Part 3 - Data analysis

Uber's Driver team is interested in predicting which driver signups are most likely to start driving. To help explore this question, we have provided a sample1 dataset of a cohort of driver signups in January 2015. The data was pulled a few months after they signed up to include the result of whether they actually completed their first trip. It also includes several pieces of background information gather about the driver and their car.

We would like you to use this data set to help understand what factors are best at predicting whether a signup will start to drive, and offer suggestions to operationalize those insights to help Uber

Comments:

- I have given myself about 2hr and 10 min for this part of the take home challenge.
- I have utilized python logging module to log the program execution as well as to display outputs
- I have noted my observations, comments and inferences following the output/log/plots for each cell
- Few of the plots are interactive plots that are best viewed in the .html version of this notebook
- If I were to have more time I would optimize the machine learning algorithms and would spend more time on identifying critical patterns in the behaviour of driver signups taking first trip

1. Perform any cleaning, exploratory analysis, and/or visualizations to use the provided data for this analysis (a few sentences/plots describing your approach will suffice). What fraction of the driver signups took a first trip?

In [1]:
```
import pandas as pd
import numpy as np
import logging
logging.basicConfig(level=logging.DEBUG)


import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import plotly.plotly as py
import plotly.tools as tls
from plotly.graph_objs import *
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
tls.set_credentials_file(username="*****", api_key="*****")

from imblearn.under_sampling import RandomUnderSampler

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import StratifiedKFold, cross_val_score,train_test_split
```

In [2]:
```
def loadData(filename):
    """
    # read data from file
    # show basic properties of the input dataset at a glance
    :param filename: absolute name of the data file
    :return: pd.DataFrame
    """
    logging.info("reading in data")
    # read data from file
    df = pd.read_csv(filename)
    # show basic properties of input dataset
    logging.info("properties of data at a glance:")
    logging.info("#rows:{}, #cols:{}".format(df.shape[0], df.shape[1]))
    logging.info("columns:{}".format(list(df.columns)))
    logging.info("#missing data points \n{}".format(df.isnull().sum()))
    return df

df = loadData(r'...\dat\ds_challenge_v2_1_data.csv')
```

```
INFO:root:reading in data
INFO:root:properties of data at a glance:
INFO:root:#rows:54681, #cols:11
INFO:root:columns:['id', 'city_name', 'signup_os', 'signup_channel', 'signup_date', 'bgc_date', 'vehicle_added_date', 'vehicle_make', 'vehicle_model', 'vehicle_year', 'first_completed_date']
INFO:root:#missing data points
id                          0
city_name                   0
signup_os                6857
signup_channel              0
signup_date                 0
bgc_date                21785
vehicle_added_date      41547
vehicle_make            41458
vehicle_model           41458
vehicle_year            41458
first_completed_date    48544
dtype: int64
```

I have converted the given .xlsx as a .csv just to speed things up. The pandas read command can be modified accordingly to read in data from various formats

From the above log, I notice a lot of missing data points in majority of the columns. This may be due to one of the many issues like non-mandatory fields, delays in getting the background check and/or adding the vehicle, non-user friendly partner-app(assuming that this data is from 2016 and new uber-partner app is launched by then) etc.

I also notice that a lot of driver signups have no first_completed_date. This could be due to a server error unable to log the data or the driver has really not taken a first trip. Assuming that this is not the case of a server error, **let us say that if there is a date in the first_completed_date field then the driver signup took a first trip** and move ahead with our analysis

