 'B82', 'S59',
       'Bx28', 'B1', 'B26', 'Bx39', 'M66', 'Bx31', 'Bx36', 'M96', 'Q4',
       'Q54', 'B6', 'B4', 'M101', 'Bx5', 'Q12', 'B43', 'M100', 'M11',
       'B11', 'M2', 'B41', 'M34A-SBS', 'Bx32', 'X10', 'B14', 'B62',
       'Bx17', 'Bx4', 'M103', 'M50', 'B25', 'Bx19', 'Q15A', 'Q44-SBS',
       'Bx9', 'S78', 'Q48', 'X17', 'Bx11', 'B2', 'B44', 'Bx2', 'B35',
       'B12', 'B38', 'B3', 'B15', 'M10', 'B17', 'M60-SBS', 'Bx7', 'B67',
       'M55', 'B46', 'Bx6', 'M102', 'M104', 'Q1', 'Bx35', 'S53',
       'M23-SBS', 'B24', 'Bx22', 'Bx12', 'Q20B', 'B45', 'S48', 'Bx30',
       'Q20A', 'B9', 'B63', 'Q30', 'Bx40', 'Q84', 'M15', 'Q17', 'M22',
       'Q59', 'B49', 'Bx33', 'Bx42', 'B57', 'Q83', 'Q32', 'B65', 'Q3',
       'Bx13', 'M7', 'B47', 'B52', 'S66', 'B20', 'X1', 'Bx21', 'B7',
       'Q55', 'Q31', 'Q13', 'M5', 'S74', 'M14D', 'M79-SBS', 'M3', 'S46',
       'M31', 'B70', 'Bx41', 'M8', 'S52', 'M42', 'Q28', 'S76', 'B61',
       'S40', 'Bx3', 'B60', 'Q58', 'S42', 'Q27', 'S44', 'B42', 'X27',
       'Bx27', 'B64', 'Bx15', 'Q43', 'M72', 'B16', 'M20', 'S51', 'B68',
       'Q46', 'M35', 'M4', 'Q2', 'B36', 'B54', 'S62', 'B37', 'M14A',
       'B48', 'Q24', 'Q36', 'M86-SBS', 'S57', 'M57', 'Q16', 'X28', 'Q85',
       'Q88', 'Bx34', 'M34-SBS', 'S79-SBS', 'Bx16', 'B13', 'Bx24', 'Q56',
       'M116', 'Bx46', 'Bx29', 'Q15', 'X17J', 'X7', 'X22', 'S89',
       'M15-SBS', 'X12', 'X2', 'Bx12-SBS', 'Q42', 'X5', 'X15', 'Q76',
       'B74', 'B44-SBS', 'X9', 'X14', 'X8', 'X30', 'X31', 'X19',
       'B46-SBS', 'B84', 'Bx8', 'Bx4A', 'X21', 'Bx26', 'Bx41-SBS', 'S54',
       'Q26', 'B69', 'Bx38', 'S96', 'Q77', 'S93', 'X63', 'S55', 'Bx18',
       'M98', 'S94', 'X22A', 'S92', 'S91', 'S98', 'X38', 'X4', 'S56',
       'S90', 'X68', 'X37', 'X11', 'X64', 'X17A', 'X10B', 'X42', 'X3',
       'M106', 'M9', 'Bx20', 'B32', 'M21', 'B39', 'M12', 'Q8', 'Q113',
       'Q33', 'Q22', 'BxM2', 'Q70-SBS', 'Q111', 'Q9', 'Q50', 'Q66',
       'BxM6', 'QM5', 'Q37', 'Q40', 'BxM1', 'Q72', 'Q10', 'Q39', 'Q49',
       'Q34', 'Q65', 'QM15', 'Q53', 'Q52', 'Q19', 'Q69', 'Q7', 'Q25',
       'BxM7', 'Q110', 'Q100', 'BxM9', 'Q67', 'Q103', 'BxM11', 'QM2',
       'Q60', 'B103', 'Q114', 'Q6', 'BxM10', 'Q29', 'Q35', 'BxM8', 'BM4',
       'Q102', 'QM6', 'BM3', 'BxM3', 'Q23', 'Bx23', 'Q11', 'Q41', 'B100',
       'Q18', 'Q112', 'BM1', 'Q101', 'BxM4', 'Q64', 'Q38', 'Q104', 'Q21',
       'QM20', 'BM5', 'QM7', 'Q47', 'BM2', 'QM4', 'S84', 'S86', 'S81',
       'QM12', 'BxM18', 'QM32', 'QM40', 'QM31', 'QM17', 'QM44', 'QM1',
       'QM35', 'QM36', 'QM8', 'QM16', 'QM10', 'QM25', 'QM3', 'QM11',
       'QM34', 'QM24', 'QM21', 'QM18', 'QM42', 'Shuttle-M2', 'Shuttle-M3',
       'Shuttle-M1', 'Shuttle-M', 'Bx6-SBS', 'M Shuttle Bus', 'Q52-SBS',
       'Q53-SBS'], dtype=object)
```

In [12]:
```
# df_eda = df_original.copy()
```

In [13]:
```
1 ▸ # group the lines, ↔
```

```
Bronx:  ['Bx10' 'Bx1' 'Bx28' 'Bx39' 'Bx31' 'Bx36' 'Bx5' 'Bx32' 'Bx17' 'Bx4' 'Bx
19'
 'Bx9' 'Bx11' 'Bx2' 'Bx7' 'Bx6' 'Bx35' 'Bx22' 'Bx12' 'Bx30' 'Bx40' 'Bx33'
 'Bx42' 'Bx13' 'Bx21' 'Bx41' 'Bx3' 'Bx27' 'Bx15' 'Bx34' 'Bx16' 'Bx24'
 'Bx46' 'Bx29' 'Bx12-SBS' 'Bx8' 'Bx4A' 'Bx26' 'Bx41-SBS' 'Bx38' 'Bx18'
 'Bx20' 'BxM2' 'BxM6' 'BxM1' 'BxM7' 'BxM9' 'BxM11' 'BxM10' 'BxM8' 'BxM3'
 'Bx23' 'BxM4' 'BxM18' 'Bx6-SBS']
Brooklyn:  ['B8' 'B31' 'B83' 'B82' 'B1' 'B26' 'B6' 'B4' 'B43' 'B11' 'B41' 'B14'
'B62'
 'B25' 'B2' 'B44' 'B35' 'B12' 'B38' 'B3' 'B15' 'B17' 'B67' 'B46' 'B24'
 'B45' 'B9' 'B63' 'B49' 'B57' 'B65' 'B47' 'B52' 'B20' 'B7' 'B70' 'B61'
 'B60' 'B42' 'B64' 'B16' 'B68' 'B36' 'B54' 'B37' 'B48' 'B13' 'B74'
 'B44-SBS' 'B46-SBS' 'B84' 'B69' 'B32' 'B39' 'B103' 'BM4' 'BM3' 'B100'
 'BM1' 'BM5' 'BM2']
Manhattan:  ['M1' 'M66' 'M96' 'M101' 'M100' 'M11' 'M2' 'M34A-SBS' 'M103' 'M50'
'M10'
 'M60-SBS' 'M55' 'M102' 'M104' 'M23-SBS' 'M15' 'M22' 'M7' 'M5' 'M14D'
 'M79-SBS' 'M3' 'M31' 'M8' 'M42' 'M72' 'M20' 'M35' 'M4' 'M14A' 'M86-SBS'
 'M57' 'M34-SBS' 'M116' 'M15-SBS' 'M98' 'M106' 'M9' 'M21' 'M12'
 'Shuttle-M2' 'Shuttle-M3' 'Shuttle-M1' 'Shuttle-M' 'M Shuttle Bus']
Queens:  ['Q5' 'Q4' 'Q54' 'Q12' 'Q15A' 'Q44-SBS' 'Q48' 'Q1' 'Q20B' 'Q20A' 'Q30'
 'Q84' 'Q17' 'Q59' 'Q83' 'Q32' 'Q3' 'Q55' 'Q31' 'Q13' 'Q28' 'Q58' 'Q27'
 'Q43' 'Q46' 'Q2' 'Q24' 'Q36' 'Q16' 'Q85' 'Q88' 'Q56' 'Q15' 'Q42' 'Q76'
 'Q26' 'Q77' 'Q8' 'Q113' 'Q33' 'Q22' 'Q70-SBS' 'Q111' 'Q9' 'Q50' 'Q66'
 'QM5' 'Q37' 'Q40' 'Q72' 'Q10' 'Q39' 'Q49' 'Q34' 'Q65' 'QM15' 'Q53' 'Q52'
 'Q19' 'Q69' 'Q7' 'Q25' 'Q110' 'Q100' 'Q67' 'Q103' 'QM2' 'Q60' 'Q114' 'Q6'
 'Q29' 'Q35' 'Q102' 'QM6' 'Q23' 'Q11' 'Q41' 'Q18' 'Q112' 'Q101' 'Q64'
 'Q38' 'Q104' 'Q21' 'QM20' 'QM7' 'Q47' 'QM4' 'QM12' 'QM32' 'QM40' 'QM31'
 'QM17' 'QM44' 'QM1' 'QM35' 'QM36' 'QM8' 'QM16' 'QM10' 'QM25' 'QM3' 'QM11'
 'QM34' 'QM24' 'QM21' 'QM18' 'QM42' 'Q52-SBS' 'Q53-SBS']
Staten Island:  ['S61' 'S59' 'S78' 'S53' 'S48' 'S66' 'S74' 'S46' 'S52' 'S76' 'S
40' 'S42'
 'S44' 'S51' 'S62' 'S57' 'S79-SBS' 'S89' 'S54' 'S96' 'S93' 'S55' 'S94'
 'S92' 'S91' 'S98' 'S56' 'S90' 'S84' 'S86' 'S81']
others: ['X1' 'X10' 'X10B' 'X11' 'X12' 'X14' 'X15' 'X17' 'X17A' 'X17J' 'X19' 'X
2'
 'X21' 'X22' 'X22A' 'X27' 'X28' 'X3' 'X30' 'X31' 'X37' 'X38' 'X4' 'X42'
 'X5' 'X63' 'X64' 'X68' 'X7' 'X8' 'X9']
```

We found a hidden layer

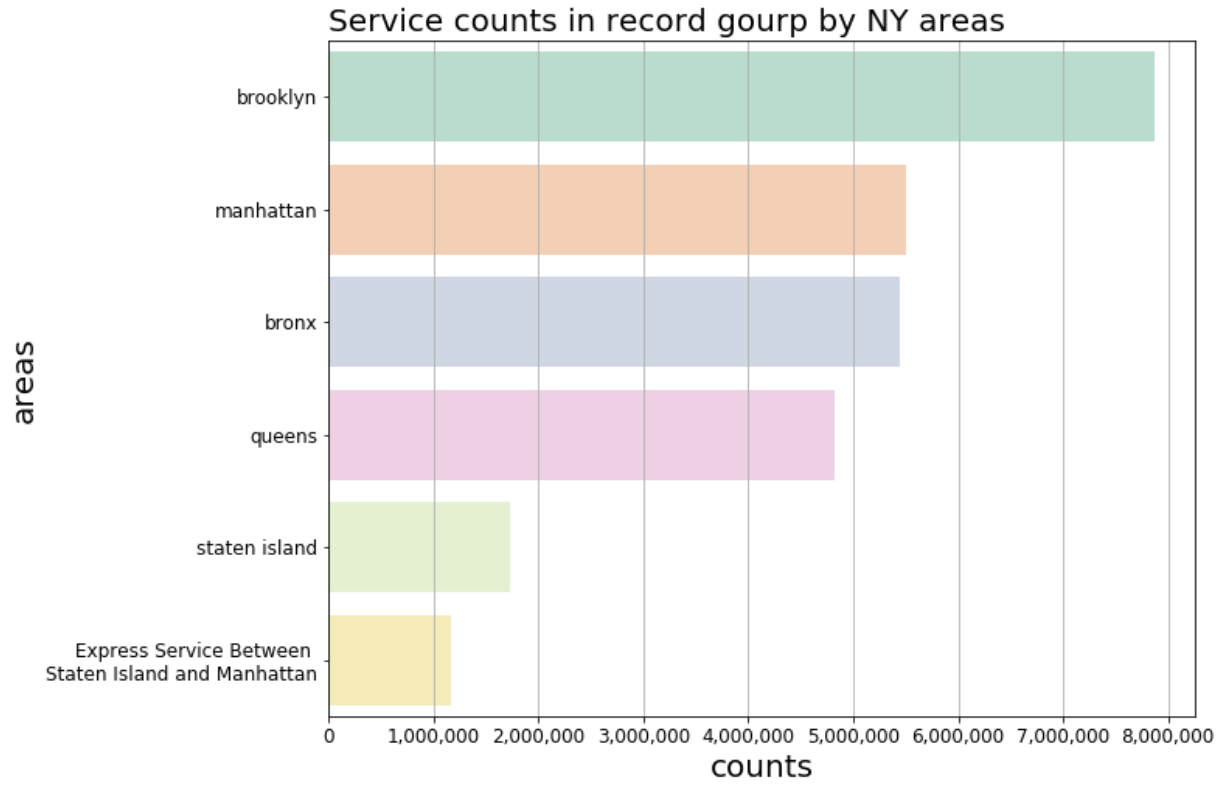
As you can see, we found a hidden layer of the dataset--areas.

Even though there should be trasnportation service between areas, i.e. "Express Service Between Staten Island and Manhattan". Most of the bus service is limited to local area (or brough). This is important because the operation pattern for, say, Mahattan can be very different from that of Staten Island. That's the main reason we would like to first group the bus service lines according to the area.

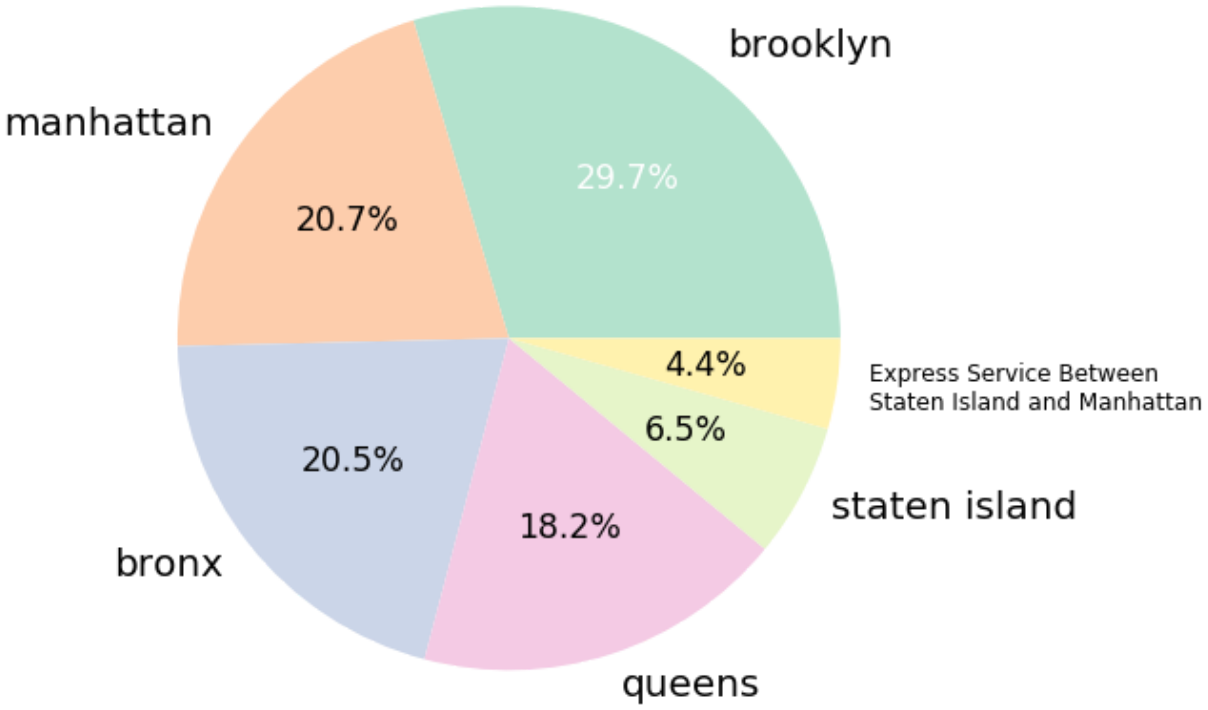
The goal of such grouping is to help finding the underlying pattern. And normally this kind of feature engineering is very efficient and work better than using machine learning clustering methods.

In [14]:
```
# A brief stats of line counts among bronx, brooklyn, manhattan, etc.↔
```

In [15]:
```
# plot counts stats by areas↔
```



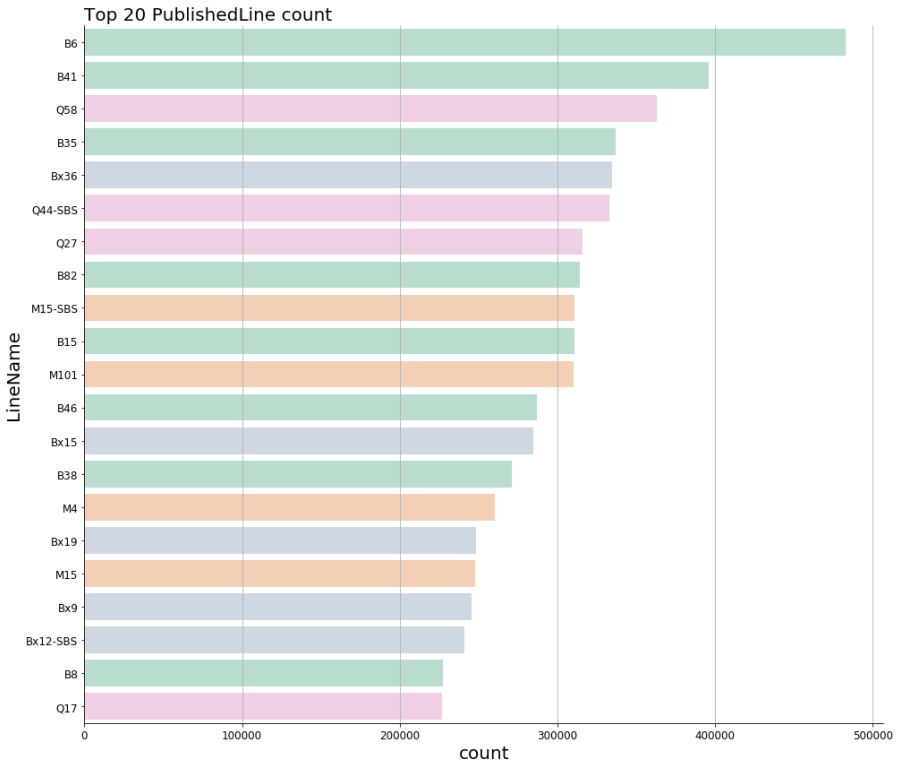
In [16]:
```
1 ▸ # take another look↔
```



In [17]:
```
1 ▾ # df_line_count.head(10)
```

In [18]:
```
1 ▸ # df_line_count_by_areas ↔
```

In [192]:
```
1 ▸ # visulize PublishedLineName↔
```



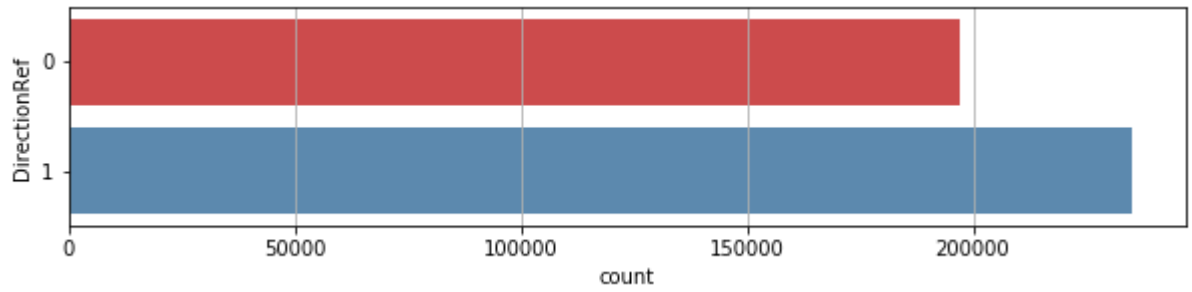
Decision time.

Let's focus on the 'B6' line, so that we would not be overwhelmed by the data, also 'B6' has the most record counts, so the analysis would not be unrepresentative.

In [20]:
```
1 ▾ # customer function: get target sub-dataset
2 ▸ def reset_dataset():↔
17  df_trim = reset_dataset()
```

In [193]:
```
1 ▾ # df_trim.groupby(['DirectionRef', ])[['OriginName']].count().reset_index(
```

In [223]:
```
1 ▸ # visualize counts by direction_ref↔
```



From the anaysis above, it seems DirectionRef=1 has more operations according to the records. Let's dive in even more.

note*: from now on, I will designate color code 'red' for DirectionRef=0, and 'blue' for DirectionRef=1

In [23]:
```
# nested info ↔
```

Out[23]:

			count
DirectionRef	**OriginName**	**DestinationName**	
1	**ROCKAWAY STATION/ROCKAWAY STATION**	**BENSONHURST HARWAY AV**	58519
		AVENUE J CONEY IS AV	8836
		LTD BENSONHURST HARWAY AV	4814
	LIVONIA AV/ASHFORD ST	**LTD BENSONHURST HARWAY AV**	124709
		BENSONHURST HARWAY AV	34356
	FLATLANDS AV/ELTON ST	**BENSONHURST HARWAY AV**	764
	E 45 ST\GLENWOOD RD	**BENSONHURST HARWAY AV**	282
	BEDFORD AV/CAMPUS ROAD	**BENSONHURST HARWAY AV**	2765
	BAY PY/60 ST	**BENSONHURST HARWAY AV**	112
0	**HARWAY AV/BAY 37 ST**	**LTD EAST NY NEW LOTS STA**	105715
		ROCK PKY STA	52684
		EAST NY NEW LOTS STA	27872
	FLATLANDS AV/E 82 ST	**EAST NY NEW LOTS STA**	138
		ROCK PKY STA	98
	AV J/CONEY ISLAND AV	**ROCK PKY STA**	10501

From the nested table group by 1.'DirectionRef', 2.'OriginName', 3.'DestinationName', we now know that there are 15 routes running on 'b6' line, with 9 routes running on directionRef=1, 6 routes running on directionRef=0 seperately.

Also, for directionRef=1, the rounts are from 6 different origins to 3 different destinations. For directionRef=0, the rounts are from 3 different origins to 3 different destinations.

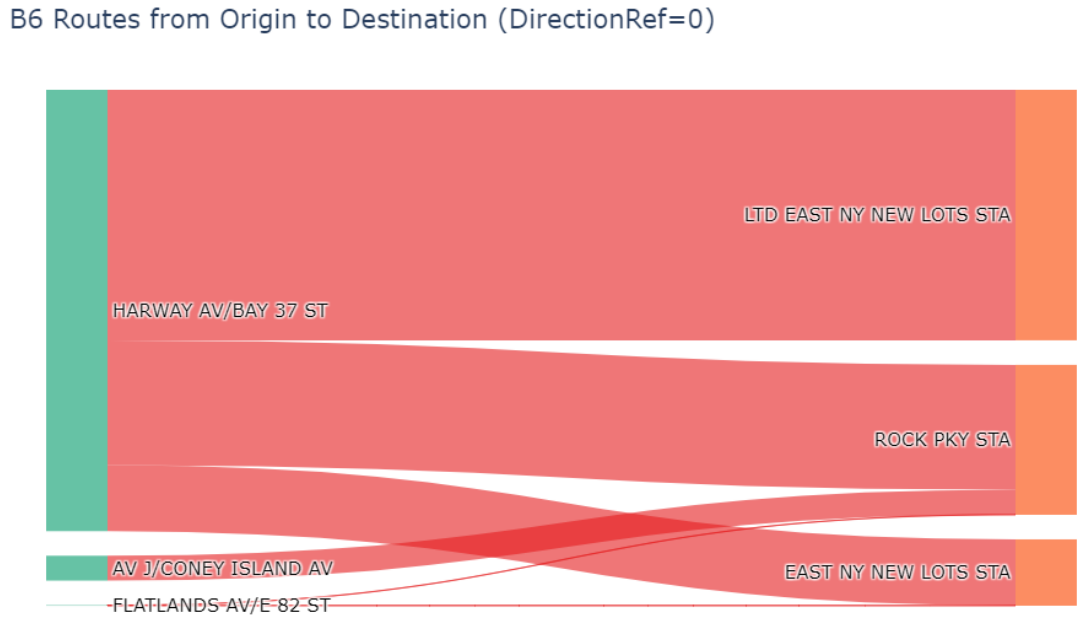
We might also realize that the busy routs are 'LIVONIA AV/ASHFORD ST' to 'LTD BENSONHURST HARWAY AV', 'HARWAY AV/BAY 37 ST' to 'LTD EAST NY NEW LOTS STA', etc.

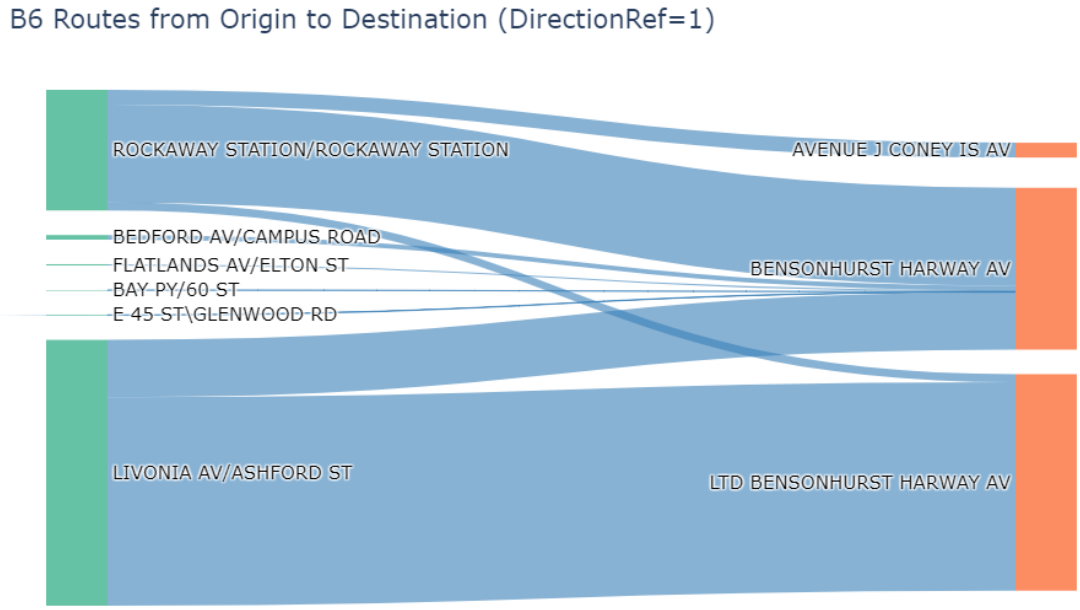
The nested relationships might at best visualize using sankey diagram, here is an example.

In [24]:
```
# prepare data for sankey diagram↔
```

In [224]:
```
# sankey diagram visulize origin and destination (enable that so you can pl
```

In [225]:
```
1 ▸ # static figure. just in case you cannot load the interactive plot, ↔
```





Now the operation information of each rount is much clearer. From the sankey diagram we can

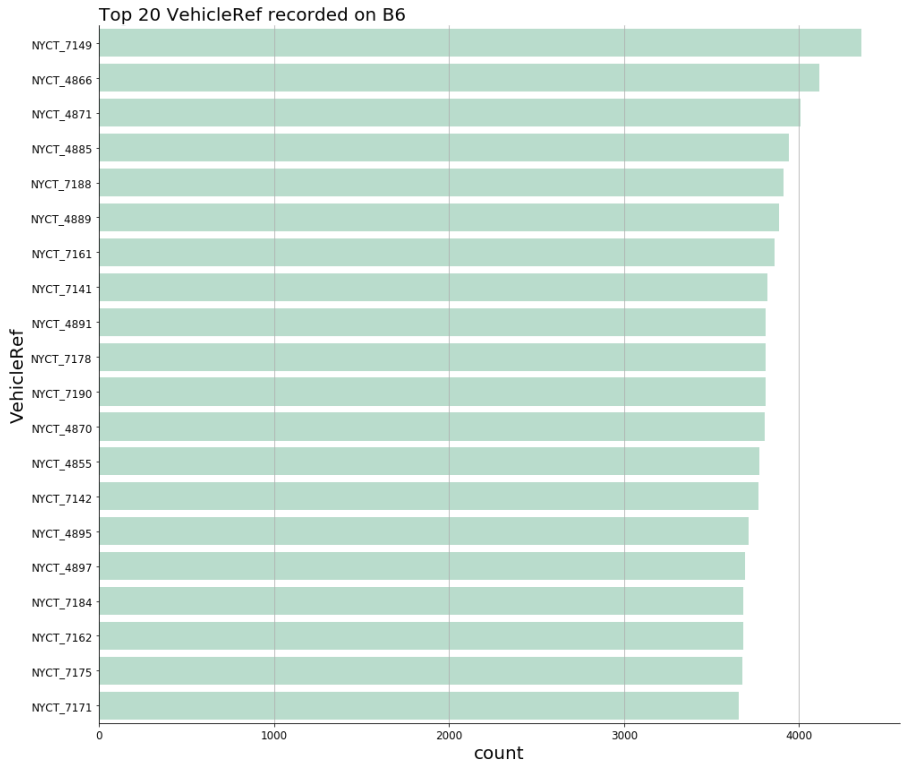
have a better sense which route(s) has/have more records (indicating such route(s) are operated more frequently).

But for the better part, we can easily spot which origin to which destination(s) and vice versa.

(A side note about visualization here, ask me about it you are interested.)

In [27]:
```
# analyze VehicleRef↔
```

In [28]:
```
# visulize VehicleRef↔
```



Short summary here

Here is a brief analysis on the operation info, 'PublishedLineName', 'DirectionRef', 'DestinationName', 'NextStopPointName', and 'VehicleRef'. Since all of them are categorical variables, I just analyze and visualize the counts of these features. Nothing very exciting so far, let's move on and take a look at the geometric data.

In [ ]:
```
### details alert ↔
```

Geometric features

back to main

Things start to become more interesting when combining geometric data

recall the geo data ara

gps coordinates

- OriginLat (static)
- OriginLong (static)
- DestinationLat (static)
- DestinationLong (static)
- VehicleLocation.Latitude (dynamic)
- VehicleLocation.Longitude (dynamic)

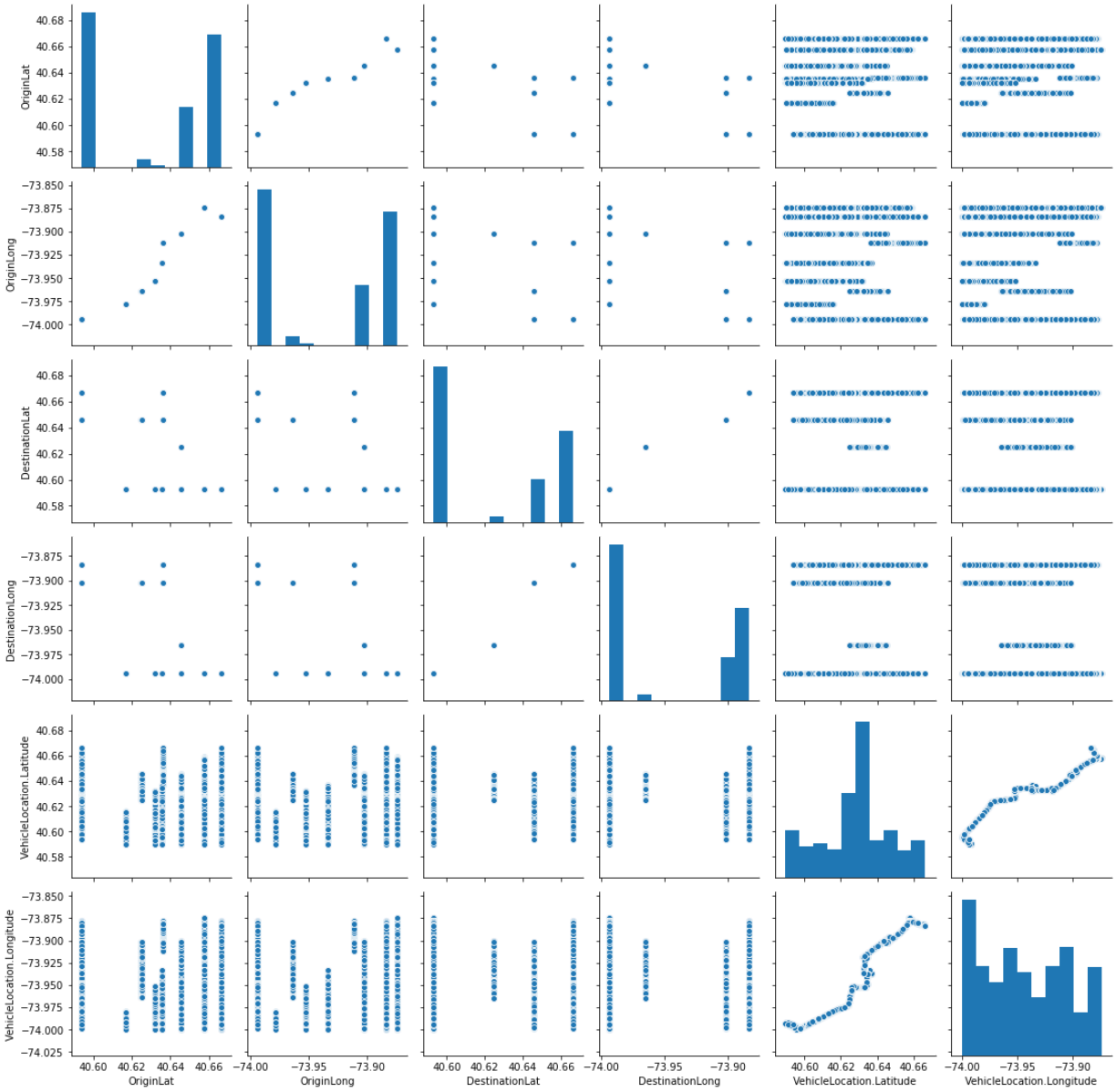
relative location

- ArrivalProximityText (dynamic)
- DistanceFromStop (dynamic)

