6. Data Analytics Practicum: NYC Taxi data

이번 시간에는 뉴욕 Manhattan 지역에서 Taxi 운행 기록으로부터 흥미로운 예측 문제를 정의하고, 이를 순차적으로 해결, 발전시켜가는 작업을 해보겠습니다.

먼저 수업 자료가 sharepoint로부터 데이터 파일을 받아서, access 가능한 Google Drive 공간에 위치시킵니다. 오늘 사용할 데이터는 총 2개의 csv 파일로 구성되어 있습니다:

- trip_data_1.csv
- trip_fare_1.csv

필요한 라이브러리들을 불러들이고, Google Drive와 연결하여 데이터 파일들을 불러올 준비를 합니다.

```
import numpy as np
import operator

# Mount your Google drive
from google.colab import drive
drive.mount('/content/drive')


```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_rem
```

이 시간에는 pandas라는 라이브러리 패키지를 사용하여 데이터를 사용하고자 합니다. Pandas가 제공하는 DataFrame 객체는 데이터분석 작업에 편리한 다양한 기능들을 제공합니다. 다음의 코드에서 등장하는 `read_csv()`는 csv파일을 손쉽게 불러오고, 관리할 수 있게 해줍니다.

```
import pandas as pd

N = 8e6
data = pd.read_csv('drive/My Drive/2019-Summer/camp_ML/data/trip_data_1.csv', nrows=N)
```

생성된 data 변수는 바로 DataFrame 객체입니다. DataFrame 형태로 저장된 데이터는 Notebook에서 바로 테이블 형태로 출력해볼 수 있습니다. 또한 DataFrame이 제공하는 `join()` 함수를 이용하면, 두 개의 csv파일로부터 identifier가 같은 instances끼리 합치는 작업도 가능합니다.

```
fare_data = pd.read_csv('drive/My Drive/2019-Summer/camp_ML/data/trip_fare_1.csv', nrows=N)
fare_cols = [u' payment_type', u' fare_amount', u' surcharge', u' mta_tax', u' tip_amount',
             u' tolls_amount', u' total_amount']
data = data.join(fare_data[fare_cols])
del fare_data
data[:10]
print(data.shape)
```

```
(49999, 21)
```

```
data.loc[:5, data.columns[:5]]
```

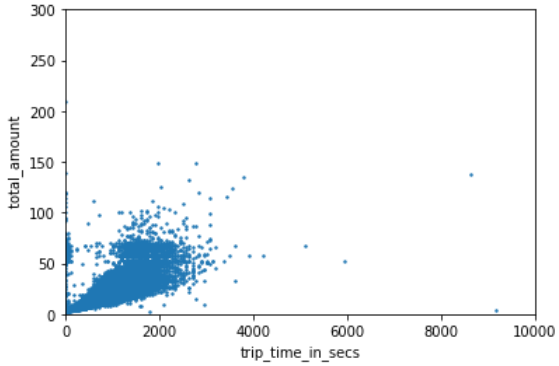
	medallion	hack_license	vendor_id	rate_code	store_and_fwd_flag
0	89D227B655E5C82AECF13C3F540D4CF4	BA96DE419E711691B9445D6A6307C170	CMT	1	N
1	0BD7C8F5BA12B88E0B67BED28BEA73D8	9FD8F69F0804BDB5549F40E9DA1BE472	CMT	1	N
2	0BD7C8F5BA12B88E0B67BED28BEA73D8	9FD8F69F0804BDB5549F40E9DA1BE472	CMT	1	N
3	DFD2202EE08F7A8DC9A57B02ACB81FE2	51EE87E3205C985EF8431D850C786310	CMT	1	N
4	DFD2202EE08F7A8DC9A57B02ACB81FE2	51EE87E3205C985EF8431D850C786310	CMT	1	N
5	20D9ECB2CA0767CF7A01564DF2844A3E	598CCE5B9C1918568DEE71F43CF26CD2	CMT	1	N

Visual data analysis Using Matplotlib

이제 데이터를 이해해보는 첫 단계 입니다. matplotlib을 사용하여 데이터를 여러 각도에서 시각화하며, 변수들 간의 상관 관계를 파악해봅니다. 먼저 trip_time_in_secs(이동 소요 시간)과 total_amount (이용 총 요금) 간의 관계입니다.

