Electricity_Reliability_MODELS

December 28, 2022

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings

# Statistical Packages
import statsmodels.api as sm
from scipy.interpolate import interp1d
from scipy import stats
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import patsy as pt

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV



import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
# reading in the final dataset
data = pd.read_csv('Electricity_Reliability_Dataset.csv')

data.columns
```

[2]:
```
Index(['DAY', 'MONTH', 'YEAR', 'OUTAGE_DURATION',
       'NUMBER_OF_CUSTOMERS_AFFECTED', 'TOWN', 'LATITUDE', 'LONGITUDE',
       'AVERAGE_DAILY_WIND_SPEED', 'PRECIPITATION', 'SNOW', 'TEMP_MAX',
       'TEMP_MIN', 'FASTEST_TWO_MIN_WIND_SPEED', 'THUNDER', 'ICE_PELLETS',
       'HAIL', 'GLAZE', 'SMOKE', 'DRIFTING_SNOW', 'HIGH_DAMAGING_WINDS',
       'COUNTY', 'POPULATION', 'INCOME', 'INCOME_PER_CAPITA', 'INCOME_GROUP',
       'POP_DENSITY', 'DENSITY_GROUP', 'WHITE_PERCENT', 'BLACK_PERCENT',
       'ASIAN_PERCENT', 'HISPANIC_PERCENT', 'ALASKAN_NATIVE', 'PACIFIC_NATIVE',
       'OTHER_RACE', 'TWO_OR_MORE_RACES', 'DIVERSITY_GROUP',
```

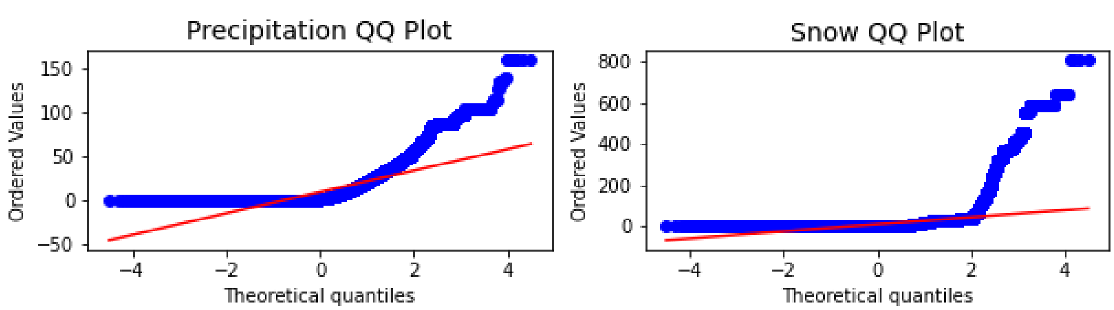
```
 'CUSTOMER_OUTAGE_HOURS', 'SAIDI'],
dtype='object')
```

1 Checking for normality

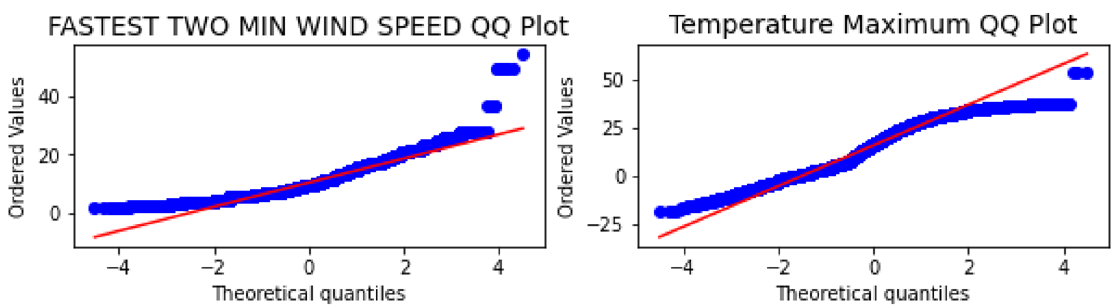
[3]:
```
print('PRECIPITATION:' )
print(stats.shapiro(data['PRECIPITATION']))
print('\nSNOW:' )
print(stats.shapiro(data['SNOW']))
print('\nFASTEST TWO MIN WIND SPEED:' )
print(stats.shapiro(data['FASTEST_TWO_MIN_WIND_SPEED']))
print('\nTEMPERATURE MAXIMUM:' )
print(stats.shapiro(data['TEMP_MAX']))
```