```
Finally, we have some numerical variables to play with, hooray :)


```

In [29]:
```
1 sns.pairplot(data=df_trim, vars=['OriginLat', 'OriginLong', 'DestinationLa
```

Out[29]: <seaborn.axisgrid.PairGrid at 0x28fa218e088>



Many times, it is very convenient to use sns.pairplot for EDA, since it can show both univariant distribution and co-relationship. But for our GPS data, it doesn't make much sense to point out high-correlationship like the one between 'VehicleLocation.Latitude' and 'VehicleLocation.Longitude'. Also GPS coordinates are more likely to be useful when they are paired to each other, that being said, we probabely do not have to pay that much attention to the univariant distribution either.

So how to use such GPS data, probabely it is easier for me to just show you. As a reminder, don't forget to use the hierachical relationship about operation info.

Extract the bus stop coordinate. Feature engeering again. I told you it is important.

In [30]:
```
1 ▸ # this one to check legit (), by rule, the origin and desination GPS should
```

In [197]:
```
1 ▸ # legit check cont'↔
```

In [244]:
```
# use this intimidating pivot table to route info↔
```

Out[244]:

	DirectionRef	OriginName	DestinationName	OriginLat	OriginLong	DestinationLat	DestinationLo
4	0	HARWAY AV/BAY 37 ST	LTD EAST NY NEW LOTS STA	40.593510	-73.993996	40.666420	-73.883
5	0	HARWAY AV/BAY 37 ST	ROCK PKY STA	40.593510	-73.993996	40.645775	-73.902
3	0	HARWAY AV/BAY 37 ST	EAST NY NEW LOTS STA	40.593510	-73.993996	40.666420	-73.883
0	0	AV J/CONEY ISLAND AV	ROCK PKY STA	40.624829	-73.964228	40.645775	-73.902
1	0	FLATLANDS AV/E 82 ST	EAST NY NEW LOTS STA	40.636307	-73.911844	40.666420	-73.883

In [33]:
```
# feature engineering add next stop lat and long↔
```

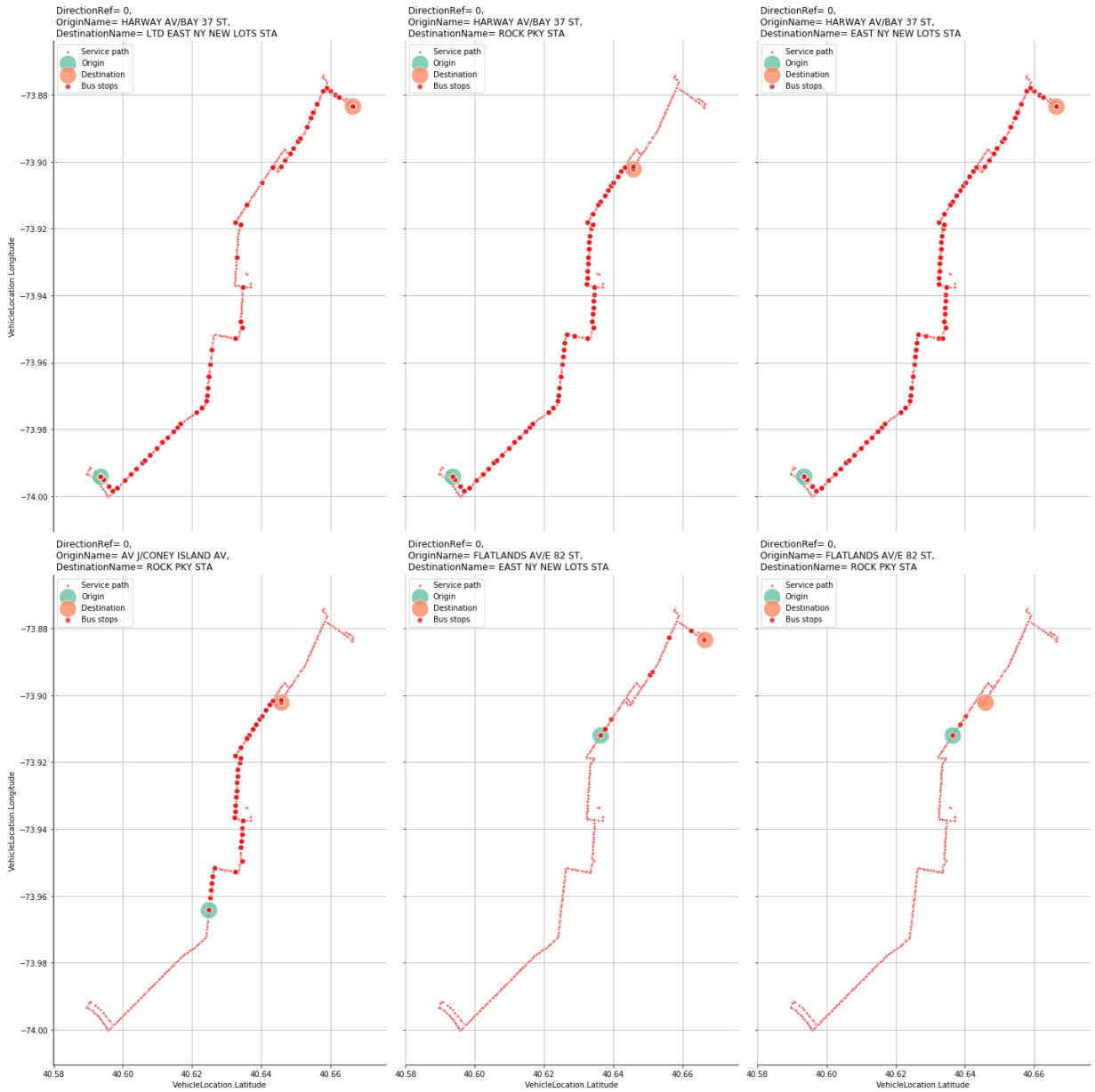
Out[33]:

	RecordedAtTime	DirectionRef	PublishedLineName	OriginName	OriginLat	OriginLong	Dest
20	2017-06-01 00:03:41	0	B6	HARWAY AV/BAY 37 ST	40.593510	-73.993996	E
91	2017-06-01 00:03:42	1	B6	LIVONIA AV/ASHFORD ST	40.666382	-73.883614	BEI
191	2017-06-01 00:03:27	1	B6	LIVONIA AV/ASHFORD ST	40.666382	-73.883614	BEI
221	2017-06-01 00:03:25	0	B6	HARWAY AV/BAY 37 ST	40.593510	-73.993996	E
294	2017-06-01 00:03:49	1	B6	LIVONIA AV/ASHFORD ST	40.666382	-73.883614	BEI

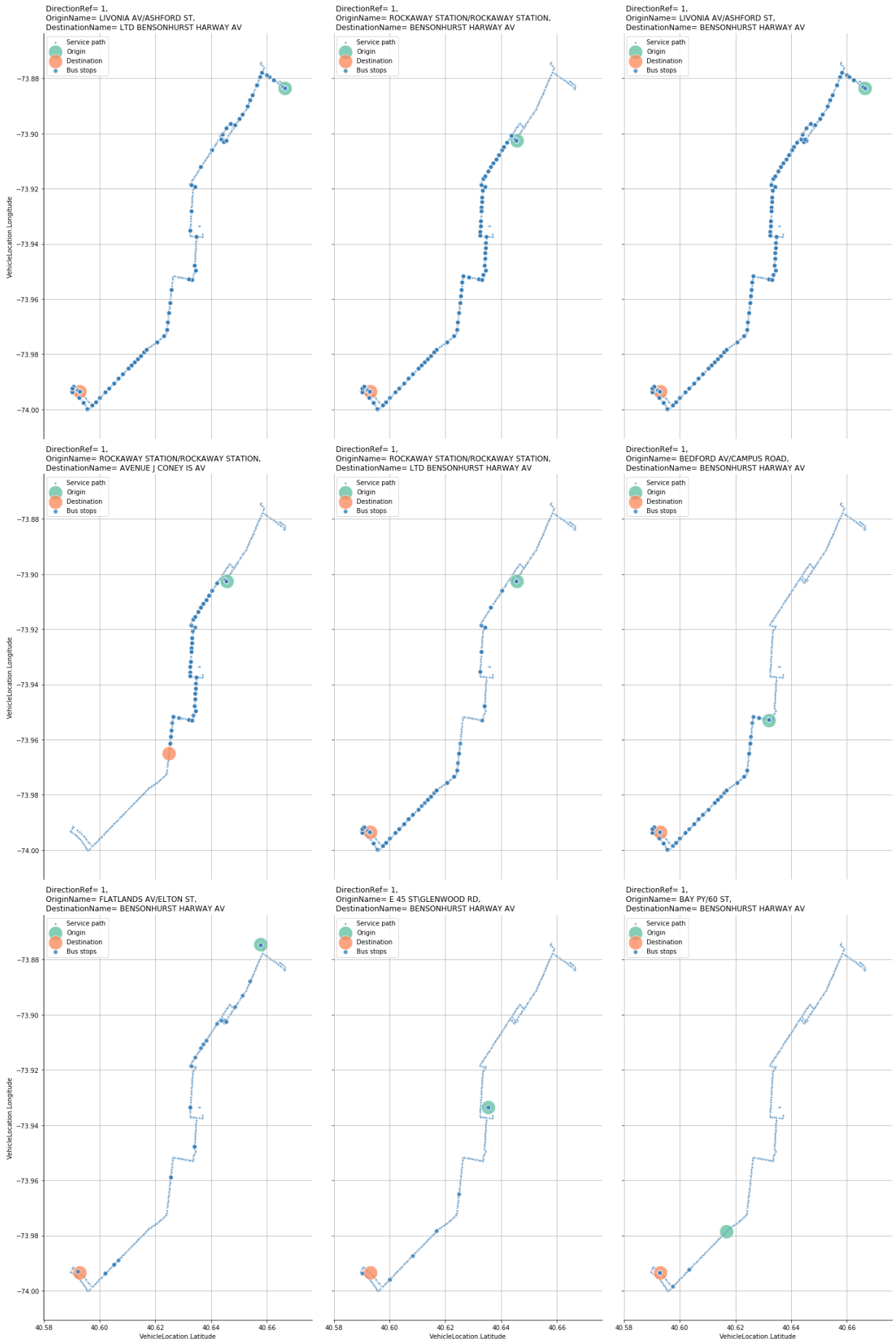
In [34]:
```
### details alert ↔
```

In [35]:
```
# customer function use to plot bus route, use:↔
```

In [36]:
```
1 ▸ # customer function use to plot bus route,↔
```



In [37]:
```
1 ▸ # customer function use to plot bus route,↔
```



Wake up

If the former analysis is too boring I hope this plot can wake you up, because it tells a more detailed story about the B6 service line.

Recall that there 15 different routes running on B6 service line, the one you are looking right now is from 'HARWAY AV/BAY 37 ST' to 'LTD EAST NY NEW LOTS STA' and its DirectionRef is '0'.

The GPS coordinates show the route path. (Note0*). Also we know the origin and destination of this route and we can almost assure that 'DirectionRef==0' indicates that the buses operate from down left to upper right or from South West to North East (remember that the buses are in NY).

The red dots display the bus stops on this route (Note1*). You might wonder how I achieve this information from the data, since the raw data doesn't apply bus stop GPS features. Again, if you are interested I can talk about how I dig out such information in more details. Just a hint, I performed feature engineering used all the geo info data especially 'DistanceFromStop'. This is once again a reminder that feature engineering is so powerful and essential in the machine learning pipeline.

Now it makes more sense now why some routes have more records than the others. Because they have to operate at longer system and/or have to operate at more stops. Also it would bring our attention to details which might otherwise ignore. i.e. route from 'HARWAY AV/BAY 37 ST' to 'LTD EAST NY NEW LOTS STA' and route from 'HARWAY AV/BAY 37 ST' to 'EAST NY NEW LOTS STA' have exactely the same origin and destination. But we cannot rename 'LTD EAST NY NEW LOTS STA' to 'EAST NY NEW LOTS STA' or vice versa, because it would ignore the fact that the former route has less stops than the latter's. Which might as well to make us realize that the name 'LTD' indicates somthing like 'limited transportaion district' or something else that sounds more meaningful to you.

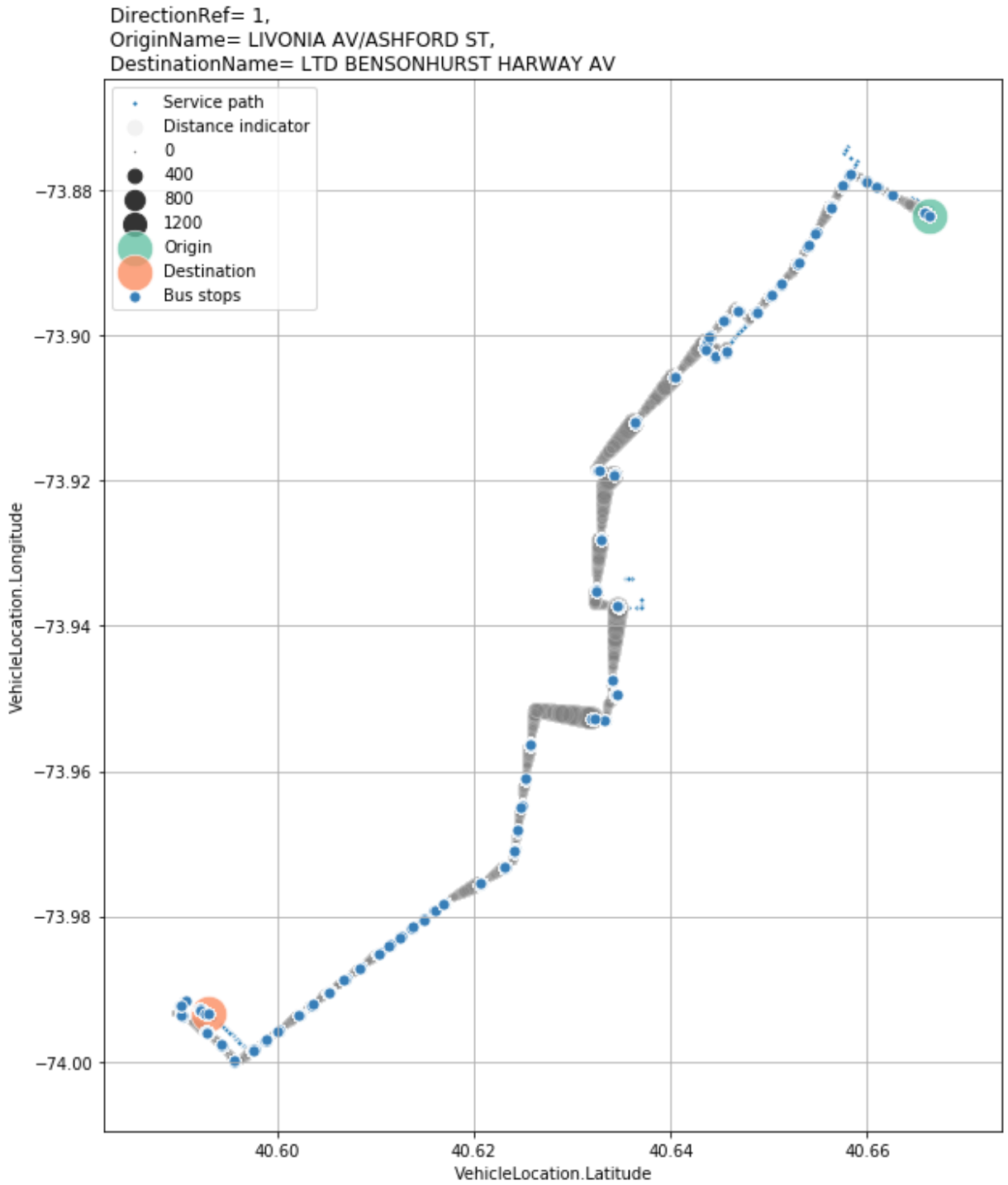
Note0*: I plot GPS for buses with both 'DirectionRef==0' and 'DirectionRef==1', and distinguished by red and blue color.

Note1*: when I said 'this route' I mean one of the 15 routes runing on B6, i.e. from 'HARWAY AV/BAY 37 ST' to 'LTD EAST NY NEW LOTS STA'.

In [ ]:
```
1 ▸ # A note for myself↔
```

There is more story to tell

In [38]:
```
1 ▸ # pick a route to an analyze, i.e. route_index = 6, 11, 9, ↔
```



A detour: talk about measurement quality

One thing should draw our attention when dealing with spatial data like GPS coordinates is the measurement precision and trueness(wikipedia) (https://en.wikipedia.org/wiki/Accuracy_and_precision). For our cases the desimal at 1e-6, associated with 10 cm (4 inches), this is denfinetely sufficient for bus operation study.

Though it is non-trivial to judge the trueness of the measuremnt by a simple glance, one way to do that is by evaluating the consistency. From our data, the orgin and/or destination GPS has 1e-6 difference but not constant. (It would raise a red flag if measuremnt has 0-variance since no measuremnt in the real world is constant.)

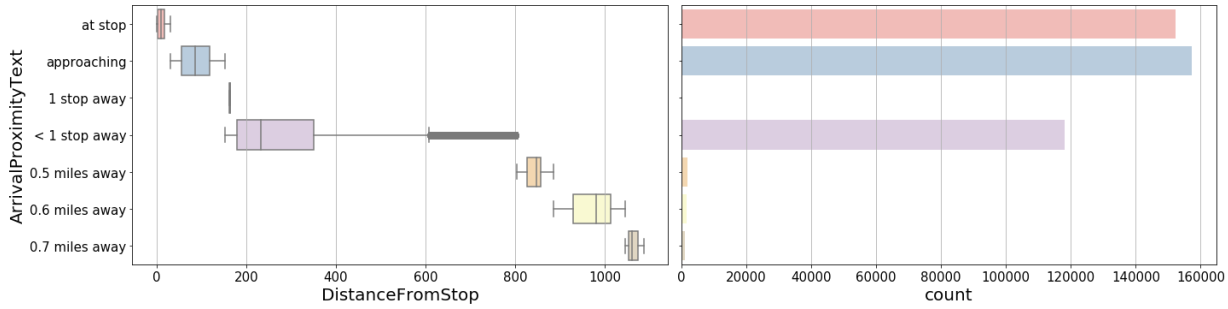
Some data is directely measurement like GPS coordinates but some are from derivation, i.e. 'DistanceFromStop', normally such data can be derive from GPS coordinates, but this is not trivial. i.e. It would not be precise if calculate the distance between two points on the bus route without considering the local geometry, like streets. The following plot is an example to use visualization to check the quality of such data. Note that I use marker size to indicate the value of 'DistanceFromStop' and immediately you can see how the marker size shrink when the bus is approching the bus stops. You can tell the derivated distance data is good since the change of the marker size is smooth. And you can tell the distance calulation algorithm has considered the actual local enviroment instead of measuring a straight line bewteen two points.

Also, even though we can pretty much guess the meaning of the variables by their names, but don't take it for granted. Whoever named the variable might have a different idea as what the name means in your head. i.e. when it says 'at stop', what does 'NextStopPointName' indicate? Is it the stop right now or the one it is going to? So, always check with several features and try to prove that.

A small challenge. It didn't say what unit is used for DistanceFromStop. What would be your approach to find out what unit is used? meter? feet?

Relative location data analysis

In [139]:
```
1 ▸ # relation of features↔
```



Immediately we can see that 'DistanceFromStop' is highly correlated with 'ArrivalProximityText', if not somewhat deterministic.

Also you can see that a large proportion of recorded buses 'ArrivalProximityText' are 'approaching' or 'at stop', which indicates that the distance between bus stops are relatively short.

Also it is not very clear what's difference between '1 stop away' and '<1 stop away' based on the name.

DateTime features

back to main

Let's move on and analyze the time info

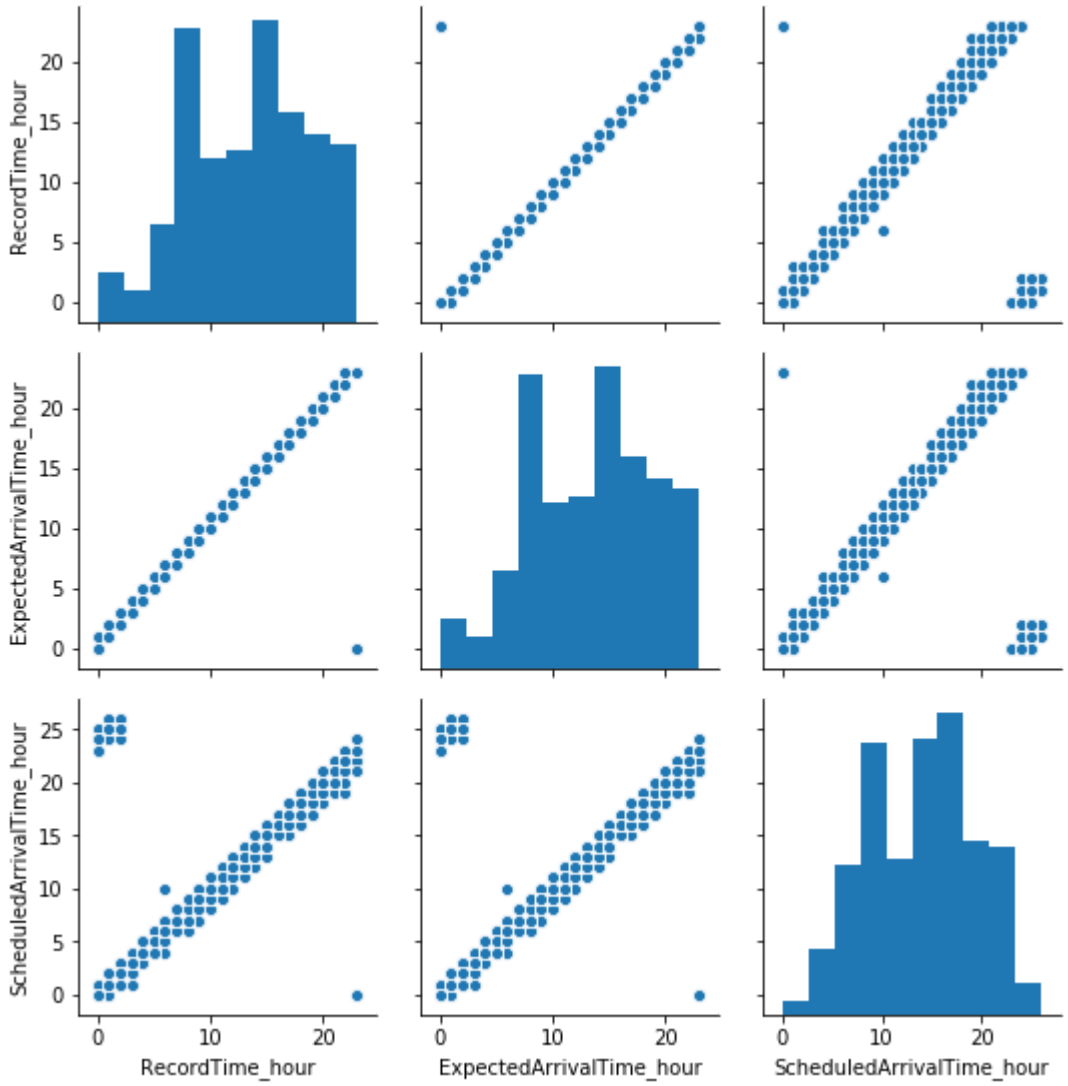
- To help our analysis, it is convenient to parse the datetime data into different features, i.e. year, month, day, etc.
- Special attention should be paid to 'ScheduledArrivalTime'. In the original dataset, instead of using standard 24 hours system, 'ScheduledArrivalTime' has ours from 0 to 26. Also date info is not included in the feature. There is some tedious data wraglling work to do for this feature.

In [41]:
```
# parse recordtime to hour, min, sec ↔
```

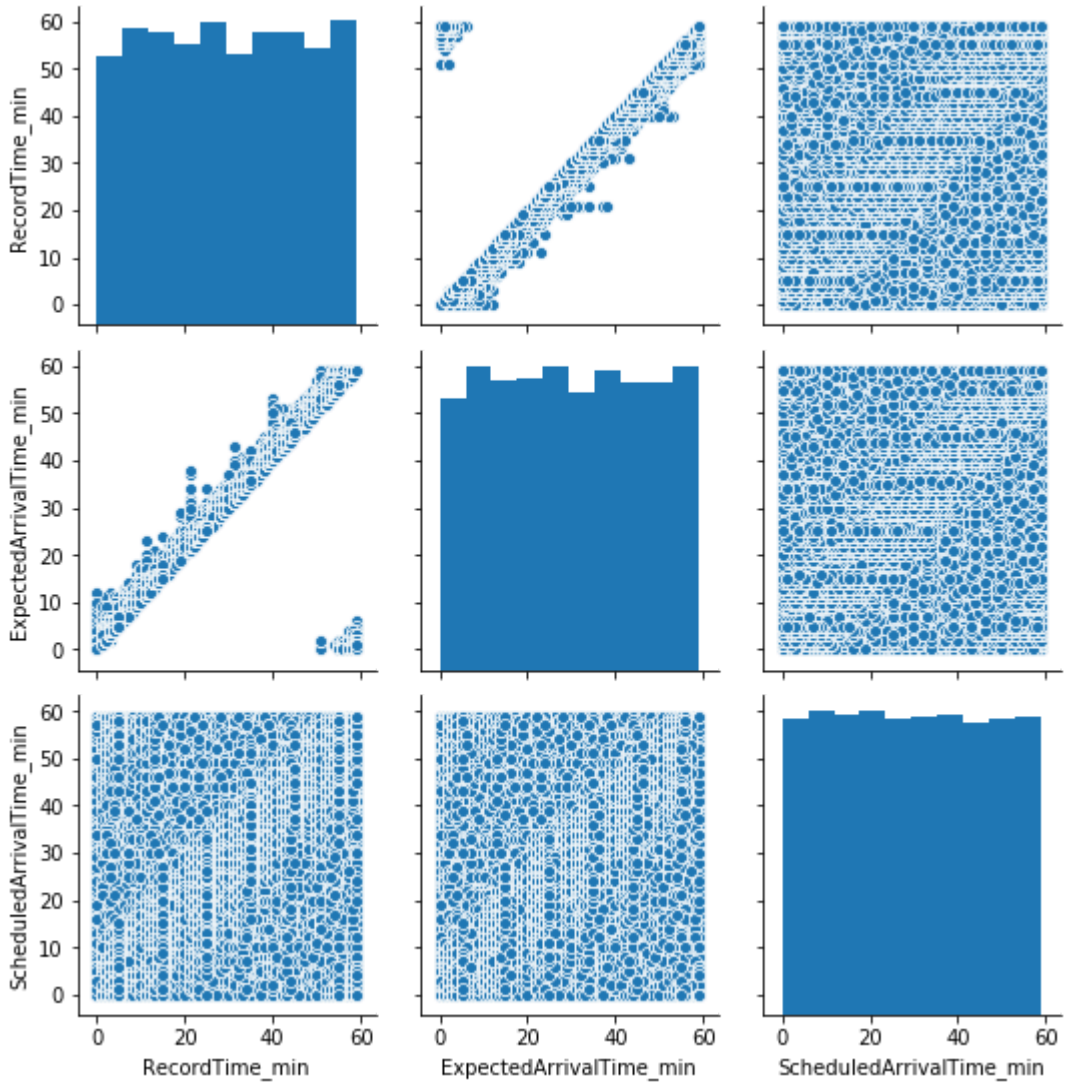
In [195]:
```
# df_trim.head(5)
```

pairplot on hour, min, seconds

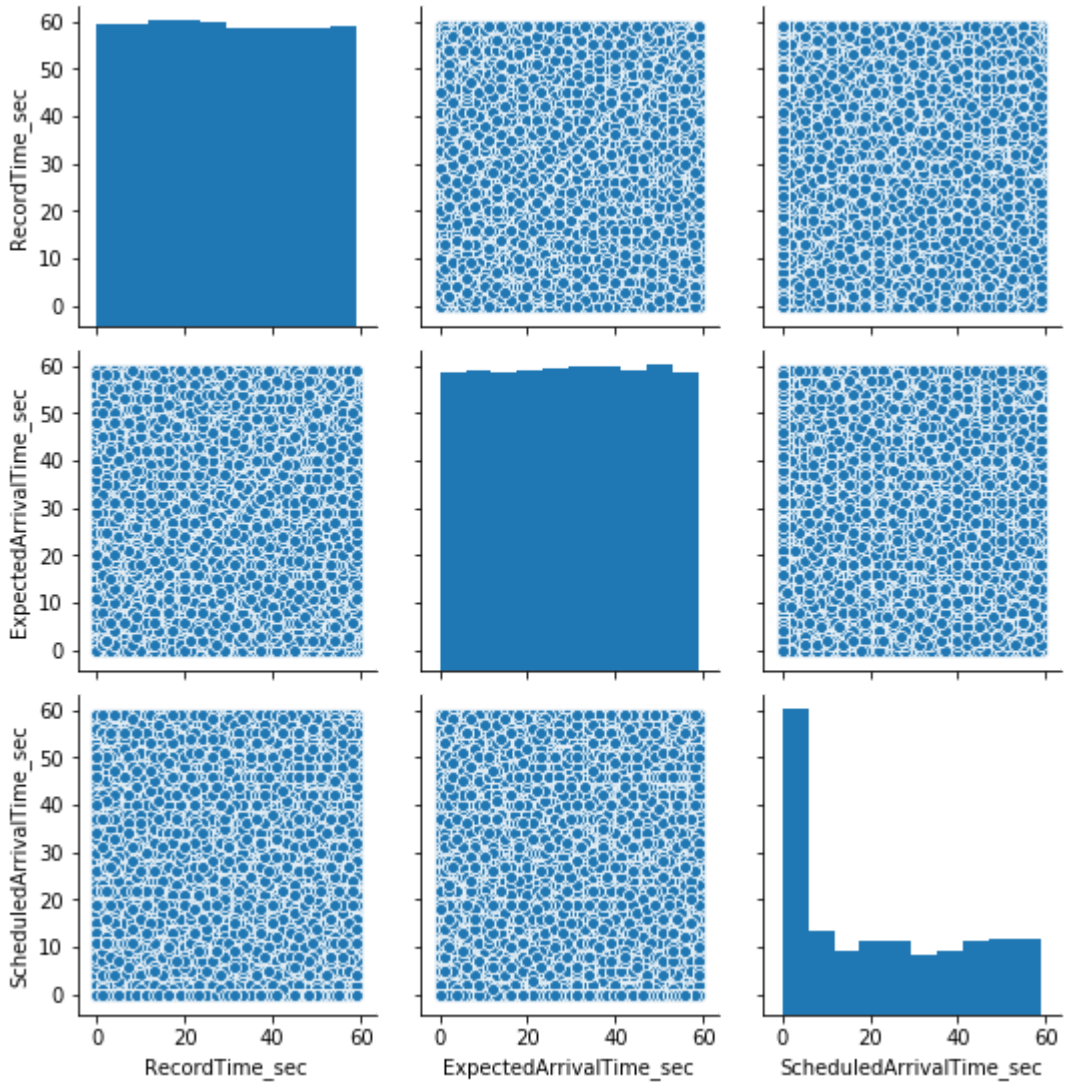
In [246]:
```
1 ▸ # hour↔
```


RecordTime_hour
ExpectedArrivalTime_hour
ScheduledArrivalTime_hour
RecordTime_hour
ExpectedArrivalTime_hour
ScheduledArrivalTime_hour

In [245]:
```
1 ▸ # minute↔
```



In [45]:
```
1 ▸ # seconds↔
```



It is never too crazy to look into details

The data is not very surprising, hour data are highly correlated, which makes sense, since the shedule hour and actual arrival hour would not be that much of different. One thing you might notice is that the schedule arrival hour use a weird hour counting system, the maxium hour is 26, while the record hour and expected hour are using standard 0-23 hour system. We might want to parse the schedule hour to 0-23 later, while handeling the day passing to another one. Also we can 9am and around 16:00 are the busy hours since there are more records around that time.

The expect arrival minute and record minute are also high correlated but not with the scheduled minute. The former high-correlation indicates that the bus is very close to the stop when recorded, which is consistent with the fact that a large propotion of data are captured when "distance from the stop" is approximiately 0, also it indicates that the bus stops are kinda close to each other. While the schedule minute would not be that precise as to very close to the actual arrival minute (assuming actual arrival minute is expected arrival minute). To find out how precise the buses scheduling are, we can use frequency analysis, but this is out of the scope of this project.

It is not surprising that there is almost no correlation between second-related data. I made the pairplot on second level mainly to check if the sampling rate has enough requency. But I am surprised to see that the schedule arrival time is planned at second level, which makes me curious about how they make the scheduling decision to such details.

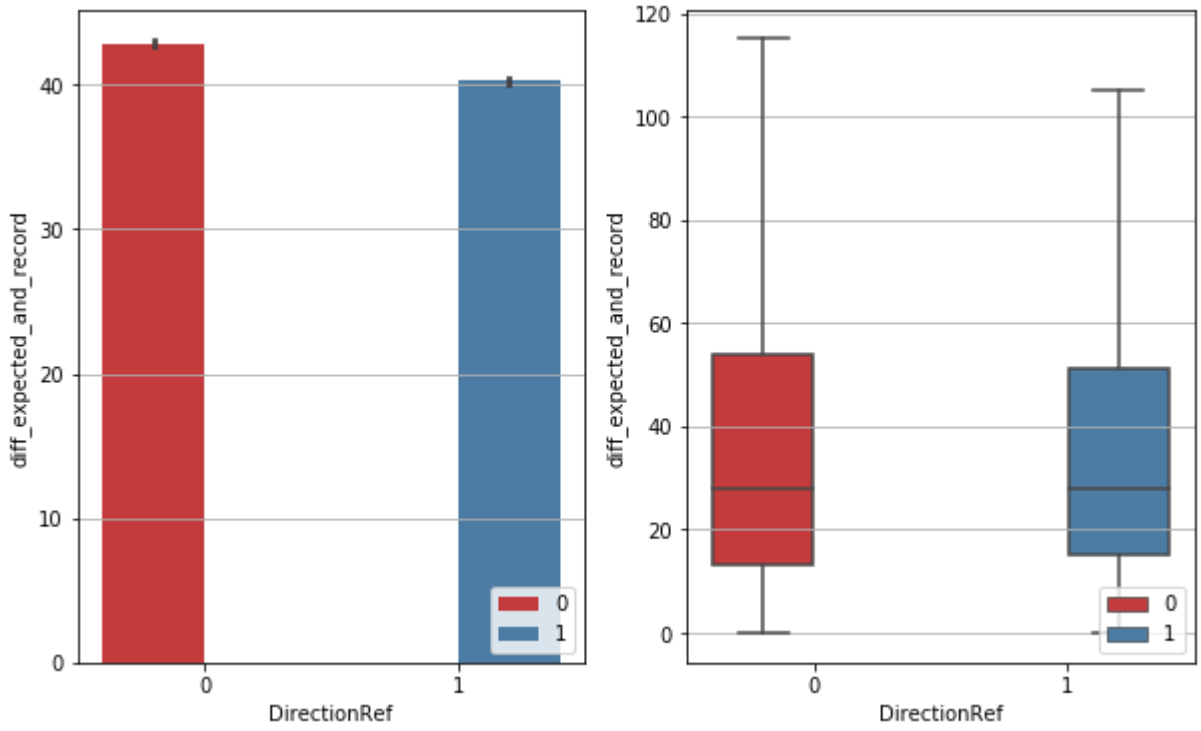
Insights of

Expected time to arrive

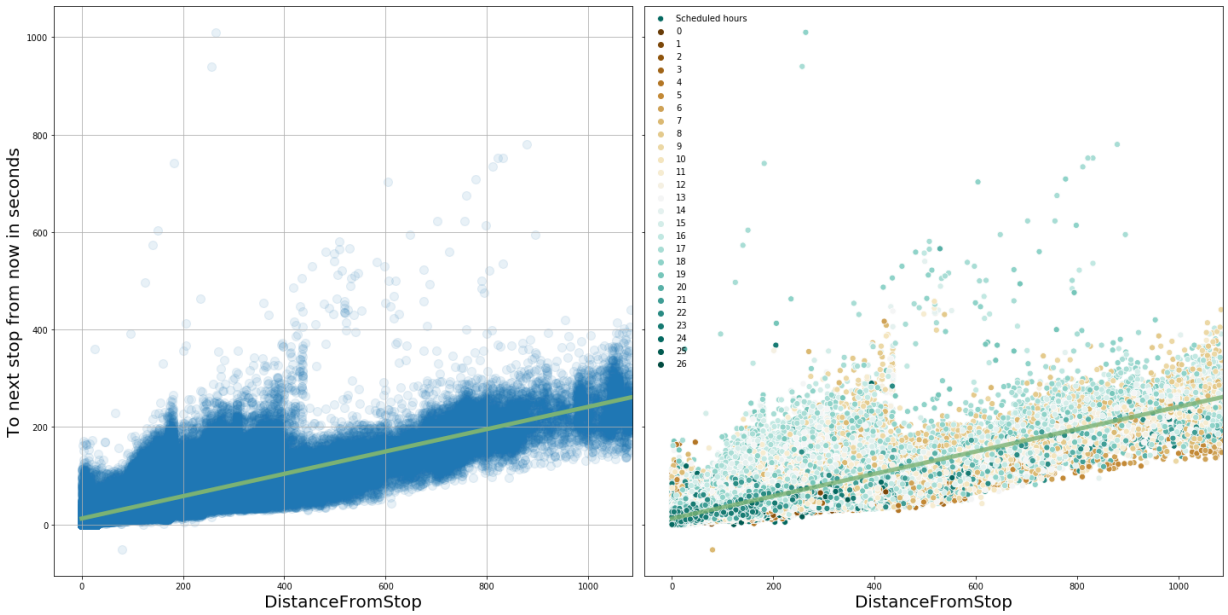
back to main