```
%pylab inline

data.plot(x="trip_time_in_secs", y=" total_amount", kind="scatter", s=2)
xlim(0,1e4)
ylim(0,300)
```

```
Populating the interactive namespace from numpy and matplotlib
(0, 300)
```



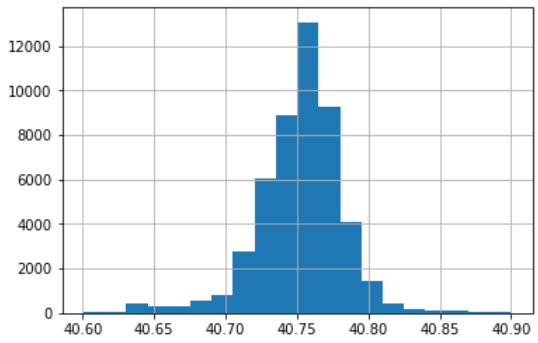
위의 scatter plot은 몇몇 outlier들을 보여줍니다 (이동 시간이나 요금이 지나치게 큰 경우). 이러한 특이한 instances를 제거하여 앞으로의 분석이 좀더 안정적으로 이루어질 수 있게 합니다.

```
ind = where(logical_and(data.trip_time_in_secs < 500, data[' total_amount'] > 30))[0]
data = data.drop(ind)
data.shape
```

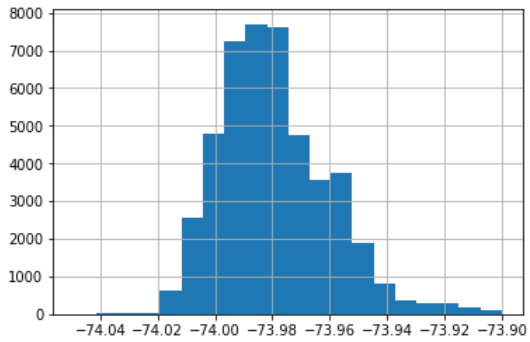
```
(49833, 21)
```

이제 outliers를 제거한 데이터셋으로부터, 히스토그램(histogram)을 작성하여 하차지점의 지역적 (in terms of latitudes) 분포를 살펴봅니다.

```
data[logical_and(data.dropoff_latitude > 40.6,data.dropoff_latitude < 40.9)].dropoff_latitude.hist(bins=20);
```

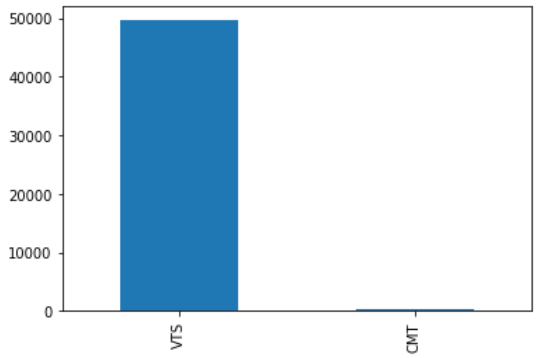



```
data[logical_and(data.dropoff_longitude > -74.05,data.dropoff_longitude < -73.9)].dropoff_longitude.hist(bins=20);
```



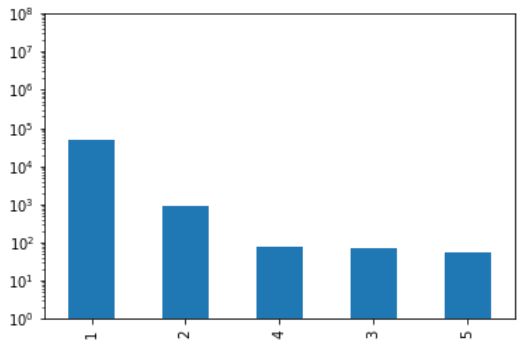
데이터에 포함되어 있는 택시 회사의 분포도 살펴봅니다.

```
data.vendor_id.value_counts().plot(kind="bar");
```



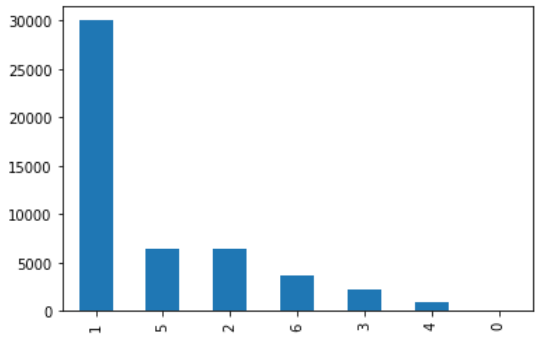
요금 rate (지역 및 시간에 따라 1~5 구간의 다른 rate가 적용 됨) 에 따른 데이터의 분포를 살펴 봅니다.