In [3]:
```python
def cleanData(df):
    """
    # clean input data
    # create important features and/or flags
    # extract relevant features
    # answer question 1
    :param df: input dataframe
    :return: cleaned pd.DataFrame
    """
    logging.info("cleaning data...")
    # typecast columns to datetime
    for i in ["signup_date","bgc_date","vehicle_added_date"]:
        df[i] = pd.to_datetime(df[i])
    logging.info("typecasted string to datetime columns")
    # clean vehicle_year column
    df['vehicle_year'] = df['vehicle_year'].replace(to_replace=[0], value=np.NaN)
    logging.info("cleaned vehicle year values")
    # create missing data flags for columns with missing values
    for i in ["signup_os","bgc_date","vehicle_added_date","vehicle_make","vehicle_model","vehicle_year"]:
        col_notnull = i+"_exists"
        df[col_notnull] = df[i].notnull().astype("int")
    logging.info("created exists flags for columns with missing values")
    # handle missing dates
    for i in ["bgc_date","vehicle_added_date"]:
        df[i]=df[i].fillna(value=pd.to_datetime('1/1/2015'))
    logging.info("handled missing dates")
    # compute number of days between each of the major steps in the process
    df["days_signupToBgc"]=(df["bgc_date"]-df["signup_date"]).dt.days
    df["days_signupToVehicleAdd"]=(df["vehicle_added_date"]-df["signup_date"]).dt.days
    df["days_bgcToVehicleAdd"]=(df["vehicle_added_date"]-df["bgc_date"]).dt.days
    logging.info("created #days between signup_date,bgc_date and vehicle_added_date")
    # number of days cannot be negative
    for i in ["days_signupToBgc","days_signupToVehicleAdd","days_bgcToVehicleAdd"]:
        df[i] = df[i].clip(lower=0)
    logging.info("clipped days to min of 0")
    # extract day,month,year,week features from "signup_date", "bgc_date", "vehicle_added_date"
    for i in ["signup_date", "bgc_date", "vehicle_added_date"]:
        i_day, i_month, i_year,i_week = i + "_day", i + "_month", i + "_year",i + "_week"
        df[i_day] = df[i].dt.day
        df[i_month] = df[i].dt.month
        df[i_year] = df[i].dt.year
#         df[i_week] = pd.Series(df[i].dt.strftime("%U").astype(int)+1).astype(str)
        df[i_week] = df[i].dt.strftime("%U")
    # extract year_week features for "signup_date", "bgc_date", "vehicle_added_date"
    df["signup_year_week"] = df["signup_date_year"].astype(str)+df["signup_date_week"].astype(str)
    df["bgc_year_week"] = df["bgc_date_year"].astype(str)+df["bgc_date_week"].astype(str)
    df["vehicle_added_year_week"] = df["vehicle_added_date_year"].astype(str)+df["vehicle_added_date_week"].astype(str)
    logging.info("extracted datetime features")
    # create tookFirstTrip feature
    df["tookFirstTrip"] = df["first_completed_date"].notnull().astype("int")
    logging.info("created took first trip flag")
    # answer question 1
    logging.info("{0:.2f}% of drivers took at a first trip".format(df["tookFirstTrip"].sum() * 100 / df.shape[0]))
    logging.info(df.isnull().sum())
    return df

df_cleaned = cleanData(df)
```

```
INFO:root:cleaning data...
INFO:root:typecasted string to datetime columns
INFO:root:cleaned vehicle year values
INFO:root:created exists flags for columns with missing values
INFO:root:handled missing dates
INFO:root:created #days between signup_date,bgc_date and vehicle_added_date
INFO:root:clipped days to min of 0
INFO:root:extracted datetime features
INFO:root:created took first trip flag
```

```
INFO:root:11.22% of drivers took at a first trip
INFO:root:id                                    0
city_name                        0
signup_os                     6857
signup_channel                   0
signup_date                      0
bgc_date                         0
vehicle_added_date               0
vehicle_make                 41458
vehicle_model                41458
vehicle_year                 41462
first_completed_date         48544
signup_os_exists                 0
bgc_date_exists                  0
vehicle_added_date_exists        0
vehicle_make_exists              0
vehicle_model_exists             0
vehicle_year_exists              0
days_signupToBgc                 0
days_signupToVehicleAdd          0
days_bgcToVehicleAdd             0
signup_date_day                  0
signup_date_month                0
signup_date_year                 0
signup_date_week                 0
bgc_date_day                     0
bgc_date_month                   0
bgc_date_year                    0
bgc_date_week                    0
vehicle_added_date_day           0
vehicle_added_date_month         0
vehicle_added_date_year          0
vehicle_added_date_week          0
signup_year_week                 0
bgc_year_week                    0
vehicle_added_year_week          0
tookFirstTrip                    0
dtype: int64
```

I have tried to improve the signal from the dataset by typecasting the columns appropriately, replacing erroneous data with np.NaN, creating features that flag missing values in columns, extracted datetime features, created features that quantify a delay effect and also our **primary focus feature: tookFirstTrip**

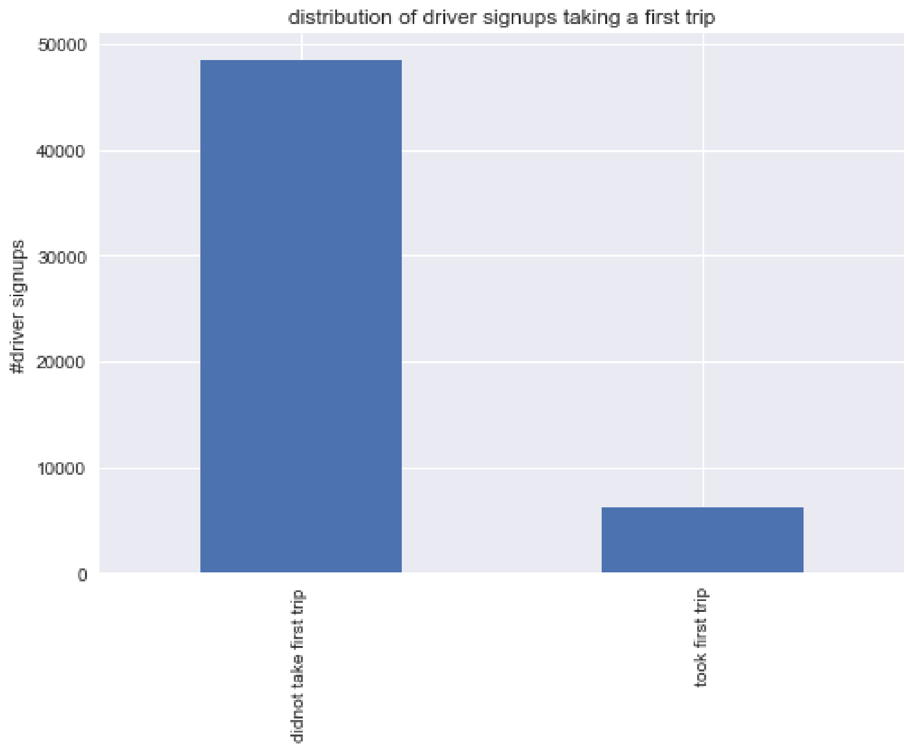
I have assumed that the data that is missing is random and cannot be filled in manually and/or interpolating existing data. I am not fully aware of the business reasons behind the missing data. So I felt it would be safe to create flags that would encode the information if a particular date is available or not and provide the relevant signal

About 11.22% of driver signups took a first trip

In [4]:
```
def plotTookFirstTripDistribution(df_cleaned):
    logging.info("plotting took first trip distribution")
    ax = df_cleaned["tookFirstTrip"].value_counts().plot(kind="bar",title="distribution of driver signups taking a first trip")
    ax.set_ylabel("#driver signups")
    ax.set_xticklabels(["didnot take first trip","took first trip"])
    return

plotTookFirstTripDistribution(df_cleaned)
```

