Dual Prediction Model with Velocity Prediction for Trip Time prediction

June 12, 2023

[3]:
```
import numpy as np
import pandas as pd
from datetime import datetime
import re
import math
```

[4]:
```
#Read into Dataframe
taxi_data = pd.read_csv("kaggle_data/train.csv")
taxi_data['ORIGIN_STAND'] = taxi_data['ORIGIN_STAND'].fillna(0)

#Calculate and Create Time Column - only for training label creation and test␣
 ↪validation
def travel_time(polyline):
    return max(polyline.count("[") - 2, 0) * 15

def parse_timestamp(taxi_data):
    date_time = datetime.fromtimestamp(taxi_data["TIMESTAMP"])
    return date_time.year, date_time.month, date_time.day, date_time.hour,␣
 ↪date_time.weekday()

taxi_data["LEN"] = taxi_data["POLYLINE"].apply(travel_time)

mean_duration = taxi_data["LEN"].mean()
standard_deviation = taxi_data["LEN"].std()
median = taxi_data["LEN"].median()
taxi_data = taxi_data[taxi_data["LEN"] < mean_duration + 3*standard_deviation]

outlier_threshold = 3
total_size = len(taxi_data)
trimmed_taxi_data = taxi_data[taxi_data["LEN"] < mean_duration +␣
 ↪outlier_threshold * standard_deviation]
print(f"Using: {len(trimmed_taxi_data)}/{total_size}")

trimmed_taxi_data[["YR", "MON", "DAY", "HR", "WK"]] =␣
 ↪trimmed_taxi_data[["TIMESTAMP"]].apply(parse_timestamp, axis=1,␣
 ↪result_type="expand")
```

```
Using: 1692771/1692771
```

0.0.1 Incorporating Velocity in Dataset

This factor is calculated using an initial partial trajectory provided for the taxi. It is useful for the duration calculation since speed plays a major role in travel time along with the distance. Since there is no clear way to accurately determine the total length of a trip before it ends, we can assume that an average velocity based on a taxi's initial trajectory will be a good estimate of its overall speed and will be a factor in determining trip duration

[5]:
```
#Velocity Calculation adapted from:
#https://www.ridgesolutions.ie/index.php/2013/11/14/
 ↪algorithm-to-calculate-speed-from-two-gps-latitude-and-longitude-points-and-time-difference
 ↪

def velocity(lat1, lon1, lat2, lon2):

    #Convert degrees to radians
    lat1 = lat1 * math.pi / 180.0;
    lon1 = lon1 * math.pi / 180.0;
    lat2 = lat2 * math.pi / 180.0;
    lon2 = lon2 * math.pi / 180.0;

    #radius of earth in metres
    r = 6378100;

    #P
    rho1 = r * math.cos(lat1)
    z1 = r * math.sin(lat1)
    x1 = rho1 * math.cos(lon1)
    y1 = rho1 * math.sin(lon1)

    #Q
    rho2 = r * math.cos(lat2)
    z2 = r * math.sin(lat2)
    x2 = rho2 * math.cos(lon2)
    y2 = rho2 * math.sin(lon2)

    #Dot product
    dot = (x1 * x2 + y1 * y2 + z1 * z2)
    cos_theta = dot / (r * r)

    if(cos_theta > 1):
        cos_theta = 1

    theta = math.acos(cos_theta)

    #Distance in Metres
```

```
    distance = r * theta

    return distance/15 #speed in meters per second
```

[6]:
```
trimmed_taxi_data.reset_index(drop=True, inplace=True)
```

[7]:
```
avg_velocities = []

#Average Velocity
def avg_velo(taxi_data):

    k = 10

    poly = taxi_data["POLYLINE"]

    for i in range(0, len(taxi_data), 1):
        coord = poly[i]
        coord = re.split(r',|\[|\]', coord)
        count = 0

        coordinates = []

        for value in coord:
            if (count > 2*(k-2) or count > len(coord)-2):
                break
            else:
                #print(value + str(value.isnumeric()))
                if(value != ''):
                    coordinates.append(float(value))

        velocities = []

        for j in range(0, int(len(coordinates)/4), 1):
            velocities.append(velocity(coordinates[j], coordinates[j+1],␣
↪coordinates[j+2], coordinates[j+3]))

        sum_velo = 0.0

        for velo in velocities:
            sum_velo += velo

        if(len(velocities)==0):
            num_velo = 1
        else:
            num_velo = len(velocities)

        avg_velocities.append(sum_velo/num_velo)
```