```
data.rate_code.value_counts().plot(kind="bar", logy=True, ylim=(1,1e8));
```



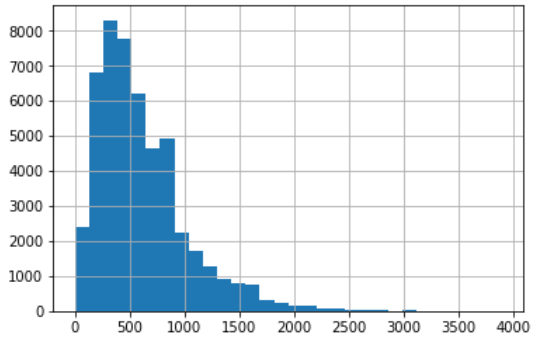
히스토그램을 통해 몇 명의 승객이 동시에 탑승했는지 여부도 살펴봅니다.

```
data.passenger_count.value_counts().plot(kind="bar");
```



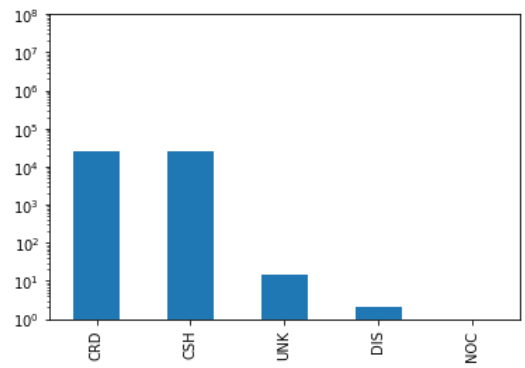
이번엔 이동에 소요된 시간들의 분포를 살펴봅니다.

```
data.trip_time_in_secs[data.trip_time_in_secs < 4000].hist(bins=30);
```



마지막으로 지불 방법을 살펴봅니다. CRD = 크레딧카드로 지불, CSH = 현금으로 지불, UNK, DIS, NOD = 기타 비현금성 매체를 이용하여 지불 (교통카드와 유사한 형태)

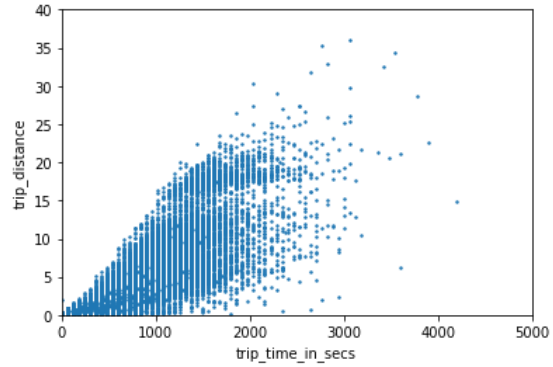
```
data[' payment_type'].value_counts().plot(kind="bar", logy=True, ylim=(1,1e8));
```



스캐터플랏(scatterplot)을 통해 이동 거리와 이동 시간 간의 관계도 그려보겠습니다.

```
data.plot(x="trip_time_in_secs", y="trip_distance", kind="scatter", s=2)
xlim(0,5000)
ylim(0,40)
```

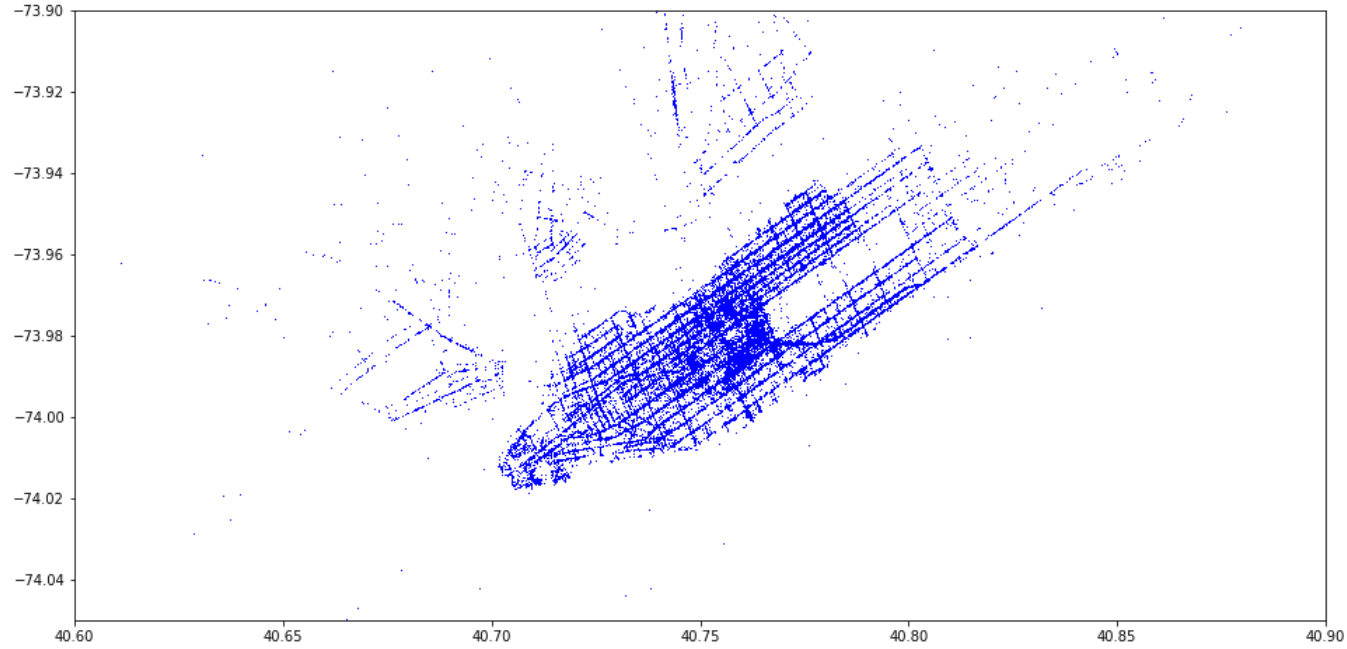
```
(0, 40)
```



이번엔 탑승 지점 좌표를 화면에 도식해보겠습니다. Manhattan 거리의 형태가 대략 파악이 되나요?