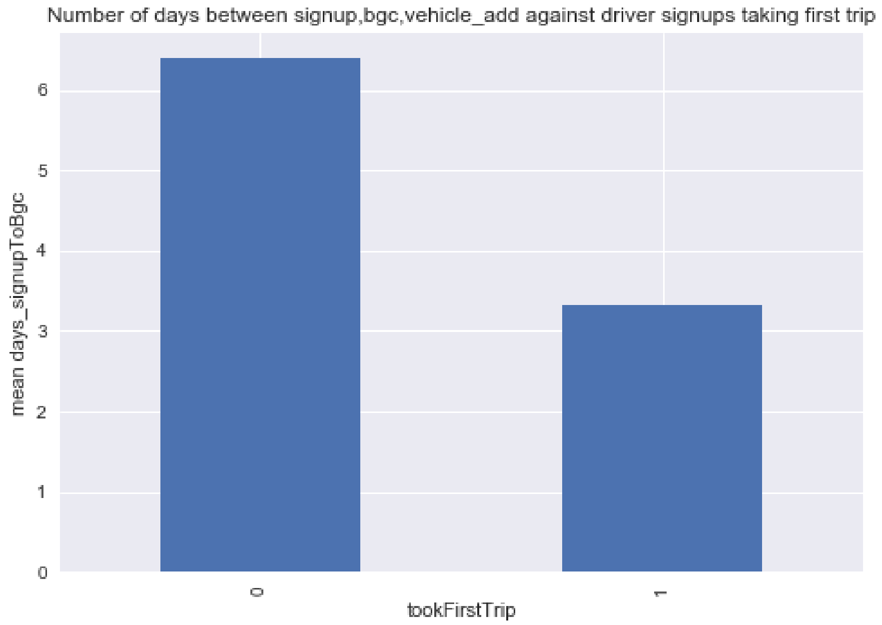
```
INFO:root:plotting took first trip distribution
```

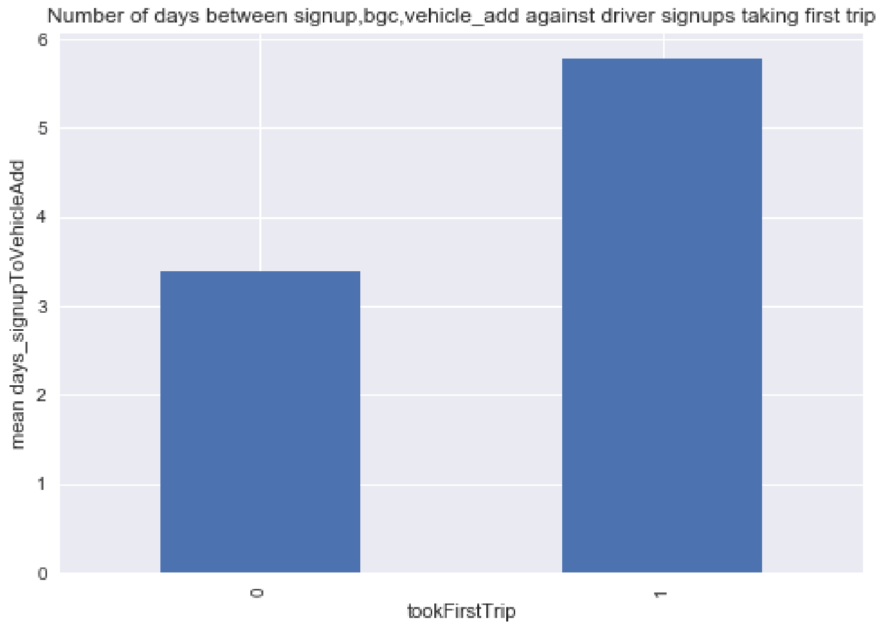


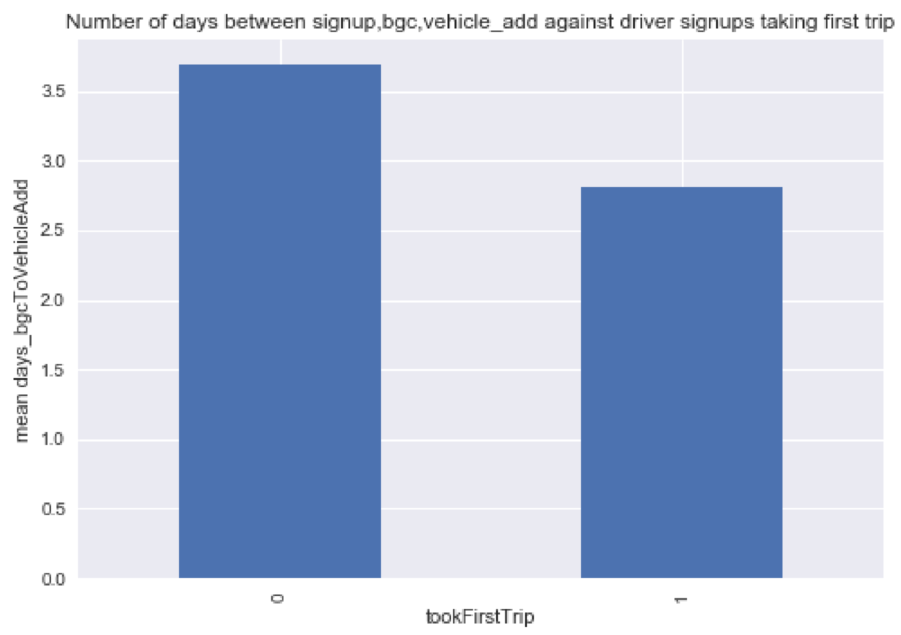
This looks like a heavily unbalanced distribution where only 11.22% of driver signups take first trip. Being oblivious of the industry norm, I would say that is a number that can be improved (hence, the point of this data challenge)