[8]:
```
avg_velo(trimmed_taxi_data)
```

[9]:
```
#Mapping Call Type Letters to Numbers
letter_to_num = {
    "A" : 1,
    "B" : 2,
    "C" : 3
}
num_to_letter = {
    1 : "A",
    2 : "B",
    3 : "C"
}

duration = trimmed_taxi_data["LEN"].tolist()

hour = trimmed_taxi_data["HR"].tolist()
month = trimmed_taxi_data["MON"].tolist()
week = trimmed_taxi_data["WK"].tolist()
day = trimmed_taxi_data["DAY"].tolist()
calltype = trimmed_taxi_data["CALL_TYPE"].tolist()
taxi = trimmed_taxi_data["TAXI_ID"].tolist()
origin = trimmed_taxi_data["ORIGIN_STAND"].tolist()

for count in range(0, len(calltype), 1):
    calltype[count] = (letter_to_num[calltype[count]])
```

[10]:
```
#Combine Input Vectors
inputs = []
for count in range(0, len(hour), 1):
    inputs.append([hour[count], month[count], week[count], day[count],␣
 ↪calltype[count], origin[count]])

dur_inputs = []
for count in range(0, len(hour), 1):
    dur_inputs.append([hour[count], month[count], week[count], day[count],␣
 ↪calltype[count], origin[count], avg_velocities[count]])
```

[11]:
```
import torch
from torch.utils.data import TensorDataset, DataLoader

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

0.0.2 Model

[12]:
```
velo_in_tensor = torch.tensor(inputs, dtype=torch.float32).to(device)
dur_in_tensor = torch.tensor(dur_inputs, dtype=torch.float32).to(device)
target_tensor = torch.tensor(duration, dtype=torch.float32).to(device)
target_velocity_tensor = torch.tensor(avg_velocities, dtype=torch.float32).
 ↪to(device)

velo_dataset = TensorDataset(velo_in_tensor, target_velocity_tensor)
duration_dataset = TensorDataset(dur_in_tensor, target_tensor)
```

[13]:
```
class MLR(torch.nn.Sequential):
    # Object Constructor
    def __init__(self, input_features, output_features):
        super().__init__()
        self.linear = torch.nn.Linear(input_features, 12)
        self.dropout = torch.nn.Dropout(0.5)
        self.linear2 = torch.nn.Linear(12, 16)
        self.linear3 = torch.nn.Linear(16, 20)
        self.linear4 = torch.nn.Linear(20, output_features)
        self.relu = torch.nn.ReLU()
        self.hiddennorm1 = torch.nn.BatchNorm1d(12)
        self.hiddennorm2 = torch.nn.BatchNorm1d(16)
        self.norm = torch.nn.BatchNorm1d(input_features)

    # define the forward function for prediction
    def forward(self, x):
        x = self.norm(x)
        x = self.dropout(self.relu(self.linear(x)))
        x = self.hiddennorm1(x)
        x = self.dropout(self.relu(self.linear2(x)))
        x = self.hiddennorm2(x)
        x = self.dropout(self.relu(self.linear3(x)))
        x = self.relu(self.linear4(x))
        return x

predict_velocity = MLR(6, 1).to(device)
predict_duration = MLR(7, 1).to(device)

print(predict_velocity)
print(predict_duration)
```

```
MLR(
  (linear): Linear(in_features=6, out_features=12, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=12, out_features=16, bias=True)
  (linear3): Linear(in_features=16, out_features=20, bias=True)
  (linear4): Linear(in_features=20, out_features=1, bias=True)
```

```
  (relu): ReLU()
  (hiddennorm1): BatchNorm1d(12, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm2): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (norm): BatchNorm1d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
MLR(
  (linear): Linear(in_features=7, out_features=12, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=12, out_features=16, bias=True)
  (linear3): Linear(in_features=16, out_features=20, bias=True)
  (linear4): Linear(in_features=20, out_features=1, bias=True)
  (relu): ReLU()
  (hiddennorm1): BatchNorm1d(12, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm2): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (norm): BatchNorm1d(7, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