```
PRECIPITATION:
ShapiroResult(statistic=0.6288862824440002, pvalue=0.0)

SNOW:
ShapiroResult(statistic=0.23575669527053833, pvalue=0.0)

FASTEST TWO MIN WIND SPEED:
ShapiroResult(statistic=0.9450325965881348, pvalue=0.0)

TEMPERATURE MAXIMUM:
ShapiroResult(statistic=0.9672042727470398, pvalue=0.0)
```

[4]:
```
plt.figure(figsize=(10,2))
plt.subplot(121)
stats.probplot(data['PRECIPITATION'], dist="norm", plot=plt)
plt.title('Precipitation QQ Plot',fontsize = 14)
plt.subplot(122)
stats.probplot(data['SNOW'], dist="norm", plot=plt)
plt.title('Snow QQ Plot',fontsize = 14)
plt.show()
```



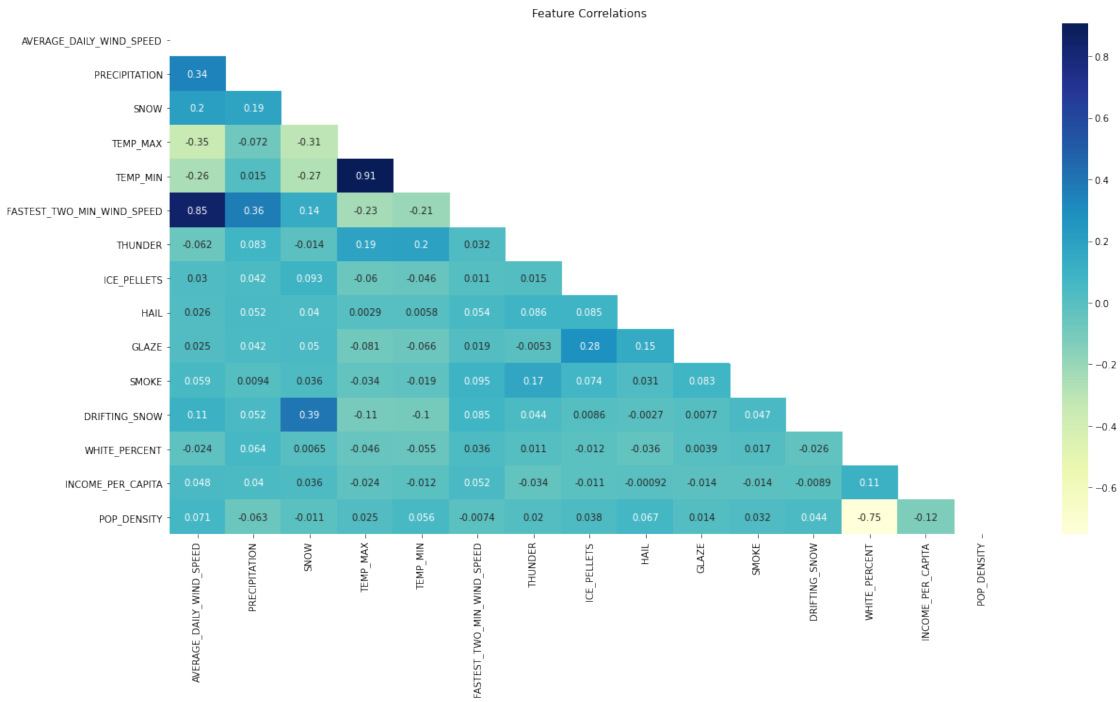
[5]:
```
plt.figure(figsize=(10,2))
plt.subplot(121)
stats.probplot(data['FASTEST_TWO_MIN_WIND_SPEED'], dist="norm", plot=plt)
plt.title('FASTEST TWO MIN WIND SPEED QQ Plot',fontsize = 14)
plt.subplot(122)
stats.probplot(data['TEMP_MAX'], dist="norm", plot=plt)
plt.title('Temperature Maximum QQ Plot',fontsize = 14)
plt.show()
```



2 CORRELATION MATRIX

[6]:
```
#selecting features to use in corr matrix
feature_check = data[['AVERAGE_DAILY_WIND_SPEED', 'PRECIPITATION', 'SNOW',␣
↪'TEMP_MAX',
                      'TEMP_MIN', 'FASTEST_TWO_MIN_WIND_SPEED', 'THUNDER',␣
↪'ICE_PELLETS',
                      'HAIL', 'GLAZE', 'SMOKE', 'DRIFTING_SNOW',␣
↪'WHITE_PERCENT',
                     ␣
↪'INCOME_PER_CAPITA','POP_DENSITY','CUSTOMER_OUTAGE_HOURS']]


