Multinomial Logit (MNL) Model Estimation

Adrita Islam & Raymond Gerte

In [1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import scipy.stats as stats
import pylogit as pl
from collections import OrderedDict
from scipy.stats import chisqprob

from patsy import dmatrices

%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:
```
aargau_data = pd.read_csv('./Trip_File_Aargau.csv')
```

Filter Data to only include adult work trips during the week

In [3]:
```
aargau_adult = aargau_data[aargau_data['age']>=18]
```

In [4]:
```
aargau_am_peak = aargau_adult[aargau_adult['tod']==1]
```

In [5]:
```
aargau_work = aargau_am_peak[aargau_am_peak['trip_pur']==1]
```

In [6]:
```
aargau_weekday = aargau_work[aargau_work['weekday']<= 5]
```

In [7]:
```
print ("Total number of rows in the dataset is = {0}").format(len(aargau_data))
print ("Total number of rows after restricting the dataset for work trips made by
```

```
Total number of rows in the dataset is = 17651
Total number of rows after restricting the dataset for work trips made by an ad
ult during AM peak in a weekday = 1172
```

Replace invalid values and remove them from the dataset

In [8]:
```
aargau_weekday["n_auto"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["n_bike"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["dis_work"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["n_trp_wk"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["mode_5"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["h_sub"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["o_sub"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["g_sub"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["age"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["hhincome"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["edu_lev"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["pklot_wk"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["rent_own"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["av_auto"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["av_bike"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["owt_mor"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["emp_sit"].replace([-1, -2], value=np.nan, inplace=True)
```

In [9]:
```
aargau_data_1 = aargau_weekday.dropna()
```

Question 1

In [10]:
```
aargau_wide_data = aargau_data_1[['tripnum', 'person', 'hhsize', 'hhnr', 'age', '
                                  'dis_work', 'emp_sit', 'n_trp_wk', 'n_auto', 'n_b
                                  'pklot_wk', 'av_auto', 'av_bike', 'oev', 'iveh_mv
                                  'ntrf_mor', 'freq_mor', 'mode_5']]
```

In [11]:
```
#Here we are creating a new dummy variable for senior person
aargau_wide_data.loc[:,'senior'] =  np.where(aargau_wide_data['age']>65, 1, 0)

#Here we are creating a new dummy variable for wealthy person
aargau_wide_data.loc[:,'rich'] =  np.where(aargau_wide_data['hhincome']>5, 1, 0)

#Here we are creating a new dummy variable for persons with a bachelors degree or
aargau_wide_data.loc[:,'bachelors'] =  np.where(aargau_wide_data['edu_lev']>5, 1,

#Here we are creating a new dummy variable for employed persons
aargau_wide_data.loc[:, "employed"] =  np.where(aargau_wide_data['emp_sit']<4, 1,

#Here we are creating a new dummy variable for person living in city
aargau_wide_data.loc[:,'city'] =  np.where(aargau_wide_data['city_rur']>1, 0, 1)

#Here we are creating a new dummy variable for persons who own home
aargau_wide_data.loc[:, "home_own"] =  np.where(aargau_wide_data['rent_own']>1, 1

#Here we are creating a new dummy variable for persons with no reserved parking n
aargau_wide_data.loc[:, "no_pklot"] =  np.where(aargau_wide_data['pklot_wk']>2, 1

#Here we are creating a new dummy variable for sov availability
aargau_wide_data.loc[:, "sov_av"] =  np.where(aargau_wide_data['av_auto']<3, 1, 0

#Here we are creating a new dummy variable for hov availability
aargau_wide_data.loc[:, "hov_av"] =  np.where(aargau_wide_data['av_auto']<3, 1, 0

#Here we are creating a new dummy variable for bicycle availability
aargau_wide_data.loc[:, "bike_av"] =  np.where(aargau_wide_data['av_bike']<3, 1, (

#Here we are creating a new dummy variable for transit availability depending on
aargau_wide_data.loc[:, 'transit_av'] =  np.where(aargau_wide_data['oev']<4, 1, 0

#Here we are creating a new dummy variable for walk availability depending on dis
#aargau_wide_data.loc[:, 'walk_av'] =  np.where(aargau_wide_data['dis_work']<5, 1,

aargau_wide_data['walk_av'] = 1

#Here we are creating a new dummy variable if person has to wait at transit stati
aargau_wide_data.loc[:, 'long_wait'] =  np.where(aargau_wide_data['owt_mor']>15 ,

#Here we are creating a new variable for total transit travel time
aargau_wide_data.loc[:, "time_transit"] = aargau_wide_data["trvt_mor"]

#Here we are creating a new dummy variable if person has to take more than one tr
aargau_wide_data.loc[:, 'multp_transfer'] =  np.where(aargau_wide_data['ntrf_mor'

#Here we are creating a new dummy variable if there is highly frequent transit se
aargau_wide_data.loc[:, 'high_freq'] =  np.where(aargau_wide_data['freq_mor']>5,

#Here we are creating a new dummy variable if person has any transit subscription
aargau_wide_data.loc[:, 'transit_sub'] = np.where((aargau_wide_data["h_sub"] < 4)

#Here we are creating a new variable for SOV and HOV travel times in vehicle plus
aargau_wide_data.loc[:, 'time_car'] = aargau_wide_data['iveh_mva'] + 2*aargau_wid
```

```
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pandas\core\indexing.py:297: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  self.obj[key] = _infer_fill_value(value)
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pandas\core\indexing.py:477: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  self.obj[item] = s
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\ipykernel\__main__.py:37: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
```

