# 城市治理大数据应用创意赛

# “行程时间管家”—基于大数据的乘车时间预估软件

# 大纲

# 1. 引言

# 2. 市场背景分析

# 3. 产品介绍

# 4. 核心技术

### 4.1 数据使用 ¶

- 4.1.1 数据
- 4.1.2 问题定义

### 4.2 技术架构

### 4.3 技术细节

#### 4.3.1 数据导入与初步分析

- 4.3.1.1 导入工具包
- 4.3.1.2 导入原始数据
- 4.3.1.3 数据初步分析+清洗

#### 4.3.2 深度数据分析+特征工程

- 4.3.2.1 行车记录与provider的类型(vendor_id)
- 4.3.2.2 上车的时间
- 4.3.2.3 上车的人数
- 4.3.2.4 上车的地点
- 4.3.2.5 起点与终点的距离
- 4.3.2.6 街道的数量
- 4.3.2.7 其他特征
- 4.3.2.8 对数据集3的具体分析

#### 4.3.3 模型训练与预测

- 4.3.3.1 模型的训练
- 4.3.3.2 模型的测试

#### 4.3.4 参考资料

# 5. 原型测试

- 5.1 详细说明原型的使用方法
- 5.2 测试用例
- 5.3 原型局限性
- 5.4 改进方向

## 6. 价值导向

- 6.1 商业价值
- 6.2 社会价值

## 7. 代码附件

# 1. 引言

近年来，我国在大力提倡和发展智慧城市。在建设智慧城市的过程中，有一个非常重要的内容就是要发展智能交通。智能交通的发展可以大大提高人们出行的效率和满意度，同时智能交通可以节省能源，减少城市污染，帮助城市建设和管理者更好地建设发展智慧城市。 然而，众所周知，交通拥堵、交通污染等依然是城市智能化进程中急需解决的问题。在智慧城市的建设中产生的大量丰富的数据，如交通卡口的摄像，公交车的出行和城市交通流量的实时检测等数据，为我们解决交通问题提供了新的思路和方法。借助这些海量交通数据，我们可以精确预测出行时间，作为出行者规划出行选择、管理者制定统筹决策的可靠参考工具，以此缓解城市拥堵，节省能源，助力智慧城市的建设。

# 2. 市场背景分析

据公安部交通管理局统计，截至 2014 年底，我国机动车驾驶员已突破 3 亿大关，全国汽车保有量达2.6 亿辆，驾驶人数量位列世界第一。由于车辆保有量的快速增长使得高速公路交通需求量剧增，加剧了诸如交通拥堵、道路安全、环境污染等问题，影响了城市的可持续发展。根据中国交通部发表的数据显示，全国因交通拥堵带来的日经济损失约 **80** 亿元，交通拥堵时，车辆在道路上的平均时速为**10-15km/h**，北京每日人均延误约一个小时。交通拥堵还带来了一系列的生态环境问题，北京由于拥堵每日多排放二氧化碳 **1.7** 万吨，颗粒物等有害颗粒物或气体约 **9** 吨，北京每年由于交通拥堵带来的生态环境污染损失约为 **46** 亿。

有效的交通信息不仅可以为交通管理部门提供决策支持，引导合理的交通行为模式，使交通行为规范化，提高道路网络的运行效率，还可以为广大出行者提供各种便利，如：出行者利用网络、无线广播、可变信息情报板可变标志、车辆导航等提供的各种交通信息，选择与出行相关的信息如出行方式、路线选择，以便顺利、快速地到达目的地。因此，交通管理部门及普通大众对交通信息的需求越来越高。

出行时间作为一种重要的交通信息，对缓解城市交通拥堵、解决城市交通问题具有至关重要的意义。先进的交通管理系统以出行时间的预测结果为决策依据，可以对道路交通运行状态进行识别、对道路交通流进行控制、对紧急事件做出快速反应，达到缩短出行时间、减少交通事故、降低能耗，提高交通管理水平的目标。此外，出行时间的预测结果可为出行者或驾驶员提供多种出行计划以及路线引导，通过改变出行路线或出行时间来避免交通拥堵，从而大大减少交通出行的盲目性。欧洲国家、美国、日本等国家都有明确的智能交通系统的应用效益预期指标，其中减少出行者的出行时间是非常重要的指标。

目前，市场上对出行时间预测这一交通信息的应用主要可以分为三大类。第一类是出行者个人使用，如人们常常在出行前使用百度地图，腾讯地图和高德地图等**AP**查询自己的行程所需的时间，根据**APP**给出的相应的行程预测时间，决定自己的出发时间和选择出行方式等。第二类是企业层面的应用，主要是出租车公司，网约车公司如滴滴打车、神州专车等，这些运营商在自己的软件中嵌套了出行时间预测算法来规划出行路线，预测每个出行订单的所需的时间。第三类是政府层面的应用，交通相关的政府管理和决策人员需要参考交通出行时间具体制定相应的管理调度策略，调控改善整个城市的交通状况。

出行时间的预测这一交通信息非常重要，它是智能交通系统中的重要组成部分，它已经成为人们生活不可见但是必不可少的信息。虽然有众多学者和互联网企业对出行时间预测都进行过研究，但是目前存在的和市面上使用的预测出行时间的算法和模型都还很粗糙，预测得到的出行时间并不够精确，因此，研究如何精准地预测出行时间是非常有意义的和市场的。

基于此，我们采用大数据分析技术设计了“行程时间管家”这一精确估计乘车时间的算法。该算法通过以往大量的出租车、**Uber**、滴滴车的载客记录，采用大数据模型进行训练和学习，精准地预测从出发地到目的地的驾车的所需的时间，从而帮助出行者更好的安排自己的时间。

# 3. 产品介绍

本次方案中，我们小组设计编写了用于精确预测行程时间的算法，该算法是基于数据的，主要是通过对大量的出租车历史载客数据进行深度数据分析，提取特征工程，之后选用合适的模型进行训练，使其能够精准地预测出行时间。

本产品的目标用户是需要借助预测的出行时间信息来规划出行的人们，企业和交通管理人员。主要是希望通过精准地预测出行所需的时间帮助人们更好地规划自己的出行，减少不必要的等待时间和出行费用；帮助出租车、滴滴等企业精准地知晓订单所需出行时间，更好地规划出行路线和接单计划；帮助交通相关的政府管理和决策人员更好地制定调控策略，改善城市交通。

# 4. 核心技术

## 4.1 数据使用与问题定义

### 4.1.1 数据

- 本次比赛，我们选用的是Kaggle上的纽约的出租车数据，具体的链接如下：
- 数据集链接1: https://www.kaggle.com/c/nyc-taxi-trip-duration/data (https://www.kaggle.com/c/nyc-taxi-trip-duration/data);
- 数据集链接2:https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm (https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm)
- 数据集链接3:https://www.kaggle.com/atmarouane/nyc-taxi-trip-noisy/data (https://www.kaggle.com/atmarouane/nyc-taxi-trip-noisy/data)

### 4.1.2 问题定义

问题描述

通过分析数据集合中上百万条的出租车载客记录，城市交通情况等信息，来预估用户从出发地到目的地所需的精确时间。

评估指标

为了减少奇异值带来的影响，此处我们选用(RMSLE)Root Mean Squared Logarithmic Error: https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError (https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError) 作为我们的评估指标。

我们的问题的定义已经相对具体，所以我们会重点介绍算法分析与模型构造部分。

## 4.2 技术架构

## 4.3 技术细节

### 4.3.1 数据导入与初步分析

#### 4.3.1.1 导入工具包

为了方便后续的所有数据的处理，我们先导入常用的可视化和数据分析的工具包。

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import urllib
import re
import datetime
import calendar
import time
import scipy
import math
import seaborn as sns
import os
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
import xgboost as xgb
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings('ignore')
plotly.offline.init_notebook_mode()
%matplotlib inline
```

#### 4.3.1.2 导入原始数据

每一维度的意思为：

- id: 对应一笔订单的标志
- vendor_id: 行程记录与司机的关系
- pickup_datetime: 用户上车的时间
- dropoff_datetime: 用户下车的时间
- passenger_count: 上车的用户数量
- pickup_longitude: 上车地点的经度
- pickup_latitude: 上车地点的纬度
- dropoff_longitude: 下车地点的经度
- dropoff_latitude: 下车地点的纬度
- store_and_fwd_flag: 原文的意思,This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- total_distance: 从出发地点到达目的地的距离
- total_travel_time: 汽车的行驶时间(除去红绿灯不行驶的时间)
- number_of_steps: 有多少个路口(从出发点到目的地一共的行程数)
- trip_duration: 用户从出发地点到达目的地一共用了多长时间,单位是秒

In [2]:

```
train1 = pd.read_csv('./data/fastest_routes_train_part_1.csv')
train2 = pd.read_csv('./data/fastest_routes_train_part_2.csv')
train_ = pd.concat([train1, train2])
train_add = train_[['id', 'total_distance', 'total_travel_time', 'number_of_steps']]
train = pd.read_csv('./data/train.csv')
train = pd.merge(train, train_add, on = 'id', how = 'left')
train_df = train.copy()
train_df.head()
```

Out[2]:

| | id | vendor_id | pickup_datetime | dropoff_datetime | passenger_count | p |
|---|---|---|---|---|---|---|
| **0** | id2875421 | 2 | 2016-03-14 17:24:55 | 2016-03-14 17:32:30 | 1 | - |
| **1** | id2377394 | 1 | 2016-06-12 00:43:35 | 2016-06-12 00:54:38 | 1 | - |
| **2** | id3858529 | 2 | 2016-01-19 11:35:24 | 2016-01-19 12:10:48 | 1 | - |
| **3** | id3504673 | 2 | 2016-04-06 19:32:31 | 2016-04-06 19:39:40 | 1 | - |
| **4** | id2181028 | 2 | 2016-03-26 13:30:55 | 2016-03-26 13:38:10 | 1 | - |

In [3]:

```
train_df['vendor_id'].unique()
```

Out[3]:

```
array([2, 1], dtype=int64)
```

#### 4.3.1.3 数据初步分析+清洗

- 对数据中的记录进行检查分析，判断是否存在奇异值。我们主要研究最有可能出现奇异值的下列一些特征：

> id(记录的id)
> passenger_count(乘车的用户数量)
> trip_duration(用户从出发地点到达目的地一共用了多长时间,单位是秒)
> pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude(用户上下车的经纬度)
> total_distance(车总的行驶距离)
> total_travel_time(车行驶的时间,不包括等红绿灯之类的时间)
> number_of_steps(街道拐弯数)

- 对分析得到的奇异值的情况进行集中删除和纠正处理。

下面我们就先对数据中的记录进行检查分析，判断是否存在奇异值。

**id(每个出租车的记录id)**

- 是否唯一，是否存在缺失值，如果不唯一考虑删除重复id，如果数据的缺失情况严重则考虑删除特征等。
- 我们发现记录的id是唯一的，但是数据中一共出现了3个缺失值，其中total_distance、total_travel_time、number_of_steps分别缺失了一个，不算很严重，所以暂时不做处理。

In [4]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1458644 entries, 0 to 1458643
Data columns (total 14 columns):
id                    1458644 non-null object
vendor_id             1458644 non-null int64
pickup_datetime       1458644 non-null object
dropoff_datetime      1458644 non-null object
passenger_count       1458644 non-null int64
pickup_longitude      1458644 non-null float64
pickup_latitude       1458644 non-null float64
dropoff_longitude     1458644 non-null float64
dropoff_latitude      1458644 non-null float64
store_and_fwd_flag    1458644 non-null object
trip_duration         1458644 non-null int64
total_distance        1458643 non-null float64
total_travel_time     1458643 non-null float64
number_of_steps       1458643 non-null float64
dtypes: float64(7), int64(3), object(4)
memory usage: 166.9+ MB
```

In [5]:

```
if train_df.id.nunique() == train_df.shape[0]:
    print("训练集的id是唯一的")
print("一共有缺失值 - {}.".format(train_df.isnull().sum().sum()))
```

```
训练集的id是唯一的
一共有缺失值 - 3.
```

**passenger_count(乘车的用户数量)**

- 我们发现有的乘车用户数量是0，这个可能是异常情况，因为不太可能出现一个订单中没有用户上车的情况，除非是货车。

我们对于上述两个情况进行进一步的数据分析。

In [6]:

```
train_df.describe()
```

Out[6]:

| | vendor_id | passenger_count | pickup_longitude | pickup_latitude | dropo |
|---|---|---|---|---|---|
| **count** | 1.458644e+06 | 1.458644e+06 | 1.458644e+06 | 1.458644e+06 | 1.458 |
| **mean** | 1.534950e+00 | 1.664530e+00 | -7.397349e+01 | 4.075092e+01 | -7.397 |
| **std** | 4.987772e-01 | 1.314242e+00 | 7.090186e-02 | 3.288119e-02 | 7.064 |
| **min** | 1.000000e+00 | 0.000000e+00 | -1.219333e+02 | 3.435970e+01 | -1.219 |
| **25%** | 1.000000e+00 | 1.000000e+00 | -7.399187e+01 | 4.073735e+01 | -7.399 |
| **50%** | 2.000000e+00 | 1.000000e+00 | -7.398174e+01 | 4.075410e+01 | -7.397 |
| **75%** | 2.000000e+00 | 2.000000e+00 | -7.396733e+01 | 4.076836e+01 | -7.396 |
| **max** | 2.000000e+00 | 9.000000e+00 | -6.133553e+01 | 5.188108e+01 | -6.133 |

我们进一步分析该问题,发现这样的情况并不多见，一共才出现60次，占据了所有记录的万分之一都不到，所以我们可以认为这些是奇异值，可以直接将这类记录进行删除。

In [7]:

```
train_df[train_df['passenger_count'] == 0].shape
```

Out[7]:

(60, 14)

In [8]:

```
60 / train_df.shape[0]
```

Out[8]:

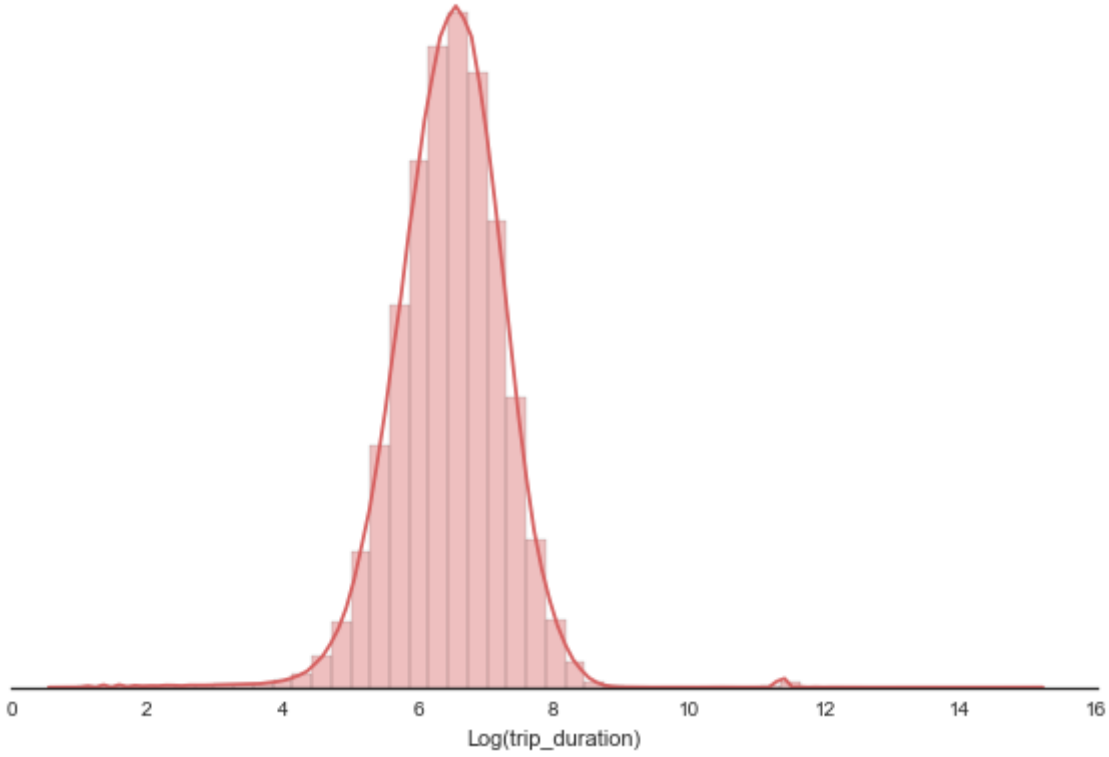
4.1134094405488936e-05

**trip_duration(用户从出发地点到达目的地一共用了多长时间，单位是秒)**

- 我们的行车记录中发现有些的行程时间居然是3.526282e+06 / 3600 = 979个小时，不太可能有乘客一次坐979个小时的车，所以行程时间中也存在奇异的情况。同时我们发现车开的做多时间也就是2小时。所以我们首先考虑将这一类的记录通过相近信息进行还原，如果不行则直接将记录删除。
- 通过绘制的坐车时间我们发现行车记录时间整体上是符合正态分布的，但是在两端处都有一些极端情况。我们还发现坐车的时间超过一天的一共有4个记录，我们认为这个很大可能是奇异值，所以考虑将这四条记录删除。

In [9]:

```
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
sns.despine(left=True)
sns.distplot(np.log(train_df['trip_duration'].values+1), axlabel = 'Log(trip_duration)', label =
 'log(trip_duration)', bins = 50, color="r")
plt.setp(axes, yticks=[])
plt.tight_layout()
```




In [10]:

```
train_df[train_df['trip_duration']>=24*3600].shape
```

Out[10]:

(4, 14)

In [11]:

```
train_df[train_df['trip_duration']>=24*3600]
```

Out[11]:

| | id | vendor_id | pickup_datetime | dropoff_datetime | passenger_co |
|---|---|---|---|---|---|
| **355003** | id1864733 | 1 | 2016-01-05 00:19:42 | 2016-01-27 11:08:38 | 1 |
| **680594** | id0369307 | 1 | 2016-02-13 22:38:00 | 2016-03-08 15:57:38 | 2 |
| **924150** | id1325766 | 1 | 2016-01-05 06:14:15 | 2016-01-31 01:01:07 | 1 |
| **978383** | id0053347 | 1 | 2016-02-13 22:46:52 | 2016-03-25 18:18:14 | 1 |

我们可以通过用户上车和下车的时间来尝试进行还原。

- 实际中我们发现通过用户上下车时间计算出来的结果和我们的trip_duration是100%吻合的，所以有很大的理由相信我们的数据至少在trip_duration这一块是没有问题的，而且我们也发现确实这些大的trip_duration的出租车也开了10多天，所以我们选择保留这些数据。

In [12]:

```
train_df['pickup_datetime'] = pd.to_datetime(train_df.pickup_datetime)
train_df['dropoff_datetime'] = pd.to_datetime(train_df.dropoff_datetime)
train_df['diff_datetime'] = train_df['dropoff_datetime'] - train_df['pickup_datetime']
train_df.loc[:, 'traval_time'] = (train_df['diff_datetime'].dt.seconds) + (train_df['diff_dateti
me'].dt.days) * 24 * 3600
```

In [13]:

```
train_df.loc[:, 'gap'] = train_df.loc[:, 'traval_time'] - train_df.loc[:, 'trip_duration']
```

In [14]:

```
np.sum(train_df['gap'] == 0) / train_df.shape[0]
```

Out[14]:

1.0

In [15]:

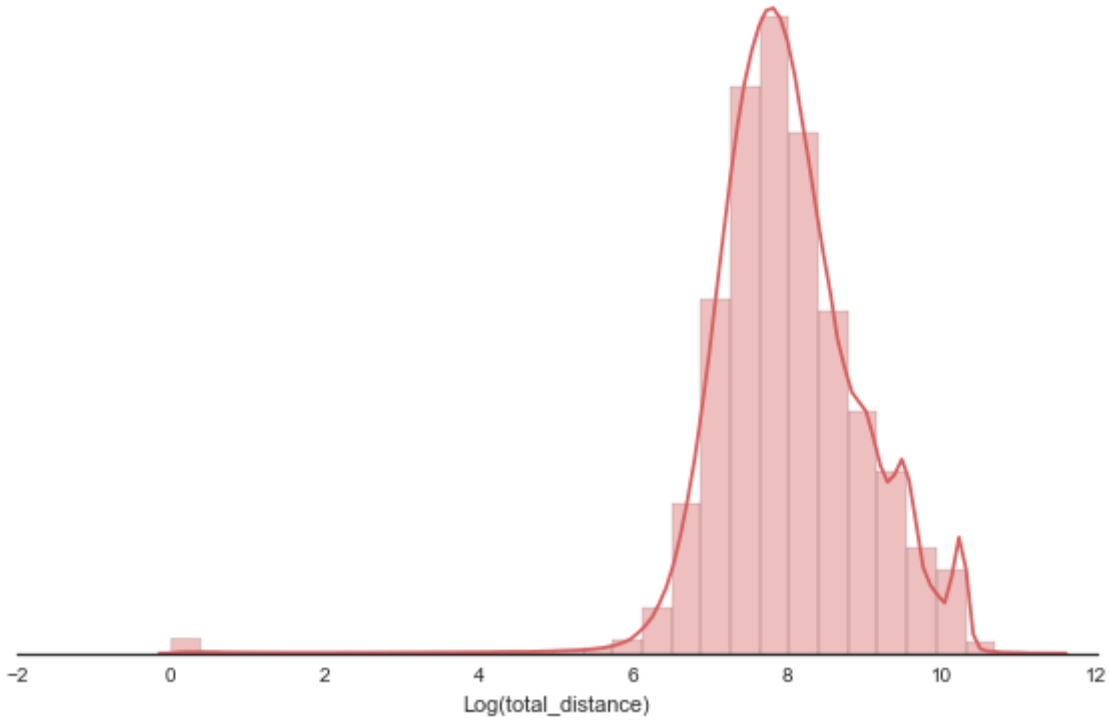
```
train_df = train_df.drop('traval_time',axis=1)
train_df = train_df.drop('gap',axis=1)
train_df = train_df.drop('diff_datetime',axis=1)
```

**total_distance(车总的行驶距离)**

- 我们发现行驶距离的最大值和最小值都比较符合常理，最远100KM不到，最小的是0，即用户取消了行程。
- 我们发现有些情况虽然总的行驶距离为0，但是trip_duration却不为0，这些也可能是奇异值，但是因为这些记录较多，可能是一些我们不太清楚的其他情况，所以我们暂时保留这些信息。

In [16]:

```
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
sns.despine(left=True)
sns.distplot(np.log(train_df['total_distance'].dropna().values+1), axlabel = 'Log(total_distanc
e)', label = 'log(total_distance)', bins = 30, color="r")
plt.setp(axes, yticks=[])
plt.tight_layout()
```




In [17]:

```
train_df['total_distance'].dropna().describe()
```

Out[17]:

```
count    1.458643e+06
mean     4.626383e+03
std      5.303878e+03
min      0.000000e+00
25%      1.666100e+03
50%      2.755100e+03
75%      5.113800e+03
max      9.420420e+04
Name: total_distance, dtype: float64
```

In [18]:

```
train_df[(train_df['total_distance']== 0) & (train_df['trip_duration']>0)].shape
```

Out[18]:

(6009, 14)

In [19]:

```
6009 / train_df.shape[0]
```

Out[19]:

0.0041195795547097165

**pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude(用户上下车的经纬度)**

- 从整体来看，经纬度不存在明显的错误，例如经度大于180，维度大于180等等这种情况在该数据集中不存在。
- 经纬度存在明显的中心，与我们的数据较为符合，因为我们的数据来自于某一个城市的出租车，所以总体看来暂时未发现奇异情况。
- 从散点图中，我们发现有一些地方上下出租车的情况最多，大概在经度-75，维度41左右。可能是繁华地带。

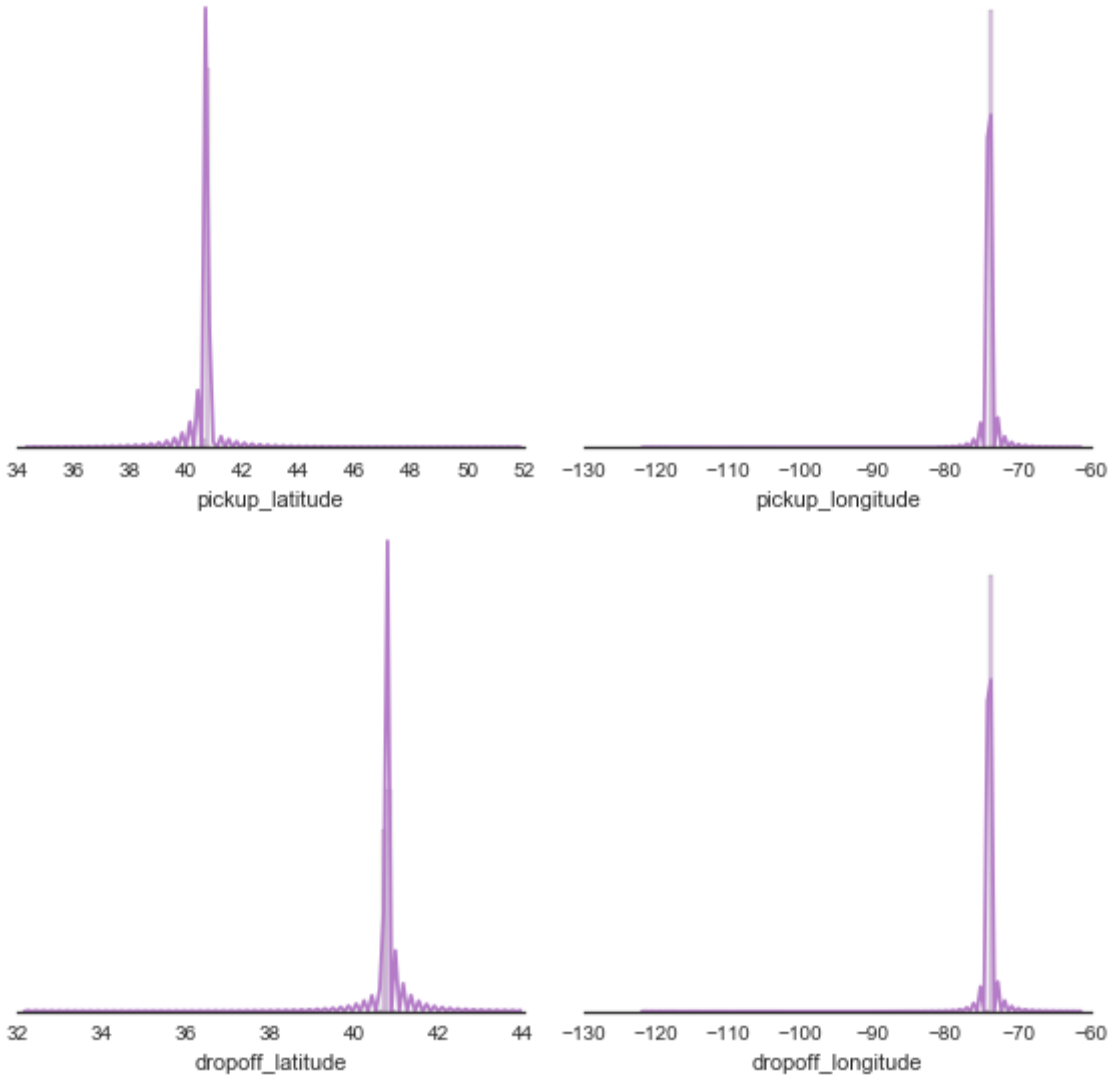
In [20]:

```
print(train_df[(train_df['pickup_latitude']<-180)].shape[0],train_df[(train_df['pickup_longitud
e']<-180)].shape[0],
      train_df[(train_df['dropoff_latitude']<-180)].shape[0],train_df[(train_df['dropoff_longitu
de']<-180)].shape[0])


print(train_df[(train_df['pickup_latitude']>180)].shape[0],train_df[(train_df['pickup_longitude']
180)].shape[0],
      train_df[(train_df['dropoff_latitude']>180)].shape[0],train_df[(train_df['dropoff_longitud
e']>180)].shape[0])
```

0 0 0 0
0 0 0 0

In [21]:

```
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(2,2,figsize=(8, 8), sharex=False, sharey = False)#
sns.despine(left=True)
sns.distplot(train_df['pickup_latitude'].values,axlabel ='pickup_latitude',label = 'pickup_latit
ude',color="m",bins = 100, ax=axes[0,0])
sns.distplot(train_df['pickup_longitude'].values,axlabel ='pickup_longitude', label = 'pickup_lo
ngitude',color="m",bins =100, ax=axes[0,1])
sns.distplot(train_df['dropoff_latitude'].values, axlabel ='dropoff_latitude',label = 'dropoff_l
atitude',color="m",bins =100, ax=axes[1, 0])
sns.distplot(train_df['dropoff_longitude'].values,axlabel ='dropoff_longitude', label = 'dropoff
_longitude',color="m",bins =100, ax=axes[1, 1])
plt.setp(axes, yticks=[])
plt.tight_layout()
```

