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 chose New Youk city bus (https://www.kaggle.com/stoney71/new-york-city-transport-statistics) 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.

Note that I code-folded most of the shells. Unfold them as you wish.

Here is the main content of this project.

Table of Contents

In [18]:
```
# 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 [19]:
```
#  Load data↔
```

In [20]:
```
# data_head↔
```

Out[20]:

inLat	OriginLong	DestinationName	DestinationLat	DestinationLong	VehicleRef	VehicleLocation.Latitude	VehicleLocation.Longitude	NextStopPointName	ArrivalProximityText	DistanceFromStop	ExpectedArrivalTime	ScheduledArrivalTime
6104	-74.031143	BROWNSVILLE ROCKAWAY AV	40.656048	-73.907379	NYCT_430	40.635170	-73.960803	FOSTER AV/E 18 ST	approaching	76.0	2017-06-01 00:03:59	24:06:14
3169	-74.073494	S I MALL YUKON AV	40.575935	-74.167686	NYCT_8263	40.590802	-74.158340	MERRYMOUNT ST/TRAVIS AV	approaching	62.0	2017-06-01 00:03:56	23:58:02
5008	-73.880142	RIVERDALE 263 ST	40.912376	-73.902534	NYCT_4223	40.886010	-73.912647	HENRY HUDSON PKY E/W 235 ST	at stop	5.0	2017-06-01 00:03:56	24:00:53
1748	-73.802399	ROSEDALE LIRR STA via MERRICK	40.666012	-73.735939	NYCT_8422	40.668002	-73.729348	HOOK CREEK BL/SUNRISE HY	< 1 stop away	267.0	2017-06-01 00:04:03	24:03:00
1187	-73.909340	MOTT HAVEN 136 ST via CONCOURSE	40.809654	-73.928360	NYCT_4710	40.868134	-73.893032	GRAND CONCOURSE/E 196 ST	at stop	11.0	2017-06-01 00:03:56	23:59:38

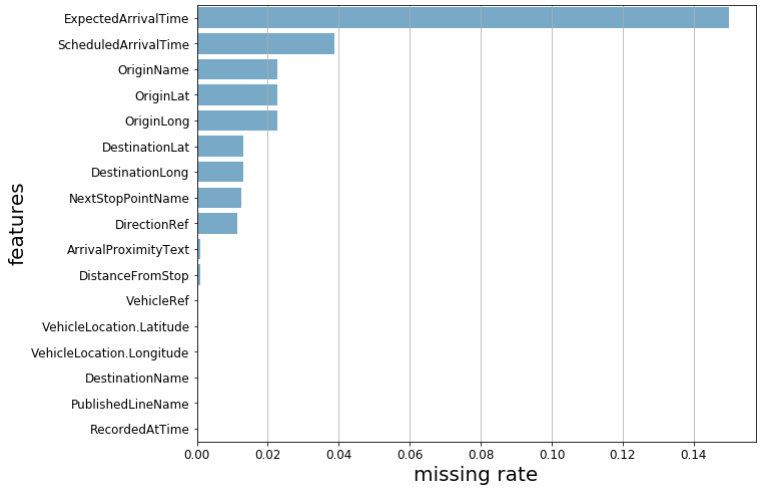
In [21]:
```
# data_info↔
```

```
<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

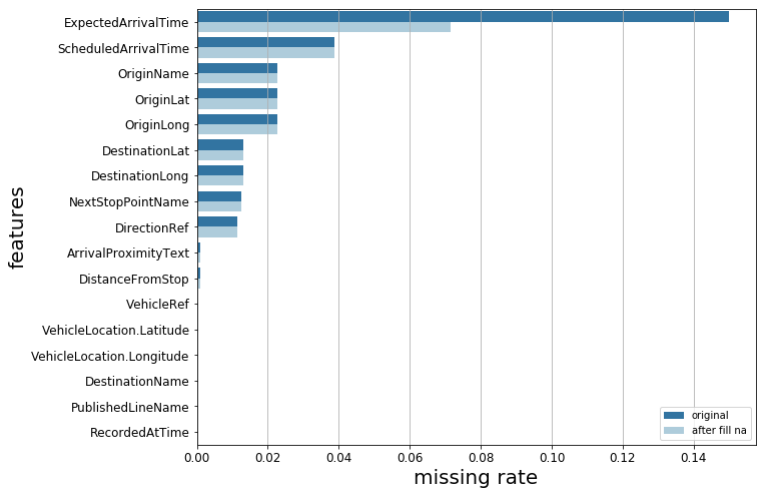
In [22]:
```
# null value percentage↔
```



In [23]:
```
# a simple logic to fill nan↔
```

In [24]:
```
df_fillna = get_df_fillna()
```

In [25]:
```
# 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 [26]:
```
# detail alart: how I fill na↔
```

Exploratory Data Analysis (EDA)

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 the meaning indicated by the feature names, i.e. 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' indicate the name of the service bus line (i.e. B6, B8, etc.), 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 labeled '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 analysing. But for individual bus that is on service, the next bus stop is changing all the time. Also 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 [27]:
```
# 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 [28]:
```
# df_eda↔
```

In [29]:
```
for col in df_eda.columns:
    print(col, df_eda[col].nunique())
```

```
RecordedAtTime 938045
DirectionRef 2
PublishedLineName 334
OriginName 629
OriginLat 1166
OriginLong 1266
DestinationName 848
DestinationLat 962
DestinationLong 1022
VehicleRef 5974
VehicleLocation.Latitude 386725
VehicleLocation.Longitude 497810
NextStopPointName 11286
ArrivalProximityText 212
DistanceFromStop 24684
ExpectedArrivalTime 2912670
ScheduledArrivalTime 95826
```

In [30]:
```
# take a look at the DirectionRef
df_eda.DirectionRef.unique()[:50]
```

Out[30]:
```
array([ 0.,  1., nan])
```

In [91]:
```
# take a look at the ArrivalProximityText
df_eda.ArrivalProximityText.unique()[:20]
```

Out[91]:
```
array(['approaching', 'at stop', '< 1 stop away', '0.6 miles away',
       '0.9 miles away', '5.2 miles away', '1.1 miles away',
       '2.5 miles away', '4.4 miles away', '1.5 miles away',
       '0.7 miles away', '2.7 miles away', '2.1 miles away',
       '0.8 miles away', '1.2 miles away', nan, '3.7 miles away',
       '0.5 miles away', '2.3 miles away', '2.6 miles away'], dtype=object)
```

In [32]:
```
# take a look at the line services
df_eda.PublishedLineName.unique()[:50]
```

Out[32]:
```
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'], dtype=object)
```

In [33]:
```
# group the lines, ↔
```

```
Bronx:  ['Bx10' 'Bx1' 'Bx28' 'Bx39' 'Bx31' 'Bx36' 'Bx5' 'Bx32' 'Bx17' 'Bx4' 'Bx19'
 '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' 'S40' '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' 'X2'
 'X21' 'X22' 'X22A' 'X27' 'X28' 'X3' 'X30' 'X31' 'X37' 'X38' 'X4' 'X42'
 'X5' 'X63' 'X64' 'X68' 'X7' 'X8' 'X9']
```

We found a hidden layer

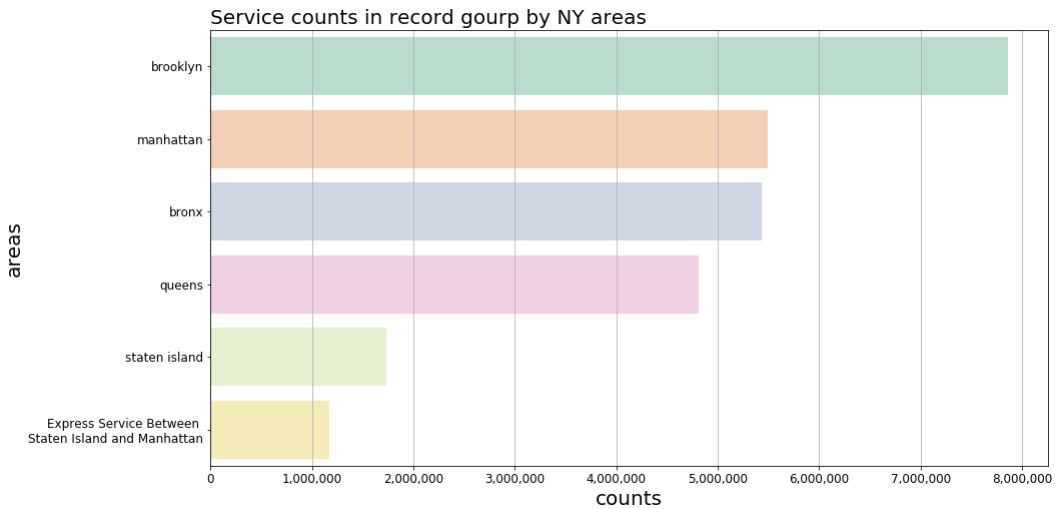
As you can see, we found a hidden layer of the dataset -- areas, i.e. Bronx, Broolklyn, etc.

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. Even though there could be trasnportation service between areas, i.e. "Express Service Between Staten Island and Manhattan".

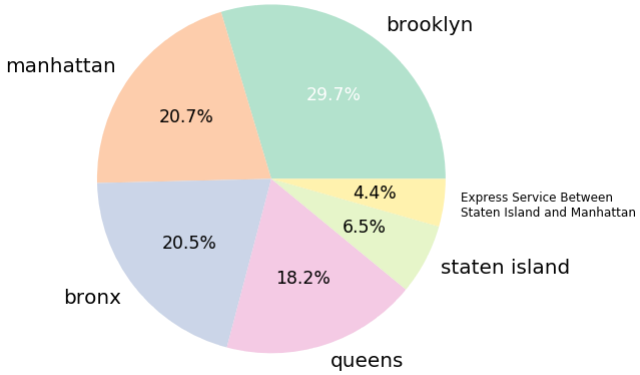
The goal of such grouping is to help finding the underlying pattern. And normally this kind of feature engineering is very efficient and is favored over using machine learning clustering methods in interpretability sense.

In [34]:
```
# A brief stats of line counts among bronx, brooklyn, manhattan, etc.↔
```

In [97]:
```
# plot counts stats by areas
from matplotlib.ticker import StrMethodFormatter
f, ax = plt.subplots(figsize=(15, 8))
sns.barplot(x=df_line_count_by_areas.counts, y=df_line_count_by_areas.areas,
            label="Service counts in record",  ax=ax,
           palette="Pastel2",
           )
ax.grid(axis='x')
ax.set_title("Service counts in record gourp by NY areas", loc='left', fontsize=20)
ax.xaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}'))

ax.set_xlabel(ax.get_xlabel(), fontsize=20)
# ax.set_xlabel('missing rate', fontsize=20)
ax.set_ylabel(ax.get_ylabel(), fontsize=20)
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)

plt.show()
```



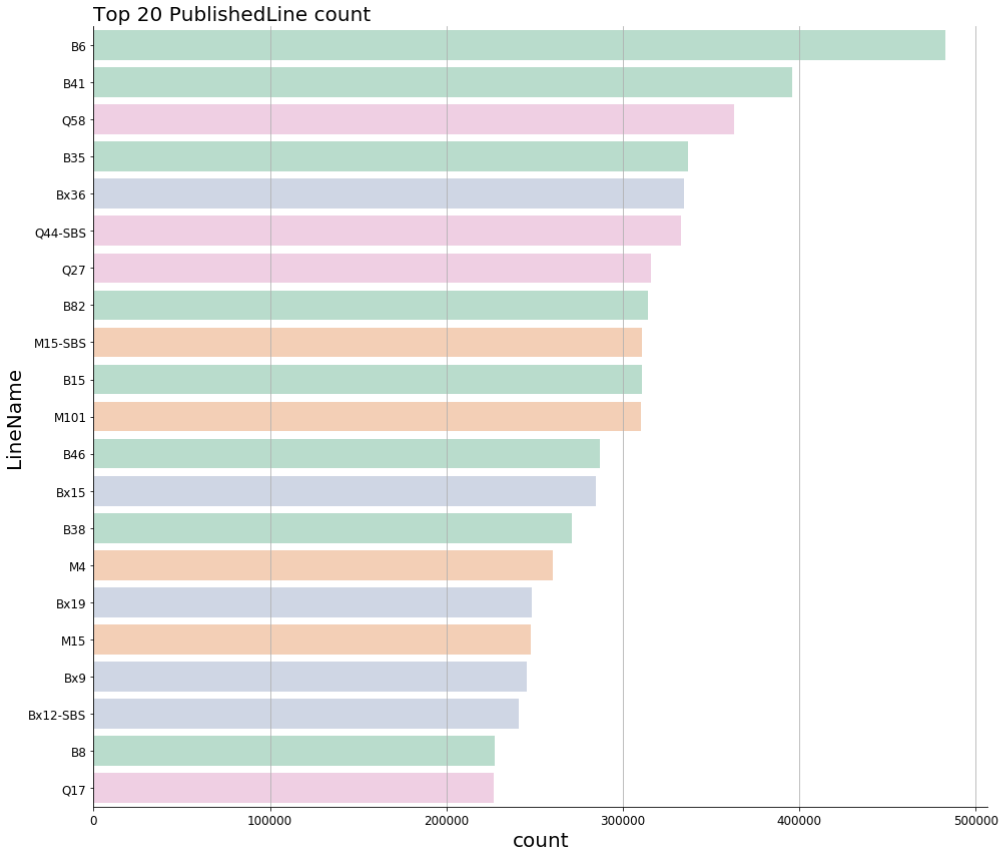
In [36]:
```
# take another look↔
```



In [37]:
```
# df_line_count.head(10)
```

In [38]:
```
# df_line_count_by_areas ↔
```

In [96]:
```
# visulize PublishedLineName↔
```



In [ ]:
```
1
```

In [ ]:
```
1
```

NEW FORK HERE

Group Segmentation Using Clustering

As mentioned before, we seperate the bus lines according to the areas, mainly based on the the service line names, i.e. 'B' in 'B6' indicate Brooklyn, 'Bx' in 'Bx5' indecates Bronx, etc. This is a naive approach of group segmentation, and we have very good reason to do that because the bus lines within the same area would operate similar.

an unsupervised learning approach to identify the underlying structure in data and grouping points based on similarity. These groups (known as clusters) should be homogeneous and distinct. In other words, the members within a group should be very similar to each other and very distinct from members of any other group.

From an applied perspective, the ability to segment members into groups based on similarity and without any guidance from labels is very powerful. For example, such a technique could be applied to find different consumer groups for online retailers, customizing a marketing strategy for each of the distinct groups (i.e., budget shoppers, fashionistas, sneakerheads, techies, audiophiles, etc.).

Following that idea, we can apply machine learning for group segmentation on the New York Bus Tranit data. i.e. we will group the bus line in Brooklyn.

Note: for demonstration purpose, I will work on a row=10,000 sample data. While the total sample size of Broolyn is 7M.

In [40]:
```
# customer function: get target sub-dataset↔
```

In [41]:
```
# df_fillna[df_fillna.PublishedLineName.isin(lines_brooklyn)].count()
```

In [42]:
```
df_brooklyn_cluster_trim = reset_dataset_cluster()

```

In [43]:
```
# patch add_ # feature engineering add next stop lat and long↔
```

In [44]:
```
df_cluster_trim= add_feature_location_cluster(df_brooklyn_cluster_trim)
# df_cluster_trim.head(5)
```

In [45]:
```
# parse recordtime to hour, min, sec ↔
```

In [46]:
```
# customer function create target↔
```

In [47]:
```
# feature engineering↔
```

In [48]:
```
# df_work_on_cluster.head()
```

In [49]:
```
# create df_X_cluster↔
```

In [50]:
```
df_X_cluster.head()
```

Out[50]:

	OriginLat	OriginLong	DestinationLat	DestinationLong	next_stop_Lat	next_stop_Long	RecordTime_month	RecordTime_day	RecordTime_weekday	RecordTime_hour	RecordTime_min	RecordTime_sec	schedule_month	sched
2984636	40.664654	-73.862892	40.704865	-73.902145	40.705175	-73.901876	6	14	2	12	15	26	6	
14760540	40.664654	-73.862894	40.704864	-73.902143	40.657654	-73.883346	10	8	6	0	0	18	10	
455878	40.704739	-73.902275	40.664642	-73.863068	40.684998	-73.912419	6	2	4	18	56	29	6	
1102526	40.664654	-73.862892	40.704865	-73.902145	40.668803	-73.868576	6	6	1	7	31	58	6	
7334043	40.664654	-73.862892	40.678780	-73.903442	40.667086	-73.871573	8	3	3	12	36	26	8	

In [51]:
```
df_X_cluster.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 2984636 to 23457682
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   OriginLat                 10000 non-null  float64
 1   OriginLong                10000 non-null  float64
 2   DestinationLat            10000 non-null  float64
 3   DestinationLong           10000 non-null  float64
 4   next_stop_Lat             10000 non-null  float64
 5   next_stop_Long            10000 non-null  float64
 6   RecordTime_month          10000 non-null  int32
 7   RecordTime_day            10000 non-null  int32
 8   RecordTime_weekday        10000 non-null  int32
 9   RecordTime_hour           10000 non-null  int64
 10  RecordTime_min            10000 non-null  int64
 11  RecordTime_sec            10000 non-null  int64
 12  schedule_month            10000 non-null  int64
 13  schedule_day              10000 non-null  int64
 14  schedule_hour_num         10000 non-null  int64
 15  schedule_hour_num_overlap 10000 non-null  int64
 16  schedule_day_num          10000 non-null  int64
 17  schedule_weekday_num      10000 non-null  int64
 18  schedule_minute_num       10000 non-null  int64
dtypes: float64(6), int32(3), int64(10)
memory usage: 1.4 MB
```

In [95]:
```
# df_y_cluster.head()
```

In [32]:
```
# check labels
print(len(df_y_cluster.PublishedLineName.unique()))
df_y_cluster.PublishedLineName.unique()
```

```
54
```

Out[32]:
```
array(['B25', 'B46', 'B3', 'B11', 'B46-SBS', 'B38', 'B1', 'B68', 'B15',
       'B26', 'B61', 'B14', 'B49', 'B82', 'B48', 'B44', 'B54', 'B7',
       'B43', 'B63', 'B35', 'B57', 'B6', 'B8', 'B9', 'B60', 'B62', 'B70',
       'B47', 'B41', 'B13', 'B16', 'B52', 'B45', 'B20', 'B31', 'B44-SBS',
       'B83', 'B42', 'B12', 'B32', 'B4', 'B36', 'B17', 'B37', 'B84',
       'B24', 'B65', 'B74', 'B67', 'B2', 'B64', 'B39', 'B69'],
      dtype=object)
```

In [168]:
```
# label encoding, ↔
```

Next we will perform clustering. you might have realized I choose a label 'PublishedLineName' as a guidance. So it is more tangible to evaluate the performance.

Before clustering, let's scale and perform PCA on the data.

In [34]:
```
# df_X_train = df_X_cluster.copy()
```

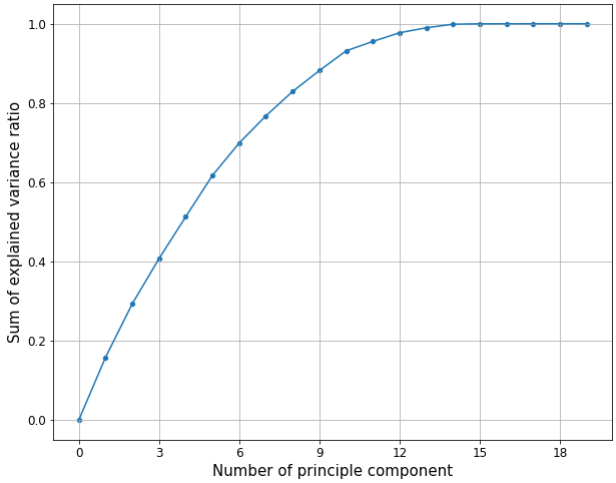
In [56]:
```
# preprocessing scaling↔
```

In [57]:
```
# pca to handle correlation and sparsity↔
```

In [58]:
```
pca.explained_variance_ratio_.cumsum()
```

Out[58]:
```
array([0.15806076, 0.29317733, 0.40703518, 0.51278225, 0.61714902,
       0.69890663, 0.7675316 , 0.82913461, 0.88174704, 0.93153002,
       0.95526602, 0.97727202, 0.98975462, 0.99878586, 0.99980325,
       0.9999218 , 1.        , 1.        , 1.        ])
```

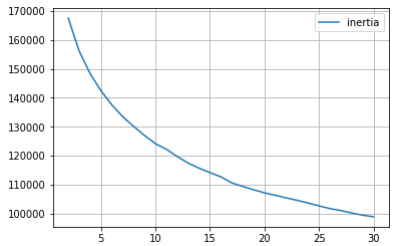
In [59]:
```
# plot explained_variance cumsum↔
```



In [60]:
```
# choose the top PCA components with threshold = 0.99, here I kept them all without touch (# is_cut = False)↔
```

In [64]:
```
# K-means - Inertia as the number of clusters varies
from sklearn.cluster import KMeans

n_clusters = 10
n_init = 10
max_iter = 300
tol = 0.0001
random_state = RANDOM_STATE
n_jobs = -1

range_tmp = range(2,31)
kMeans_inertia = pd.DataFrame(data=[],index=range_tmp, \
                              columns=['inertia'])
for n_clusters in range_tmp:
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state, \
                n_jobs=n_jobs)

    cutoff = cut_off
    kmeans.fit(df_X_train_pca_cluster.loc[:,0:cutoff])
    kMeans_inertia.loc[n_clusters] = kmeans.inertia_
```

In [65]:
```
kMeans_inertia.plot()
plt.grid()
```



As it shows, the inertia decreases as the number of clusters increases. This makes sense. The more clusters we have, the greater the homogeneity among observations within each cluster. However, fewer clusters are easier to work with than more, so finding the right number of clusters to generate is an important consideration when running k-means.