```
figure(figsize=(16,8))
plot(data["pickup_latitude"], data["pickup_longitude"], 'b,')
xlim(40.6, 40.9)
ylim(-74.05, -73.9)
```

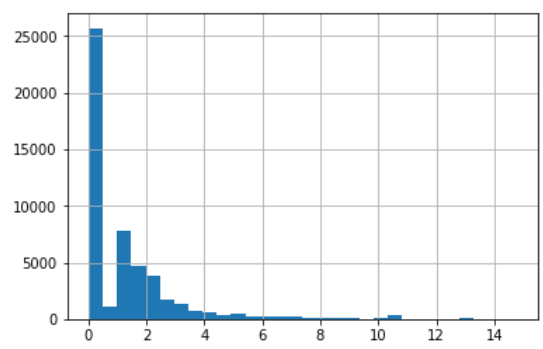
```
(-74.05, -73.9)
```



문제 구상: 오늘 우리는 택시 탑승/운행 기록으로부터 탑승객이 기사님에게 팁을 지불할지 아닐지를 예측해보고자 합니다.

승객이 지불한 팁 금액의 분포는 다음과 같습니다. 우리는 이를 binary화 하여, 단순히 팁을 지불하는지 아닌지를 예측해보고자 합니다.

```
data[data[' tip_amount'] < 15][' tip_amount'].hist(bins=30);
```



신용카드를 사용하여 거래를 한 탑승객의 정보만을 사용하여 data_subset을 만듭니다.

```
len(data)
data_subset = data[data[' payment_type'] != "CSH"]
data_subset.reset_index(inplace=True, drop=True)
len(data_subset)
```

```
25143
```

6.2 Conduct a Model-based Real World Data Analysis

```
# Setup target
data_subset['tipped'] = (data_subset[' tip_amount'] > 0).astype("int")
data_subset['tipped'].value_counts()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
  """Entry point for launching an IPython kernel.
1    24179
0      964
Name: tipped, dtype: int64
```

```
feats1 = [u'rate_code', 'passenger_count', u'trip_time_in_secs', u'trip_distance',
          u'pickup_longitude', u'pickup_latitude', u'dropoff_longitude', u'dropoff_latitude',
          ' fare_amount', u' surcharge', u' mta_tax', ' tolls_amount']
```

