Heetch Data Scientist, Algorithms - Technical Test

Author: Ramy Ghorayeb
Date: June 2019

1.1 Context

Users request rides in the Heetch mobile app. Assuming nearby drivers are available, the Heetch backend sends a booking requests to a driver, who can accept or decline the ride.
Note: If the driver declines, Heetch can query one or more extra drivers (under certain conditions), therefore issuing more booking requests for the same ride request.

Build a model that predicts whether or not a driver will accept a given booking request.

2 Loading

In [21]:
```
# General
import io, os, sys, types, time, datetime, math, random, subprocess, t
empfile
import random
from progressbar import ProgressBar
pb = ProgressBar()
import warnings
warnings.filterwarnings('ignore')

# Data manipulation
import datetime
import numpy as np
import pandas as pd
from geopy import distance
from geopy.geocoders import Nominatim
geolocator = Nominatim()

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

# Feature Selection and Encoding
from sklearn.feature_selection import RFE, RFECV
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_b
inarize

# Resampling, Split, Grid and Random Search
from imblearn.over_sampling import SMOTE
from scipy import stats
import scipy.stats as st
from scipy.stats import boxcox
from scipy.stats import randint as sp_randint
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,
train_test_split

# Machine learning
import sklearn.ensemble as ske
from sklearn import datasets, model_selection, tree, preprocessing, me
trics, linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingC
lassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression,
Ridge, Lasso, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# Deep learning
from keras.models import Sequential, Input, Model
```

```
from keras.layers import Dense, Activation, Dropout, LSTM, GRU, Add, C
oncatenate, BatchNormalization

# Metrics
from sklearn.metrics import precision_recall_fscore_support, roc_curve
, auc
```

In [10]:
```
rides = pd.read_csv('data/rideRequests.log')
bookings = pd.read_csv('data/bookingRequests.log')
drivers = pd.read_csv('data/drivers.log')
```

In [123]:
```
requests = pd.read_csv('data/requests_new.csv')
```

3 Data Exploration

3.1 Overview of the dataset

3.1.1. Timeframe of the dataset

Our dataset represents all the rides of the 24h-period from 29/10/18 1PM to 30/10/18 1PM:

In [12]:
```
import datetime

def date_time(timestamp):
    return datetime.datetime.fromtimestamp(timestamp)

first_date = date_time(rides['created_at'][0])
last_date = date_time(rides['created_at'][len(rides)-1])
print('first ride: ',first_date,'\nlast ride: ',last_date)
```

```
first ride:  2018-10-29 13:04:49.561193
last ride:  2018-10-30 12:53:58.996917
```

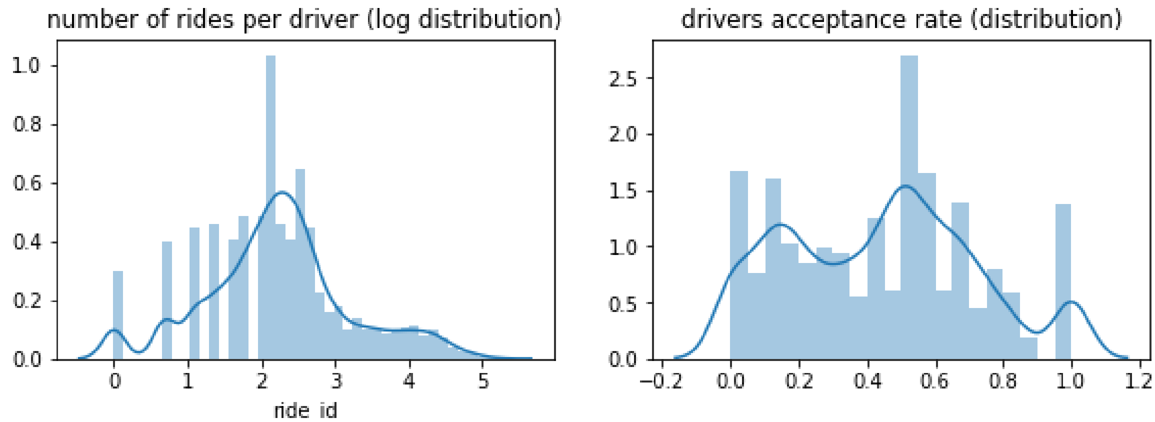
3.1.2 Rides per driver and acceptance rate

Looking at the distributions of rides and acceptance rate, there is a wide diversity of drivers behavior, but as drivers take more and more rides, the average acceptance rate tends to converge to 10% which is quite low.

In [13]:
```
group = bookings.groupby('driver_id')
count_group = bookings.groupby('driver_id').count().describe()
sum_group = bookings.groupby('driver_id').sum().describe()


print('nb of rides: ',len(rides),
      '\nnb of drivers: ',len(drivers),
     '\n-\navg nb of rides per driver',count_group['driver_accepted'][
'mean'],
     '\nmin nb of rides per driver',count_group['driver_accepted']['mi
n'],
      '\nmedian nb of rides per driver',count_group['driver_accepted']
['50%'],
     '\nmax nb of rides per driver',count_group['driver_accepted']['ma
x'],
    '\n-\navg acceptance rate',(group['driver_accepted'].sum()/group[
'driver_accepted'].count()).mean(),
    '\nmin acceptance rate',(group['driver_accepted'].sum()/group['dri
ver_accepted'].count()).min(),
     '\nmedian acceptance rate',(group['driver_accepted'].sum()/group[
'driver_accepted'].count()).median(),
     '\nmax acceptance rate',(group['driver_accepted'].sum()/group['dr
iver_accepted'].count()).max())
```

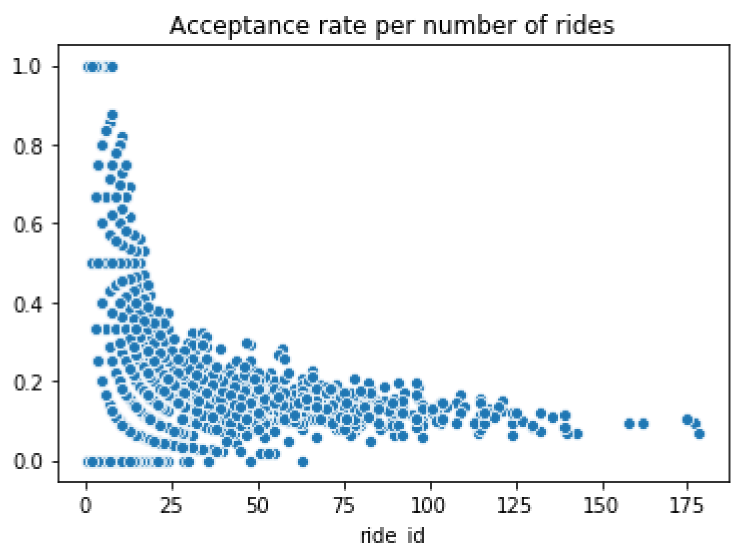
```
nb of rides:  27635
nb of drivers:  51211
-
avg nb of rides per driver 15.495590088198236
min nb of rides per driver 1.0
median nb of rides per driver 9.0
max nb of rides per driver 178.0
-
avg acceptance rate 0.4403658333015167
min acceptance rate 0.0
median acceptance rate 0.46153846153846156
max acceptance rate 1.0
```

In [20]:
```
plt.figure(figsize=(10,3))

plt.subplot(1,2,1)
sns.distplot(group.count()['ride_id'].apply(lambda x:np.log(x)))

plt.title('number of rides per driver' + ' (log distribution)')

plt.subplot(1,2,2)
sns.distplot((group['driver_accepted'].sum()/group['ride_id'].count
()))
plt.title('drivers acceptance rate' + ' (distribution)')

plt.show()
```



In [29]:
```
sns.scatterplot(x=group.count()['ride_id'],y=group['driver_accepted'].
sum()/group['ride_id'].count())
plt.title('Acceptance rate per number of rides')
```

Out[29]: Text(0.5, 1.0, 'Acceptance rate per number of rides')



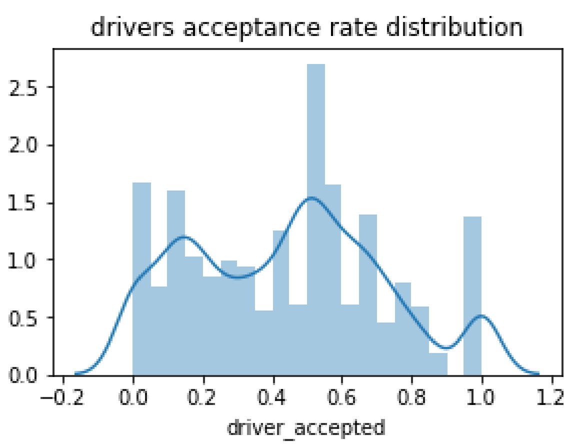
3.1.3 Acceptance rate segmentation

Looking at the distribution of the acceptance rate, we can identify four local maximums showing four segments:

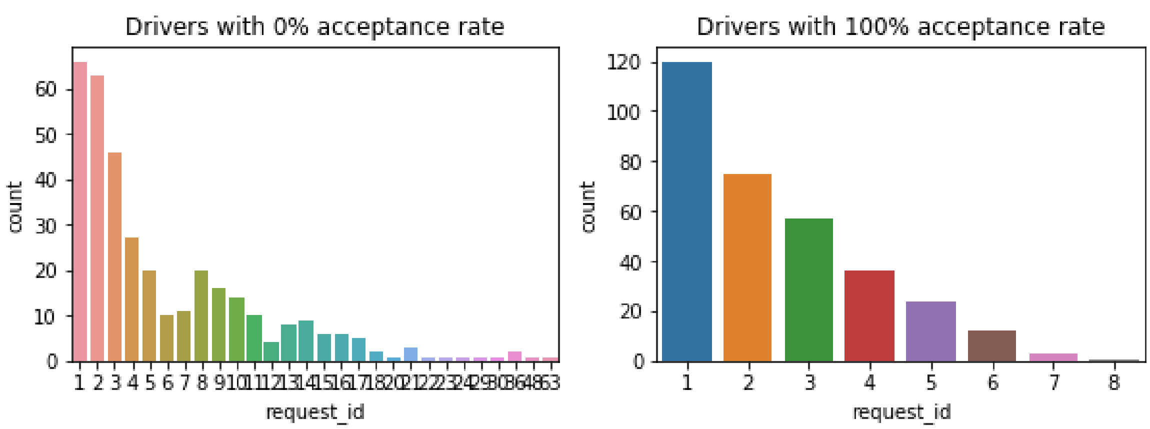
- drivers with an acceptance rate of 0%
- drivers with an acceptance rate of 0-30%
- drivers with an acceptance rate of 30-90%
- drivers with an acceptance rate of 100%

We take a closer look at the drivers with 0% and 100% acceptance rate, to see how much of it we can remove because of drivers taking too few rides to have their behavior realistically measured. By removing the riders with 3 or less rides, we keep around half of the relevant dataset.