Goodness of the Clusters

Before we build our first clustering application, let's introduce a function to analyze the goodness of the clusters we generate using the clustering algorithms. Specifically, we will use the concept of homogeneity to assess the goodness of each cluster.

If the clustering algorithm does a good job separating the bus lines in the New York Bus Transit dataset, each cluster should have bus lines that are very similar to each other and dissimilar to those in other groups.

Presumably, bus lines that are similar to each other and grouped together should have similar operation schedule and GPS info—in other words, their profile to manage them should be similar.

If this is the case (and with real-world problems, a lot of these assumptions are only partially true), bus lines in a given cluster should generally be assigned to the same class, which we will validate using the encoded 'PublishedLineName' we set aside in df_y_encode.

The higher the percentage of bus lines that have the most frequently 'PublishedLineName' and every cluster, the better the clustering application.

As an example, consider a cluster with one hundred bus lines. If 30 borrowers have a PublishedLineName=="B1", 25 bus lines have a PublishedLineName=='B6", 20 bus lines have PublishedLineName=="B20", and the remaining borrowers have some other names, we would say that the cluster has a 30% accuracy, given that the most frequently occuring loan grade for that cluster applies to just 30% of the bus lines in that cluster.

Also plot sklearn.metrics.homogeneity_score and sklearn.metrics.completeness_score

In [69]:
```
from sklearn.metrics import homogeneity_score, completeness_score
```

In [67]:
```
# customer func to evaluate goodness of clusters
def analyzeCluster(clusterDF, labelsDF):↔
```

Accuracy plot

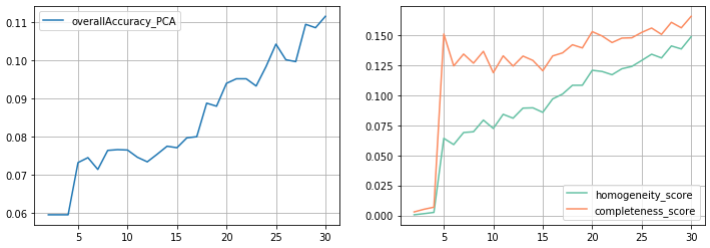
maybe it is more intuitive to use accuraccy for evaluation. here is the accuracy-number of cluster plot.

I will use overallAccuracy for evaluation. The formulat is given by:

overallAccuracy = countMostFrequent.sum()/ clusterCount.sum()

In [70]:
```
# K-means - Accuracy as the number of clusters varies↔
```

In [71]:
```
# plot result↔
```



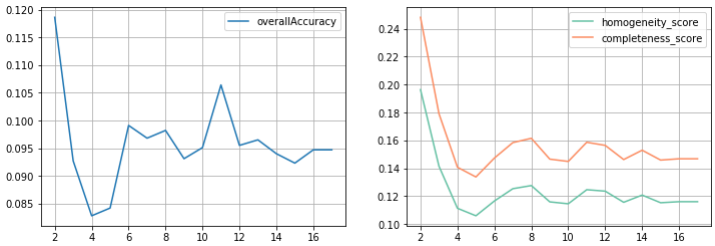
As mentioned earlier, when the number of clusters increse, the accuracy would increase. But since it is not practical to work on too many clusters, also the accuracy around 25 clusters are more stable. Personaly I would choose low varaince over low bias. So, I will suggest to take n=20 cluster.

number of PCA component

It is also worthy exploring the overal accuracy over number of PCA component chosen

In [82]:
```
# K-means - Accuracy as the number of components varies

n_clusters = 20
n_init = 10
max_iter = 300
tol = 0.0001
random_state = RANDOM_STATE
n_jobs = -1

cutoff_list = range(2,18)
kMeans_inertia = pd.DataFrame(data=[],index=cutoff_list,columns=['inertia'])

overallAccuracy_kMeansDF = pd.DataFrame(data=[],index=cutoff_list, \
                    columns=['overallAccuracy'])

homogeneity_score_list = []
completeness_score_list = []
for cutoffNumber in cutoff_list:
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state, \
                n_jobs=n_jobs)

    cutoff = cutoffNumber
    kmeans.fit(df_X_train_pca_cluster.loc[:,0:cutoff])
    kMeans_inertia.loc[cutoff] = kmeans.inertia_
    X_train_kmeansClustered = kmeans.predict(df_X_train_pca_cluster.loc[:,0:cutoff])
    X_train_kmeansClustered = pd.DataFrame(data=X_train_kmeansClustered, \
                              index=df_X_train_pca_cluster.index, columns=['cluster'])

    countByCluster_kMeans, countByLabel_kMeans, countMostFreq_kMeans, \
        accuracyDF_kMeans, overallAccuracy_kMeans, accuracyByLabel_kMeans \
        = analyzeCluster(X_train_kmeansClustered, df_y_cluster)

    overallAccuracy_kMeansDF.loc[cutoff] = overallAccuracy_kMeans

    homogeneity_score_list.append(homogeneity_score(labels_true=df_y_cluster.PublishedLineName.values,\
                                                    labels_pred=X_train_kmeansClustered.cluster.values))
    completeness_score_list.append(completeness_score(labels_true=df_y_cluster.PublishedLineName.values,\
                                                      labels_pred=X_train_kmeansClustered.cluster.values))
```

In [83]:
```
fig, axes = plt.subplots(1,2, figsize=(12,4))
ax = axes[0]
overallAccuracy_kMeansDF.plot(ax=ax)
ax.grid()

ax = axes[1]
# cutoff_list = cutoff_list
sns.lineplot(cutoff_list, homogeneity_score_list, label='homogeneity_score', ax=ax, color = sns.color_palette("Set2")[0])
sns.lineplot(cutoff_list, completeness_score_list, label='completeness_score', ax=ax,color = sns.color_palette("Set2")[1])
plt.legend()
ax.grid()
plt.show()
```

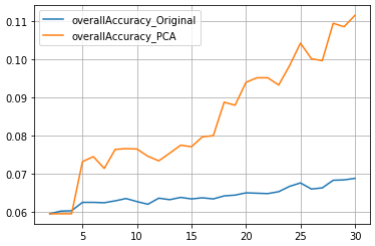


This dataset doesn't have too many features (18 columns). But we still observe that the best cutoff number of PCA is around 10, which is associated with 0.99 cumulated sum of explained variance. So, it is we don't take all the components even not for dimenionsality reduction reason.

try original data

In [72]:
```
# K-means - Accuracy as the number of clusters varies

# n_clusters = 5
n_init = 10
max_iter = 300
tol = 0.0001
random_state = RANDOM_STATE
n_jobs = -1

range_tmp = range(2,31)
kMeans_inertia = \
    pd.DataFrame(data=[],index=range_tmp,columns=['inertia'])
overallAccuracy_kMeansDF_original = \
    pd.DataFrame(data=[],index=range_tmp,columns=['overallAccuracy_Original'])


for n_clusters in range_tmp:
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state, \
                n_jobs=n_jobs)

#     cutoff = 10
    kmeans.fit(df_X_cluster)
    kMeans_inertia.loc[n_clusters] = kmeans.inertia_
    X_train_kmeansClustered = kmeans.predict(df_X_cluster)
    X_train_kmeansClustered = \
        pd.DataFrame(data=X_train_kmeansClustered, index=df_X_cluster.index, \
                     columns=['cluster'])

    countByCluster_kMeans, countByLabel_kMeans, countMostFreq_kMeans, \
        accuracyDF_kMeans, overallAccuracy_kMeans, accuracyByLabel_kMeans \
        = analyzeCluster(X_train_kmeansClustered, df_y_cluster)

    overallAccuracy_kMeansDF_original.loc[n_clusters] = overallAccuracy_kMeans
```

In [73]:
```
pd.concat([overallAccuracy_kMeansDF_original,↔
plt.grid()
```

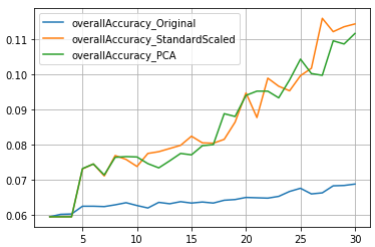


Clearly we can see that PCA data has better performance than original one.

One more demo, let's try the standardScaler scaling data but without PCA.

In [77]:
```
# K-means - Accuracy as the number of clusters varies

# n_clusters = 5
n_init = 10
max_iter = 300
tol = 0.0001
random_state = RANDOM_STATE
n_jobs = -1

range_tmp = range(2,31)
kMeans_inertia = \
    pd.DataFrame(data=[],index=range_tmp,columns=['inertia'])
overallAccuracy_kMeansDF_ss = \
    pd.DataFrame(data=[],index=range_tmp,columns=['overallAccuracy_StandardScaled'])

for n_clusters in range_tmp:
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state, \
                n_jobs=n_jobs)

#     cutoff = 10
    kmeans.fit(df_X_train_ss)
    kMeans_inertia.loc[n_clusters] = kmeans.inertia_
    X_train_kmeansClustered = kmeans.predict(df_X_train_ss)
    X_train_kmeansClustered = \
        pd.DataFrame(data=X_train_kmeansClustered, index=df_X_train_ss.index, \
                     columns=['cluster'])

    countByCluster_kMeans, countByLabel_kMeans, countMostFreq_kMeans, \
        accuracyDF_kMeans, overallAccuracy_kMeans, accuracyByLabel_kMeans \
        = analyzeCluster(X_train_kmeansClustered, df_y_cluster)

    overallAccuracy_kMeansDF_ss.loc[n_clusters] = overallAccuracy_kMeans
```

In [84]:
```
pd.concat([overallAccuracy_kMeansDF_original,
           overallAccuracy_kMeansDF_ss,
                    overallAccuracy_kMeansDF_PCA], axis = 1).plot()
plt.grid()
```



Not bad

The performance of standstscaled data is not bad, but not as good as PCA. Because PCA gradually climbs up without vibrations.

For a worse case, we should use scaling-transformed data. Since most, if not all, clustering algorithms are based on distance and are sensitive to scaling.

Let's move on to try DBSCAN

In [52]:
```
from sklearn.cluster import DBSCAN

eps = 3
min_samples = 5
leaf_size = 30
n_jobs = 4

db = DBSCAN(eps=eps, min_samples=min_samples, leaf_size=leaf_size,
            n_jobs=n_jobs)

cutoff = cut_off
X_train_PCA_dbscanClustered = db.fit_predict(df_X_train_pca_cluster.loc[:,0:cutoff])
X_train_PCA_dbscanClustered = \
    pd.DataFrame(data=X_train_PCA_dbscanClustered, index=df_X_train_pca_cluster.index, \
                 columns=['cluster'])

countByCluster_dbscan, countByLabel_dbscan, countMostFreq_dbscan, \
    accuracyDF_dbscan, overallAccuracy_dbscan, accuracyByLabel_dbscan \
    = analyzeCluster(X_train_PCA_dbscanClustered, df_y_encode)

print("Overall accuracy from DBSCAN: ",overallAccuracy_dbscan)
```

```
Overall accuracy from DBSCAN:  0.0661
```

In [53]:
```
print("Cluster results for DBSCAN")
countByCluster_dbscan
```

```
Cluster results for DBSCAN
```

Out[53]:

	cluster	clusterCount
0	0	9656
1	1	228
2	-1	116

The result might not be very useful. Since the cluster size is too imbalanced. Let's move on to HDBSCAN

In [54]:
```
# HDBSCAN
```

In [171]:
```
import hdbscan

min_cluster_size = 30
min_samples = None
alpha = 1.0
cluster_selection_method = 'eom'

hdb = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size, \
        min_samples=min_samples, alpha=alpha, \
        cluster_selection_method=cluster_selection_method)

cutoff = cut_off
X_train_PCA_hdbscanClustered = \
    hdb.fit_predict(df_X_train_pca_cluster.loc[:,0:cutoff])

X_train_PCA_hdbscanClustered = \
    pd.DataFrame(data=X_train_PCA_hdbscanClustered, \
    index=df_X_train_pca_cluster.index, columns=['cluster'])

countByCluster_hdbscan, countByLabel_hdbscan, \
    countMostFreq_hdbscan, accuracyDF_hdbscan, \
    overallAccuracy_hdbscan, accuracyByLabel_hdbscan \
    = analyzeCluster(X_train_PCA_hdbscanClustered, df_y_encode)
```

In [56]:
```
print("Overall accuracy from HDBSCAN: ",overallAccuracy_hdbscan)
```

```
Overall accuracy from HDBSCAN:  0.0717
```

In [57]:
```
print("Cluster results for HDBSCAN")
countByCluster_hdbscan
```

```
Cluster results for HDBSCAN
```

Out[57]:

	cluster	clusterCount
0	1	9570
1	-1	387
2	0	43

HDBSCAN result is not very good either.

sumary on clustering

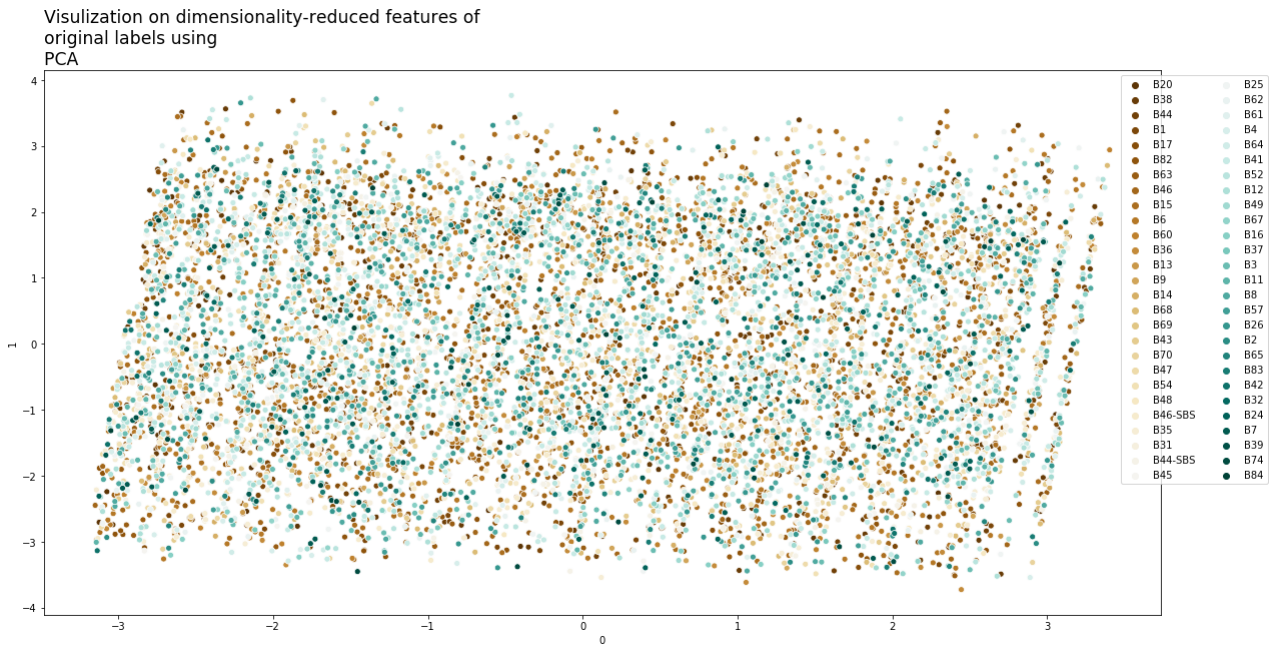
So far, we have used several clustering models for group segmentation and evaluate the results using metrics like accuracy, completeness and homogeneity. Let's try a different point to VIEW to LOOK that the clustering result. Let's VISUALIZE the clusters using dimensionality techniques.

Visualize the clusters

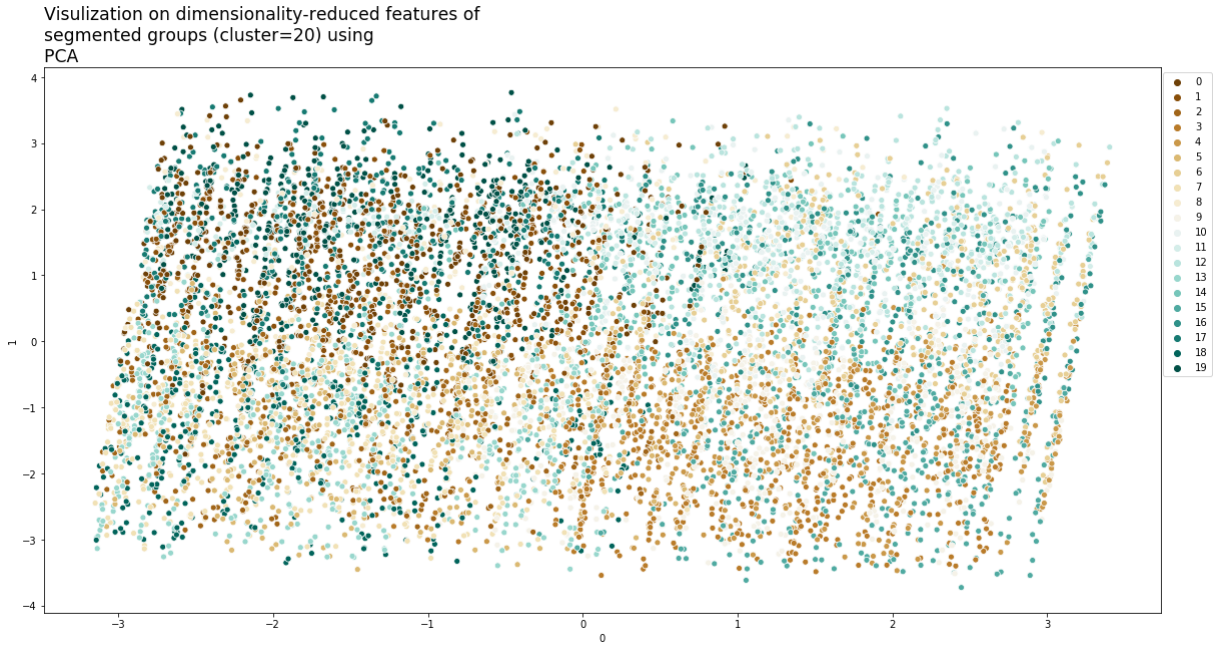
```
### first try n_cluster = 20
From the ealer experiement, the metircs told us n=25 clusters can give us a decent result (no worse than smaller n numbers). Now let's visualize it use both PCA and T-SNE
```

In [85]:
```
# K-means - for plot usage↔
```

In [86]:
```
# visulization on dimensionality-reduced features of \noriginal labels using \nPCA ↔
```



In [87]:
```
# Visulization on dimensionality-reduced features of \nsegmented groups (cluster=25) using \nPCA ↔
```

