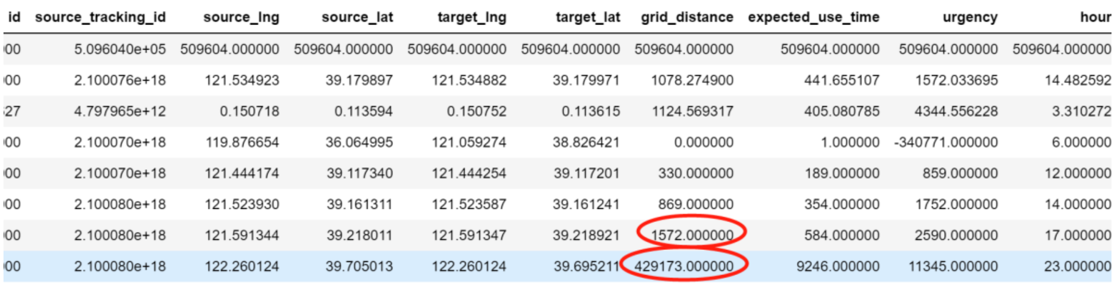
Regression

Question(a)

Data Pre-Processing

First, we described the data to see low variance variables and whether there exist some outliers:



id	source_tracking_id	source_lng	source_lat	target_lng	target_lat	grid_distance	expected_use_time	urgency	hour
00	5.096040e+05	509604.000000	509604.000000	509604.000000	509604.000000	509604.000000	509604.000000	509604.000000	509604.000000
00	2.100076e+18	121.534923	39.179897	121.534882	39.179971	1078.274900	441.655107	1572.033695	14.482592
27	4.797965e+12	0.150718	0.113594	0.150752	0.113615	1124.569317	405.080785	4344.556228	3.310272
00	2.100070e+18	119.876654	36.064995	121.059274	38.826421	0.000000	1.000000	-340771.000000	6.000000
00	2.100070e+18	121.444174	39.117340	121.444254	39.117201	330.000000	189.000000	859.000000	12.000000
00	2.100080e+18	121.523930	39.161311	121.523587	39.161241	869.000000	354.000000	1752.000000	14.000000
00	2.100080e+18	121.591344	39.218011	121.591347	39.218921	1572.000000	584.000000	2590.000000	17.000000
00	2.100080e+18	122.260124	39.705013	122.260124	39.695211	429173.000000	9246.000000	11345.000000	23.000000

We found that `grid_distance` has some outliers because 75% quantile is 1572 whereas its max value is 429173, which is approximate 300 times of 75% quantile. Therefore, we removed those observations by counting the number of values which more than 10000 meters, 20000 meters and decided deleting observations whose `grid_distance` is more than 10000 meters.

```
# remove outliers
df_train = df_train.drop(df_train[df_train.grid_distance>10000].index)
```