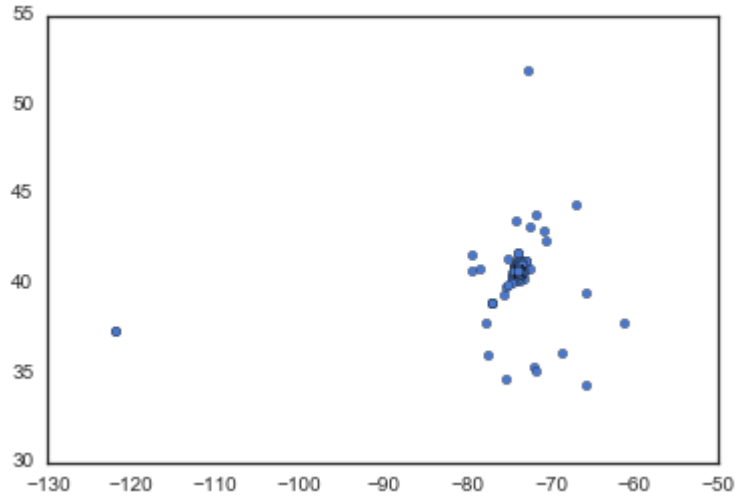


In [22]:

```
plt.scatter(x = train_df['pickup_longitude'], y =train_df['pickup_latitude'] )
```

Out[22]:

<matplotlib.collections.PathCollection at 0x6d055de518>




** total_travel_time(车行驶的时间，不包括等红绿灯等) **

- 车行驶的时间没有出现非常奇异的情况，最大值也就是2个小时不到。主要乘客坐车的时间都是在30min到60min中之间，较为符合我们的常识。

In [23]:

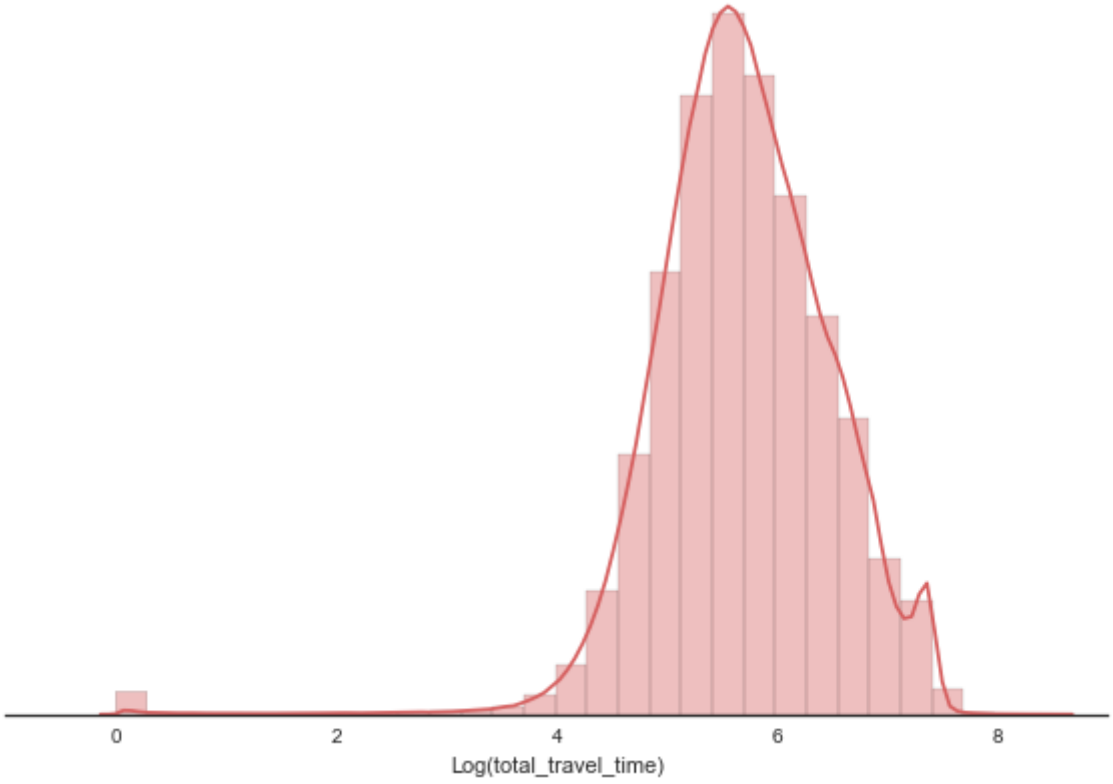
```
train_df['total_travel_time'].dropna().describe()
```

Out[23]:

```
count    1.458643e+06
mean     3.893719e+02
std      3.144679e+02
min      0.000000e+00
25%      1.815000e+02
50%      2.895000e+02
75%      4.907000e+02
max      5.135400e+03
Name: total_travel_time, dtype: float64
```

In [24]:

```
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
sns.despine(left=True)
sns.distplot(np.log(train_df['total_travel_time'].dropna().values+1), axlabel = 'Log(total_trave
l_time)', label = 'log(total_travel_time)', bins = 30, color="r")
plt.setp(axes, yticks=[])
plt.tight_layout()
```




**number_of_steps(街道拐弯数)**

- 我们发现总的街道拐弯的数量一般都是5-10个之间，最多的是46个，最少的是2个，较为符合我们的常识，所以暂时不做任何处理。

In [25]:

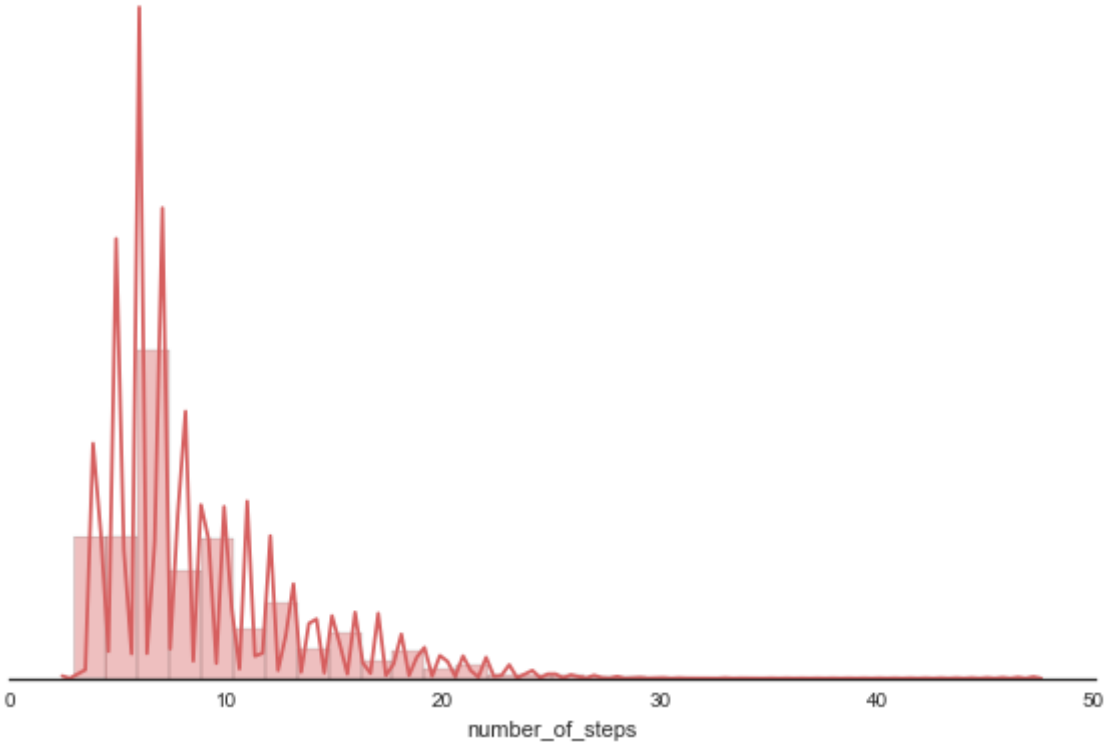
```
train_df['number_of_steps'].dropna().describe()
```

Out[25]:

```
count    1.458643e+06
mean     7.547126e+00
std      4.432504e+00
min      2.000000e+00
25%      5.000000e+00
50%      6.000000e+00
75%      9.000000e+00
max      4.600000e+01
Name: number_of_steps, dtype: float64
```

In [26]:

```
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
sns.despine(left=True)
sns.distplot((train_df['number_of_steps'].dropna().values+1), axlabel = 'number_of_steps', label
 = 'total_travel_time', bins = 30, color="r")
plt.setp(axes, yticks=[])
plt.tight_layout()
```




对分析得到的奇异值的情况进行集中删除和纠正处理。通过上面的分析，我们主要做如下操作。

- 删除乘客数为0的记录。

In [27]:

```
train_df = train_df[train_df['passenger_count'] > 0]
```

### 4.3.2 深度数据分析+特征工程

因为我们的核心目标是为了预测从A地乘车到B地所需的时间，而对我们的目标有影响的大概有以下几个重要的方面。

- 行车记录与**provider**的类型**(vendor_id)**，不同的类型旅程的时间不一，可能这边是大型车或者小型车(个人理解)，不管怎么，先看一下有没有很好的性质。
- 上车的时间，在高峰期和非高峰期上车，同样从A到B，时间差也会大不相同。在节假日和非节假日，在周末或者周中都会有很大的区别。
- 上车的人数，当乘车人数较多的时候，一般会是聚会或者其他活动，所以我们猜测会有一些影响，同时因为人数的不同，我们认为其与vendor_id也会有一些交集，所以我们将二者放在一起考虑。
- 上车的地点，繁华的地段和不太热闹的地段拥堵情况不太一样，会较大的影响行车的时间。
- 起点与终点的距离，距离越远，肯定相对行驶的时间就会越长，所以这个对于我们的目标也会有较大的影响。

#### 4.3.2.1 行车记录与**provider**的类型**(vendor_id)**

- 我们发现一共有2种类型，而且行车的时间还是有很明显的差别，vendor_id为2的特征比vendor_id为1的普遍旅程时间要多了200多。
- 结论：<font color = red> **vendor_id**是一个需要保留的较好的特征。 </font>

In [28]:

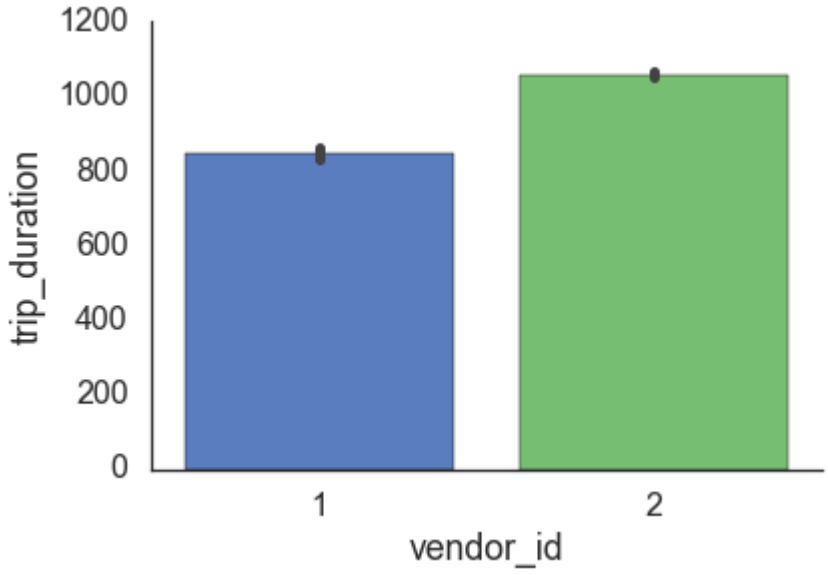
```
train_df['vendor_id'].unique()
```

Out[28]:

```
array([2, 1], dtype=int64)
```

In [29]:

```
sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.barplot(data=train_df, x="vendor_id", y="trip_duration")
sns.despine(bottom = False)
```



#### 4.3.2.2 上车的时间

- 从周中周末的分析来看，0表示周日，我们发现这个特征很具有代表性，周六周日大家的行程都比较短，大概是周六周日大家都会在住所附近走走，好好休息，不想远出。周一到周五都普遍比周末行程要远，应该是上班的地方离家远的缘故。
- 从小时来看，我们发现7:00-9:00之间有个明显的上涨，可能是上班的缘故，而在15:30左右达到高峰，这个很奇怪，可能是因为美国的通勤、作息时间等与中国不太一样，如果是国内肯定会是17点左右。
- 从周中周末以及各个时间段放在一起分析来看，不出意外，周末周日两条线明显在下方，而且每天的时间段的情况都类似，符合我们的预期。
- 从月份来分析，我们发现1月2月明显要低，因为这个时候刚好是假期，但是4、5、6就要高很多，这个时候主要还是工作期间。
- 从每年的某一天来看的话，因为波动比较大，没有很明显的特征，如果要细看的话可能需要将节假日的信息加入。
- 从每年的每天的大的载客趋势图来看的话，我们发现存在周期性的变化。可能与周末之类的有关，同时可能是节假日的原因，所以会出现局部较大的gap。而且与我们乘客乘车的时间有很大的相似性，因为接单多了，路可能会有更大几率出现拥堵，相同两地的乘车时间就会变大。
- 结论：<font color = red> 我们发现，时间特征的影响是非常明显的，对我们的预测能带来非常大的帮助。因此保留时间特征同时进行编码。

In [30]:

```
train_df['pickup_datetime'] = pd.to_datetime(train_df.pickup_datetime)
train_df.loc[:, 'pick_month'] = train_df['pickup_datetime'].dt.month
train_df.loc[:, 'hour'] = train_df['pickup_datetime'].dt.hour
train_df.loc[:, 'week_of_year'] = train_df['pickup_datetime'].dt.weekofyear
train_df.loc[:, 'day_of_year'] = train_df['pickup_datetime'].dt.dayofyear
train_df.loc[:, 'day_of_week'] = train_df['pickup_datetime'].dt.dayofweek
train_df.loc[:, 'pickup_date'] = train_df['pickup_datetime'].dt.date
```

In [31]:

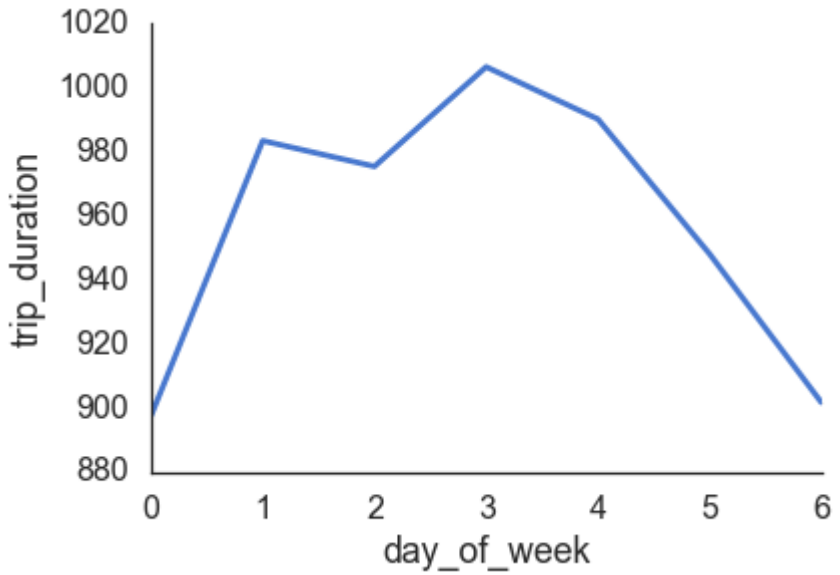
```
train_df.shape
```

Out[31]:

(1458584, 20)

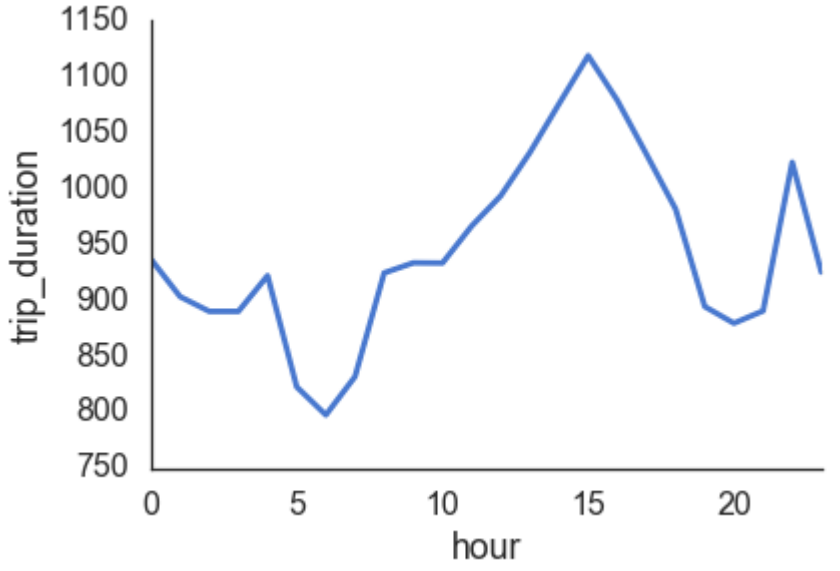
In [32]:

```
summary_wdays_avg_duration = pd.DataFrame(train_df.groupby(['day_of_week'])['trip_duration'].mean())
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1
sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="day_of_week", unit = "unit",  value="trip_duration")
sns.despine(bottom = False)
end = time.time()
```



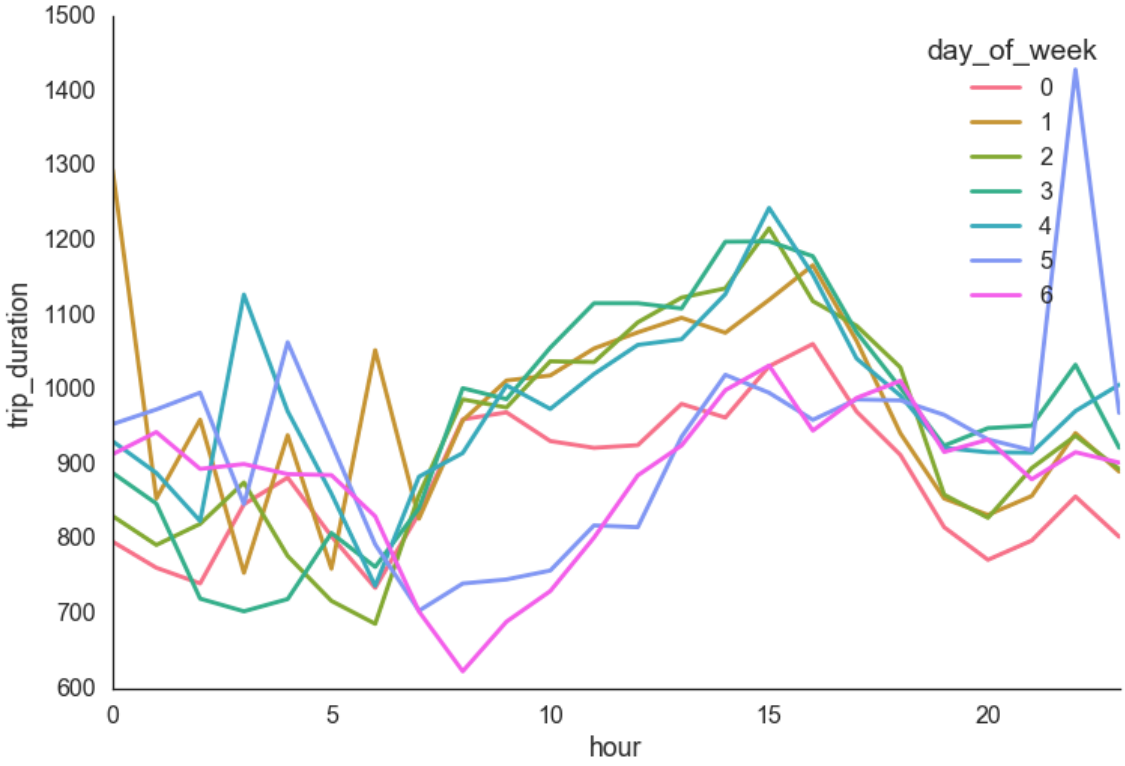

In [33]:

```
summary_wdays_avg_duration = pd.DataFrame(train_df.groupby(['hour'])['trip_duration'].mean())
summary_wdays_avg_duration.reset_index(inplace = True)

summary_wdays_avg_duration['unit']=1
sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="hour", unit = "unit",  value="trip_duration")
sns.despine(bottom = False)
end = time.time()
```



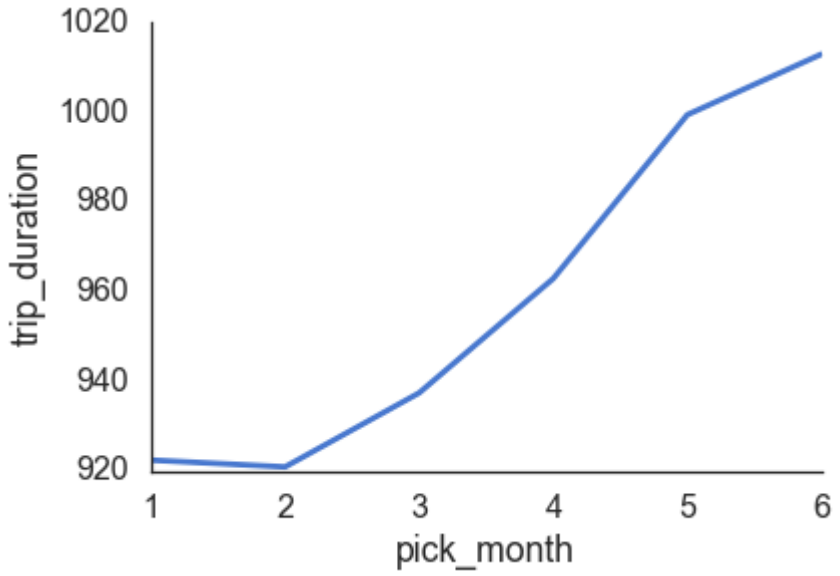
In [34]:

```
plt.figure(figsize=[12,8])
summary_hour_duration = pd.DataFrame(train_df.groupby(['day_of_week','hour'])['trip_duration'].mean())
summary_hour_duration.reset_index(inplace = True)
summary_hour_duration['unit']=1
sns.set(style="white", palette="muted", color_codes=False)
sns.set_context("poster")
sns.tsplot(data=summary_hour_duration, time="hour", unit = "unit", condition="day_of_week", value="trip_duration")
sns.despine(bottom = False)
```



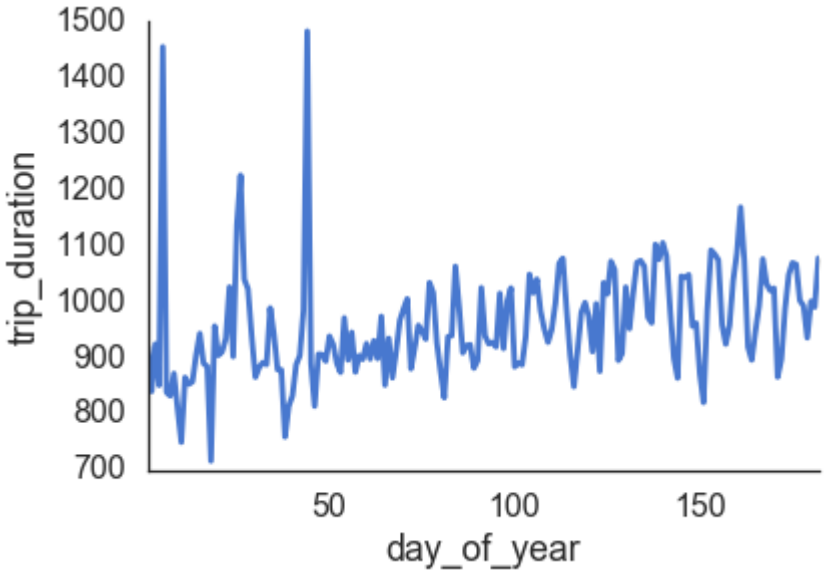
In [35]:

```
summary_wdays_avg_duration = pd.DataFrame(train_df.groupby(['pick_month'])
['trip_duration'].mean())
summary_wdays_avg_duration.reset_index(inplace = True)

summary_wdays_avg_duration['unit']=1
sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="pick_month", unit = "unit",  value="trip_durat
ion")
sns.despine(bottom = False)
end = time.time()
```



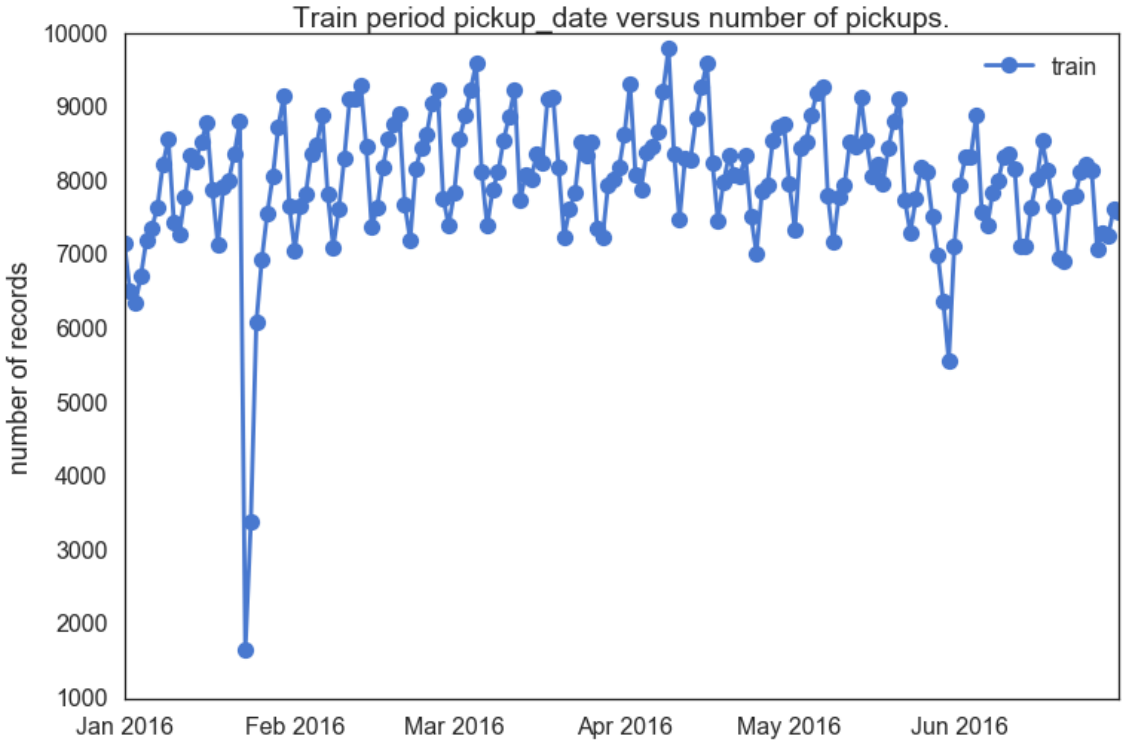
In [36]:

```
summary_wdays_avg_duration = pd.DataFrame(train_df.groupby(['day_of_year'])['trip_duration'].mean())
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1
sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.tsplot(data=summary_wdays_avg_duration, time="day_of_year", unit = "unit",  value="trip_duration")
sns.despine(bottom = False)
end = time.time()
```



In [37]:

```
plt.figure(figsize=[12,8])
plt.plot(train_df.groupby('pickup_date').count()[['id']], 'o-', label='train')
plt.title('Train period pickup_date versus number of pickups.')
plt.legend(loc=0)
plt.ylabel('number of records')
plt.show()
```

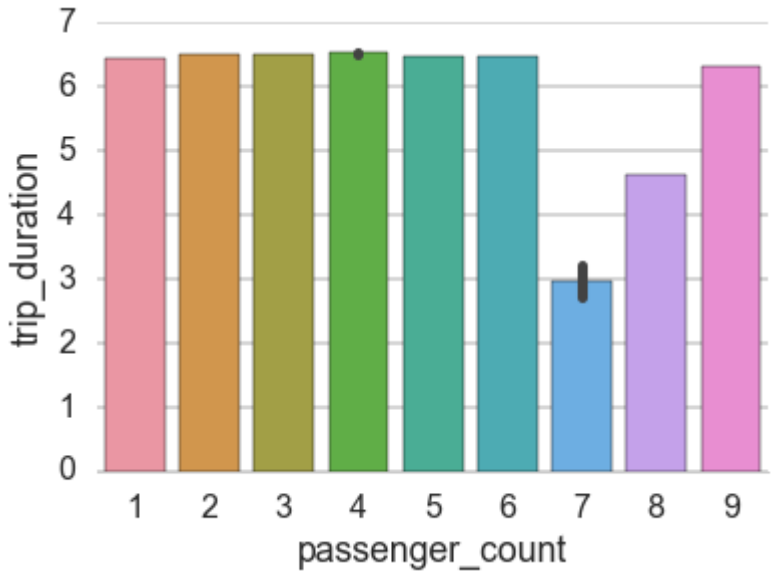



#### 4.3.2.3 上车的人数

- 仅从第一张图来看，我们发现上车的人数和我们要预测的行程的时间关系影响并不是很大。虽然7、8、9时在图上有较大的偏差，但是因为样本个数只有1、2、3个，所以代表性不强。
- 进一步挖掘，我们将上车的人数与vendor_id结合在一起看，我们发现当上车人数小于等于6个的时候，vendor_id为1或者2的影响差不多，但是当其大于6的时候基本都是vendor_id=2的情况。
- 结论：<font color = red> 我们发现，乘客的数量对于旅程的时间影响不大，但为了保险起见，我们暂时先保留该特征。