In [31]:
```
group = bookings.groupby('driver_id')

plt.figure(figsize=(10,3))
plt.subplot(1,2,2)
sns.distplot((group['driver_accepted'].sum()/group['driver_accepted'].
count()))
plt.title('drivers acceptance rate' + ' distribution')

plt.show()
```

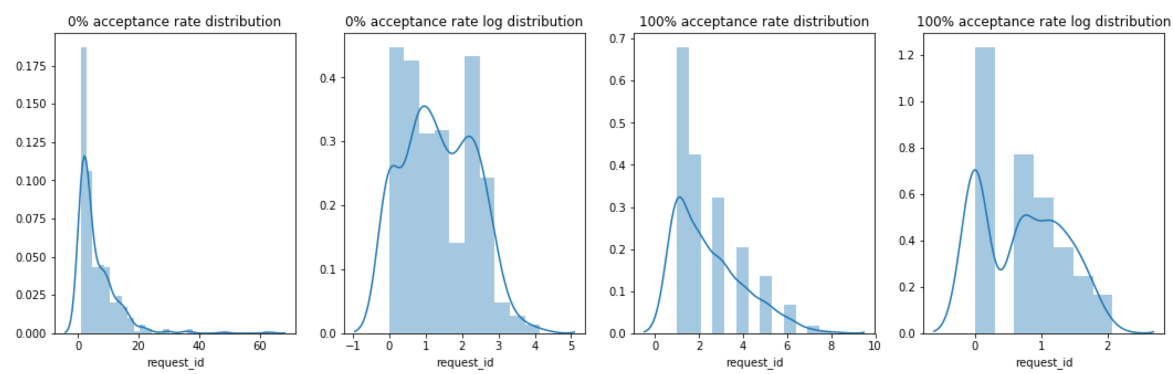


In [33]:
```
plt.figure(figsize=(10,3))

plt.subplot(1,2,1)
zero_rate_count = group.count()[(group.sum()/group.count())['driver_accepted']==0]['request_id']
sns.countplot(zero_rate_count)
plt.title('Drivers with 0% acceptance rate')

plt.subplot(1,2,2)
one_rate_count = group.count()[(group.sum()/group.count())['driver_accepted']==1]['request_id']
sns.countplot(one_rate_count)
plt.title('Drivers with 100% acceptance rate')

plt.show()

print('% of zero-rate drivers kept after removing the ones below 3 requests:',round((zero_rate_count>2).sum()/len(zero_rate_count)*100,1),'%')
print('% of zero-rate drivers kept after removing the ones below 3 requests:',round((one_rate_count>2).sum()/len(one_rate_count)*100,1),'%')
```




```
% of zero-rate drivers kept after removing the ones below 3 requests: 63.8 %
% of zero-rate drivers kept after removing the ones below 3 requests: 40.5 %
```

In [505]:
```
plt.figure(figsize=(18,5))

plt.subplot(1,4,1)
sns.distplot(zero_rate_count)
plt.title('0% acceptance rate distribution')

plt.subplot(1,4,2)
sns.distplot(zero_rate_count.apply(lambda x:np.log(x)))
plt.title('0% acceptance rate log distribution')

plt.subplot(1,4,3)
sns.distplot(one_rate_count)
plt.title('100% acceptance rate distribution')

plt.subplot(1,4,4)
sns.distplot(one_rate_count.apply(lambda x:np.log(x)))
plt.title('100% acceptance rate log distribution')

plt.show()
```



In [45]:
```
# Remove drivers with not enough rides
def remove_rates(drivers):
    return drivers[~drivers['driver_id'].isin(zero_rate_count.index)]

drivers = remove_rates(drivers)
```

3.1.4 Session time and shifts

The dataset is very clean in terms of session time. The drivers connect and disconnect from the app only one time in the day. They don't disconnect when they want to take a break, or when they want to use a potential competitor.

In [47]:
```
drivers.groupby('driver_id').count().sort_values(by='logged_at',ascending=False).describe()
```

Out[47]:

	logged_at	new_state
count	4591.000000	4591.000000
mean	10.999564	10.999564
std	5.604231	5.604231
min	2.000000	2.000000
25%	6.000000	6.000000
50%	12.000000	12.000000
75%	14.000000	14.000000
max	40.000000	40.000000

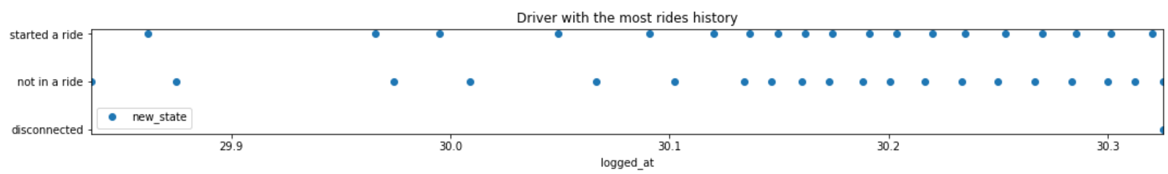
In [48]:
```
# Median driver shift
median_id = len(drivers.groupby('driver_id').count())//2
median_driver = drivers.groupby('driver_id').count().sort_values(by='logged_at',ascending=False).iloc[median_id].name
pd.concat([drivers[drivers['driver_id']==median_driver]['logged_at'].apply(lambda x:date_time(x)),
          drivers[drivers['driver_id']==median_driver]['new_state']],
          axis=1)
```

Out[48]:

	logged_at	new_state
21234	2018-10-29 23:17:13.244222	connected
23954	2018-10-29 23:47:58.999753	began_ride
26272	2018-10-30 00:12:40.613066	ended_ride
31091	2018-10-30 01:11:58.995947	began_ride
32720	2018-10-30 01:31:50.765517	ended_ride
33542	2018-10-30 01:41:58.999751	began_ride
35481	2018-10-30 02:06:48.775279	ended_ride
37724	2018-10-30 02:35:58.999893	began_ride
39383	2018-10-30 02:56:16.851425	ended_ride
41094	2018-10-30 03:17:58.999933	began_ride
42650	2018-10-30 03:37:22.702189	ended_ride
42888	2018-10-30 03:41:58.999945	disconnected

In [49]:
```
# Maximum driver shift
max_id = 0
max_driver = drivers.groupby('driver_id').count().sort_values(by='logg
ed_at',ascending=False).iloc[max_id].name

state = {'disconnected':0,'connected':1,'began_ride':2,'ended_ride':1}
y_labels = ['disconnected','disconnected','not in a ride','started a r
ide']

fig = plt.figure(figsize=(60,6))
ax1 = plt.subplot2grid((3,3),(0,0))
pd.concat([drivers[drivers['driver_id']==max_driver]['logged_at'].appl
y(lambda x:date_time(x).day+date_time(x).hour/24+date_time(x).minute/1
440),
          drivers[drivers['driver_id']==max_driver]['new_state'].apply
(lambda x:state[x])],
          axis=1).plot(x='logged_at',y='new_state',style='o',ax=ax1)
ax1.set_yticklabels(y_labels)
ax1.set_title('Driver with the most rides history')
```

Out[49]: Text(0.5, 1.0, 'Driver with the most rides history')



In [50]:
```
# Drivers states showing there is only one session per driver per day.
drivers[drivers['new_state']=='disconnected'].groupby('driver_id').cou
nt().describe()
```

Out[50]:

	logged_at	new_state
count	4557.0	4557.0
mean	1.0	1.0
std	0.0	0.0
min	1.0	1.0
25%	1.0	1.0
50%	1.0	1.0
75%	1.0	1.0
max	1.0	1.0

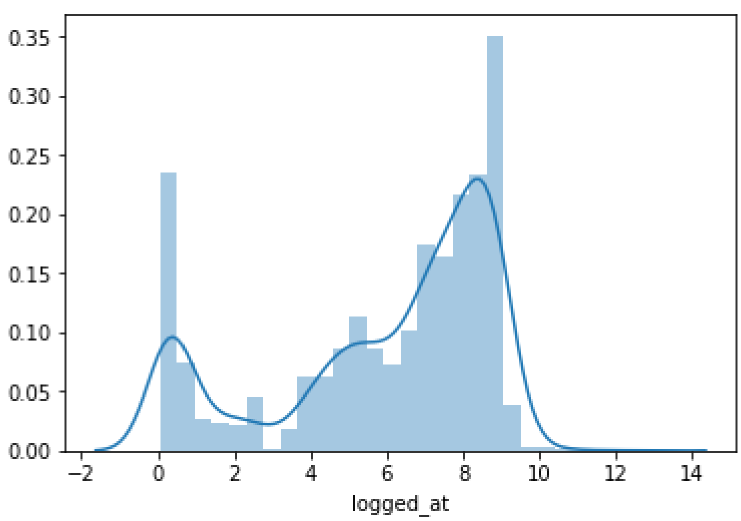
Most drivers hav every short shifts (<1h) or day long shifts (6-9h).

(Looking at the data, one may say that we have many short shifts because our dataset ends before all drivers end their session but it isn't the cause, as those drivers only represent 0.25% of the dataset.)

In [51]:
```
session_time = (drivers.groupby('driver_id')['logged_at'].max().apply(
lambda x:date_time(x))-drivers.groupby('driver_id')['logged_at'].min()
.apply(lambda x:date_time(x))).apply(lambda x:x.total_seconds()/3600)
print(session_time.describe())
sns.distplot(session_time)
```

```
count    4591.000000
mean        5.962959
std         2.908880
min         0.057082
25%         4.503123
50%         7.043604
75%         8.355757
max        12.672522
Name: logged_at, dtype: float64
```

Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1a42b57f28>



In [52]:
```
drivers[drivers['driver_id']=='3DCB309A-3766-4FB8-9FA7-33F983309095'][
'logged_at'].apply(lambda x:date_time(x))
```

Out[52]:
```
51031   2018-10-30 12:36:58.285496
51046   2018-10-30 12:40:23.780926
Name: logged_at, dtype: datetime64[ns]
```

In [53]:
```
unfinished = len(drivers[drivers['driver_id'].isin(drivers[drivers['ne
w_state']=='disconnected']['driver_id'])])/len(drivers)

print('drivers that were in a shift at the end of our dataset: ',round
((1-unfinished)*100,2),'%')
```

```
drivers that were in a shift at the end of our dataset:  0.24 %
```

4 Features Engineering & Exploration around the new-added features

4.1 Used

4.1.1 Booking and driver matching trial for each ride

We join rides and booking to create a request table containing all the rides with each driver match. We drop the requests that didn't matched a driver

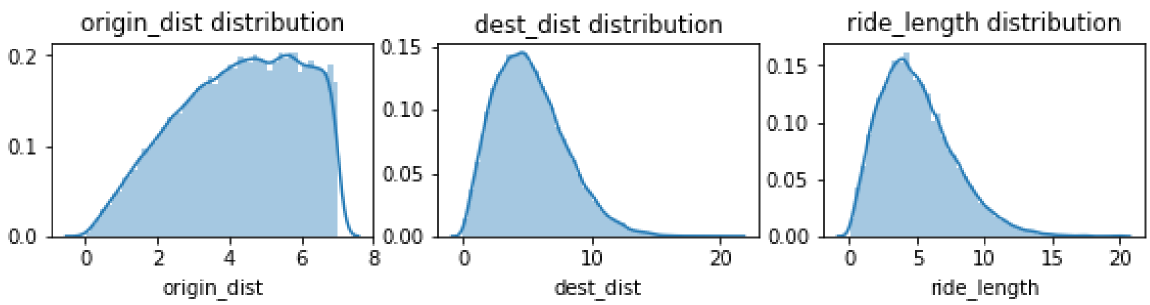
In [70]:
```
# Merge the tables
def request_build(rides,bookings):
    requests = pd.merge(rides,bookings,how='outer',on='ride_id')
    requests = requests.dropna()
    return requests

requests = request_build(rides,bookings)
```

4.1.2 Distance between the driver, rider, destination

We leverage the geolocations of the dataset by computing the different distances, as they are important in the answer of the driver.
Looking at their distribution, we see some of them are skewed normal distributions. We transform them using the log.
We also notice some outliers. As per Chebychev's rule, 3 std. deviations account for 99% of data. Using this approach, we filter out the rest of the data.

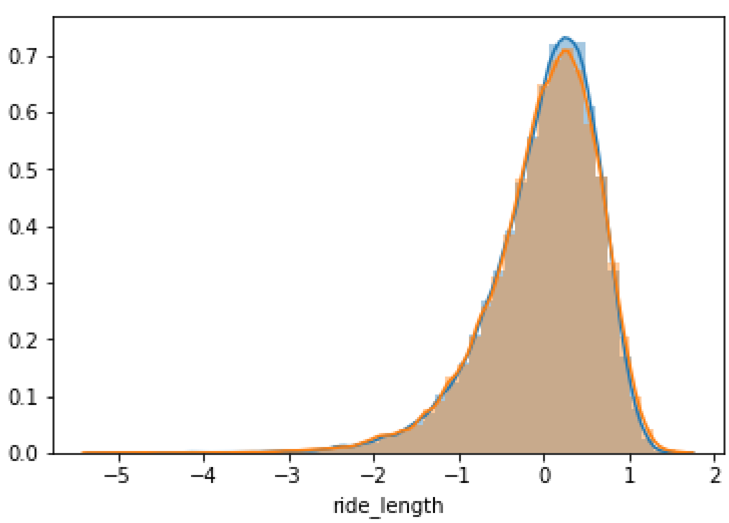
In [18]:
```
# Get one distance
def distances(lat1,lon1,lat2,lon2):
    return distance.distance((lat1,lon1),(lat2,lon2)).km

# Get all distances (iterative solution)
def comp_distances(requests):

    distances = pd.DataFrame(columns = ['ride_id','driver_id','origin_
dist','dest_dist','ride_length'])

    for req in pb(requests.iterrows()):
        ride_id = req[1]['ride_id']
        driver_id = req[1]['driver_id']
        origin_dist = distance.distance((req[1]['origin_lat'],req[1][
'origin_lon']),(req[1]['driver_lat'],req[1]['driver_lon'])).km
        dest_dist = distance.distance((req[1]['destination_lat'],req[1
]['destination_lon']),(req[1]['driver_lat'],req[1]['driver_lon'])).km
        ride_length = distance.distance((req[1]['destination_lat'],req
[1]['destination_lon']),(req[1]['origin_lat'],req[1]['origin_lon'])).k
m
        distances.loc[len(distances)] = [ride_id,driver_id,origin_dist
,dest_dist,ride_length]

    return pd.merge(requests,distances,how='outer',on=['ride_id','driv
er_id'])

# Get all distances (matrix optimized solution)
def comp_distances(requests):
    requests['origin_dist'] = np.vectorize(distances)(requests['origin
_lat'], requests['origin_lon'],requests['driver_lat'],requests['driver
_lon'])
    requests['dest_dist'] = np.vectorize(distances)(requests['destinat
ion_lat'], requests['destination_lon'],requests['driver_lat'],requests
['driver_lon'])
    requests['ride_length'] = np.vectorize(distances)(requests['destin
ation_lat'], requests['destination_lon'],requests['origin_lat'], reque
sts['origin_lon'])
    return requests

requests = comp_distances(requests)
```