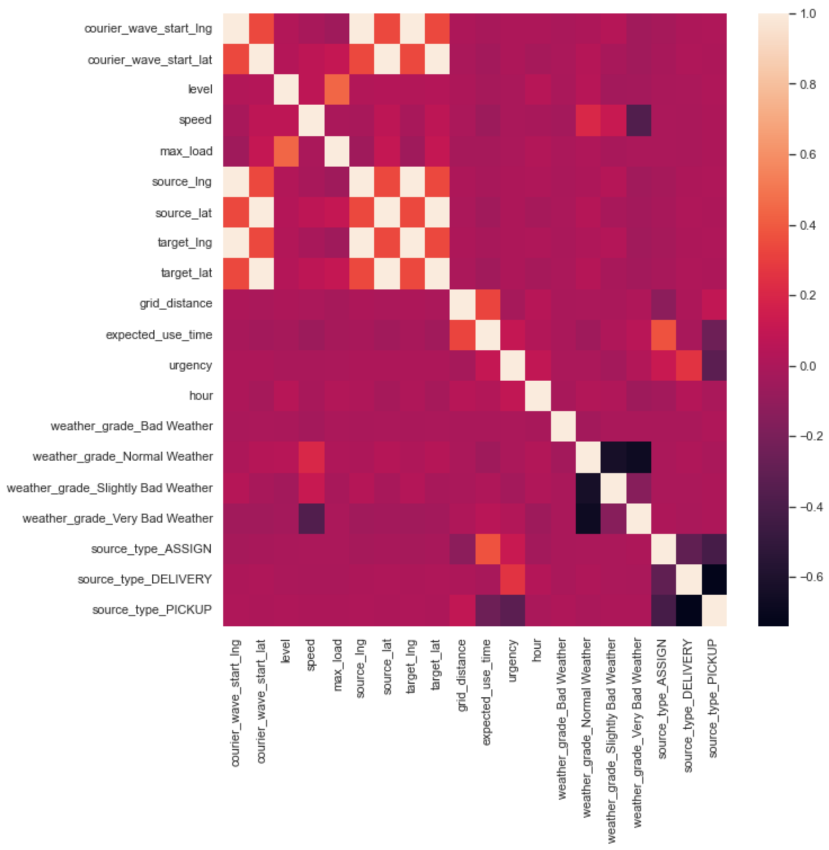
Second, we down-sampled the training data set into 10000 and dropped some unreasonable variables like id and some variables we cannot use to predict the testing data set. Then, we converted some discrete variables into numerical data by getting dummies. Incidentally, we generated our labels which represents action type is pick-up when label equals 1.

```
# down-sample randomly
df = df_train.sample(n=10000,random_state=666,axis=0)

# drop unreasonable variables and variables we can't use in test set
df = df.drop(columns = ['courier_id', 'wave_index', 'tracking_id','date','group',
                        'id','shop_id','aoi_id','source_tracking_id','expected_use_time'])

# convert into numerical data
df = pd.get_dummies(df, drop_first=True)
df['action_type_DELIVERY'] = np.array(df['action_type_PICKUP']==0).astype(int)
df = df.drop(columns=['action_type_PICKUP'])
```

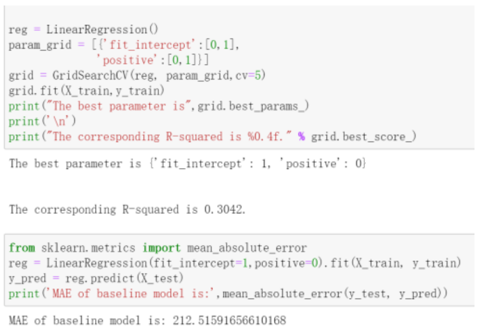
Third, we standardized the data sample and ran a heat map to reveal the correlations between all the variables to see whether independent variables have strong correlations and which independent variables have correlations with label.



According to the figure above, we chose the following variables as our features for now: speed', 'grid_distance','urgency','weather_grade_Normal Weather', 'weather_grade_Very Bad Weather', 'source_type_ASSIGN','source_type_PICKUP'

Question(b)

Baseline Model



```
reg = LinearRegression()
param_grid = [{'fit_intercept':[0,1],
              'positive':[0,1]}]
grid = GridSearchCV(reg, param_grid,cv=5)
grid.fit(X_train,y_train)
print("The best parameter is",grid.best_params_)
print('\n')
print("The corresponding R-squared is %0.4f." % grid.best_score_)
```

```
The best parameter is {'fit_intercept': 1, 'positive': 0}


The corresponding R-squared is 0.3042.
```

```
from sklearn.metrics import mean_absolute_error
reg = LinearRegression(fit_intercept=1,positive=0).fit(X_train, y_train)
y_pred = reg.predict(X_test)
print('MAE of baseline model is:',mean_absolute_error(y_test, y_pred))
```

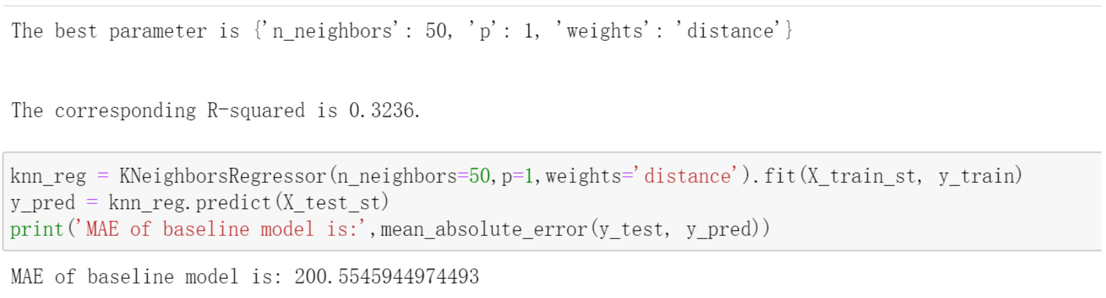
```
MAE of baseline model is: 212.51591656610168
```

We used GridSearchCV to cross validate the linear regression model which is our baseline model. The MAE is 212.5, which is a relatively high error.