```
The difference between "ExpectedArrivalTime_hour" and "RecordedAtTime" can read as "expected time to arrive".

Let's try out if there is any trend across the categorical features. The idea here is to show the time difference distribution of feautres in intersts by thier categories. i.e. we use bar plot to show the average time difference when directionreference is 0 and 1, and found out when directionreferece=1 on average it would has more delay than the bus with directionreferece=0, and such difference is statistically significant. But consider that the difference is no more than two minutes (less than 10 seconds), also from boxplot we spot that the 1st-3rd quantiles are mostly overlapped, so we would not consider that the bus driving from one direction would be more likely to delay than another. (Note that I didn't plot outliers on boxplot in order to have a better view on the 1st-3rd quantile data.)

Apply the similar idea to another categorical feauture 'OrginalName', we can also discovery some interesting statistics. i.e. Bus from 'FLATLANDS AV/ELTON ST' has longer delay than those from 'E 45 ST\\GLENWOOD RD'. (which is not shown here)
```

In [258]:
```
# show expected time to arrive using bar plot and box plot ↔
```



In [46]:
```
# distance from the stop and expected arrival time↔
```



- From this one, we can clearly see that the expected time to arrive at next stop from now is highly correlated to the distance from next stop.
- We can use a line to represent such correlationship, and the slope of this line can indicate the speed of the bus. Which is about 6 meters/second or 13 miles/hour or 21.6 km/hour. Considering the bus has to stop during operation, this speed is not too slow.
- Also we can read the plot as when the dots are above the regression line, it means the operation speed is slower than average, while the dots underneath the regression line indicate that thery operated faster than average. When we color code the scatter by scheduled time hours, we can easily find out buses operate from 6:00 to 18:00 are mainly above the regression line, and the bus at midnight are underneath the regression line. Which is consistent to the actual situation.
- We also spot some outliers in the plot, with some buses around 18:00 running much slower than average. Also there is an outlier that the expected time is earlier than the record time. Unlike 'ScheduledArrivalTime', 'ExpectedArrivalTime' should be something that update frequentely. The below zero outlier should be considered as an error and probabely due to the delay of system expectedArrival time calculation. (we can talk a little bit about how the bus is calculate the expected arrival time)

In [198]:
```
1 ▸ # check weird data↔
```

Insights

bus delays

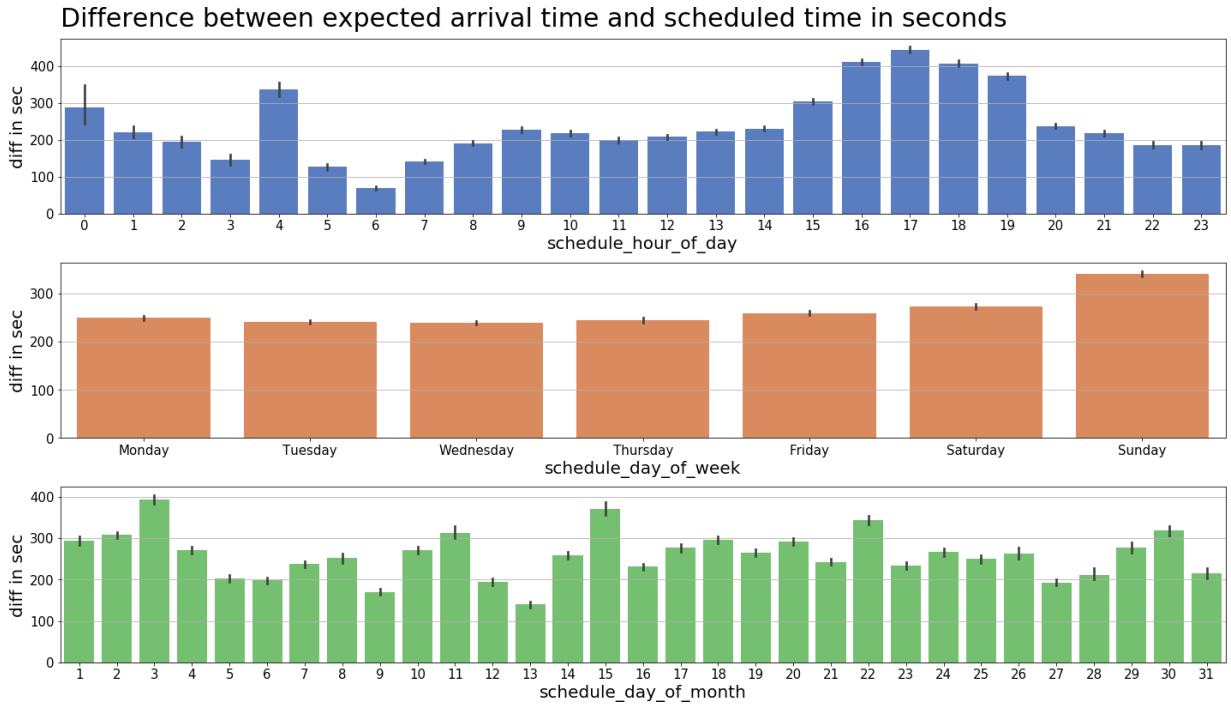
back to main

This is a different time difference

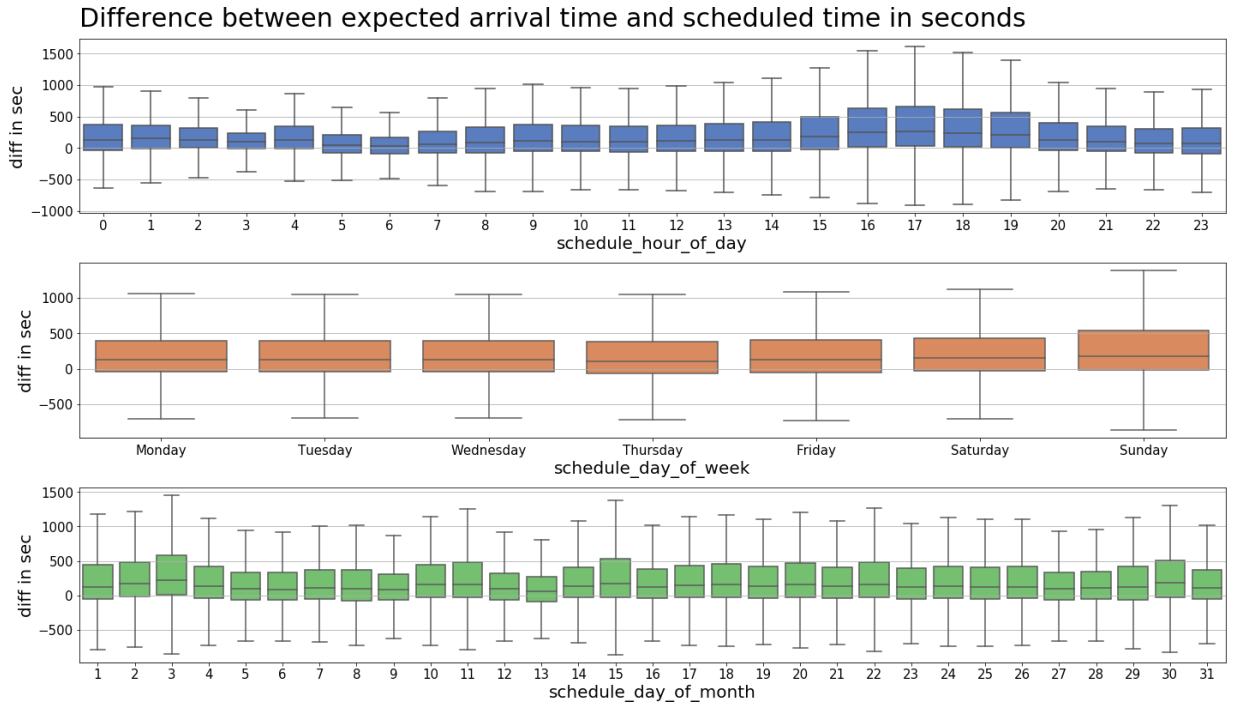
Earlier we looked at the difference between "ExpectedArrivalTime" and "RecordedAtTime", while this time we are considering the difference between "ExpectedArrivalTime" and "ScheduledArrivalTime", which can be interpreted as bus delay. (when the difference is negative, we can say negative delay, or get earlier).

Also this time we are analysis across periodic time data extracted from the original time stamp, i.e. hours, date, day of week. So, let's display the trend of time difference across the timestamp features the same one we analyse not long ago. Here I also create some new features that might have the potential to reveal the pattern of time difference, i.e. bus schedule day of the week, quater-period of the day, etc.

In [48]:
```
1 ▸ # %matplotlib inline↔
```



In [261]:
```
# %matplotlib inline↔
```



Sumary of EDA

We have spent a lot of time on EDA, I called a story about B6, we have used EDA style method to tell a story about such data. We act like a detector and used domain knowledge to explore information underlying. We explore the mechanism how the data is generated, how precise the data is. How to interprete the features besides thier names. i.e. We assume that "expectedarrivaltime" is the "actualarrivaltime" and justify such assumption. Also we parse the schedulearrivaltime to standard form and combine with the date information, in that way we can calculate the differece between schedule arrival time and actual/expected arrival time, and furth create a target called "is_off_time" to reflect the time differece information.

You might have realized data analysis takes half length of this notebook. But I feel it is necessary, because we are performing an unsupervised learning task, without understanding the data, it is almost impossible to evaluate the result nor to interprete that because we don't have labels to guide us.

Before we moving to the next session, I just cannot emphasize enough the importance of feature engineering and the benefit from it for feature extraction and data analysis. But I also want point out that such analysis should not be down to every single details , i.e. there would be easily over hundreds of plots if we try to cross-analyze every features. Rather, it is more important to assure ourselves that the measurement is precise and true, the observation consistent with reality, and confirm/interprete the meaning of variable/feature names.

Just one more example to show how to practice the afforementioned principle. For the last two plots, if we focus on the bus delay by hours. We can see that from the barplot, (which indicate the average difference,) hour 16,17,18 are significant delay than the others. But I would not take it too seriously and give advice based on that. i.e. give a data with difference around 420 secs, I would preict that it is probabely from hour 16, 17, 18. No. I would not do that. Rather, I would go to the boxplot, and say, okay, the i.e. hour 16,17,18 has higher variance, probability because they busy hours, which means there are probablity more counts in the record.

In [270]:
```
1 ▸ # use this check the count↔
```

Design the Task

> ## An unconventional unsupervised learning task demo

back to main

Finally, let's do some machine learning stuff.

As mentioned before, a big issue for unsupervised learning is many times it is hard to evaluate the result. The goal of unsupervised the learning is to find some underlying pattern, but who will be the judge it is a useful patter. Think about the MNIST digit dataset if we don't know what the acutal number is, we might not condident enough to barely rely on metrics to evaluate the result.

Let's create a off-time lable to make this problem into a classification problem so that we have some guidance to evaluate the result. And I will show you how to use PCA as a feature engeering tools and show its benifits by comparing the performance with PCA transformed data and without.

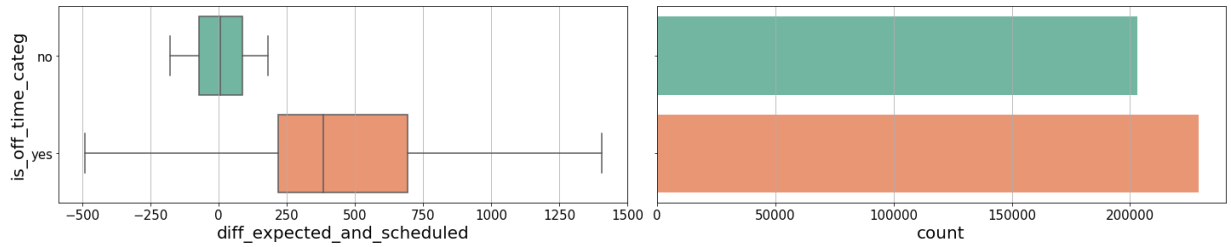
First of all let's assign some labels and design a supervised learning task.

Assign target labels

We can assign an label called "is_off_time", when the absolute difference between "ExpectedArrivalTime" and "ScheduledArrivalTime" is larger than 3 minutes, then its label=1 (off-time), otherwise label=0 (not off-time)

In [49]:
```
1 ▸ # customer function create target↔
```

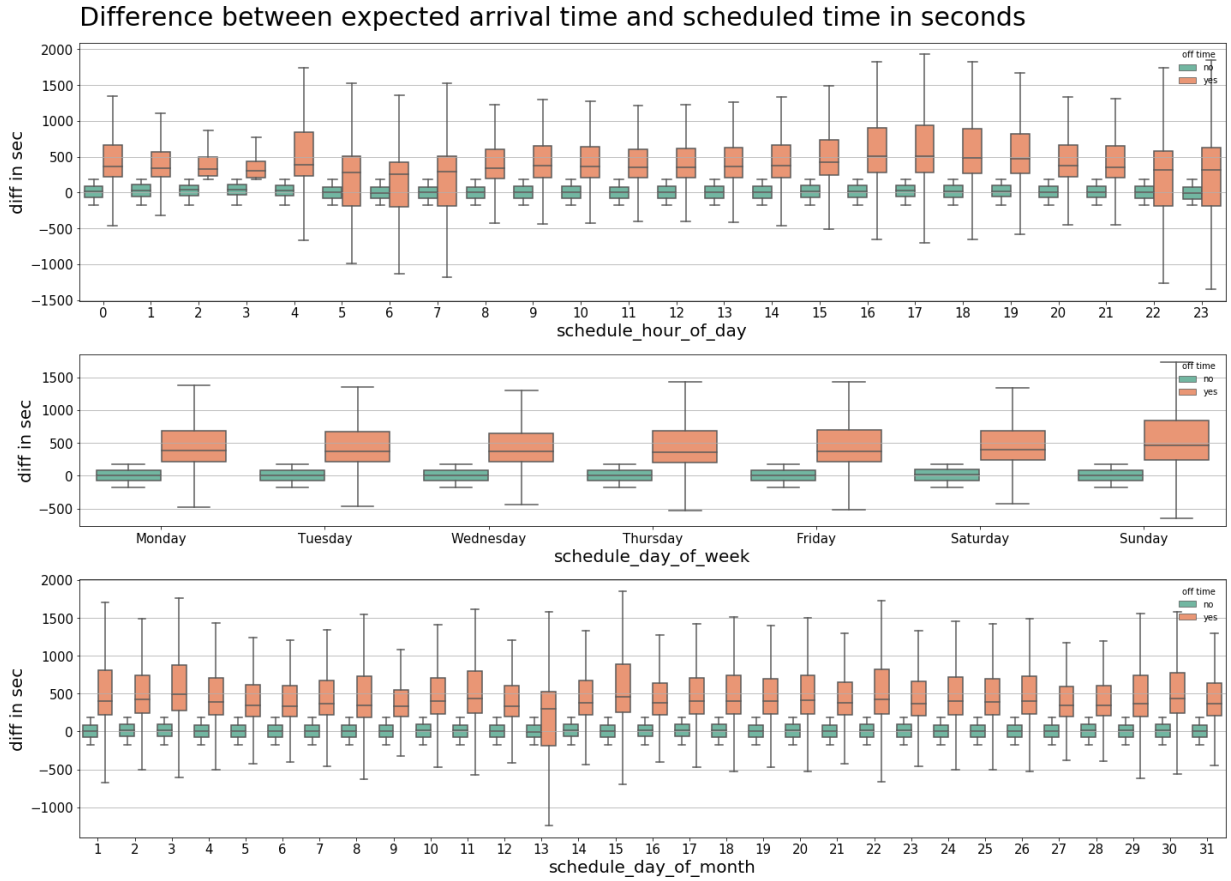
In [272]:
```
# EDA one more time↔
```



In [273]:
```
# proportion of label classes↔
```

```
is_off_time = 0: 0.47027177119850055
is_off_time = 1: 0.5297282288014994
```

In [52]:
```
# EDA cont'↔
```



The new features

In [211]:
```
1 ▸ # create df_work_on↔
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 122843 entries, 91 to 26520637
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   OriginLat                       122843 non-null  float64
 1   OriginLong                      122843 non-null  float64
 2   DestinationLat                  122843 non-null  float64
 3   DestinationLong                 122843 non-null  float64
 4   next_stop_Lat                   122843 non-null  float64
 5   next_stop_Long                  122843 non-null  float64
 6   VehicleLocation.Latitude        122843 non-null  float64
 7   VehicleLocation.Longitude       122843 non-null  float64
 8   RecordTime_month                122843 non-null  int32  
 9   RecordTime_day                  122843 non-null  int32  
 10  RecordTime_weekday              122843 non-null  int32  
 11  RecordTime_hour                 122843 non-null  int64  
 12  RecordTime_min                  122843 non-null  int64  
 13  RecordTime_sec                  122843 non-null  int64  
 14  DirectionRef                    122843 non-null  int32  
 15  OriginName                      122843 non-null  object 
 16  DestinationName                 122843 non-null  object 
 17  NextStopPointName               122843 non-null  object 
 18  VehicleRef                      122843 non-null  object 
 19  RecordedAtTime                  122843 non-null  object 
 20  ScheduledArrivalTime_standand   122843 non-null  object 
 21  diff_expected_and_scheduled     122843 non-null  float64
 22  is_off_time                     122843 non-null  int32  
 23  ArrivalProximityText            122843 non-null  object 
dtypes: float64(9), int32(5), int64(3), object(7)
memory usage: 21.1+ MB
```

This is not a trivial problem?

Given the features above, do you think you have a good strategy to solve that? i.e. label the data as is_off_time=0 or 1. Note that the ExpectedArrivalTime data should not be included in the estimator, otherwise it would be cheating, a so-called lookahead leakage. So you cannot subtract scheduleTime from ExpectedArrivalTime to get the difference. Besides, suppose you don't know how many minutes would be consider off-time.

Also, the data is from bus that are not "at stop" or "approching" the store. Actually you don't even know how far you are away from the stop, all you have is the current GPS coordinates and "ArrivalProximityText" to indicates bus is "1 stop away", "<1 stop away", etc.

Don't worry, I am not serioiusly asking you to solve the problem but just to claim that this problem is non-trivial, even though you might have some clue about the underlying mechanism.

For the following part, we will sort to machine learning models to build a estimator that predict the "is_off_time" label. Especially, I will demonstrate how to utilize the result from unsupervised learning and integrate the results into a supervised learning task.

(I will still work on a trimmed version dataset, just to show the workflow to build the pipeline, which can be later applied to the original dataset)

In [199]:
```
# feature engineering↔
```

In [275]:
```
# df_work_on.info()
```

In [201]:
```
# df_work_on.columns
```

In [57]:
```
# df_X and df_y↔
```

In [210]:
```
# df_X.head(5)
```

In [212]:
```
# df_X.info()
```

Machine Learning

Data preparation

> **Train-test split**
>
> **PCA**

back to main

Train-test split

I choose the train:test=2:1 split.

Last time, somebody asked me why I choose that value and wonder if the size of test set is a bit large, and I didn't have a very good answer to that. This time, if somebody asks me the same question then my answer would be, this is Sklearn's choice in thier website demo. Also I would be concerned if the test size is too small since it might cause unrevealed overfitting.

Basically it became a semi-supervised learning problem if we have small amount of labeled target.

In [448]:
```
# train test split↔
```

PCA

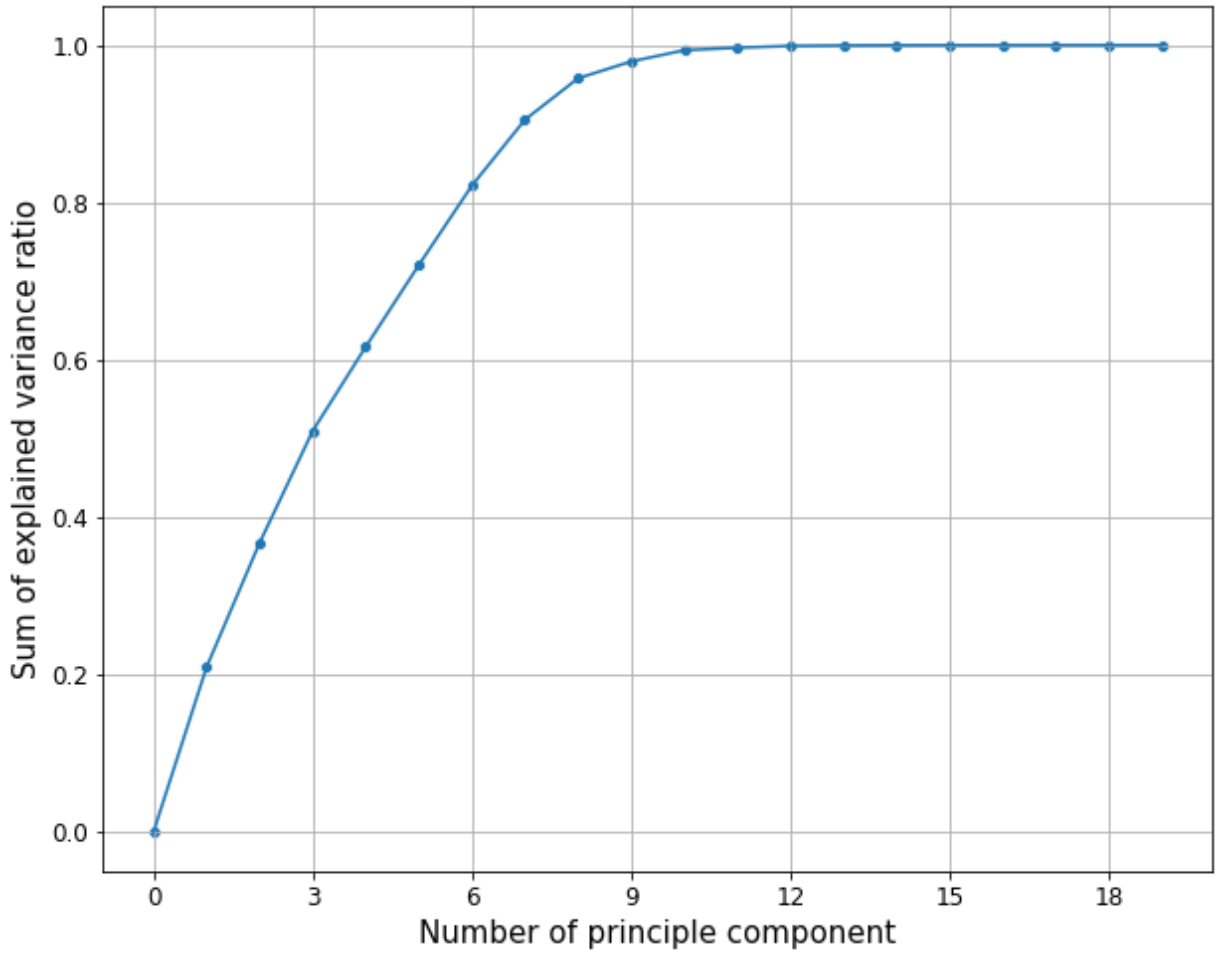
In [449]:
```
# preprocessing scaling↔
```

In [450]:
```
# pca to handle correlation and sparsity↔
```

In [451]:
```
pca.explained_variance_ratio_.cumsum()
```

Out[451]:
```
array([0.20941386, 0.36745551, 0.50962413, 0.61654536, 0.72047286,
       0.82141115, 0.90519341, 0.95779428, 0.97913786, 0.99341059,
       0.99667152, 0.99893656, 0.99958044, 0.99985671, 0.99991   ,
       0.99995923, 0.9999931 , 1.        , 1.        ])
```

In [64]:
```
# plot explained_variance cumsum↔
```



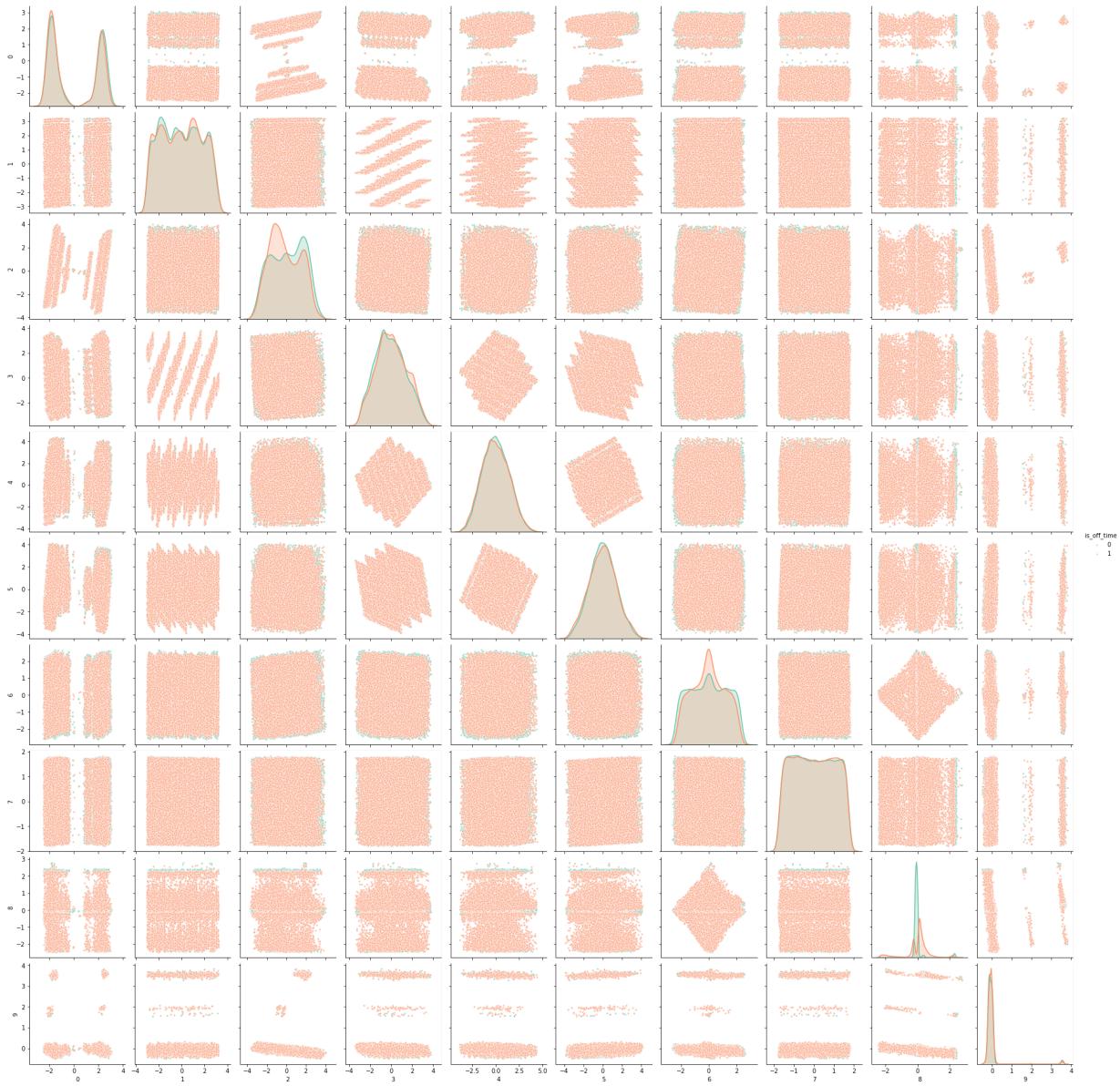
The first 10 principle components already explained 99% of the variance, so it is reasonable to just take the first 10 priciple componenets.

In [65]:
```
# choose the top PCA components with threshold = 0.99↔
```

First let's try our luck to see if the data after PCA show any pattern that might help. i.e. the different target would form some cluster.

In [67]:
```
1 ▸ # pair plot↔
```

Out[67]: <seaborn.axisgrid.PairGrid at 0x28f68513788>



At first sight, the transformed data has a kind of weird distribution, this is mainly becasue many features like hours, day of weeks is encoded from categorical data.

Also it doesn't seem like PCA is very helpful either. i.e. both is_off_time==0 and is_off_time==1 have very similiar distribution. In other word, the data is not linearly seperatable. (What you see in the scatterplot that is all orange is because of data from the two classes are overlapping with each other and the orange scatters plot after the green ones).

Model training

back to main

Choice of models

Let's try if machine learning algorithm can learn from the data. Here I choose

- Logistic regression
- Random forest
- XGBoost
- LightGBM

Logistic regression is chosen as a base-line model. Random forest is chosen because it can mimic the many people voting situation. XGBoost and LightGBM are the stars in the supervised learning family and potentially perform well in many situations.

The emphasize here is not demonstrate how to fine tune the machine learning models, so all the models use their default hyper-parameter settings. And there is definetely better cross-validation schemes to help imorve performance and justify evaluation.

Training Demo

In [452]:
```
# copy training data↔
```

try logistic regression

In [454]:
```
# train with the default model
from sklearn.linear_model import LogisticRegression

lr1 = LogisticRegression(
    max_iter=1e4,
    random_state= RANDOM_STATE)
lr1.fit(X=X_train_1, y=y_train_1.is_off_time.values)
lr1.get_params

print("f1-score on training set: {}".format(f1_score(y_true=y_train_1, y_pr
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=lr

```