Here we check to see if the availability actually checks out when compared to the mode selected, essentially ensuring there are no data errors where individuals selected modes that are not available to them

In [12]:
```
aargau_wide_data.loc[:, 'sov_av_check'] = np.where((aargau_wide_data['sov_av']==0
aargau_wide_data.loc[:, 'hov_av_check'] = np.where((aargau_wide_data['hov_av']==0
aargau_wide_data.loc[:, 'transit_av_check'] = np.where((aargau_wide_data['transit
aargau_wide_data.loc[:, 'bicycle_av_check'] = np.where((aargau_wide_data['bike_av
aargau_wide_data.loc[:, 'walk_av_check'] = np.where((aargau_wide_data['walk_av']=
```

In [13]:
```
wide_aargu = aargau_wide_data.dropna()

wide_aargu.shape
```

Out[13]:
```
(822, 53)
```

In [14]:
```
wide_data_1 = wide_aargu[['tripnum', 'person', 'hhsize', 'hhnr', 'dis_work', 'n_t
                          'rich', 'employed', 'home_own', 'city', 'senior', 'time_
                          'ntrf_mor', 'freq_mor', 'mode_5', 'no_pklot', 'sov_av',
                          'long_wait', 'time_transit', 'multp_transfer', 'high_fre
```

In [15]:
```
include_criteria = (wide_data_1.mode_5.isin([1, 2, 3, 4, 5]))

wide_data = wide_data_1.loc[include_criteria].copy()
wide_data.head()
wide_data1a = wide_data[['tripnum', 'person', 'hhsize', 'hhnr', 'dis_work', 'n_tr
                         'rich', 'employed', 'home_own', 'city', 'senior', 'time_
                         'mode_5', 'no_pklot', 'sov_av', 'hov_av', 'bike_av', 'tr
                         'long_wait', 'time_transit', 'transit_sub', 'time_car']]
```

In [16]:
```
wide_data1a.shape
```

Out[16]: (822, 29)

In [17]:
```
# Here we create the list of individual specific variables and prep data for conv
ind_variables = wide_data.columns.tolist()[:14]

alt_varying_variables = {u'travel_time': dict([(1, 'time_car'),
                                               (2, 'time_car'),
                                               (3, 'time_transit'),
                                               (4, 'time_bic'),
                                               (5, 'time_wk')]),
                         u'frequency': dict([(3, 'freq_mor')]),
                         u'transfers': dict([(3, 'ntrf_mor')]),
                         u'transit_subscription': dict([(3, 'transit_sub')]),
                         u'transit_wait_time': dict([(3, 'long_wait')]),
                         u'parking_near_work': dict([(1, 'no_pklot'),
                                                     (2, 'no_pklot')])}

availability_variables = {1: 'sov_av',
                          2: 'hov_av',
                          3: 'transit_av',
                          4: 'bike_av',
                          5: 'walk_av'}


custom_alt_id = "mode_id"

obs_id_column = "custom_id"
wide_data1a[obs_id_column] = np.arange(wide_data1a.shape[0],
                                       dtype=int) + 1

choice_column = "mode_5"
```

```
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\ipykernel\__main__.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
```

In [18]:
```
wide_data1a.shape
```

Out[18]: (822, 30)

In [19]:
```
long_data = pl.convert_wide_to_long(wide_data1a,
                                    ind_variables,
                                    alt_varying_variables,
                                    availability_variables,
                                    obs_id_column,
                                    choice_column,
                                    new_alt_id_name=custom_alt_id)

long_data.head(30).T
```

Out[19]:

	0	1	2	3	4	5	6
custom_id	1.00	1.00	1.0	1.0	2.00000	2.00000	2.0000
mode_id	1.00	2.00	3.0	5.0	1.00000	2.00000	4.0000
mode_5	1.00	0.00	0.0	0.0	0.00000	1.00000	0.0000
tripnum	1.00	1.00	1.0	1.0	1.00000	1.00000	1.0000
person	1.00	1.00	1.0	1.0	3.00000	3.00000	3.0000
hhsize	1.00	1.00	1.0	1.0	4.00000	4.00000	4.0000
hhnr	1502.00	1502.00	1502.0	1502.0	6070.00000	6070.00000	6070.0000
dis_work	20.00	20.00	20.0	20.0	2.00000	2.00000	2.0000
n_trp_wk	5.00	5.00	5.0	5.0	20.00000	20.00000	20.0000
n_auto	1.00	1.00	1.0	1.0	1.00000	1.00000	1.0000

In [20]:
```
basic_specification = OrderedDict()
basic_names = OrderedDict()

basic_specification["intercept"] = [2, 3, 4, 5]
basic_names["intercept"] = ['ASC HOV',
                            'ASC Transit',
                            'ASC Bicycle',
                            'ASC Walk']

basic_specification["travel_time"] = [[1, 2,], 3, 4, 5]
basic_names["travel_time"] = ['Travel Time, units:min (SOV & HOV)',
                              'Travel Time, units:min (Transit)',
                              'Travel Time, units:min (Bike)',
                              'Travel TIme, units:min (Walk)']


basic_specification["parking_near_work"] = [[1, 2]]
basic_names['parking_near_work'] = ["No available parking (SOV, HOV)"]

basic_specification["frequency"] = [3]
basic_names["frequency"] = ['Frequency (Transit)']

basic_specification["transfers"] = [3]
basic_names["transfers"] = ['Number of transfers (Transit)']

basic_specification["transit_subscription"] = [3]
basic_names["transit_subscription"] = ['Has transit subscription (Transit)']

basic_specification["transit_wait_time"] = [3]
basic_names["transit_wait_time"] = ['Long waiting time (Transit)']
```