# selecting train test data
X_train = feature_check.drop(['CUSTOMER_OUTAGE_HOURS'], axis=1)
y_train = feature_check[['CUSTOMER_OUTAGE_HOURS']]
# creating corr matrix
corr = X_train.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
fig, ax = plt.subplots(figsize=(20,10))
ax.set_title('Feature Correlations')
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,
```

```
        mask = mask, cmap="YlGnBu",
        annot=True)
plt.show()

#plt.savefig('feature correlations.png',bbox_inches="tight" )
```



3 RANDOM FOREST FOR FEATURE SELECTION

[4]:
```
# Stop the random forest from outputting multiple warnings
warnings.filterwarnings('ignore')
# Train the tree using a randomized grid search and set the parameters equal to␣
↪the best parameters found
rf_tree = RandomForestRegressor()
rf_tree.fit(X_train, y_train)

param_dist = {'n_estimators': stats.randint(10, 100),
              'max_leaf_nodes': stats.randint(3, 100),
              'max_features': ["auto"],
              'max_depth': stats.randint(1, 10),
              'min_samples_leaf': stats.randint(1, 30),
              'min_samples_split': stats.randint(2, 20)}

rnd_search = RandomizedSearchCV(rf_tree, param_distributions=param_dist, cv =␣
↪10, n_iter = 50)
```

```
rnd_search.fit(X_train, y_train)