```
f1-score on training set: 0.6328348992104451
f1-score on test set: 0.6313850577294577
```

Evaluation

back to main

In [455]:
```
from sklearn.metrics import classification_report, confusion_matrix, roc_cu
from sklearn.metrics import precision_recall_curve, average_precision_score
```

In [456]:
```
# Customer function # evaluation and visulization function
def model_evaluation(y_test, y_pred, y_pred_prob=None, is_print_classificat
```

In [457]:
```
# customer func for evaluation and visualization
def plot_prediction_probability(y_test, y_pred, y_pred_prob=None):↔
```

In [458]:
```
# customer functioin plot precision
def plot_probability_calibration(y_test, y_pred, y_pred_prob=None):↔
```

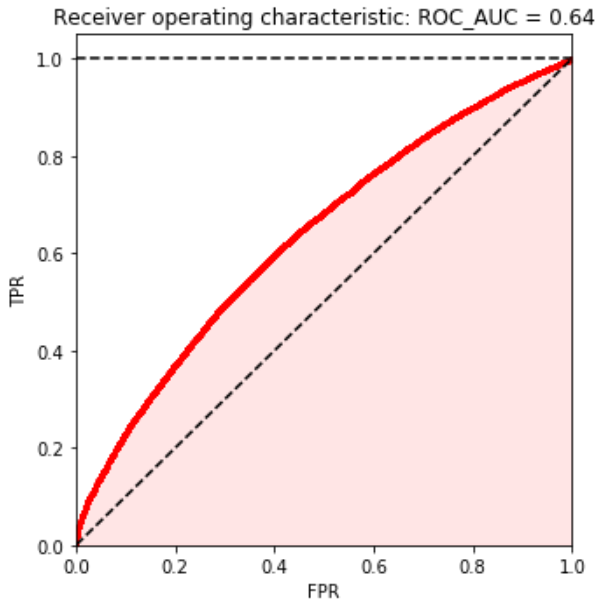
In [459]:
```
# Customer func for precision analysis
def precisionAnalysis(true_label, proba, recall_at):↔
```

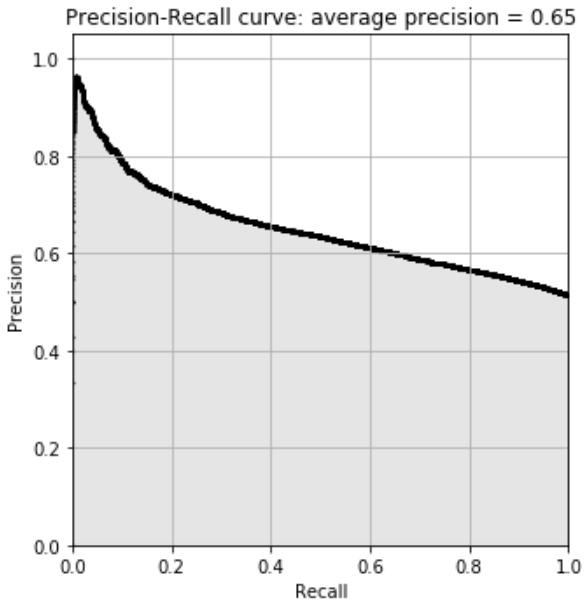
evaluation on logistic regression

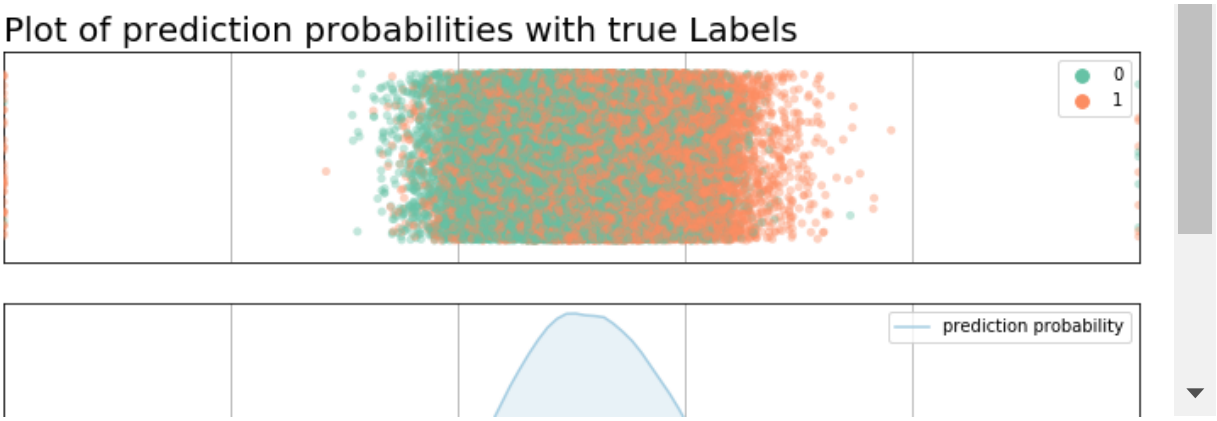
In [460]:

```
# evaluation metrics and visualization
y_pred_model = lr1.predict(X_test_1)
y_prob_model = lr1.predict_proba(X_test_1)[:,1]
# y_prob_model

model_evaluation(y_test=y_test_1, y_pred=pd.DataFrame({'pred_yes':y_pred_m
plot_prediction_probability(y_test=y_test_1, y_pred=pd.DataFrame({'pred_ye
precision = precisionAnalysis(true_label=y_test_1, proba=y_prob_model, rec
print("When recall is 75%, Precison is: ", precision)
plot_probability_calibration(y_test=y_test_1, y_pred=pd.DataFrame({'pred_y
```

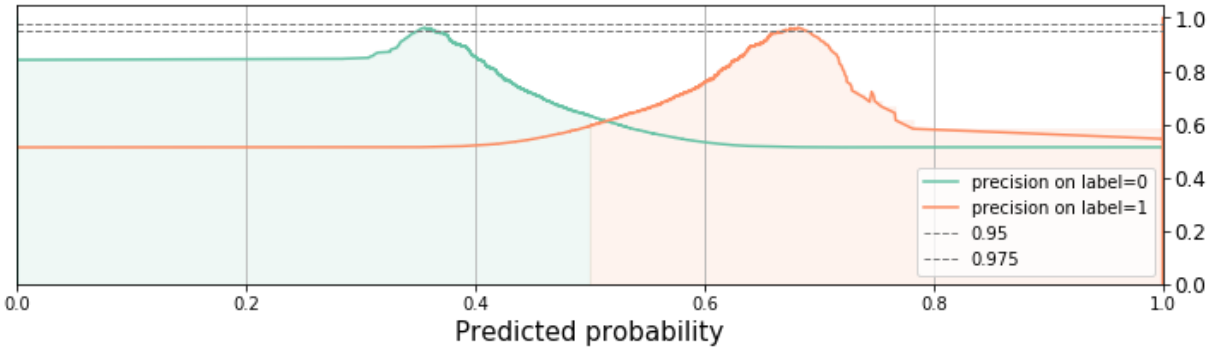
```
classification_report
              precision    recall  f1-score   support

           0       0.60      0.51      0.55     19708
           1       0.59      0.67      0.63     20831

    accuracy                           0.60     40539
   macro avg       0.60      0.59      0.59     40539
weighted avg       0.60      0.60      0.59     40539
```








```
When recall is 75%, Precison is:  0.5775486064907223
```



Explain on the result

The evaluation metrics can be read directely from the classificaiton report, including accuracy, precision, recall and f1 score on both 0 and 1 class.

- The ROC curve is displayed. For ROC curve we usually care about the area under curve (AUC). Note that there is an diagonal line. I don't want to go to the details to talk about that, just remember, we desire an ROC curve above the diagonal line as much as possible which will gives us an maximum AUC ROC value of 1, while the diagonal line is associated with AUC ROC = 0.5.
- Precision-recall curve is also ploted. For this curve, we normally care about average precision and the precision at certain recall level, i.e. at 75% reacall.

The performance of logistic regression model is not very good in this case. We were already expecting this since the data is not linearly seperatable.

- Among the accuracy is around 0.55, recall=0.60, and precision=0.57. Note that such metrics are calculated assuming the threshold is 0.5. note*
- ROC AUC is a 0.57, which we can see from the plot that ROC curve is just above the diagonal line.
- The average precision is 0.57, and it is pretty flat across all the recall value, (most of the case, precision would drop when recall increase), with precision=0.56 when recall is 75%.

Next, I would like to introduce to you the predication probability strip plot. I found it very intuitive to show the distribution of prediction probablity with repective to the true label. This plot should be consistent with the conclusion drawn from ROC curve and precision-recall curve, but it also tells a story in more details about the prediction probability distribution, which can be helpful for making

decision in proability calibration. i.e. we an pick up predictions that when we are very sure according to its prediction proability. I will explain this concept on the following machine learning models and the idea should be immediately clearer to you.

note*: some metrics might change due to I choose different sample size, but they should not be dramatically different.

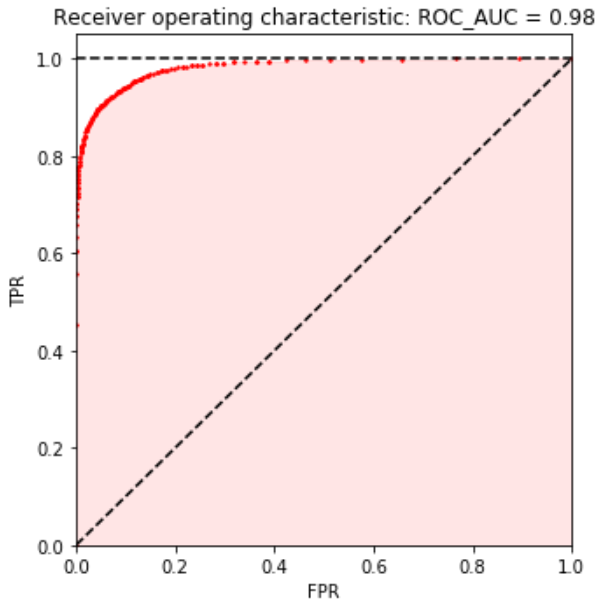
try random forest

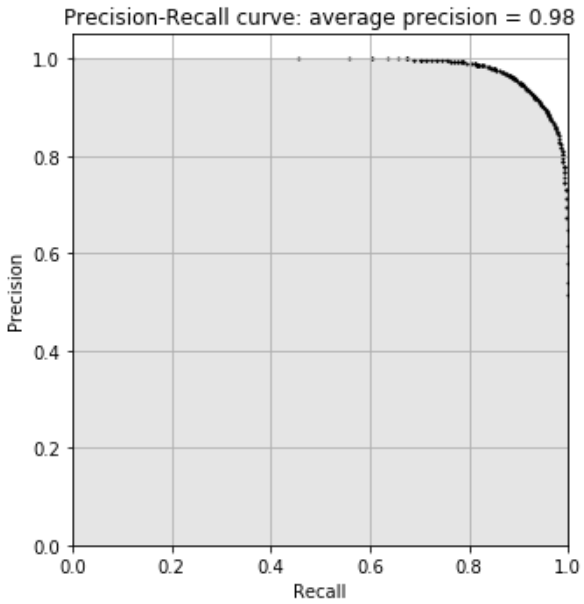
In [462]:
```
from sklearn.ensemble import RandomForestClassifier

rfc1 = RandomForestClassifier(random_state= RANDOM_STATE)
rfc1.fit(X=X_train_1, y=y_train_1.is_off_time.values)
y_pred_model = rfc1.predict(X_test_1)
y_prob_model = rfc1.predict_proba(X_test_1)[:,1]

print("f1-score on training set: {}".format(f1_score(y_true=y_train_1, y_pr
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=r
```

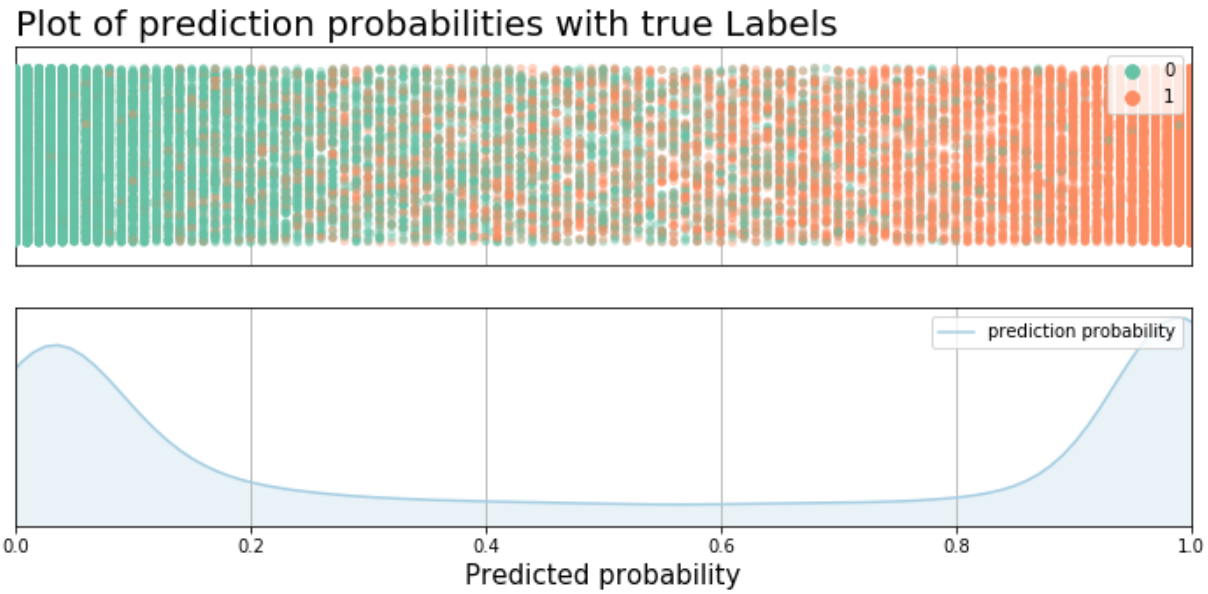
```
f1-score on training set: 0.9999646030229018
f1-score on test set: 0.9265798124923902
```

In [463]:
```
1 ▸ # evaluation metrics and visualization↔
```

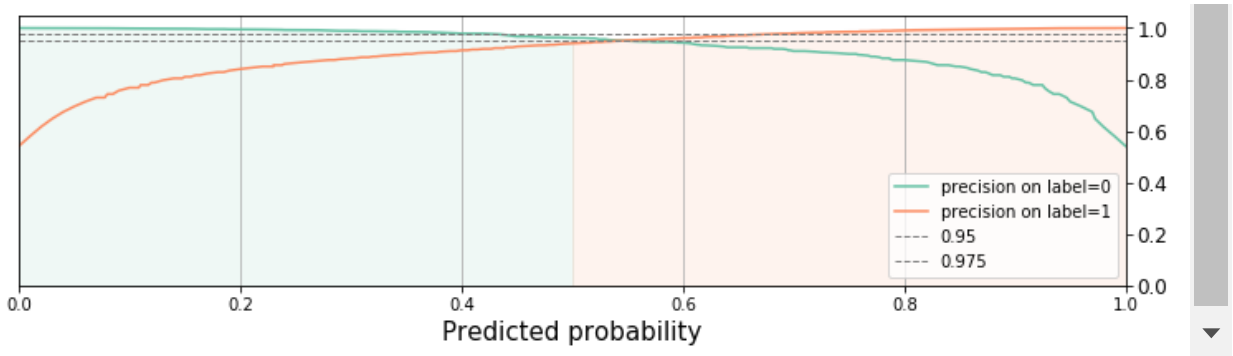
```
classification_report
              precision    recall  f1-score   support

           0       0.91      0.94      0.92     19708
           1       0.94      0.91      0.93     20831

    accuracy                           0.93     40539
   macro avg       0.93      0.93      0.93     40539
weighted avg       0.93      0.93      0.93     40539
```








```
When recall is 75%, Precison is:  0.9961746891934969
```



In [464]:
```
1 ▸ # calculation cast rate at (0.2 to 0.8)↔
```

Out[464]: 0.18929919336934803

evaluation on random forest

The performance has improved a lot,

- accuracy, precision, f1 score are over 0.90, with recall a short of 0.89. Again, all these are associated with proability threshold==0.5.
- the AUC ROC and average precision is close to a full score. with AUC of ROC =0.97, average precision = 0.98.

Let's put our atttention to prediction proability stip plot. This time the prediction probability distribution is seperated on two side. With the actual labels color code with green=0, orange=1, you should realize that when an orage lable comes to the left side it indicates the model made a wrong prediction. (Here assuming the arbitrary threshold is set to 0.5).

Next, a good nature of many machine learning models is when it has a prediction value close to 1 or 0, pretty good chance that this prediction is correct. This phenomenon can be easily spot in the precision-probability plot (the last plot). i.e. when the prediction probability is over 0.8, the precion on label=1 is over 0.975, and when the prediction probability is less than 0.2, the precision on label=0 is also over 0.975. Actually it is very straightforward to see that as the prediction probability increase, the precision on label=1 also increase, on the other hand, when the prediction probability decrease, the precision on label=1 also increase. (Hopefully you see what I am doing now).

What I describe is the idea of probability calibration. Namely, we can cast away those predictions with value around 0.5, which can help to improve precision (at the cost of decrease of recall of course.).

Note that if we throw away those prediction is within (0.2-0.8), the precision on both label=0 and lable=1 are above 0.975. Then how much element we have lost? We will lost 20% of the prediction. Since the logistic regression is too bad, for now just remember this number and we will compare the rest of the models.

notes* I drop the color to represent a more realistic condition, that is when you look at the prediction probaility distribution, there is no color to tell you the actual label. But this should not

In [ ]:
```
1 ▸ ### challenge for you↔
```

try XGBoost

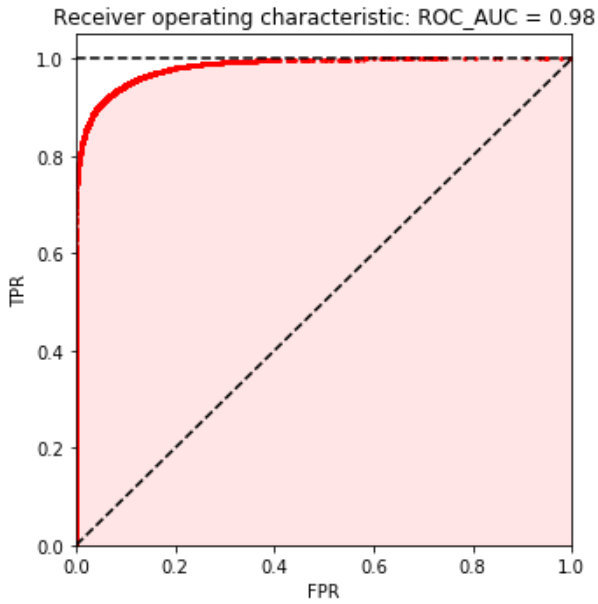
In [466]:

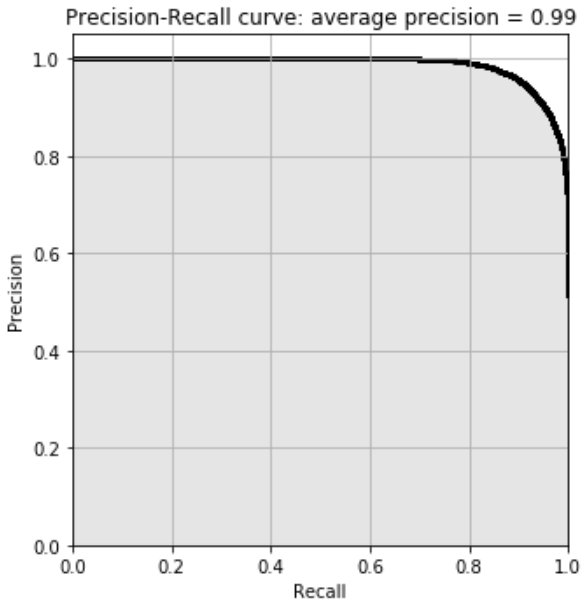
```
from xgboost import XGBClassifier

xgbc = XGBClassifier()
xgbc.fit(X=X_train_1, y=y_train_1.is_off_time.values)
xgbc.get_params

print("f1-score on training set: {}".format(f1_score(y_true=y_train_1, y_pr
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=xg
```

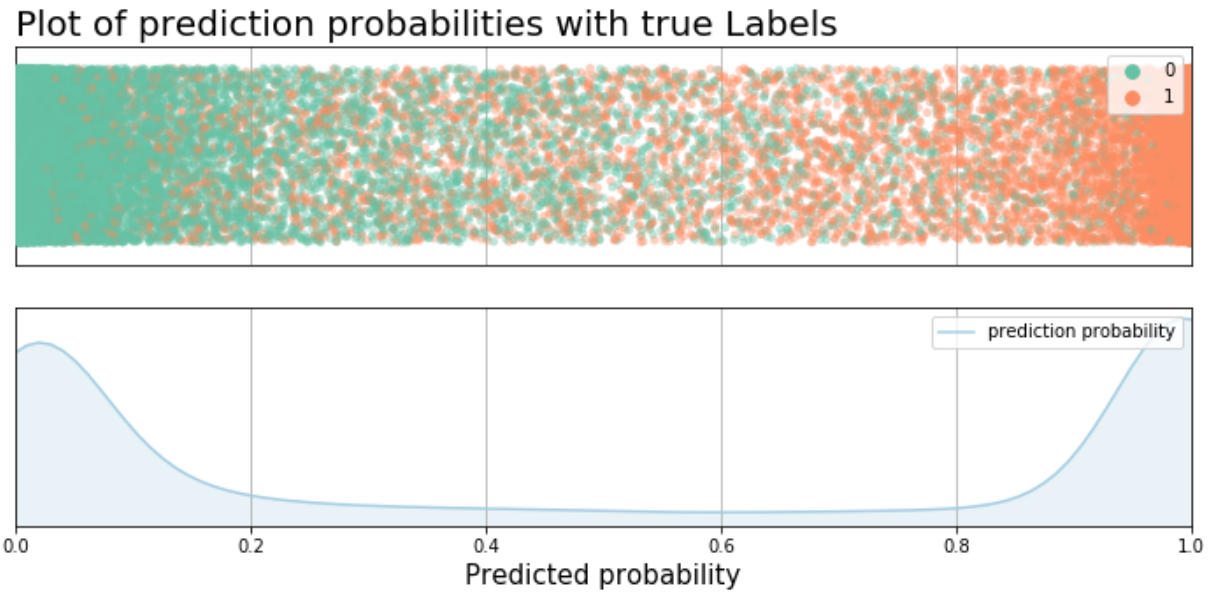
```
f1-score on training set: 0.9599579259400923
f1-score on test set: 0.9294292465469781
```

In [467]:
```
1 ▸ # evaluation metrics and visualization↔
```

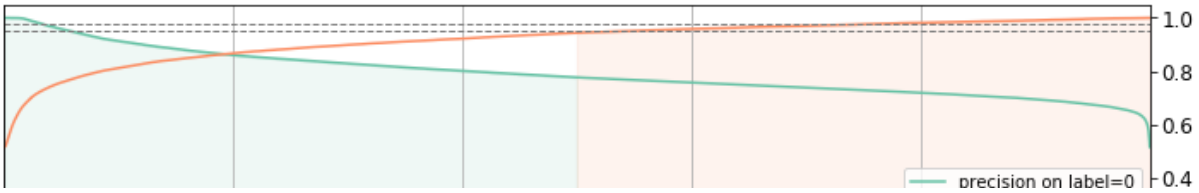
```
classification_report
              precision    recall  f1-score   support

           0       0.91      0.94      0.93     19708
           1       0.94      0.92      0.93     20831

    accuracy                           0.93     40539
   macro avg       0.93      0.93      0.93     40539
weighted avg       0.93      0.93      0.93     40539
```