```
M = len(data_subset)
rand_idx = arange(M)
random.shuffle(rand_idx)
train_idx = rand_idx[int(M*0.2):]
test_idx = rand_idx[:int(M*0.2)]
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
data_subset_scaled = sc.fit_transform(data_subset[feats1])
data_subset_scaled[train_idx.tolist(),:].shape
```

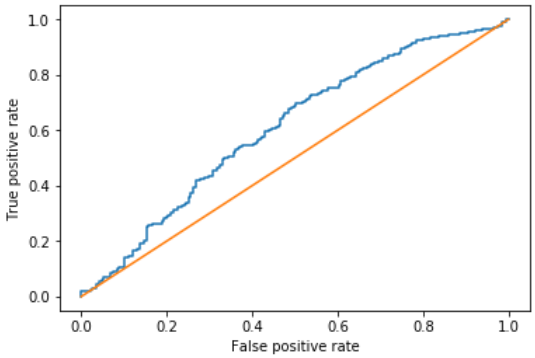
```
(20115, 12)
```

Building Baseline Classifiers

먼저 logistic regression을 사용하여 데이터를 학습하고, tip을 받게 될지, 아닐지 여부를 예측하여 봅니다. 학습 이후에는 accuracy와 auroc라는 평가 방법으로 모델의 성능을 평가해봅니다.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score, f1_score, log_loss,
                             roc_curve, roc_auc_score)

logreg = LogisticRegression()
parameters = {'penalty': ['l2'],
              'C': [10e-5, 10e-3, 10e-2, 10e-1, 10e0, 10e1, 10e2, 10e3, 10e5]}
gridsearch = GridSearchCV(logreg, parameters, scoring='accuracy', cv=5)
gridsearch.fit(data_subset.loc[train_idx,feats1], data_subset['tipped'].loc[train_idx])
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

# Step 3: Get model with best hyperparameters
best_logreg = gridsearch.best_estimator_

# Step 4: Get best model performance from testing set
# binary prediction (0/1)
y_pred = best_logreg.predict(data_subset.ix[test_idx,feats1])
# probability estimation [0, 1]
y_prob = best_logreg.predict_proba(data_subset.ix[test_idx,feats1])
test_acc = accuracy_score(data_subset['tipped'].loc[test_idx], y_pred)
print(f'test_acc = {test_acc}')

fpr, tpr, thr = roc_curve(data_subset['tipped'].loc[test_idx], y_prob[:,1])
test_auc = roc_auc_score(data_subset['tipped'].loc[test_idx], y_prob[:,1])
print(f'test_auc = {test_auc}')

plot(fpr,tpr)
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")

```

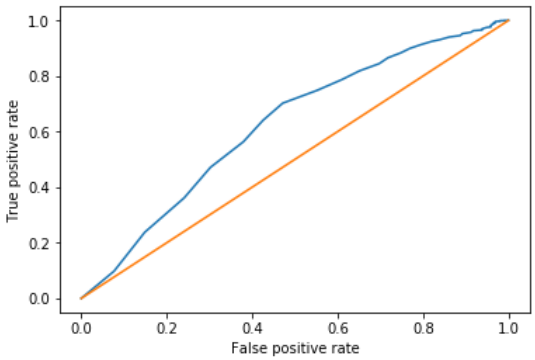
```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/base.py:929: ConvergenceWarning: Liblinear failed to converge, inc
  "the number of iterations.", ConvergenceWarning)
gridsearch.best_params_ = {'C': 0.1, 'penalty': 'l2'}
test_acc = 0.9618138424821002
test_auc = 0.6086568737707917
Text(0, 0.5, 'True positive rate')
```



이번엔 random forest를 사용하여 같은 작업을 반복해 봅니다. 앞서 logistic regression의 성능과 비교를 해볼까요?

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
parameters = {'n_estimators': [150],
              'criterion': ['gini', 'entropy']}
gridsearch = GridSearchCV(rf, parameters, scoring='accuracy', cv=5)
gridsearch.fit(data_subset.loc[train_idx,feats1], data_subset['tipped'].loc[train_idx])
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

# Step 3: Get model with best hyperparameters
best_rf = gridsearch.best_estimator_

# Step 4: Get best model performance from testing set
# binary prediction (0/1)
y_pred = best_rf.predict(data_subset.loc[test_idx,feats1])
# probability estimation [0, 1]
y_prob = best_rf.predict_proba(data_subset.loc[test_idx,feats1])
test_acc = accuracy_score(data_subset['tipped'].loc[test_idx], y_pred)
print(f'test_acc = {test_acc}')

fpr, tpr, thr = roc_curve(data_subset['tipped'].loc[test_idx], y_prob[:,1])
test_auc = roc_auc_score(data_subset['tipped'].loc[test_idx], y_prob[:,1])
print(f'test_auc = {test_auc}')

plot(fpr,tpr)
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")

```

```
gridsearch.best_params_ = {'criterion': 'entropy', 'n_estimators': 150}
test_acc = 0.961217183770883
test_auc = 0.623946479067522
Text(0, 0.5, 'True positive rate')
```



Random Forest가 제공하는 멤버 `feature_importances_`를 사용하여, input features를 중요도 순으로 나열해 봅니다.

