Latish Khubnani for Auptix

In [1]:
```
import pandas as pd
import json
import pandas as pd
import matplotlib
import cufflinks as cf
import plotly
import plotly.offline as py
import plotly.graph_objs as go
import numpy as np
import datetime
from fancyimpute import KNN
import math

cf.go_offline() # offline
py.init_notebook_mode() # inline graph charts

from pymongo import MongoClient
client = MongoClient('localhost', 27017)
db = client['auptix']

import mysql.connector as mariadb
from sqlalchemy import create_engine
engine = create_engine('mysql+mysqlconnector://root:password@127.0.
```

```
/Users/lkhubnani/anaconda/lib/python3.6/site-packages/h5py/__ini
t__.py:34: FutureWarning:

Conversion of the second argument of issubdtype from `float` to
`np.floating` is deprecated. In future, it will be treated as `n
p.float64 == np.dtype(float).type`.

Using TensorFlow backend.
/Users/lkhubnani/anaconda/lib/python3.6/importlib/_bootstrap.py:
219: RuntimeWarning:

compiletime version 3.5 of module 'tensorflow.python.framework.f
ast_tensor_util' does not match runtime version 3.6
```

In [12]:
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
```

In [3]:
```
# Utility
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 75)

def distance(lat1, lon1, lat2, lon2):
    """
         :return: distance in meters between two co-ordinates
         ref:- http://www.movable-type.co.uk/scripts/latlong.html;
         for accuracy to 1mm use : http://www.movable-type.co.uk/sc
         converted javascript code into python by Latish K
    """

    if lat1 and lon1 and lat2 and lon2:

        radian_lat1 = math.radians(lat1)
        radian_lat2 = math.radians(lat2)
        delta_radians = math.radians(lon2 - lon1)
        R = 6371000
        try:
            d_in_meters = math.acos(
                math.sin(radian_lat1) * math.sin(radian_lat2) + mat
                    radian_lat2) * math.cos(delta_radians)) * R
        except ValueError:
            d_in_meters = 0
        return d_in_meters
    else:
        return -1
```

1. The data in tracking_activity.csv gives scheduled pickup and delivery times for a group of shipments. If a pickup or delivery is going to be late, we add a new tracking record and indicate which party is responsible for the change. We want to generate a report indicating whether each shipment is scheduled to be picked up on time and who is responsible if it is going to be late. Write a query to transform the data into the required format.

Sol:

Reading file into DataFrame. prepare the collection, find the latest pickup and oldest pickup date, diff between them will be delay

In [4]:
```
df = pd.read_csv('Assessment/tracking_activity.csv')
df['date_value'] = pd.to_datetime(df['date_value'], format='%Y-%m-%
# df.to_sql(name='tracking', con=engine, if_exists='replace', index
```

In [5]:
```
result = df.groupby(['shipment_id', 'tracking_activity_type'])['dat
result['diff'] = result['max'] - result['min']
result = result.unstack()
result.columns = ['_'.join(col).strip() for col in result.columns.v
result.reset_index(inplace=True)
new = pd.merge(result, df, on='shipment_id', how='left')
```

Filter the latest ones, as those are the most current orderes. Get latest tracking ids by group

In [6]:
```
new["latest_tracking_id"] = new.groupby(['shipment_id', 'tracking_a
latest_data = new[new['latest_tracking_id'] == new['tracking_activi
latest_data
```

Out[6]:

	shipment_id	max_DELIVERY_ESTIMATED	max_PICKUP_SCHEDULED	min_DELIVERY_I
6	163831	2018-08-21	2018-08-13	
8	163831	2018-08-21	2018-08-13	
13	163966	2018-08-15	2018-08-13	
14	163966	2018-08-15	2018-08-13	
15	163979	2018-08-23	2018-08-16	
17	163979	2018-08-23	2018-08-16	
20	163981	2018-08-15	2018-08-14	

In [7]:
```
pd.options.mode.chained_assignment = None
```

In [8]:
```
latest_data["days_pickup_delayed"] = latest_data["diff_PICKUP_SCHED
latest_data["days_delivery_delayed"] = latest_data["diff_DELIVERY_E