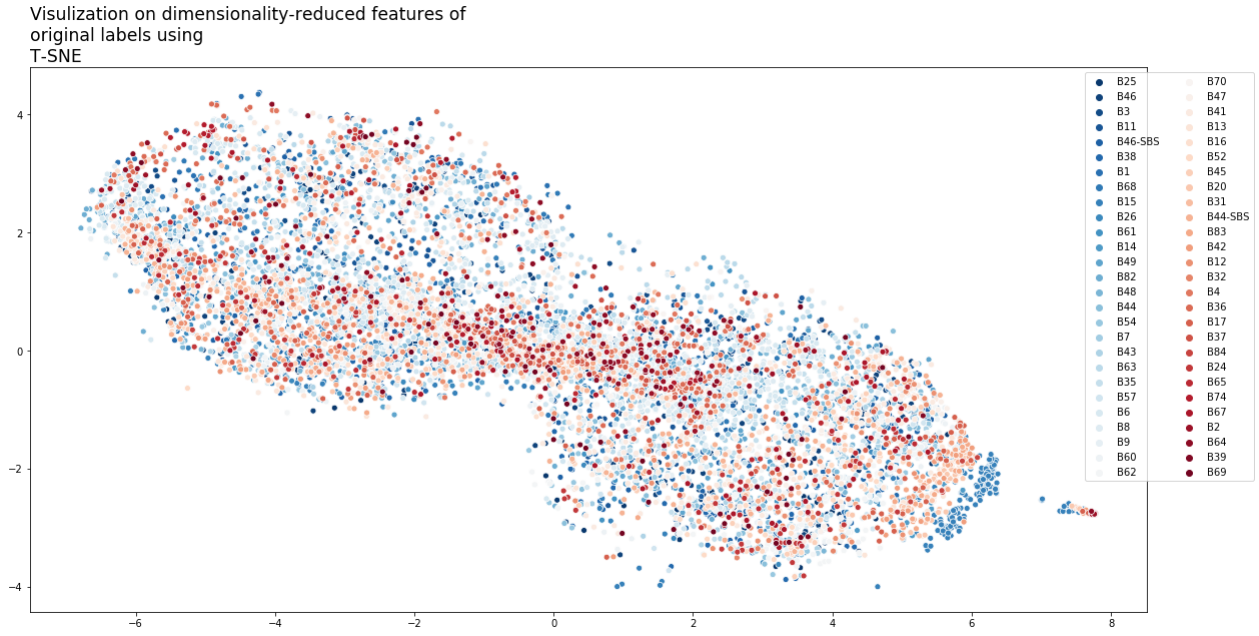


The plot is very overwhelming, let's move on and try T-SNE

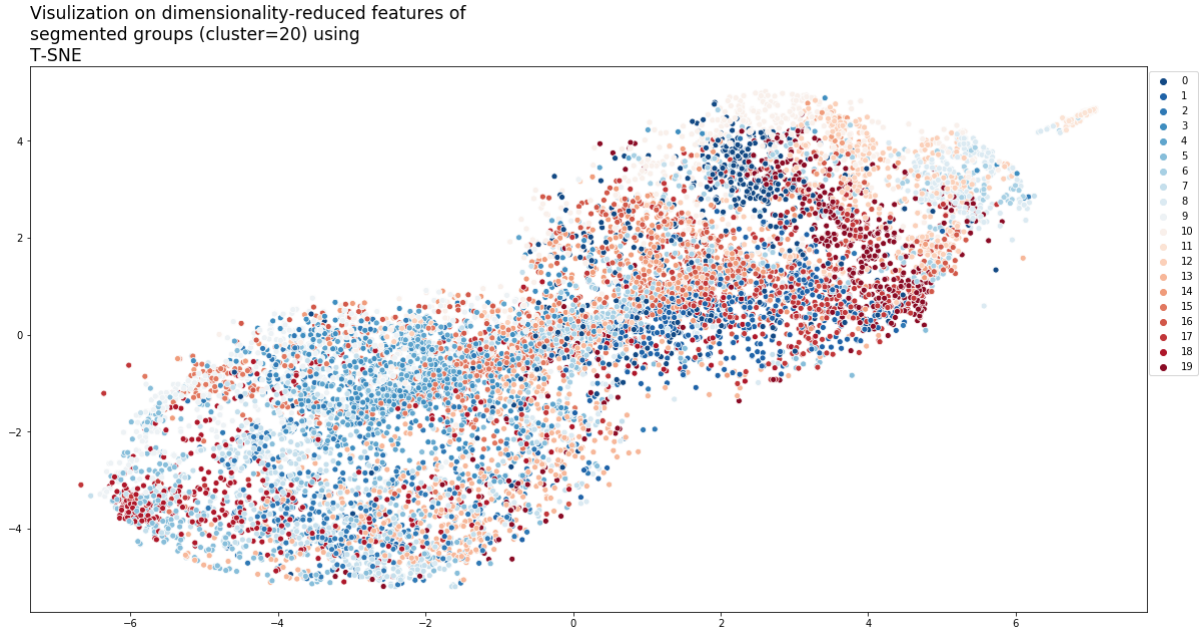
Try T-SNE

In [89]:
```
from sklearn.manifold import TSNE

# time_start = time.time()
tsne = TSNE(n_components=2, perplexity=40, n_iter=300, n_jobs=-1,
#            verbose=1,
            random_state = RANDOM_STATE
           )
tsne_results = tsne.fit_transform(df_X_train_pca_cluster)
```

In [62]:
```
# Visulization on dimensionality-reduced features of \noriginal labels using \nT-SNE↔
```



In [90]:
```
# Visulization on dimensionality-reduced features of \nsegmented groups (cluster=25) using \nT-SNE ↔
```



Comment

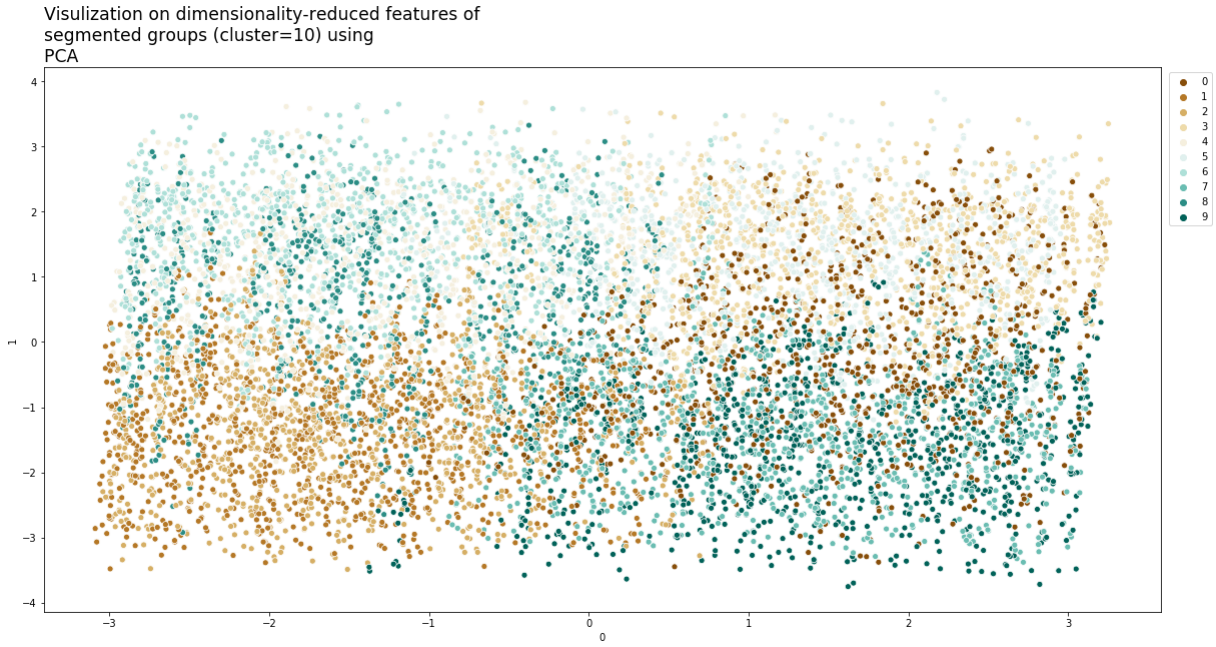
Looks like T-SNE gives us nicer visualization over PCA. Since T-SNE is a non-linear dimensionality reduction algorithm and it can capture the local geometry while PCA only consider the global geometry.

But the number of cluster is still too many, we are overwhelmed by the different colors. Just for demonstration, let's try to visualize the segementation groups by using small n_cluster, even it gives us less accuracy.

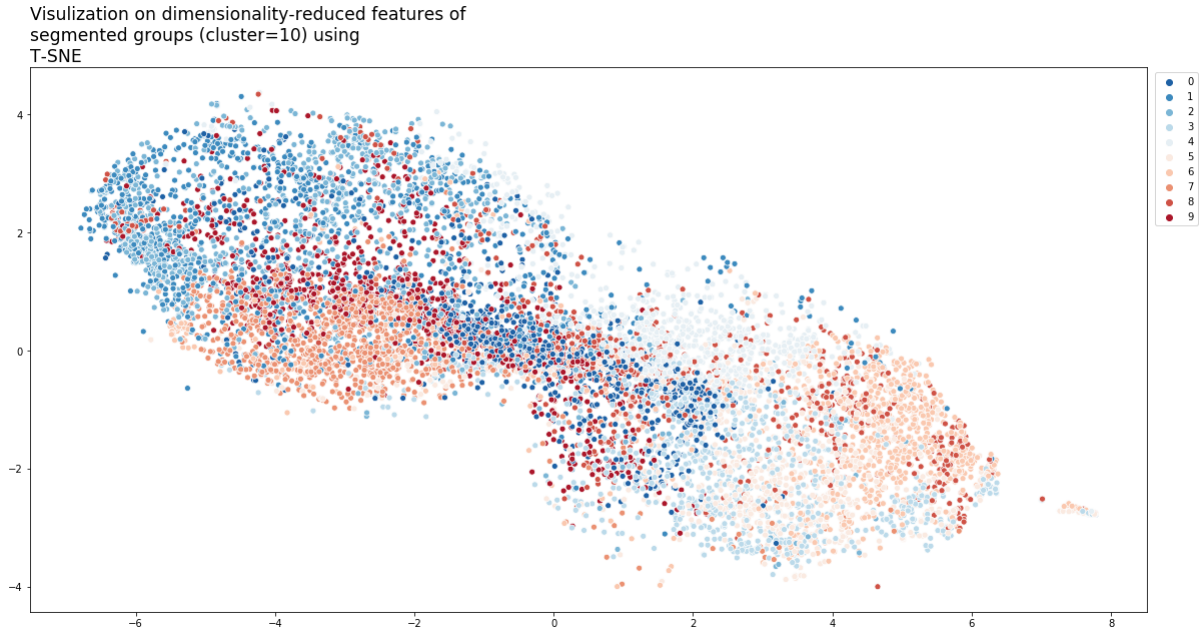
TIPS: use diverging palettes to plot the scatter when there is too many hues, the default "rainbow" palettes will give us an even overwhelmed visualization.

In [68]:
```
# K-means - for plot usage↔
```

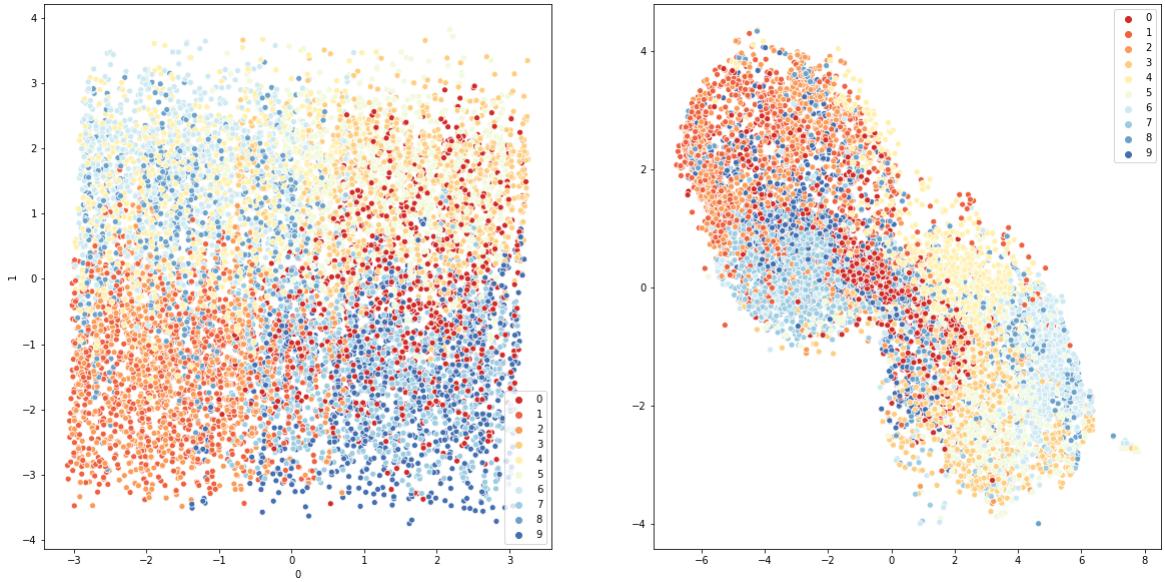
In [69]:
```
# Visulization on dimensionality-reduced features of \nsegmented groups (cluster=10) using \nPCA
```



In [70]:
```
# Visulization on dimensionality-reduced features of \nsegmented groups (cluster=25) using \nT-SNE
```



In [72]:
```
1 # compare PCA and T-SNE side by side↔
```



Comment

When the number of clusters decreases, it is easier for us human to figure out some patterns. Even for the PCA plot we can see there are some obvious clusters there.

Personally, I think T-SNE is more efficient to represent the underlyting patterns of cluster. But this can be personal taste, also it is effected by the choice of color. i.e. some palatte is not colorblind safe. But just to make a point, I plot the PCA plot and T-SNE plot side by using the same palete "RdYlBu" which is considered colorblind safe. From that comparison, I still think T-SNE is more visually persuasive to represent the clusters.

You see, even the metrics like accuracy, completeness_score and homogeneity_score might tell us lower score when the number of cluster is low, but viually speaking we might have different opinion on that.

That brings up the question: what is the right number of clusters? Or we can even go further to ask: what is the right assignment for the clusters? Because it might bring us to an awkward situation, when we look at the clustering results, we don't know how to use that.

That being said, I am sure there are some reasons K-means, DBSCAN or other clustering method gave us such result. But it is very hard to utilize such result, i.e. "B6" line is assigned to several different clusters (low completeness) , or one cluster has many bus lines (low homogeneity), it become very impractical to manage such organization design.

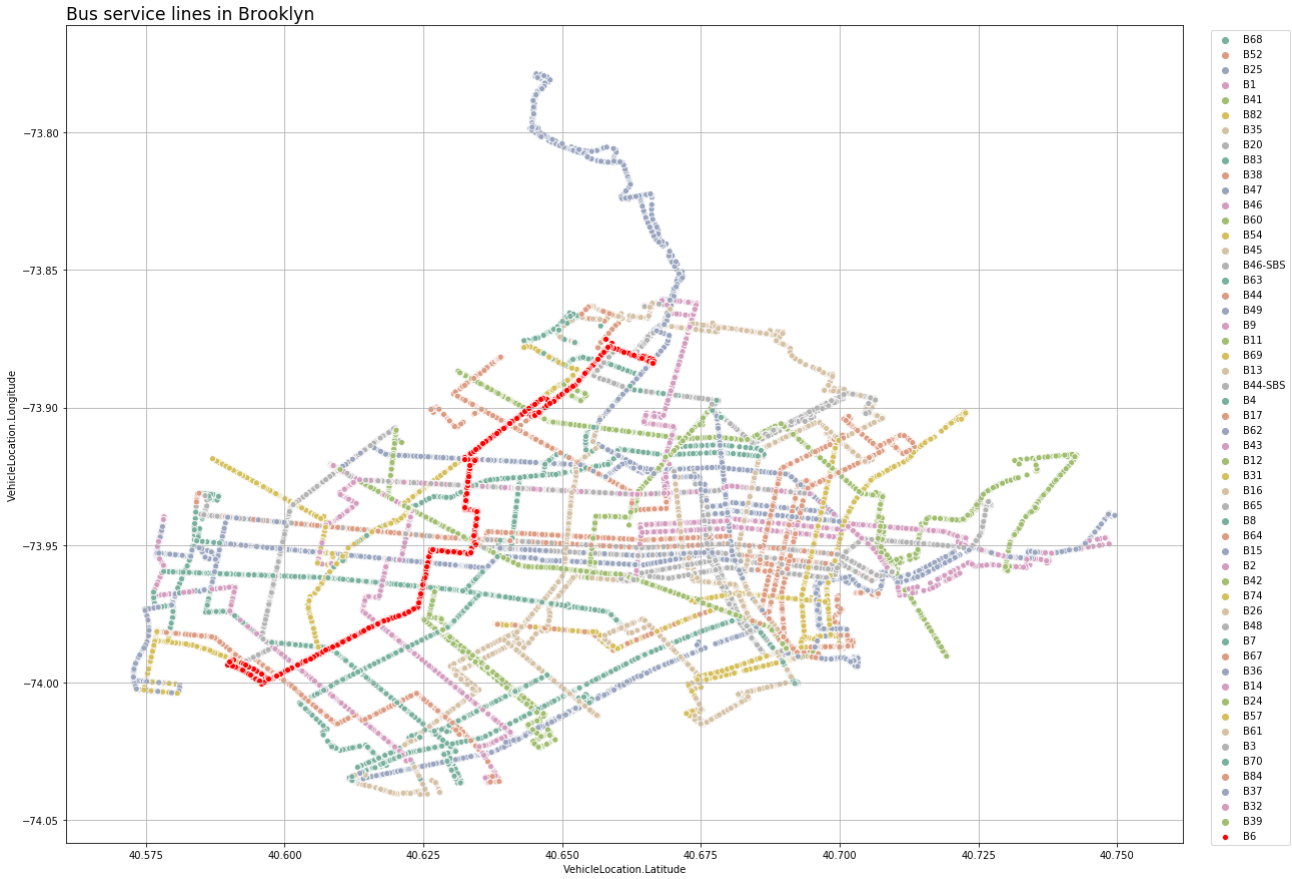
Ideally, it should be some way to include certain structure contraints into machine learining. Proababely we can find that in deep learning. But at this stage with the tools in hand, I cannot do that.

reference: Homogeneity and Completeness (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_completeness_v_measure.html)

Here is a good news: we still can use some domain knowledge to perform high-level feature engineering. The following is an example that I feature engineeredthe data and humanly reduce dimensionalityies.

In [184]:

```
# plot a brooklyn lines
df_lines_brooklyn = df_fillna[df_fillna.PublishedLineName.isin(lines_brooklyn)]
df_lines_brooklyn_trim = df_lines_brooklyn.dropna().sample(n=100000)

fig, ax = plt.subplots(figsize=(20,15))
sns.scatterplot(data=df_lines_brooklyn_trim[df_lines_brooklyn_trim.PublishedLineName!='B6'], ax=ax,
                x='VehicleLocation.Latitude', y='VehicleLocation.Longitude',
               hue=df_lines_brooklyn_trim.loc[df_lines_brooklyn_trim.PublishedLineName!='B6','PublishedLineName'].values,
#                label='lines',
                palette=sns.color_palette("Set2", len(df_y_cluster.PublishedLineName.unique())-1, desat=0.6, ))
sns.scatterplot(data=df_lines_brooklyn_trim[df_lines_brooklyn_trim.PublishedLineName=='B6'], ax=ax,
                x='VehicleLocation.Latitude', y='VehicleLocation.Longitude',
#                hue=df_lines_brooklyn_trim['PublishedLineName'].values,
                color='red',
                label='B6',
#                palette=sns.color_palette("Set2", len(df_y_cluster.PublishedLineName.unique()))
               )

# handles = g._legend_data.values()
# labels = g._legend_data.keys()
plt.legend(loc='upper right',
#            ncol=2,
           bbox_to_anchor=(1.1, 1.0)
          )
plt.title("Bus service lines in Brooklyn", loc='left', size="xx-large")
plt.grid()
plt.show()
```



BACK TO THE MAIN PATH

In [ ]:
```
1
```

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 we can argue that the analysis is not unrepresentative.