```
fi = zip(feats1, best_rf.feature_importances_)
# fi.sort(key=lambda x: -x[1])
df=pd.DataFrame(fi, columns=["Feature","Importance"])
df.sort_values(["Importance"], ascending=False)
```

	Feature	Importance
5	pickup_latitude	0.157129
6	dropoff_longitude	0.156450
7	dropoff_latitude	0.154654
4	pickup_longitude	0.154497
3	trip_distance	0.143540
8	fare_amount	0.090359
2	trip_time_in_secs	0.083046
1	passenger_count	0.037790
9	surcharge	0.011775
0	rate_code	0.005064
11	tolls_amount	0.005022
10	mta_tax	0.000674

지금까지 택시 운행 기록의 위치, 시간, 이동 거리 및 요금 등의 정보를 바탕으로, 탑승객이 기사님께 팁을 줄지, 아니면 주지 않을지 여부를 예측해 보았습니다.

Improving Input Features #1

그런데, '이동 속도'라는 기존에 고려되지 않던 정보를 이용하면, 탑승객의 팁 지불 여부를 더 정확하게 예측 할 수 있을 것이라는 인사이트가 떠올랐습니다. 속도는 이동거리/이동시간을 계산하여 구할 수 있습니다. 해당 정보를 모델에 어떻게 추가할 수 있을까요?

```
data_subset['trip_time_in_secs'][data_subset['trip_time_in_secs'] < 1e-3] = -1
data_subset['speed'] = data_subset['trip_distance'] / data_subset['trip_time_in_secs']
```

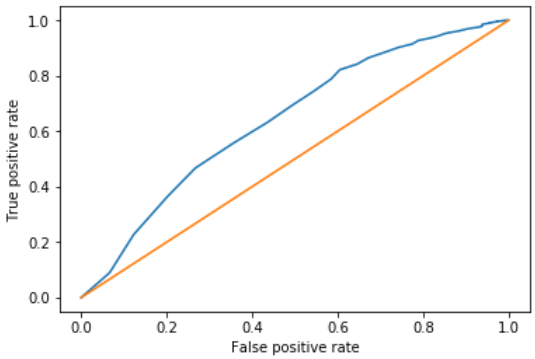
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/pandas/core/generic.py:8682: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
  self._update_inplace(new_data)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
```

```
feats2 = feats1 + ['speed']
feats2.remove('trip_time_in_secs')
```

```
rf = RandomForestClassifier()
parameters = {'n_estimators': [150],
              'criterion': ['gini', 'entropy']}
gridsearch = GridSearchCV(rf, parameters, scoring='accuracy', cv=5)
gridsearch.fit(data_subset.loc[train_idx,feats2], data_subset['tipped'].loc[train_idx])
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

# Step 3: Get model with best hyperparameters
best_rf = gridsearch.best_estimator_

# Step 4: Get best model performance from testing set
# binary prediction (0/1)
y_pred = best_rf.predict(data_subset.ix[test_idx,feats2])
# probability estimation [0, 1]
y_prob = best_rf.predict_proba(data_subset.ix[test_idx,feats2])
test_acc = accuracy_score(data_subset['tipped'].loc[test_idx], y_pred)
print(f'test_acc = {test_acc}')

fpr, tpr, thr = roc_curve(data_subset['tipped'].ix[test_idx], y_prob[:,1])
test_auc = roc_auc_score(data_subset['tipped'].ix[test_idx], y_prob[:,1])
```

```
print(f'test_auc = {test_auc}')

plot(fpr,tpr)
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")

```

```
gridsearch.best_params_ = {'criterion': 'entropy', 'n_estimators': 150}
test_acc = 0.9608194112967383
test_auc = 0.6422734724410701
Text(0, 0.5, 'True positive rate')
```




```
fi = zip(feats2, best_rf.feature_importances_)
# fi.sort(key=lambda x: -x[1])
df=pd.DataFrame(fi, columns=["Feature","Importance"])
df.sort_values(["Importance"], ascending=False)
```

	Feature	Importance
5	dropoff_longitude	0.148205
3	pickup_longitude	0.145782
4	pickup_latitude	0.144440
6	dropoff_latitude	0.143288
11	speed	0.139764
2	trip_distance	0.134540
7	fare_amount	0.086822
1	passenger_count	0.035472
8	surcharge	0.011805
0	rate_code	0.004826
10	tolls_amount	0.004287
9	mta_tax	0.000771