[14]:
```python
class MLR_highdim(torch.nn.Sequential):
    # Object Constructor
    def __init__(self, input_features, output_features):
        super().__init__()
        self.linear1 = torch.nn.Linear(input_features, 128)
        self.linear2 = torch.nn.Linear(128, 64)
        self.linear3 = torch.nn.Linear(64, 16)
        self.linear4 = torch.nn.Linear(16, 8)
        self.linear5 = torch.nn.Linear(8, output_features)

        self.relu = torch.nn.ReLU()

        self.dropout = torch.nn.Dropout(0.3)

        self.norm = torch.nn.BatchNorm1d(input_features)
        self.hiddennorm1 = torch.nn.BatchNorm1d(128)
        self.hiddennorm2 = torch.nn.BatchNorm1d(64)
        self.hiddennorm3 = torch.nn.BatchNorm1d(16)

    # define the forward function for prediction
    def forward(self, x):
        x = self.norm(x)
        x = self.dropout(self.relu(self.linear1(x)))
        x = self.hiddennorm1(x)
```

```
        x = self.dropout(self.relu(self.linear2(x)))
        x = self.hiddennorm2(x)
        x = self.dropout(self.relu(self.linear3(x)))
        x = self.hiddennorm3(x)
        x = self.dropout(self.relu(self.linear4(x)))
        x = self.relu(self.linear5(x))
        return x

predict_velocity_hd = MLR_highdim(6, 1).to(device)
predict_duration_hd = MLR_highdim(7, 1).to(device)

print(predict_velocity_hd)
print(predict_duration_hd)
```

```
MLR_highdim(
  (linear1): Linear(in_features=6, out_features=128, bias=True)
  (linear2): Linear(in_features=128, out_features=64, bias=True)
  (linear3): Linear(in_features=64, out_features=16, bias=True)
  (linear4): Linear(in_features=16, out_features=8, bias=True)
  (linear5): Linear(in_features=8, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.3, inplace=False)
  (norm): BatchNorm1d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm3): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
MLR_highdim(
  (linear1): Linear(in_features=7, out_features=128, bias=True)
  (linear2): Linear(in_features=128, out_features=64, bias=True)
  (linear3): Linear(in_features=64, out_features=16, bias=True)
  (linear4): Linear(in_features=16, out_features=8, bias=True)
  (linear5): Linear(in_features=8, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.3, inplace=False)
  (norm): BatchNorm1d(7, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (hiddennorm3): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

)

[15]:
```
# Define optimizer (this will perform your parameter updates use)
lr = 0.0001
slr = 1e-4
#opt_velocity = torch.optim.Adam(predict_velocity.parameters(), lr=lr)
#opt_duration = torch.optim.Adam(predict_duration.parameters(), lr=lr)
opt_velocity_hd = torch.optim.SGD(predict_velocity_hd.parameters(), lr=lr)
opt_duration_hd = torch.optim.SGD(predict_duration_hd.parameters(), lr=lr)

#opt_velocity = torch.optim.Adam(predict_velocity.parameters(), lr=lr)
#opt_duration = torch.optim.Adam(predict_duration.parameters(), lr=lr)
```

0.0.3 Train

[16]:
```
batch_size = 64

train_err = []
parameters = []

velocitytrainloader = DataLoader(velo_dataset, batch_size, shuffle=True)
trainloader = DataLoader(duration_dataset, batch_size, shuffle=True)
```

[17]:
```
def train(epochs, model, optimize, loader):
    for epoch in range(epochs):
        for x, y in loader:
            model.train()
            prediction = model(x)
            loss = torch.sqrt(torch.nn.functional.mse_loss(prediction, torch.
 ↪unsqueeze(y, 1)))
            optimize.zero_grad()
            loss.backward()
            optimize.step()

        print("Epoch: " + str(epoch) + "\t" + "Loss: " + str(loss.tolist()))
```

[18]:
```
epochsV = 20
epochsD = 50
train(epochsV, predict_velocity_hd, opt_velocity_hd, velocitytrainloader)
train(epochsD, predict_duration_hd, opt_duration_hd, trainloader) #Best so far

#train(epochsV, predict_velocity, opt_velocity, velocitytrainloader)
#train(epochsD, predict_duration, opt_duration, trainloader) #Best so far
```

```
Epoch: 0        Loss: 4.13284969329834
Epoch: 1        Loss: 2.9492053985595703
Epoch: 2        Loss: 4.038605690002441
```