rf_tree.set_params( n_estimators=rnd_search.best_params_['n_estimators'],
                    max_leaf_nodes = rnd_search.best_params_['max_leaf_nodes'],
                    max_features = rnd_search.best_params_['max_features'],
                    max_depth = rnd_search.best_params_['max_depth'],
                    min_samples_leaf = rnd_search.
 ↪best_params_['min_samples_leaf'],
                    min_samples_split = rnd_search.
 ↪best_params_['min_samples_split'])
```

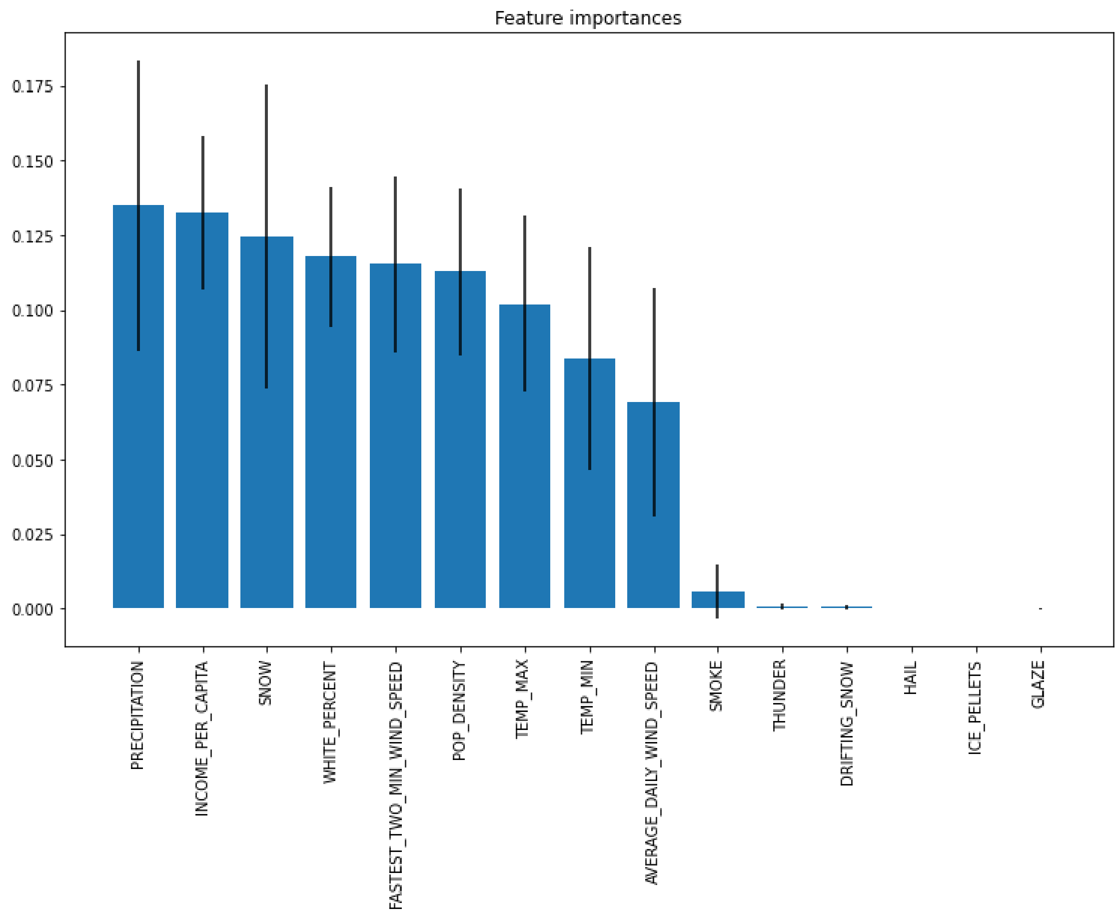
[4]:
```
RandomForestRegressor(max_depth=8, max_leaf_nodes=82, min_samples_leaf=11,
                      n_estimators=38)
```

[5]:
```
# Plot table of feature importances sorted by feature importance
importances = rf_tree.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_tree.estimators_],
             axis=0)

indices = np.argsort(importances)[::-1]
importance_list = []
for f in range(X_train.shape[1]):
    variable_importance_array = [X_train.columns[indices[f]],␣
 ↪round(importances[indices[f]], 4), round(std[indices[f]], 4)]
    importance_list.append(variable_importance_array)
importance_array = np.array(importance_list)
pd.DataFrame(importance_array).rename(columns={0: 'Features', 1: 'Importance␣
 ↪Value', 2: 'Standard Deviations'})
```

[5]:

	Features	Importance Value	Standard Deviations
0	PRECIPITATION	0.135	0.0485
1	INCOME_PER_CAPITA	0.1326	0.0259
2	SNOW	0.1246	0.0509
3	WHITE_PERCENT	0.1177	0.0233
4	FASTEST_TWO_MIN_WIND_SPEED	0.1153	0.0294
5	POP_DENSITY	0.1129	0.028
6	TEMP_MAX	0.102	0.0294
7	TEMP_MIN	0.0838	0.0374
8	AVERAGE_DAILY_WIND_SPEED	0.069	0.0383
9	SMOKE	0.0058	0.009
10	THUNDER	0.0006	0.0009
11	DRIFTING_SNOW	0.0005	0.0007
12	HAIL	0.0001	0.0001
13	ICE_PELLETS	0.0001	0.0001
14	GLAZE	0.0001	0.0003

[6]:
```
# Plot the feature importances with standard deviation bars
plt.figure(figsize = (12.5,7.5))
plt.title("Feature importances")
plt.bar(X_train.columns[indices], importances[indices], yerr=std[indices],␣
↪align="center")
plt.xticks(X_train.columns[indices], rotation = 90)
plt.show()