In [21]:
```
aargau_mode_choice = pl.create_choice_model(data=long_data,
                                            alt_id_col=custom_alt_id,
                                            obs_id_col=obs_id_column,
                                            choice_col=choice_column,
                                            specification=basic_specification,
                                            model_type="MNL",
                                            names=basic_names)

aargau_mode_choice.fit_mle(np.zeros(13))

aargau_mode_choice.get_statsmodels_summary()
```

```
Log-likelihood at zero: -1,254.1704
Initial Log-likelihood: -1,254.1704
Estimation Time: 0.14 seconds.
Final log-likelihood: -588.2526

C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\scipy\optimize\_minimize.py:385: RuntimeWarning: Method BFGS does not use
 Hessian information (hess).
  RuntimeWarning)
```

Out[21]:

Multinomial Logit Model Regression Results

Dep. Variable:	mode_5	**No. Observations:**	822
Model:	Multinomial Logit Model	**Df Residuals:**	809
Method:	MLE	**Df Model:**	13
Date:	Tue, 31 Oct 2017	**Pseudo R-squ.:**	0.531
Time:	21:09:21	**Pseudo R-bar-squ.:**	0.521
converged:	False	**Log-Likelihood:**	-588.253
		LL-Null:	-1,254.170

	coef	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
ASC HOV	-2.4749	0.161	-15.404	0.000	-2.790 -2.160
ASC Transit	-5.8372	0.582	-10.029	0.000	-6.978 -4.696
ASC Bicycle	-2.6870	0.326	-8.243	0.000	-3.326 -2.048
ASC Walk	-1.9146	0.333	-5.744	0.000	-2.568 -1.261
Travel Time, units:min (SOV & HOV)	-0.0704	0.012	-5.779	0.000	-0.094 -0.047
Travel Time, units:min (Transit)	-0.0235	0.012	-1.900	0.057	-0.048 0.001
Travel Time, units:min (Bike)	-0.0879	0.013	-6.825	0.000	-0.113 -0.063
Travel TIme, units:min (Walk)	-0.0643	0.009	-7.317	0.000	-0.082 -0.047
No available parking (SOV, HOV)	-2.4640	0.223	-11.056	0.000	-2.901 -2.027
Frequency (Transit)	0.0162	0.016	0.998	0.318	-0.016 0.048
Number of transfers (Transit)	0.0246	0.168	0.146	0.884	-0.305 0.355

Has transit subscription (Transit)	2.4123	0.379	6.370	0.000	1.670 3.155
Long waiting time (Transit)	-0.6890	0.627	-1.098	0.272	-1.919 0.541

In [22]:
```
np.round(aargau_mode_choice.summary, 3)
```

Out[22]:

	parameters	std_err	t_stats	p_values	robust_std_err	robust_t_stats	robus
ASC HOV	-2.475	0.161	-15.404	0.000	0.161	-15.404	0.000
ASC Transit	-5.837	0.582	-10.029	0.000	0.571	-10.229	0.000
ASC Bicycle	-2.687	0.326	-8.243	0.000	0.330	-8.141	0.000
ASC Walk	-1.915	0.333	-5.744	0.000	0.352	-5.437	0.000
Travel Time, units:min (SOV & HOV)	-0.070	0.012	-5.779	0.000	0.013	-5.452	0.000
Travel Time, units:min (Transit)	-0.024	0.012	-1.900	0.057	0.012	-1.933	0.053
Travel Time, units:min (Bike)	-0.088	0.013	-6.825	0.000	0.012	-7.562	0.000
Travel TIme, units:min (Walk)	-0.064	0.009	-7.317	0.000	0.010	-6.510	0.000
No available parking (SOV, HOV)	-2.464	0.223	-11.056	0.000	0.226	-10.886	0.000
Frequency (Transit)	0.016	0.016	0.998	0.318	0.016	0.989	0.323
Number of transfers (Transit)	0.025	0.168	0.146	0.884	0.157	0.157	0.876
Has transit subscription (Transit)	2.412	0.379	6.370	0.000	0.383	6.291	0.000
Long waiting time (Transit)	-0.689	0.627	-1.098	0.272	0.524	-1.316	0.188

a)

1. We included travel time as alternitive specific variables (after aggregating SOV and HOV together) because we felt that travel time would influence the auto modes similarly, while probably having a different effect on transit, walking and biking. We expected these corefficients to be negative since as travel time increase, people are less likely to want to use that mode. This was reflected in our model output as all travel time coefficients were negative
2. We included the no available parking dummy variable for SOV and HOV modes because we felt that it helped to encompass the influence of space availiabilty on auto based modes. We expected this to have a negative influence on ones probability to use auto modes, which was reflected in the model as the coefficient was negative, and this makes sense as people would perfer an alternative no matter how easy the commute if they cannont park close to their destination.
3. We included the transit specific frequency because we felt that higher freuquency service would have a noted positive impact on travelers choice for transit. In the model this turned out to be true, as indicated by the positive coefficient, but was not significant
4. We included the number of transfers to also account for some of the negative aspects of transit, transfers. Typically, if individuals have to transfer oftern then they will select that mode less often, so we anticipated a negative coefficient. In the model this wasn't true as we indicate a positive coefficient. this also turned out to be insignificant
5. We also included a dummy variable to indicate if the user owned a transit subscription, this is because if someone owns a transit subscription they are pprobably a regular user so we expected a positive coefficient which was reflected in the model with substantial magnitude as well.
6. Finally, we included a long wait time varibale if someone had to wiat longer than 15 min for transit which we expected to have a negative sign, which we saw in our model. It turned out not to be significant which could be due to limited number of long waits in the dataset.