In [38]:

```
sns.set(style="whitegrid", palette="pastel", color_codes=True)
sns.set_context("poster")
train_df['trip_duration']= np.log(train_df['trip_duration'])
sns.barplot(x="passenger_count", y="trip_duration", data=train_df)
sns.despine(left=True)
```




In [39]:

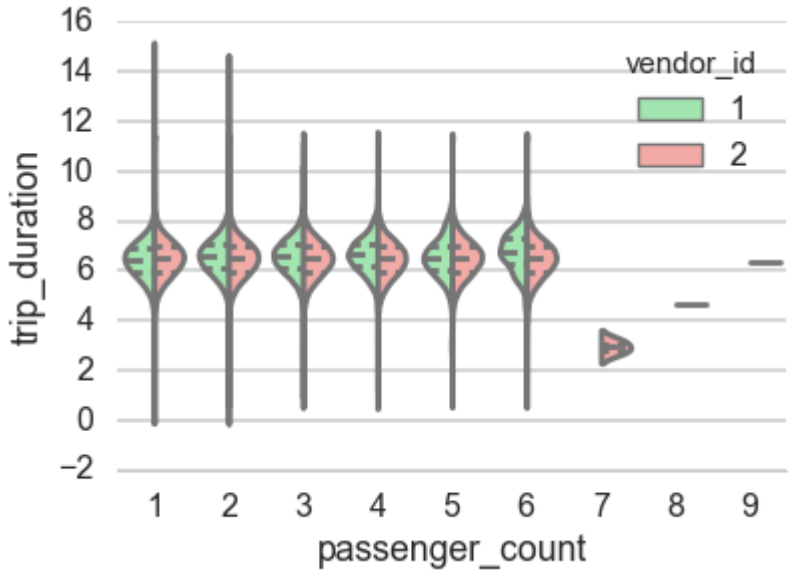
```
train_df.groupby(['passenger_count'])['trip_duration'].count()
```

Out[39]:

```
passenger_count
1    1033540
2     210318
3      59896
4      28404
5      78088
6      48333
7          3
8          1
9          1
Name: trip_duration, dtype: int64
```

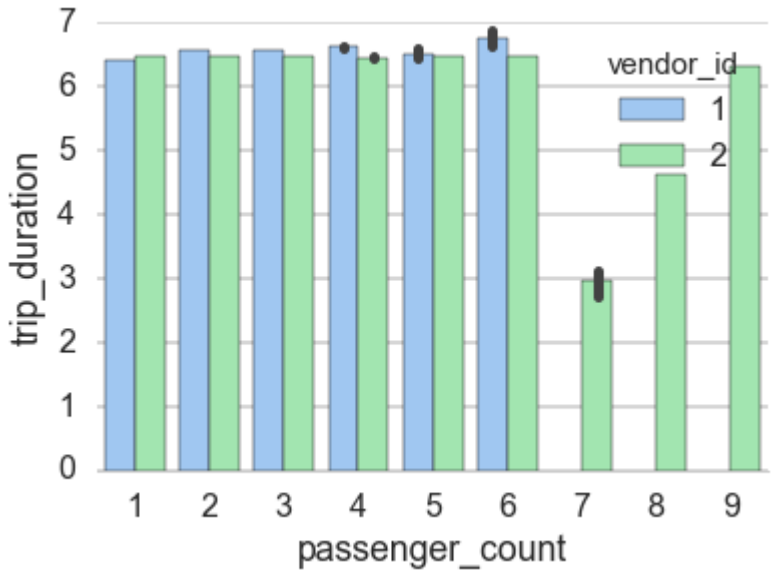
In [40]:

```
sns.set(style="whitegrid", palette="pastel", color_codes=True)
sns.set_context("poster")
train_data = train_df.copy()
train_data['trip_duration']= train_df['trip_duration']
sns.violinplot(x="passenger_count", y="trip_duration", hue="vendor_id",
data=train_data.dropna(), split=True,
               inner="quart",palette={1: "g", 2: "r"})
sns.despine(left=True)
```




In [41]:

```
sns.set(style="whitegrid", palette="pastel", color_codes=True)
sns.set_context("poster")
train_data['trip_duration']= train_df['trip_duration']
sns.barplot(x="passenger_count", y="trip_duration", hue="vendor_id", data=train_df)
sns.despine(left=True)
```



#### 4.3.2.4 上车的地点

繁华的地段和不太热闹的地段拥堵情况不太一样，这个因素会较大的影响行车的时间。

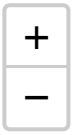
- 上车地点是非常重要的因素，我们考虑对上车地点进行聚类。从聚类的结果我们发现，不同的地点上车之后的行程的时间差异比较大。
- 此处我们利用一些有趣的可视化的方法进行阐述述，方便大家理解。
- 结论：<font color = red> 上车的地点对于我们的行程时间预测能带来较大的帮助，所以我们考虑对其进行聚类。

这边我们可以方便的直接调用谷歌的地图的api 。

In [42]:

```
import folium # goelogical map
map_1 = folium.Map(location=[40.767937,-73.982155 ],tiles='OpenStreetMap',
 zoom_start=12)
#tile: 'OpenStreetMap','Stamen Terrain','Mapbox Bright','Mapbox Control room'
for each in train_df[:200].iterrows():
    folium.CircleMarker([each[1]['pickup_latitude'],each[1]['pickup_longitude']],
                        radius=3,
                        color='red',
                        popup=str(each[1]['pickup_latitude'])+','+str(each[1]
['pickup_longitude']),
                        fill_color='#FD8A6C'
                        ).add_to(map_1)
map_1
```

Out[42]:





In [43]:

```
rgb = np.zeros((4000, 4000, 3), dtype=np.uint8)
rgb[..., 0] = 0
rgb[..., 1] = 0
rgb[..., 2] = 0
train_df['pick_lat_new'] = list(map(int, (train_df['pickup_latitude'] - (40.6000))*10000))
train_df['drop_lat_new'] = list(map(int, (train_df['dropoff_latitude'] - (40.6000))*10000))
train_df['pick_lon_new'] = list(map(int, (train_df['pickup_longitude'] - (-74.050))*10000))
train_df['drop_lon_new'] = list(map(int,(train_df['dropoff_longitude'] - (-74.050))*10000))

summary_plot = pd.DataFrame(train_df.groupby(['pick_lat_new', 'pick_lon_new'])['id'].count())

summary_plot.reset_index(inplace = True)
summary_plot.head(120)
lat_list = summary_plot['pick_lat_new'].unique()
for i in lat_list:
    if i < 0:
        continue
    if i > 4000:
        continue
    #print(i)
    lon_list = summary_plot.loc[summary_plot['pick_lat_new']==i]['pick_lon_new'].tolist()
    unit = summary_plot.loc[summary_plot['pick_lat_new']==i]['id'].tolist()
    for j in lon_list:
        if j < 0:
            continue
        if j > 4000:
            continue
        #j = int(j)
        a = unit[lon_list.index(j)]
        #print(a)
        if (a//50) >0:
            rgb[i][j][0] = 255
            rgb[i,j, 1] = 0
            rgb[i,j, 2] = 255
        elif (a//10)>0:
            rgb[i,j, 0] = 0
            rgb[i,j, 1] = 255
            rgb[i,j, 2] = 0
        else:
            rgb[i,j, 0] = 255
            rgb[i,j, 1] = 0
            rgb[i,j, 2] = 0
fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(14,20))
ax.imshow(rgb, cmap = 'hot')
ax.set_axis_off()
```

- 上图中红色的区域表示在该区域上车的有1-10个出租车订单
- 绿色的区域表示在该区域上车的有10 - 50个出租车订单
- 黄色的区域表示在该区域上车的有50+个出租车订单

In [44]:

```
def assign_cluster(df, k):
    """function to assign clusters """
    df_pick = df[['pickup_longitude','pickup_latitude']]
    df_drop = df[['dropoff_longitude','dropoff_latitude']]
    init = np.array([[ -73.98737616,   40.72981533],
       [-121.93328857,   37.38933945],
       [ -73.78423222,   40.64711269],
       [ -73.9546417 ,   40.77377538],
       [ -66.84140269,   36.64537175],
       [ -73.87040541,   40.77016484],
       [ -73.97316185,   40.75814346],
       [ -73.98861094,   40.7527791 ],
       [ -72.80966949,   51.88108444],
       [ -76.99779701,   38.47370625],
       [ -73.96975298,   40.69089596],
       [ -74.00816622,   40.71414939],
       [ -66.97216034,   44.37194443],
       [ -61.33552933,   37.85105133],
       [ -73.98001393,   40.7783577 ],
       [ -72.00626526,   43.20296402],
       [ -73.07618713,   35.03469086],
       [ -73.95759366,   40.80316361],
       [ -79.20167796,   41.04752096],
       [ -74.00106031,   40.73867723]])
    k_means_pick = KMeans(n_clusters=k, init=init, n_init=1)
    k_means_pick.fit(df_pick)
    clust_pick = k_means_pick.labels_
    df['label_pick'] = clust_pick.tolist()
    df['label_drop'] = k_means_pick.predict(df_drop)
    return df, k_means_pick
```

In [45]:

```
train_cl, k_means = assign_cluster(train_df, 20)  # make it 100 when extracting features
centroid_pickups = pd.DataFrame(k_means.cluster_centers_, columns = ['centroid_pick_long', 'cent
roid_pick_lat'])
centroid_dropoff = pd.DataFrame(k_means.cluster_centers_, columns = ['centroid_drop_long', 'cent
roid_drop_lat'])
centroid_pickups['label_pick'] = centroid_pickups.index
centroid_dropoff['label_drop'] = centroid_dropoff.index

train_cl = pd.merge(train_cl, centroid_pickups, how='left', on=['label_pick'])
train_cl = pd.merge(train_cl, centroid_dropoff, how='left', on=['label_drop'])
```

In [46]:

```
train_cl.head()
```

Out[46]:

| | id | vendor_id | pickup_datetime | dropoff_datetime | passenger_count | p |
|---|---|---|---|---|---|---|
| **0** | id2875421 | 2 | 2016-03-14 17:24:55 | 2016-03-14 17:32:30 | 1 | - |
| **1** | id2377394 | 1 | 2016-06-12 00:43:35 | 2016-06-12 00:54:38 | 1 | - |
| **2** | id3858529 | 2 | 2016-01-19 11:35:24 | 2016-01-19 12:10:48 | 1 | - |
| **3** | id3504673 | 2 | 2016-04-06 19:32:31 | 2016-04-06 19:39:40 | 1 | - |
| **4** | id2181028 | 2 | 2016-03-26 13:30:55 | 2016-03-26 13:38:10 | 1 | - |

5 rows × 30 columns

In [47]:

```
train_cl.groupby(['centroid_drop_long','centroid_drop_lat'])['trip_duration'].mean()
```

Out[47]:

```
centroid_drop_long  centroid_drop_lat
-121.933289         37.389339           6.610103
-79.201678          41.047521           6.807314
-76.997797          38.473706           5.726615
-74.008229          40.714102           6.662452
-74.001070          40.738659           6.309370
-73.988619          40.752707           6.445928
-73.987417          40.729722           6.388395
-73.980021          40.778359           6.294476
-73.973191          40.758148           6.427936
-73.969750          40.690916           7.033099
-73.957584          40.803168           6.513568
-73.954646          40.773771           6.281341
-73.870405          40.770167           7.153110
-73.784231          40.647110           7.297653
-73.076187          35.034691           6.082286
-72.006265          43.202964           6.472194
-66.972160          44.371944           7.030857
-66.841403          36.645372           6.205758
-61.335529          37.851051           5.910797
Name: trip_duration, dtype: float64
```

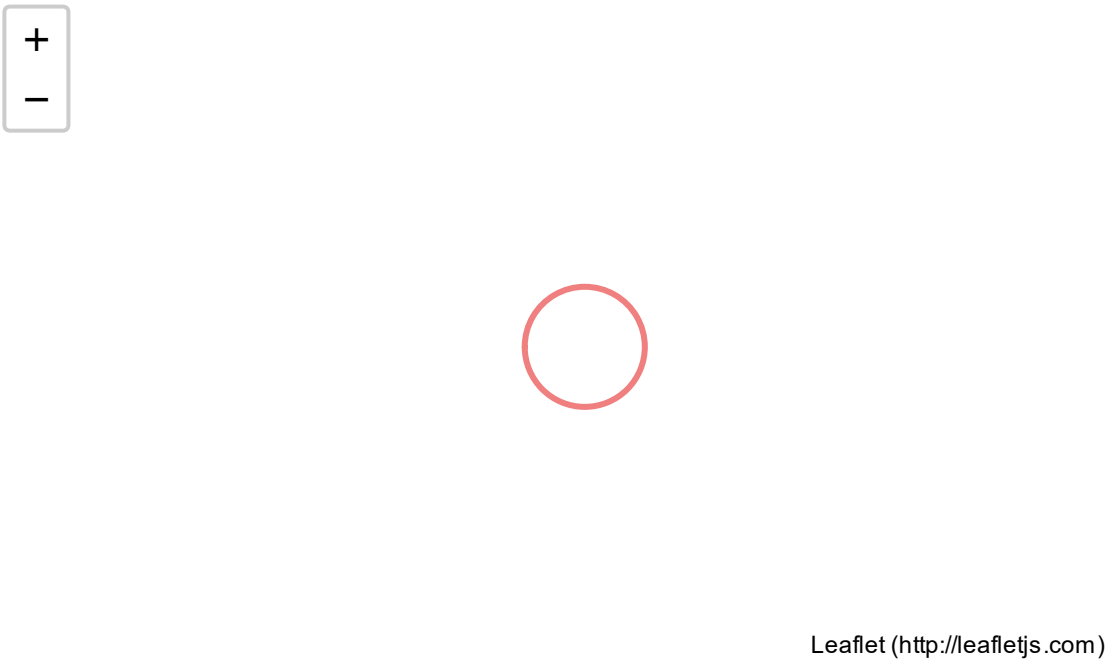
In [48]:

```
def cluster_summary(sum_df):
    """function to calculate summary of given list of clusters """
    #agg_func = {'trip_duration':'mean','label_drop':'count','bearing':'mean','id':'count'} # th
at's how you use agg function with groupby
    summary_avg_time = pd.DataFrame(sum_df.groupby('label_pick')['trip_duration'].mean())
    summary_avg_time.reset_index(inplace = True)
    summary_pref_clus = pd.DataFrame(sum_df.groupby(['label_pick', 'label_drop'])['id'].count())
    summary_pref_clus = summary_pref_clus.reset_index()
    summary_pref_clus = summary_pref_clus.loc[summary_pref_clus.groupby('label_pick')['id'].idxm
ax()]
    summary =pd.merge(summary_avg_time, summary_pref_clus, how = 'left', on = 'label_pick')
    summary = summary.rename(columns={'trip_duration':'avg_triptime'})
    return summary

def clusters_map(clus_data, full_data, tile = 'OpenStreetMap', sig = 0, zoom = 12, circle = 0, r
adius_ = 30):
    """ function to plot clusters on map"""
    map_1 = folium.Map(location=[40.767937, -73.982155], zoom_start=zoom,tiles= tile) # 'Mapbox'
 'Stamen Toner'
    summary_full_data = pd.DataFrame(full_data.groupby('label_pick')['id'].count())
    summary_full_data.reset_index(inplace = True)
    if sig == 1:
        summary_full_data = summary_full_data.loc[summary_full_data['id']>70000]
    sig_cluster = summary_full_data['label_pick'].tolist()
    clus_summary = cluster_summary(full_data)
    for i in sig_cluster:
        pick_long = clus_data.loc[clus_data.index ==i]['centroid_pick_long'].values[0]
        pick_lat = clus_data.loc[clus_data.index ==i]['centroid_pick_lat'].values[0]
        clus_no = clus_data.loc[clus_data.index ==i]['label_pick'].values[0]
        most_visited_clus = clus_summary.loc[clus_summary['label_pick']==i]
['label_drop'].values[0]
        avg_triptime = clus_summary.loc[clus_summary['label_pick']==i]['avg_triptime'].values[0]
        pop = 'cluster = '+str(clus_no)+' & most visited cluster = ' +str(most_visited_clus) +'
 & avg triptime from this cluster =' + str(avg_triptime)
        if circle == 1:
            folium.CircleMarker(location=[pick_lat, pick_long], radius=radius_,
                    color='#F08080',
                    fill_color='#3186cc', popup=pop).add_to(map_1)
        folium.Marker([pick_lat, pick_long], popup=pop).add_to(map_1)
    return map_1
```