In [5]:
```python
def plotNumberOfDaysWithDriverSignupsTakingFirstTrip(df_cleaned):
    for i in ['days_signupToBgc','days_signupToVehicleAdd','days_bgcToVehicleAdd']:
        ax = df_cleaned.groupby(df_cleaned["tookFirstTrip"]).agg({i:"mean"}).plot(
            kind="bar",legend = False,
            title="Number of days between signup,bgc,vehicle_add against driver signups taking first trip")
        ax.set_ylabel("mean {}".format(i))
    return

plotNumberOfDaysWithDriverSignupsTakingFirstTrip(df_cleaned)
```





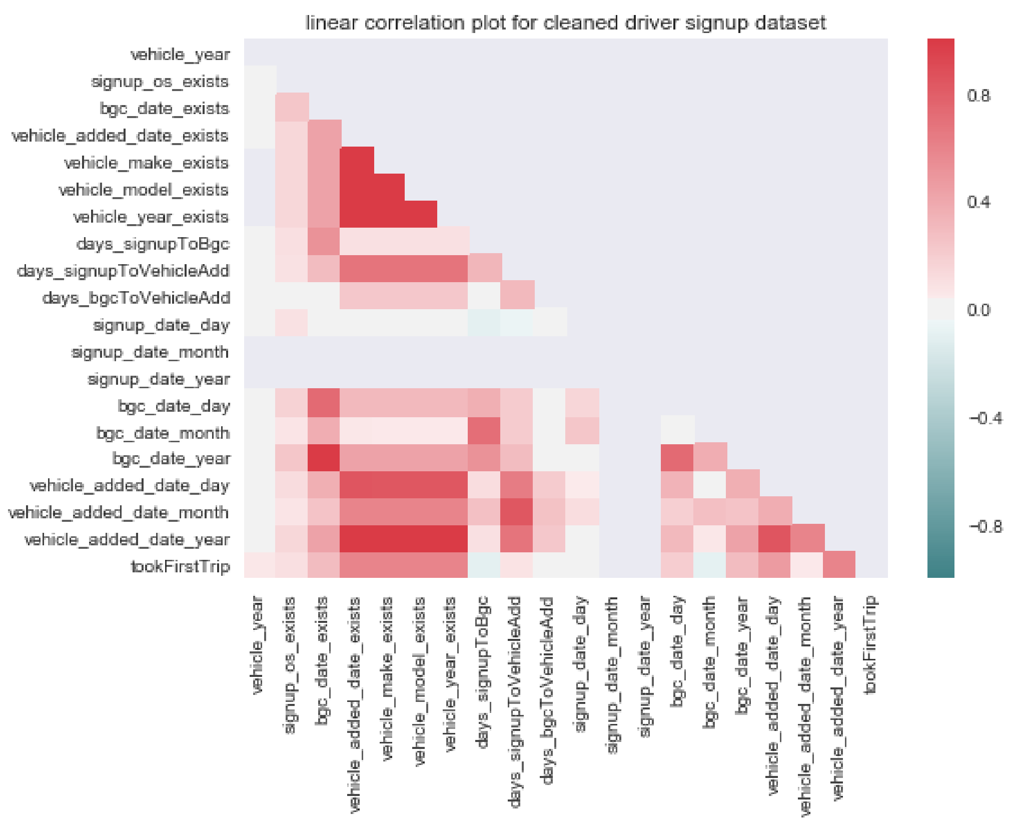


I expected all the bars for 0 to be higher than for 1 but apparently majority of driver signups took first trip despite of having a greater mean #days from signup to vehicle_add. **Interesting!**

In [6]:
```
def plotCorr(df_cleaned):
    logging.info("plotting linear correlations")
    cols = [i for i in df_cleaned.columns if i not in ["id"]]
    corr = df_cleaned.loc[:, cols].corr()
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    sns.heatmap(corr, cmap=sns.diverging_palette(200, 10, as_cmap=True), mask=mask)
    plt.title("linear correlation plot for cleaned driver signup dataset")
    return

plotCorr(df_cleaned)
```

```
INFO:root:plotting linear correlations
```