# plt.savefig('feature_importance.png',bbox_inches="tight" )
```



4 SPLINE ANALYSIS

[7]:
```
# COLS for spline analysis
data_vars = data[['PRECIPITATION', 'SNOW', 'TEMP_MAX',␣
↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]
```

```
[8]: def spline_optimal(df, k, degree, degree_of_freedom_max, predictor_name,␣
     ↪response_name):
         # Fill in the ellipses to complete part (a)
         mses = pd.DataFrame()
         fold = 0

         kf = KFold(n_splits=k, shuffle=True, random_state=0)
         for train_index, val_index in kf.split(df):
             # Separate each array into respective variables
             df_train = np.asarray(df)[train_index]
             df_train = pd.DataFrame(df_train, columns = df.columns)
             df_val = np.asarray(df)[val_index]
             df_val = pd.DataFrame(df_val, columns = df.columns)

             # Calculate the MSE for each degree of freedom
             MSE_array = []
             for deg_of_freedom in range(degree + 1, degree_of_freedom_max + 1):


                 string = 'bs('+predictor_name+', df='+str(deg_of_freedom)+',␣
     ↪degree='+str(degree)+', include_intercept=True)'
                 X_train = pt.dmatrix(string, df_train)
                 X_val = pt.dmatrix(string, df_val)

                 model = linear_model.LinearRegression().fit(X_train, np.
     ↪asarray(df_train[response_name]))
                 y_pred = model.predict(X_val)

                 MSE = mean_squared_error(np.asarray(df_val[response_name]), y_pred)
                 MSE_array.append(MSE)


             mses[fold] = MSE_array
             fold = fold+1

         dof = np.arange(degree+1, degree_of_freedom_max+1)

         # Average the MSE across folds
         mses['mses_ave'] = mses.mean(axis=1)
         mses['mses_std'] = mses.std(axis=1)

         # Determine the minimum average MSE and the polynomial order where it occurs
         MSE_ave_min = min(mses.mses_ave)
         degree_of_freedom_MSE_ave_min = mses['mses_ave'].idxmin()+degree+1

         one_std_err_limit = MSE_ave_min + mses['mses_std'][mses['mses_ave'].
     ↪idxmin()]
```

```
#     print(one_std_err_limit)

    MSE_ave_min_one_std_err = 0
    for item in mses['mses_ave']:
        if item <= one_std_err_limit:
            MSE_ave_min_one_std_err = item
            break

    degree_of_freedom_one_std_err = mses.index[mses['mses_ave'] ==␣
↪MSE_ave_min_one_std_err]+degree+1

    return MSE_ave_min, degree_of_freedom_MSE_ave_min,␣
↪degree_of_freedom_one_std_err[0]
```

[9]:
```
def spline_optimal_cubic(df, k, degree, degree_of_freedom_max, predictor_name,␣
↪response_name):
    # Fill in the ellipses to complete part (a)
    mses = pd.DataFrame()
    fold = 0

    kf = KFold(n_splits=k, shuffle=True, random_state=0)
    for train_index, val_index in kf.split(df):
        # Separate each array into respective variables
        df_train = np.asarray(df)[train_index]
        df_train = pd.DataFrame(df_train, columns = df.columns)
        df_val = np.asarray(df)[val_index]
        df_val = pd.DataFrame(df_val, columns = df.columns)