Improving Input Features #2

아직 사용하지 않은 몇 가지 categorical features를 사용하면, 모델의 성능을 더 높일 수 있을 것 같은 생각이 들었습니다. 이를 위해 `cat_to_num()`이라는 binarization 함수를 만들어 줍니다.

```
from sklearn.feature_extraction import DictVectorizer

def cat_to_num(data):
    categories = unique(data)
    features = {}
    for cat in categories:
        binary = (data == cat)
        features["%s:%s"%(data.name, cat)] = binary.astype("int")
    return pd.DataFrame(features)
```

```
feats3 = feats2

payment_type_cats = cat_to_num(data_subset[' payment_type'])
vendor_id_cats = cat_to_num(data_subset['vendor_id'])
# store_and_fwd_flag_cats = cat_to_num(str(data_subset['store_and_fwd_flag']))
rate_code = cat_to_num(data_subset['rate_code'])

data_subset = data_subset.join(payment_type_cats)
# feats3 += payment_type_cats.columns
feats3.extend(list(payment_type_cats.columns))
```

```
data_subset = data_subset.join(vendor_id_cats)
# feats3 += vendor_id_cats.columns
feats3.extend(list(vendor_id_cats.columns))

# # data_subset = data_subset.join(store_and_fwd_flag_cats)
# # feats3 += store_and_fwd_flag_cats.columns
data_subset = data_subset.join(rate_code)
# feats3 += rate_code.columns
feats3.extend(list(rate_code.columns))
```

```
# show the list of features
feats3
```

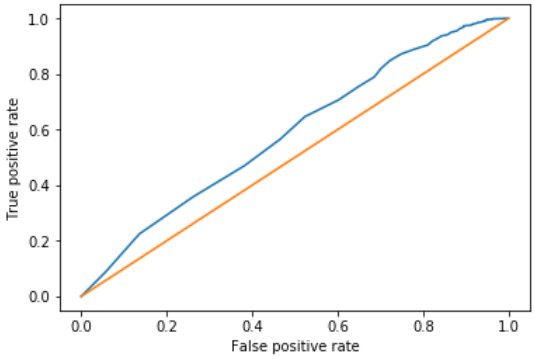
```
['rate_code',
 'passenger_count',
 'trip_distance',
 'pickup_longitude',
 'pickup_latitude',
 'dropoff_longitude',
 'dropoff_latitude',
 ' fare_amount',
 ' surcharge',
 ' mta_tax',
 ' tolls_amount',
 'speed',
 ' payment_type:CRD',
 ' payment_type:DIS',
 ' payment_type:NOC',
 ' payment_type:UNK',
 'vendor_id:CMT',
 'vendor_id:VTS',
 'rate_code:1',
 'rate_code:2',
 'rate_code:3',
 'rate_code:4',
 'rate_code:5']
```

확장된 데이터셋을 활용하여 모델을 다시 학습시켜보겠습니다.

```
rf = RandomForestClassifier()
parameters = {'n_estimators': [150],
              'criterion': ['gini', 'entropy']}
gridsearch = GridSearchCV(rf, parameters, scoring='accuracy', cv=5)
gridsearch.fit(data_subset.loc[train_idx,feats3], data_subset['tipped'].loc[train_idx])
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

# Step 3: Get model with best hyperparameters
best_rf = gridsearch.best_estimator_

# Step 4: Get best model performance from testing set
# binary prediction (0/1)
y_pred = best_rf.predict(data_subset.ix[test_idx,feats3])
# probability estimation [0, 1]
y_prob = best_rf.predict_proba(data_subset.ix[test_idx,feats3])
test_acc = accuracy_score(data_subset['tipped'].loc[test_idx], y_pred)
print(f'test_acc = {test_acc}')

fpr, tpr, thr = roc_curve(data_subset['tipped'].ix[test_idx], y_prob[:,1])
test_auc = roc_auc_score(data_subset['tipped'].ix[test_idx], y_prob[:,1])
print(f'test_auc = {test_auc}')

plot(fpr,tpr)
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")

fi = zip(feats3, best_rf.feature_importances_)
# fi.sort(key=lambda x: -x[1])
df=pd.DataFrame(fi, columns=["Feature","Importance"])
### Improving Input Features #1
df.sort_values(["Importance"], ascending=False)
```

```
gridsearch.best_params_ = {'criterion': 'entropy', 'n_estimators': 150}
test_acc = 0.9622116149562451
test_auc = 0.5864707798849835
```

	Feature	Importance
5	dropoff_longitude	0.146973
4	pickup_latitude	0.142643
3	pickup_longitude	0.141393
6	dropoff_latitude	0.141276
11	speed	0.138894
2	trip_distance	0.131572
7	fare_amount	0.094082
1	passenger_count	0.036805
8	surcharge	0.010434
10	tolls_amount	0.004900
0	rate_code	0.002880
18	rate_code:1	0.001370
22	rate_code:5	0.001370
17	vendor_id:VTS	0.001152
13	payment_type:DIS	0.001008
16	vendor_id:CMT	0.000902
9	mta_tax	0.000595
19	rate_code:2	0.000543
12	payment_type:CRD	0.000505
21	rate_code:4	0.000393
14	payment_type:NOC	0.000140
15	payment_type:UNK	0.000096
20	rate_code:3	0.000073



Improving Input Features #3

마지막으로 지금까지 사용하지 않은 시간 정보를 이용하여 탑승객의 팁 지불 여부를 예측해보겠습니다. 데이터셋에는 탑승 및 하차 시간이 timestamp의 형태로 기입되어 있습니다. 하지만 우리의 ML 모델들은 timestamp 형태의 데이터를 쉽게 처리하지 못합니다.