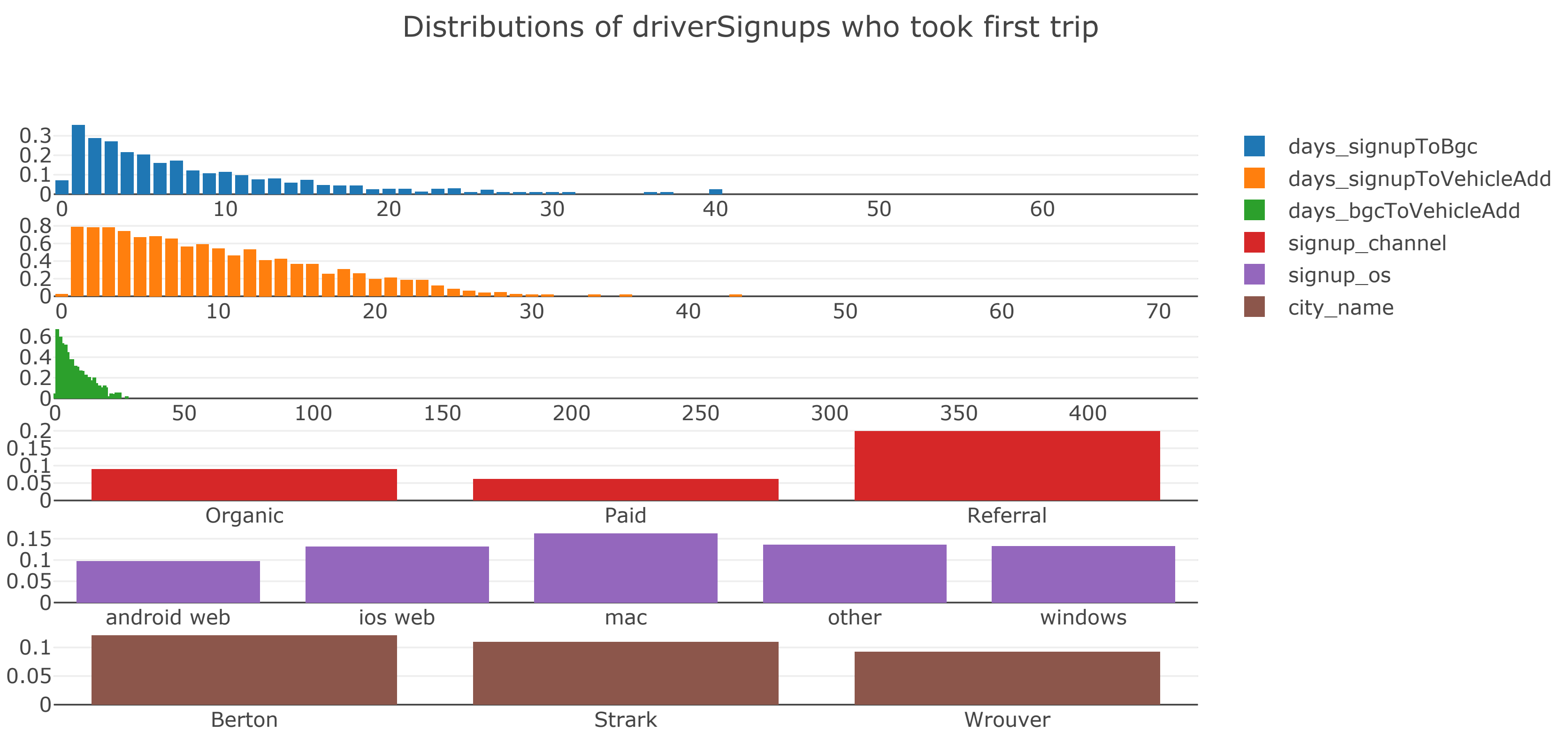
Looking at the above correlation plot I would say everything looks as I would expect. Nothing significant to report here that would be relevant to the problem at hand except for the negative correlation between days from signup to bgc, days from bgc to adding vehicle, day,month of signup date, and driver taking a first trip

In [7]:
```
def plotYDist(df_cleaned):
    logging.info("plotting distribution of driver signups who took first trip across days, signup_channel,signup_os,city_name")
    barCols = ["days_signupToBgc","days_signupToVehicleAdd","days_bgcToVehicleAdd", "signup_channel", "signup_os", "city_name"]
    fig = tls.make_subplots(rows=len(barCols), cols=1)
    x = 1
    for i in barCols:
        df_gby = pd.DataFrame(df_cleaned.groupby(i).agg({"tookFirstTrip": "mean"}).reset_index())
        trace = Bar(x=df_gby[i], y=df_gby["tookFirstTrip"], name=i)
        fig.append_trace(trace, x, 1)
        x += 1
    fig['layout'].update(title='Distributions of driverSignups who took first trip')
    iplot(fig)
    return

plotYDist(df_cleaned)
```

```
INFO:root:plotting distribution of driver signups who took first trip across days, signup_channel,signup_os,city_name

This is the format of your plot grid:
[ (1,1) x1,y1 ]
[ (2,1) x2,y2 ]
[ (3,1) x3,y3 ]
[ (4,1) x4,y4 ]
[ (5,1) x5,y5 ]
[ (6,1) x6,y6 ]
```



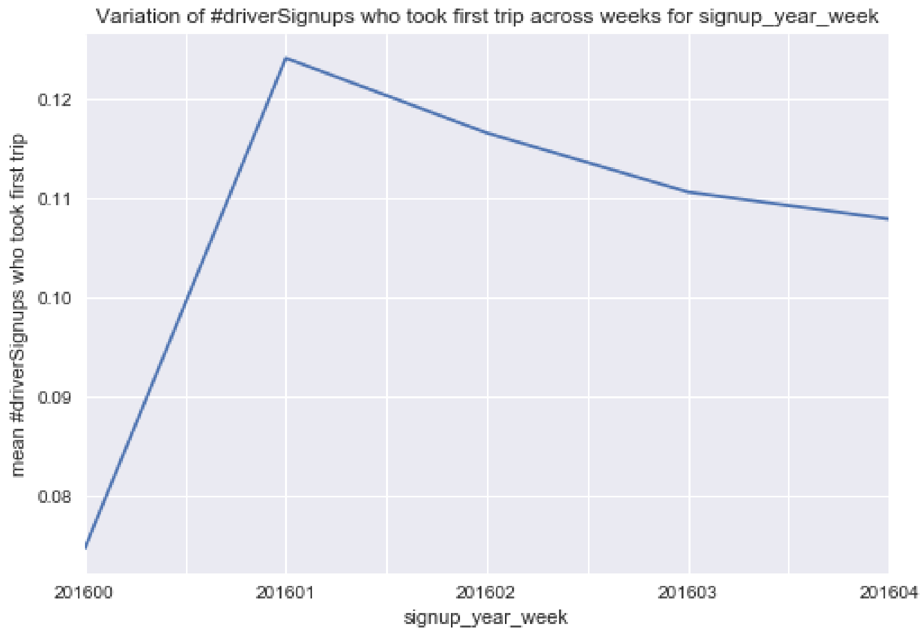
Export to plot.ly »