```
Epoch: 3        Loss: 3.3954899311065674
Epoch: 4        Loss: 4.525208950042725
Epoch: 5        Loss: 4.651245594024658
Epoch: 6        Loss: 4.604705810546875
Epoch: 7        Loss: 3.895814895629883
Epoch: 8        Loss: 3.986433744430542
Epoch: 9        Loss: 3.660259246826172
Epoch: 10       Loss: 2.8639087677001953
Epoch: 11       Loss: 3.572251558303833
Epoch: 12       Loss: 4.181982040405273
Epoch: 13       Loss: 4.146925449371338
Epoch: 14       Loss: 5.921876430511475
Epoch: 15       Loss: 3.9241158962249756
Epoch: 16       Loss: 3.593061923980713
Epoch: 17       Loss: 5.42855978012085
Epoch: 18       Loss: 3.128194570541382
Epoch: 19       Loss: 5.916820526123047
Epoch: 0        Loss: 854.5045166015625
Epoch: 1        Loss: 424.1079406738281
Epoch: 2        Loss: 437.63446044921875
Epoch: 3        Loss: 390.8373107910156
Epoch: 4        Loss: 497.8941345214844
Epoch: 5        Loss: 312.143310546875
Epoch: 6        Loss: 385.2196044921875
Epoch: 7        Loss: 378.6519775390625
Epoch: 8        Loss: 361.8486633300781
Epoch: 9        Loss: 403.2025146484375
Epoch: 10       Loss: 315.97808837890625
Epoch: 11       Loss: 369.1615295410156
Epoch: 12       Loss: 579.7718505859375
Epoch: 13       Loss: 449.4812316894531
Epoch: 14       Loss: 359.3123474121094
Epoch: 15       Loss: 471.12640380859375
Epoch: 16       Loss: 442.71484375
Epoch: 17       Loss: 385.62890625
Epoch: 18       Loss: 443.7497863769531
Epoch: 19       Loss: 474.4811096191406
Epoch: 20       Loss: 476.165771484375
Epoch: 21       Loss: 433.75970458984375
Epoch: 22       Loss: 427.1034240722656
Epoch: 23       Loss: 326.68829345703125
Epoch: 24       Loss: 534.5984497070312
Epoch: 25       Loss: 354.1719970703125
Epoch: 26       Loss: 257.80145263671875
Epoch: 27       Loss: 434.271484375
Epoch: 28       Loss: 397.7559509277344
Epoch: 29       Loss: 523.023193359375
Epoch: 30       Loss: 316.5309143066406
```

```
Epoch: 31       Loss: 502.1172790527344
Epoch: 32       Loss: 381.8683776855469
Epoch: 33       Loss: 504.8301086425781
Epoch: 34       Loss: 357.6747131347656
Epoch: 35       Loss: 407.3233947753906
Epoch: 36       Loss: 448.68878173828125
Epoch: 37       Loss: 506.8689270019531
Epoch: 38       Loss: 340.9544982910156
Epoch: 39       Loss: 500.2999572753906
Epoch: 40       Loss: 447.43072509765625
Epoch: 41       Loss: 325.0626525878906
Epoch: 42       Loss: 405.0512390136719
Epoch: 43       Loss: 447.7801818847656
Epoch: 44       Loss: 478.7785339355469
Epoch: 45       Loss: 408.60858154296875
Epoch: 46       Loss: 542.0421752929688
Epoch: 47       Loss: 364.03973388671875
Epoch: 48       Loss: 255.4182891845703
Epoch: 49       Loss: 372.7362976074219
```

[19]:
```
torch.save(predict_velocity_hd, "VelocityPredictor.pt")
torch.save(predict_duration_hd, "DurationPredictor.pt")
```

[20]:
```
#epochs = 10
#train(epochs, predict_velocity, opt_velocity, velocitytrainloader)
```

[21]:
```
#train(epochs, predict_duration, opt_duration, trainloader) #Best so far
```

[71]:
```
#predict_without_velocity = MLR(6, 1).to(device)
#opt_without_velocity = torch.optim.Adam(predict_without_velocity.parameters(),␣
 ↪lr=lr)
#without_velo_dataset = TensorDataset(velo_in_tensor, target_tensor)
#withoutvelocitytrainloader = DataLoader(without_velo_dataset, batch_size,␣
 ↪shuffle=True)
#train(epochs, predict_without_velocity, opt_without_velocity,␣
 ↪withoutvelocitytrainloader)
```

1 PREDICT

[45]:
```
#Read into Dataframe
test_data = pd.read_csv("kaggle_data/test_public.csv")
test_data['ORIGIN_STAND'] = taxi_data['ORIGIN_STAND'].fillna(0)
test_data[["YR", "MON", "DAY", "HR", "WK"]] = taxi_data[["TIMESTAMP"]].
 ↪apply(parse_timestamp, axis=1, result_type="expand")