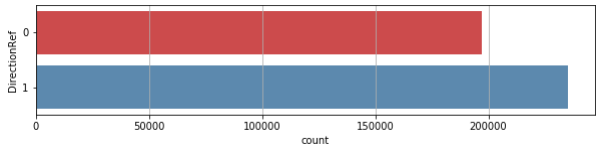
In [74]:
```
# focus on 'B6' line
def reset_dataset():↔
# df_trim = reset_dataset()
```

In [75]:
```
# df_trim = df_trim_archive.copy()
```

In [76]:
```
df_trim = reset_dataset()
```

In [77]:
```
# df_trim.groupby(['DirectionRef', ])[['OriginName']].count().reset_index().rename(columns={'OriginName': 'count'})
```

In [78]:
```
# 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*: a reminder, I will designate color code 'red' for DirectionRef=0, and 'blue' for DirectionRef=1

In [79]:
```
# nested info ↔
```

Out[79]:

			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 (note*) 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. While for directionRef=0, the rounts are from 3 different origins to 3 different destinations.

You 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. Since it has the most counts in the records, assuming the data was sampled randomly.

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

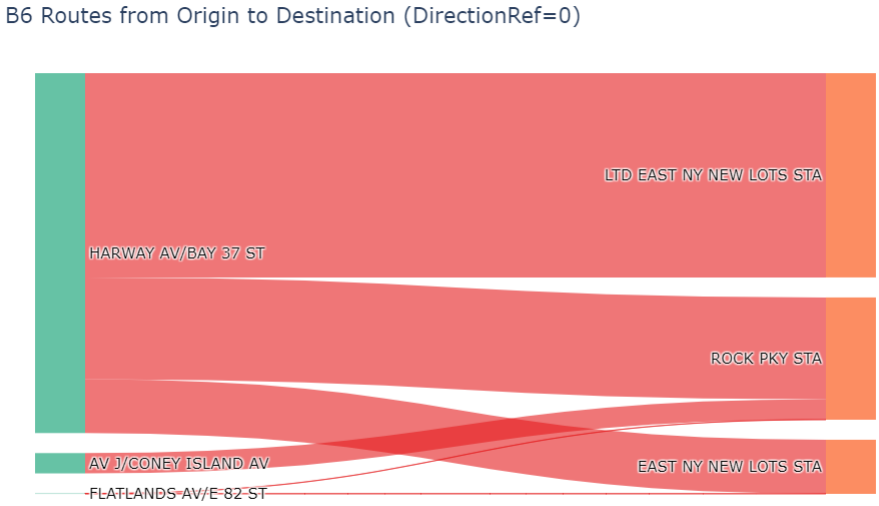
Note*: when I said 'this route' I mean one of the 15 routes runing on B6, i.e. from 'HARWAY AV/BAY 37 ST' to 'LTD EAST NY NEW LOTS STA'.

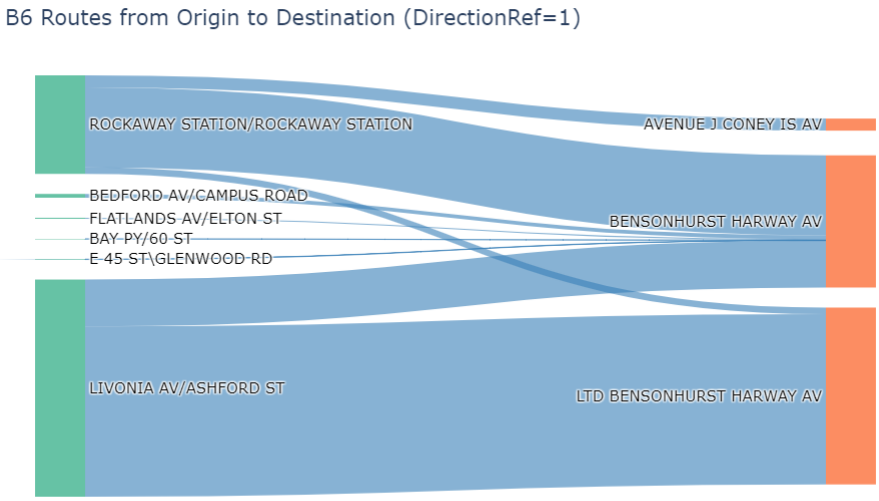
In [80]:
```
# prepare data for sankey diagram↔
```

In [81]:
```
# sankey diagram visulize origin and destination (enable that so you can play with the interative plot)↔
```

In [82]:
```
# static figure. just in case you cannot load the interactive plot, ↔
```

```
C:\Users\kefei\Anaconda3\envs\thinkful\lib\site-packages\plotly\io\_base_renderers.py:126: ResourceWarning:

unclosed file <_io.BufferedWriter name='nul'>
```





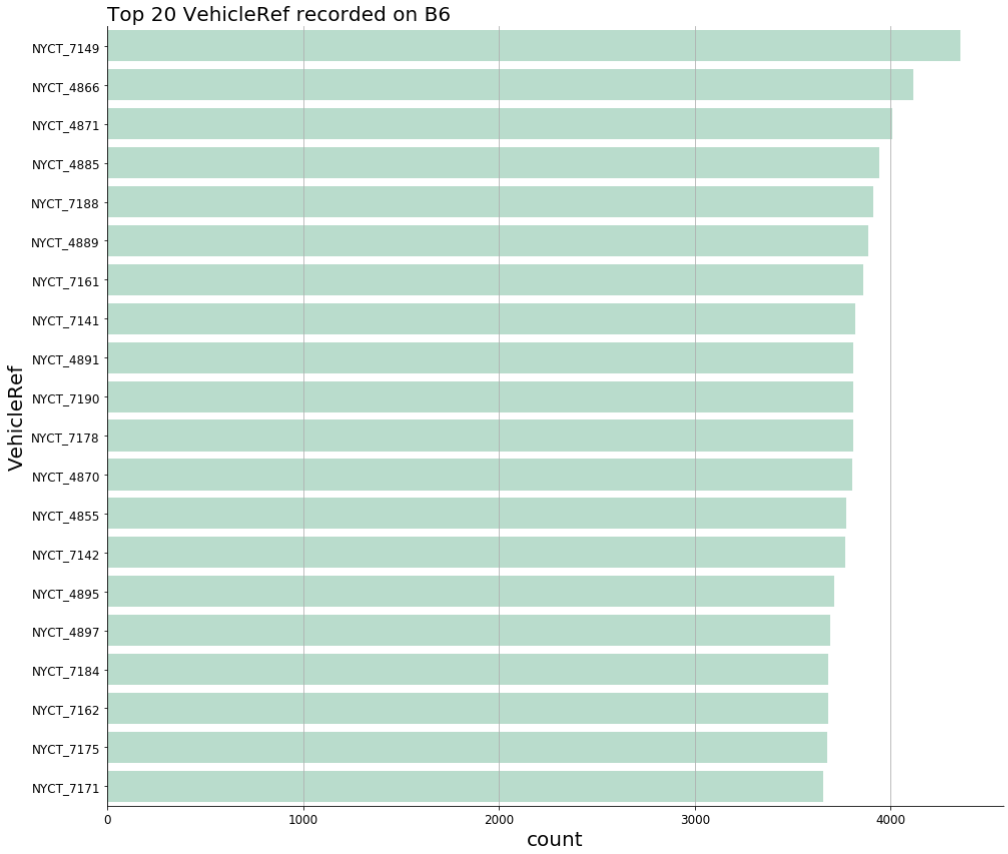
Now the operation information of each rount is much clearer. From the sankey diagram we can have a better sense which route(s) has/have more records (indicating such route(s) are operated more frequently).

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

In [83]:
```
# (A side note about visualization here, ask me about it you are interested.)↔
```

In [84]:
```
# analyze VehicleRef↔
```

In [85]:
```
# 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 [86]:
```
### 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)

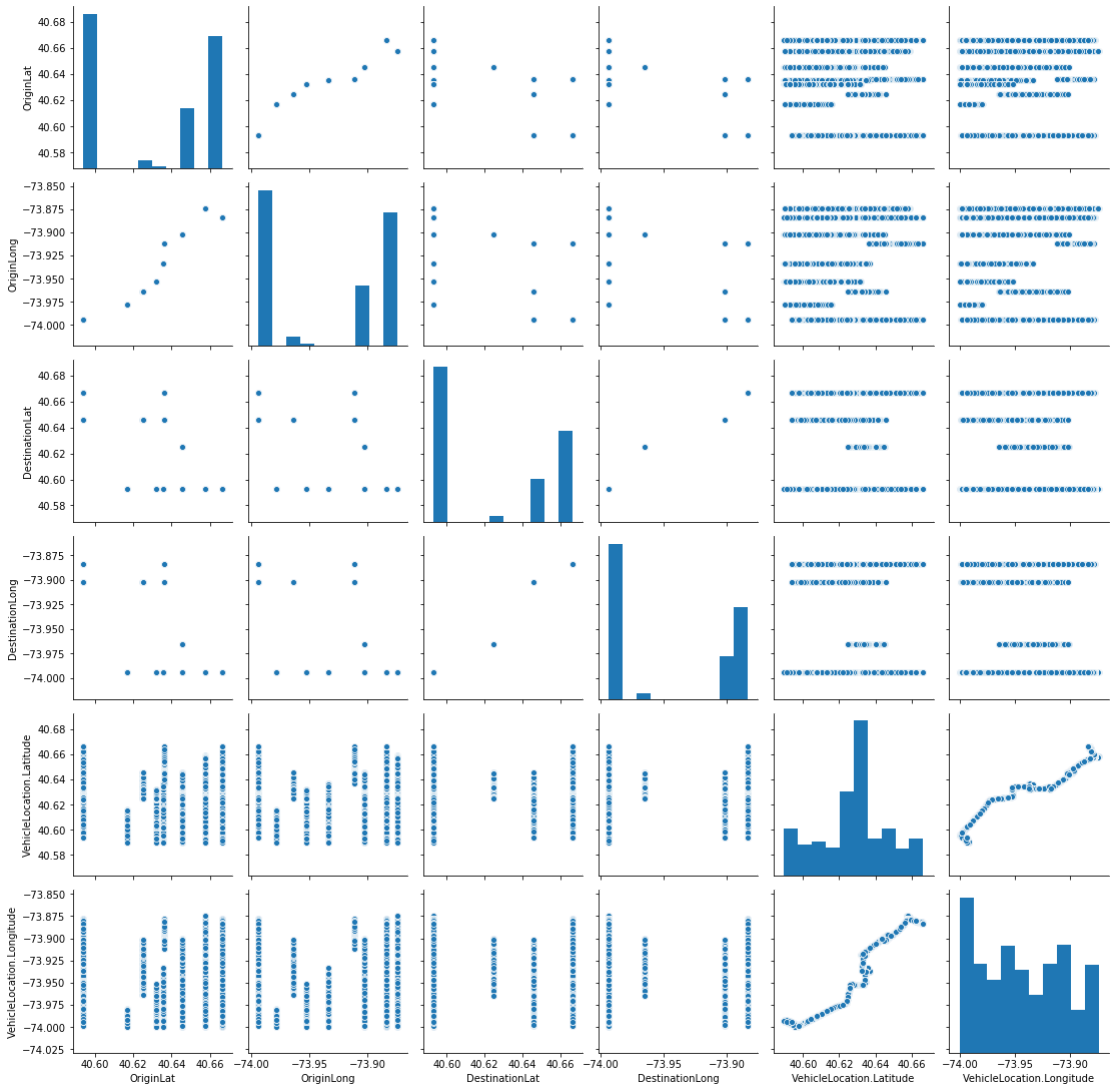
relative location

- ArrivalProximityText (dynamic)
- DistanceFromStop (dynamic)

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

In [87]:
```
sns.pairplot(data=df_trim, vars=['OriginLat', 'OriginLong', 'DestinationLat', 'DestinationLong', 'VehicleLocation.Latitude', 'VehicleLocation.Longitude'])
```

Out[87]: <seaborn.axisgrid.PairGrid at 0x1bb66690e88>



Many times, it is very convenient to use sns.pairplot for EDA(notes*), since it can show both univariant distribution and co-relationship. But for GPS data, we should use pairplot in a different way. i.e.it doesn't make much sense to point out high-correlationship like the one between 'VehicleLocation.Latitude' and 'VehicleLocation.Longitude'. Since GPS coordinates are more likely to be useful when they are paired to each other. On the other hand, we probably should not take it seriously about univariant distribution either, i.e. a bus service can operate in a east to west, so with large variance in latitude but small in longitude, so, we can not draw conclusion about the overal operation range based on individual GPS columns.

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.

notes*: Choose wisely. Put less than 20 features for pairplot. Actually, I would say 10 is the maxium you should try. It is overwhelming for both of your eyes and the computer when the number of features to pair is too large.

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

In [88]:
```
# this one to check legit (), by rule, the origin and desination GPS should be unique for each route↔
```

In [89]:
```
# legit check cont'↔
```

In [90]:
```
# use this intimidating pivot table to route info↔
```

Out[90]:

	DirectionRef	OriginName	DestinationName	OriginLat	OriginLong	DestinationLat	DestinationLong	NextStopPointName_noOrder	count
4	0	HARWAY AV/BAY 37 ST	LTD EAST NY NEW LOTS STA	40.593510	-73.993996	40.666420	-73.883386	[AV J/E 5 ST, GLENWOOD RD/FLATBUSH AV, BAY PY/...	105715
5	0	HARWAY AV/BAY 37 ST	ROCK PKY STA	40.593510	-73.993996	40.645775	-73.902070	[BAY PY/86 ST, AV J/OCEAN PY, AV J/CONEY ISLAN...	52684
3	0	HARWAY AV/BAY 37 ST	EAST NY NEW LOTS STA	40.593510	-73.993996	40.666420	-73.883386	[GLENWOOD RD/E 98 ST, AV H/E 56 ST, BAY PY/STI...	27872
0	0	AV J/CONEY ISLAND AV	ROCK PKY STA	40.624829	-73.964228	40.645775	-73.902070	[BEDFORD AV/AV J, GLENWOOD RD/NOSTRAND AV, GLE...	10501
1	0	FLATLANDS AV/E 82 ST	EAST NY NEW LOTS STA	40.636307	-73.911844	40.666420	-73.883386	[FLATLANDS AV/E 88 ST, GLENWOOD RD/WILLIAMS AV...	138

In [91]:
```
# feature engineering add next stop lat and long↔
```

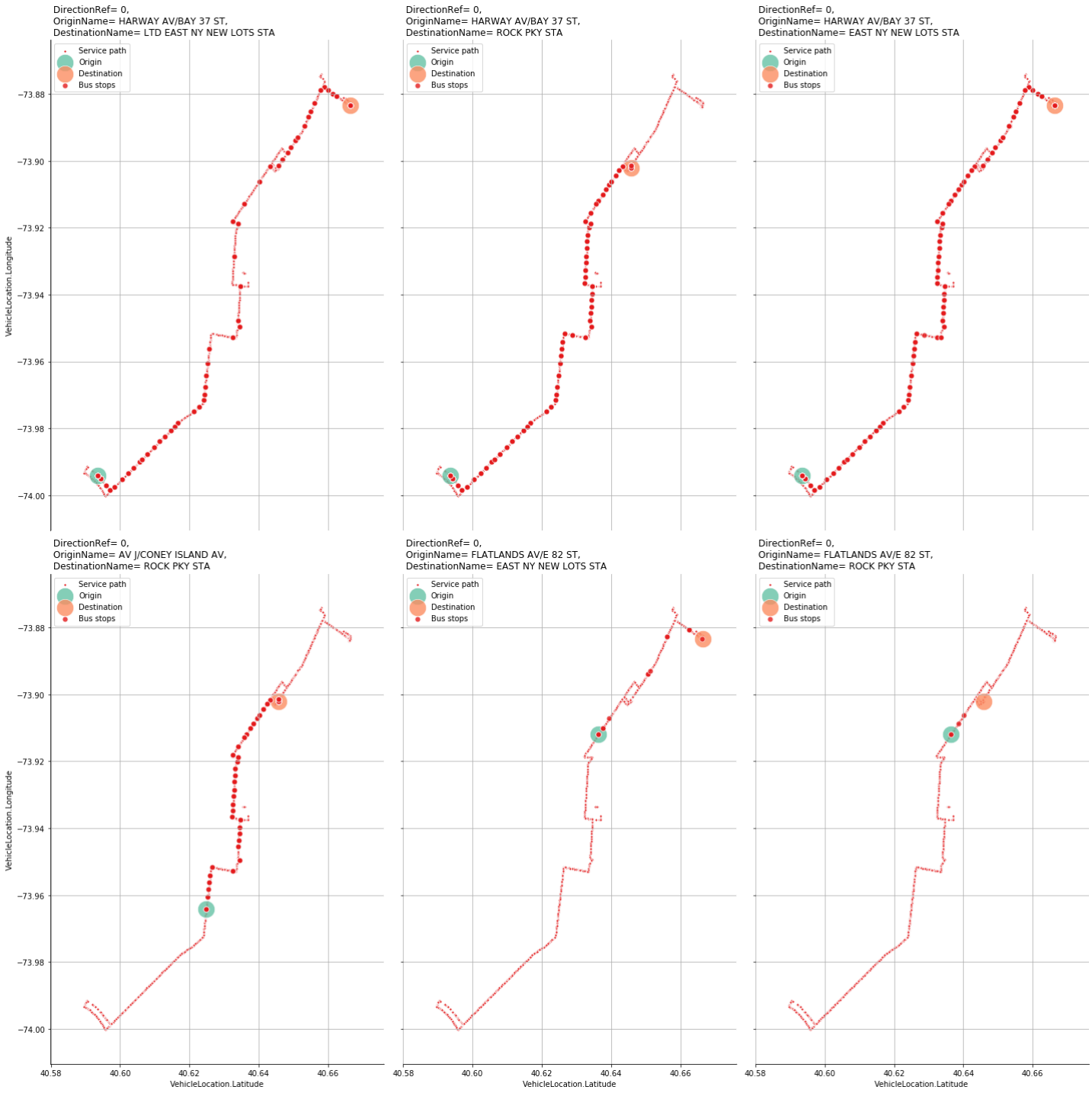
Out[91]:

	RecordedAtTime	DirectionRef	PublishedLineName	OriginName	OriginLat	OriginLong	DestinationName	DestinationLat	DestinationLong	VehicleRef	VehicleLocation.Latitude	VehicleLocation.Longitude	NextStopPointName	Arriva
20	2017-06-01 00:03:41	0	B6	HARWAY AV/BAY 37 ST	40.593510	-73.993996	EAST NY NEW LOTS STA	40.666420	-73.883385	NYCT_7158	40.645676	-73.901474	GLENWOOD RD/E 98 ST	
91	2017-06-01 00:03:42	1	B6	LIVONIA AV/ASHFORD ST	40.666382	-73.883614	BENSONHURST HARWAY AV	40.592949	-73.993385	NYCT_4989	40.666288	-73.883834	ASHFORD ST/NEW LOTS AV	
191	2017-06-01 00:03:27	1	B6	LIVONIA AV/ASHFORD ST	40.666382	-73.883614	BENSONHURST HARWAY AV	40.592949	-73.993385	NYCT_4861	40.642107	-73.903071	FLATLANDS AV/E 94 ST	
221	2017-06-01 00:03:25	0	B6	HARWAY AV/BAY 37 ST	40.593510	-73.993996	EAST NY NEW LOTS STA	40.666420	-73.883385	NYCT_7151	40.633125	-73.923898	AV H/E 56 ST	
294	2017-06-01 00:03:49	1	B6	LIVONIA AV/ASHFORD ST	40.666382	-73.883614	BENSONHURST HARWAY AV	40.592949	-73.993385	NYCT_7174	40.626191	-73.952191	AV J/E 23 ST	

In [92]:
```
### details alert ↔
```

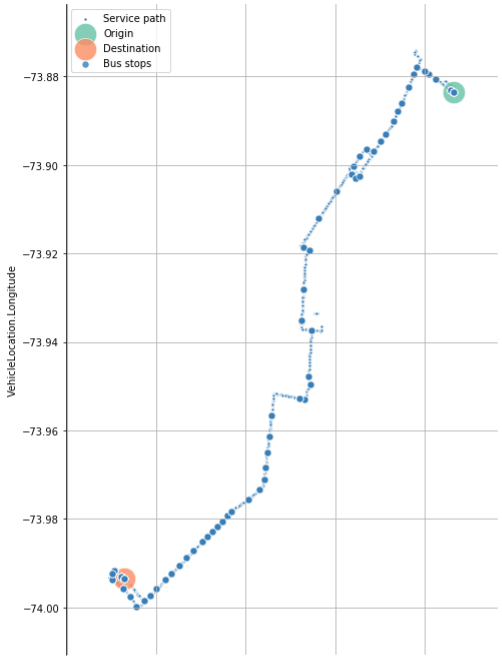
In [93]:
```
# customer function use to plot bus route, use:↔
```

In [94]:
```
# customer function use to plot bus route,↔
```

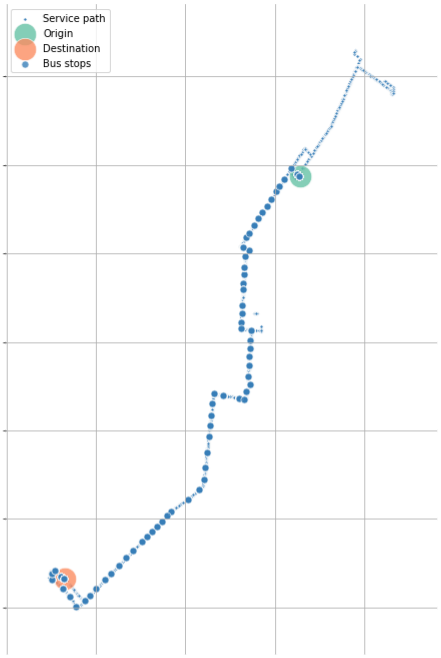


In [95]:
```
# customer function use to plot bus route,↔
```

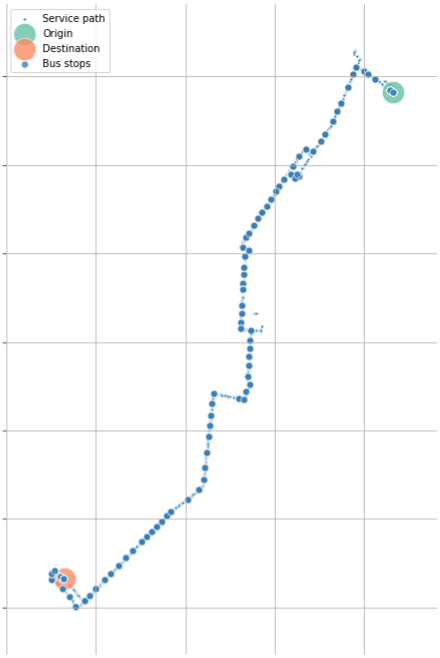
DirectionRef= 1,
OriginName= LIVONIA AV/ASHFORD ST,
DestinationName= LTD BENSONHURST HARWAY AV



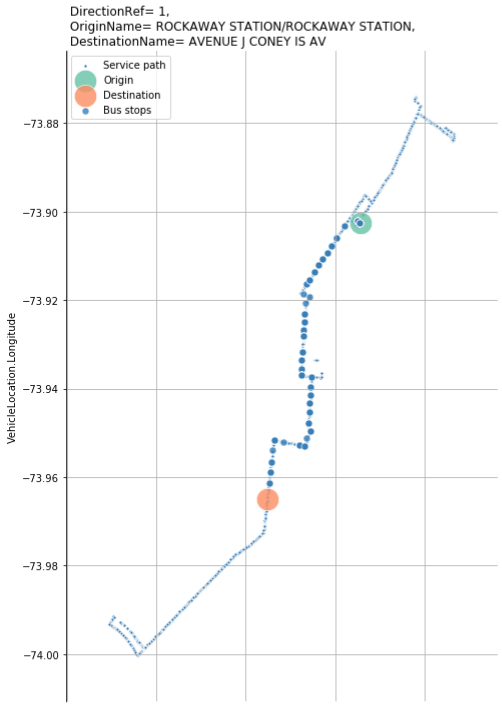
DirectionRef= 1,
OriginName= ROCKAWAY STATION/ROCKAWAY STATION,
DestinationName= BENSONHURST HARWAY AV



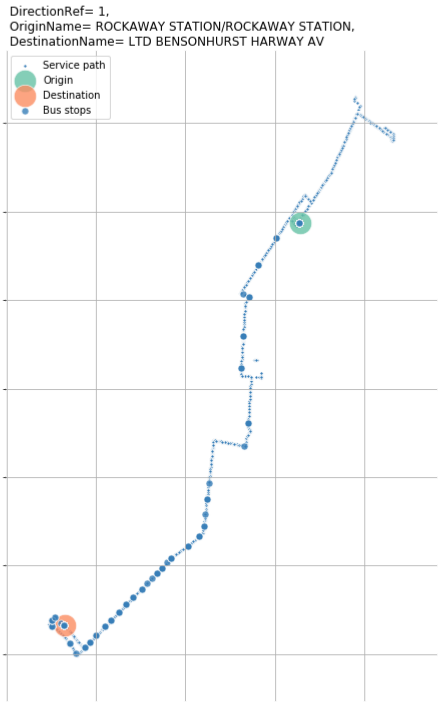
DirectionRef= 1,
OriginName= LIVONIA AV/ASHFORD ST,
DestinationName= BENSONHURST HARWAY AV



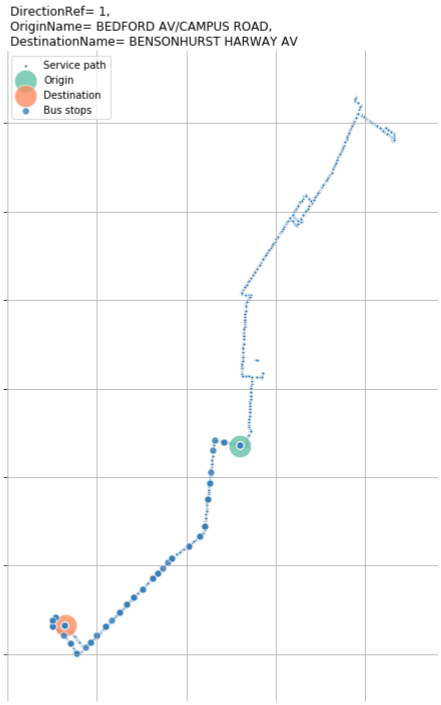
DirectionRef= 1,
OriginName= ROCKAWAY STATION/ROCKAWAY STATION,
DestinationName= AVENUE J CONEY IS AV



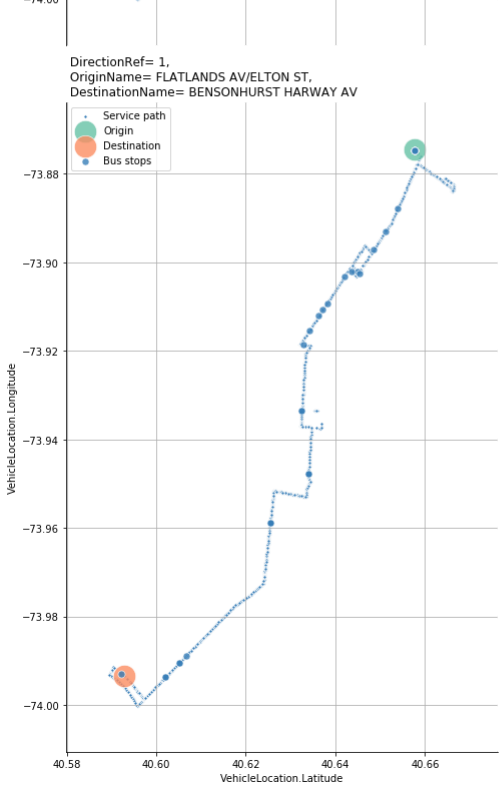
DirectionRef= 1,
OriginName= ROCKAWAY STATION/ROCKAWAY STATION,
DestinationName= LTD BENSONHURST HARWAY AV



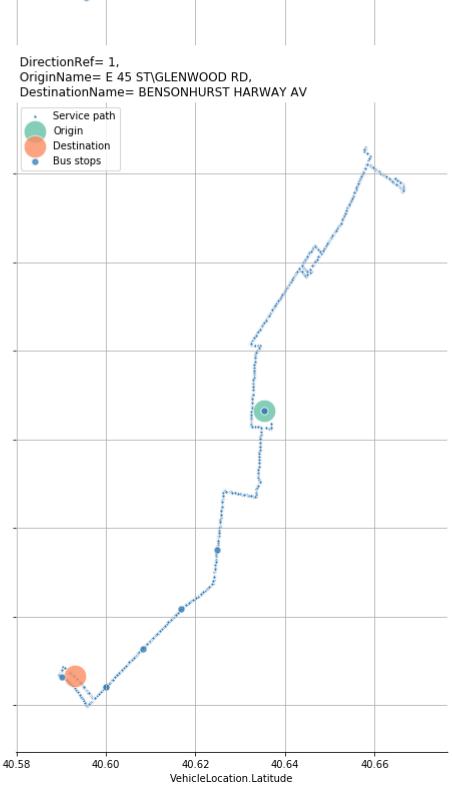
DirectionRef= 1,
OriginName= BEDFORD AV/CAMPUS ROAD,
DestinationName= BENSONHURST HARWAY AV



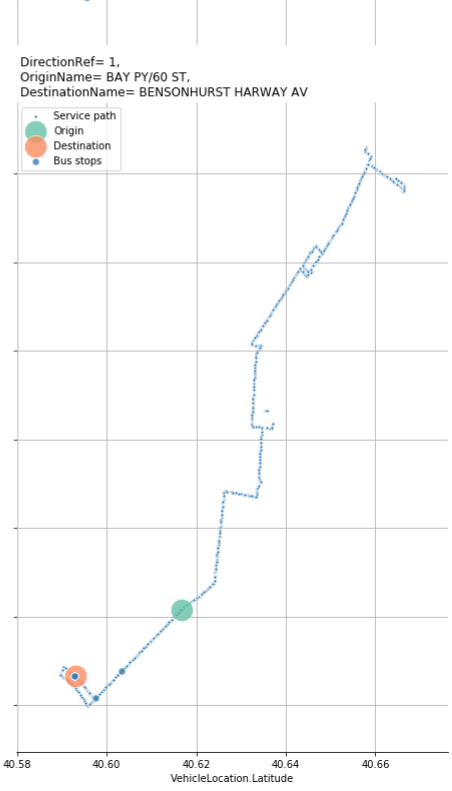
DirectionRef= 1,
OriginName= FLATLANDS AV/ELTON ST,
DestinationName= BENSONHURST HARWAY AV



DirectionRef= 1,
OriginName= E 45 ST\GLENWOOD RD,
DestinationName= BENSONHURST HARWAY AV



DirectionRef= 1,
OriginName= BAY PY/60 ST,
DestinationName= BENSONHURST HARWAY AV



GPS data is fun

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. 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 distance and/or have to operate at more stops. Or simply becasue the bus operation company dispatch more buses on such routes, while the other routes with very limited stops were just emergy operation.

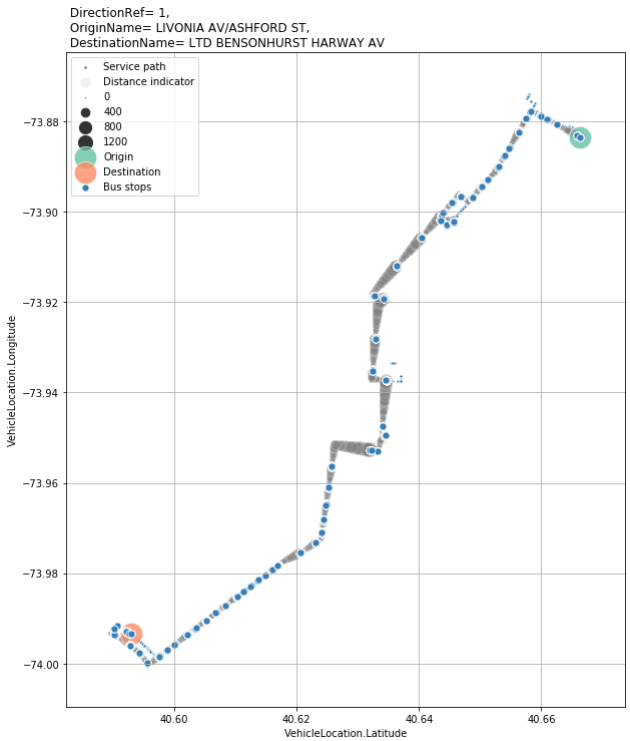
Also it would bring our attention to details which might otherwise ignore. i.e. route from 'HARWAY AV/BAY 37 ST' to 'LTD EAST NY NEW LOTS STA' and route from 'HARWAY AV/BAY 37 ST' to 'EAST NY NEW LOTS STA' have exactly the same origin and destination. But we cannot rename 'LTD EAST NY NEW LOTS STA' to '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.

In [96]:
```
# A note for myself↔
```

There is more story to tell

In [97]:
```
# 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) ({}). For this data set the desimal is at 1e-6, associated with 10 cm (4 inches), which is denfinetely precise enough for bus operation study.