```
When recall is 75%, Precison is:  0.997701296213524
```



In [468]:
```
1 ▸ # calculation cast rate at (0.2 to 0.8)↔
```

Out[468]:
```
0.1323416956511014
```

sumary of XGBoost

I will save some words here but just talk about one thing. Remember the 20% prediction random forest cast away. What about XGBoost? This number is down to 13%. Wow, great improve.

From the classification report you can see that, even the metrics didn't seem better than the one of random forest. But when look into the prediction probability, we would realize the result from XGBoost is actually even better.

Let's move on to take a look at the last candidate. (Let me if proability calibration doesn't make much sense to you)

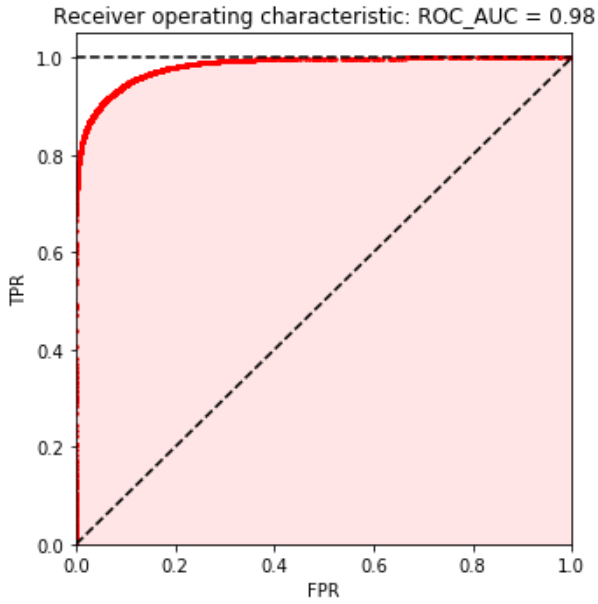
In [469]:
```
1 ▾ # try lightGBM
```

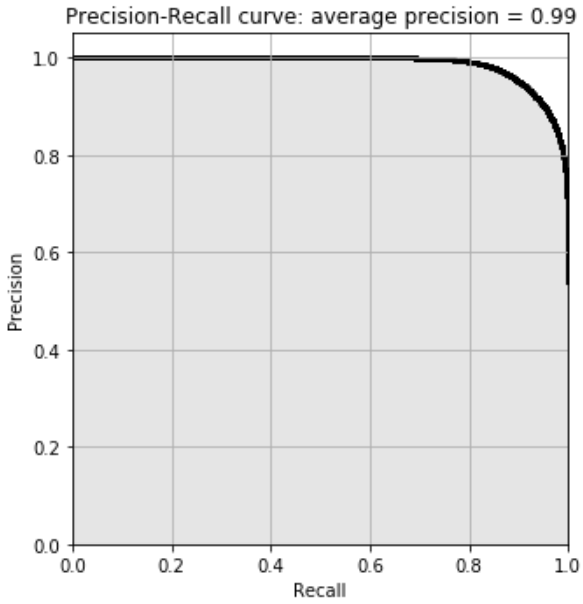
In [470]:
```
1  from lightgbm import LGBMClassifier
2
3  lgbm = LGBMClassifier()
4  lgbm.fit(X=X_train_1, y=y_train_1.is_off_time.values)
5  lgbm.get_params
6
7  print("f1-score on training set: {}".format(f1_score(y_true=y_train_1, y_pr
8  print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=lg
```

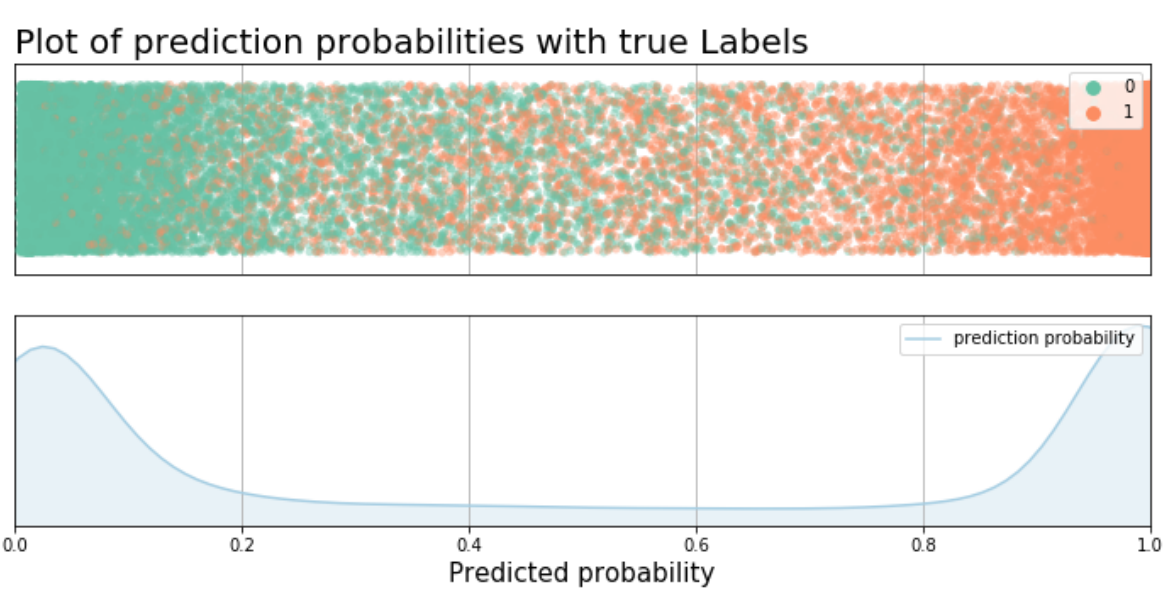
```
f1-score on training set: 0.9382898917743746
f1-score on test set: 0.9270177655722167
```

In [471]:
```
1 ▸ # evaluation metrics and visualization↔
```

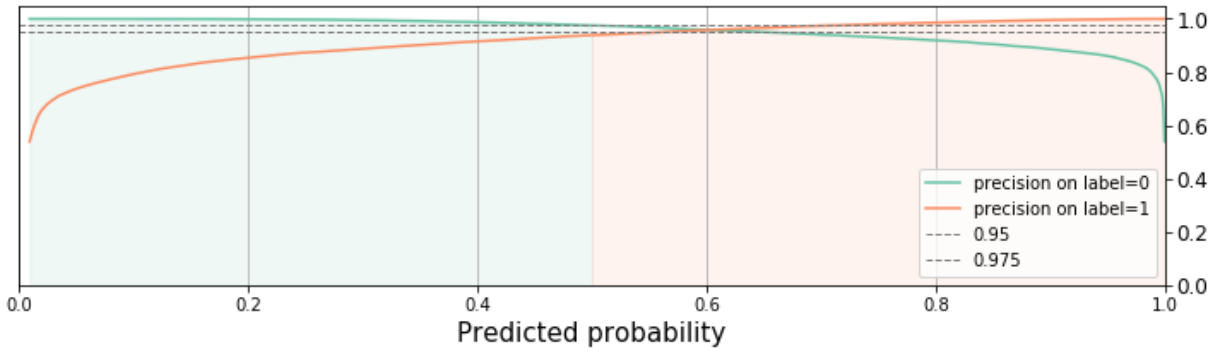
```
classification_report
              precision    recall  f1-score   support

           0       0.91      0.94      0.92     19708
           1       0.94      0.92      0.93     20831

    accuracy                           0.93     40539
   macro avg       0.93      0.93      0.93     40539
weighted avg       0.93      0.93      0.93     40539
```







```
When recall is 75%, Precison is:  0.9972555527189175
```



In [472]:
```
1 ▸ # calculation cast rate at (0.2 to 0.8)↔
```

```
0.1564912800019734
0.0496558869237031
```

Summary on lightGBM

I will just brief the result here. The performance based on metrics is very similar to XGBoost. From probability calibration point of view, the cast rate is 0.16, a bit higher than the 13% of XGBoost, when choosing threshold 0.2 and 0.8. If you think this is too aggressive then you can choose threshold 0.4 and 0.6. While the precision is still above 0.95, and the cast rate will down to 0.04.

without PCA

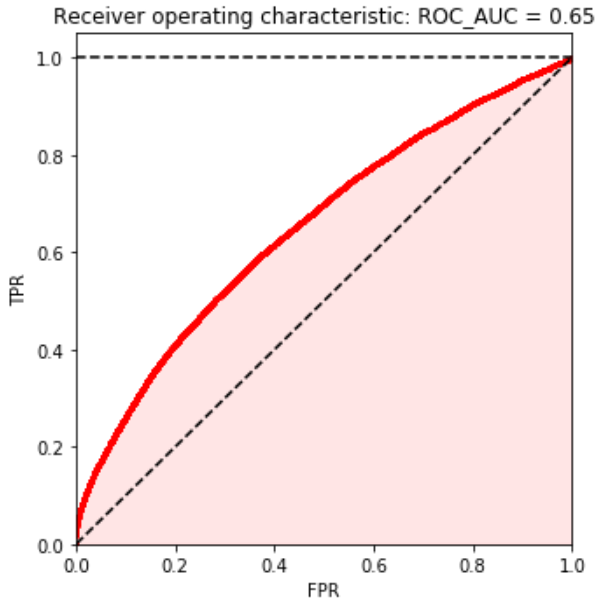
In [473]:
```
1 ▸ # X_train_2 = df_X_train.copy()↔
```

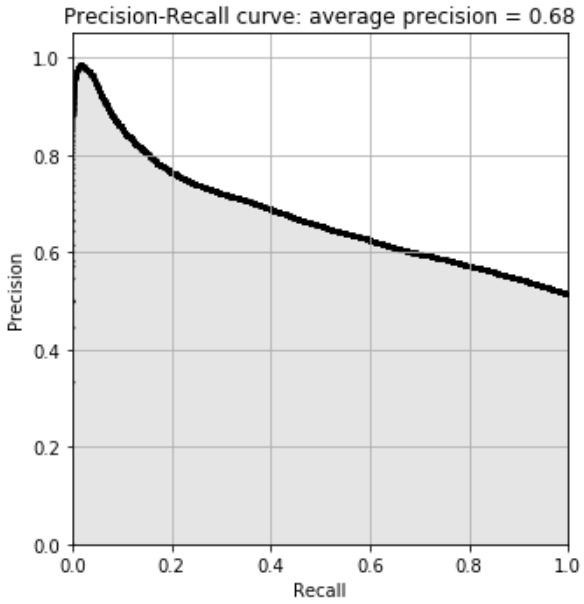
In [474]:
```
1 ▸ # logistic regression, train with the default model↔
```

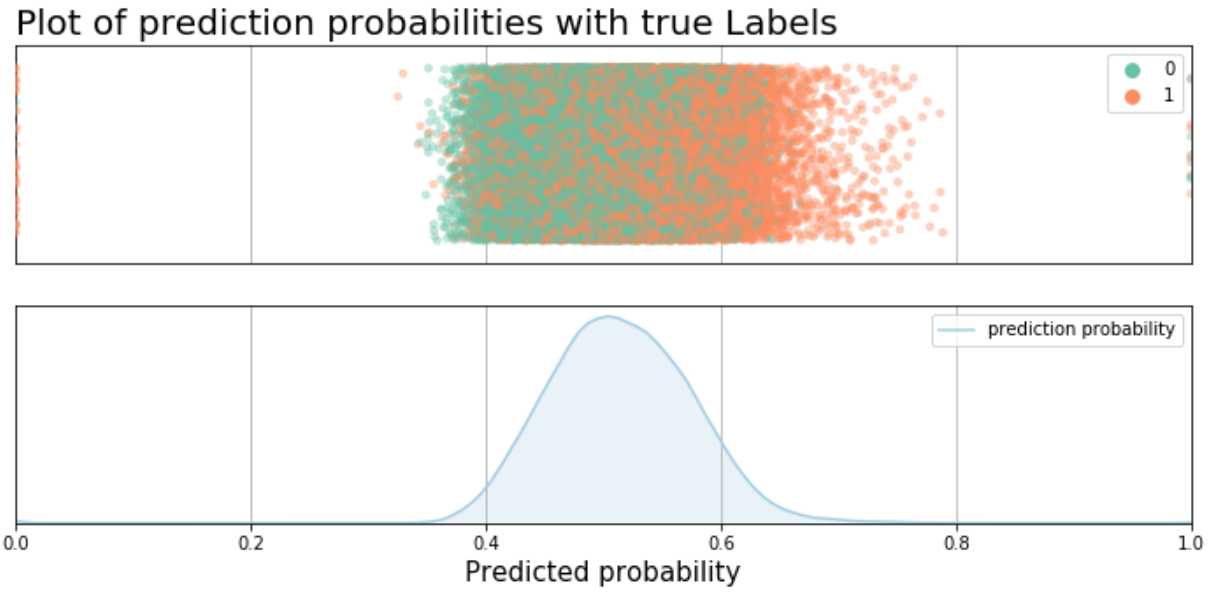
```
f1-score on training set: 0.6391839170654613
f1-score on test set: 0.6385227835005173
```

In [475]:
```
1 ▸ # evaluation metrics and visualization↔
```

```
classification_report
              precision    recall  f1-score   support

           0       0.61      0.52      0.56     19708
           1       0.60      0.68      0.64     20831

    accuracy                           0.60     40539
   macro avg       0.60      0.60      0.60     40539
weighted avg       0.60      0.60      0.60     40539
```







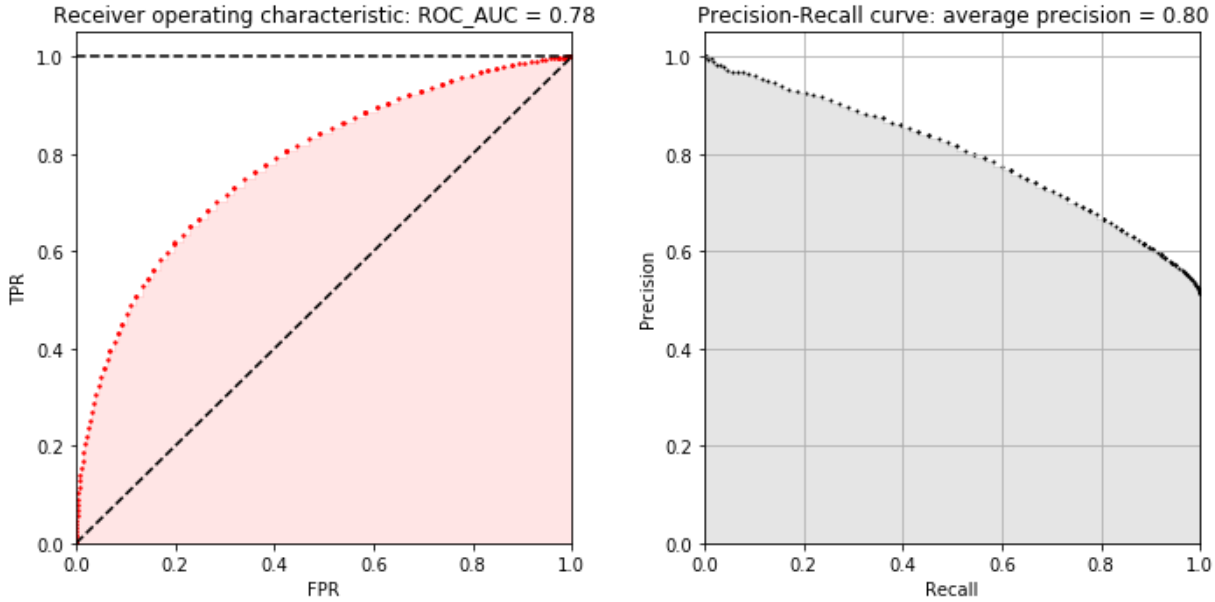

```
When recall is 75%, Precison is:  0.5862819406401261
```

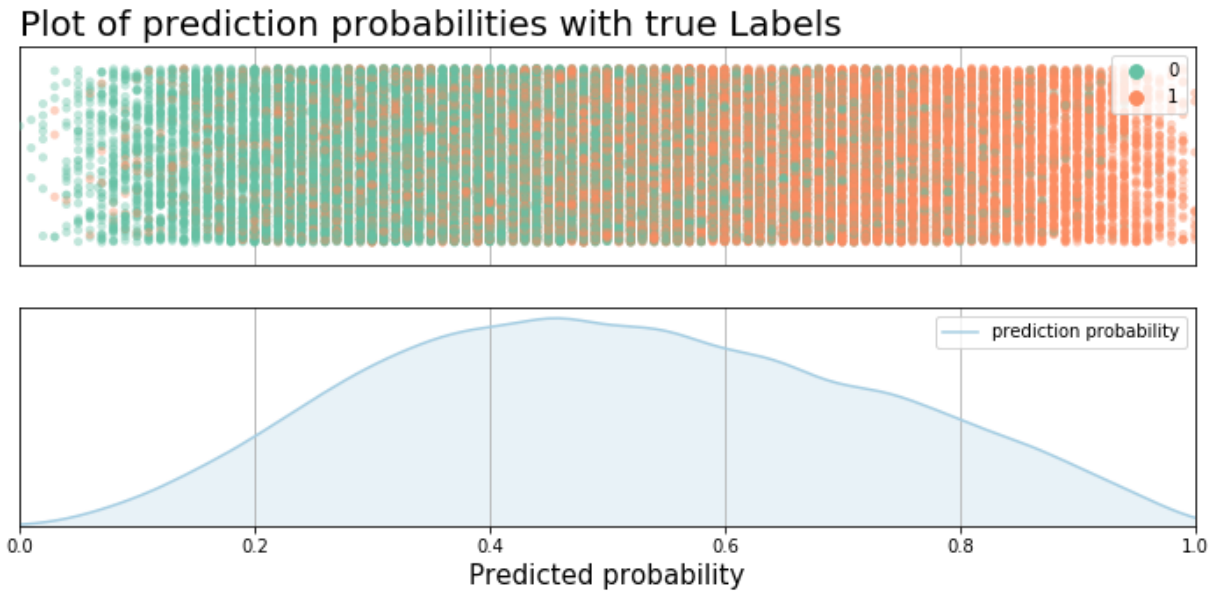
In [476]:
```
1 ▸ # random forest↔
```

```
f1-score on training set: 0.9999646021875848
f1-score on test set: 0.7118718455123986
```

In [477]:
```
1 ▸ # evaluation metrics and visualization↔
```

```
classification_report
              precision    recall  f1-score   support

           0       0.69      0.72      0.71     19708
           1       0.72      0.70      0.71     20831

    accuracy                           0.71     40539
   macro avg       0.71      0.71      0.71     40539
weighted avg       0.71      0.71      0.71     40539
```





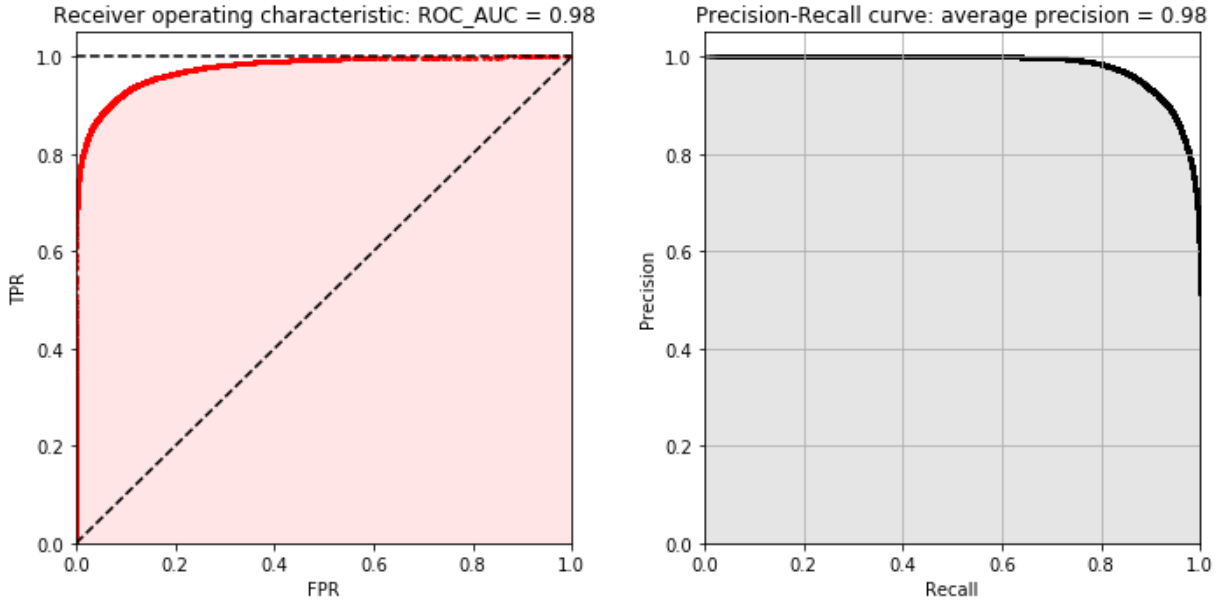

```
When recall is 75%, Precison is:  0.6974201035529369
```

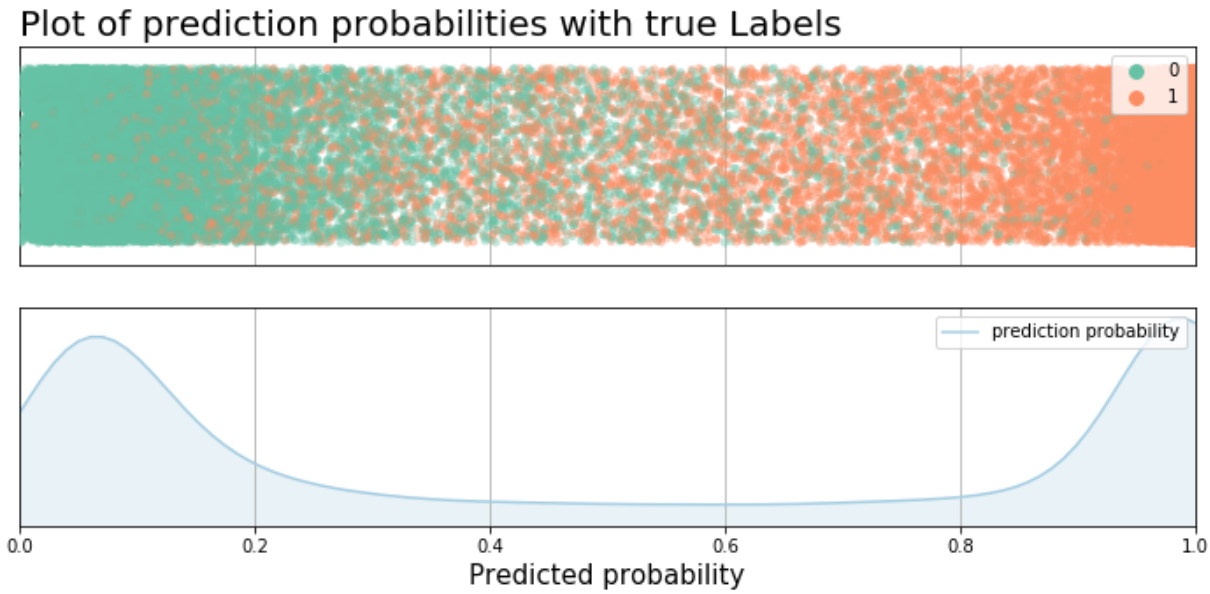
In [478]:
```
1 ▸ # xgboost↔
```

```
f1-score on training set: 0.9347634404705826
f1-score on test set: 0.9151271489529635
```

In [479]:
```
1 # evaluation metrics and visualization↔
```

```
classification_report
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     19708
           1       0.94      0.89      0.92     20831

    accuracy                           0.92     40539
   macro avg       0.92      0.92      0.92     40539
weighted avg       0.92      0.92      0.92     40539
```





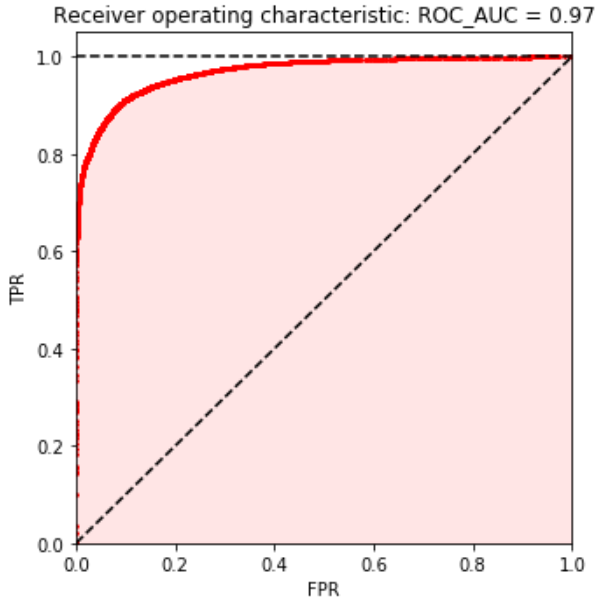

