In [26]:
```
using JuMP, Gurobi, LinearAlgebra, CSV, DataFrames, Pkg, Distances, Random,
```

In [27]:
```
items = CSV.read("items.csv",DataFrame; header=1);
sales = CSV.read("sales.csv",DataFrame; header=1);
side = CSV.read("sideinformation.csv",DataFrame; header=1);
```

In [100]:
```
Q = [50,100, 150, 200, 250, 300]
merged = innerjoin(items, sales, on = :item_nbr)
first(merged, 5);
```

2.b.i

In [86]:
```
dbf = merged[joined_data.date .== "14/08/2017", :];
dbf2 = merged[joined_data.date .== "15/08/2017", :];
d_np = dbf[dbf.perishable .== 0, :].unit_sales;
e_p = dbf[dbf.perishable .== 1, :].unit_sales;
d_np_15 = dbf2[dbf2.perishable .== 0, :].unit_sales;
e_p_15 = dbf2[dbf2.perishable .== 1, :].unit_sales;
p_p = dbf[dbf.perishable .== 1, :].price;
p_np = dbf[dbf.perishable .== 0, :].price;
c_p = dbf[dbf.perishable .== 1, :].cost;
c_np = dbf[dbf.perishable .== 0, :].cost;
n_p = nrow(dbf[dbf.perishable .== 1, :])
n_np = nrow(dbf[dbf.perishable .== 0, :]);
```

In [87]:
```
function optimize_values(Q; solver_output = 0)
model = Model(with_optimizer(Gurobi.Optimizer))
set_optimizer_attribute(model, "OutputFlag", solver_output)

@variable(model, s_np[i=1:n_np]>=0, Int)
@variable(model, t_p[j=1:n_p]>=0, Int)
@variable(model, phi[i=1:n_np])
@variable(model, theta[j=1:n_p])

@constraint(model, [j=1:num_p], t_p[j] <= (1/20*Q))
@constraint(model, [i=1:num_np], s_np[i] <= (1/20*Q))
@constraint(model, [j=1:num_p], theta[j] <= e_p[j])
@constraint(model, [j=1:num_p], theta[j] <= t_p[j])
@constraint(model, [i=1:num_np], phi[i] <= d_np[i])
@constraint(model, [i=1:num_np], phi[i] <= s_np[i])
@constraint(model, [i=1:num_np, j=1:n_p], sum(s_np[i] for i=1:n_np) + sum(t
@objective(model,Max, sum(p_p[j]*theta[j] - c_p[j]*t_p[j] for j=1:n_p)+ sum
JuMP.optimize!(model)
obj_val = JuMP.objective_value(model)
return obj_val
end
```

Out[87]:
```
optimize_values (generic function with 1 method)
```

In [88]:
```
profit = 0
for i in Q
    profit = profit + optimize_values(i)
end
println("Average Profit: ", profit/length(Q))
```

```
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Average Profit: 508.2975666666666
```

2.b.ii

In [90]:
```
dbf = merged[joined_data.date .== "15/08/2017", :]
d_np = dbf[dbf.perishable .== 0, :].unit_sales;
e_p = dbf[dbf.perishable .== 1, :].unit_sales;
p_p = dbf[dbf.perishable .== 1, :].price;
p_np = dbf[dbf.perishable .== 0, :].price;
c_p = dbf[dbf.perishable .== 1, :].cost;
c_np = dbf[dbf.perishable .== 0, :].cost;
n_p = nrow(dbf[dbf.perishable .== 1, :])
n_np = nrow(dbf[dbf.perishable .== 0, :]);
```

In [91]:
```
profit = 0
for i in Q
    profit = profit + optimize_values(i)
end
println("Average Profit: ", profit/length(Q))
```

```
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Average Profit: 460.2348866666667
```

2.b.iii

The approach used below judges the five nearest neighbors as those which are most similar in terms of holiday status and oil price (isHoliday, oilPrice). Both are an effort to indicate days with similar economic and consumer patterns.