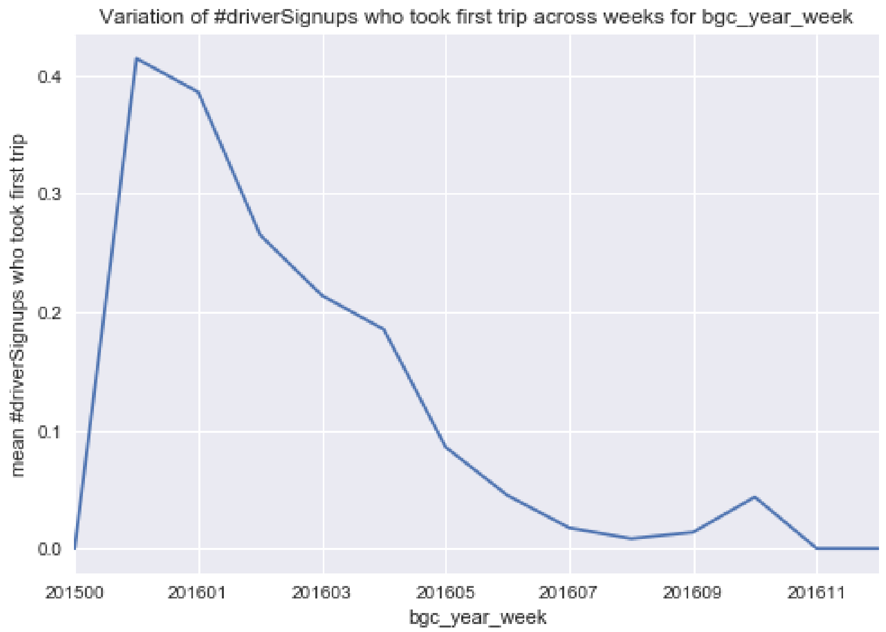
- Looking at the above distributions of driver signups who took first trip, is is obvious that the driver signups show a positively skewed distribution. This implies that the driver signups taking first trip decrease with the increase in the days between signup,bgc and adding the vehicle.
- I notice that the driver signups taking a first trip are noticably higher when they are signed up using a referral. I assume this referral is from other Uber partners because the person who referred and the person who is referred would receive some sort of an incentive/Uber credits.
- I guess better/easy UI on the Apple ecosystem may be a reason for the higher numbers of mac and ios web.
- Either there are more drivers in Berton or the process of getting to the first trip is relatively faster in that city

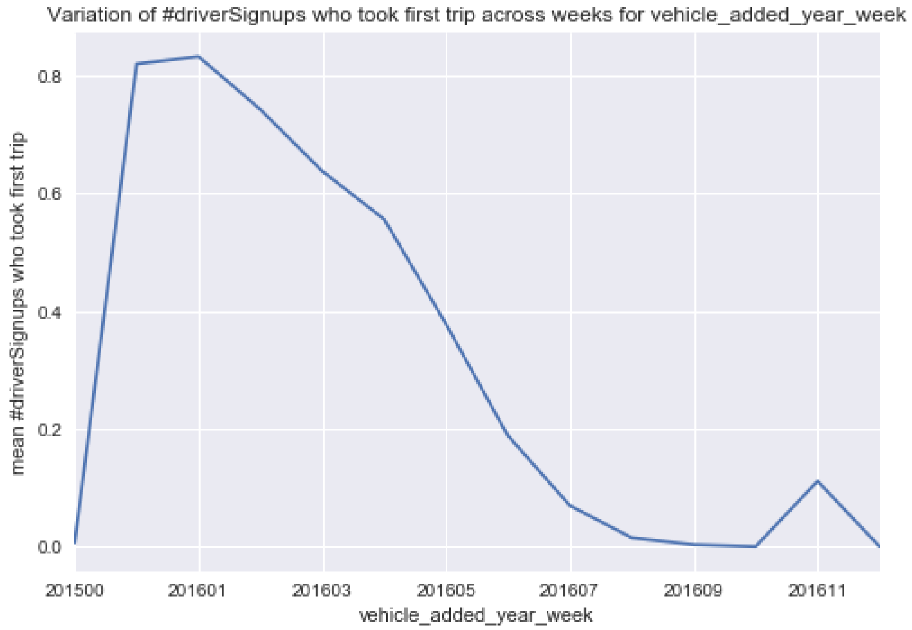
In [8]:
```python
def plotVariationOfNumOfdriverSignupsWhoTookFirstTripAcrossWeeks(df_cleaned):
    logging.info("plotting distribution of number of driver signups who took first trip across different weeks")
    for i in ["signup_year_week","bgc_year_week","vehicle_added_year_week"]:
        df_cleaned[i] = df_cleaned[i].astype('str')
        ax = df_cleaned.groupby([i]).agg({"tookFirstTrip":"mean"}).plot(
            legend=False,title="Variation of #driverSignups who took first trip across weeks for {}".format(i))
        ax.set_ylabel("mean #driverSignups who took first trip")
    return

plotVariationOfNumOfdriverSignupsWhoTookFirstTripAcrossWeeks(df_cleaned)
```

```
INFO:root:plotting distribution of number of driver signups who took first trip across different weeks
```







As expected the driver signups making first trip followed similar trend(seasonality) with bgc and vehicle added along *time dimension* I notice a decreasing trend in the driver signups taking a first trip. Is that *seasonal?*

Driver signups completed first trips more in early January may be because it is a holiday season and people are returning back or are travelling to new places. **May the driver signups taking first drive are influenced by holiday seasonality**

2. Build a predictive model to help Uber determine whether or not a driver signup will start driving. Discuss why you chose your approach, what alternatives you considered, and any concerns you have. How valid is your model? Include any key indicators of model performance