In [124]:
```
cat_names=['origin_dist','dest_dist','ride_length']

plt.figure(figsize=(10,4))

for i,name in enumerate(cat_names):
    plt.subplot(2,3,i+1)
    sns.distplot(requests[name])
    plt.title(name + ' distribution')

plt.show()
```



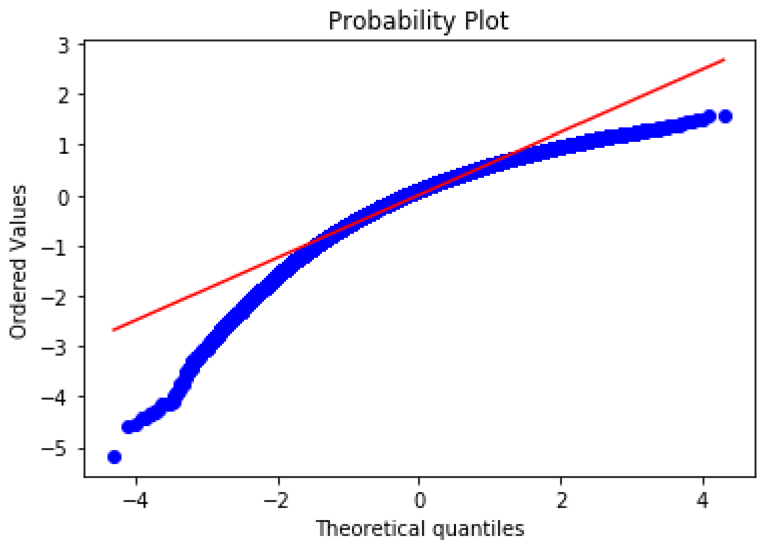
In [125]:
```
# Reduction of the skewness
def right_skew(requests,columns):
    for c in columns:
        dest_mean = requests[c].apply(lambda x:np.log(x)).mean()
        requests[c] = requests[c].apply(lambda x:np.log(x)-dest_mean)
    return requests

requests = right_skew(requests,['dest_dist','ride_length'])

sns.distplot(requests['dest_dist'])
sns.distplot(requests['ride_length'])
```

Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x1a46e9ceb8>

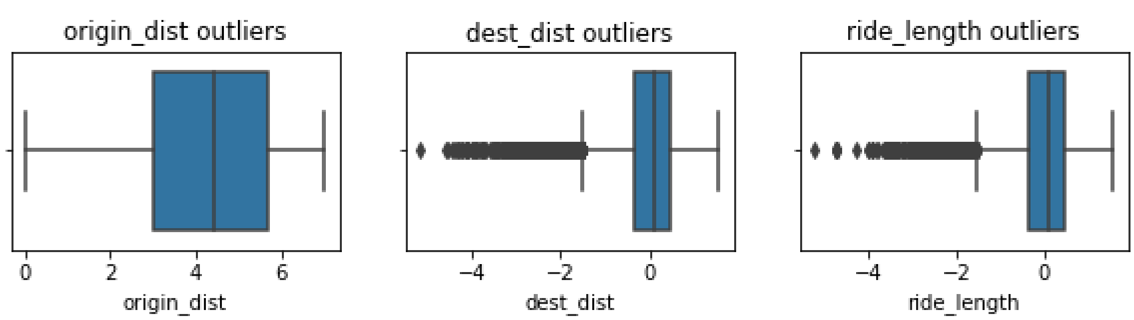


In [126]:
```
fig,axes = plt.subplots(ncols=1,nrows=1)
stats.probplot(requests['dest_dist'], dist='norm', fit=True, plot=axes
)
#stats.probplot(requests['ride_length'], dist='norm', fit=True, plot=a
xes)
```

Out[126]:
```
((array([-4.3008248 , -4.10005959, -3.99078924, ...,  3.99078924,
          4.10005959,  4.3008248 ]),
  array([-5.17086059, -4.57270413, -4.54680669, ...,  1.50381156,
          1.55876386,  1.55899888])),
 (0.6223122802758778, 7.866567257502474e-15, 0.9692866134755399))
```



In [131]:
```
cat_names=['origin_dist','dest_dist','ride_length']

plt.figure(figsize=(10,4))

for i,name in enumerate(cat_names):
    plt.subplot(2,3,i+1)
    sns.boxplot(requests[name])
    plt.title(name + ' outliers')

plt.show()
```



In [137]:
```
def rem_outliers(requests, columns):
    for c in columns:
        q99=np.percentile(requests[c].values,[99])
        requests=requests[requests[c]<q99[0]]
    return requests

requests = rem_outliers(requests,['dest_dist','ride_length'])
```

4.1.3 Distance between the destination and the driver's home

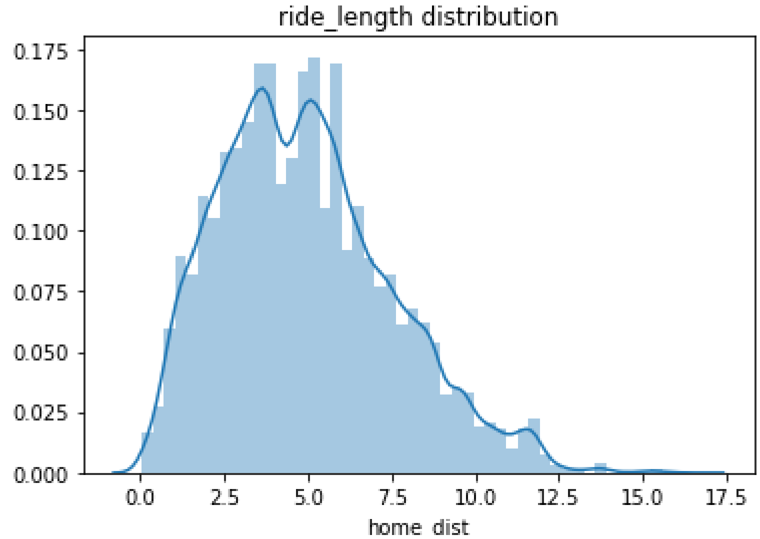
We formulate the hypothesis that the driver's home location is the location he is at at his first request. We want to take a look at if this distance could be important, especially at the end of his shift.
As its distribution is also skewed, we apply a log transformation.

In [46]:
```
# compute home distance
def comp_home_distance(requests):
    home_distances = pd.DataFrame(columns = ['driver_id','home_dist'])

    for req in pb(requests.loc[requests.groupby('driver_id')['logged_a
t'].idxmin()].iterrows()):
        driver_id = req[1]['driver_id']
        home_dist = distance.distance((req[1]['destination_lat'],req[1
]['destination_lon']),(req[1]['driver_lat'],req[1]['driver_lon'])).km
        home_distances.loc[len(home_distances)] = [driver_id,home_dist
]

    return pd.merge(requests,home_distances,how='outer',on=['driver_i
d'])

requests = comp_home_distance(requests)
```

```
| |      #                                         | 1849 Elapsed Tim
e: 0:26:42
```

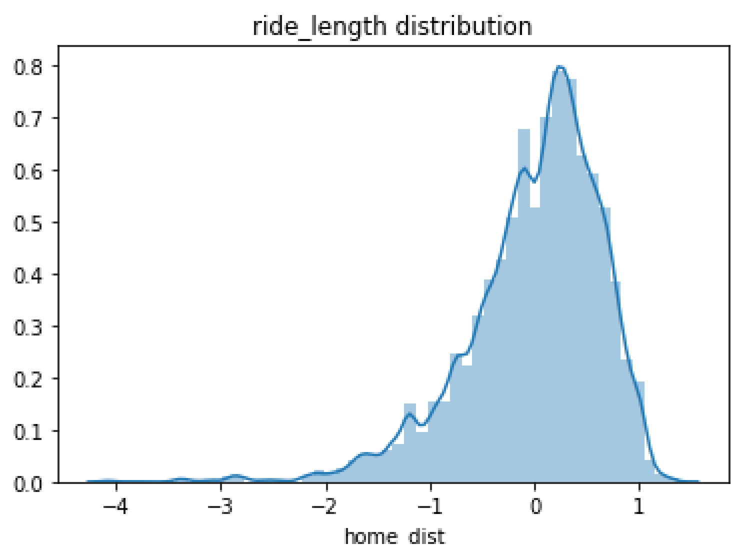
In [138]:
```
sns.distplot(requests['home_dist'])
plt.title(name + ' distribution')
```

Out[138]: Text(0.5, 1.0, 'ride_length distribution')



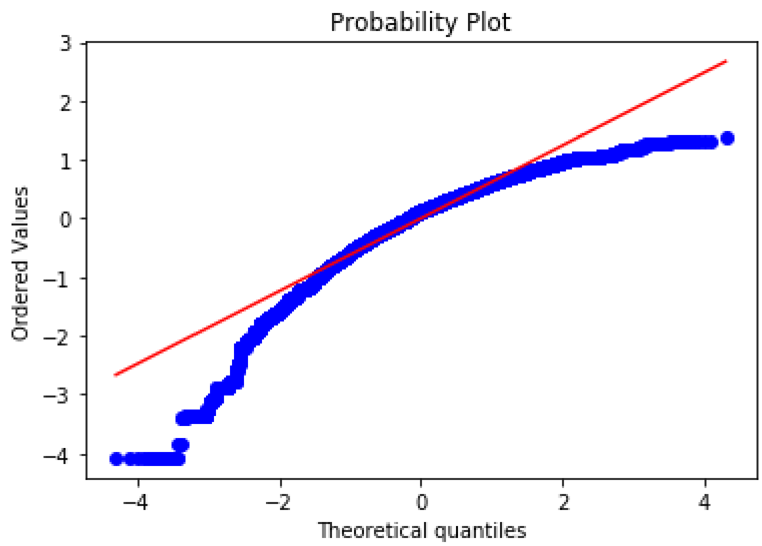
In [139]:
```
requests = right_skew(requests,['home_dist'])

sns.distplot(requests['home_dist'])
plt.title(name + ' distribution')
```

Out[139]: Text(0.5, 1.0, 'ride_length distribution')



In [141]:
```
fig,axes = plt.subplots(ncols=1,nrows=1)
stats.probplot(requests['home_dist'], dist='norm', fit=True, plot=axes
)
#stats.probplot(requests['ride_length'], dist='norm', fit=True, plot=a
xes)
```

Out[141]:
```
((array([-4.29636396, -4.09540027, -3.9860146 , ...,  3.9860146 ,
          4.09540027,  4.29636396]),
  array([-4.07522365, -4.07522365, -4.07522365, ...,  1.32766668,
          1.32766668,  1.37164795])),
 (0.6210263280046395, -1.4731530442102992e-13, 0.9678022830164078))
```



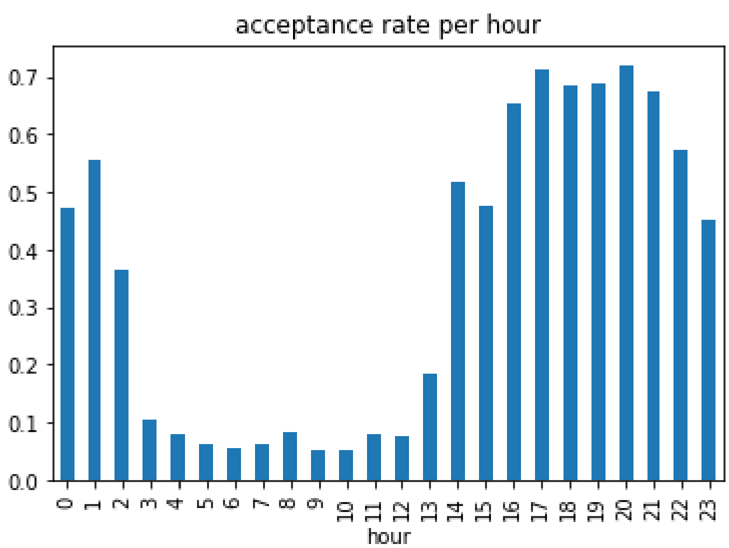
4.1.4 Hours of the day

Drivers may accept less rides at certain times of the day.
Let's add the hour of the day he receives the request as a feature and look at it.
Let's look at the acceptance rate per hour. We can see it is very low from 3am to 1pm.
One reason could be a lack of data at that period making outliers and anomalies stronger but it's not the case. In fact, 3-5 am contain most of the data. And it's not just the bookings, it is also the case for the initial rides requests, so the multiplication of bookings per ride is not the cause.