In [92]:
```
side[!,:dt]=Date.(side.date, "dd/mm/yyyy");
side[!,:oil_p]=(side[!,:oilPrice].-minimum(side[!,:oilPrice]))./(maximum(si
#obtaining the most recent 100 days
recent=subset(side, :dt => ByRow( >=(Date(2017,8,15)-Dates.Day(100))));
oil_p_15 = subset(recent, :date => ByRow(isequal("15/08/2017")))[:, 3];
hol_15 = subset(recent, :date => ByRow(isequal("15/08/2017")))[:, 2];
#calculating euclidean distance
side.dist = ((oil_p_15 .- side.oil_p).^2 + (hol_15 .- side.isHoliday).^2).^
#finding the 5 "nearest" neighbors
dist_sorted = sort!(side, [:dist]);
fiveNN_dates = dist_sorted[:, :date][2:6, :]
```

Out[92]:
```
5×1 Matrix{String}:
 "3/06/2017"
 "14/08/2017"
 "12/03/2017"
 "2/05/2017"
 "2/06/2017"
```

In [93]:
```
#subsetting data to five nearest dates
df2 = joined_data[∈(fiveNN_dates).(merged.date), :];
df2_p = df2[df2.perishable .== 1, :];
df2_np = df2[df2.perishable .== 0, :];
```

In [94]:
```
#subsetting data to five nearest dates for sales records and splitting into
knn_1 = sales[sales[!,:date] .== fiveNN_dates[1], :];
knn_2 = sales[sales[!,:date] .== fiveNN_dates[2], :];
knn_3 = sales[sales[!,:date] .== fiveNN_dates[3], :];
knn_4 = sales[sales[!,:date] .== fiveNN_dates[4], :];
knn_5 = sales[sales[!,:date] .== fiveNN_dates[5], :];
#obtaining unit_sales for all five neighbors
items.knn_demand_1 = knn_1[:, 5];
items.knn_demand_2 = knn_2[:, 5];
items.knn_demand_3 = knn_3[:, 5];
items.knn_demand_4 = knn_4[:, 5];
items.knn_demand_5 = knn_5[:, 5];
d_np = Matrix(items[items.perishable .== 0, :][:, Not(1:6)]);
e_p = Matrix(items[items.perishable .== 1, :][:, Not(1:6)]);
p_p = items[items.perishable .== 1, :].price;
p_np = items[items.perishable .== 0, :].price;
c_p = items[items.perishable .== 1, :].cost;
c_np = items[items.perishable .== 0, :].cost;
n_p = nrow(items[items.perishable .== 1, :]);
n_np = nrow(items[items.perishable .== 0, :]);
```

In [97]:
```
function optimize_values(Q; solver_output = 0)
    model = Model(with_optimizer(Gurobi.Optimizer))
    set_optimizer_attribute(model, "OutputFlag", solver_output)

    K = 5
    @variable(model, s_np[i=1:n_np]>=0, Int)
    @variable(model, t_p[j=1:n_p]>=0, Int)
    @variable(model, phi[i=1:n_np, k =1:K])
    @variable(model, theta[j=1:n_p, k =1:K])

    @constraint(model, [j=1:n_p], t_p[j] <= (1/20*Q))
    @constraint(model, [i=1:n_np], s_np[i] <= (1/20*Q))

    @constraint(model, [j=1:n_p, k=1:K], theta[j, k] <= e_p[j, k])
    @constraint(model, [j=1:n_p, k=1:K], theta[j, k] <= t_p[j])

    @constraint(model, [i=1:n_np, k=1:K], phi[i, k] <= d_np[i, k])
    @constraint(model, [i=1:n_np, k=1:K], phi[i, k] <= s_np[i])

    @constraint(model, [i=1:n_np, j=1:n_p], sum(s_np[i] for i=1:num_np) + s
    @objective(model,Max,sum(1/K*sum(p_p[j]*theta[j, k] - c_p[j]*t_p[j] for
    sum(1/K*sum(p_np[i]*phi[i, k] - c_np[i]*phi[i, k] for i=1:n_np) for k=1

    JuMP.optimize!(model)
    obj_val = JuMP.objective_value(model)
    return (obj_val, JuMP.value.(s_np), JuMP.value.(t_p), JuMP.value.(theta
end
```

Out[97]: optimize_values (generic function with 1 method)

In [96]:
```
profit = 0
actual_profit = 0
for i in Q
    vals = optimize_values(i)
    obj_value = vals[1]
    opt_s = vals[2]
    opt_t = vals[3]
    profit = profit + obj_value
    ac_prof = sum(p_p[j]*min(e_p_15[j],opt_t[j]) - c_p[j]*opt_t[j] for j=1:
    sum(p_np[i]*min(d_np_15[i], opt_s[i])- c_np[i]*min(d_np_15[i], opt_s[i]
    actual_profit = actual_profit + ac_prof
end
println("Average Optimized Profit: ", profit/length(Q))
println("Average Actual Profit: ", actual_profit/length(Q))
```

```
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Average Optimized Profit: 384.325
Average Actual Profit: 333.22822
```

2.b.iv

Throughout this course we have demonstrated the empirical dominance of optimization based machine learning as opposed to heuristic methods. We would expect the same performance differential in this setting. Thus, with optimization based machine learning methods we'd expect the profit to be even higher.