In [9]:
```
def prepData(df_cleaned):
    """
    prep dataframe to input to machine learning model
    :param df_cleaned: cleaned dataframe
    :return: cleaned and prepped pd.DataFrame
    """
    logging.info("prepping data for fitting machine learning models")
    # create dummies for categorical features
    list_dummCols = [i for i in df_cleaned.columns if df_cleaned[i].dtype.name =="object" and
                     i not in ["first_completed_date"]]
    df_cleaned_prepped = pd.get_dummies(df_cleaned, columns=list_dummCols)
    logging.info("created dummies for categorical columns")
    # subset relevant columns
    list_excludeCols = [i for i in df_cleaned_prepped.columns if i not in ["id","first_completed_date", "signup_date",
                                                                           "bgc_date", "vehicle_added_date","vehicle_year"]]
    df_cleaned_prepped = df_cleaned_prepped.loc[:, list_excludeCols]
    return df_cleaned_prepped

df_cleaned_prepped = prepData(df_cleaned)
```

```
INFO:root:prepping data for fitting machine learning models
INFO:root:created dummies for categorical columns
```

Data preparation for fitting machine learning models to get predictions for a driver signup to take first trip includes creating dummies of categorical features and dropping certain columns

In [10]:
```
def balanceClasses(df_cleaned_prepped,seed):
    """
    handle class imbalance problem
    :param df_cleaned_prepped: cleaned and prepped dataframe
    :return: cleaned,prepped and balanced pd.DataFrame
    """
    logging.info("handling class imbalance problem")
    # random undersampling of majority class
    us = RandomUnderSampler(ratio=0.5,random_state=seed)
    X = df_cleaned_prepped.drop("tookFirstTrip",axis=1)
    y = df_cleaned_prepped["tookFirstTrip"]
    X_bal,y_bal = us.fit_sample(X,y)
    logging.info("undersampled not_taken_first_trip class randomly")
    logging.info("dimensions of df_cleaned_prepped:{}".format(df_cleaned_prepped.shape))
    X_bal_df = pd.DataFrame(X_bal,columns=X.columns)
    X_bal_df["tookFirstTrip"] = pd.Series(y_bal)
    logging.info("dimensions of df_cleaned_prepped_balanced:{}".format(X_bal_df.shape))
    logging.info(pd.Series(y).value_counts())
    logging.info(pd.Series(y_bal).value_counts())
    return X_bal_df

df_cleaned_prepped_balanced = balanceClasses(df_cleaned_prepped,3)
```

```
INFO:root:handling class imbalance problem
C:\Users\sg0222350\AppData\Local\Continuum\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:75: DeprecationWarning:

Function _ratio_float is deprecated; Use a float for 'ratio' is deprecated from version 0.2. The support will be removed in 0.4. Use a dict, str, or a callable instead.

INFO:root:undersampled not_taken_first_trip class randomly
INFO:root:dimensions of df_cleaned_prepped:(54681, 508)
INFO:root:dimensions of df_cleaned_prepped_balanced:(18411, 508)
INFO:root:0    48544
1     6137
Name: tookFirstTrip, dtype: int64
INFO:root:0    12274
1     6137
dtype: int64
```

A look at the Y Distribution plot would reveal how unbalanced the distribution of driver signups taking a first trip is. I have made the following the considerations with regards to this fact:

- Anyone can naively predict that all unseen driver signups would not take a first trip and you can be about 87% accurate. So this means that **accuracy might not be a good metric for the model and we need to handle the class imbalance problem**
- In order to come up with something quickly, I tried a few approaches for dealing with class imbalance like Random Over Sampling of minority class, SMOTE and Random Under Sampling. I have noticed that **Random Under Sampling technique returned results that are relatively better than the other techniques**
- I have randomly undersampled the majority class such that the ratio of majority class to minority class is brought down to 2:1
- If I have more time I can run the model mulitple times and come up with a tighter number

In [11]:
```python
def crossValidateModel(df_cleaned_prepped_balanced, seed):
    """
    cross validate and measure generalization of the model
    :param df_cleaned_prepped_balanced: cleaned,prepped and balanced dataframe
    """
    logging.info("assessing how well the Logistic Regression model generalizes using 5-Fold Stratified Cross Validation")
    estimator = LogisticRegression(n_jobs=-1,C=1.0)
    kfold = StratifiedKFold(n_splits=5, random_state=seed)
    X, y = df_cleaned_prepped_balanced.drop("tookFirstTrip",axis=1), df_cleaned_prepped_balanced["tookFirstTrip"]
    for i in ["accuracy","f1_macro"]:
        cvscores = cross_val_score(estimator=estimator, X=X, y=y, cv=kfold, scoring=i, n_jobs=-1)
        logging.info("mean cv {0}:{1}".format(i,cvscores.mean()))
    return

crossValidateModel(df_cleaned_prepped_balanced,3)
```

```
INFO:root:assessing how well the Logistic Regression model generalizes using 5-Fold Stratified Cross Validation
INFO:root:mean cv accuracy:0.9297158275479696
INFO:root:mean cv f1_macro:0.9217119449132678
```

The given problem is a binary classification problem: To predict whether a driver signup will be classified as take_first_trip or doesnt_take_first_trip. In order to iterate rapidly and to avoid overfitting the dataset, I have chosen **logistic regression** model as a baseline model. The results of a quick **5-fold Stratified Cross Validation** measuring both **accuracy and f1 score** for each of the folds prove that the model generalizes well and it is a good baseline model.