In [49]:

```
clus_map = clusters_map(centroid_pickups, train_cl, sig =0, zoom =3.2, circle =1, tile = 'Stamen
 Terrain')
clus_map
```

Out[49]:



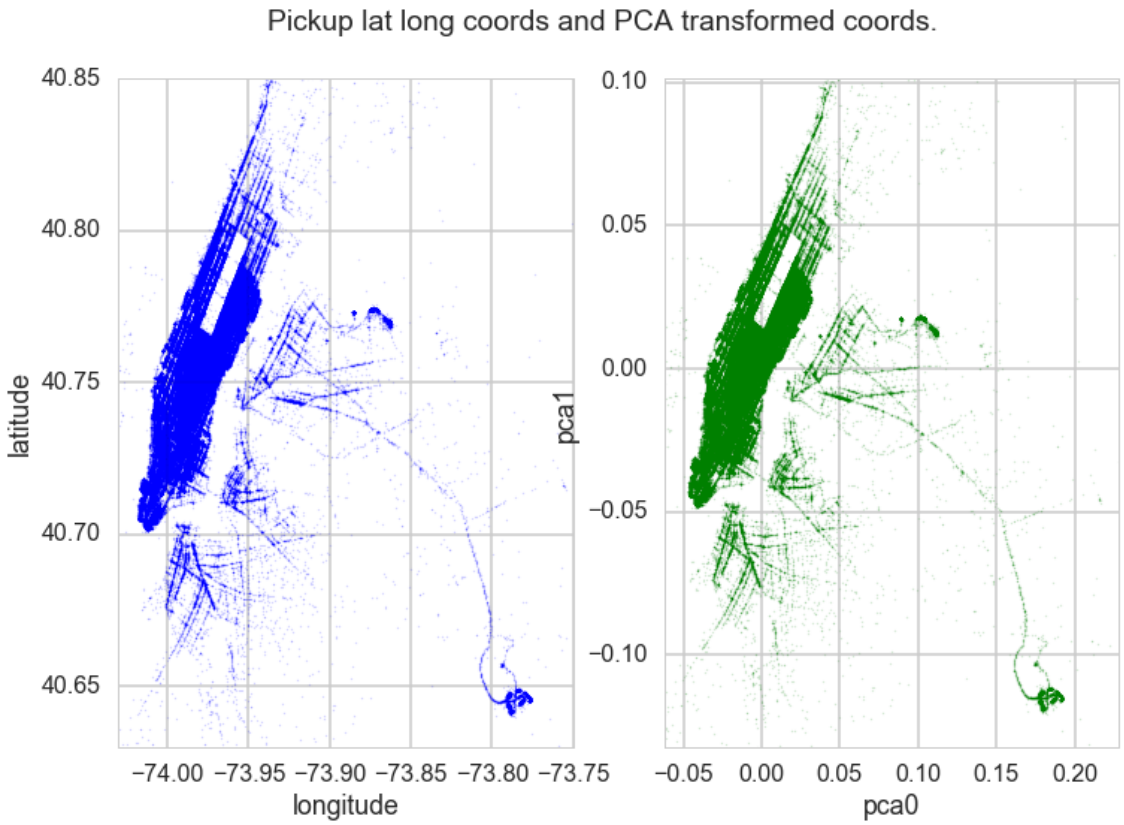

- PCA特征的引入主要来源于kaggle上优秀选手的分享，<font color = red>进行PCA变换的特征会使得在使用树模型的时候能更好的方便我们的树模型进行split</font>，实验中我们也发现加入PCA的特征确实能帮助模型带来增益。

In [50]:

```
from sklearn.decomposition import PCA
coords = np.vstack((train_df[['pickup_latitude', 'pickup_longitude']].values,
                    train_df[['dropoff_latitude', 'dropoff_longitude']].values))

pca = PCA().fit(coords)
train_df['pickup_pca0'] = pca.transform(train_df[['pickup_latitude', 'pickup_longitude']])[:, 0]
train_df['pickup_pca1'] = pca.transform(train_df[['pickup_latitude', 'pickup_longitude']])[:, 1]
train_df['dropoff_pca0'] = pca.transform(train_df[['dropoff_latitude', 'dropoff_longitude']])[:,
 0]
train_df['dropoff_pca1'] = pca.transform(train_df[['dropoff_latitude', 'dropoff_longitude']])[:,
 1]
```

In [51]:

```
fig, ax = plt.subplots(figsize = [12,8],ncols=2)
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)
ax[0].scatter(train_df['pickup_longitude'].values[:500000],
train_df['pickup_latitude'].values[:500000],
               color='blue', s=1, alpha=0.1)
ax[1].scatter(train_df['pickup_pca0'].values[:500000], train_df['pickup_pca1'].values[:500000],
               color='green', s=1, alpha=0.1)
fig.suptitle('Pickup lat long coords and PCA transformed coords.')

ax[0].set_ylabel('latitude')
ax[0].set_xlabel('longitude')
ax[1].set_xlabel('pca0')
ax[1].set_ylabel('pca1')
ax[0].set_xlim(city_long_border)
ax[0].set_ylim(city_lat_border)
pca_borders = pca.transform([[x, y] for x in city_lat_border for y in city_long_border])
ax[1].set_xlim(pca_borders[:, 0].min(), pca_borders[:, 0].max())
ax[1].set_ylim(pca_borders[:, 1].min(), pca_borders[:, 1].max())
plt.show()
```



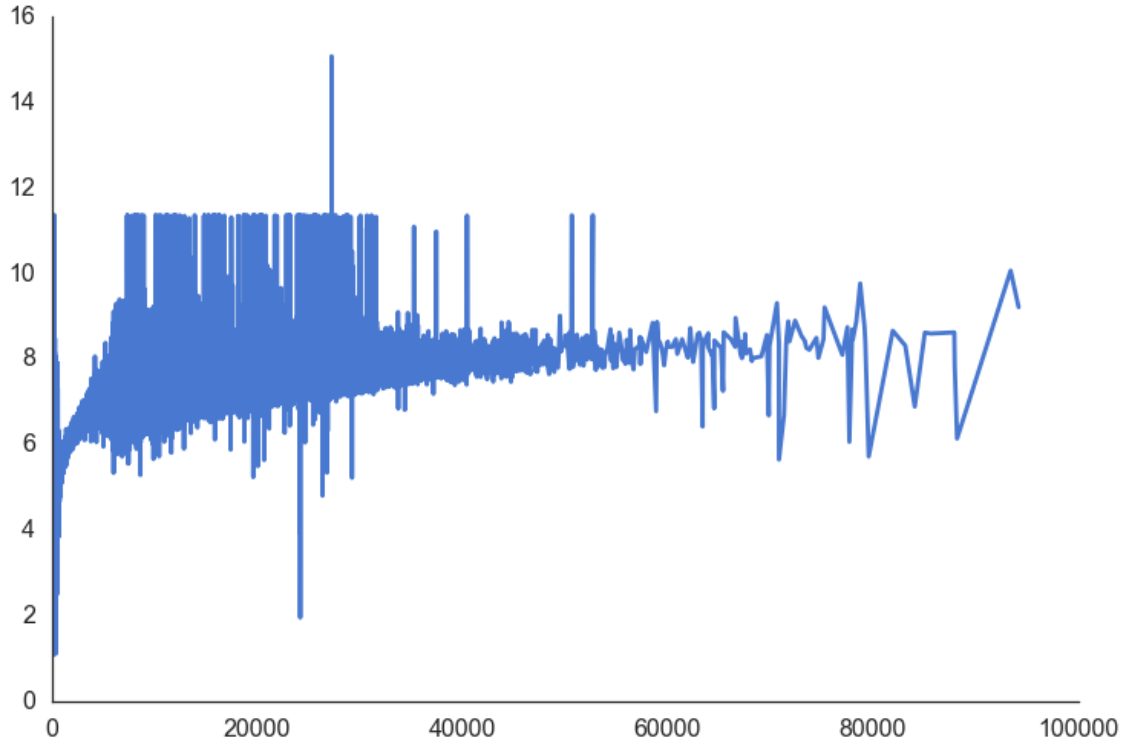
#### 4.3.2.5 起点与终点的距离

乘车的起点与终点距离较远的话乘车时间自然就会较长，反之亦然。

- 我们发现随着距离的增加,我们的乘车时间是线性增加的，但是存在很多的噪音。猜测这些噪音是由于堵车等情况造成。
- 我们用Wiki的算法通过经纬度重新计算了距离（欧几里得距离）,发现情况类似，但是存在一些差距，猜测是因为经纬度之间存在噪音导致。
- 我们又计算了街道距离，因为实际情况中，行车无法两地直向行驶，而街道距离是能更好的模拟真正的实际距离。
- 此外我们还计算了两地之间的角度，出发地到目的地的角度(这个是参考的 https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367 (https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367))
- 结论：<font color = red> 起点与终点的距离(欧几里得，街道距离)对于我们的行程预测能带来较大的帮助，我们选择将其加入我们的模型。

In [52]:

```
plt.figure(figsize=[12,8])
summary_total_distance_duration = pd.DataFrame(train_df.groupby(['total_distance'])['trip_durati
on'].mean())
summary_total_distance_duration.columns = ['trip_duration']
summary_total_distance_duration.reset_index(inplace = True)
summary_total_distance_duration = summary_total_distance_duration.sort_values('total_distance')
# summary_total_distance_duration.set_index('total_distance',inplace=True)
sns.set(style="white", palette="muted", color_codes=False)
sns.set_context("poster")
plt.plot(summary_total_distance_duration['total_distance'].values,summary_total_distance_duratio
n['trip_duration'].values)
        # sns.(data=summary_total_distance_duration, row="total_distance", col="trip_duration")
sns.despine(bottom = False)
```



In [54]:

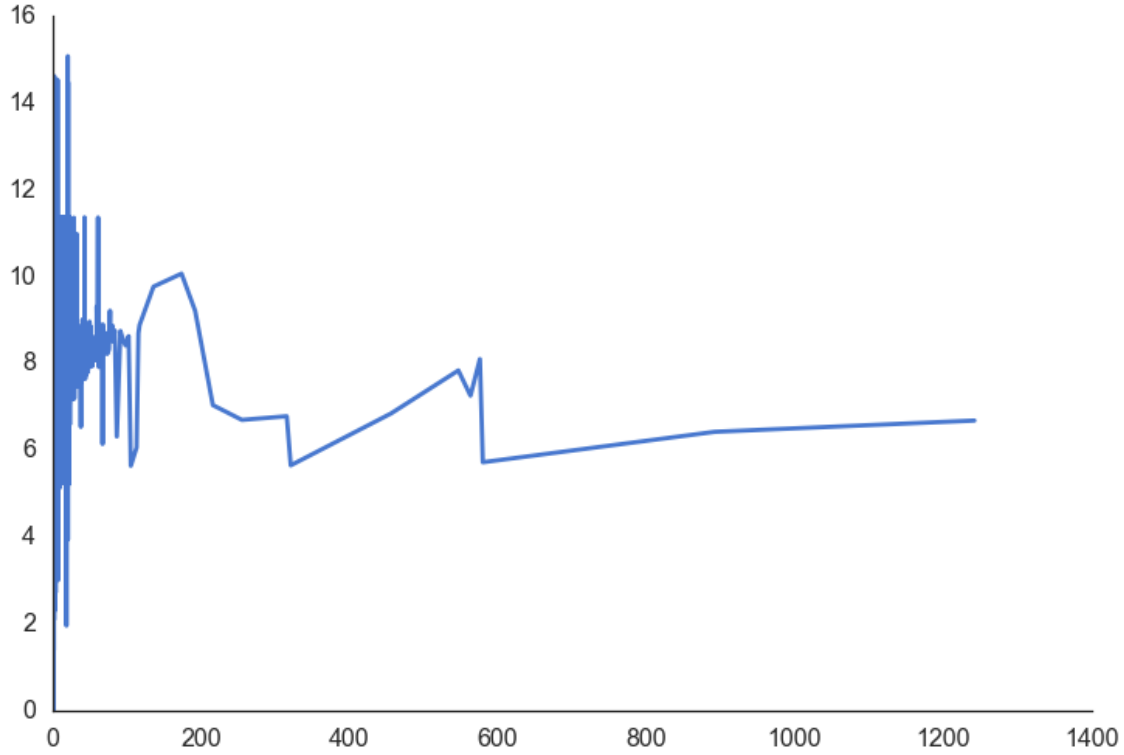
```
'''
    地球半径：6378.137KM ,https://en.wikipedia.org/wiki/Haversine_formula

'''
EARTH_RADIUS = 6378.137;

def getDistanceOfKM(x):
    splits = x.split(' ')
    radLng1 = np.float32(splits[0])
    radLat1 = np.float32(splits[1])
    radLng2 = np.float32(splits[2])
    radLat2 = np.float32(splits[3])

    radLat1 = radLat1 * math.pi / 180.0;
    radLat2 = radLat2 * math.pi / 180.0;
    a = radLat1 - radLat2;
    radLng1 = radLng1 * math.pi / 180.0
    radLng2 = radLng2 * math.pi / 180.0

    if radLat1 == radLat2 and radLng1 == radLng2:
        return 0

    d =
math.acos(math.sin(radLat1)*math.sin(radLat2)+math.cos(radLat1)*math.cos(radLat2)*math.cos(radLng2-radLng1))*EARTH_RADIUS;

    return d;
```