```
When recall is 75%, Precison is:  0.9935143383989318
```

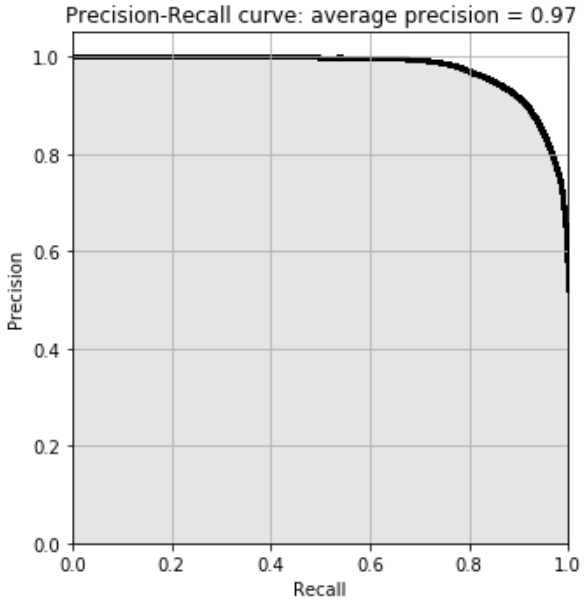
In [480]:
```
1 ▸ # lightGBM↔
```

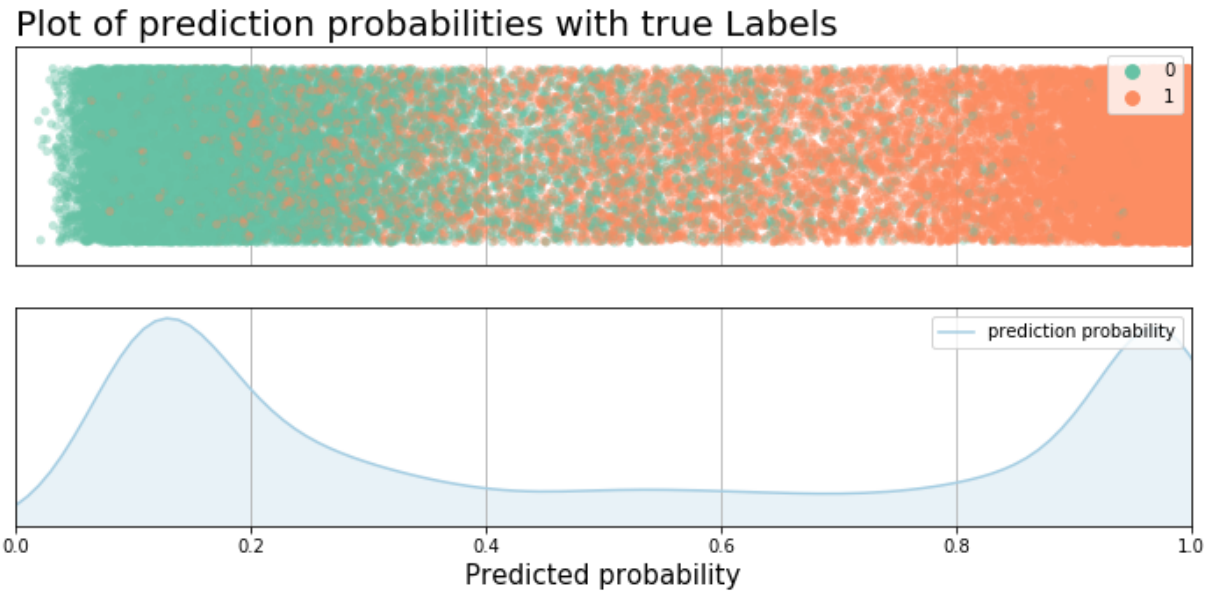
```
f1-score on training set: 0.9060396088259866
f1-score on test set: 0.901736414740787
```

In [481]:
```
1 ▸ # evaluation metrics and visualization↔
```

```
classification_report
              precision    recall  f1-score   support

           0       0.87      0.94      0.90     19708
           1       0.94      0.87      0.90     20831

    accuracy                           0.90     40539
   macro avg       0.90      0.90      0.90     40539
weighted avg       0.91      0.90      0.90     40539
```