        # Calculate the MSE for each degree of freedom
        MSE_array = []
        for deg_of_freedom in range(degree + 1, degree_of_freedom_max + 1):

            string = 'cr('+predictor_name+', df='+str(deg_of_freedom)+')'
            X_train = pt.dmatrix(string, df_train)
            X_val = pt.dmatrix(string, df_val)

            model = linear_model.LinearRegression().fit(X_train, np.
↪asarray(df_train[response_name]))
            y_pred = model.predict(X_val)

            MSE = mean_squared_error(np.asarray(df_val[response_name]), y_pred)
            MSE_array.append(MSE)


        mses[fold] = MSE_array
        fold = fold+1
```

```
    dof = np.arange(degree+1, degree_of_freedom_max+1)

    # Average the MSE across folds
    mses['mses_ave'] = mses.mean(axis=1)
    mses['mses_std'] = mses.std(axis=1)

    # Determine the minimum average MSE and the polynomial order where it occurs
    MSE_ave_min = min(mses.mses_ave)
    degree_of_freedom_MSE_ave_min = mses['mses_ave'].idxmin()+degree+1

    one_std_err_limit = MSE_ave_min + mses['mses_std'][mses['mses_ave'].
 ↪idxmin()]
#     print(one_std_err_limit)

    MSE_ave_min_one_std_err = 0
    for item in mses['mses_ave']:
        if item <= one_std_err_limit:
            MSE_ave_min_one_std_err = item
            break

    degree_of_freedom_one_std_err = mses.index[mses['mses_ave'] ==␣
 ↪MSE_ave_min_one_std_err]+degree+1

    return MSE_ave_min, degree_of_freedom_MSE_ave_min,␣
 ↪degree_of_freedom_one_std_err[0]
```

[10]:
```
k = 5
degree_of_freedom_max = 10
response_name = 'CUSTOMER_OUTAGE_HOURS'

predictors = data_vars.drop(['CUSTOMER_OUTAGE_HOURS'], axis=1).columns

degree = 2

optimal_dof = []
for predictor in predictors:
    predictor_name = predictor
    df = data_vars[['CUSTOMER_OUTAGE_HOURS', predictor_name]]
    MSE_ave_min, degree_of_freedom_MSE_ave_min, dof_ose = spline_optimal(df,
                                                                           k,
                                                                           degree,
                                                                          ␣
 ↪degree_of_freedom_max,
                                                                          ␣
 ↪predictor_name,
                                                                          ␣
 ↪response_name)
```

```
    optimal_dof.append(dof_ose)
```

[11]:
```
k = 5
degree_of_freedom_max = 10
response_name = 'CUSTOMER_OUTAGE_HOURS'

predictors = data_vars.drop(['CUSTOMER_OUTAGE_HOURS'], axis=1).columns

optimal_dof_cubic = []
for predictor in predictors:
    predictor_name = predictor
    df = data_vars[['CUSTOMER_OUTAGE_HOURS', predictor_name]]
    MSE_ave_min, degree_of_freedom_MSE_ave_min, dof_ose =␣
 ↪spline_optimal_cubic(df,
                                                                                 k,
                                                                                 ␣
 ↪degree,
                                                                                 ␣
 ↪degree_of_freedom_max,
                                                                                 ␣
 ↪predictor_name,
                                                                                 ␣
 ↪response_name)
    optimal_dof_cubic.append(dof_ose)
```

5 CONFIDENCE INTERVAL equation

[12]:
```
#defining confidence intervals
def confidence_interval(X, y, y_pred):
    mse = np.sum(np.square(y_pred - y)) / y.size
    cov = mse * np.linalg.inv(X.T @ X)
    var_f = np.diagonal((X @ cov) @ X.T)
    se = np.sqrt(abs(var_f))
    conf_int = 2*se
    return conf_int
```

6 INCOME

[13]:
```
income_l = data[data['INCOME_GROUP']=='lower income'].loc[:, ['PRECIPITATION',␣
 ↪'SNOW', 'TEMP_MAX',
                                                                    ␣
 ↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]

income_lm = data[data['INCOME_GROUP']=='lower middle income'].loc[:,␣
 ↪['PRECIPITATION', 'SNOW', 'TEMP_MAX',
```

