

Supplementary Information

Generating Travel Diary from National Household Travel Survey to Analyze Trip Times

In the National Household Travel survey, trip-level information is reported in a detailed format, with each trip having its row. However, to understand travel behavior comprehensively, it is crucial to summarize this trip information at the person or household level. A travel diary is created for each household member based on the trip file.

The R Statistical software (v4.3.0) is used to generate the travel diary. The NHTS data is imported into R and then the trip information is transformed from a longer format to a wider format using the “tidyverse” library. First, the households that own an EV are separated from households that do not own an electric vehicle (EV). This is done by filtering out the households using the FUEL type column.

```
EV_Vehicle<-subset(Vehicle, FUELTYPE == 3)
NonEV_Vehicle<-subset(Vehicle, FUELTYPE != 3)
```

Then, the trip data corresponding to each household is separated from the NHTS trip file to ensure that the travel diary for EVs and non EVs are separated.

```
filtered_EV <- Trip[, c("HOUSEID", "PERSONID", "VEHID",
"STRTTIME", "ENDTIME", "TDTRPNUM")]
unique_houseid <- unique(EV_Vehicle$HOUSEID)
subset_df2 <- filtered_EV[filtered_EV$HOUSEID %in%
unique_houseid, ]
```

The transformation then involves considering the number of trips, the start and end times, and the vehicle ID for each respondent. The household and personal characteristics are merged with the travel diary along with formatting the time format to ensure consistency. This integration ensures that relevant information from the household and personal files is combined with the travel data.

```
subset_df2$STRTTIME <- sprintf("%02d:%02d",
subset_df2$STRTTIME %/% 100, subset_df2$STRTTIME %% 100)
subset_df2$ENDTIME <- sprintf("%02d:%02d",
subset_df2$ENDTIME %/% 100, subset_df2$ENDTIME %% 100)
final_df <- subset_df2 %>%
  pivot_wider(names_from = TDTRPNUM,
              values_from = c(STRTTIME, ENDTIME, VEHID),
              names_prefix = "TRIP",
```

```

        values_fn = list(STRTTIME = first, ENDTIME =
first, VEHID = first))
merged_df <- merge(final_df, Household[, c("HOUSEID",
"HHFAMINC",
"HHVEHCNT", "HHSIZE", "HHSTATE", "HH_RACE", "HH_HISP", "HH_CBSA"
, "HBHUR" )], by = "HOUSEID")
merged_df2 <- merge(merged_df, Person[,
c("HOUSEID", "PERSONID", "R_AGE_IMP", "R_SEX_IMP", "DRIVER"
)], by = c("HOUSEID", "PERSONID"))

```

The comprehensive dataset includes the summarized travel information and the corresponding household and personal characteristics.

Calculating Occupancy Rate

EV vs Non-EV Households

The occupancy rate is then calculated to understand the occupancy rates across EV-owning households and non-EV-owning households. The datasets generated from the travel diary for both EVs and non-EVs are loaded. Unique household IDs are identified and randomly shuffled.

```

NHTS_EV_df = pd.read_csv(working_path +
'/data/NHTS_EV.csv', low_memory=False)
NHTS_non_EV_df = pd.read_csv(working_path + '/data/Non-EV
dataset.csv', low_memory=False)

EV_home_ID_unique = NHTS_EV_df.HOUSEID.unique()
random.shuffle(EV_home_ID_unique)
number_of_EV_home = len(EV_home_ID_unique)

non_EV_home_ID_unique = NHTS_non_EV_df.HOUSEID.unique()
random.shuffle(non_EV_home_ID_unique)
number_of_non_EV_home = len(non_EV_home_ID_unique)

total_number_of_home = number_of_EV_home +
number_of_non_EV_home

```

A function “process_NHTS” then calculates a minute-by-minute occupancy schedule for each household. When no person in a household is identified on a trip, it is assumed that that house has an occupancy.

```

def process_NHTS(NHTS_df, HOUSEID): house_attribute_list =
['HOUSEID', 'HHFAMINC', 'HHVEHCNT', 'HHSIZE', 'HHSTATE',
'HH_RACE', 'HH_HISP', 'HH_CBSA', 'HBHUR']

people_attribute_list = ['PERSONID', 'R_AGE_IMP',
'R_SEX_IMP', 'DRIVER']

building_dictionary = NHTS_df.loc[NHTS_df.HOUSEID ==
HOUSEID][house_attribute_list].iloc[0,:].to_dict()

people_df = NHTS_df.loc[NHTS_df.HOUSEID ==
HOUSEID][people_attribute_list].set_index(['PERSONID'])

people_dictionary = {index: row.to_dict() for index, row in
people_df.iterrows()}

building_dictionary['People'] = people_dictionary

start_time_df = NHTS_df.loc[NHTS_df.HOUSEID ==
HOUSEID].filter(like='STRTTIME_TRIP')

end_time_df = NHTS_df.loc[NHTS_df.HOUSEID ==
HOUSEID].filter(like='ENDTIME_TRIP')

schedule_array_sum = 0

for (index_start, row_start), (index_end, row_end) in
zip(start_time_df.iterrows(), end_time_df.iterrows()):
    schedule_array = 0 for (start, stop) in
zip(row_start.dropna(), row_end.dropna()):
        schedule_temp_array =
np.array(start_stop_schedule(start, stop))

        schedule_array = schedule_array + schedule_temp_array
        schedule_array = -schedule_array + 1 # Invert to get
occupancy (1 when home) schedule_array_sum =

pd.DataFrame([schedule_array_sum], index =
['occupancy']).

    T return building_dictionary, schedule_array_sum