# temp = pd.pivot_table(.dropna(subset=['responsible_party']), aggf
a = latest_data[['tracking_activity_type', 'responsible_party', 'da
```

The Data Set is

In [9]:
```
a.head(20)
```

Out[9]:

	tracking_activity_type	responsible_party	days_pickup_delayed	latest_tracking_id	da
6	PICKUP_SCHEDULED	CUSTOMER	3	414587	
13	PICKUP_SCHEDULED	CARRIER	3	413664	
14	DELIVERY_ESTIMATED	CARRIER	3	413665	
17	DELIVERY_ESTIMATED	AUPTIX	0	412259	
20	PICKUP_SCHEDULED	CARRIER	1	416606	
21	DELIVERY_ESTIMATED	CARRIER	1	416609	
26	PICKUP_SCHEDULED	CUSTOMER	4	416605	
27	DELIVERY_ESTIMATED	CUSTOMER	4	416608	
31	PICKUP_SCHEDULED	CARRIER	1	415735	
32	DELIVERY_ESTIMATED	CARRIER	1	415736	
35	PICKUP_SCHEDULED	CARRIER	3	412539	
36	DELIVERY_ESTIMATED	CARRIER	3	412540	
39	DELIVERY_ESTIMATED	CARRIER	0	413929	
49	PICKUP_SCHEDULED	CUSTOMER	1	416306	
50	DELIVERY_ESTIMATED	CUSTOMER	1	416309	
55	PICKUP_SCHEDULED	CUSTOMER	0	414501	
65	PICKUP_SCHEDULED	AUPTIX	3	412462	
66	DELIVERY_ESTIMATED	CARRIER	3	412464	
69	PICKUP_SCHEDULED	CUSTOMER	3	414356	
70	DELIVERY_ESTIMATED	CUSTOMER	3	414357	

2. Again using the tracking activity data, write a script that will help you determine whether any of the responsible parties produce longer delays in pickup or delivery times. Assume that this is just a small subset of the overall data, so write your script assuming there will be a much larger input file with the same format and don't worry about interpreting the results for the sample data.

Possible soultions for large datasets:

- can use sql based dataset with indexing postgres.
- can use same script as above with Dask package which is parallel processing for large dataframes http://dask.pydata.org/en/latest/ (http://dask.pydata.org/en/latest/)
- can use mongodb

Question to be answered:

1. % of delays caused in pickup and delivery by different parties (AUPTIX, Carrier, Customer)
    - get tables of distribution of delays in pickup and delivery

Question to be answered can be answered
(with new external data)

1. With the deliveries delayed by carrier:
    - distribution of delays caused by different carriers (find the tracking_id and join with carrier table to get distribution)
    - is it limited to the carrier only or the region?
    - dates associated, are they due to external factors? (storms, distruption in transport)
2. For the ones which our company, Auptix is responsible did it affect same customer?
    - would need find customer affected from extra data to be joined by shipment_id/tracking_id and then count instances

In [10]:
```
final_data = pd.pivot_table(a, aggfunc='count', columns='responsibl
final_data.rename(columns={'shipment_id':'responsible_party'}, inpl
temp.iplot(kind='bar', xTitle='days_delivery_delayed', yTitle = 'Co
```

```
---------------------------------------------------------------------
-----------
NameError                                 Traceback (most recent
 call last)
<ipython-input-10-166bbdd81988> in <module>()
      1 final_data = pd.pivot_table(a, aggfunc='count', columns=
'responsible_party', index= ['days_delivery_delayed'], fill_valu
e=0)
      2 final_data.rename(columns={'shipment_id':'responsible_pa
rty'}, inplace=True)
----> 3 temp.iplot(kind='bar', xTitle='days_delivery_delayed', y
Title = 'Count' , barmode='stack', title='Count of Responsible P
arty vs # of Days Delivery Delayed')

NameError: name 'temp' is not defined
```

In [ ]:
```
a = latest_data[['responsible_party','days_pickup_delayed','shipmer
temp = pd.pivot_table(a, aggfunc='count', columns='responsible_part
temp.rename(columns={'shipment_id':'responsible_party'}, inplace=Tr
temp.iplot(kind='bar', xTitle='days_pickup_delayed', yTitle = 'Cour
```

3. Assume you have a list of free-text notes taken by sales reps during their calls and you know they indicate their estimate of a customer's size using hashtags: #small, #medium, or #large. Assume each note begins with the company's name followed by a newline character. Take whatever approach you consider appropriate to analyze and report on these notes.

Sol:

- if each file is like