Question(c)

More Complex Models

K-NN:



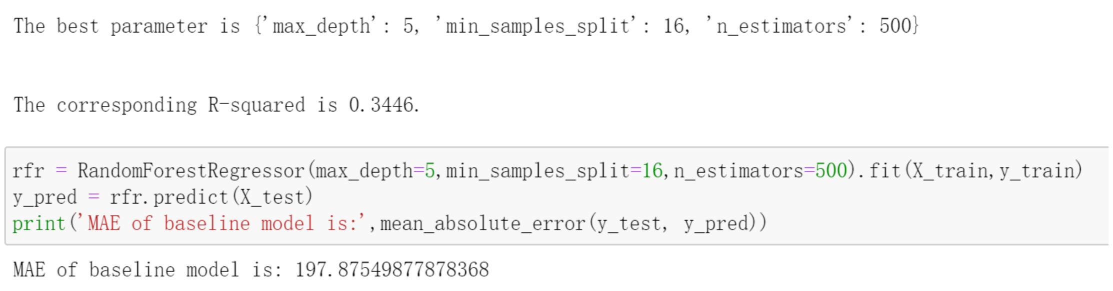
```
The best parameter is {'n_neighbors': 50, 'p': 1, 'weights': 'distance'}


The corresponding R-squared is 0.3236.
```

```
knn_reg = KNeighborsRegressor(n_neighbors=50,p=1,weights='distance').fit(X_train_st, y_train)
y_pred = knn_reg.predict(X_test_st)
print('MAE of baseline model is:',mean_absolute_error(y_test, y_pred))
```

```
MAE of baseline model is: 200.5545944974493
```

Random Forest:



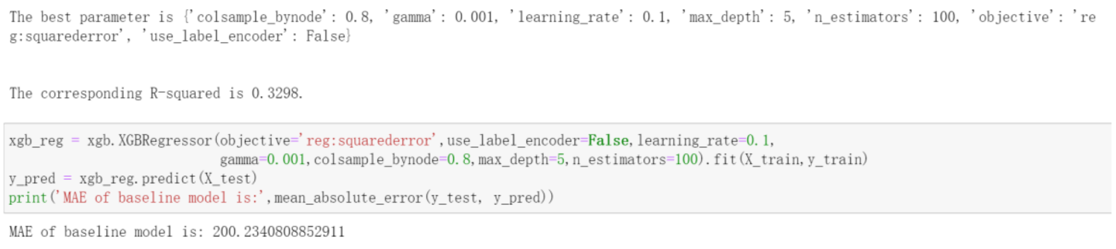
```
The best parameter is {'max_depth': 5, 'min_samples_split': 16, 'n_estimators': 500}


The corresponding R-squared is 0.3446.
```

```
rfr = RandomForestRegressor(max_depth=5,min_samples_split=16,n_estimators=500).fit(X_train,y_train)
y_pred = rfr.predict(X_test)
print('MAE of baseline model is:',mean_absolute_error(y_test, y_pred))
```

```
MAE of baseline model is: 197.87549877878368
```

XGBT:



```
The best parameter is {'colsample_bynode': 0.8, 'gamma': 0.001, 'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'objective': 're
g:squarederror', 'use_label_encoder': False}


The corresponding R-squared is 0.3298.
```

```
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror',use_label_encoder=False,learning_rate=0.1,
                           gamma=0.001,colsample_bynode=0.8,max_depth=5,n_estimators=100).fit(X_train,y_train)
y_pred = xgb_reg.predict(X_test)
print('MAE of baseline model is:',mean_absolute_error(y_test, y_pred))
```

