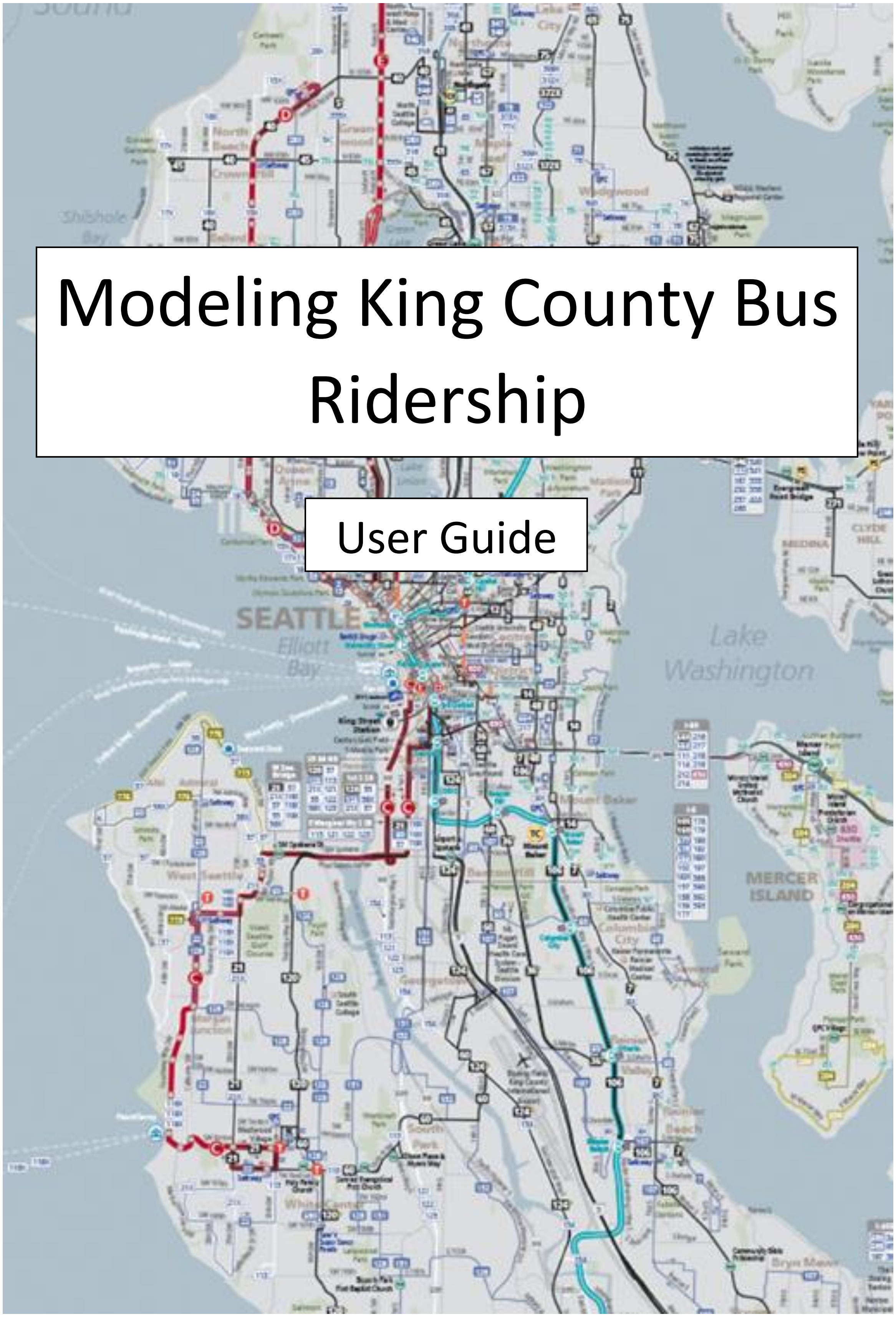
Modeling King County Bus Ridership

User Guide



Setup Environment:

1. Clone the repository:

```
>> git clone https://github.com/jacobw125/uw-msds-trac-capstone.git
```

2. Create a new python environmet using the command:

```
>> conda env create UWTRAC
```

3. Activate UWTRAC by using the command:

```
>> conda activate UWTRAC
```

4. Install the required python packages using:

```
>> pip install –r requirements.txt
```

Create Training Sets:

Regenerate the training sets from 2019 winter and/or summer APC survey period:

Summer:

```
>> python3 pipeline/01_filter_apc.py Summer_APC_File True

>> python3 pipeline/02_filter_orca.py Summer_ORCA_FILE True

>> python3 pipeline/03_agg_orca.py True

>> python3 pipeline/04_merge.py True

>> python3 pipeline/05_create_training.py S
```

Winter:

```
>> python3 pipeline/01_filter_apc.py Winter_APC_File False
>> python3 pipeline/02_filter_orca.py Winter_ORCA_FILE  False

>> python3 pipeline/03_agg_orca.py False

>> python3 pipeline/04_merge.py False

>> python3 pipeline/05_create_training.py W
```

Combined:

Run steps 1-4 for for Summer and Winter, then:

```
>> python3 pipeline/05_create_training.py B
```

Generate new training sets from different APC survey period:

1. **Update constants in pipeline/constants.py:**

   a. Check column headers of new files for ORCA (`ORCA_SUMMER_COLUMNS, ORCA_WINTER_COLUMNS`) and APC (`APC_COLUMNS`) match constants. If not, either update constants.py or relabel your file.

   b. `SUMMER_DAYS` or `WINTER_DAYS` to desired date range. Possibly excluding extreme weather (i.e. snowstorm) that could potentially negatively impact your model.

2. **Run pipeline:**

   Note:

   For steps 1-4, the True/False parameter refers to is this is a summer survey data set, directing the programs to use `SUMMER` constants instead of `WINTER` constants.

   For step 5, S means use `SUMMER` constants and W means use `WINTER`. In order to use the parameter B you much run steps 1-4 on both a `WINTER` and `SUMMER` set to create the `COMBINED` training set.

```
>> python3 pipeline/01_filter_apc.py NEW_APC_FILE_PATH.csv  [True/False]

>> python3 pipeline/02_filter_orca.py NEW_ORCA_FILE_PATH.csv  [True/False]

>> python3 pipeline/03_agg_orca.py [True/False]

>> python3 pipeline/04_merge.py [True/False]
>> python3 pipeline/05_create_training.py [S/W/B]
```

Add weather data to the model:

1. **Get data:**

   a. If using the winter or summer 2019 survey periods, the weather files can be found here.
   b. Otherwise, you can get a new dataset from NOAA.

2. **Update constants in pipeline/constants.py:**

   a. Valid dates: `SUMMER_DAYS` or `WINTER_DAYS`
   b. Weather File: `WEATHER_FILE`

3. **Uncomment code in pipeline/05_create_training.py:**

```
## Add weather data
# print('Adding weather data')
# WEATHER = pd.read_csv(c.MERGE_DIR + TAG1 + '/' + c.WEATHER_FILE)
# WEATHER = WEATHER[WEATHER['REPORT_TYPE'] == 'FM-15']
# WEATHER['dt'] = WEATHER['DATE'].apply(lambda dt: datetime.strptime(dt, '%Y-%m-
%dT%H:%M:%S'))
# WEATHER['date_part'] = WEATHER['dt'].apply(lambda dt: dt.strftime('%Y-%m-%d'))
# WEATHER['hour_part'] = WEATHER['dt'].apply(lambda dt: dt.strftime('%H'))
#
# WEATHER_COLS = ['HourlyDryBulbTemperature', 'HourlyPrecipitation',
#               'HourlyRelativeHumidity', 'HourlySeaLevelPressure', 'HourlyWindSpeed']
# WEATHER = WEATHER[['date_part', 'hour_part'] + WEATHER_COLS]
#
# WEATHER['HourlyDryBulbTemperature'] =
WEATHER.HourlyDryBulbTemperature.apply(func.remove_t_and_s)
# WEATHER['HourlyPrecipitation'] = WEATHER.HourlyPrecipitation.apply(func.remove_t_and_s)
# WEATHER['HourlyRelativeHumidity'] =
WEATHER.HourlyRelativeHumidity.apply(func.remove_t_and_s)
# WEATHER['HourlySeaLevelPressure'] =
WEATHER.HourlySeaLevelPressure.apply(func.remove_t_and_s)
# WEATHER['HourlyWindSpeed'] = WEATHER.HourlyWindSpeed.apply(func.remove_t_and_s)
#
# # mean inpute missing weather values, except precip which is assumed to be 0 when NA
# WEATHER['HourlyPrecipitation'] = WEATHER['HourlyPrecipitation'].fillna(0.)
# WEATHER['HourlyRelativeHumidity'] = func.mean_input(WEATHER['HourlyRelativeHumidity'])
# WEATHER['HourlySeaLevelPressure'] = func.mean_input(WEATHER['HourlySeaLevelPressure'])
# WEATHER['HourlyWindSpeed'] = func.mean_input(WEATHER['HourlyWindSpeed'])
#
# # In case I need to re-merge weather, here's how to drop those columns
# DATA.drop(columns=['HourlyDryBulbTemperature', 'HourlyPrecipitation',
'HourlyRelativeHumidity',
#               'HourlySeaLevelPressure', 'HourlyWindSpeed',], inplace=True)
# DATA = pd.merge(DATA, WEATHER, how='left', left_on=[c.APC_DATE, c.HDT],
#             right_on=['date_part', 'hour_part'])
# for col in WEATHER_COLS:
#     AGG_TYPES.add('col', 'mean')
```