```
COMPANY_NAME
#size
```

- Will *parse* every file and get the info, customer, size and when note was created/modified. And insert it into a Data Base.
- Get the customer info from database/file/dataFrame and perform analytics like
  - Customer change in size, and report the same
  - Biggest customers based on size
  - Filter out the ones which reduced in size, so that sales can possibly look into it
  - Highlight the success with increase in size
  - Get customer_id and *join* with previous records and see the avg amount of business given to us weekly/monthly/yearly
  - Can create dashboard for all these to update as data updates

In [ ]:
```
import sys
import glob
import errno

customer_size = {}
path = '../sales_notes'
notes = glob.glob(path)  #get paths of all sales notes
for note in notes:
    try:
        with open(note) as f: # read files
            lines = f.readlines()
            if lines:
                customer_size['company_name'] = lines[0].strip() #
                customer_size['size'] = lines[1].strip('#')
                try:
                    ctime = os.path.getctime(note)
                except OSError:
                    ctime = 0
                last_modified_date = datetime.fromtimestamp(ctime)
                customer_size['last_reported'] = last_modified_date
    except IOError as exc:
        if exc.errno != errno.EISDIR:
            print(exc)

if customer_size: # check if we retrieved any new data
  updated_data = pd.DataFrame(customer_size)
  updated_data.to_csv("updated_customer_info.csv")
# or insert into sql.
```

4. Assume you have a list of orders placed by customers with the following structure:

customer_id	order_date	order_amount
1	2018-06-15	100.00
1	2018-08-02	250.00
50	2018-05-29	900.00
...	...	...
1000	2018-05-29	550.00

write a script to produce a monthly summary indicating whether each customer spent more, less, or the same amount as the previous month. Describe any assumptions you need to make and/or additional resources you would want to use for this task.

Sol:

Assumptions:

- 'previous month' means last month, if there is no data for last month it not appear in solution.
- If we want the 'previous month' as latest month will have to basically choose `month` = min(month)

In [ ]:
```
new_df = pd.DataFrame({'customer_id':[1,1,1,50,50,1000, 1000],
 'order_date':['2018-06-15','2018-05-15','2018-07-15','2018-08-02',
 'order_amount':[100,150, 45, 200,300,900,550]})
new_df['date_value'] = pd.to_datetime(new_df['order_date'], format=
# new_df.to_sql(name='tracking_4', con=engine, if_exists='replace',
```

Outputs and queries

```
select * from tracking_4;
```

```
+-------------+------------+--------------+----------------
-----+
| customer_id | order_date | order_amount | date_value
     |
+-------------+------------+--------------+----------------
-----+
|           1 | 2018-06-15 |          100 | 2018-06-15 00:0
0:00 |
|           1 | 2018-05-15 |          150 | 2018-05-15 00:0
0:00 |
|           1 | 2018-07-15 |           45 | 2018-07-15 00:0
0:00 |
|          50 | 2018-08-02 |          200 | 2018-08-02 00:0
0:00 |
|          50 | 2018-05-29 |          300 | 2018-05-29 00:0
0:00 |
|        1000 | 2018-05-30 |          900 | 2018-05-30 00:0
0:00 |
|        1000 | 2018-05-29 |          550 | 2018-05-29 00:0
0:00 |
+-------------+------------+--------------+----------------
-----+
7 rows in set (0.000 sec)
```

```
select c.customer_id, c.month_, c.amount - d.amount, (c.mon
th_ - d.month_) month_differnce
from (select customer_id, month(date_value) month_, sum(ord
er_amount) amount
      from tracking_4
      group by customer_id, month(date_value)) c
        join (select customer_id, month(date_value) month_,
 sum(order_amount) amount
             from tracking_4
             group by customer_id, month(date_value)) d on
 c.customer_id = d.customer_id
where (c.month_ - d.month_) = 1
order by c.customer_id, c.month_;
```

```
+-------------+--------+---------------------+------------
----+
| customer_id | month_ | c.amount - d.amount | month_differ
nce |
+-------------+--------+---------------------+------------
----+
|           1 |      6 |                 -50 |
  1 |
```