In [25]:
```
# Get one hour
def booking_hour(timestamp):
    return datetime.datetime.fromtimestamp(timestamp).hour

# Get all hours (matrix optimized solution)
def get_hours(requests):
    requests['hour'] = np.vectorize(booking_hour)(requests['logged_at'
])
    return requests

requests = get_hours(requests)
```

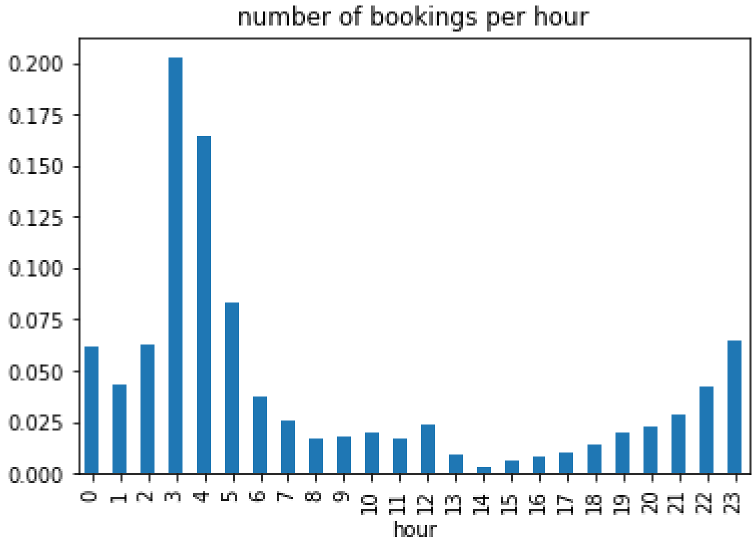
In [53]:
```
(requests.groupby('hour')['driver_accepted'].sum()/requests.groupby('h
our')['driver_accepted'].count()).plot(kind='bar',title='acceptance ra
te per hour')
```

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2712a438>



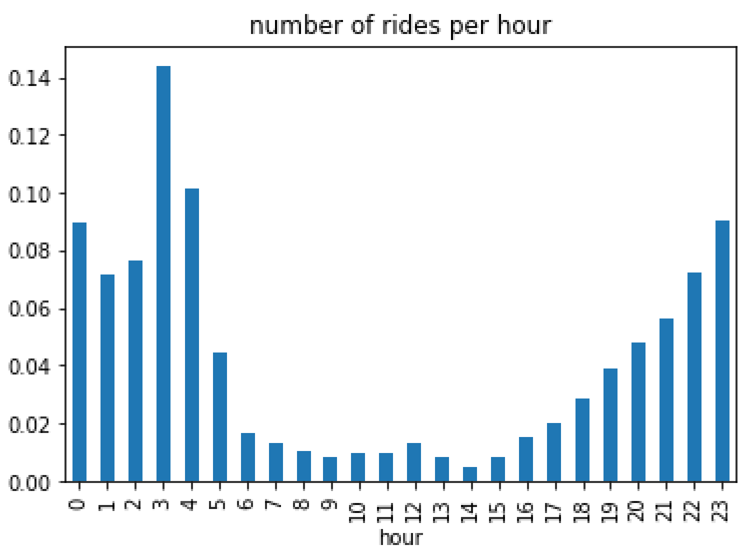
In [55]:
```
(requests.groupby('hour')['driver_accepted'].count()/len(requests)).pl
ot(kind='bar',title='number of bookings per hour')
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1a27304eb8>



In [61]:
```
rides['hour'] = np.vectorize(booking_hour)(rides['created_at'])
(rides.groupby('hour')['ride_id'].count()/len(rides)).plot(kind='bar',
title='number of rides per hour')
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1a29692780>



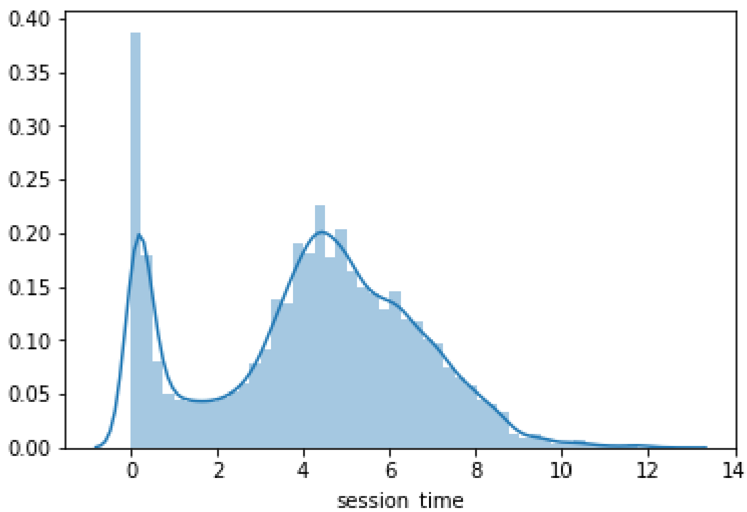
4.1.5 Session time segmentation

We want to see how much time drivers have spent on the app. We notice most of the session time is short (<1h) or between 3-8h with a peak at 4h.

In [142]:
```
# Get all sessions duration up to the booking (iterative solution)
def comp_sessions(requests,drivers):
    sessions = []
    for req in pb(requests.iterrows()):
        current_time = date_time(req[1]['logged_at'])
        start_time = date_time(drivers[drivers['driver_id']==req[1]['d
river_id']]['logged_at'].min())
        session_time = (current_time - start_time).total_seconds()/360
0
        sessions.append(session_time)
    requests['session_time'] = sessions
    return requests

requests = comp_sessions(requests,drivers)
```

In [144]:
```
sns.distplot(requests['session_time'])
```

Out[144]: <matplotlib.axes._subplots.AxesSubplot at 0x1a43c87588>



4.2 Not used

4.2.1 (need API subscription) Destination zipcode

The dataset is extremely large (>50k) and most free geocoding API services are limited (10k calls/m). The code cannot be performed at the desired scale without subscribing to an API service, such as Google Maps API.

We want to segment the different locations of the destination by their area (Paris, N/S/W/E suburbs). We do that to not loose geographic informations from the latitude and longitude (we only kept distances so far). This could be interesting if there are some area in Paris that the driver wants to avoid driving into.

In [10]:
```
# Get one zipcode
def zipcode(lat,lon):
    lat = round(lat,6)
    lon = round(lon,6)
    location = geolocator.reverse(str(lat)+','+str(lon))
    return location.address.split("opolitaine",1)[1][2:7]

# Get all zipcodes (iterative version)
def get_zipcodes(requests):
    zips = []
    for req in requests.iterrows():
        print(req[0])
        lat = req[1]['destination_lat']
        lon = req[1]['destination_lon']
        zips.append(zipcode(lat,lon))
    return zips

# Get all zipcodes (matrix optimized version)
def get_zipcodes(requests):
    return np.vectorize(zipcode)(requests['destination_lat'], requests
['destination_lon'])

# requests['zipcode'] = get_zipcodes(requests)
```

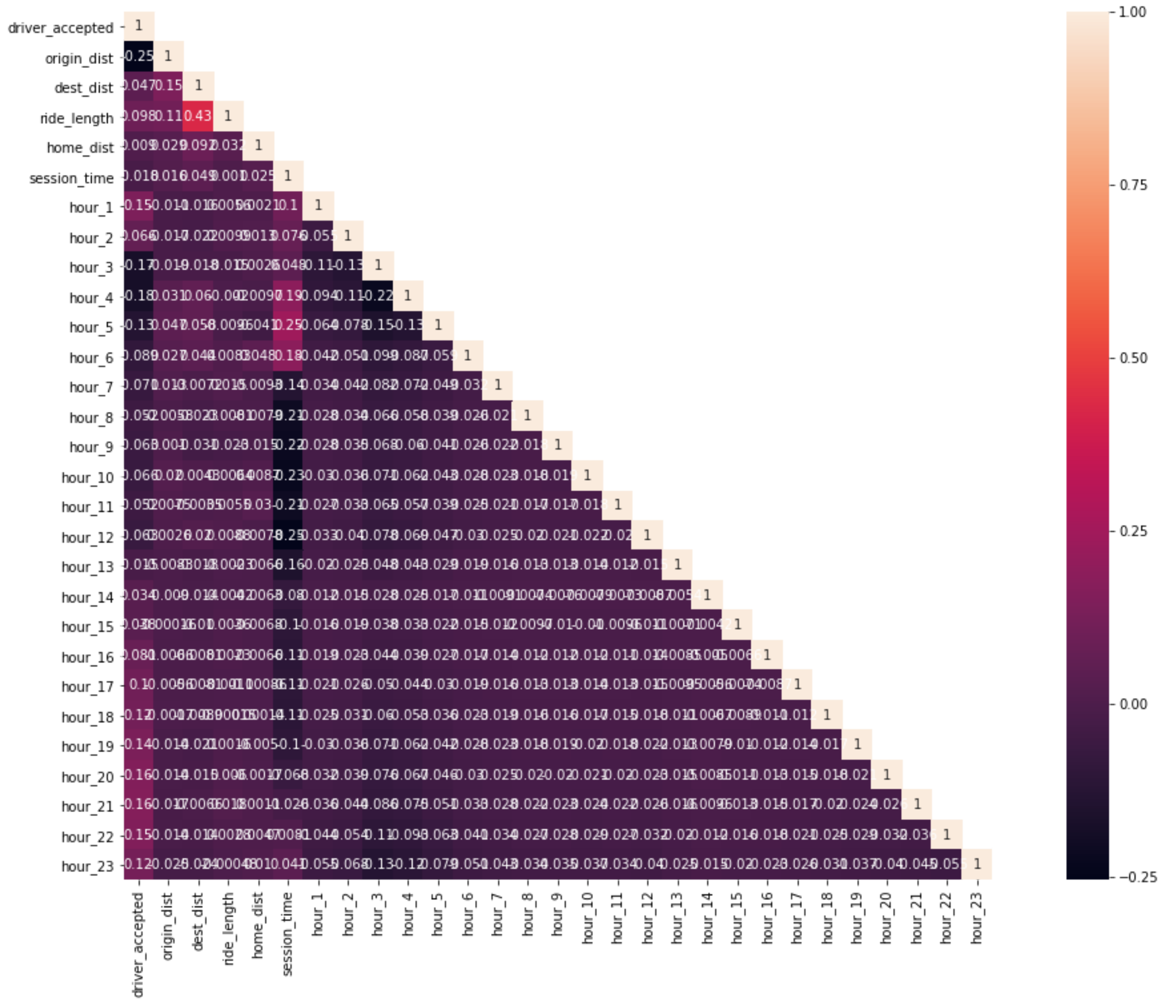
4.3 Final Preprocessing

4.3.1 Features Selection and Columns dropping

Looking at the heatmap, we don't see any high correlation that we should get worried about and we decide to not do any feature selection.