```
When recall is 75%, Precison is:  0.9867382380802021
```

Compare them all

If you forgot about the performance, don't worry, I will sumerize here. The candidates are:

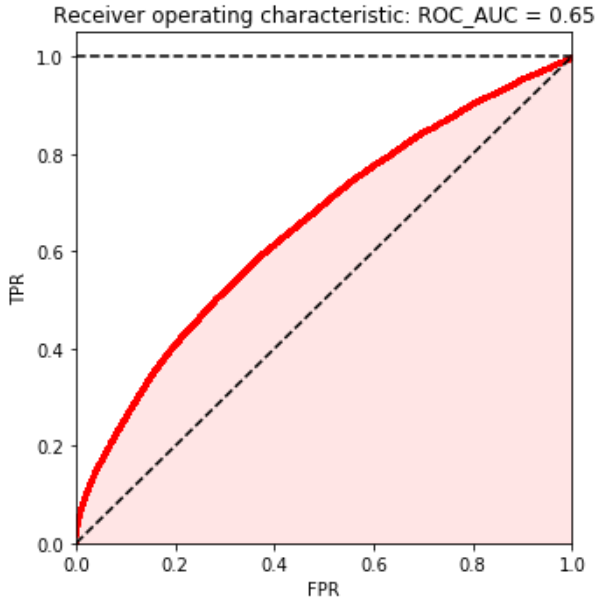
- Logistic regression without PCA
- Random forest without PCA
- XGBoost without PCA
- LightGBM without PCA

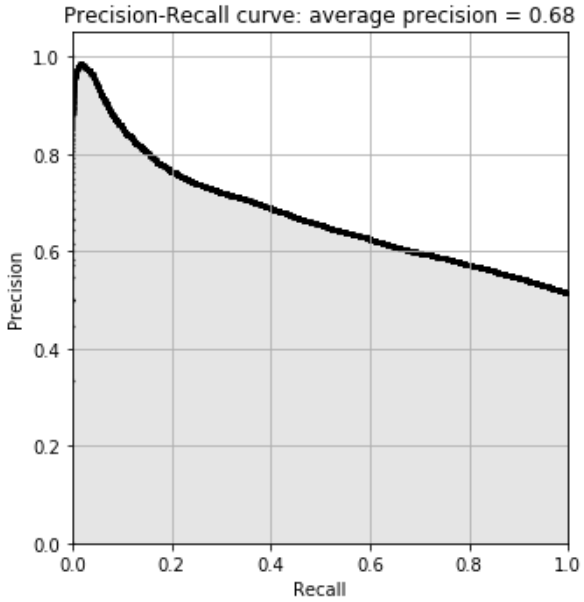
then

- Logistic regression with PC
- Random forest with PCA
- XGBoost with PCA
- LightGBM with PCA

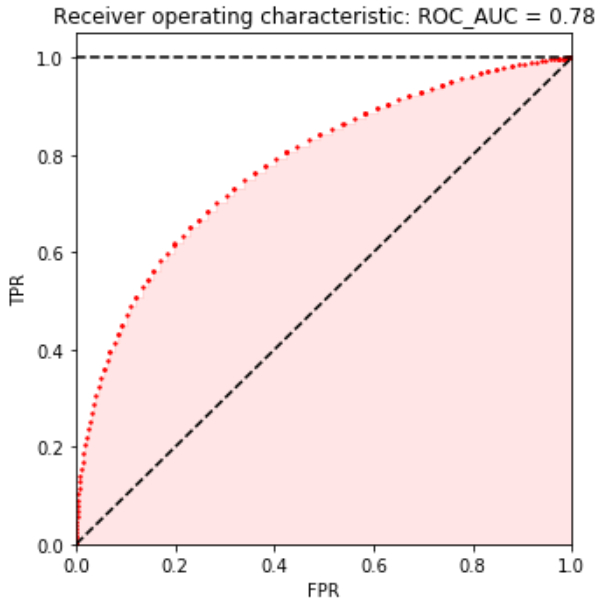
In [482]:
```
# comapre them all↔
```

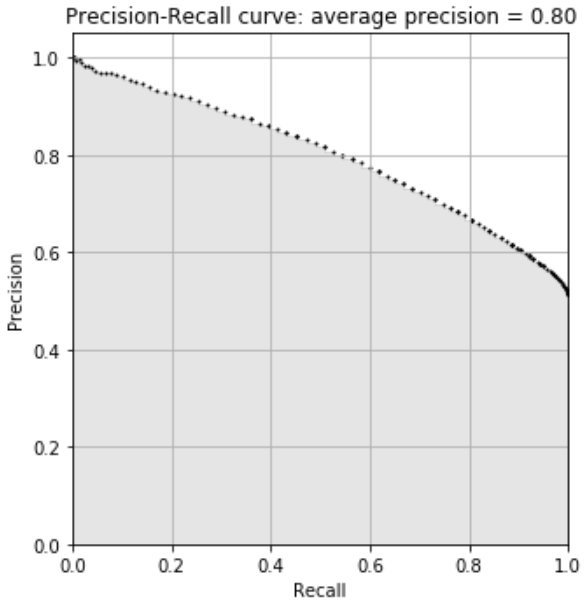
Logistic regression without PCA



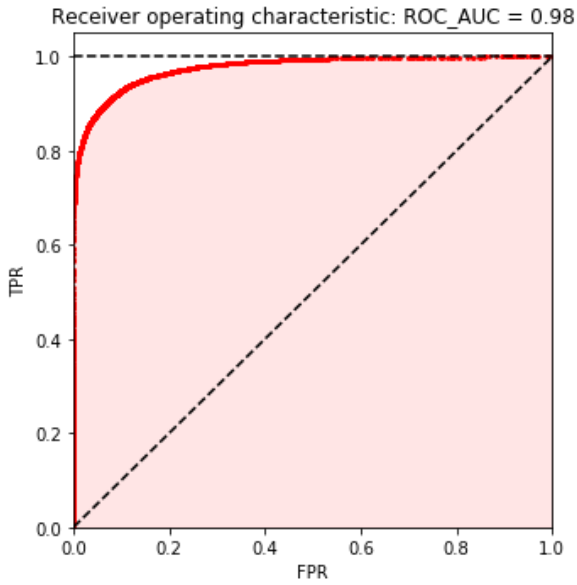


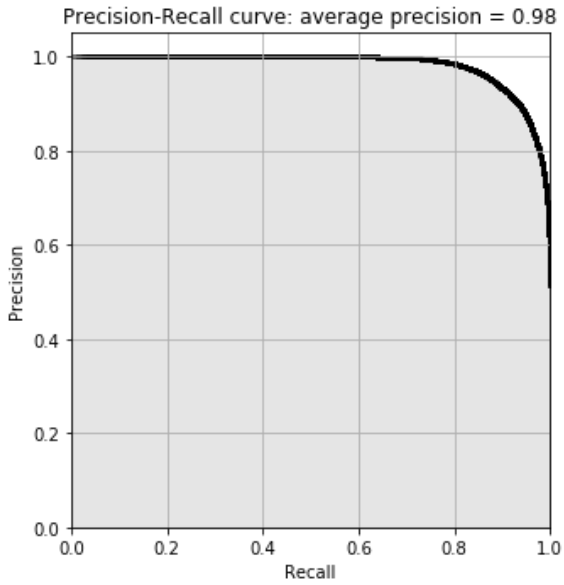
Random forest without PCA



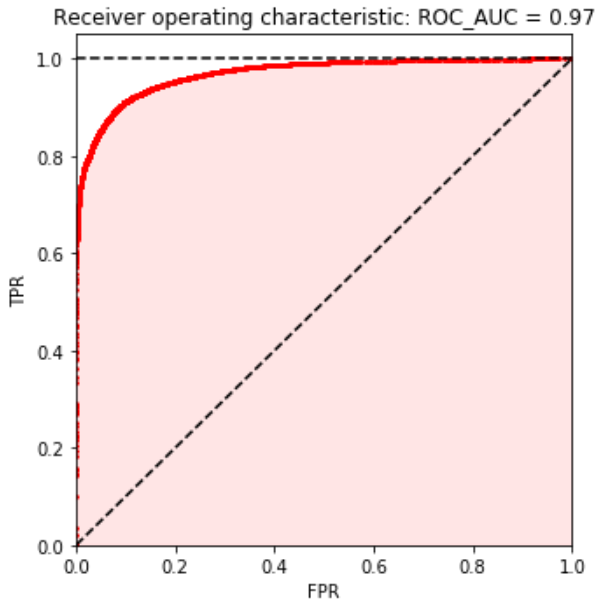


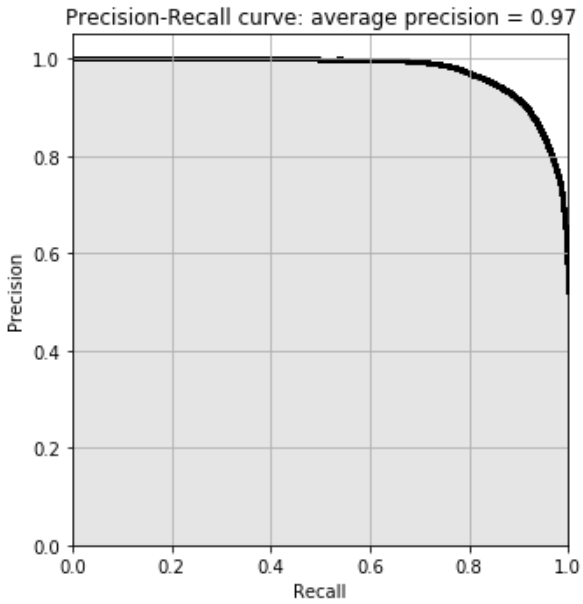
XGBoost without PCA



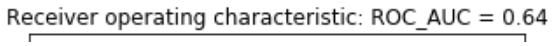


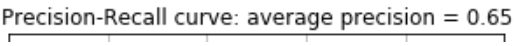
LightGBM without PCA



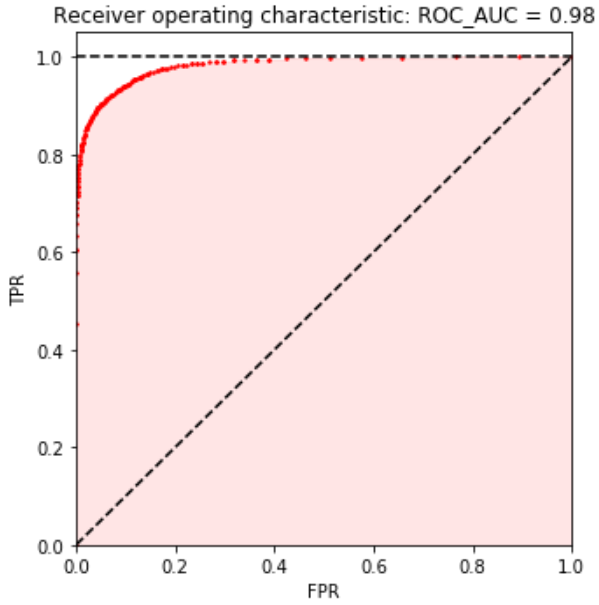


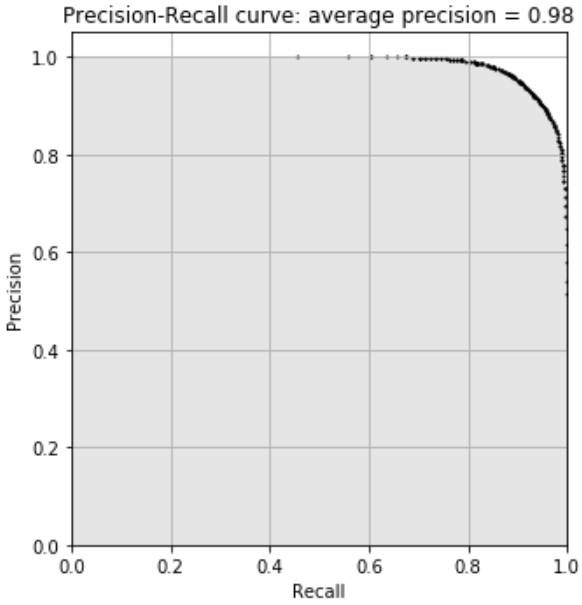
Logistic regression with PCA



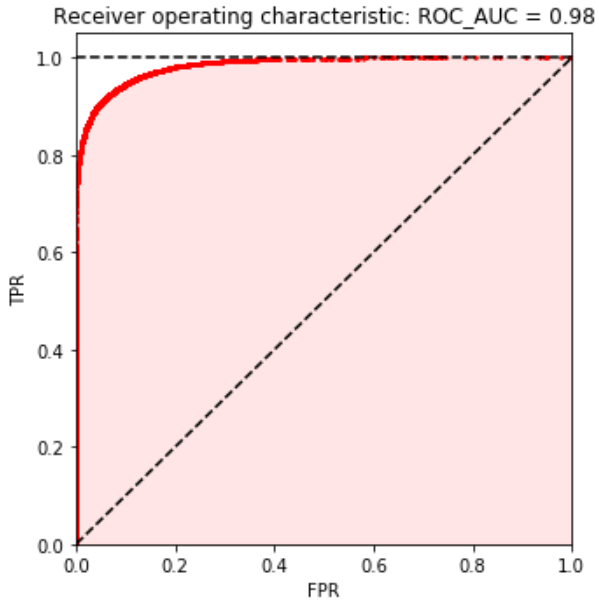


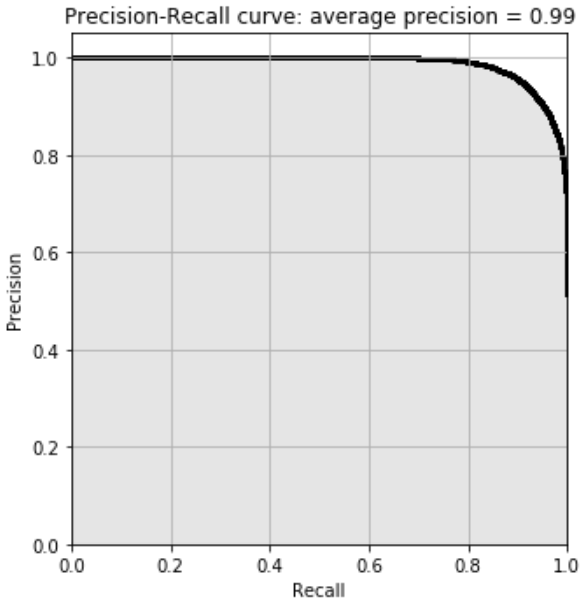
Random forest with PCA



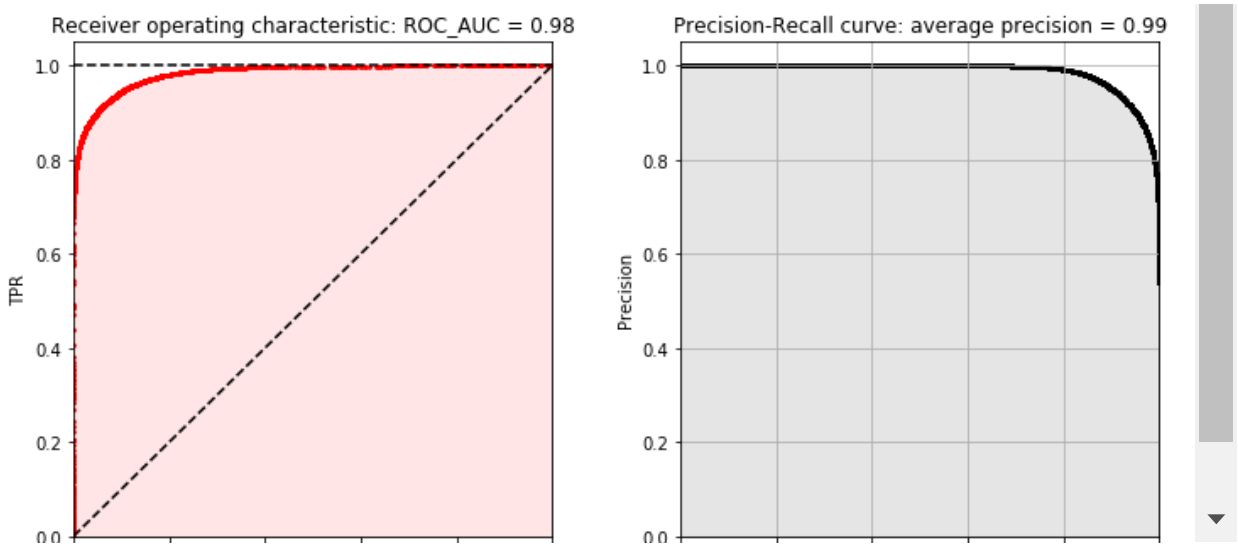


XGBoost with PCA





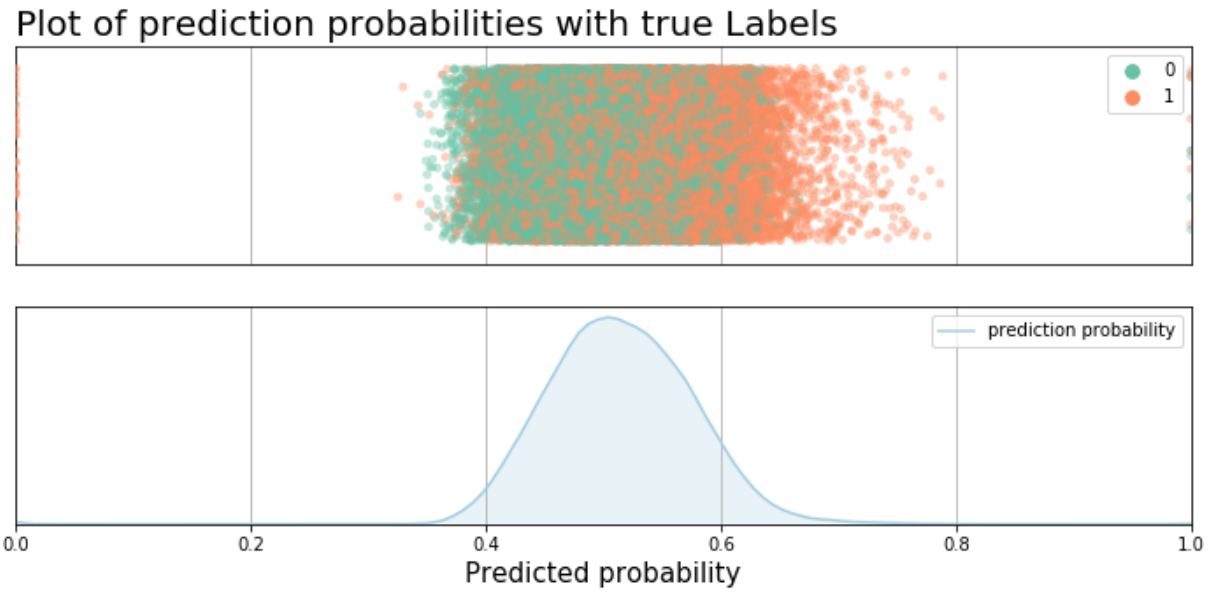
LightGBM with PCA



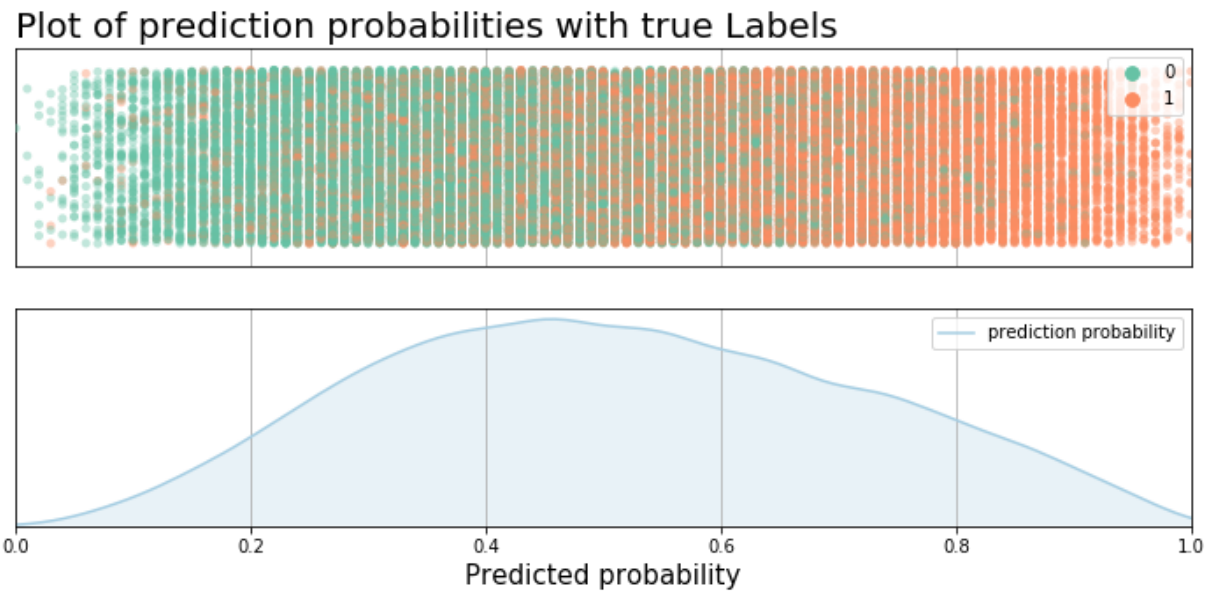
One more time, the prediction probability

In [484]:
```
1 ▸ # comapre↔
```

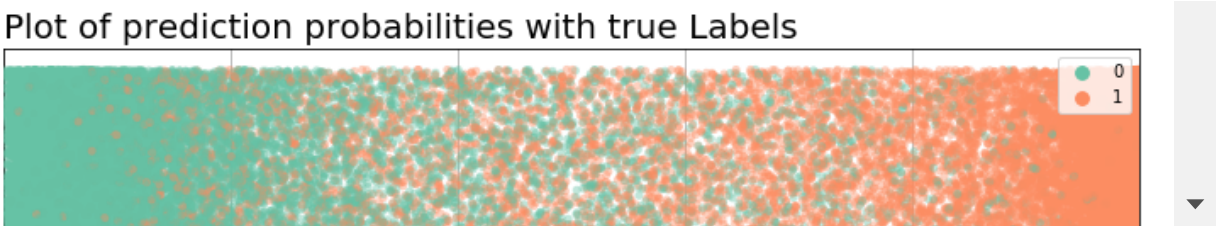
Logistic regress without PCA



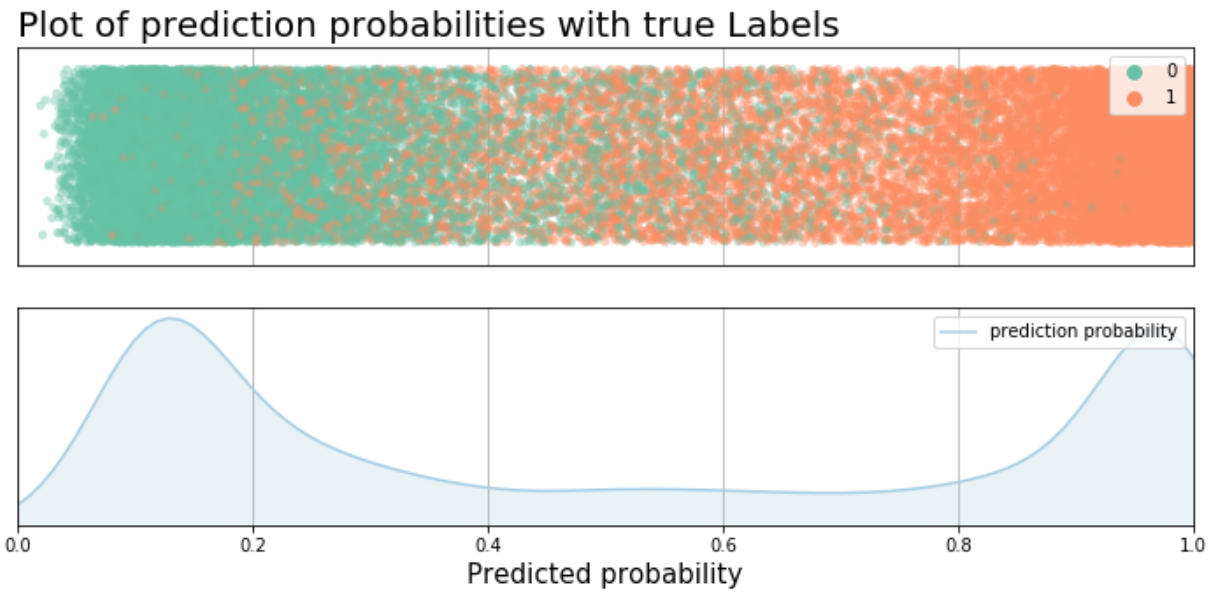
Random forest without PCA



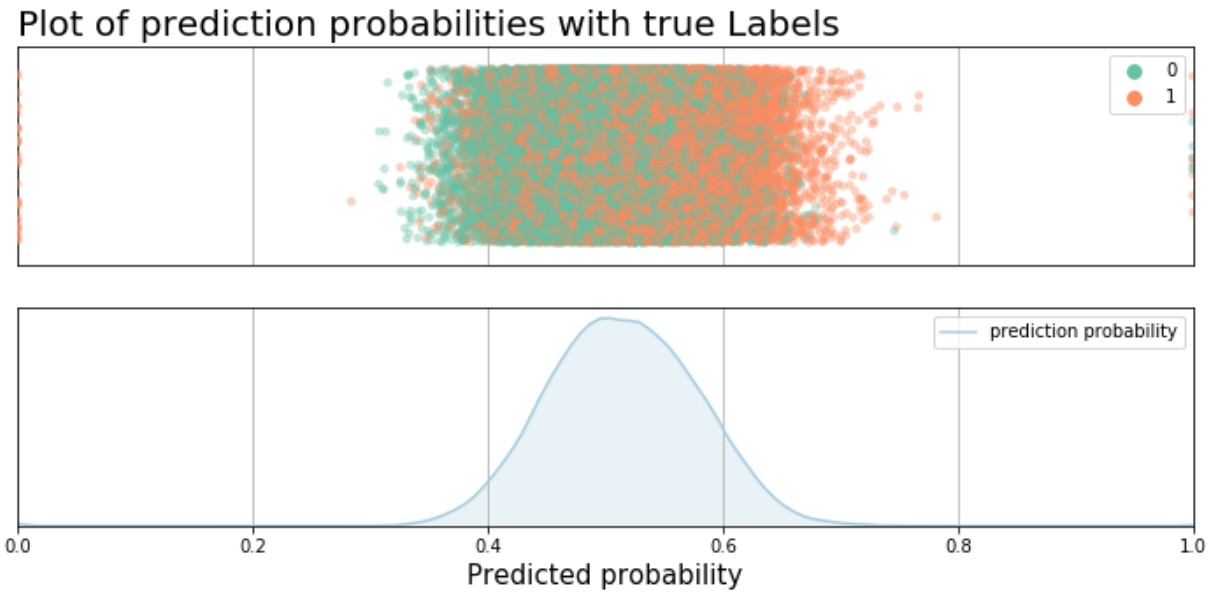
XGBoost without PCA



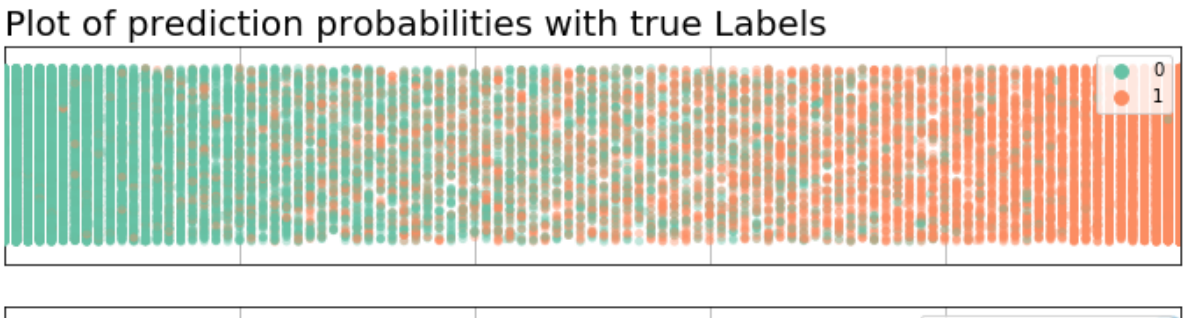
LightGBM without PCA



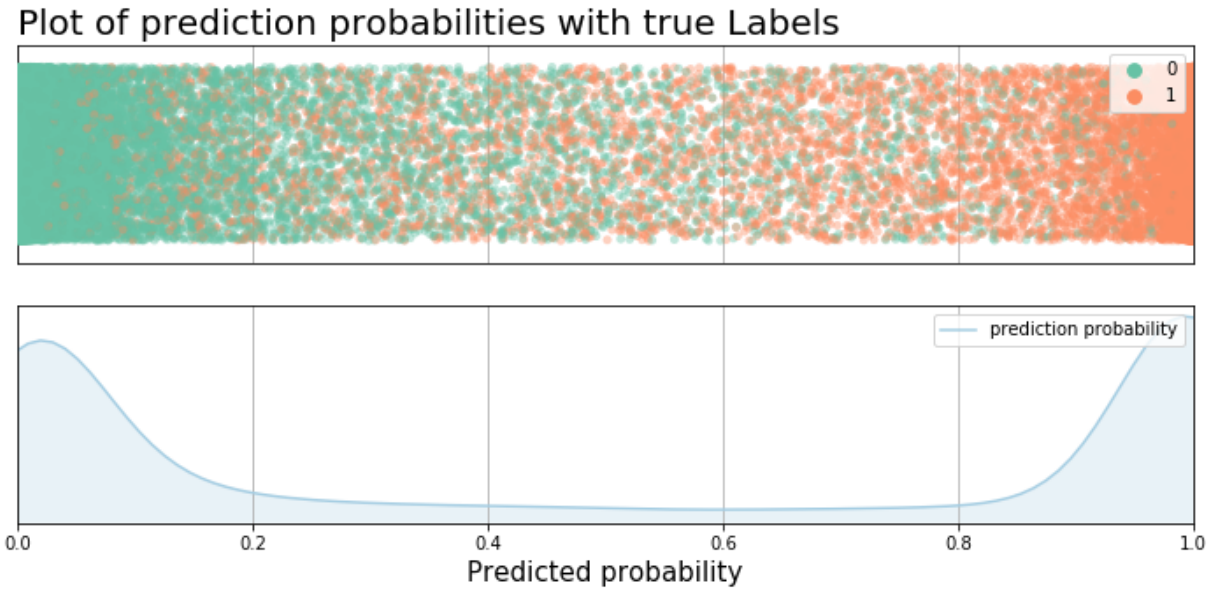
Logistic regress with PCA



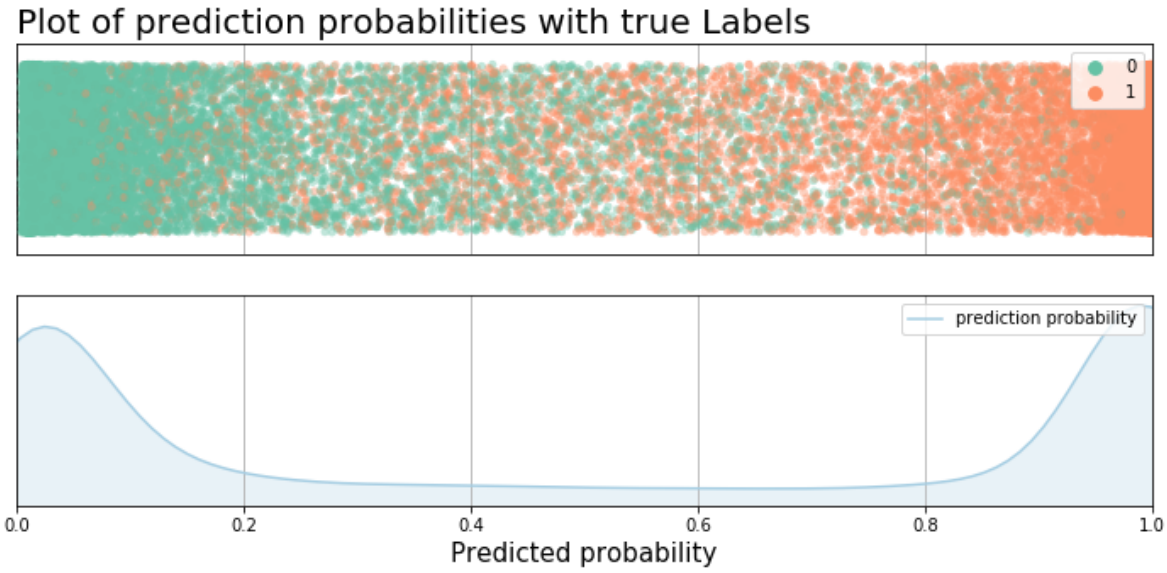
Random forest with PCA



XGBoost with PCA



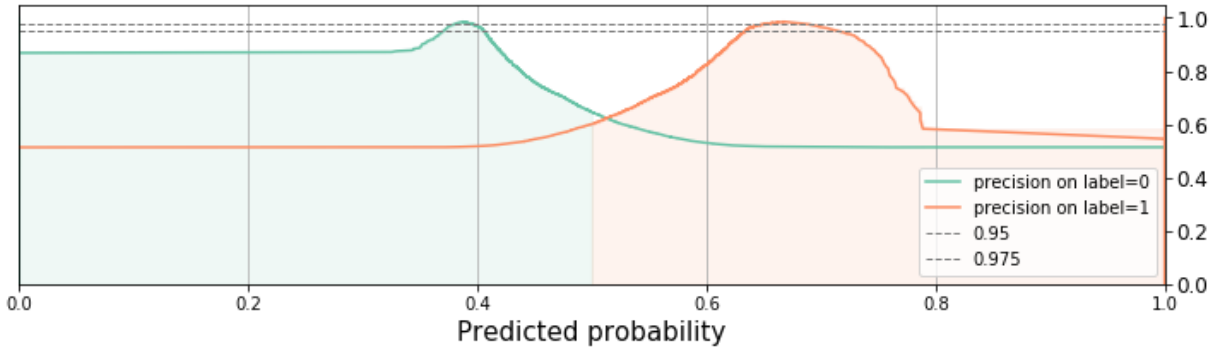
LightGBM with PCA



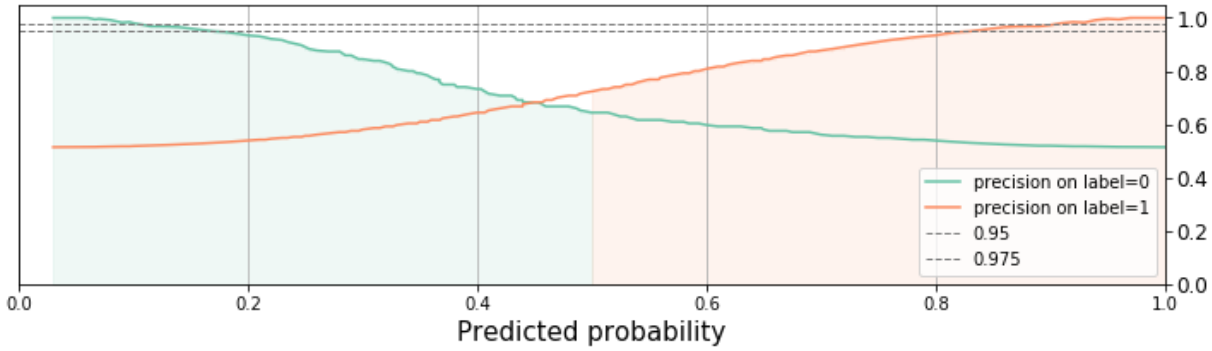
Sorry, I lied. One last comparison

In [485]:
```
1 ▸ # compare↔
```

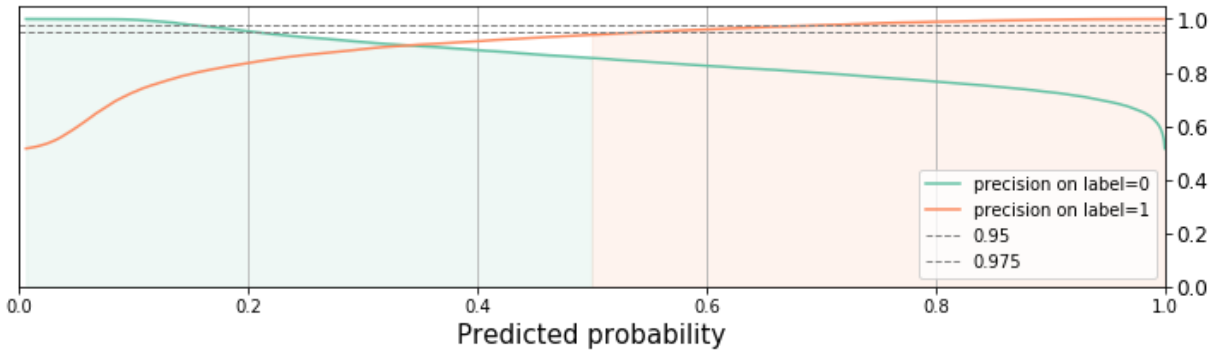
Logistic regress without PCA



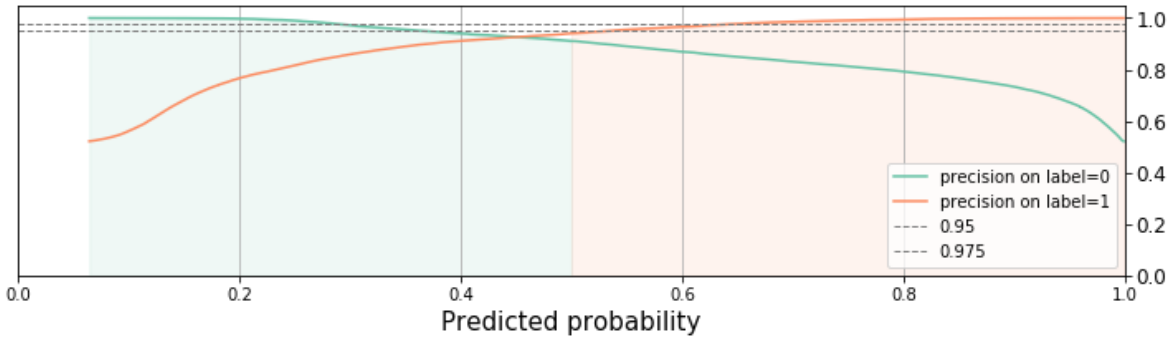
Random forest without PCA



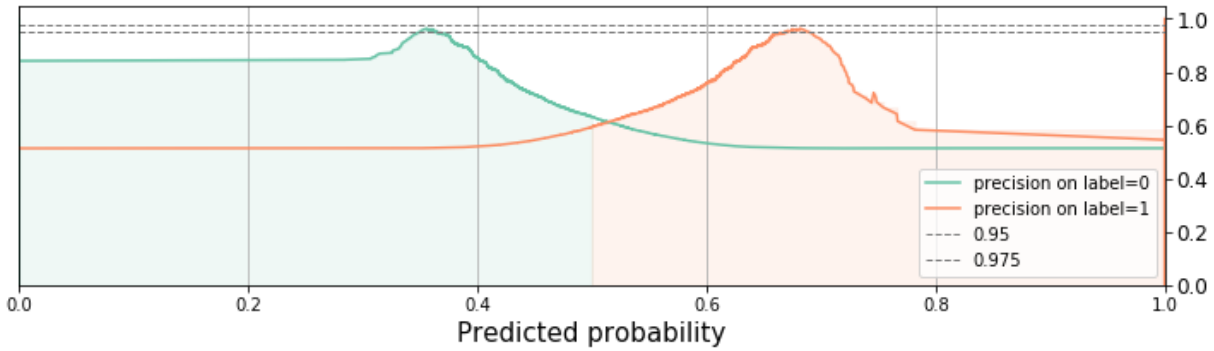
XGBoost without PCA



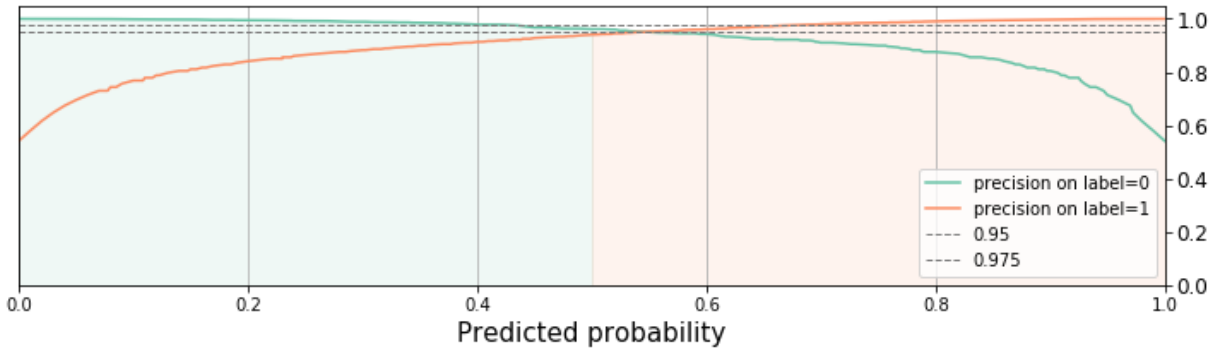
LightGBM without PCA



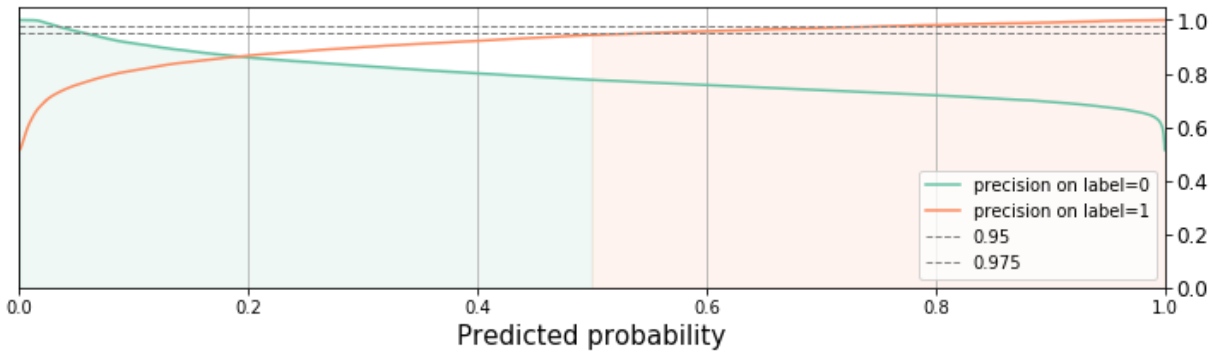
Logistic regress with PCA



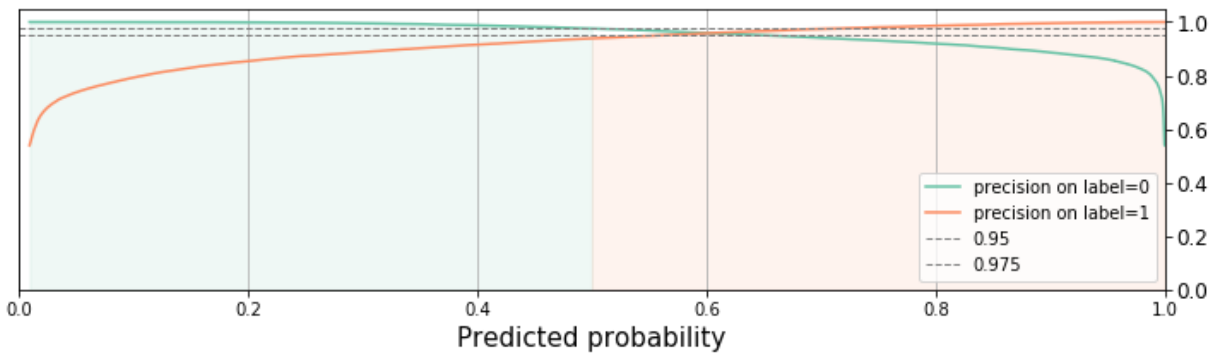
Random forest with PCA



XGBoost with PCA



LightGBM with PCA



It is better that we compare the result with and without PCA and visualize them side by side. But I am tired, I think you got the idea.

Interpretability

In [486]:
```
1 ▸ # check the feature importance, lightGBM↔
```

Out[486]:

	index	features	importance
0	10	RecordTime_min	1105
1	18	schedule_minute_num	1101
2	4	next_stop_Lat	242
3	9	RecordTime_hour	95
4	5	next_stop_Long	70
5	15	schedule_hour_num_overlap	64
6	11	RecordTime_sec	63
7	14	schedule_hour_num	52
8	7	RecordTime_day	44
9	6	RecordTime_month	38
10	0	OriginLat	38
11	8	RecordTime_weekday	37
12	2	DestinationLat	17
13	1	OriginLong	11
14	17	schedule_weekday_num	11
15	13	schedule_day	8
16	3	DestinationLong	3
17	12	schedule_month	1
18	16	schedule_day_num	0

In [487]:
```
1 ▸ # check the feature importance, xgboost using PCA features↔
```

Out[487]:

	index	features	importance
0	12	12	0.485462
1	8	8	0.208673
2	16	16	0.048021
3	10	10	0.032577
4	11	11	0.031507
5	5	5	0.024431
6	9	9	0.018190
7	4	4	0.017848
8	0	0	0.016979
9	13	13	0.015521
10	7	7	0.015500
11	2	2	0.014247
12	3	3	0.013560
13	14	14	0.012605
14	17	17	0.011456
15	18	18	0.009111
16	1	1	0.008907
17	6	6	0.007806
18	15	15	0.007598

Sumary

back to main

Feature engineering

Perform unsupervised "human"learning to extract hidden features, i.e. belonging areas data, bus stop GPS coordinates. Feature engineering the datetime data, and wraggling the unstandard shceduledArrivalTime features.

Data analysis and data visulization

Focus B6 service line, analyse the stats and provide insights about its operation. Adopt different data visulization tools/formations and apply aesthetic principles to tell a story about the data.

Machine learning

Explored the probability calibration applications (stated in my capstone 2 future work). Demonstrate the importance of PCA techniques, which can be considered as an unconventional unsupervised learning methods to generate patterns that cooperate with supervised learning or semi-supervised learning tasks.

future work

- Explore using unsupervised machine learning method to see if it can cluster the service line into different areas, or find hidden layers between areas layer and publicserveline layer.
- Expore and design tasks to see if unsupervised machine learning can feature out the bus stop latitude and longitude which I find out manually.
- Check Sklearn's probaility calibration package, since my approach might be too naive.
- Find way to imporve my coding capability.

Resource

New York City Bus Schedules: (https://new.mta.info/schedules/bus)

Brooklyn Bus Schedules: (https://new.mta.info/schedules/bus/Brooklyn)

realtime bus GPS: (https://bustime.mta.info/#b6)

coordinates to distance: (https://gps-coordinates.org/distance-between-coordinates.php)

color picker: (https://colorbrewer2.org/#type=diverging&scheme=PiYG&n=3) color picker2: (https://stackoverflow.com/questions/40673490/how-to-get-plotly-js-default-colors-list)

In [ ]:
```
1
```