b)

It is important to note that according the output this model did not converge, but we did see favorable significance in the model over the null-formulation. Presented below is the LLR test:

In [23]:
```
# Log-Likelihood ratio
LLnull = -1254.170
LLmodel = -588.253
diff = LLmodel-LLnull
LR = 2*diff
p = chisqprob(LR, 13)
print (LR, p)
```

```
(1331.834, 7.3979092156499267e-277)
```

Here we can see that with a Log-Likelihood ratio test statistic of 1331.834, we get a p-value of essentially 0 so this model is significantly different from a null formulation

In terms of our model coefficients, all turned out to be significant at or near the 95% level except for the transit specific variables of frequency, long wait time, and number of transfers. This could possibly be explained but the limited variability in those variables due to number of transit observations

Finally, our model indicated fairly reasonable goodness-of-fit measures with the pseudo-R^2 value at 0.531

c)

We have done some of this exploration above, in regard to the motorized modes it would make sense to check to see if the individuals had vehicles availaible. In order to check to see if individuals had transit availiable we compared their responses in the transit_av variable to include only those that indicated that they live near or between transit stops. For biking, we could build a filter that would indicate if they owned a bike as well as how long the required trip would be on the scale of possibly 10km. For walking, we could institue a distance filter that would remove walk availability if the user was outside 5km away from thier destination, and even lower if they also indicate senior status.

Question 2

Here we explore the Maringal Effects for the has transit subscription variable on the Transit mode

In [24]:
```
#first we need to determine the probabilities of selecting each availiable mode by

# Create an ordered dictionary mapping shortened model
# names to estimated model objects.
names_to_obj = OrderedDict()
names_to_obj["MNL"] = aargau_mode_choice

# Initialize a dictionary to store the log-likelihoods
# on the prediction dataset.
prediction_log_likelihoods = OrderedDict()
for key in names_to_obj:
    prediction_log_likelihoods[key] = None

# Calculate and store the predictions for all models besides
# the mixed logit model. The predictions will be stored on
# trp_sub
for model_name in names_to_obj:
    if model_name == "Mixed":
        # Don't make predictions for the Mixed Logit Model
        # using the for-loop
        continue
    else:
        # Get the model object to be used in making predictions
        model_obj = names_to_obj[model_name]

        # Note that by default, the predict method returns the
        # predicted probabilities for each available alternative
        # for each choice situation.
        prediction_array = model_obj.predict(long_data)

        # We can use the "return_long_probs" and "choice_col"
        # keyword arguments to ensure that we only return the
        # probability of the chosen alternative for each
        # choice situation.
        chosen_predictions = model_obj.predict(long_data,
                                               choice_col=choice_column,
                                               return_long_probs=False)

        # Calculate the predicted log-likelihood in two ways and,
        # for demonstration, ensure that they agree.
        log_likelihood_calc_1 = np.log(chosen_predictions).sum()

        log_predictions = np.log(prediction_array)
        choice_array = long_data[choice_column].values
        log_likelihood_calc_2 = choice_array.dot(log_predictions)

        assert np.allclose(log_likelihood_calc_1,
                           log_likelihood_calc_2)

        # Store the log-likelihood
        prediction_log_likelihoods[model_name] = log_likelihood_calc_1

        # Create a column name for the predictions
        prediction_col = model_name + "_Predictions"

        # Store the predicted probabilities
```

```
        long_data[prediction_col] = prediction_array
```

In [25]:
```
long_data[['custom_id','mode_id', 'mode_5', 'MNL_Predictions', 'transit_subscript
```

Out[25]:

	custom_id	mode_id	mode_5	MNL_Predictions	transit_subscription
0	1	1	1	0.911695	0.0
1	1	2	0	0.076736	0.0
2	1	3	0	0.011416	0.0
3	1	5	0	0.000153	0.0
4	2	1	0	0.684430	0.0

In [27]:
```
# Direct Effect on Transit Modes
tranit_use_pred = long_data.loc[long_data['mode_id']==3, ('transit_subscription',
marg_eff_transub  = (1-(tranit_use_pred['MNL_Predictions']))*tranit_use_pred['MNL
avg_marg_eff_transsub = marg_eff_transub.sum() / len(tranit_use_pred)
print ("The average direct marginal effect of having a transit subscritption on T
```

```
The average direct marginal effect of having a transit subscritption on Transit
Mode Choice Probability is: 0.208483209731
```