test_hour = test_data["HR"].tolist()
```

```
test_month = test_data["MON"].tolist()
test_week = test_data["WK"].tolist()
test_day = test_data["DAY"].tolist()
test_calltype = test_data["CALL_TYPE"].tolist()
test_taxi = test_data["TAXI_ID"].tolist()
test_origin = test_data["ORIGIN_STAND"].tolist()

for count in range(0, len(test_calltype), 1):
    test_calltype[count] = (letter_to_num[test_calltype[count]])

test_inputs = []
for count in range(0, len(test_hour), 1):
    test_inputs.append([test_hour[count], test_month[count], test_week[count],␣
 ↪test_day[count], test_origin[count], test_calltype[count]])

test_tensor = torch.tensor(test_inputs, dtype=torch.float32).to(device)

test_dataset = TensorDataset(test_tensor)
testloader = DataLoader(test_dataset, batch_size, shuffle=True)
```

[73]:
```
#test_ids = test_data["TRIP_ID"].tolist()
#test_velo = predict_velocity(test_tensor)

#velo = []
#for i in test_velo.tolist():
#    velo.append(i[0])

#velo_test_inputs = []
#for count in range(0, len(test_hour), 1):
#    velo_test_inputs.append([test_hour[count], test_month[count],␣
 ↪test_week[count], test_day[count], test_calltype[count], test_taxi[count],␣
 ↪velo[count]])

#velo_test_tensor = torch.tensor(velo_test_inputs, dtype=torch.float32).
 ↪to(device)

#test_duration = predict_duration(velo_test_tensor)
#test_duration = test_duration.tolist()

#for i in range (0, len(test_ids), 1):
#    print("\""+str(test_ids[i])+"\"","+str(test_duration[i][0]))
```

[74]:
```
#test_duration_without_velocity = predict_without_velocity(test_tensor)
#test_duration_without_velocity = test_duration_without_velocity.tolist()

#print(len(test_ids))
```

```
#for i in range (0, len(test_ids), 1):
#    ␣
 ↪print("\""+str(test_ids[i])+"\"","+str(test_duration_without_velocity[i][0]))
```

[48]:
```
predict_velocity.eval()
predict_duration.eval()

test_ids = test_data["TRIP_ID"].tolist()
test_velo = predict_velocity(test_tensor)

velo_hd = []
for i in test_velo.tolist():
    velo_hd.append(i[0])

velo_test_inputs = []
for count in range(0, len(test_hour), 1):
    velo_test_inputs.append([test_hour[count], test_month[count],␣
 ↪test_week[count], test_day[count], test_calltype[count], test_origin[count],␣
 ↪velo_hd[count]])

velo_hd_test_tensor = torch.tensor(velo_test_inputs, dtype=torch.float32).
 ↪to(device)

test_duration_hd = predict_duration(velo_hd_test_tensor)
test_duration_hd = test_duration_hd.tolist()

for i in range (0, len(test_ids), 1):
    print("\""+str(test_ids[i])+"\"","+str(test_duration_hd[i][0]))
```

```
"T1",624.7883911132812
"T2",620.5599975585938
"T3",624.7883911132812
"T4",624.7883911132812
"T5",624.7883911132812
"T6",627.220703125
"T7",624.7883911132812
"T8",627.220703125
"T9",624.7883911132812
"T10",624.7883911132812
"T11",624.7883911132812
"T12",617.8628540039062
"T13",617.8628540039062
"T14",617.8628540039062
"T15",617.8628540039062
"T16",596.8775634765625
"T17",584.4378051757812
"T18",617.8628540039062
```