If there is more time,resources and better quality data I would use more powerful ML algorithms like SVM or tree based ensembles like Random forest,GBM,xgboost. I would also take some time to use GridSearchCV to come with optimal hyper parameters for the Logistic Regression model

The reason behind using f1 score besides accuracy is that I am interested in knowing how senstive the model is in predicting driver signups who take a first trip and not just how well the model predicts true positives and true negatives

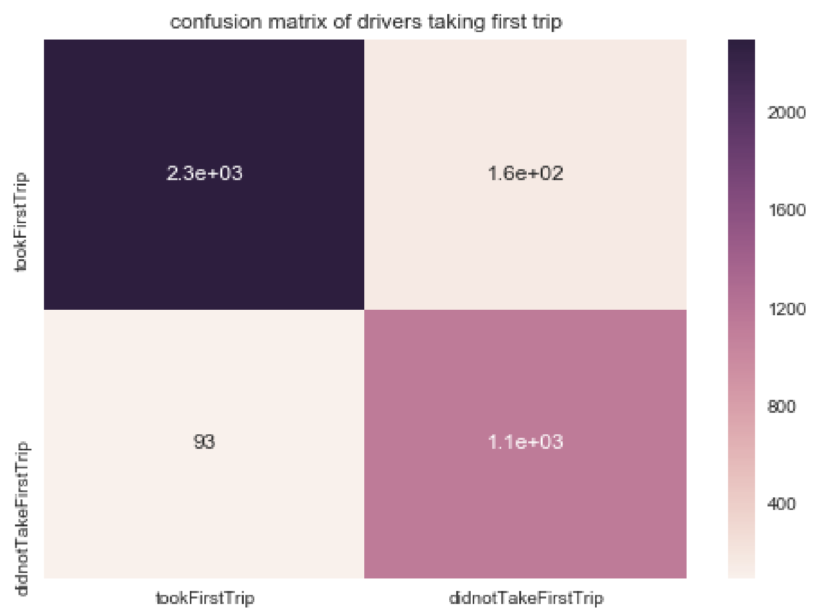
In [12]:
```python
def predictTookFirstTrip(df_cleaned_prepped_balanced,seed):
    """
    predict if a driver signup would take first trip
    :param df_cleaned_prepped_balanced: cleaned,prepped and balanced dataframe
    :param seed: random seed to reproduce results
    :return: estimator,prediction_probability,confusion matrix, classification report, accuracy
    """
    logging.info("assessing the predictive power of the model")
    X, y = df_cleaned_prepped_balanced.drop("tookFirstTrip",axis=1), df_cleaned_prepped_balanced["tookFirstTrip"]
    logging.info("splitting into training and test sets")
    X_train,X_test,y_train,y_test = train_test_split(X,y,stratify=y,test_size=0.2,random_state=seed)
    estimator = LogisticRegression(C=1.0)
    logging.info("fitting the estimator")
    estimator.fit(X_train,y_train)
    logging.info("generating predictions")
    y_pred = estimator.predict(X_test)
    pred_prob = estimator.predict_proba(X_test)
    conf_mat = confusion_matrix(y_test,y_pred)
    logging.info("computing accuracy and classification report")
    class_rep = classification_report(y_test,y_pred)
    accuracy = accuracy_score(y_test,y_pred)
    logging.info("plotting confusion matrix")
    df_confMat = pd.DataFrame(conf_mat, index = ["tookFirstTrip","didnotTakeFirstTrip"],
                              columns = ["tookFirstTrip","didnotTakeFirstTrip"])
    sns.heatmap(df_confMat,annot=True)
    plt.title("confusion matrix of drivers taking first trip")
    TP,FP,FN,TN = conf_mat[0][0],conf_mat[0][1],conf_mat[1][0],conf_mat[1][1]
    logging.info("sensitivity:{}".format(TP/(TP+FN)))
    logging.info("specificity:{}".format(TN/(TN+FP)))
    logging.info("accuracy:{:.2f}".format(accuracy))
    logging.info("classification report: \n{}".format(class_rep))
    list_return = [estimator,pred_prob,conf_mat,class_rep,accuracy]
    return list_return

estimator,pred_prob,conf_mat,class_rep,accuracy = predictTookFirstTrip(df_cleaned_prepped_balanced,3)
```

```
INFO:root:assessing the predictive power of the model
INFO:root:splitting into training and test sets
INFO:root:fitting the estimator
INFO:root:generating predictions
INFO:root:computing accuracy and classification report
INFO:root:plotting confusion matrix
INFO:root:sensitivity:0.9610878661087866
INFO:root:specificity:0.8778035576179428
INFO:root:accuracy:0.93
INFO:root:classification report:
             precision    recall  f1-score   support

          0       0.96      0.94      0.95      2455
          1       0.88      0.92      0.90      1228

avg / total       0.93      0.93      0.93      3683
```



In the absence of a test-set, I have split the available data into training and test sets randomly in 80:20 ratio to assess the performance of the model on unseen data.

A quick look at the confusion matrix also indicates that the model has good sensitivity to predict the signup of drivers who would take a first trip

Both accuracy and f1 score indicate that the model performed well on unseen data

3. Briefly discuss how Uber might leverage the insights gained from the model to generate more first trips (again, a few ideas/sentences will suffice)

Uber might want to do something about the time taken from signup for each of the subsequent steps like bgc and adding vehicle. The delay is somewhat tied to the makes of the vehicles or the time of their signup(too many in a week leading to delays) or problems with the new partner app. These are some of the drivers of driver signups taking a first trip that Uber can use to generate more first trips

I would recommend that we focus not just on predictive accuracy of the model but also on how sensitively the model is able to predict both the classes. This would be helpful to minimize driver signups who dont take a first trip and/or to maximize driver signups who take a first trip

Connecting Dots: Connecting all 3 parts of the challenge

Looks like **all the 3 parts in this data challenge are centered around improving driver-partner Uber app**.

Part1: First week of 2016 is expected to have higher driver_signups taking a first trip. I guess you are trying to give some incentives to drivers who took their first trip within 168 hours of sign_up date in those specific cities (may be they have traditionally low first trip rates)

Part2: I think that you might have deduced that the lower number of driver signups taking first trip is due to a non-ideal uber-partner app. So you are looking for new metrics and sound A/B testing framework to make sure that you have got it right this time with the new Uber driver-partnet app

Part3: You are looking for a predictive model that would predict whether a driver would take first trip or not so that you can restart the cycle from part1 through to part3.

In [ ]:
```
```