In [28]:
```
# Cross Effect on SOV Mode
# Direct Effect on Transit Modes
tranit_use_pred_sov = long_data.loc[long_data['mode_id']==1, ('transit_subscripti
tranit_use_pred_sov.reset_index(inplace=True)
marg_eff_transub_sov  = -(tranit_use_pred['MNL_Predictions'])*tranit_use_pred_sov
avg_marg_eff_transsub_sov = marg_eff_transub_sov.sum() / len(tranit_use_pred_sov)
print ("The average cross marginal effect of having a transit subscritption on SO
```

```
The average cross marginal effect of having a transit subscritption on SOV Mode
Choice Probability is: -0.0626150372768
```

In [29]:
```
# Cross Effect on HOV Mode
tranit_use_pred_hov = long_data.loc[long_data['mode_id']==2, ('transit_subscripti
tranit_use_pred_hov.reset_index(inplace=True)
marg_eff_transub_hov  = -(tranit_use_pred['MNL_Predictions'])*tranit_use_pred_hov
avg_marg_eff_transsub_hov = marg_eff_transub_hov.sum() / len(tranit_use_pred_hov)
print ("The average cross marginal effect of having a transit subscritption on HO
```

```
The average cross marginal effect of having a transit subscritption on HOV Mode
Choice Probability is: -0.00527020354167
```

In [30]:
```
# Cross Effect on Bike Mode
tranit_use_pred_bk = long_data.loc[long_data['mode_id']==4, ('transit_subscriptio
tranit_use_pred_bk.reset_index(inplace=True)
marg_eff_transub_bk  = -(tranit_use_pred['MNL_Predictions'])*tranit_use_pred_bk['
avg_marg_eff_transsub_bk = marg_eff_transub_bk.sum() / len(tranit_use_pred_bk)
print ("The average cross marginal effect of having a transit subscritption on Bi
```

```
The average cross marginal effect of having a transit subscritption on Bike Mod
e Choice Probability is: -0.0120237227364
```

In [31]:
```
# Cross Effect on Walk Mode
tranit_use_pred_wk = long_data.loc[long_data['mode_id']==5, ('transit_subscriptio
tranit_use_pred_wk.reset_index(inplace=True)
marg_eff_transub_wk  = -(tranit_use_pred['MNL_Predictions'])*tranit_use_pred_wk['
avg_marg_eff_transsub_wk = marg_eff_transub_wk.sum() / len(tranit_use_pred_wk)
print ("The average cross marginal effect of having a transit subscritption on Wa
```

```
The average cross marginal effect of having a transit subscritption on Walk Mod
e Choice Probability is: -0.00502390199452
```

Here we have addressed the maringal effects of owning a transit subscription on users mode choice probability. It is not surprising that we see a substantial jump in transit mode selection probability (about 21%) when the user has a subsctription. What was interesting, is that transit subscription ownership does have a stronger average corss marginal effect on SOV mode choice (about 6% decrease) than it does on HOV mode choice (less than 1% decrease). Both are auto modes, but possibly transit pass holders are more comfortable traveling together and thus are less influenced by the amount of people within the vehicle. Similar small decreases in probablilty (around 1%) were observed for the cross effects of both walk and bike modes.

Nested Logit Model Estimation

Adrita Islam and Raymond Gerte

In [1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import scipy.stats as stats
import pylogit as pl
from collections import OrderedDict
from scipy.stats import chisqprob

from patsy import dmatrices

%matplotlib inline
import matplotlib.pyplot as plt
pd.options.display.max_columns = 150
pd.options.display.max_colwidth = 500
```

Load in the data and perform neccessary filtering

In [2]:
```
aargau_data = pd.read_csv('../HW 4/Trip_File_Aargau.csv')
aargau_adult = aargau_data[aargau_data['age']>=18]
aargau_am_peak = aargau_adult[aargau_adult['tod']==1]
aargau_work = aargau_am_peak[aargau_am_peak['trip_pur']==1]
aargau_weekday = aargau_work[aargau_work['weekday']<= 5]
print ("Total number of rows in the dataset is = {0}").format(len(aargau_data))
print ("Total number of rows after restricting the dataset for work trips made by
```

```
Total number of rows in the dataset is = 17651
Total number of rows after restricting the dataset for work trips made by an ad
ult during AM peak in a weekday = 1172
```

Here we replace invalid data and drop NaN values

In [3]:
```
aargau_weekday["n_auto"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["n_bike"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["dis_work"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["n_trp_wk"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["mode_5"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["h_sub"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["o_sub"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["g_sub"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["age"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["hhincome"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["edu_lev"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["pklot_wk"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["rent_own"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["av_auto"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["av_bike"].replace([-1, -2], value=np.nan, inplace=True)
aargau_weekday["owt_mor"].replace([-1], value=np.nan, inplace=True)
aargau_weekday["emp_sit"].replace([-1, -2], value=np.nan, inplace=True)
```

In [4]:
```
aargau_data_1 = aargau_weekday.dropna()
```

Question 1

In [5]:
```
aargau_wide_data = aargau_data_1[['tripnum', 'person', 'hhsize', 'hhnr', 'age', '
                                  'dis_work', 'emp_sit', 'n_trp_wk', 'n_auto', 'n_b
                                  'pklot_wk', 'av_auto', 'av_bike', 'oev', 'iveh_mv
                                  'ntrf_mor', 'freq_mor', 'mode_5']]
```

Creating Variables

In [6]:
```
#Here we are creating a new dummy variable for senior person
aargau_wide_data.loc[:,'senior'] =  np.where(aargau_wide_data['age']>65, 1, 0)