```
"T19",624.7883911132812
"T20",617.8628540039062
"T21",624.7883911132812
"T22",627.220703125
"T23",627.220703125
"T24",582.2228393554688
"T25",624.7883911132812
"T26",624.7883911132812
"T27",624.7883911132812
"T28",626.2550048828125
"T29",573.1555786132812
"T30",626.2550048828125
"T31",624.7883911132812
"T32",624.7883911132812
"T33",617.8628540039062
"T34",617.8628540039062
"T35",617.8628540039062
"T36",617.8628540039062
"T37",627.220703125
"T38",624.7883911132812
"T39",592.5403442382812
"T40",624.7883911132812
"T41",593.4191284179688
"T42",617.8628540039062
"T43",617.8628540039062
"T44",624.7883911132812
"T45",624.7883911132812
"T46",624.7883911132812
"T47",617.8628540039062
"T48",621.431640625
"T49",606.8375854492188
"T50",626.2550048828125
"T51",598.7593383789062
"T52",624.7883911132812
"T53",606.8375854492188
"T54",624.7883911132812
"T55",575.1022338867188
"T56",624.7883911132812
"T57",597.8797607421875
"T58",624.7883911132812
"T59",602.2409057617188
"T60",594.713623046875
"T61",626.2550048828125
"T62",617.8628540039062
"T63",621.431640625
"T64",623.8051147460938
"T65",624.7883911132812
"T66",590.029052734375
```

"T67",624.7883911132812
"T68",631.1296997070312
"T69",617.8628540039062
"T70",626.2550048828125
"T71",621.431640625
"T72",621.431640625
"T73",626.2550048828125
"T74",617.8628540039062
"T75",626.2550048828125
"T76",624.9984130859375
"T77",631.1910400390625
"T78",597.7302856445312
"T79",617.8628540039062
"T80",617.8628540039062
"T81",627.220703125
"T82",624.7883911132812
"T83",653.0037231445312
"T84",631.1296997070312
"T85",624.7883911132812
"T86",627.220703125
"T87",617.8628540039062
"T88",617.8628540039062
"T90",624.7883911132812
"T91",631.1296997070312
"T92",602.85205078125
"T93",621.431640625
"T94",621.431640625
"T95",653.0037231445312
"T96",624.7883911132812
"T98",617.8628540039062
"T99",653.0037231445312
"T100",653.0037231445312
"T101",586.25439453125
"T102",621.431640625
"T103",624.7883911132812
"T104",624.7883911132812
"T107",617.8628540039062
"T109",626.2550048828125
"T110",623.9822387695312
"T111",627.220703125
"T112",626.2550048828125
"T113",624.7883911132812
"T114",626.2550048828125
"T115",627.220703125
"T116",631.1296997070312
"T117",602.5510864257812
"T118",580.2774658203125
"T119",583.6659545898438

```
"T120",653.0037231445312
"T121",598.5970458984375
"T122",631.1910400390625
"T123",587.4086303710938
"T124",624.9984130859375
"T125",631.1910400390625
"T126",631.1296997070312
"T127",577.8555908203125
"T128",628.7254638671875
"T129",625.5889282226562
"T130",624.7883911132812
"T131",624.9984130859375
"T132",636.0429077148438
"T133",653.0037231445312
"T134",617.8628540039062
"T135",613.8649291992188
"T136",579.6318359375
"T137",631.1910400390625
"T138",617.8628540039062
"T139",597.7302856445312
"T140",624.9984130859375
"T141",580.3570556640625
"T142",624.7883911132812
"T143",624.7883911132812
"T144",631.1296997070312
"T145",589.4658813476562
"T146",611.01318359375
"T147",626.2550048828125
"T148",653.0037231445312
"T149",631.1910400390625
"T151",606.4459838867188
"T152",653.0037231445312
"T153",601.4736938476562
"T154",624.9984130859375
"T155",631.1910400390625
"T156",631.1910400390625
"T157",625.2571411132812
"T158",631.1296997070312
"T159",624.9984130859375
"T160",653.0037231445312
"T161",591.5228271484375
"T162",626.2550048828125
"T163",626.2550048828125
"T164",676.9107666015625
"T166",631.1910400390625
"T167",582.8139038085938
"T168",611.7853393554688
"T169",579.103271484375
```