In [55]:

```
train_df['long_lat'] = train_df['pickup_longitude'].apply(str) + ' ' + train_df['pickup_latitude'].apply(str) + ' ' + train_df['dropoff_longitude'].apply(str) + ' ' + train_df['dropoff_latitude'].apply(str)
train_df['Distance'] = train_df['long_lat'].apply(getDistanceOfKM)
```

In [56]:

```
plt.figure(figsize=[12,8])
summary_distance_duration = pd.DataFrame(train_df.groupby(['Distance'])['trip_duration'].mean())
summary_distance_duration.columns = ['trip_duration']
summary_distance_duration.reset_index(inplace = True)
summary_distance_duration = summary_distance_duration.sort_values('Distance')
# summary_total_distance_duration.set_index('total_distance',inplace=True)
sns.set(style="white", palette="muted", color_codes=False)
sns.set_context("poster")
plt.plot(summary_distance_duration['Distance'].values,summary_distance_duration['trip_duration'].
ues)
        # sns.(data=summary_total_distance_duration, row="total_distance", col="trip_duration")
sns.despine(bottom = False)
```



In [57]:

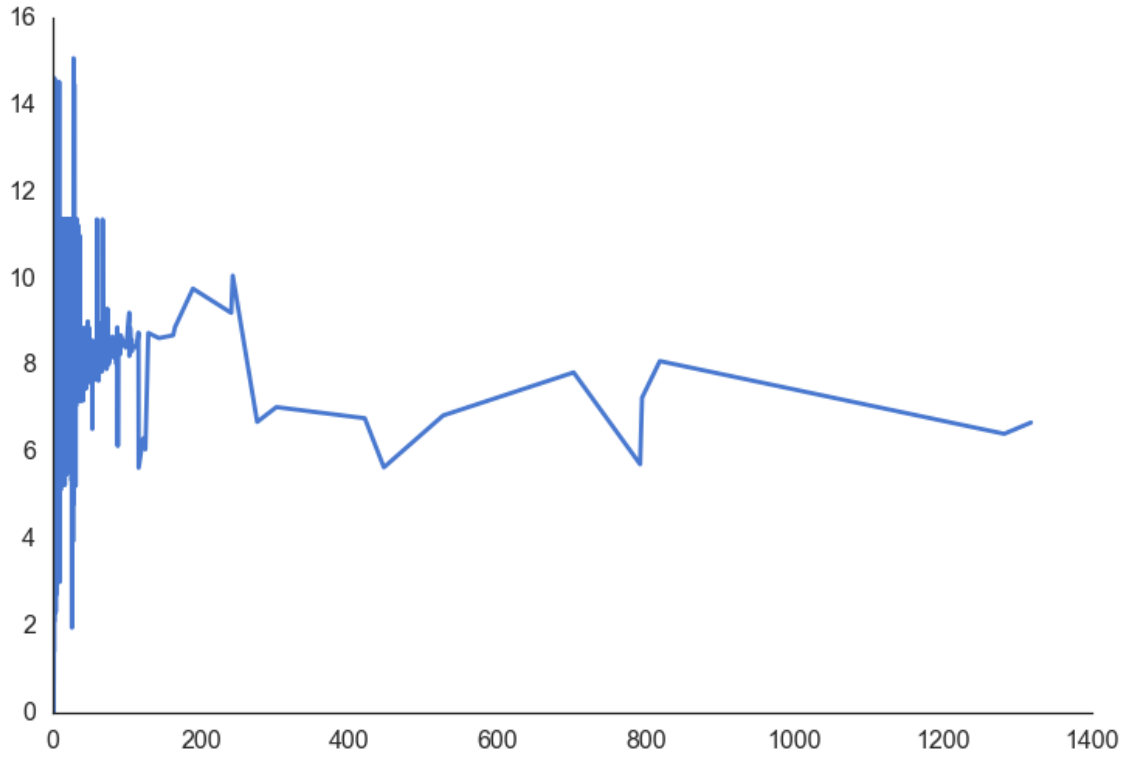
```
### 参考Kernel分享https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367
def haversine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

def dummy_manhattan_distance(lat1, lng1, lat2, lng2):
    a = haversine_array(lat1, lng1, lat1, lng2)
    b = haversine_array(lat1, lng1, lat2, lng1)
    return a + b

def bearing_array(lat1, lng1, lat2, lng2):
    AVG_EARTH_RADIUS = 6371  # in km
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))
```

In [58]:

```
train_df.loc[:, 'distance_dummy_manhattan'] = dummy_manhattan_distance(train_df['pickup_latitud
e'].values, train_df['pickup_longitude'].values, train_df['dropoff_latitude'].values,
train_df['dropoff_longitude'].values)
train_df.loc[:, 'direction'] = bearing_array(train_df['pickup_latitude'].values, train_df['picku
p_longitude'].values, train_df['dropoff_latitude'].values, train_df['dropoff_longitude'].values)
```

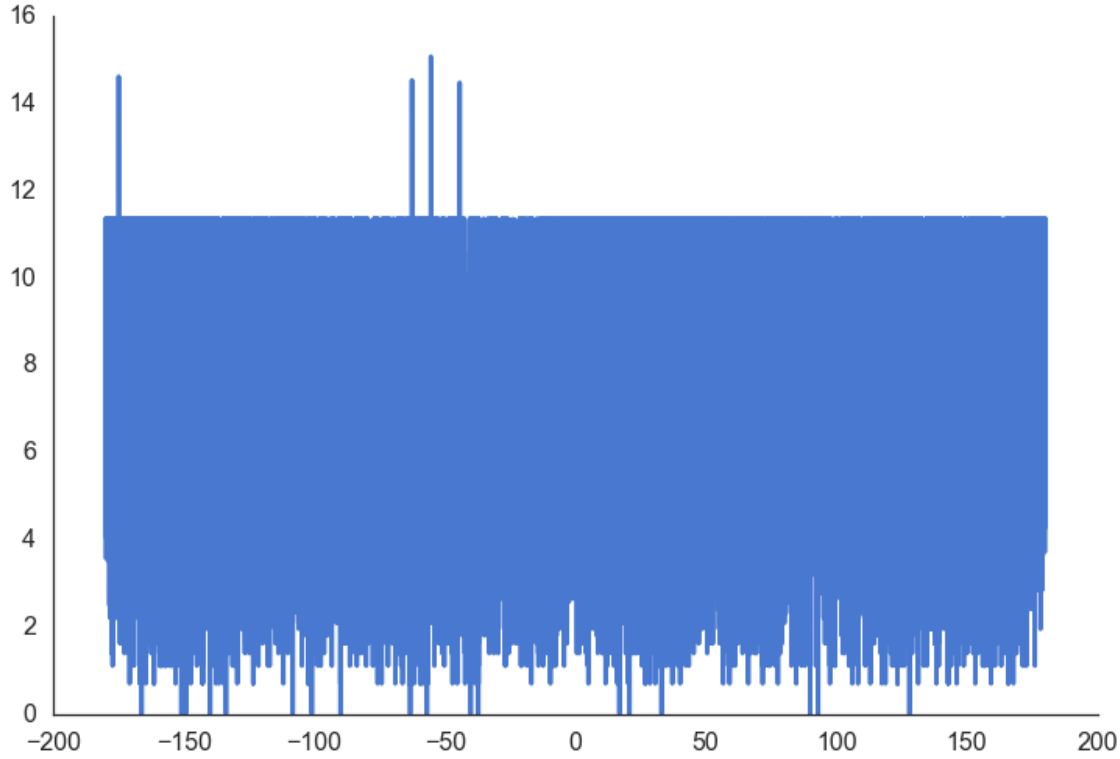
In [59]:

```
plt.figure(figsize=[12,8])
summary_distance_duration = pd.DataFrame(train_df.groupby(['distance_dummy_manhattan'])['trip_du
ration'].mean())
summary_distance_duration.columns = ['trip_duration']
summary_distance_duration.reset_index(inplace = True)
summary_distance_duration = summary_distance_duration.sort_values('distance_dummy_manhattan')
sns.set(style="white", palette="muted", color_codes=False)
sns.set_context("poster")
plt.plot(summary_distance_duration['distance_dummy_manhattan'].values,summary_distance_duration['
p_duration'].values)
sns.despine(bottom = False)
```



In [60]:

```
plt.figure(figsize=[12,8])
summary_distance_duration = pd.DataFrame(train_df.groupby(['direction'])
['trip_duration'].mean())
summary_distance_duration.columns = ['trip_duration']
summary_distance_duration.reset_index(inplace = True)
summary_distance_duration = summary_distance_duration.sort_values('direction')
# summary_total_distance_duration.set_index('total_distance',inplace=True)
sns.set(style="white", palette="muted", color_codes=False)
sns.set_context("poster")
plt.plot(summary_distance_duration['direction'].values,summary_distance_duration['trip_duration']
values)
        # sns.(data=summary_total_distance_duration, row="total_distance", col="trip_duration")
sns.despine(bottom = False)
```



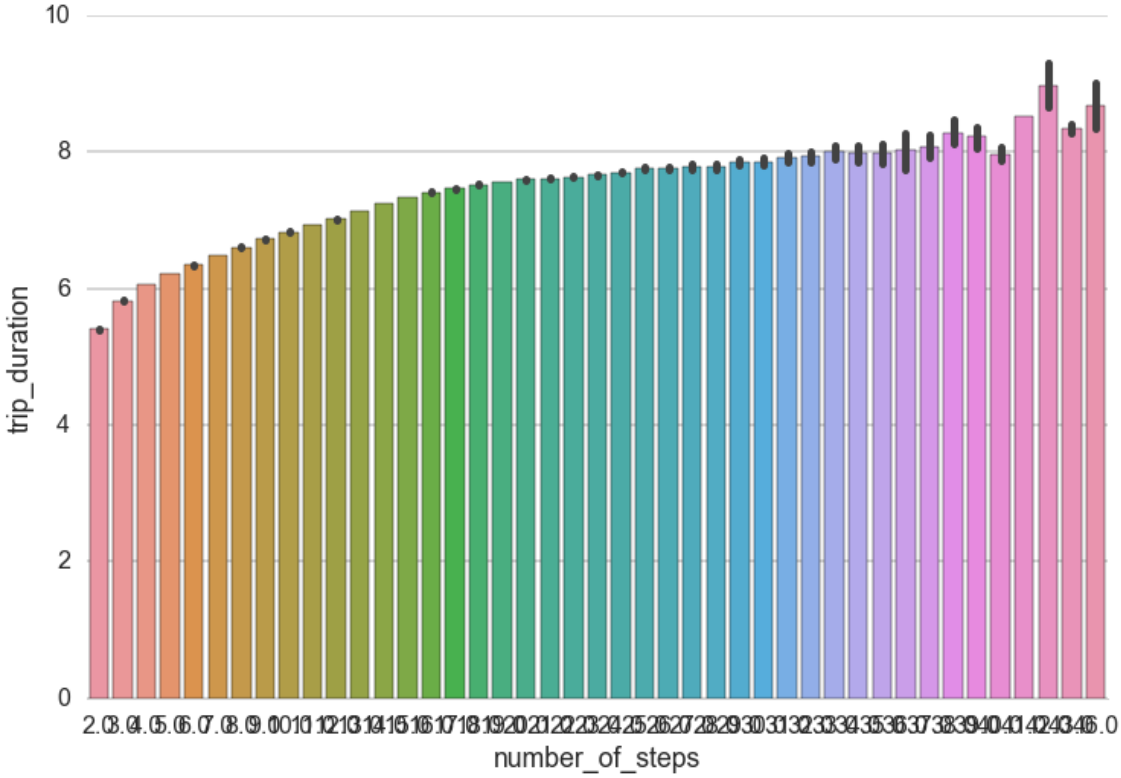

#### 4.3.2.6 街道的数量

因为每条街道都基本上有红绿灯，经过的街道越多，需要通过的红绿灯数也就越多，所以这个也会对我们的行程时间造成非常大的影响。

- 从图中我们很容易发现，街道的数量越多，我们的行程的时间明显就越长。
- 结论：<font color = red> 街道的数量对于我们的行程预测能带来较大的帮助，是一个非常好的特征。

In [61]:

```
plt.figure(figsize=[12,8])
sns.set(style="whitegrid", palette="pastel", color_codes=True)
sns.set_context("poster")
sns.barplot(x="number_of_steps", y="trip_duration", data=train_df)
sns.despine(left=True)
```




#### 4.3.2.7 其他特征

除了上面几个特征之外，我们的算法还加入了很多其他的特征，包括关于某个区域的出租车的平均时速，某个区域每小时驶入的出租车的数量，驶出的出租车数量等等，这些数据可以实时计算得到，是判定某一个地方拥堵情况的很好的指标。

- 每60min(时间可以自己拟定)中有多少载客的出租车驶入某个地区(聚类个数来区分)；
- 计算每个聚类区域每小时驶入的出租车或者驶出的出租车数量等等；
- 我们提供一些原始代码如下，其余的特征工程请参考源代码。

In [62]:

```
df_counts = train_df.set_index('pickup_datetime')[['id']].sort_index()
df_counts['count_60min'] = df_counts.isnull().rolling(3600).count()['id']
train_df = train_df.merge(df_counts, on='id', how='left')


group_freq = '60min'

df_dropoff_counts = train_df \
    .set_index('pickup_datetime') \
    .groupby([pd.TimeGrouper(group_freq), 'dropoff_cluster']) \
    .agg({'id': 'count'}) \
    .reset_index().set_index('pickup_datetime') \
    .groupby('dropoff_cluster').rolling('240min').mean() \
    .drop('dropoff_cluster', axis=1) \
    .reset_index().set_index('pickup_datetime').shift(freq='-120min').reset_index() \
    .rename(columns={'pickup_datetime': 'pickup_datetime_group', 'id': 'dropoff_cluster_count'})

train_df['pickup_datetime_group'] = train_df['pickup_datetime'].dt.round(group_freq)

train_df['dropoff_cluster_count'] = \
    train_df[['pickup_datetime_group', 'dropoff_cluster']].merge(df_dropoff_counts,
        on=['pickup_datetime_group', 'dropoff_cluster'], how='left')['dropoff_cluster_count'].fi
llna(0)
```

#### 4.3.2.8 对数据集3的具体分析

新增的一些重要维度的意义：

- motorway, trunk, primary, secondary, tertiary, unclassified, residential：各种路段，整个路段每个路段占了百分之多少
- nTrafficSignals: 交通标记的个数
- nCrossing: 人行横道的个数
- nStop: 停止信号的个数
- nIntersection：交叉路口的个数 (Intersection can be crossroad, but not a highway exit...)
- srcCounty, dstCounty: 上车和下车的城市