But what about the trueness. 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.)

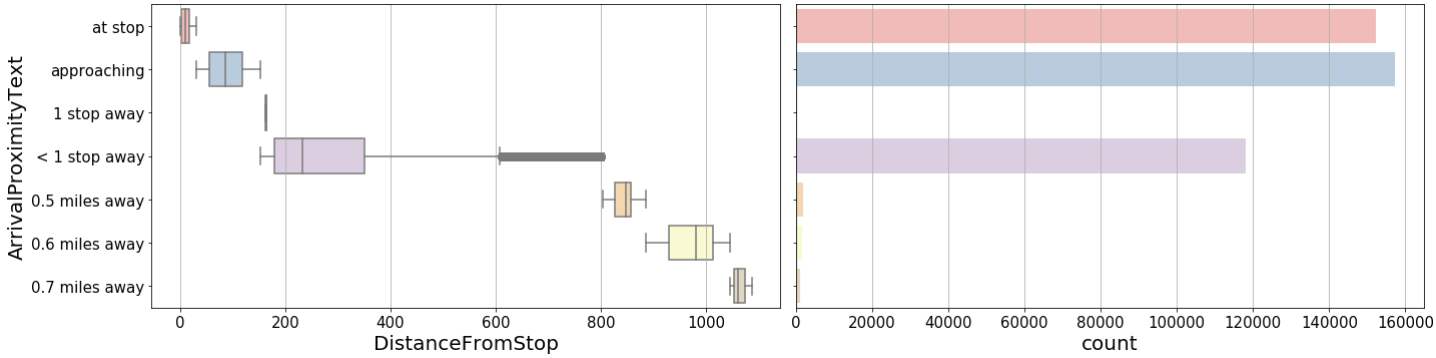
Some data is directely measurement like GPS coordinates but some are from derivation, i.e. 'DistanceFromStop', normally such data can be derive from GPS coordinates, but this is not trivial. i.e. It would not be correct if we calculate the distance between two points on the bus route without considering 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 NEXT bus stops. You can tell the derived 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. i.e. when it says 'at stop', what does 'NextStopPointName' indicate? Is it the stop right now or the one it is going to? Whoever named the variable might have a different idea as what the name means in your head. So, always check with several features and try to prove that.

In [98]:
```
# A small challenge. ↔
```

Relative location data analysis

In [99]:
```
# 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.

But it is not very clear what's difference between '1 stop away' and '<1 stop away' based on the name, i.e. why '1 stop away' has shorter distanceFromStop than '<1 stop away'?

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 [100]:
```
# parse recordtime to hour, min, sec
def feature_engeering_dataset(): ↔

df_trim = feature_engeering_dataset()
```

In [101]:
```
df_trim[['RecordedAtTime', 'ExpectedArrivalTime', 'ScheduledArrivalTime', 'ScheduledArrivalTime_standand']][(df_trim.ScheduledArrivalTime_hour>=24) |\
                                                                                                            (df_trim.ScheduledArrivalTime_hour<=2)].head()
```

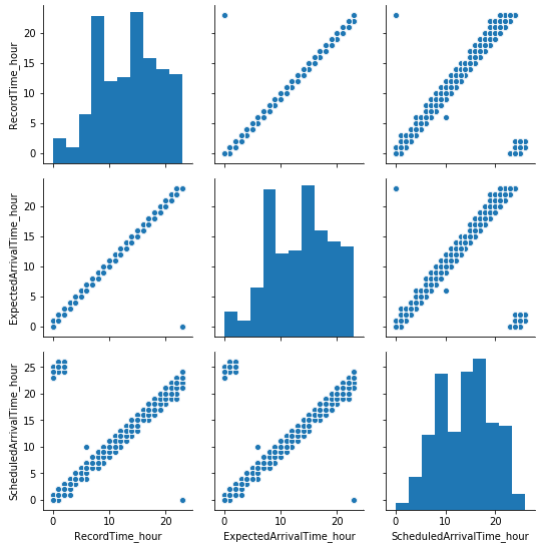
Out[101]:

	RecordedAtTime	ExpectedArrivalTime	ScheduledArrivalTime	ScheduledArrivalTime_standand
20	2017-06-01 00:03:41	2017-06-01 00:03:56	24:05:00	2017-06-01 00:05:00
91	2017-06-01 00:03:42	2017-06-01 00:03:56	24:04:27	2017-06-01 00:04:27
221	2017-06-01 00:03:25	2017-06-01 00:04:05	24:05:48	2017-06-01 00:05:48
294	2017-06-01 00:03:49	2017-06-01 00:04:22	24:05:08	2017-06-01 00:05:08
514	2017-06-01 00:03:37	2017-06-01 00:04:17	24:04:29	2017-06-01 00:04:29

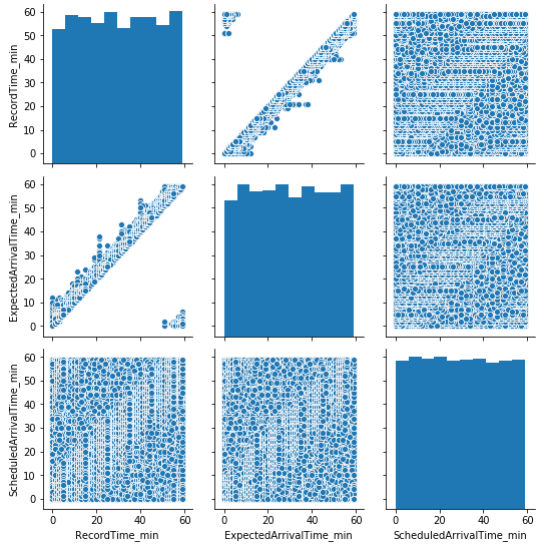
In [102]:
```
# df_trim.head(5)
```

pairplot on hour, min, seconds

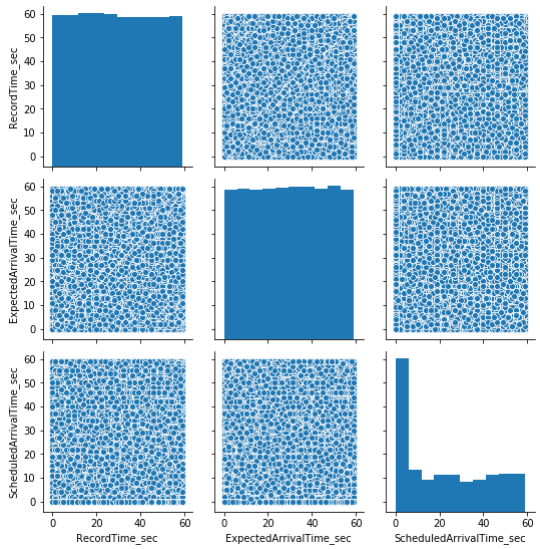
In [103]:
```
# hour↔
```



In [104]:
```
# minute↔
```



In [105]:
```
# seconds↔
```



It is not crazy to care about 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 beteen expect-record paired 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 the bus is 'at stop' or 'approaching', which also indicates that the bus stops are kinda close to each other. On the other hand, 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 might be 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 and/or whether the sampled object is randomly distributed. Both case will contribute to a uniform distribution. But what surprises me is to see that the schedule arrival time is planned at second level, my original guess was all the scheduledArrivalTime_sec would be zero. This makes me curious about how they make the scheduling decision to such details.

Insights of

Expected time to arrive

back to main

Relationship with categorical variables

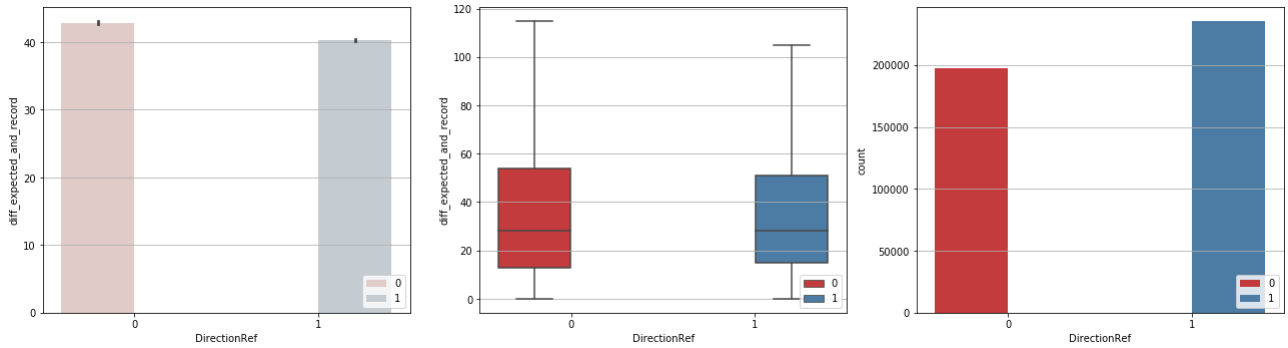
The difference between "ExpectedArrivalTime_hour" and "RecordedAtTime" can read as "expected time to arrive".

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.). Basically, I don't suggest rely too much on the result from average. That's why I faded the color of average analysis using barplot. Rather we can include the count plot to indicate the balance among classes.

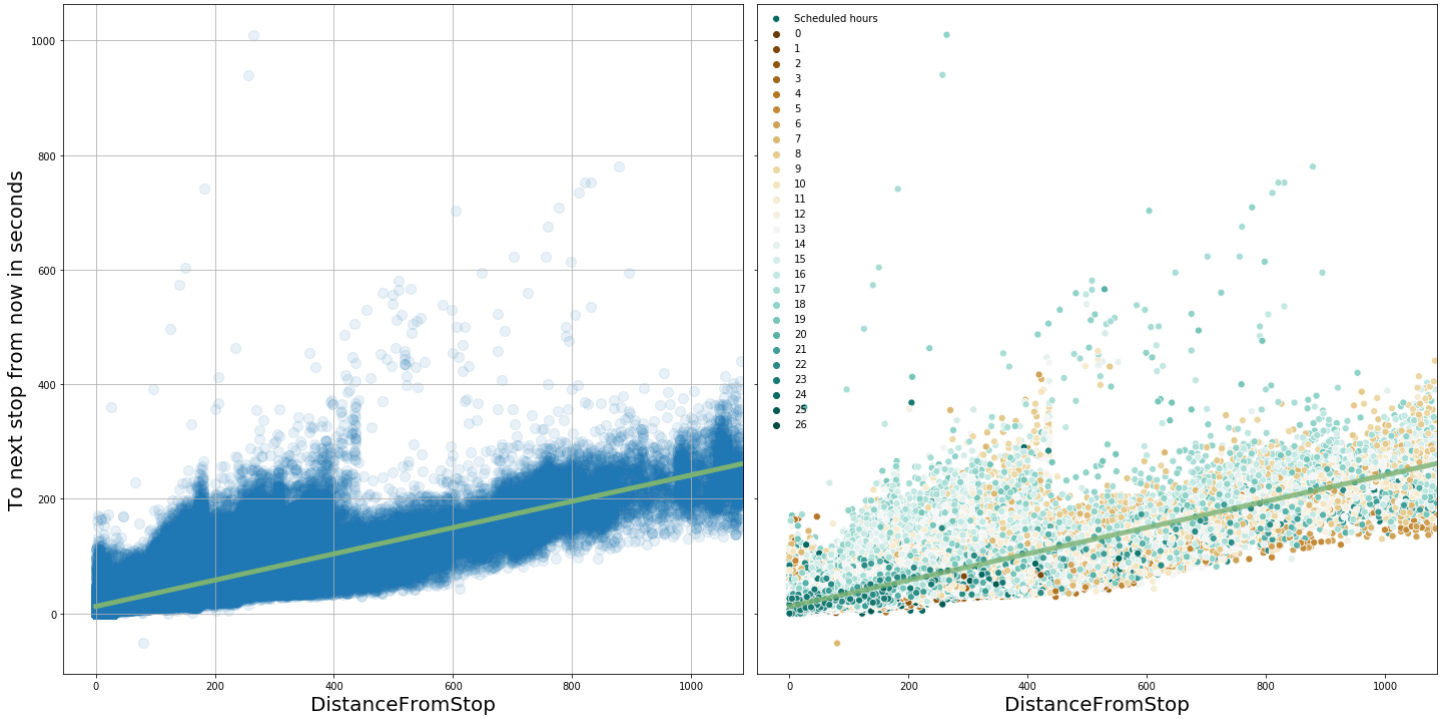
Similarly we can apply idea to another categorical feauture 'OrginalName', we can also discovery some interesting statistics. i.e. Bus from 'FLATLANDS AV/ELTON ST' has longer delay than those from 'E 45 ST\GLENWOOD RD'. (which is not shown here)

In [106]:
```
# show expected time to arrive using bar plot and box plot ↔
```



Relationship with numerical variables

In [107]:
```
# 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 [108]:
```
# check weird data↔
```