#Here we are creating a new dummy variable for wealthy person
aargau_wide_data.loc[:,'rich'] =  np.where(aargau_wide_data['hhincome']>6, 1, 0)

#Here we are creating a new dummy variable for persons with a bachelors degree or
aargau_wide_data.loc[:,'bachelors'] =  np.where(aargau_wide_data['edu_lev']>5, 1,

#Here we are creating a new dummy variable for employed persons
aargau_wide_data.loc[:, "employed"] =  np.where(aargau_wide_data['emp_sit']<4, 1,

#Here we are creating a new dummy variable for person living in city
aargau_wide_data.loc[:,'city'] =  np.where(aargau_wide_data['city_rur']>1, 0, 1)

#Here we are creating a new dummy variable for persons who own home
aargau_wide_data.loc[:, "home_own"] =  np.where(aargau_wide_data['rent_own']>1, 1

#Here we are creating a new dummy variable for persons with no reserved parking n
aargau_wide_data.loc[:, "no_pklot"] =  np.where(aargau_wide_data['pklot_wk']>2, 1

#Here we are creating a new dummy variable for sov availability
aargau_wide_data.loc[:, "sov_av"] =  np.where(aargau_wide_data['av_auto']<3, 1, 0

#Here we are creating a new dummy variable for hov availability
aargau_wide_data.loc[:, "hov_av"] =  np.where(aargau_wide_data['av_auto']<3, 1, 0

#Here we are creating a new dummy variable for bicycle availability
aargau_wide_data.loc[:, "bike_av"] =  np.where(aargau_wide_data['av_bike']<3, 1, (

#Here we are creating a new dummy variable for transit availability depending on
aargau_wide_data.loc[:, 'transit_av'] =  np.where(aargau_wide_data['oev']<4, 1, 0

aargau_wide_data.loc[:, 'walk_av'] = 1

#Here we are creating a new dummy variable if person has to wait at transit stati
aargau_wide_data.loc[:, 'long_wait'] =  np.where(aargau_wide_data['owt_mor']>15 ,

#Here we are creating a new variable for total transit travel time
aargau_wide_data.loc[:, "time_transit"] = aargau_wide_data["trvt_mor"]

#Here we are creating a new dummy variable if person has to take more than one tr
aargau_wide_data.loc[:, 'multp_transfer'] =  np.where(aargau_wide_data['ntrf_mor'

#Here we are creating a new dummy variable if there is highly frequent transit se
aargau_wide_data.loc[:, 'high_freq'] =  np.where(aargau_wide_data['freq_mor']>4, 1

#Here we are creating a new dummy variable if person has any transit subscription
aargau_wide_data.loc[:, 'transit_sub'] = np.where((aargau_wide_data["h_sub"] < 4)

#Here we are creating a new variable for SOV and HOV travel times in vehicle plus
aargau_wide_data.loc[:, 'time_car'] = aargau_wide_data['iveh_mva'] + aargau_wide_(
```

```
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pandas\core\indexing.py:297: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  self.obj[key] = _infer_fill_value(value)
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pandas\core\indexing.py:477: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  self.obj[item] = s
```

In [7]:
```
aargau_wide_data.loc[:, 'sov_av_check'] = np.where((aargau_wide_data['sov_av']==0
aargau_wide_data.loc[:, 'hov_av_check'] = np.where((aargau_wide_data['hov_av']==0
aargau_wide_data.loc[:, 'transit_av_check'] = np.where((aargau_wide_data['transit_
aargau_wide_data.loc[:, 'bicycle_av_check'] = np.where((aargau_wide_data['bike_av
aargau_wide_data.loc[:, 'walk_av_check'] = np.where((aargau_wide_data['walk_av']=
```

In [8]:
```
wide_aargau = aargau_wide_data.dropna()

wide_aargau.shape
```

Out[8]: (822, 53)

In [9]:
```
wide_data_1 = wide_aargau[['tripnum', 'person', 'hhsize', 'hhnr', 'n_trp_wk', 'n_
                           'rich', 'employed', 'home_own', 'city', 'senior', 'time_
                            'mode_5', 'no_pklot', 'sov_av', 'hov_av', 'bike_av', 't
                            'multp_transfer', 'high_freq', 'transit_sub']]
```

In [10]:
```
include_criteria = (wide_data_1.mode_5.isin([1, 2, 3, 4, 5]))

wide_data = wide_data_1.loc[include_criteria]
wide_data.shape
```

Out[10]: (822, 29)

Converting to long format

In [11]:
```
# Here we create the list of individual specific variables and prep data for conv
ind_variables = wide_data.columns.tolist()[:14]

alt_varying_variables = {u'travel_time': dict([(1, 'time_car'),
                                               (2, 'time_car'),
                                               (3, 'time_transit'),
                                               (4, 'time_bic'),
                                               (5, 'time_wk')]),
                         u'transit_subscription': dict([(3, 'transit_sub'),
                                                        (1, 'transit_sub'),
                                                        (2, 'transit_sub')]),
                         u'parking_near_work': dict([(1, 'no_pklot'),
                                                     (2, 'no_pklot')])}

availability_variables = {1: 'sov_av',
                          2: 'hov_av',
                          3: 'transit_av',
                          4: 'bike_av',
                          5: 'walk_av'}

wide_data["custom_id"] = np.arange(wide_data.shape[0], dtype=int) + 1

new_alt_id = "mode_id"