```
|            1 |       7 |                   -55 |
  1 |
+--------------+---------+-----------------------+-------------
----+
```

5. Examine the data in the excel file and use it to recommend potential changes to our pricing. This is an intentionally open-ended task and we are primarily interested in your process, not your conclusions.

- Include any tables, visualizations, or other summary output you generate.
- Use any tools and techniques you like.
- If applicable, briefly describe any additional analysis you would recommend beyond what you have time to complete. Assume other analysts might need to repeat or modify your analysis in the future, so make an effort to document your analysis accordingly.
- Feel free to note areas where you would want clarification in a real-world setting, but for the sake of this exercise please use your best judgment about what these fields represent and how they relate to one another.

Sol:

Optimizing the pricing is finding the trade-off between cost, profit, and volumne

Analysis Required:

- (1) Average pricing (& average pricing per mile per pound), total revenue per mile, total_profit, volume, average margin, total_margin
  - grouped by categories (identify as we go)
  - group by tier

- (2) If margins from (1) are positive for a group:
  - if margin is above our `required` margins/profits, we can distribute the difference amongst customers of that category with negative margins as compared to target
  - if margin is negative we will have to increase price
- (3) compare the numbers from (1) above with each order or customer:
  - if average margins were not met with the customer we would have to dig deeper and see the reasons
    - was it the volume/ load size/ shipping company price?
    - based on that set volume limitation or distance limitation
- (4) Futher analysis would be based on origin and destination:
  - Which routes have the most traffic
    - one quick analysis would be sorting by
      - origin and/or destination lat,lon
      - then by origin and destination region,

        - then by distance (Currently the precision is 3 decimal places (upto 110 meters) and this can be reduced to 1 to have precision of 6 miles or 11 km this would help in grouping close places but not too close enough)
      - then compare the average price of these and change pricing similarly as in (3) above
- (5) The customers with large volume can be provided pricings at decreased margins and lower volumes, at increased margins*

Assumptions:

- `shipment_id` for one ordered is unique i.e two shipping_ids for same customer id would mean two different orders

- `lane tiers` are traffic densities of the road

- WonLoss = `Won` and `Status != ORDERED` are the orders for which deals have been struck, therefore, can be used as dataset to understand the profits and suggest pricing

- `tier_id` is tier based on the order, (this can be verified based on co-relation with volumn and distance or confirmed with the data owners, as something which has been tagged, doesn't need to be *re-derived*)

Importing tables into the notebook

In [ ]:
```
xls = pd.ExcelFile('Assessment/Take-home Test.xlsx')
shipment = pd.read_excel(xls, 'shipment')
quote = pd.read_excel(xls, 'quote')
shipment_item = pd.read_excel(xls, 'shipment_item')
freight_bill = pd.read_excel(xls, 'freight_bill')
win_loss_key = pd.read_excel(xls, 'win_loss_key')

# can export to database

# new_df.to_sql(name='tracking_4', con=engine, if_exists='replace',
# shipment.to_sql(name='shipment', con=engine, if_exists='replace',
# quote.to_sql(name='quote', con=engine, if_exists='replace', index
# shipment_item.to_sql(name='shipment_item', con=engine, if_exists=
# freight_bill.to_sql(name='freight_bill', con=engine, if_exists='r
# win_loss_key.to_sql(name='win_loss_key', con=engine, if_exists='r
```

Data Cleaning

shipment table

Shipment has invalid postal codes, missing `approximate_driving_route_mileage` and geo-cordinates. Out of this approximate_driving_route_mileage is required and that can be filled with calculating distance from geo points. Will use the havershine distance funciton above to do so.

For now, if it's missing co-ordinates and also `approximate_driving_route_mileage` then I shall remove the row. Given more time I would have reverse searched based on postal code and put approximate distance

gross margin has missing values, Sorted the data based on net invoice amount(and gross profit), visually observed that margin values have good co-relation with net invoice amount. Can use KNN on it to fill empty values

In [ ]:
```
def fill_missing_distance(row, mileage_col='approximate_driving_rou
                          orign_lat='origin_approximate_latitude',
                          origin_lon='origin_approximate_longitude'
                          des_lat='destination_approximate_latitude
                          des_lon='destination_approximate_longitud
    lat1 = row[orign_lat]
    lon1 = row[origin_lon]
    lat2 = row[des_lat]
    lon2 = row[des_lon]

    if np.isnan(row[mileage_col]):
        row[mileage_col] = round(distance(lat1, lon1, lat2, lon2) *
    return row
```

In [ ]:
```
shipment = shipment.apply (lambda row: fill_missing_distance(row),a
```

For easier visual relational analysis of attribute columns, joining all the tables.

In [ ]:
```
ship_quote = pd.merge(shipment, quote, on='quote_id', how='left', s
ship_quote_item = pd.merge(ship_quote, shipment_item, on='shipment_
ship_quote_item_freight = pd.merge(ship_quote_item, freight_bill, l
ship_quote_item_freight_win_loss_key = pd.merge(ship_quote_item_fre
complete = ship_quote_item_freight_win_loss_key


complete['density'] = complete['weight_lbs']/(complete['length_in']