```
"T170",653.0037231445312
"T171",608.5689086914062
"T172",582.3717651367188
"T173",626.2550048828125
"T174",570.6784057617188
"T175",598.5938720703125
"T176",602.186767578125
"T177",623.9822387695312
"T178",648.8450927734375
"T179",616.1883544921875
"T180",636.0429077148438
"T181",606.8130493164062
"T182",648.8450927734375
"T183",596.8775634765625
"T184",606.8130493164062
"T185",611.7853393554688
"T186",676.9107666015625
"T187",613.5406494140625
"T188",648.8450927734375
"T189",624.7883911132812
"T190",599.9088745117188
"T191",636.0429077148438
"T192",624.7883911132812
"T193",580.3570556640625
"T194",676.9107666015625
"T195",631.1910400390625
"T196",648.8450927734375
"T197",606.6560668945312
"T198",676.9107666015625
"T199",582.8139038085938
"T200",653.0037231445312
"T201",636.0429077148438
"T202",606.8130493164062
"T203",636.0429077148438
"T204",602.186767578125
"T205",648.8450927734375
"T206",636.0429077148438
"T207",613.8649291992188
"T208",602.186767578125
"T209",676.9107666015625
"T210",648.8450927734375
"T211",611.01318359375
"T212",624.9984130859375
"T213",648.8450927734375
"T214",648.8450927734375
"T215",653.0037231445312
"T216",653.0037231445312
"T217",621.431640625
```

```
"T218",648.8450927734375
"T219",631.1910400390625
"T220",636.0429077148438
"T221",631.1910400390625
"T222",648.8450927734375
"T223",590.7828979492188
"T224",631.1910400390625
"T225",584.8052368164062
"T226",648.8450927734375
"T227",611.2921142578125
"T228",659.6390991210938
"T229",624.9984130859375
"T230",629.9693603515625
"T231",624.9984130859375
"T232",593.9093017578125
"T233",597.0070190429688
"T234",636.0429077148438
"T235",589.4658813476562
"T236",648.8450927734375
"T237",624.9984130859375
"T238",631.1910400390625
"T239",636.0429077148438
"T240",636.0429077148438
"T241",636.0429077148438
"T242",593.4169921875
"T243",617.8628540039062
"T244",659.6390991210938
"T245",616.815673828125
"T246",636.0429077148438
"T247",648.8450927734375
"T248",603.9527587890625
"T249",659.6390991210938
"T250",674.1363525390625
"T251",676.9107666015625
"T252",625.6486206054688
"T253",636.0429077148438
"T254",617.6331787109375
"T255",636.0429077148438
"T256",636.0429077148438
"T257",648.8450927734375
"T258",588.6917114257812
"T259",636.0429077148438
"T260",667.8104248046875
"T261",603.9483642578125
"T262",626.6279296875
"T263",617.8628540039062
"T264",612.01806640625
"T265",581.0181274414062
```

```
"T266",618.059326171875
"T267",659.6390991210938
"T268",659.6390991210938
"T269",659.6390991210938
"T270",631.5888061523438
"T271",659.6390991210938
"T272",659.6390991210938
"T273",674.1363525390625
"T274",682.0025634765625
"T275",636.0429077148438
"T276",659.6390991210938
"T277",659.6390991210938
"T278",601.2545776367188
"T279",621.431640625
"T280",636.0429077148438
"T281",631.1910400390625
"T282",636.0429077148438
"T283",674.1363525390625
"T284",583.4685668945312
"T285",659.6390991210938
"T286",659.6390991210938
"T287",659.6390991210938
"T288",636.0429077148438
"T289",624.9984130859375
"T290",584.1752319335938
"T291",674.1363525390625
"T292",659.6390991210938
"T293",627.220703125
"T294",636.0429077148438
"T295",682.0025634765625
"T296",591.5228271484375
"T297",636.0429077148438
"T298",593.870361328125
"T299",659.6390991210938
"T300",687.6590576171875
"T301",682.0025634765625
"T302",687.6590576171875
"T303",669.9996948242188
"T304",687.8783569335938
"T305",674.1363525390625
"T306",659.6390991210938
"T307",682.0025634765625
"T308",659.6390991210938
"T309",612.01806640625
"T310",676.9107666015625
"T311",682.0025634765625
"T312",608.9841918945312
"T313",607.7596435546875
```

```
"T314",687.6590576171875
"T315",687.8783569335938
"T316",617.8628540039062
"T317",599.7437744140625
"T318",604.9571533203125
"T319",648.5307006835938
"T320",636.5235595703125
"T321",659.6390991210938
"T322",659.6390991210938
"T323",687.8783569335938
"T324",674.1363525390625
"T325",636.0429077148438
"T326",600.5679321289062
"T327",687.6590576171875
```