obs_id_column = "custom_id"

choice_column = "mode_5"
```

In [12]:
```
# Here we convery the wide dataset to a long dataset
long_data = pl.convert_wide_to_long(wide_data,
                                    ind_variables,
                                    alt_varying_variables,
                                    availability_variables,
                                    obs_id_column,
                                    choice_column,
                                    new_alt_id_name=new_alt_id)

long_data.head(50).T
```

```
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pa
ckages\pylogit\choice_tools.py:421: UserWarning: Note, there are 30 variables
in wide_data but the inputs ind_vars, alt_specific_vars, and subset_specific_
vars only account for 27 variables.
  msg_2 + msg_3.format(num_vars_accounted_for))
```

Out[12]:

	0	1	2	3	4	5	6
custom_id	1.00	1.00	1.0	1.0	2.00000	2.00000	2.0000
mode_id	1.00	2.00	3.0	5.0	1.00000	2.00000	4.0000
mode_5	1.00	0.00	0.0	0.0	0.00000	1.00000	0.0000
tripnum	1.00	1.00	1.0	1.0	1.00000	1.00000	1.0000
person	1.00	1.00	1.0	1.0	3.00000	3.00000	3.0000
hhsize	1.00	1.00	1.0	1.0	4.00000	4.00000	4.0000
hhnr	1503.00	1503.00	1503.0	1503.0	6070.00000	6070.00000	6070.0000

Specifying Nesting Structure

In [13]:
```
nest_membership = OrderedDict()
nest_membership["autos"] = [1]
nest_membership["group"] = [2, 3]
nest_membership["other"] = [4, 5]
```

In [14]:
```
basic_specification = OrderedDict()
basic_names = OrderedDict()

basic_specification["intercept"] = [2, 3, 4, 5]
basic_names["intercept"] = ['ASC HOV',
                            'ASC Transit',
                            'ASC Bicycle',
                            'ASC Walk']

basic_specification["travel_time"] = [1, [2, 3], [4, 5]]
basic_names["travel_time"] = ['Travel Time, units:min (SOV)',
                              'Travel Time, units:min (HOV & Transit)',
                              'Travel TIme, units:min (Walk & Bike)']

basic_specification["dis_work"] = [[4, 5]]
basic_names["dis_work"] = ['Distance to work, units:Km (Bike & Walk)']


basic_specification["n_auto"] = [1, 2]
basic_names["n_auto"] = ['Number of autos in HH (SOV)',
                         'Number of autos in HH (HOV)']

basic_specification["n_bike"] = [[4, 5]]
basic_names["n_bike"] = ['Number of bike in HH (Bike)']

basic_specification["parking_near_work"] = [1, 2]
basic_names['parking_near_work'] = ["No available parking (SOV)",
                                    "No available parking (HOV)",]

basic_specification["transit_subscription"] = [1 , 3]
basic_names["transit_subscription"] = ['Has transit subscription (SOV)',
                                       'Has transit subscription (Transit)']
```

Nested Logit Model

In [15]:
```
def logit(x):

    return np.log(x/(1.0 - x))
```

In [16]:
```
aargau_nested = pl.create_choice_model(data=long_data,
                                       alt_id_col=new_alt_id,
                                       obs_id_col=obs_id_column,
                                       choice_col=choice_column,
                                       specification=basic_specification,
                                       model_type="Nested Logit",
                                       names=basic_names,
                                       nest_spec=nest_membership)

init_nests = np.array([40, logit(2.05**-1)])

init_coefs = np.zeros(16)

init_values = np.concatenate((init_nests, init_coefs), axis=0)

aargau_nested.fit_mle(init_values,
                      constrained_pos=[0])
```

```
Log-likelihood at zero: -1,155.6178
Initial Log-likelihood: -1,154.6630

C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pylogit\nested_choice_calcs.py:561: RuntimeWarning: divide by zero encount
ered in log
  log_exp_sums = np.log(vector_dict["ind_sums_per_nest"])