```

The code then iterates through the first 5000 unique households in both the EV and non-EV datasets (i.e., if rerun is True). For each household, it uses the “process_NHTS” function to calculate the occupancy schedule, aggregates it to an hourly level by taking the mean of the

minute-by-minute data, and normalizes it by the household size to get an average hourly occupancy rate per person.

```
rerun = False

if rerun == True:
    HOUSEID_occupancy_hourly_summary = pd.DataFrame([])
    counter = 0
    for HOUSEID in EV_home_ID_unique[0:5000]:
        counter = counter + 1
        if counter%100 == 0:
            print(f"{counter}/{len(EV_home_ID_unique)}")
        try:
            HOUSEID_results = process_NHTS(NHTS_EV_df,
HOUSEID)

            HOUSEID_info = HOUSEID_results[0]
            HOUSEID_occupancy = HOUSEID_results[1]
            HOUSEID_occupancy_hourly =
HOUSEID_occupancy.groupby(HOUSEID_occupancy.index //
60).mean()

            HOUSEID_occupancy_hourly =
pd.concat([HOUSEID_occupancy_hourly]*365,
ignore_index=True)/HOUSEID_info['HHSIZE']
            HOUSEID_occupancy_hourly.columns =
[str(HOUSEID)]
            HOUSEID_occupancy_hourly_summary =
pd.concat([HOUSEID_occupancy_hourly_summary,
HOUSEID_occupancy_hourly], axis = 1)
        except:
            print(f'fail: {HOUSEID}')

    EV_HOUSEID_occupancy_hourly_summary =
HOUSEID_occupancy_hourly_summary

EV_HOUSEID_occupancy_hourly_summary.to_csv('data/EV_HOUSEID
_occupancy_hourly_summary.csv', index = None)
else:
    EV_HOUSEID_occupancy_hourly_summary =
pd.read_csv('data/EV_HOUSEID_occupancy_hourly_summary.csv')
    print('Since rerun = False, read data from local file')
```

Short Commute vs Long Commute

Similar to the EV versus non-EV analysis, households owning an EV are further divided into long-commute and short-commute groups based on commute times from the NHTS data.

Vehicle occupancy is measured as the hourly percentage of time the EV was detected at home, as well as the hourly percentage it was absent during working hours. These values are calculated by averaging minute-level presence data and normalizing by household size. The distinction between long- and short-commute households is made using total vehicle occupancy percentiles, capturing differences in daily vehicle availability patterns related to commute length.

Short-Commute Households

```
short_commute_ID =
EV_HOUSEID_occupancy_hourly_summary.sum(0).sort_values()[30
00:4000].index.tolist()
EV_HOUSEID_occupancy_hourly_summary_model_short_commute =
EV_HOUSEID_occupancy_hourly_summary[short_commute_ID].iloc[
0:24]

for HOUSEID in
EV_HOUSEID_occupancy_hourly_summary_model_short_commute:
    hour_of_day =
EV_HOUSEID_occupancy_hourly_summary_model_short_commute.ind
ex.tolist()
    family_size = int(process_NHTS(NHTS_EV_df,
int(HOUSEID))[0]['HHSIZE'])

    df_temp = pd.DataFrame({
        'hour_of-day': hour_of_day,
        'family_size': family_size,
        'occupancy_percent':
EV_HOUSEID_occupancy_hourly_summary_model_short_commute[HOU
SEID]
    })

    short_commute_model_data =
pd.concat([short_commute_model_data, df_temp], axis=0)
```

Long-Commute Households

```
long_commute_ID =
EV_HOUSEID_occupancy_hourly_summary.sum(0).sort_values()[10
00:2000].index.tolist()
EV_HOUSEID_occupancy_hourly_summary_model_long_commute =
EV_HOUSEID_occupancy_hourly_summary[long_commute_ID].iloc[0
:24]
for HOUSEID in
EV_HOUSEID_occupancy_hourly_summary_model_long_commute:
```

```

    hour_of_day =
EV_HOUSEID_occupancy_hourly_summary_model_long_commute.index.
x.tolist()
    family_size = int(process_NHTS(NHTS_EV_df,
int(HOUSEID))[0]['HHSIZE'])

    df_temp = pd.DataFrame({
        'hour_of-day': hour_of_day,
        'family_size': family_size,
        'occupancy_percent':
EV_HOUSEID_occupancy_hourly_summary_model_long_commute[HOUS
EID]
    })

    long_commute_model_data =
pd.concat([long_commute_model_data, df_temp], axis=0)

```

To calculate the occupancy rate, a simple feedforward neural network using PyTorch is applied. The input features are the hour of the day and family size, while the target output is the occupancy percentage. The model consists of a single hidden layer with 30 neurons and ReLU activation. Training is conducted for 1000 epochs using the Adam optimizer and mean squared error loss. The learning rate is set at 0.01.

```

class FeedforwardNeuralNetModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        return self.fc2(self.relu(self.fc1(x)))

model = FeedforwardNeuralNetModel(input_dim=2,
hidden_dim=30, output_dim=1)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(2000):
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

```

```
if (epoch+1) % 100 == 0:  
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

Once the occupancy rates are calculated for each commuter type, as well as EV and non-EV households, the occupancy rate is then used to model and simulate hourly cooling load profile using EnergyPlus v9.5.