4. **Run steps 1-5 in pipeline**

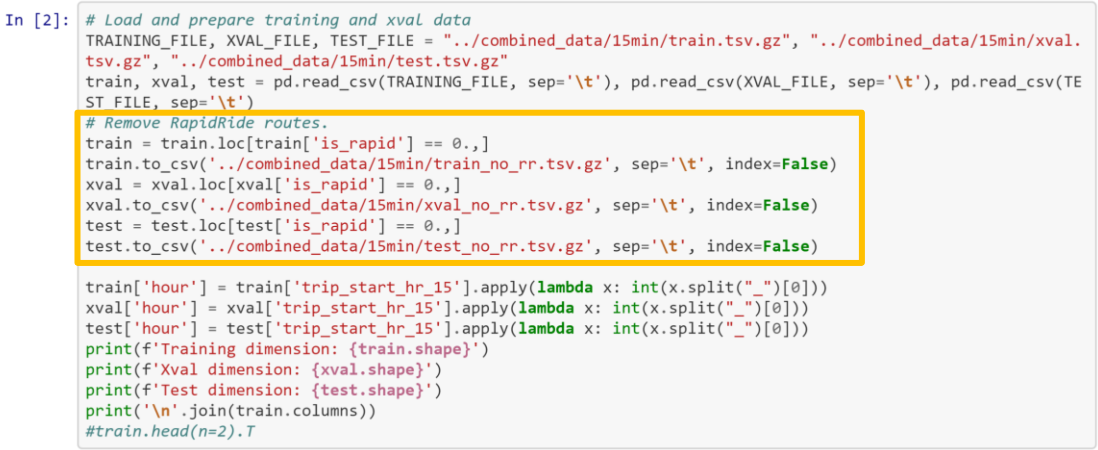
Training Model:

1. **Run code block 1, to import necessary Python packages.**

2. **Train the models:**

   15 Minute Aggregation: Step through code cells ***2-9*** of final_nn.ipynb.
   30 Minute Aggregation: Step through code cells ***11-15*** of final_nn.ipynb.
   1 Hour Aggregation Step through code cells ***17-22*** of final_nn.ipynb.

3. **If you would like your model to include rapid rides, comment out the below section in the second code block. Note this**



```
In [2]: # Load and prepare training and xval data
TRAINING_FILE, XVAL_FILE, TEST_FILE = "../combined_data/15min/train.tsv.gz", "../combined_data/15min/xval.
tsv.gz", "../combined_data/15min/test.tsv.gz"
train, xval, test = pd.read_csv(TRAINING_FILE, sep='\t'), pd.read_csv(XVAL_FILE, sep='\t'), pd.read_csv(TE
ST_FILE, sep='\t')
# Remove RapidRide routes.
train = train.loc[train['is_rapid'] == 0.,]
train.to_csv('../combined_data/15min/train_no_rr.tsv.gz', sep='\t', index=False)
xval = xval.loc[xval['is_rapid'] == 0.,]
xval.to_csv('../combined_data/15min/xval_no_rr.tsv.gz', sep='\t', index=False)
test = test.loc[test['is_rapid'] == 0.,]
test.to_csv('../combined_data/15min/test_no_rr.tsv.gz', sep='\t', index=False)

train['hour'] = train['trip_start_hr_15'].apply(lambda x: int(x.split("_")[0]))
xval['hour'] = xval['trip_start_hr_15'].apply(lambda x: int(x.split("_")[0]))
test['hour'] = test['trip_start_hr_15'].apply(lambda x: int(x.split("_")[0]))
print(f'Training dimension: {train.shape}')
print(f'Xval dimension: {xval.shape}')
print(f'Test dimension: {test.shape}')
print('\n'.join(train.columns))
#train.head(n=2).T
```