Insights

bus delays

back to main

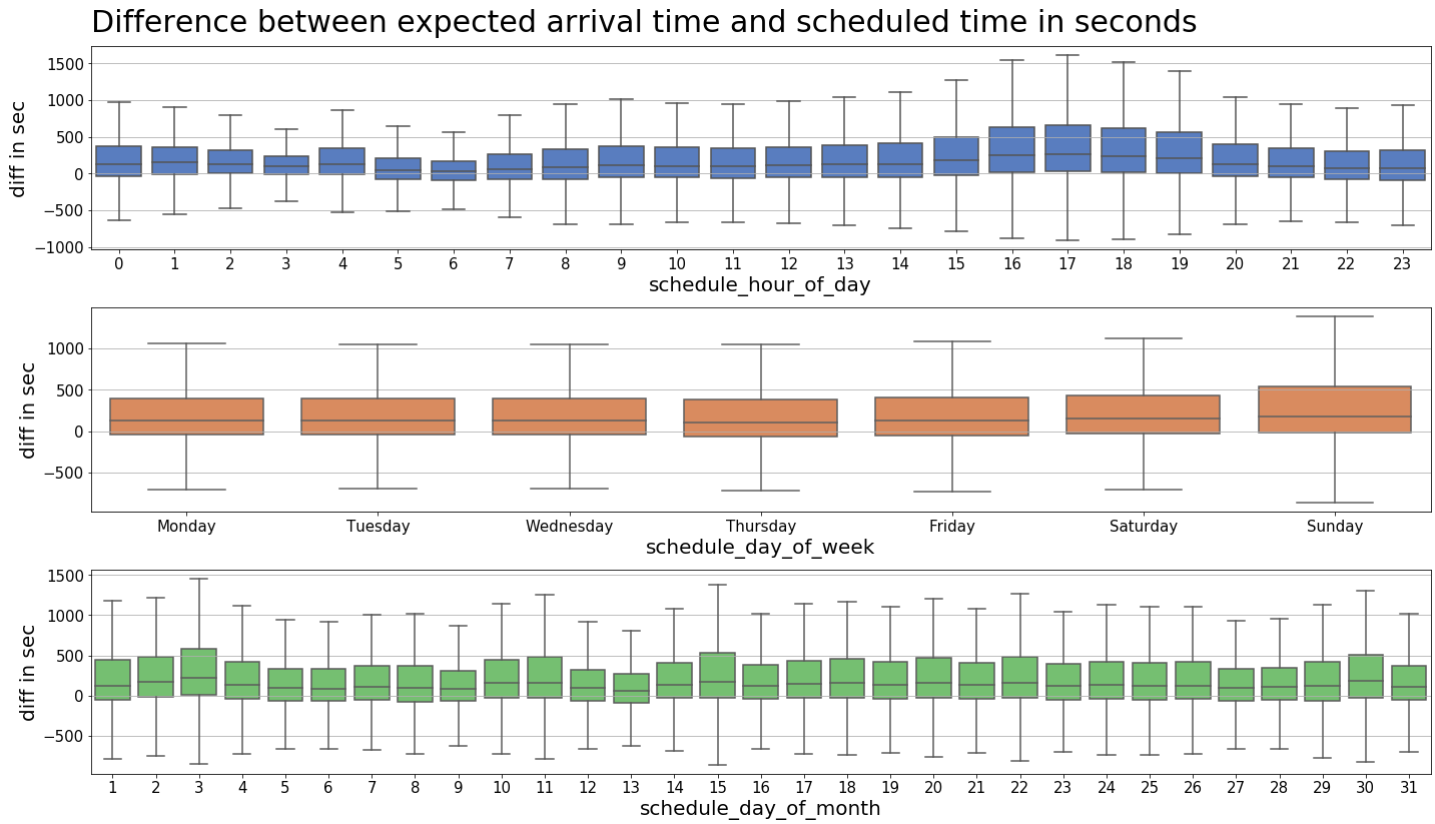
This is a different time difference

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 [109]:
```
# a plot about analysis, I suggest avoid this↔
```

In [110]:
```
# boxplot↔
```



In [111]:
```
# use this check the count↔
```

Sumary of EDA

We have spent a lot of time on EDA, which I called "a story about B6". We acted like a detector and used domain knowledge to explore information underlying. We explore the mechanism how the data is generated, evaluate how precise and truethful the data is, and 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 a way that 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 exhaustive, 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 truethful, the observation consistent with reality. And try to confirm and interprete the meaning of variable names.

In [112]:
```
# Just one more example to show how to practice the afforementioned principle. ↔
```

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 pattern. 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. The label is irrelevant to the unsupervised learning method, but rather something to provide metrics so that we can evaluate the unsupervised learning result. To be more specific, I will show you how to use PCA as a feature engeering tools and demonstrate its benifits by comparing the performance with PCA transformed data and without.

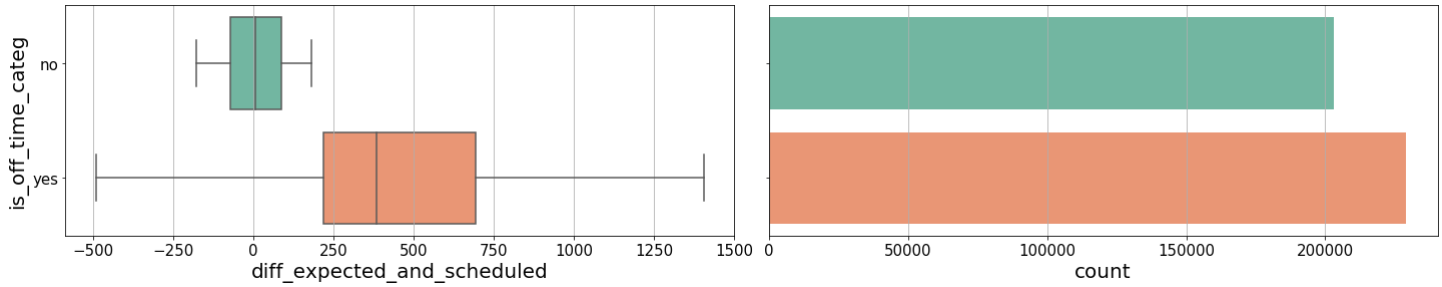
First of all let's assign some labels and design a supervised learning task.

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 [113]:
```
# customer function create target
# to-do-list, there is some repeating copy-and-paste data here, need to optimize that.
def create_df_with_target():
    df_trim = feature_engeering_dataset()

    df_trim['is_off_time'] = df_trim['diff_expected_and_scheduled'].apply(lambda x: 1 if np.abs(x)>3*60 else 0).astype('int32')
    df_trim['is_off_time_categ'] =  df_trim['is_off_time'].apply(lambda x: 'yes' if x==1 else 'no')

    return df_trim

df_trim = create_df_with_target()
```

In [114]:
```
# EDA one more time↔
```

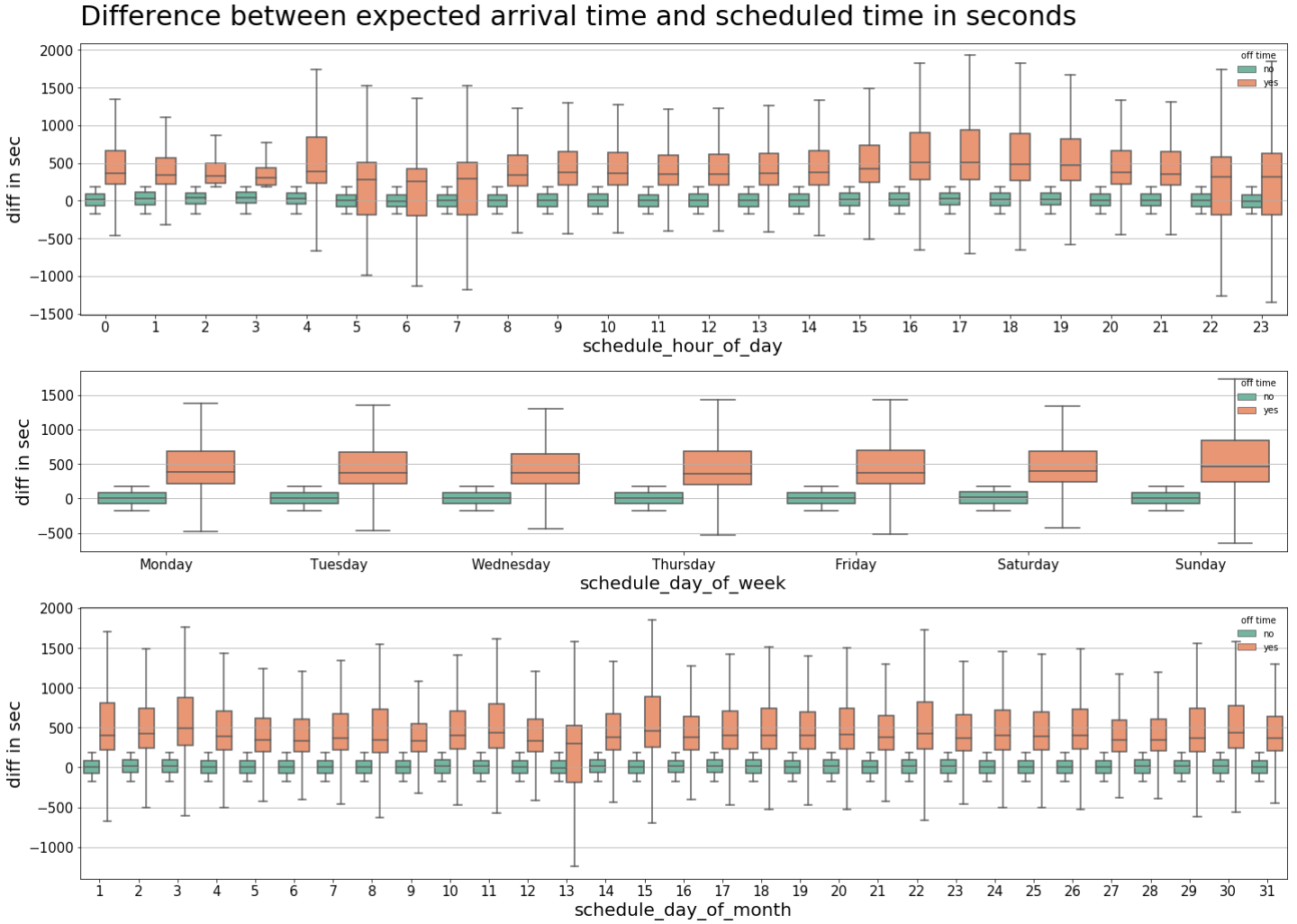


In [115]:
```
# detail alert↔
```

In [116]:
```
# proportion of label classes↔
```

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

In [117]:
```
# EDA cont'↔
```



In [118]:
```
# create df_work_on↔
```

In [119]:
```
# feature engineering↔
```

In [120]:
```
# df_work_on.info()
```

In [121]:
```
# df_work_on.columns
```

In [122]:
```
# df_X and df_y↔
```

In [123]:
```
# df_X.head(5)
```

In [124]:
```
df_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 122843 entries, 91 to 26520637
Data columns (total 19 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   RecordTime_month           122843 non-null  int32  
 7   RecordTime_day             122843 non-null  int32  
 8   RecordTime_weekday         122843 non-null  int32  
 9   RecordTime_hour            122843 non-null  int64  
 10  RecordTime_min             122843 non-null  int64  
 11  RecordTime_sec             122843 non-null  int64  
 12  schedule_month             122843 non-null  int64  
 13  schedule_day               122843 non-null  int64  
 14  schedule_hour_num          122843 non-null  int64  
 15  schedule_hour_num_overlap  122843 non-null  int64  
 16  schedule_day_num           122843 non-null  int64  
 17  schedule_weekday_num       122843 non-null  int64  
 18  schedule_minute_num        122843 non-null  int64  
dtypes: float64(6), int32(3), int64(10)
memory usage: 17.3 MB
```

In [125]:
```
df_X.head()
```

Out[125]:

	OriginLat	OriginLong	DestinationLat	DestinationLong	next_stop_Lat	next_stop_Long	RecordTime_month	RecordTime_day	RecordTime_weekday	RecordTime_hour	RecordTime_min	RecordTime_sec	schedule_month	schedule_d
91	40.666382	-73.883614	40.592949	-73.993385	40.665816	-73.883258	6	1	3	0	3	42	6	
297	40.666382	-73.883614	40.592949	-73.993385	40.633998	-73.947753	6	1	3	0	3	30	5	
518	40.593510	-73.993996	40.666420	-73.883385	40.666352	-73.883369	6	1	3	0	3	29	6	
742	40.593510	-73.993996	40.666420	-73.883385	40.666352	-73.883369	6	1	3	0	13	50	6	
934	40.593510	-73.993996	40.666420	-73.883385	40.641122	-73.904594	6	1	3	0	13	34	6	

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)

Train-test split

I choose the train:test=2:1 split. (test size = 0.33)

I was asked why I choose this value because the person is concerned this value is a bit higher.

The reasoning of this split ratio value is

- 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.

A side-note: it became a semi-supervised learning problem if we have small amount of labeled target. (final version I choose 1:99 split)

Machine Learning

Data preparation

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

back to main

In [186]:
```
# train test split
from sklearn.model_selection import train_test_split
df_X_train, df_X_test, df_y_train, df_y_test = train_test_split(df_X, df_y, test_size=0.99, random_state=RANDOM_STATE)
# df_X_train
```

PCA

In [187]:
```
# preprocessing scaling↔
```

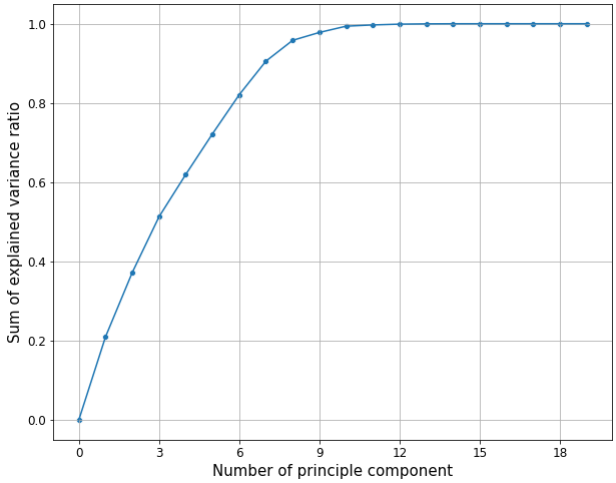
In [188]:
```
# pca to handle correlation and sparsity↔
```

In [189]:
```
# Note for myself↔
```

In [190]:
```
pca.explained_variance_ratio_.cumsum()
```

Out[190]:
```
array([0.2099846 , 0.37214232, 0.51264678, 0.61943965, 0.72204595,
       0.82064436, 0.90556719, 0.95809239, 0.97799397, 0.99380291,
       0.99694695, 0.99896263, 0.99970437, 0.99996065, 0.99999357,
       1.        , 1.        , 1.        , 1.        , 1.        ])
```

In [191]:
```
# plot explained_variance cumsum
```



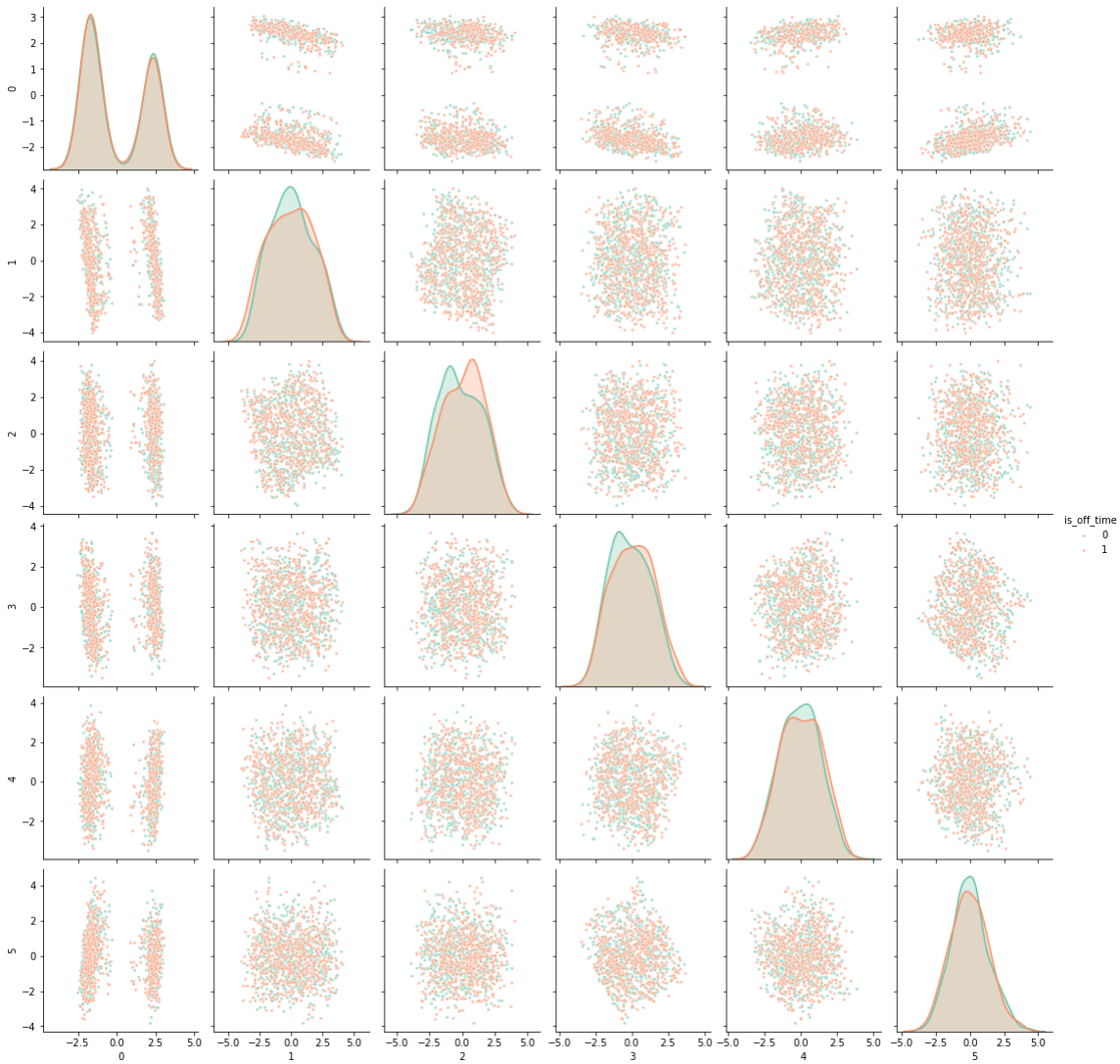
The first 10 principle components already explained 99% of the variance, so it is reasonable to just take the first 10 priciple componenets.

In [192]:
```
# choose the top PCA components with threshold = 0.99, here I kept them all without touch
```

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 [193]:
```
# pair plot, top 6 component from PCA
```

Out[193]:
```
<seaborn.axisgrid.PairGrid at 0x1be2df34e88>
```



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.

For this project I am not trying to demonstrate how to fine tune the machine learning models, so all the models will use their default hyper-parameter settings. Just you should know there is definetely better cross-validation schemes to help improve performance and justify evaluation.