In [49]:
```
cor_mat= requests[:].corr()
mask = np.array(cor_mat)
mask[np.tril_indices_from(mask)] = False
fig=plt.gcf()
fig.set_size_inches(20,12)
sns.heatmap(data=cor_mat,mask=mask,square=True,annot=True,cbar=True)
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x1a448be898>



In [164]:
```
def drop_columns(requests):

    dropped_col = ['created_at','logged_at',
                   'ride_id','request_id',
                   'origin_lat','origin_lon',
                   'destination_lat','destination_lon',
                   'driver_id','driver_lat','driver_lon']

    return requests.drop(dropped_col,axis=1,inplace=False)

requests = drop_columns(requests)
```

4.3.2 Encode features

In [6]:
```
def encode(requests):
    requests['driver_accepted'] = requests['driver_accepted'].apply(la
mbda x:int(x))
    requests = pd.get_dummies(requests,columns=['hour'],drop_first=Tru
e)
    return requests

requests = encode(requests)
```

4.3.3 Data Engineering Pipeline

In [73]:
```
# Merge the tables
def request_build(rides,bookings):
    requests = pd.merge(rides,bookings,how='outer',on='ride_id')
    requests = requests.dropna()
    return requests

# Remove drivers with not enough rides
def remove_rates(requests,drivers):
    return requests[requests['driver_id'].isin(drivers[~drivers['drive
r_id'].isin(zero_rate_count.index)]['driver_id'])]

# Get Distances
### Get one distance
def distances(lat1,lon1,lat2,lon2):
    return distance.distance((lat1,lon1),(lat2,lon2)).km

### Get all distances (matrix optimized solution)
def comp_distances(requests):

    distances = pd.DataFrame(columns = ['ride_id','driver_id','origin_
dist','dest_dist','ride_length'])

    for req in pb(requests.iterrows()):
        ride_id = req[1]['ride_id']
        driver_id = req[1]['driver_id']
        origin_dist = distance.distance((req[1]['origin_lat'],req[1][
'origin_lon']),(req[1]['driver_lat'],req[1]['driver_lon'])).km
        dest_dist = distance.distance((req[1]['destination_lat'],req[1
]['destination_lon']),(req[1]['driver_lat'],req[1]['driver_lon'])).km
        ride_length = distance.distance((req[1]['destination_lat'],req
[1]['destination_lon']),(req[1]['origin_lat'],req[1]['origin_lon'])).k
m
        distances.loc[len(distances)] = [ride_id,driver_id,origin_dist
,dest_dist,ride_length]

    return pd.merge(requests,distances,how='outer',on=['ride_id','driv
er_id'])

# Get Home Distance (iterative solution)
# We could optimize the computation time by using a hashtable of the f
irst logging time of drivers instead of searching for the min() everyt
ime
def comp_home_distance(requests):
    home_distances = pd.DataFrame(columns = ['driver_id','home_dist'])

    for req in pb(requests.loc[requests.groupby('driver_id')['logged_a
t'].idxmin()].iterrows()):
        driver_id = req[1]['driver_id']
        home_dist = distance.distance((req[1]['destination_lat'],req[1
]['destination_lon']),(req[1]['driver_lat'],req[1]['driver_lon'])).km
        home_distances.loc[len(home_distances)] = [driver_id,home_dist
]

    return pd.merge(requests,home_distances,how='outer',on=['driver_i
```

```
d'])

# Reduction of the skewness
def right_skew(requests,columns):
    for c in columns:
        dest_mean = requests[c].apply(lambda x:np.log(x)).mean()
        requests[c] = requests[c].apply(lambda x:np.log(x)-dest_mean)
    return requests

# Get Hours
### Get one hour
def booking_hour(timestamp):
    return datetime.datetime.fromtimestamp(timestamp).hour

### Get all hours (matrix optimized solution)
def get_hours(requests):
    requests['hour'] = np.vectorize(booking_hour)(requests['logged_at'
])
    return requests

# Get sessions
def comp_sessions(requests,drivers):
    sessions = []
    for req in pb(requests.iterrows()):
        current_time = date_time(req[1]['logged_at'])
        start_time = date_time(drivers[drivers['driver_id']==req[1]['d
river_id']]['logged_at'].min())
        session_time = (current_time - start_time).total_seconds()/360
0
        sessions.append(session_time)
    requests['session_time'] = sessions
    return requests

# Encoding
def encode(requests):
    requests['driver_accepted'] = requests['driver_accepted'].apply(la
mbda x:int(x))
    requests = pd.get_dummies(requests,columns=['hour'],drop_first=Tru
e)
    return requests

def pipeline(rides,bookings,drivers):
    print('Pipeline starting...')
    print('request_build...')
    requests = request_build(rides,bookings)
    print('remove_rates...')
    requests = remove_rates(requests,drivers)
    print('com_distances...')
    requests = comp_distances(requests)
    print('comp_home_distance...')
    requests = comp_home_distance(requests)
    print('right_skew...')
    requests = right_skew(requests,['dest_dist','ride_length','home_di
st'])
    print('get_hours...')
```

```
    requests = get_hours(requests)
    print('comp_sessions...')
    requests = comp_sessions(requests,drivers)
    print('encode...')
    requests = encode(requests)
    print('Pipeline finished!')
    return requests

requests = pipeline(rides,bookings,drivers)
```

```
| |         #                                    | 4099 Elapsed Tim
e: 0:02:05

Pipeline starting...
request_build OK

| |                                 #            | 77888 Elapsed Tim
e: 0:09:56

com_distances OK

| |                     #                        | 82664 Elapsed Tim
e: 0:10:08

comp_home_distance OK
get_hours OK

| |                           #                 | 164118 Elapsed Tim
e: 0:15:59

comp_sessions OK
encode OK
Pipeline finished!
```

In [74]:
```
requests.to_csv('data/requests_new.csv',index=False)
```

In [79]:
```
# Drop useless columns
# We keep it separate to have the full dataset saved
# in case we have another data engineering idea later
def drop_columns(requests):

    dropped_col = ['created_at','logged_at',
                'ride_id','request_id',
                'origin_lat','origin_lon',
                'destination_lat','destination_lon',
                'driver_id','driver_lat','driver_lon']

    return requests.drop(dropped_col,axis=1,inplace=False).dropna()

train = drop_columns(requests)
```

In [81]:
```
train.to_csv('data/train.csv',index=False)
```

5 General Model

We first build general model that will be predicting whether the driver will accept the ride or not.
The model will be the same for all drivers. It is important to keep in mind that there are two possible goals regarding such algorithm:

1. Predict the acceptance rate of the driver to find out which type of incentives would be the most relevant to minimize it.
2. Build the a better matching algorithm to improve the experience of drivers and riders

In this model, we will try to answer the goal 1.

5.1 Preparation

5.1.1 Loading and Splitting

In [272]:
```
train = pd.read_csv('data/train.csv')
```

In [273]:
```
def split(requests):
    y = train['driver_accepted']
    X = train.drop(['driver_accepted'],axis=1)
    return X,y

X,y = split(train)
```

In [265]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

5.1.2 Data Resampling

Only 25% of the dataset contains accepted request. It can be a problem for the training of the classifier. We rebalance the training set with the SMOT method to get a 1:1 ratio while keeping the test set unbalanced.

In [266]:
```
print('% of the dataset being accepted requests: ',round(train['driver
_accepted'].mean()*100,2),'%')
```

```
% of the dataset being accepted requests:  25.45 %
```

In [267]:
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))
```

```
Before OverSampling, counts of label '1': 16536
Before OverSampling, counts of label '0': 48638
```

In [268]:
```
def rebalance(X_train,y_train):
    sm = SMOTE(random_state=2)
    X_train_res, y_train_res = sm.fit_sample(X_train, y_train.ravel())
    return X_train_res,y_train_res

X_train,y_train = rebalance(X_train,y_train)
```

In [269]:
```
print("After OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train==0)))
```

```
After OverSampling, counts of label '1': 48638
After OverSampling, counts of label '0': 48638
```

5.1.3 Data Review

Let's take one last peek at our data before we start running the Machine Learning algorithms.

In [89]:
```
X_train.shape
```

Out[89]:
```
(97198, 28)
```

In [90]:
```
X_train[:5]
```

Out[90]:
```
array([[4.67948349, 3.43839953, 8.11344037, 5.56392611, 8.32852367,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [6.89692285, 7.25658507, 0.4788425 , 3.92100042, 4.91582194,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [6.98799773, 8.49284554, 5.06096691, 8.01726243, 6.52975319,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [3.08062373, 2.81356273, 1.85678479, 6.61991846, 7.06060215,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 1.        , 0.        ],
       [6.37878881, 4.560009  , 5.50365439, 5.01828054, 4.90000349,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ]])
```

In [91]:
```
y_train[:5]
```

Out[91]:
```
array([0, 0, 1, 0, 0])
```

In [92]:
```
# Setting a random seed will guarantee we get the same results
# every time we run our training and testing.
random.seed(1)
```

5.2 Modeling

5.2.1 Algorithms

We are in a supervised classification problem, with the goal of predicting if the driver will accept the ride (positive) or not (negative).

We choose to look at the **accuracy** to measure the performance of our classifiers. Type I and Type II errors have an equivalent cost and bad experience to our drivers in our case.

We will be running the following algorithms and choose the best from it.

- Logistic Regression
- KNN
- Naive Bayes
- Linear SVC
- Random Forest
- Gradient Boosted Trees

We will not combine different models together to improve our performances as we want to keep some interpretability.

5.2.1.1 Custom functions

Because there's a great deal of repetitiveness on the code for each, we'll create a custom function to analyse this.

For some algorithms, we have also chosen to run a Random Hyperparameter search, to select the best hyperparameters for a given algorithm.

In [155]:
```
# Function that runs the requested algorithm and returns the accuracy
 metrics
def fit_ml_algo(algo, X_train, y_train, X_test, cv):
    # One Pass
    model = algo.fit(X_train, y_train)
    test_pred = model.predict(X_test)
    if (isinstance(algo, (LogisticRegression,
                          KNeighborsClassifier,
                          GaussianNB,
                          RandomForestClassifier,
                          XGBClassifier))):
        probs = model.predict_proba(X_test)[:,1]
    else:
        probs = "Not Available"
    acc = round(model.score(X_test, y_test) * 100, 2)
    # CV
    train_pred = model_selection.cross_val_predict(algo,
                                                  X_train,
                                                  y_train,
                                                  cv=cv,
                                                  n_jobs = -1)
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100,
2)
    return train_pred, test_pred, acc, acc_cv, probs
```

In [156]:
```
# calculate the fpr and tpr for all thresholds of the classification
def plot_roc_curve(y_test, preds):
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

In [161]:
```
# Function that returns all the visuals needed to evaluate the model
def print_metrics(acc,acc_cv,test_time,y_train,train_pred,y_test,test_
pred,probs):
    print("Accuracy: %s" % acc)
    print("Accuracy CV 10-Fold: %s" % acc_cv)
    print("Running Time: %s" % datetime.timedelta(seconds=test_time))
    print('--\n')
    print('Train dataset :\n',metrics.classification_report(y_train, t
rain_pred))
    print('--\n')
    print('Test dataset :\n',metrics.classification_report(y_test, tes
t_pred))
    plot_roc_curve(y_test, probs)
```

In [158]:
```
# Utility function to report random search best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".form
at(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate
]))
            print("")
```

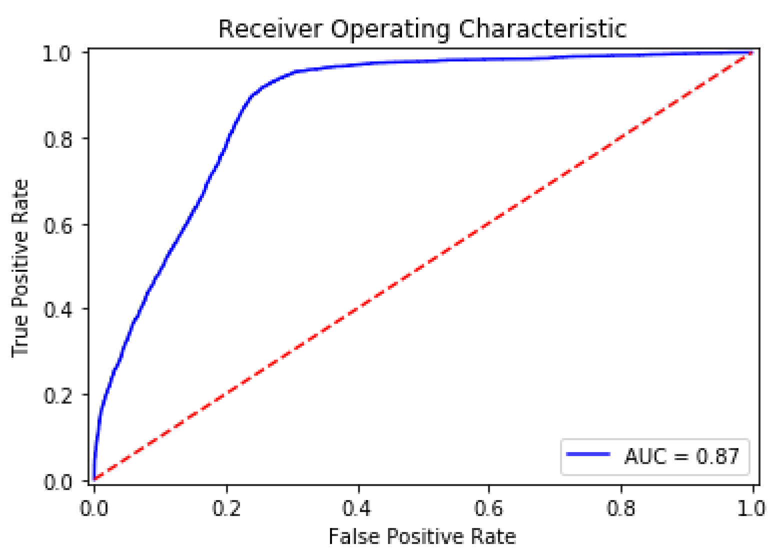
5.2.1.2 Logistic Regression

In [39]:
```
# Random Search for Hyperparameters

# Specify parameters and distributions to sample from
param_dist = {'penalty': ['l2', 'l1'],
              'class_weight': [None, 'balanced'],
              'C': np.logspace(-20, 20, 10000),
              'intercept_scaling': np.logspace(-20, 20, 10000)}
n_iter_search = 10
# Run Randomized Search
lrc = LogisticRegression()
lrc_random_search = RandomizedSearchCV(lrc,
                                       n_jobs=-1,
                                       param_distributions=param_dist,
                                       n_iter = n_iter_search)

start = time.time()
lrc_random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(lrc_random_search.cv_results_)
```

```
RandomizedSearchCV took 93.83 seconds for 10 candidates parameter set
tings.
Model with rank: 1
Mean validation score: 0.811 (std: 0.007)
Parameters: {'penalty': 'l2', 'intercept_scaling': 152.1311757796909
8, 'class_weight': None, 'C': 4.872689936995714e+17}

Model with rank: 2
Mean validation score: 0.811 (std: 0.007)
Parameters: {'penalty': 'l1', 'intercept_scaling': 0.0031750471207081
716, 'class_weight': 'balanced', 'C': 2502.3026392832}

Model with rank: 3
Mean validation score: 0.798 (std: 0.002)
Parameters: {'penalty': 'l2', 'intercept_scaling': 481801654.2811724
5, 'class_weight': 'balanced', 'C': 6.135907273413189e+18}