#Remove the empty gross margins and status in ['ORDERED', 'TESTING_
complete_won = complete[complete['WonLoss'].eq('Won') & (~complete.
complete_lost = complete[complete['WonLoss'].eq('Lost') & (~complet
```

In [ ]:
```
# a = pd.DataFrame(KNN(k=3).complete(complete_won[['net invoice amo
# a['gross margin_']
# complete_won['gross margin updated_'] = pd.Series(a['gross margin
# a['gross margin_']
```

Faced problems assigning the matrix column to data frame column so, decided to remove (2) rows with missing gross margin calues

In [ ]:
```
complete_won['margin_per_lb_per_mile'] = complete_won['gross margin
```

Though cost might be based on density and milegage, It requires more input about company's model. For now I'll use average margin per pound per mile for further analysis.

Analysis

`complete_won` is data set for which we have successful orders. We can work on it further for analysis.

Average margin per pound per mile is:

In [ ]:
```
average_margin = complete_won['margin_per_lb_per_mile'].mean()
average_margin
```

Average grossed margin per mile is positive, even though we have total gross margin negative:

In [ ]:
```
complete_won['gross margin'].sum()
```

We can distribute this amount to break even / gain profit: For this exercise purposes, I'll choose to increase the margin_per_lb_per_mile to average_margin_per_lb_per_mile in different categories. We can target based on

`carrier name, revenue category, lane tier, tier_id (order tier)`

In [ ]:
```
complete_won['average_margin'] = average_margin
```

In [ ]:
```
def get_table_data(df):
    trace = go.Table(
        header=dict(values=list(df.columns),
                    fill = dict(color='#C2D4FF'),
                    align = ['left'] * 5),
        cells=dict(values=[df.tier_id, df.margin_per_lb_per_mile],
                   fill = dict(color='#F5F8FF'),
                   align = ['left'] * 5))

    data = [trace]
    return data
```

Average Margin based on Tier

In [ ]:
```
average_margin_by_tier = complete_won.groupby(by=['tier_id'])[['mar
average_margin_by_tier['adjust_price_per_lb_per_mile_by'] = average
average_margin_by_tier
```

Tier 1 and 4 need to be looked into and prices should be adjusted by 0.000078 and 0.000084 per pound per mile in these tiers

Similarly we can break down prices based on lane tier (traffic) and size of the load (tier_id)

Converting above code into function

In [ ]:
```
def get_group_by_margins(by_list=['tier_id']):
    average_margin_by_tier = complete_won.groupby(by=by_list)[['mar
    average_margin_by_tier['adjust_price_per_lb_per_mile_by'] = rou
    return average_margin_by_tier

get_group_by_margins(['tier_id', 'lane tier'])
```

Suggestions in this case would be to change the prices for the ones where `adjust_price_per_lb_per_mile_by` is positive only as we don't want to give discounts for now.

Next grouping by `revenue category, lane tier, and carrier` to see which carriers should be negotiated with

In [ ]:
```
get_group_by_margins(['revenue category'])
```

Looks like the margin LTL is close to average. Probably due to

dataset having LTL revenue category as dominant by numbers. This also tells us that choosing a average margin as target might not be best choice. But, as mentioned before, we can also calculate the target margin by distributing the difference and using it in place of `average_margin_per_lb_per_mile : 0.000028`

In [ ]:
```
target_distribution = -complete_won['gross margin'].sum()/complete_
target_distribution
```

In [ ]:
```
print(target_distribution*100/average_margin, " %")
```

As we can see that on an average the price per pound per mile needs to be increased by 0.047% to break even.

Below table can be created with targeted margins. Which can be calulated based on profit targets and column average_margin will be adjusted in that case

In [ ]:
```
get_group_by_margins(['fulfillment region','destination fulfillment
```

In [ ]:
```
complete_won
```

In [ ]:
```

```