Training Demo

In [194]:
```
from sklearn.metrics import f1_score
```

In [195]:
```
# copy training data here↔
```

try logistic regression

In [196]:
```
# 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_pred=lr1.predict(X_train_1))))
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=lr1.predict(X_test_1))))

```

```
f1-score on training set: 0.5577758470894875
f1-score on test set: 0.5196467760476539
```

Evaluation

back to main

In [197]:
```
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from sklearn.metrics import precision_recall_curve, average_precision_score
```

In [198]:
```
# Customer function # evaluation and visulization function
def model_evaluation(y_test, y_pred, y_pred_prob=None, is_print_classification_report=True):↔
```

In [199]:
```
# customer func for evaluation and visualization↔
```

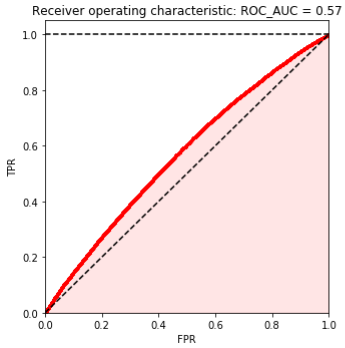
In [200]:
```
# customer functioin plot precision↔
```

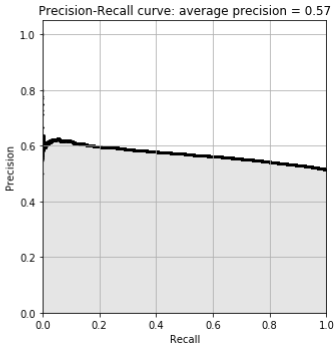
In [201]:
```
# Customer func for precision analysis↔
```

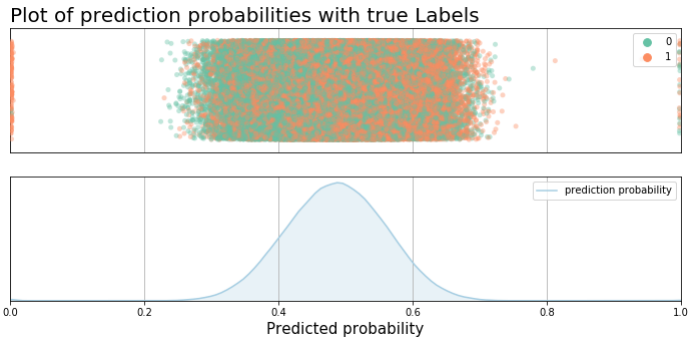
evaluation on logistic regression

In [202]:
```
# evaluation metrics and visualization↔
```

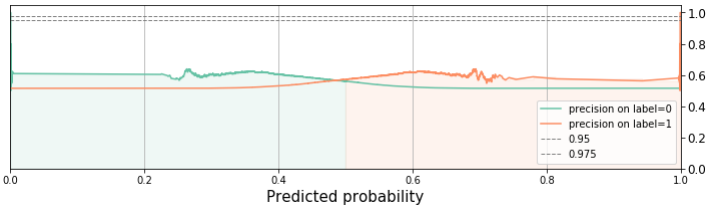
```
classification_report
              precision    recall  f1-score   support

           0       0.53      0.62      0.57     59008
           1       0.57      0.48      0.52     62607

    accuracy                           0.55    121615
   macro avg       0.55      0.55      0.55    121615
weighted avg       0.55      0.55      0.54    121615
```








```
When recall is 75%, Precison is:  0.547609884663378
```



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 am afraid I cannot go to the details about ROC. 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 predicted 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 predicted 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.

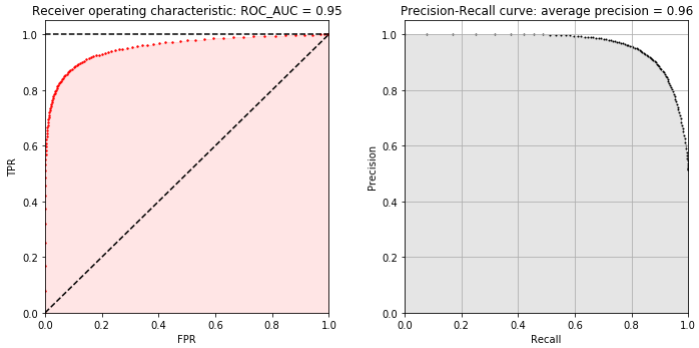
try random forest

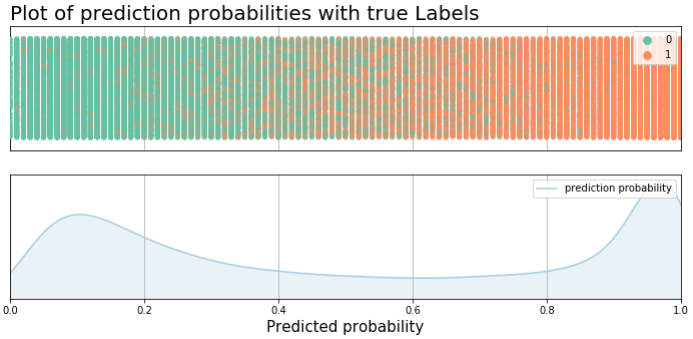
In [203]:
```
from sklearn.ensemble import RandomForestClassifier

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_pred=rfc1.predict(X_train_1))))
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=rfc1.predict(X_test_1))))
```

```
f1-score on training set: 1.0
f1-score on test set: 0.890582641870123
```

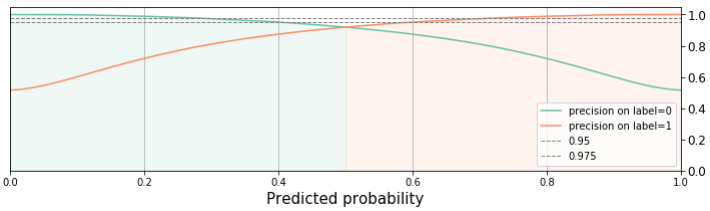
In [204]:
```
1 ▸ # evaluation metrics and visualization↔
```

```
classification_report
              precision    recall  f1-score   support

           0       0.86      0.92      0.89     59008
           1       0.92      0.86      0.89     62607

    accuracy                           0.89    121615
   macro avg       0.89      0.89      0.89    121615
weighted avg       0.89      0.89      0.89    121615
```






```
When recall is 75%, Precison is:  0.9732625862747943
```



In [205]:
```
1 ▸ # calculation cast rate at (0.2 to 0.8)↔
```

Out[205]:
```
0.3590099905439296
```

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 predicted 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 predicted probability is over 0.8, the precion on label=1 is over 0.975, and when the predicted 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 predicted probability increase, the precision on label=1 also increase, on the other hand, when the predicted 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 predicted probaility distribution, there is no color to tell you the actual label. But this should not

In [206]:
```
1 ▸ ### challenge for you↔
```

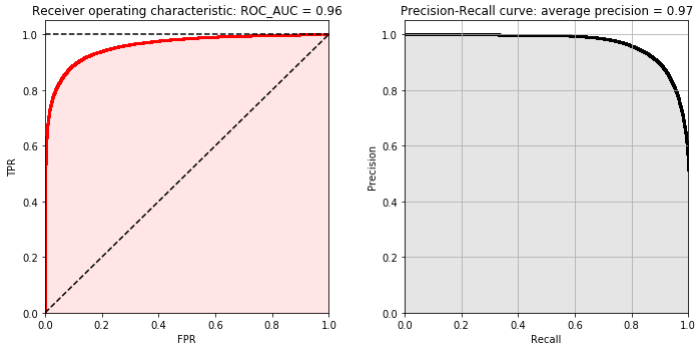
try XGBoost

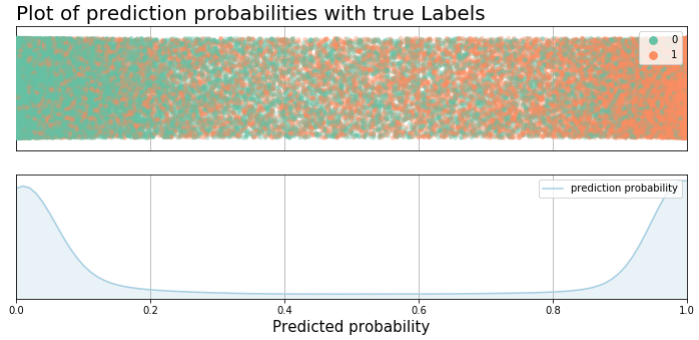
In [207]:
```
from xgboost import XGBClassifier

xgbc = XGBClassifier(random_state=RANDOM_STATE)
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_pred=xgbc.predict(X_train_1))))
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=xgbc.predict(X_test_1))))
```

```
f1-score on training set: 1.0
f1-score on test set: 0.8964839626471783
```

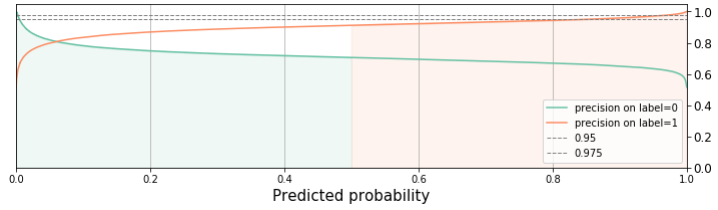
In [208]:
```
# evaluation metrics and visualization↔
```

```
classification_report
              precision    recall  f1-score   support

           0       0.88      0.91      0.89     59008
           1       0.91      0.88      0.90     62607

    accuracy                           0.90    121615
   macro avg       0.90      0.90      0.90    121615
weighted avg       0.90      0.90      0.90    121615
```






```
When recall is 75%, Precison is:  0.9751220018689648
```



In [209]:
```
# calculation cast rate at (0.2 to 0.8)↔
```

Out[209]:
```
0.09100851046334744
```

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 predicted 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 know if proability calibration doesn't make much sense to you)

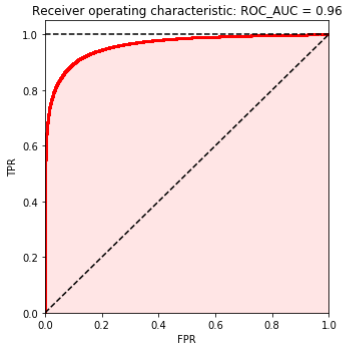
try lightGBM

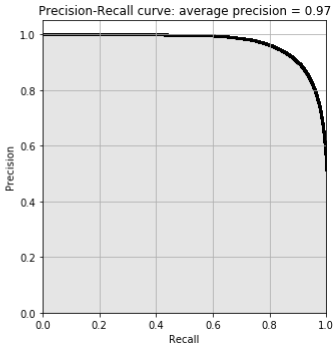
In [210]:
```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(random_state=RANDOM_STATE)
lgbm.fit(X=X_train_1, y=y_train_1.is_off_time.values)
lgbm.get_params

print("f1-score on training set: {}".format(f1_score(y_true=y_train_1, y_pred=lgbm.predict(X_train_1))))
print("f1-score on test set: {}".format(f1_score(y_true=y_test_1, y_pred=lgbm.predict(X_test_1))))
```

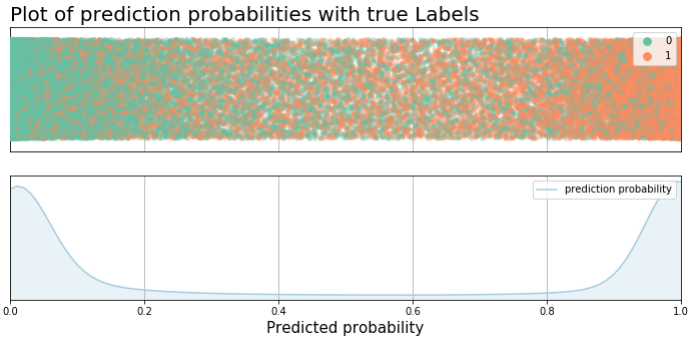
```
f1-score on training set: 1.0
f1-score on test set: 0.8985403069684618
```

In [211]:
```
# evaluation metrics and visualization
```

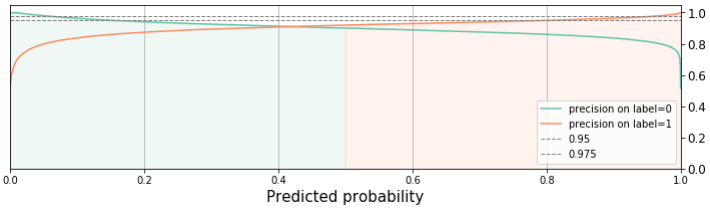
```
classification_report
              precision    recall  f1-score   support

           0       0.88      0.92      0.90     59008
           1       0.92      0.88      0.90     62607

    accuracy                           0.90    121615
   macro avg       0.90      0.90      0.90    121615
weighted avg       0.90      0.90      0.90    121615
```








```
When recall is 75%, Precison is:  0.9776394411942287
```



In [212]:
```
# calculation cast rate at (0.2 to 0.8), and (0.4 to 0.6)
```

```
0.09447025449163343
0.027504830818566788
```

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

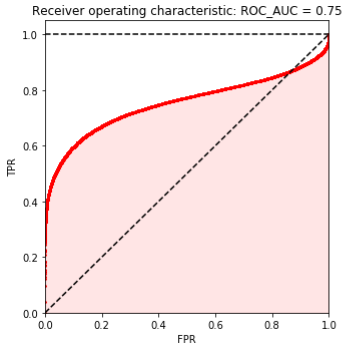
In [213]:
```
# X_train_2 = df_X_train.copy()
```

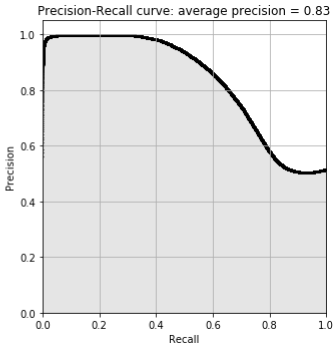
In [214]:
```
# logistic regression, train with the default model
```

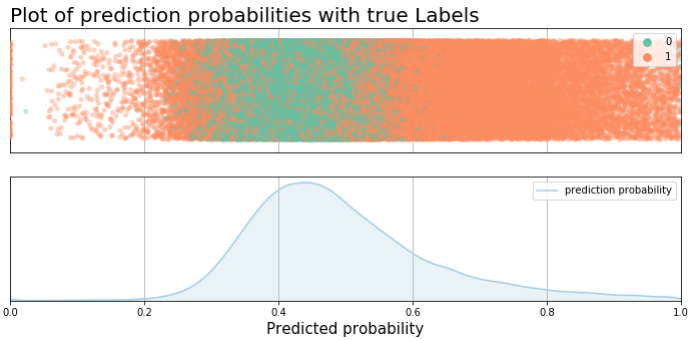
```
f1-score on training set: 0.730804810360777
f1-score on test set: 0.7183198244592719
```

In [215]:
```
# evaluation metrics and visualization
```

```
classification_report
              precision    recall  f1-score   support

           0       0.69      0.85      0.76     59008
           1       0.82      0.64      0.72     62607

    accuracy                           0.74    121615
   macro avg       0.75      0.74      0.74    121615
weighted avg       0.76      0.74      0.74    121615
```







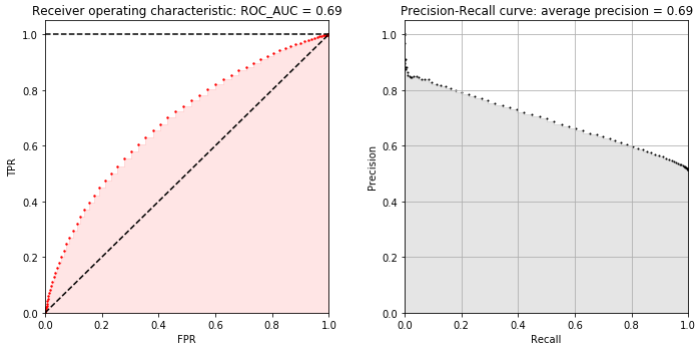

```
When recall is 75%, Precison is:  0.660538198596126
```

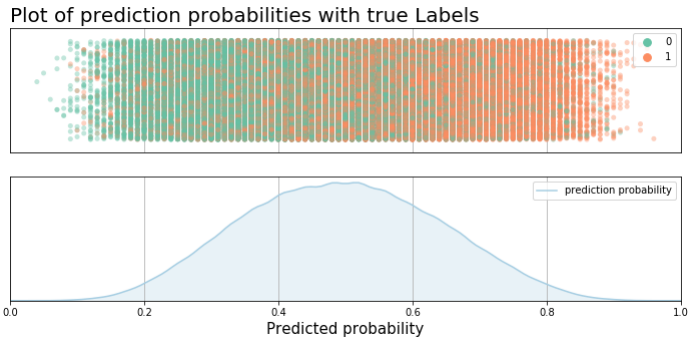
In [216]:
```
# random forest
```

```
f1-score on training set: 1.0
f1-score on test set: 0.6312185835239892
```

In [217]:
```
# evaluation metrics and visualization
```

```
classification_report
              precision    recall  f1-score   support

           0       0.62      0.67      0.64     59008
           1       0.66      0.60      0.63     62607

    accuracy                           0.64    121615
   macro avg       0.64      0.64      0.64    121615
weighted avg       0.64      0.64      0.64    121615
```





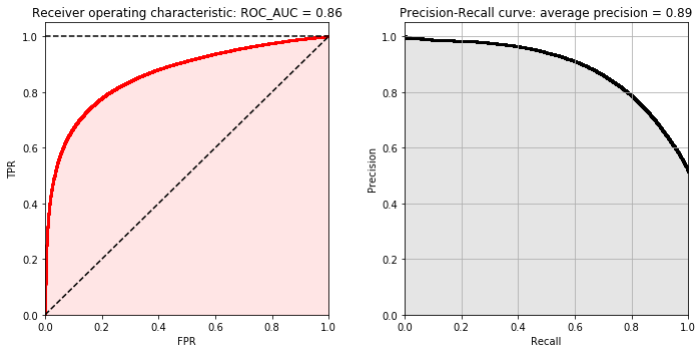

```
When recall is 75%, Precison is:  0.6153453020573975
```

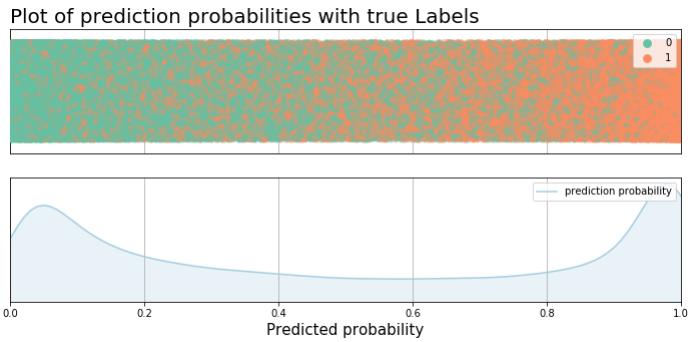
In [218]:
```
# xgboost
```

```
f1-score on training set: 1.0
f1-score on test set: 0.7900576570352276
```

In [219]:
```
# evaluation metrics and visualization
```

```
classification_report
              precision    recall  f1-score   support

           0       0.77      0.81      0.79     59008
           1       0.81      0.77      0.79     62607

    accuracy                           0.79    121615
   macro avg       0.79      0.79      0.79    121615
weighted avg       0.79      0.79      0.79    121615
```





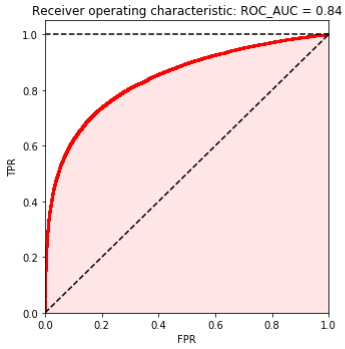

```
When recall is 75%, Precison is:  0.826605876036404
```

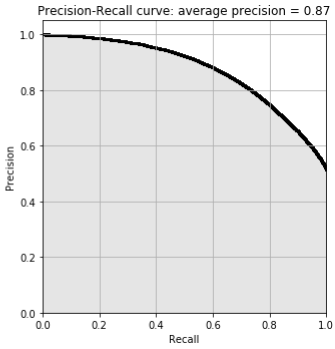
In [220]:
```
# lightGBM
```

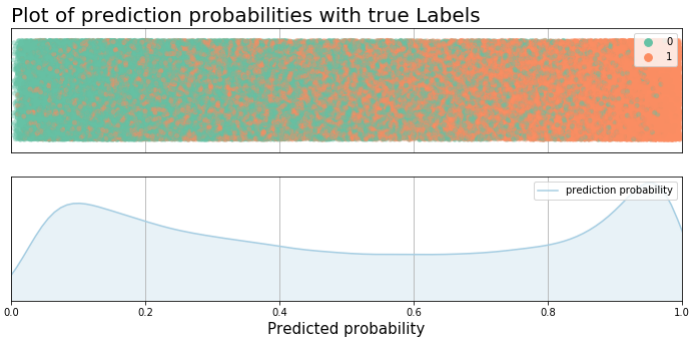
```
f1-score on training set: 0.9991659716430359
f1-score on test set: 0.7686482950363889
```

In [221]:
```
# evaluation metrics and visualization
```

```
classification_report
              precision    recall  f1-score   support

           0       0.75      0.79      0.77     59008
           1       0.79      0.75      0.77     62607

    accuracy                           0.77    121615
   macro avg       0.77      0.77      0.77    121615
weighted avg       0.77      0.77      0.77    121615
```








```
When recall is 75%, Precison is:  0.788001342507132
```