Model with rank: 3
Mean validation score: 0.798 (std: 0.002)
Parameters: {'penalty': 'l2', 'intercept_scaling': 945143060025263.2,
'class_weight': 'balanced', 'C': 4560011066.004879}
```

In [38]:
```
# Logistic Regression
start_time = time.time()
train_pred_log, test_pred_log, acc_log, acc_cv_log, probs_log = fit_ml
_algo(lrc_random_search.best_estimator_,
                                                                 X_tra
in,
                                                                 y_tra
in,
                                                                 X_tes
t,
                                                                 10)
log_time = (time.time() - start_time)

print_metrics(acc_log,acc_cv_log,log_time,y_train,train_pred_log,y_tes
t,test_pred_log,probs_log)
```

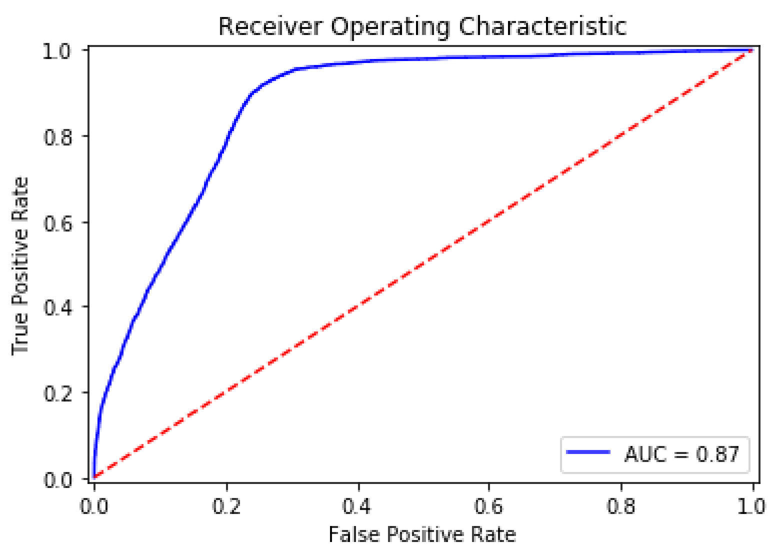
```
Accuracy: 79.66
Accuracy CV 10-Fold: 81.16
Running Time: 0:11:54.317619
--

Train dataset :
              precision    recall  f1-score   support

           0       0.82      0.79      0.81     48527
           1       0.80      0.83      0.81     48527

   micro avg       0.81      0.81      0.81     97054
   macro avg       0.81      0.81      0.81     97054
weighted avg       0.81      0.81      0.81     97054

--

Test dataset :
              precision    recall  f1-score   support

           0       0.93      0.79      0.85     12207
           1       0.57      0.82      0.67      4087

   micro avg       0.80      0.80      0.80     16294
   macro avg       0.75      0.80      0.76     16294
weighted avg       0.84      0.80      0.81     16294
```



5.2.1.3 KNN

In [90]:
```
# k-Nearest Neighbors
start_time = time.time()
train_pred_knn, test_pred_knn, acc_knn, acc_cv_knn, probs_knn = fit_ml
_algo(KNeighborsClassifier(n_neighbors = 3,

n_jobs = -1),

X_train,

y_train,

X_test,

10)
knn_time = (time.time() - start_time)

print_metrics(acc_knn,acc_cv_knn,knn_time,y_train,train_pred_knn,y_tes
t,test_pred_knn,probs_knn)
```

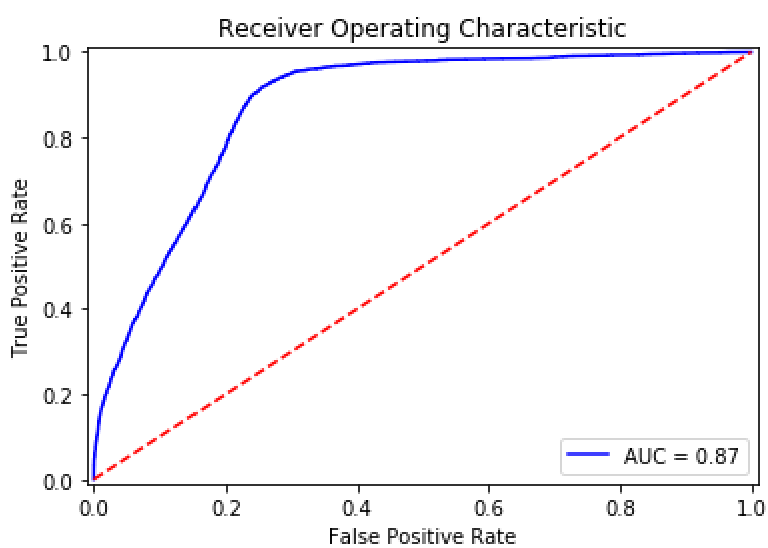
```
Accuracy: 78.92
Accuracy CV 10-Fold: 88.72
Running Time: 0:00:05.436485
--

Train dataset :
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     48527
           1       0.83      0.97      0.90     48527

   micro avg       0.89      0.89      0.89     97054
   macro avg       0.90      0.89      0.89     97054
weighted avg       0.90      0.89      0.89     97054

--

Test dataset :
              precision    recall  f1-score   support

           0       0.91      0.80      0.85     12207
           1       0.56      0.75      0.64      4087

   micro avg       0.79      0.79      0.79     16294
   macro avg       0.73      0.78      0.75     16294
weighted avg       0.82      0.79      0.80     16294
```



5.2.1.4 Gaussian Naive Bayes

In [91]:
```
# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, test_pred_gaussian, acc_gaussian, acc_cv_gaussian
, probs_gau = fit_ml_algo(GaussianNB(),

X_train,

y_train,

X_test,

10)
gaussian_time = (time.time() - start_time)
print_metrics(acc_gaussian,acc_cv_gaussian,gaussian_time,y_train,train
_pred_gaussian,y_test,test_pred_gaussian,probs_gau)
```

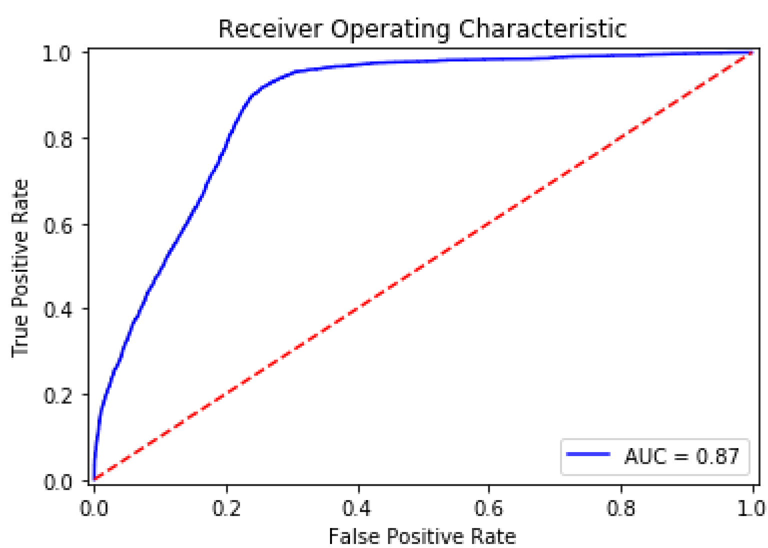
```
Accuracy: 76.84
Accuracy CV 10-Fold: 77.97
Running Time: 0:00:00.481941
--

Train dataset :
              precision    recall  f1-score   support

           0       0.79      0.76      0.78     48527
           1       0.77      0.80      0.78     48527

   micro avg       0.78      0.78      0.78     97054
   macro avg       0.78      0.78      0.78     97054
weighted avg       0.78      0.78      0.78     97054

--

Test dataset :
              precision    recall  f1-score   support

           0       0.92      0.76      0.83     12207
           1       0.53      0.81      0.64      4087

   micro avg       0.77      0.77      0.77     16294
   macro avg       0.72      0.78      0.73     16294
weighted avg       0.82      0.77      0.78     16294
```



5.2.1.5 Linear SVC

In [92]:
```
# Linear SVC
start_time = time.time()
train_pred_svc, test_pred_svc, acc_linear_svc, acc_cv_linear_svc, prob
s_svc = fit_ml_algo(LinearSVC(),

X_train,

y_train,

X_test,

10)
linear_svc_time = (time.time() - start_time)
print_metrics(acc_linear_svc,acc_cv_linear_svc,linear_svc_time,y_train
,train_pred_svc,y_test,test_pred_svc,probs_log)
```

```
Accuracy: 79.05
Accuracy CV 10-Fold: 80.4
Running Time: 0:00:36.106997
--

Train dataset :
              precision    recall  f1-score   support

           0       0.81      0.79      0.80     48527
           1       0.79      0.82      0.81     48527

   micro avg       0.80      0.80      0.80     97054
   macro avg       0.80      0.80      0.80     97054
weighted avg       0.80      0.80      0.80     97054

--

Test dataset :
              precision    recall  f1-score   support

           0       0.93      0.78      0.85     12207
           1       0.56      0.81      0.66      4087

   micro avg       0.79      0.79      0.79     16294
   macro avg       0.74      0.80      0.75     16294
weighted avg       0.83      0.79      0.80     16294
```



5.2.1.6 Random Forest

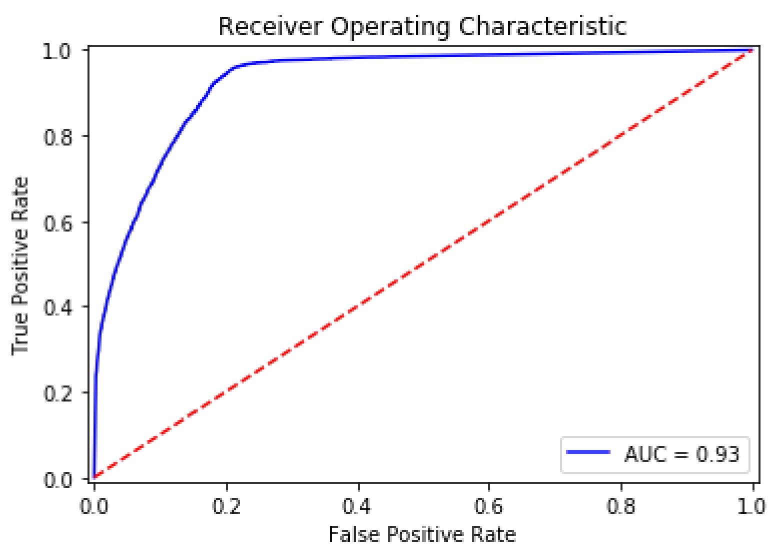
In [159]:

```
# Random Forest Classifier - Random Search for Hyperparameters

# Specify parameters and distributions to sample from
param_dist = {"n_estimators": range(1,50),
              "max_depth": list(range(1,10))+ [None],
              "max_features": sp_randint(1, X.shape[1]),
              "min_samples_split": sp_randint(2, X.shape[1]),
              "min_samples_leaf": sp_randint(1, X.shape[1]),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# Run Randomized Search
n_iter_search = 10
rfc = RandomForestClassifier()
rfc_random_search = RandomizedSearchCV(rfc,
                                       n_jobs = -1,
                                       param_distributions=param_dist,
                                       n_iter=n_iter_search)

start = time.time()
rfc_random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(rfc_random_search.cv_results_)
```

```
RandomizedSearchCV took 50.26 seconds for 10 candidates parameter set
tings.
Model with rank: 1
Mean validation score: 0.892 (std: 0.029)
Parameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth':
None, 'max_features': 24, 'min_samples_leaf': 1, 'min_samples_split':
9, 'n_estimators': 15}

Model with rank: 2
Mean validation score: 0.888 (std: 0.024)
Parameters: {'bootstrap': False, 'criterion': 'gini', 'max_depth': No
ne, 'max_features': 13, 'min_samples_leaf': 10, 'min_samples_split':
18, 'n_estimators': 47}

Model with rank: 3
Mean validation score: 0.881 (std: 0.023)
Parameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth':
None, 'max_features': 22, 'min_samples_leaf': 19, 'min_samples_spli
t': 5, 'n_estimators': 6}
```

In [163]:
```
# Random Forest Classifier
start_time = time.time()
train_pred_rf, test_pred_rf, acc_rf, acc_cv_rf, probs_rf = fit_ml_algo
(rfc_random_search.best_estimator_,
                                                                X_train,
                                                                y_train,
                                                                X_test,
                                                                10)
rf_time = (time.time() - start_time)
print_metrics(acc_rf,acc_cv_rf,rf_time,y_train,train_pred_rf,y_test,te
st_pred_rf,probs_rf)
```