모델 학습을 위해 timestamp를 좀 더 의미있는 형태로 변형시켜 보겠습니다.

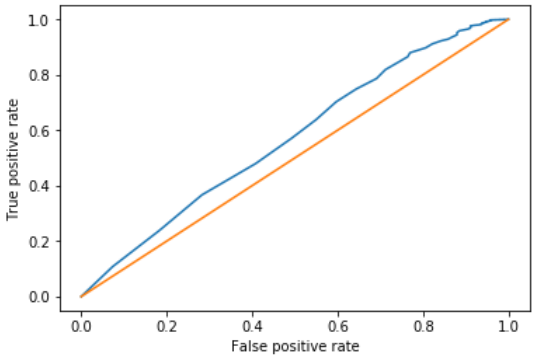
```
feats4 = feats3

# Datetime features (hour of day, day of week, week of year)
pickup = pd.to_datetime(data['pickup_datetime'])
dropoff = pd.to_datetime(data['dropoff_datetime'])
data['pickup_hour'] = pickup.apply(lambda e: e.hour)
data['pickup_day'] = pickup.apply(lambda e: e.dayofweek)
#data['pickup_week'] = pickup.apply(lambda e: e.week)
data['dropoff_hour'] = dropoff.apply(lambda e: e.hour)
data['dropoff_day'] = dropoff.apply(lambda e: e.dayofweek)
#data['dropoff_week'] = dropoff.apply(lambda e: e.week)

```

```
feats4 += ['pickup_hour', 'pickup_day', 'dropoff_hour', 'dropoff_day']
```

```
rf = RandomForestClassifier()
parameters = {'n_estimators': [150],
              'criterion': ['gini', 'entropy']}
gridsearch = GridSearchCV(rf, parameters, scoring='accuracy', cv=5)
gridsearch.fit(data_subset.loc[train_idx,feats1], data_subset['tipped'].loc[train_idx])
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

# Step 3: Get model with best hyperparameters
best_rf = gridsearch.best_estimator_

# Step 4: Get best model performance from testing set
# binary prediction (0/1)
y_pred = best_rf.predict(data_subset.ix[test_idx,feats1])
# probability estimation [0, 1]
y_prob = best_rf.predict_proba(data_subset.ix[test_idx,feats1])
test_acc = accuracy_score(data_subset['tipped'].loc[test_idx], y_pred)
print(f'test_acc = {test_acc}')

fpr, tpr, thr = roc_curve(data_subset['tipped'].ix[test_idx], y_prob[:,1])
test_auc = roc_auc_score(data_subset['tipped'].ix[test_idx], y_prob[:,1])
print(f'test_auc = {test_auc}')

plot(fpr,tpr)
plot(fpr,fpr)
xlabel("False positive rate")
ylabel("True positive rate")

fi = zip(feats1, best_rf.feature_importances_)
# fi.sort(key=lambda x: -x[1])
df=pd.DataFrame(fi, columns=["Feature","Importance"])
df.sort_values(["Importance"], ascending=False)
```

```
gridsearch.best_params_ = {'criterion': 'entropy', 'n_estimators': 150}
test_acc = 0.9620127287191726
test_auc = 0.57115093406302
```

	Feature	Importance
6	dropoff_longitude	0.159736
5	pickup_latitude	0.154377
7	dropoff_latitude	0.154167
4	pickup_longitude	0.154158
3	trip_distance	0.143323
8	fare_amount	0.088609
2	trip_time_in_secs	0.085751
1	passenger_count	0.038107
9	surcharge	0.011750
11	tolls_amount	0.004800
0	rate_code	0.004602
10	mta_tax	0.000620



Wrap-up

지금까지 NYC Taxi Data로부터 탑승객의 팁 지불 여부를 예측해보는 작업을 해보았습니다.

마지막으로 탑승객의 승하차 지점으로부터 재구성한 뉴욕의 지도입니다.

```
figure(figsize=(16,8))
plot(data_subset[data_subset['tipped'] == True]["dropoff_latitude"],
```

```
     data_subset[data_subset['tipped'] == True]["dropoff_longitude"], 'b,')
plot(data_subset[data_subset['tipped'] == False]["dropoff_latitude"],
     data_subset[data_subset['tipped'] == False]["dropoff_longitude"], 'r,')
xlim(40.6, 40.9)
ylim(-74.05, -73.9)
```

```
(-74.05, -73.9)
```