Compare them all

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

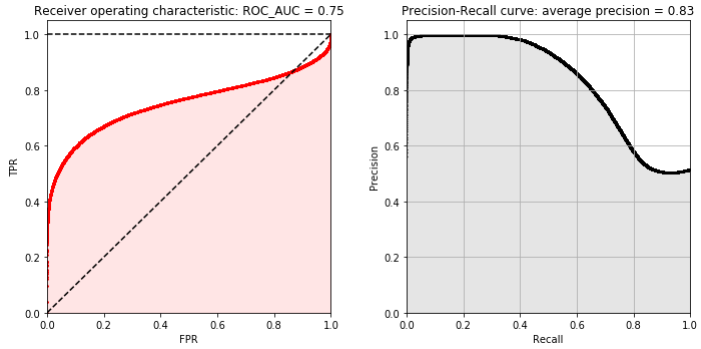
- Logistic regression without PCA
- Random forest without PCA
- XGBoost without PCA
- LightGBM without PCA

then

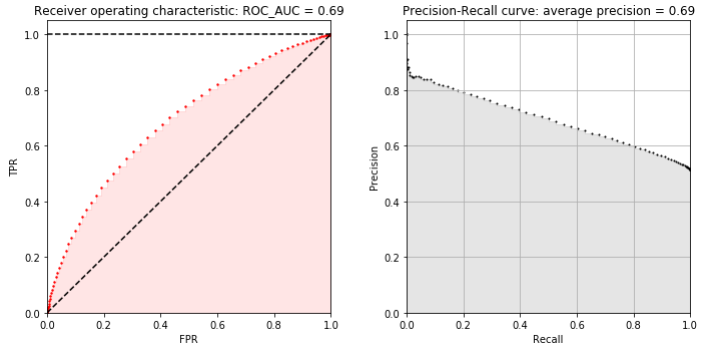
- Logistic regression with PC
- Random forest with PCA
- XGBoost with PCA
- LightGBM with PCA

In [222]:
```
1 ▸ # comapre them all↔
```

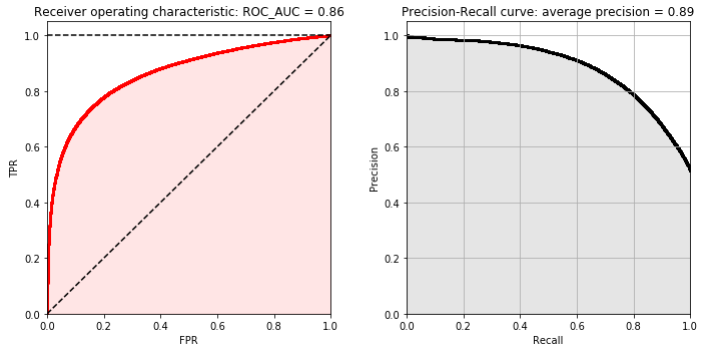
Logistic regression without PCA



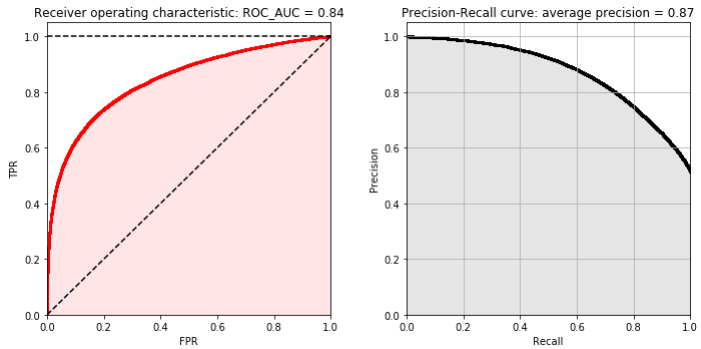
Random forest without PCA



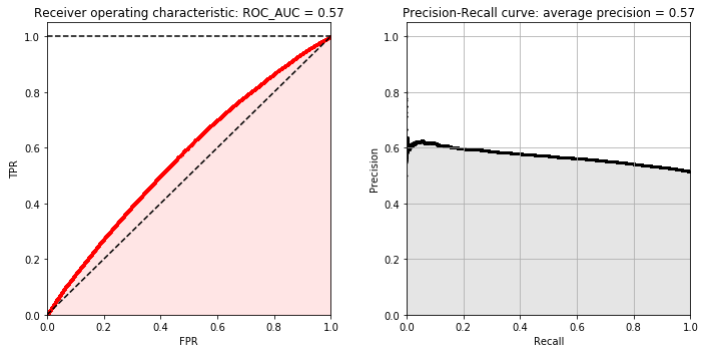
XGBoost without PCA



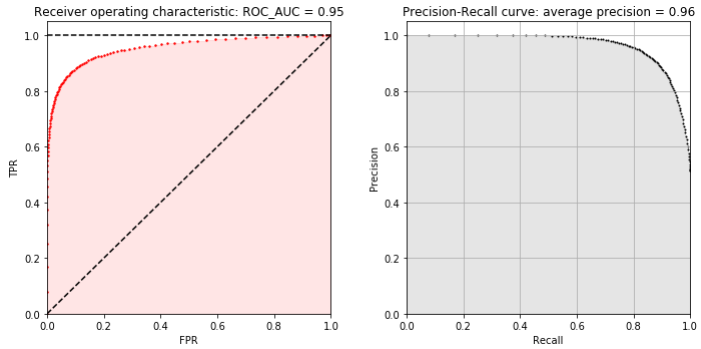
LightGBM without PCA



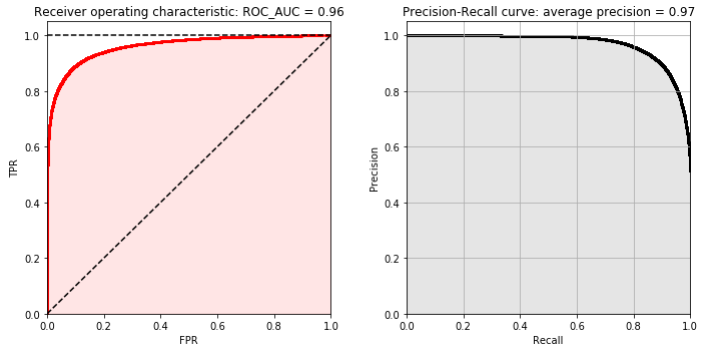
Logistic regression with PCA



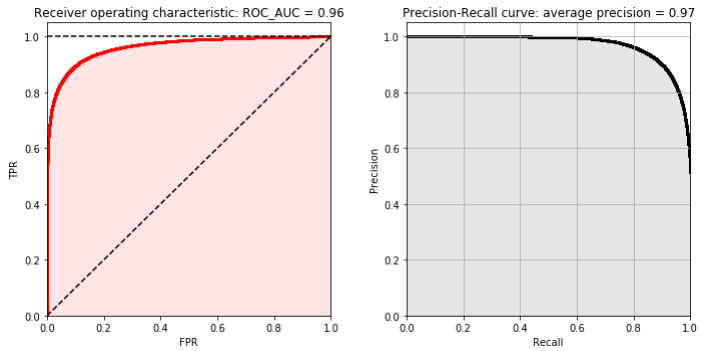
Random forest with PCA



XGBoost with PCA



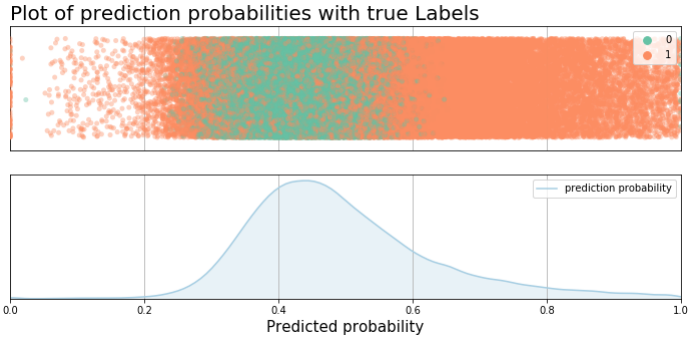
LightGBM with PCA



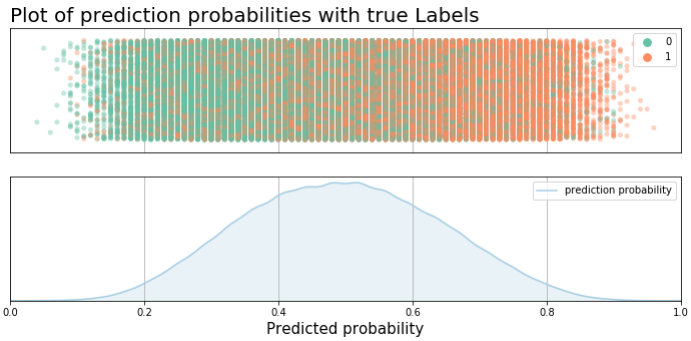
One more time, the predicted probability

In [223]:
```
1 ▸ # comapre↔
```

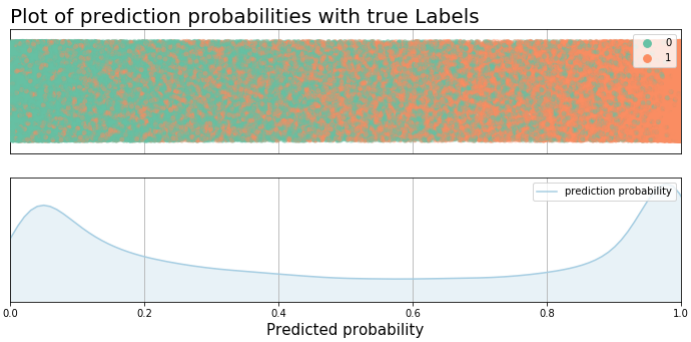
```
Logistic regress without PCA
```



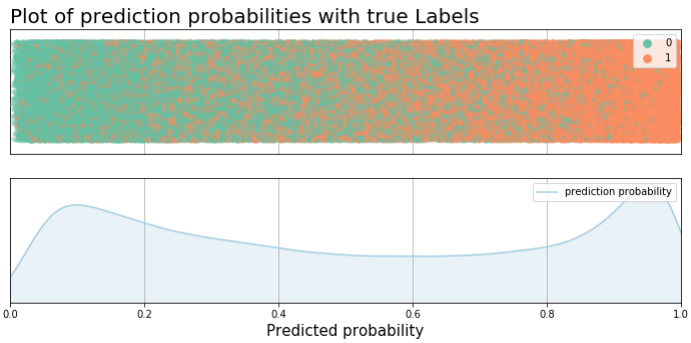

```
Random forest without PCA
```



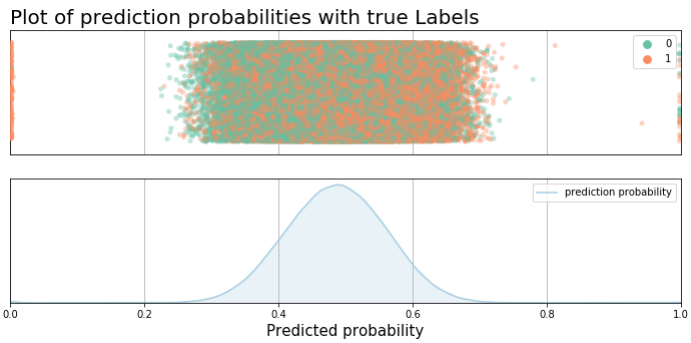

```
XGBoost without PCA
```



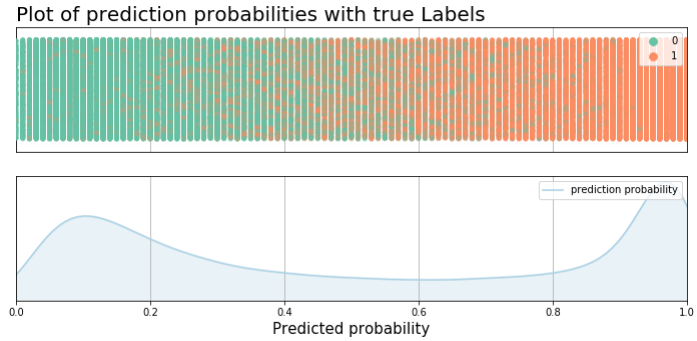

```
LightGBM without PCA
```



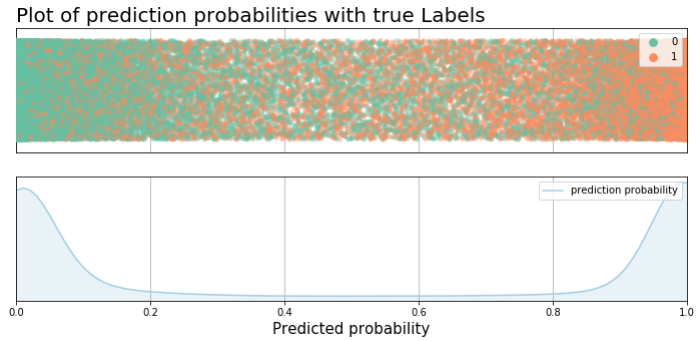

```
Logistic regress with PCA
```



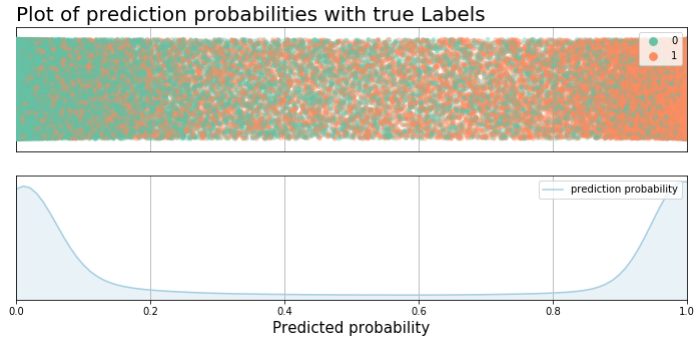

```
Random forest with PCA
```



XGBoost with PCA



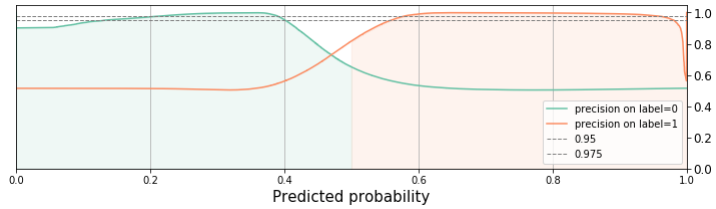
LightGBM with PCA



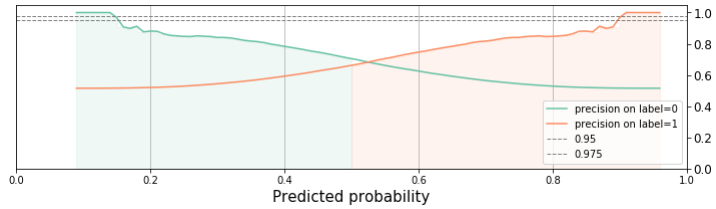
Sorry, I lied. One last comparison

In [224]:
```
# compare↔
```

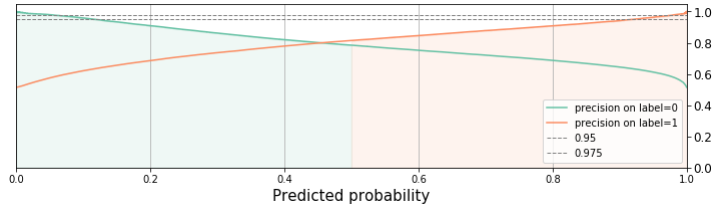
Logistic regress without PCA



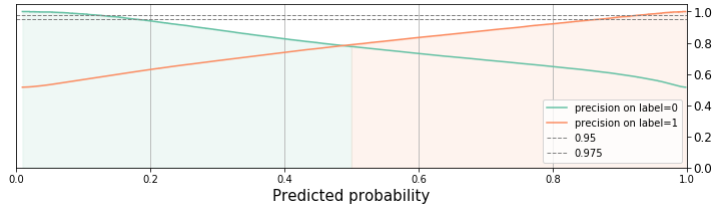
Random forest without PCA



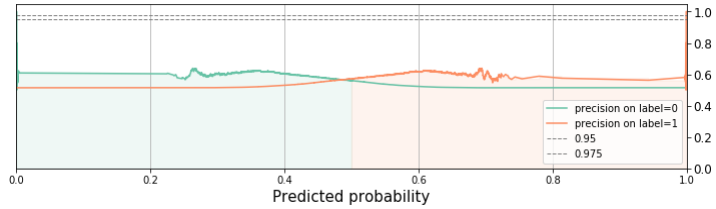
XGBoost without PCA



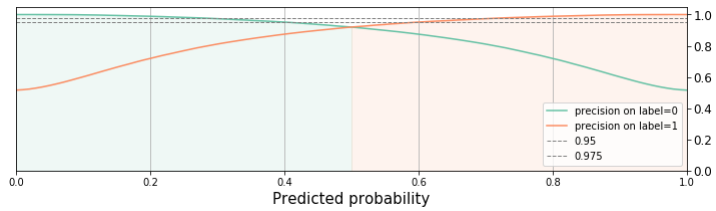
LightGBM without PCA



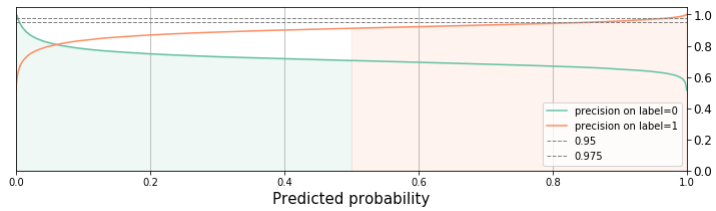
Logistic regress with PCA



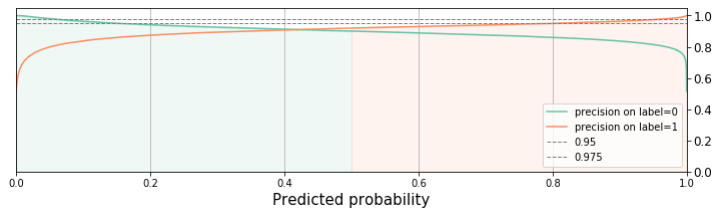
Random forest with PCA



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 [225]:
```
# check the feature importance, lightGBM
```

Out[225]:

	index	features	importance
0	10	RecordTime_min	610
1	18	schedule_minute_num	586
2	11	RecordTime_sec	358
3	4	next_stop_Lat	307
4	7	RecordTime_day	236
5	9	RecordTime_hour	204
6	5	next_stop_Long	179
7	8	RecordTime_weekday	152
8	15	schedule_hour_num_overlap	81
9	6	RecordTime_month	63
10	0	OriginLat	44
11	14	schedule_hour_num	41
12	3	DestinationLong	40
13	13	schedule_day	35
14	1	OriginLong	32
15	2	DestinationLat	31
16	17	schedule_weekday_num	1
17	12	schedule_month	0
18	16	schedule_day_num	0

In [226]:
```
# check the feature importance, xgboost using PCA features
```

Out[226]:

	index	features	importance
0	15	15	0.308280
1	8	8	0.120078
2	12	12	0.062005
3	0	0	0.056760
4	6	6	0.040000
5	14	14	0.039892
6	3	3	0.038475
7	17	17	0.035619
8	7	7	0.034730
9	1	1	0.034473
10	13	13	0.032441
11	2	2	0.032415
12	5	5	0.027722
13	10	10	0.025563
14	11	11	0.025060
15	16	16	0.024424
16	4	4	0.023494
17	9	9	0.022160
18	18	18	0.016411

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.

Thoughts and Discussions about unconventional unsupervised learning

For machine learning part, I expanded our analysis on predicted probability distribution and include that to the evaluation metrics family (i.e. we can consider cast rate is an evaluation metrics). I also demonstrate how to use predicted probability to evaluate unsupervised learning performance. i.e. under certain probability, say, 0.2 to 0.8, the lower the cast rate, the better the performance. When talking about unsupervised learning, I went back to the essential meaning of unsupervised learning, that is, to find some pattern that might or might not do some work while without any label as guidance. (To me, it is very much like a feature engineering process). To be more specific, I used both unsupervised “human” learning to extract features and PCA as my unsupervised learning demonstration. As you can see, during neither of the process there is label involved. Labels are only come into sight as a metrics to assure us we are not doing something useless. I called it “unconventional unsupervised” as many people referred supervised learning as clustering and dimensionality reduction. I purposely kept all the PCA component to make a point that my purpose is not dimensionality reduction. (I admit this is not the best option though.) The result shows that the pattern generated from PCA works better than without.

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.
- Explore 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.
- Explore the method for original feature interpretability. i.e. how to extract feature importance from PCA features.
- Find way to imporve my coding capability.
- Explore supervised learning and evaluation method on mutliclass classification and unbalanced label classsification (anomoly detection? i.e. recall the correlationship plot on expected to arrive time and distance from stop, what if we try to predict those outliers?)

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 [ ]:
```
```