```
Accuracy: 85.34
Accuracy CV 10-Fold: 90.12
Running Time: 0:00:51.088841
--

Train dataset :
              precision    recall  f1-score   support

           0       0.93      0.87      0.90     48649
           1       0.88      0.93      0.90     48649

   micro avg       0.90      0.90      0.90     97298
   macro avg       0.90      0.90      0.90     97298
weighted avg       0.90      0.90      0.90     97298

--

Test dataset :
              precision    recall  f1-score   support

           0       0.93      0.87      0.90     12085
           1       0.68      0.80      0.74      4209

   micro avg       0.85      0.85      0.85     16294
   macro avg       0.81      0.84      0.82     16294
weighted avg       0.86      0.85      0.86     16294
```



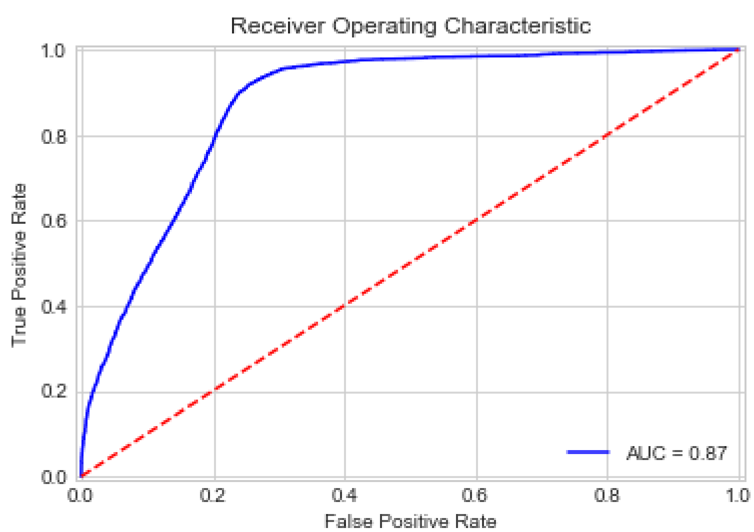
5.2.1.7 Gradient Boosting

In [69]:
```
# Gradient Boosting - Random Search for Hyperparameters

# Specify parameters and distributions to sample from
param_dist = {
        'min_child_weight': range(1,10),
        'gamma': range(1,5),
        'subsample': np.linspace(0.7,0.9,3),
        'colsample_bytree': np.linspace(0.1,1,10),
        'max_depth': range(1,10),
        'learning_rate': np.linspace(0.01,0.1,5)
        }

# Run Randomized Search
n_iter_search = 10
gbt = XGBClassifier()
gbt_random_search = RandomizedSearchCV(gbt,
                                   n_jobs = -1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
gbt_random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(gbt_random_search.cv_results_)
```

```
RandomizedSearchCV took 96.08 seconds for 10 candidates parameter set
tings.
Model with rank: 1
Mean validation score: 0.885 (std: 0.011)
Parameters: {'subsample': 0.8, 'min_child_weight': 3, 'max_depth': 9,
'learning_rate': 0.05500000000000001, 'gamma': 3, 'colsample_bytree':
0.8}

Model with rank: 2
Mean validation score: 0.884 (std: 0.013)
Parameters: {'subsample': 0.9, 'min_child_weight': 8, 'max_depth': 6,
'learning_rate': 0.1, 'gamma': 3, 'colsample_bytree': 0.7000000000000
001}

Model with rank: 3
Mean validation score: 0.882 (std: 0.011)
Parameters: {'subsample': 0.7, 'min_child_weight': 7, 'max_depth': 5,
'learning_rate': 0.1, 'gamma': 3, 'colsample_bytree': 0.8}
```

In [117]:
```
# Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, test_pred_gbt, acc_gbt, acc_cv_gbt, probs_gbt = fit_ml
_algo(gbt_random_search.best_estimator_,
                                                                 X_tra
in,
                                                                 y_tra
in,
                                                                 np.ar
ray(X_test),
                                                                 10)

gbt_time = (time.time() - start_time)
print_metrics(acc_gbt,acc_cv_gbt,gbt_time,y_train,train_pred_gbt,y_tes
t,test_pred_gbt,probs_gbt)
```

```
Accuracy: 84.07
Accuracy CV 10-Fold: 88.85
Running Time: 0:01:43.301514
--

Train dataset :
              precision    recall  f1-score   support

           0       0.96      0.82      0.88     48527
           1       0.84      0.96      0.90     48527

   micro avg       0.89      0.89      0.89     97054
   macro avg       0.90      0.89      0.89     97054
weighted avg       0.90      0.89      0.89     97054

--

Test dataset :
              precision    recall  f1-score   support

           0       0.97      0.81      0.88     12207
           1       0.62      0.93      0.75      4087

   micro avg       0.84      0.84      0.84     16294
   macro avg       0.80      0.87      0.82     16294
weighted avg       0.89      0.84      0.85     16294
```



5.2.2 Ranking Algorithms

Looking at the results of the different algorithms, we see Random Forest is our best performing model in terms of accuracy, with a similar AUC to Gradient Boosting. We will be choose it for our model.

In [98]:
```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression','Random Forest', 'Naive Bay
es','Gradient Boosting Trees'],
    'Score': [
        acc_knn,
        acc_log,
        acc_rf,
        acc_gaussian,
        acc_linear_svc,
        acc_gbt
    ]})
print('Accuracy:')
models.sort_values(by='Score', ascending=False)
```

```
Accuracy:
```

Out[98]:

	Model	Score
2	Random Forest	85.26
5	Gradient Boosting Trees	84.07
1	Logistic Regression	79.66
4	Linear SVC	79.05
0	KNN	78.92
3	Naive Bayes	76.84

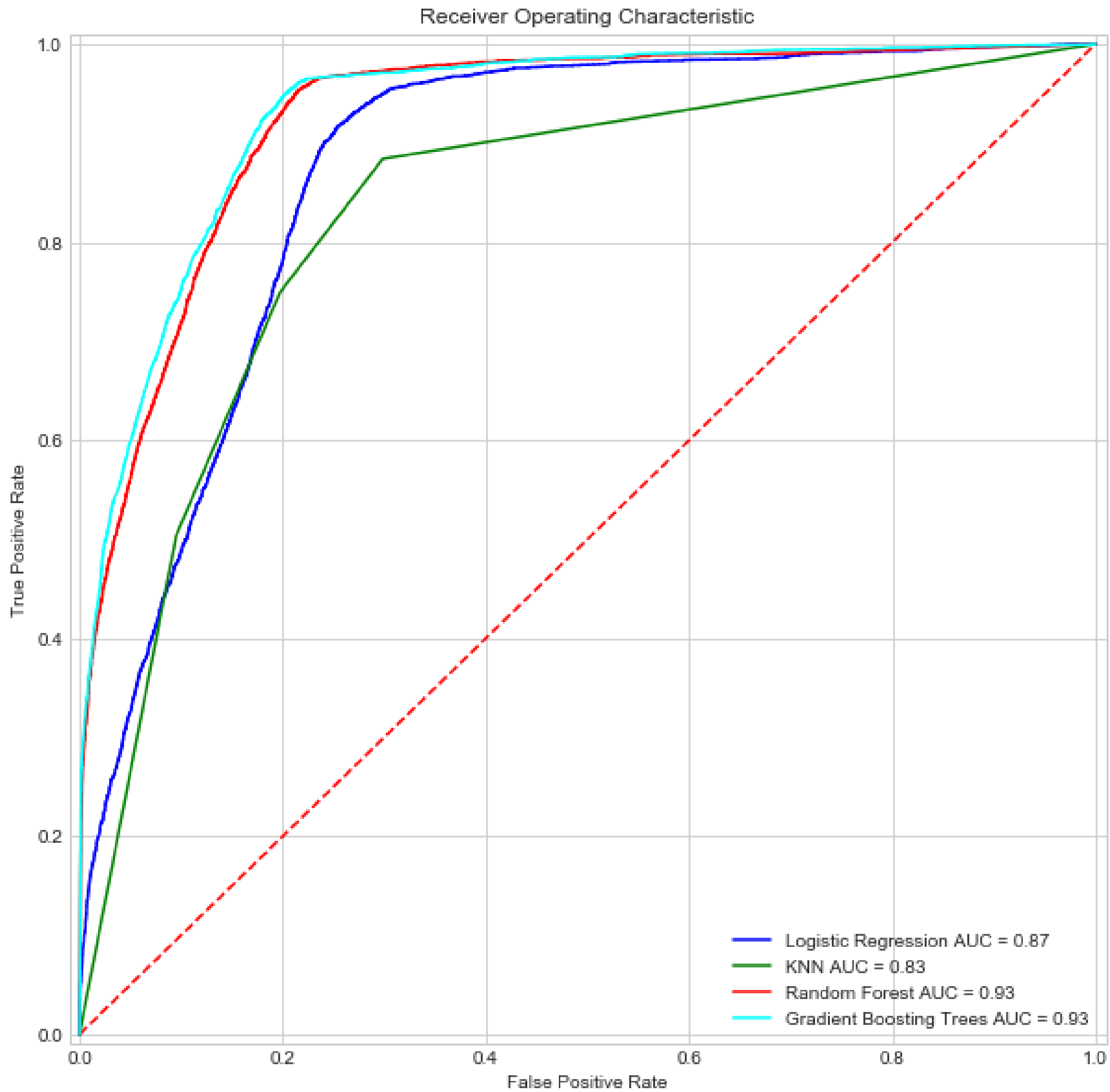
In [99]:
```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression','Random Forest', 'Naive Bay
es','Linear SVC','Gradient Boosting Trees'],
    'Score': [
        acc_cv_knn,
        acc_cv_log,
        acc_cv_rf,
        acc_cv_gaussian,
        acc_cv_linear_svc,
        acc_cv_gbt
    ]})
print('CV Accuracy:')
models.sort_values(by='Score', ascending=False)
```

```
CV Accuracy:
```

Out[99]:

	Model	Score
2	Random Forest	90.17
5	Gradient Boosting Trees	88.85
0	KNN	88.72
1	Logistic Regression	81.16
4	Linear SVC	80.40
3	Naive Bayes	77.97

In [118]:
```
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(10,10))

models = [
    'Logistic Regression',
    'KNN',
    'Random Forest',
    'Gradient Boosting Trees'
]
probs = [
    probs_log,
    probs_knn,
    probs_rf,
    probs_gbt
]
colors = [
    'blue',
    'green',
    'red',
    'cyan',
]

plt.title('Receiver Operating Characteristic')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.01])
plt.ylim([-0.01, 1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

def plot_roc_curves(y_test, prob, model):
    fpr, tpr, threshold = metrics.roc_curve(y_test, prob)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, 'b', label = model + ' AUC = %0.2f' % roc_auc,
color=colors[i])
    plt.legend(loc = 'lower right')

for i, model in list(enumerate(models)):
    plot_roc_curves(y_test, probs[i], models[i])

plt.show()
```



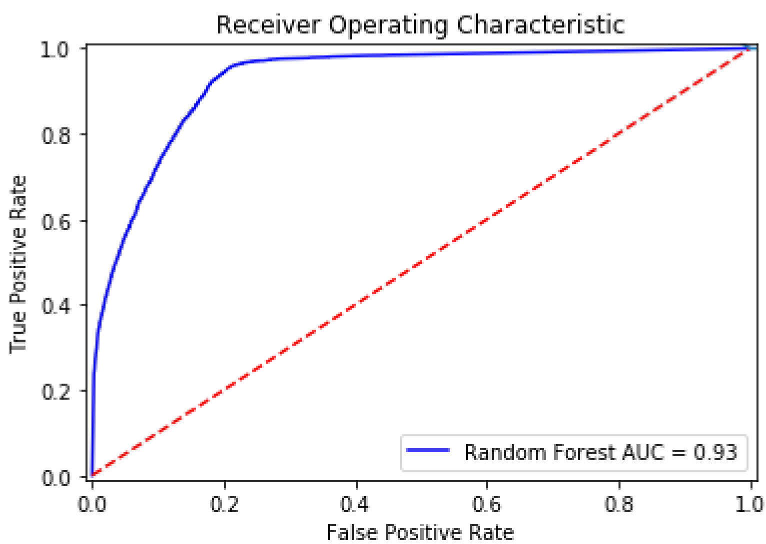
5.3 Finalization

5.3.1 Optimal threshold

Using the ROC curve, we find the optimal threshold for our classifier and use it to build our final model.

In [170]:
```
### Thresholds
fpr, tpr, threshold = metrics.roc_curve(y_test, probs_rf)
roc_auc = metrics.auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'Random Forest' + ' AUC = %0.2f' % roc
_auc, color='blue')
plt.legend(loc = 'lower right')

plt.title('Receiver Operating Characteristic')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.01])
plt.ylim([-0.01, 1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