这些新增的特征非常重要，不同路段规定的行驶速度不一样，所以会影响到行程时间，交通标记(例如限速)的数量也会带来较大影响。同样的，红绿灯、交叉路口的个数以及上车城市和下车的城市是否一致，这些信息能给我们进行的旅程时间预测带来极大的帮助。这也给我们搜集数据模型带来非常大的启发。

In [63]:

```
train_aug = pd.read_csv('./data/train_augmented.csv')
```

In [64]:

```
train_aug.head()
```

Out[64]:

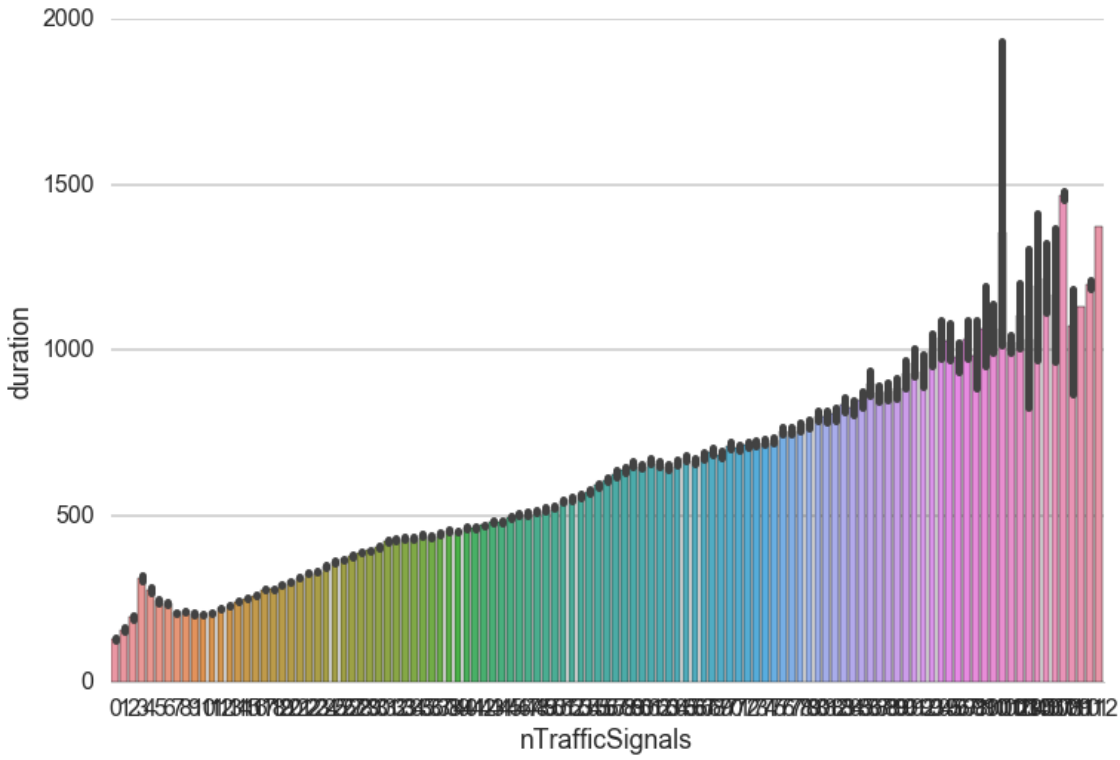
| | id | distance | duration | motorway | trunk | primary | secondary | tertiary |
|---|---|---|---|---|---|---|---|---|
| **0** | id2875421 | 2009.1 | 160.9 | 0.0 | 0.00000 | 0.0 | 0.000000 | 1.000000 |
| **1** | id2377394 | 2513.4 | 256.5 | 0.0 | 0.00000 | 0.0 | 0.348518 | 0.174776 |
| **2** | id3858529 | 9910.7 | 679.6 | 0.0 | 0.54282 | 0.0 | 0.372717 | 0.039806 |
| **3** | id3504673 | 1779.1 | 181.8 | 0.0 | 0.00000 | 0.0 | 0.000000 | 0.424452 |
| **4** | id2181028 | 1615.0 | 132.2 | 0.0 | 0.00000 | 0.0 | 0.637338 | 0.362663 |

为了验证这些数据确实能帮助我们进行行程时间的预测，我们以**nTrafficSignals**和**nCrossing**为例进行简单阐述。

- nTrafficSignals
- 我们发现交通标记越多，那么我们的行程时间往往也会越长，但是两者的关系不是递增的，可能是因为每个地方的交通标记不太一样，而有些交通标志少的地方出现的都是限速类的标记，但是大体上还是呈现出标记越多，行程时间也就越长的规律。

In [65]:

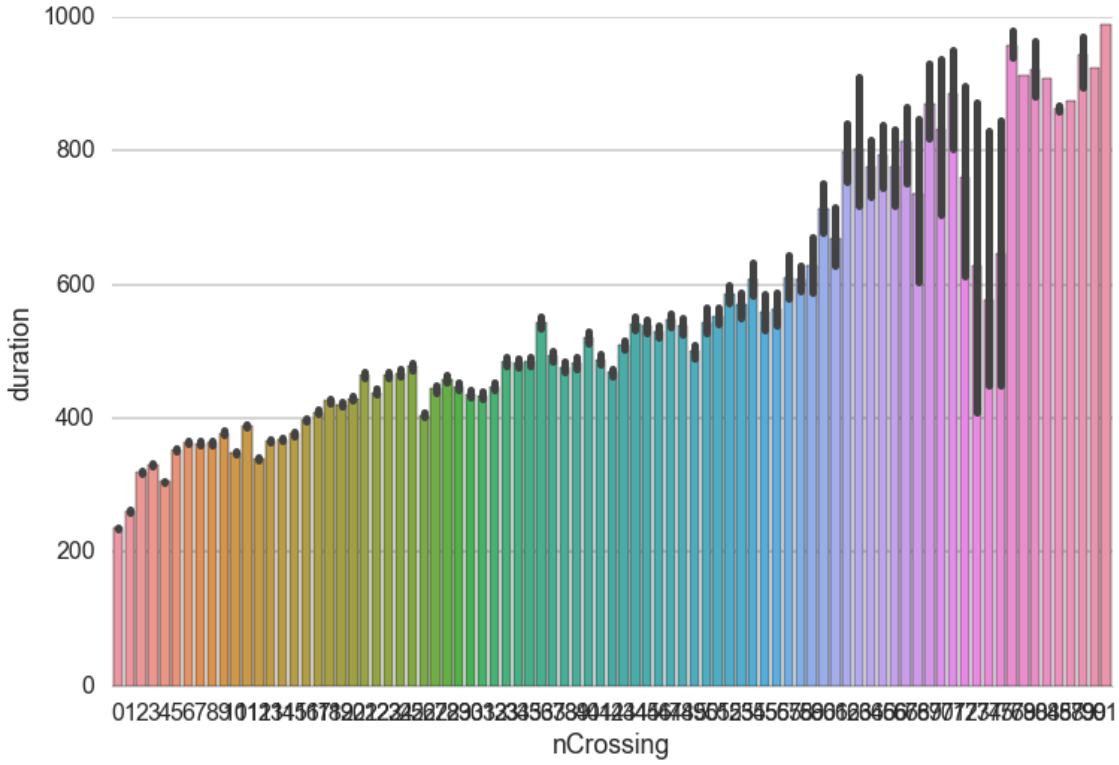
```
plt.figure(figsize=[12,8])
sns.set(style="whitegrid", palette="pastel", color_codes=True)
sns.set_context("poster")
sns.barplot(x="nTrafficSignals", y="duration", data=train_aug)
sns.despine(left=True)
```



- nCrossing
- 我们发现交叉路口越多,行程时间往往也会越长，这是一个非常好的特征。

In [69]:

```
plt.figure(figsize=[12,8])
sns.set(style="whitegrid", palette="pastel", color_codes=True)
sns.set_context("poster")
sns.barplot(x="nCrossing", y="duration", data=train_aug)
sns.despine(left=True)
```




## 4.3.3 模型训练与测试

### 4.3.3.1 模型的训练

模型的训练我们主要尝试了三个工具包：XGBoost、LightGBM、CatBoost 。

- XGBoost: https://github.com/dmlc/xgboost (https://github.com/dmlc/xgboost)
- LightGBM：https://github.com/Microsoft/LightGBM (https://github.com/Microsoft/LightGBM)
- CatBoost: https://github.com/catboost/catboost (https://github.com/catboost/catboost)

此外有一个需要注意的小细节，我们希望训练的时候能够降低模型的方差，所以我们建议用如下的code的形式进行。

### 4.3.3.2 模型的测试

- 将三个模型根据线下CV的结果选择权重$w1,w2,w3$
- 重新把所有数据放入并重新训练得到三个$model,model1,model2,model3$.
- 对test集合进行测试得到三个不同的结果$res1,res2,res3$
- 最终将三个结果用CV得到的权重进行加权$w1* res1 + w2 * res2 + w3 * res3$，并把加权后的结果作为最终的结果。

In [70]:

```
xgb = XGBRegressor(n_estimators=1000, max_depth=10, min_child_weight=150,subsample=0.9, colsampl
e_bytree=0.9)
y_test = np.zeros(len(X_test))
for i, (train_ind, val_ind) in enumerate(KFold(n_splits=3, shuffle=True,random_state=1989).spli
t(X_train)):

    xgb.fit(X_train[train_ind], y_train[train_ind],eval_set=[(X_train[val_ind],
y_train[val_ind])],early_stopping_rounds=10, verbose=25)

    y_test += xgb.predict(X_test, ntree_limit=xgb.best_ntree_limit)

y_test /= 3
```

### 4.3.4 参考资料


[1]. 根据经纬度计算路径,Wiki链接:https://en.wikipedia.org/wiki/Haversine_formula (https://en.wikipedia.org/wiki/Haversine_formula)
[2]. 数据集链接1: https://www.kaggle.com/c/nyc-taxi-trip-duration/data (https://www.kaggle.com/c/nyc-taxi-trip-duration/data)
[3]. 数据集链接2:https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm (https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm)
[4]. https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367 (https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367)
[5]. https://www.kaggle.com/headsortails/nyc-taxi-eda-update-the-fast-the-curious (https://www.kaggle.com/headsortails/nyc-taxi-eda-update-the-fast-the-curious)
[6]. https://www.kaggle.com/mubashir44/xgboost-with-kfold-valid-lb-0-368 (https://www.kaggle.com/mubashir44/xgboost-with-kfold-valid-lb-0-368)
[7]. https://www.kaggle.com/maheshdadhich/strength-of-visualization-python-visuals-tutorial (https://www.kaggle.com/maheshdadhich/strength-of-visualization-python-visuals-tutorial)
[8]. https://www.kaggle.com/misfyre/stacking-model-378-lb-375-cv (https://www.kaggle.com/misfyre/stacking-model-378-lb-375-cv)
[9]. XGBoost: https://github.com/dmlc/xgboost (https://github.com/dmlc/xgboost)
[10].CatBoost:https://github.com/catboost/catboost (https://github.com/catboost/catboost)
[11].LightGBM:https://github.com/Microsoft/LightGBM (https://github.com/Microsoft/LightGBM)
[12].Kaggle Ensembling Guide | MLWave https://mlwave.com/kaggle-ensembling-guide/ (https://mlwave.com/kaggle-ensembling-guide/)
[13].An introduction to variable and feature selection(变量和特征选择的介绍) https://zhuanlan.zhihu.com/p/27829646 (https://zhuanlan.zhihu.com/p/27829646)
[14].Principal component analysis https://en.wikipedia.org/wiki/Principal_component_analysis (https://en.wikipedia.org/wiki/Principal_component_analysis)
[15].https://www.kaggle.com/donniedarko/darktaxi-tripdurationprediction-lb-0-385 (https://www.kaggle.com/donniedarko/darktaxi-tripdurationprediction-lb-0-385)
[16].数据集链接3:https://www.kaggle.com/atmarouane/nyc-taxi-trip-noisy/data (https://www.kaggle.com/atmarouane/nyc-taxi-trip-noisy/data)
[17].数据集3解释:https://www.kaggle.com/atmarouane/nyc-taxi-trip-noisy/version/2 (https://www.kaggle.com/atmarouane/nyc-taxi-trip-noisy/version/2)

# 5. 原型测试

## 5.1 详细说明原型的使用方法

- 环境：python2.7 sklearn 0.18.2 pandas 0.20.3 numpy 1.13.1 jupyter notebook
- 数据：纽约出租车数据，路网数据，天气数据。‘比赛文件/data’文件夹下
- 运行方法：在notebook下运行Mode_Code.ipynb

## 5.2 测试用例

- 采用交叉验证和测试集验证方式
- 评价指标 RMSLE

## 5.3 原型局限性

- 基于数据驱动的模型，依赖于数据，当数据量不足的时候，模型的精度会受到限制，目前获得的外部数据有限，如果能增加数据的多样性和样本数，精度能有所提高。

## 6.3 改进方向

- 本文提出的模型是数据驱动的，依赖于数据,在使用时可以更多地结合其他的数据,例如若能获得司机个人信息数据，可以利用司机的驾驶习惯，做出更准确的预测。

# 6.价值导向

## 6.1 商业价值

- 本作品主要提出了一种高效准确的旅行时间预测算法。该算法可以作为百度、腾讯、高德地图等APP的技术支持，为出行者个人提供准确的出行时间预测，提升用户体验；另外算法可以直接接入用户界面，结合其他领域的大数据，对人们日常中其他的时间也做精确的时间预测，整合成一款帮助人们规划日常生活（包括出行，参加活动，运动等事件）的时间管理APP；算法还可以用于出租车公司，滴滴打车等网约车平台以及运输车辆，帮助企业优化车辆调用，提高运营效率，降低运营成本。

## 6.2 社会价值

- 随着大数据和人工智能时代的发展, 交通领域的问题在学术界和工业界都受到了广泛和高度的关注。越来越多学术界和工业界的专家学者们都致力于解决大数据人工智能时代交通领域中问题， 因为解决这些问题, 将对建设智慧城市和改善人们的出行体验有着重要的意义。准确地预测出行时间，可以作为交通管理部门的一个有效的参考信息，更好地帮助管理人员制定合适的调控策略，改善城市交通，提高人们的生活质量。
- 例如去年的KDD CUP 2017的赛题就是解决交通领域中的2个重要问题, 基于历史数据预测高速路口收费站的流量和通行时间。
- 在国内, DataCastle 在去年的智慧中国杯中, 同样选取了交通线路通达时间预测作为赛题。

## 7. 附件代码

核心代码都在Mode_Code.ipynb里面，里面所有的特征工程都主要基于4中的分析，具体的细节可以参考Mode_Code.ipynb文件。