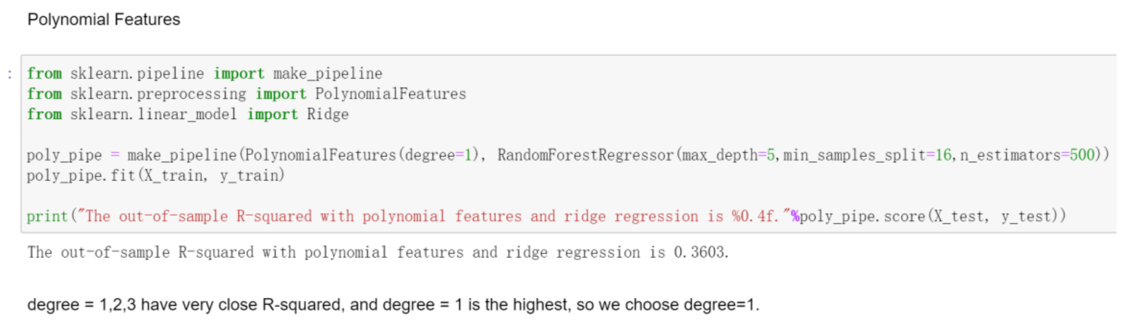
```
MAE of baseline model is: 200.2340808852911
```

After trying more complex models, we found that MAEs dropped to 200. And the three models above have approximately same results. So, we still chose XGB as our final model.

Question(d)

Feature Engineering

First, we polynomial features and found degree=1 is the best.


Polynomial Features

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge

poly_pipe = make_pipeline(PolynomialFeatures(degree=1), RandomForestRegressor(max_depth=5,min_samples_split=16,n_estimators=500))
poly_pipe.fit(X_train, y_train)

print("The out-of-sample R-squared with polynomial features and ridge regression is %0.4f."%poly_pipe.score(X_test, y_test))
```

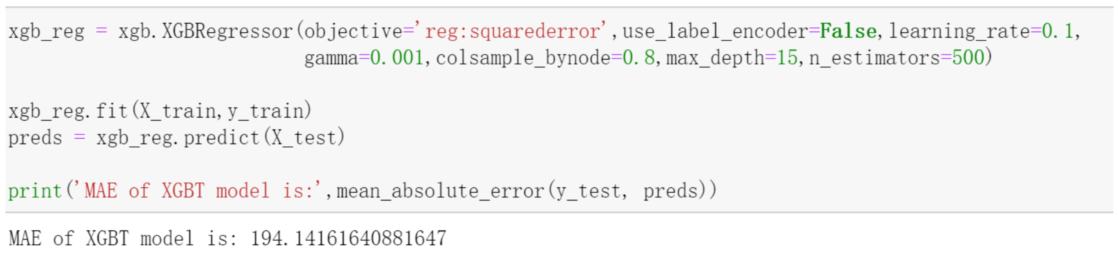
```
The out-of-sample R-squared with polynomial features and ridge regression is 0.3603.
```

degree = 1,2,3 have very close R-squared, and degree = 1 is the highest, so we choose degree=1.

Second, we conducted feature selection by RFECV using all the variables except id. This do help us to find some useful features but I still added some other features to make the MAE smaller. As a result, we chose the following features:

'grid_distance','urgency','source_type_DELIVERY','source_type_PICKUP','speed','target_lng','target_lat','source_lng','source_lat','courier_wave_start_lng','courier_wave_start_lat','hour','max_load'

Third, we split the whole training data set into train set and test set and refitted the model using the recommended features.



```
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror',use_label_encoder=False,learning_rate=0.1,
                           gamma=0.001,colsample_bynode=0.8,max_depth=15,n_estimators=500)

xgb_reg.fit(X_train,y_train)
preds = xgb_reg.predict(X_test)

print('MAE of XGBT model is:',mean_absolute_error(y_test, preds))
```

```
MAE of XGBT model is: 194.14161640881647
```

Finally, the MAE dropped to 194. We believe this would drop below 190 after fitting all the 500000 observations.

Question(e)

We used the whole training set to fit our model and made the parameters of XGBT better and predicted our testing set. We found a mistake in test data set because the tracking id and source_type were swapped.

According to our model, source_type, grid_distance and urgency are the most important features in predicting the expected use time. Weather condition seemed no effects on courier’s action type.