Out[170]:
```
[<matplotlib.lines.Line2D at 0x1a4b7905c0>]
```



In [192]:
```
opt_idx = np.argmin(np.sqrt(np.square(1-tpr) + np.square(fpr)))
optimal_threshold = threshold[opt_idx]

print('The optimal threshold is',optimal_threshold)
```

```
The optimal threshold is 0.34198653198653195
```

In [223]:
```
class threshold_model():
    def __init__(self,model,threshold):
        self.model = model
        self.threshold = threshold
    def predict(self,X):
        return (self.model.predict_proba(X)[:,1] >= self.threshold).as
type(int)

driver_acceptance = threshold_model(rfc_random_search.best_estimator_,
optimal_threshold)
```

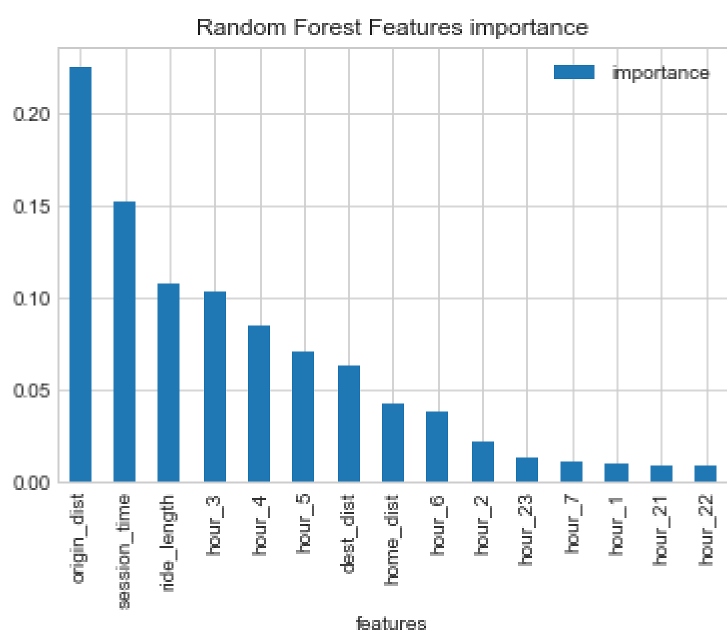
5.3.2 Features Importance

As infered before, we can see the distance between the driver and the pickup location, and ride legth are important criterias. Those are particularily important when it comes to matching the driver with the right rider. Particularily, we notice that the pickup distance is directly impacting the decision: the more the dropoff is far away from the driver, the less the driver will be inclined to accept the ride. We had already a sense of it when vizualizing the heatmap for features selection and it is remarkable as it is a constant pattern that is not influenced by any other variable, as opposed to the ride length or dropoff distance.

We also notice that the session time and the 3-5 am hour window have a big impact on weither the driver will accept the ride or not. Those are particularily important when it comes to find on what criteria should we incentivize the drivers.

In [121]:
```
forest_viz = pd.DataFrame()
forest_viz['features'] = X.columns
forest_viz['importance'] = rfc_random_search.best_estimator_.feature_i
mportances_
forest_viz = forest_viz.sort_values(by=['importance'],ascending=False)
[:15]
forest_viz = forest_viz.set_index('features')
forest_viz.plot(kind='bar',title='Random Forest Features importance')
```

Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x1a149f8080>



In [278]:
```
features = ['origin_dist','ride_length','dest_dist','home_dist']

for feat in features:
    test1 = X.copy()
    test2 = X.copy()
    test1[feat] = 0.000001
    test2[feat] = 10000000
    print('Acceptance rate for a very low', feat,driver_acceptance.pre
dict(test1).sum()/len(driver_acceptance.predict(test1)))
    print('Acceptance rate for a very high', feat,driver_acceptance.pr
edict(test2).sum()/len(driver_acceptance.predict(test2)))
```

```
Acceptance rate for a very low origin_dist 0.9975695978789217
Acceptance rate for a very high origin_dist 0.2783424166543919
Acceptance rate for a very low ride_length 0.2720700152207002
Acceptance rate for a very high ride_length 0.448286443757058
Acceptance rate for a very low dest_dist 0.33283006824765554
Acceptance rate for a very high dest_dist 0.3852678352236461
Acceptance rate for a very low home_dist 0.3515122502086709
Acceptance rate for a very high home_dist 0.3461359061226494
```

5.3.3 Model improvement ideas

There are many ways we can improve this baseline model:

- Reduce the number of features, by grouping the hours together (ex: turn the 23 features to 4 by using this grouping: 11PM-02AM,3PM-5PM,6AM-4PM,4PM-10PM...)
- Transform the distances such as pickup distance to a normal distribution using more advanced transformation functions
- Better fine-tune our model by plotting the AUC function of each hyperparameter, and identify the best choice by looking at the "elbow" on the curve.

5.3.4 Actions

Build an incentive system
We can build an incentive system that will increase the revenues of the drivers for defined criterias (ex: Increase their revenue per mile by 5% by decreasing Heetch's commission rate). If we have multiple months of data, we can model what would be the costs of such incentives and decide what should be the optimal incentive.

The criteria would be:

- Drivers taking riders far from their location. If there are no drivers around him, matching him with a driver will be very difficult without such incentive.
- Session time getting longer than usually (the 'usually' criteria can be defined as the individual monthly average daily session, for example.)
- Drivers accepting rides from 3 to 5 am

With this incentive system, we raise the acceptance rate of the drivers, thus enhancing the experience of the riders (by decreasing their waiting time) and improving the stickiness of Heetch for the drivers.

Build a better general matching algorithm
We can use the model to build a matching algorithm giving drivers the rides that has the most chances of getting approved by them.
Warning: For such matching system, we should repeat our modeling approach, but it is important to remove the features that have relevance for a general model but not for individual matching:

- Hours are not relevant for a matching algorithm: All the rides that we can match a driver at a specific time have the same hour features. If we include it, drivers will have less chances getting matched for ALL of the available ride demands and it will not help us identify the best ride for him.
- Session time is not relevant: Again, include how much time the driver has spent connected would impact all the available ride demands and will not help us choose between them.

This matching algorithm would work in a simple way: every time a rider emits a ride request, we scan the 5-10 closest drivers (not more so that we keep the rider's ETA and experience optimal) and match the driver with the highest chances of approval.

6 Personnalized Model

We can build a personnalized matching model that will be learning from the driver's behavior on a previous shift so that we take individual preferences into account for the matching prediction we couldn't leverage before such as:

- the specific times during his shift when the driver likes to take a break and refuse all rides
- the specific shifts length (session length) the driver pays attention to before getting rides (ex: get a ride close to a restaurant when 12PM approaches, or a ride near his home when he finished 8h of work...)

In this regard, we will be running a Deep Learning algorithm with a memory-based neural network architecture. The neural network will keep in memory the behavior of a driver for all the requests of his previous shift so that we can accurately predict how he will be reacting to the requests of the current one.

6.1 Preparation

6.1.1 Reshaping

We reshape the data into a 3D_array (grouped by the driver_id). This will enable us to input to the LSTM the sequence of rides of each driver.

In [13]:
```
requests = pd.read_csv('data/requests_new.csv')
```

In [23]:
```
# Drop all columns like before except the driver_id, used to reshape t
he data
def person_drop_columns(requests):

    dropped_col = ['created_at','logged_at',
                   'ride_id','request_id',
                   'origin_lat','origin_lon',
                   'destination_lat','destination_lon',
                   'driver_lat','driver_lon']

    return requests.drop(dropped_col,axis=1,inplace=False)

# Reshape the data into X = (nb_samples,nb_timesteps,nb_features) and
 y=(nb_samples,nb_timesteps,driver_response)
def reshape_3D(requests):
    driver_len = len(requests.groupby('driver_id'))
    max_nb_rides = requests.groupby('driver_id')['hour_1'].count().max
()
    nb_columns = len(requests.columns)-1

    to_reshape = requests.sort_values(by=['driver_id','session_time'])
.values
    reshaped = np.zeros((driver_len,max_nb_rides,nb_columns))

    current_driver_id = to_reshape[0][0]
    current_driver = 0
    current_ride = 0
    nb_rides = 0

    for i in to_reshape:
        driver_id = i[0]
        features = i[1:]
        if driver_id == current_driver_id:
            reshaped[current_driver][current_ride] = features
            current_ride += 1
        else:
            current_driver += 1
            current_ride = 0
            reshaped[current_driver][current_ride] = features
            current_driver_id = driver_id

    return reshaped

# Final pipeline
def person_pipeline(rides,bookings,drivers):
    requests = pipeline(rides,bookings,drivers)
    requests = person_drop_columns(requests)
    train = reshape_3D(requests)
    return train

train = person_pipeline(rides,bookings,drivers)
```

In [24]:
```
train.to_csv('data/train_personnalized.csv',index=False)
```

6.1.2 Splitting

In [25]:
```
train = pd.read_csv('data/train_personnalized.csv')
```

In [16]:
```
# Split
X = train[:,:,1:]
y = train[:,:,0]

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

6.2 Modeling

We chose a very basic architecture and GRU over LSTMs for performance issues. The program being run on a local computer and we are confronted with a major problem: the network takes too much time on local (>1h) to train so it is difficult to tune it in order to get an acceptable accuracy, and not get stuck into a bad local minima like we are in the last iteration. Lack of time is disabling me from presenting a decent final model but I hope I am able to show the general idea of what is possible in terms of personnalized model with Deep Learning.
With cloud servers and more time, we would have been able to fine-tune the current network by:

- initialize well
- overfit one batch
- complexify (number of units, stack GRUs, Dense, Conv1D layers...)
- add more data (data augmentation)
- regularize (weight decay, early stopping)
- optimize hyperparameter
- ensemble models

In [22]:
```
# LSTM-based model

nb_samples = X_train.shape[0]
nb_timesteps = X_train.shape[1]
nb_features = X_train.shape[2]

b_size = 32
ep = 10

# design network
model = Sequential()
model.add(GRU(nb_timesteps,
              input_shape = X_train.shape[1:],
              dropout = 0.2,
              recurrent_dropout = 0.2
              ))
#model.add(Dense(64,activation='softmax'))
#Dense(1,activation='softmax')
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy'])

print(model.summary())

# fit network
history = model.fit(X_train, y_train, epochs=ep, validation_data=(X_te
st, y_test), shuffle=False)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensor
flow/python/framework/op_def_library.py:263: colocate_with (from tens
orflow.python.framework.ops) is deprecated and will be removed in a f
uture version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/keras/
backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.
python.ops.nn_ops) with keep_prob is deprecated and will be removed i
n a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate
= 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
gru_1 (GRU)                  (None, 355)               408960    
=================================================================
Total params: 408,960
Trainable params: 408,960
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensor
flow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.op
s.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 3809 samples, validate on 953 samples
Epoch 1/10
3809/3809 [==============================] - 304s 80ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 2/10
3809/3809 [==============================] - 320s 84ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 3/10
3809/3809 [==============================] - 315s 83ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 4/10
3809/3809 [==============================] - 341s 89ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 5/10
3809/3809 [==============================] - 328s 86ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 6/10
3809/3809 [==============================] - 306s 80ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 7/10
3809/3809 [==============================] - 315s 83ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 8/10
3809/3809 [==============================] - 326s 85ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 9/10
```

```
3809/3809 [==============================] - 331s 87ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
Epoch 10/10
3809/3809 [==============================] - 317s 83ms/step - loss: n
an - acc: 0.6367 - val_loss: nan - val_acc: 0.6632
```

6.3 Notes on the model

Modeling the behaviors of the drivers individually is a very difficult task. For such problem to be really answered, we would need a dataset containing multiple days (and shifts) per drivers in order to evaluate our model by running it on a driver. We cannot do it with the current dataset because after charging the memory cells of our network with a shift from the driver, we don't have another day to evaluate the performance of the charged model.
What is possible is to use the model with previous timesteps of less than a shift but we are confronted to similar problems of lack of data:

- If we use too many previous timesteps, we end with a very small number of drivers (most of them have a low number of requests)
- If we use a small number of timesteps, we end with a very badly performing algorithm as we don't have enough past requests to understand the driver

Also, we end up loosing the advantage of such individual algorithm and end up with a general algorithm:

- We remove the opportunity to leverage the session length of drivers as the neural network won't have a full shift but only a part
- We remove the opportunity to leverage the seasonnality of the driver (when he likes to take breaks, etc)

That said, if tuned well, such algorithm would be promising to improve Heetch's matching system as it would use a general model (the neural network trained on all the drivers) and adapt it to the specific driver (by charging the memory cells of the GRU/LSTMs before predicting the behavior with given features)