Estimation Time: 1.24 seconds.
Final log-likelihood: -572.8684

C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pylogit\nested_choice_calcs.py:729: RuntimeWarning: divide by zero encount
ered in log
  log_exp_sums = np.log(vector_dict["ind_sums_per_nest"])
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\pylogit\base_multinomial_cm_v2.py:1162: RuntimeWarning: invalid value enco
untered in sqrt
  self._store_inferential_results(np.sqrt(np.diag(self.robust_cov)),
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\scipy\stats\_distn_infrastructure.py:875: RuntimeWarning: invalid value en
countered in greater
  return (self.a < x) & (x < self.b)
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\scipy\stats\_distn_infrastructure.py:875: RuntimeWarning: invalid value en
countered in less
  return (self.a < x) & (x < self.b)
C:\Users\adi16101.ENGR_STUDENT\AppData\Local\Continuum\Anaconda_2\lib\site-pack
ages\scipy\stats\_distn_infrastructure.py:1814: RuntimeWarning: invalid value e
ncountered in less_equal
  cond2 = cond0 & (x <= self.a)
```

In [17]:
```
aargau_nested.get_statsmodels_summary()
```

Out[17]:

Nested Logit Model Regression Results

Dep. Variable:	mode_5	**No. Observations:**	822
Model:	Nested Logit Model	**Df Residuals:**	804
Method:	MLE	**Df Model:**	18
Date:	Tue, 31 Oct 2017	**Pseudo R-squ.:**	0.504
Time:	21:12:48	**Pseudo R-bar-squ.:**	0.489
converged:	False	**Log-Likelihood:**	-572.868
		LL-Null:	-1,155.618

	coef	std err	z	P>|z|	[95.0% Conf. Int.]
autos	40.0000	nan	nan	nan	nan nan
group	1.7921	1.815	0.988	0.323	-1.765 5.349
other	4.6868	25.997	0.180	0.857	-46.267 55.641
ASC HOV	-3.4841	0.425	-8.207	0.000	-4.316 -2.652
ASC Transit	-3.5473	0.528	-6.717	0.000	-4.582 -2.512
ASC Bicycle	-2.1301	0.465	-4.578	0.000	-3.042 -1.218
ASC Walk	-1.0749	0.466	-2.309	0.021	-1.987 -0.163
Travel Time, units:min (SOV)	-0.0586	0.013	-4.451	0.000	-0.084 -0.033
Travel Time, units:min (HOV & Transit)	-0.0349	0.010	-3.598	0.000	-0.054 -0.016
Travel Tlme, units:min (Walk & Bike)	-0.0632	0.009	-6.771	0.000	-0.081 -0.045
Distance to work, units:Km (Bike & Walk)	-0.1064	0.026	-4.022	0.000	-0.158 -0.055
Number of autos in HH (SOV)	0.6591	0.178	3.697	0.000	0.310 1.009
Number of autos in HH (HOV)	0.5996	0.236	2.544	0.011	0.138 1.061
Number of bike in HH (Bike)	0.2738	0.078	3.524	0.000	0.122 0.426
No available parking (SOV)	-2.3294	0.247	-9.427	0.000	-2.814 -1.845
No available parking (HOV)	-1.3260	0.453	-2.927	0.003	-2.214 -0.438
Has transit subscription (SOV)	-0.6499	0.252	-2.580	0.010	-1.144 -0.156
Has transit subscription (Transit)	1.8700	0.477	3.924	0.000	0.936 2.804

(a) Rationale for including variables

1. We included **travel time** as alternitive specific variables (after aggregating transit and HOV and walk and bike together) because we felt that travel time would influence the modes

that are nested together similarly. We expected these corefficients to be negative since as travel time increase, people are less likely to want to use that mode. This was reflected in our model output as all travel time coefficients were negative.

1. We included the **distance to work** variable for Walk and Bike modes because with the increase of distance to work from home, the probability of choosing walk or bike decreases. The negative coefficient associated with distance to work also reflects our primary thinking.
2. We included **number of autos in the household** for SOV & HOV because we assumed that presence of automobile in the household will encourage the usage of these modes. The poistive sign associated with the coefficients supports our assumption. So with the increase of number of auto in the household, the probability of choosing SOV and HOV increases.
3. We included **number of bikes in the household** for bike because we assumed that presence of bikes in the household means people will use bike more often. The positive sign associated with the coefficient shows that with the increase of number of bikes in the household, the probability of choosing active transportation modes (walk and bike) increases.
4. We included the **no available parking** dummy variable for SOV and HOV modes because we felt that it helped to encompass the influence of space availiabilty on auto based modes. We expected this to have a negative influence on ones probability to use auto modes, which was reflected in the model as the coefficient was negative, and this makes sense as people would perfer an alternative no matter how easy the commute if they cannont park close to their destination.
5. We included the dummy for **high frequency** for transit because we felt that higher freuquency service would have a noted positive impact on travelers choice for transit and a negative impact on SOV. In the model this turned out to be true, as indicated by the sign of the coefficients.
6. We also included a dummy variable to indicate if the user **owned a transit subscription**, this is because if someone owns a transit subscription they are pprobably a regular user so we expected a positive coefficient associated with transit and negative one with SOV. This was correctly reflected in the model with substantial magnitude as well.

Some variables that were included in the multinomeal logit model but were not included in the nested logit model because they didn't turn out to be significant or the model demanded a different formulation.

(b) Model Assesment

The model is statistically significant. Because the initial log likelihood for the null model is -1,155.6178 but the final log-likelihood: -572.8684. The higher log likelihood indicates that the model is better than the null model.

All the coefficient estimates in our model are significant at 95% level as indicated by their p value in the summary table.

The pseudo R-squraed in our model is 0.504 and the adjusted pseudo R-squared is 0.489 which is reasonably high for nested logit model. So the model fit is good.

(c) Hypothesis Testing

In [18]:
```
def convert_to_parameterization_used_in_class_notes(value):
    return np.exp(value)/(1 + np.exp(value))
```

In [19]:
```
print "Value of the logsum parameter for upper nest using an alternate parameteri
print "Value of the logsum parameter for upper nest using parameterization from c
```

```
Value of the logsum parameter for upper nest using an alternate parameterizatio
n:40.0
Value of the logsum parameter for upper nest using parameterization from class
 notes: 1.0
```

In [20]:
```
print "Value of the logsum parameter for lower nest using an alternate parameteri
print "Value of the logsum parameter for lower nest using parameterization from c
```

```
Value of the logsum parameter for lower nest using an alternate parameterizatio
n:1.79207187527
Value of the logsum parameter for lower nest using parameterization from class
 notes: 0.857181106676
```

The logsum value is very close to 1 which indicates that SOV and transit are not correlated. It was not necessary to put them in a separate nest as they can be listed at the top level along with SOV mode.