```
                                                                    ␣
 ↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]

income_um = data[data['INCOME_GROUP']=='upper middle income'].loc[:,␣
 ↪['PRECIPITATION', 'SNOW', 'TEMP_MAX',
                                                                    ␣
 ↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]

income_u = data[data['INCOME_GROUP']=='upper income'].loc[:, ['PRECIPITATION',␣
 ↪'SNOW', 'TEMP_MAX',
                                                                    ␣
 ↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]
```

[14]:
```
plt.figure(figsize=(18, 12), dpi=80)
#### PRECIPITATION ####
df_prcp_l = income_l[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_prcp_lm = income_lm[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_prcp_um = income_um[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_prcp_u = income_u[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]

#### creating confidence intervals
df_prcp_l = df_prcp_l.sample(n=20000, replace=False, random_state=75)
df_prcp_lm = df_prcp_lm.sample(n=20000, replace=False, random_state=75)
df_prcp_um = df_prcp_um.sample(n=20000, replace=False, random_state=75)
df_prcp_u = df_prcp_u.sample(n=20000, replace=False, random_state=75)

df_prcp_l = df_prcp_l.sort_values(by=['PRECIPITATION'])
df_prcp_lm = df_prcp_lm.sort_values(by=['PRECIPITATION'])
df_prcp_um = df_prcp_um.sort_values(by=['PRECIPITATION'])
df_prcp_u = df_prcp_u.sort_values(by=['PRECIPITATION'])

df_prcp_l = df_prcp_l.reset_index(drop = True)
df_prcp_lm = df_prcp_lm.reset_index(drop = True)
df_prcp_um = df_prcp_um.reset_index(drop = True)
df_prcp_u = df_prcp_u.reset_index(drop = True)

X_prcp_l = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_l)
X_prcp_lm = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_lm)
X_prcp_um = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_um)
X_prcp_u = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_u)

y_prcp_l = np.asarray(df_prcp_l['CUSTOMER_OUTAGE_HOURS'])
y_prcp_lm = np.asarray(df_prcp_lm['CUSTOMER_OUTAGE_HOURS'])
y_prcp_um = np.asarray(df_prcp_um['CUSTOMER_OUTAGE_HOURS'])
y_prcp_u = np.asarray(df_prcp_u['CUSTOMER_OUTAGE_HOURS'])
```

```
model_prcp_l = sm.OLS(y_prcp_l, X_prcp_l).fit(disp=0)
model_prcp_lm = sm.OLS(y_prcp_lm, X_prcp_lm).fit(disp=0)
model_prcp_um = sm.OLS(y_prcp_um, X_prcp_um).fit(disp=0)
model_prcp_u = sm.OLS(y_prcp_u, X_prcp_u).fit(disp=0)

y_pred_prcp_l = model_prcp_l.predict(X_prcp_l)
y_pred_prcp_lm = model_prcp_lm.predict(X_prcp_lm)
y_pred_prcp_um = model_prcp_um.predict(X_prcp_um)
y_pred_prcp_u = model_prcp_u.predict(X_prcp_u)

#### visualizing CI's
ci_prcp_l = confidence_interval(X_prcp_l, y_prcp_l, y_pred_prcp_l)
ci_prcp_lm = confidence_interval(X_prcp_lm, y_prcp_lm, y_pred_prcp_lm)
ci_prcp_um = confidence_interval(X_prcp_um, y_prcp_um, y_pred_prcp_um)
ci_prcp_u = confidence_interval(X_prcp_u, y_prcp_u, y_pred_prcp_u)

plt.subplot(2, 2, 1) # row 2, col 2 index 1
plt.title('Natural Cubic Spline of Precipitation Variable by Income Groups')
plt.plot(df_prcp_l['PRECIPITATION'], y_pred_prcp_l, 'b')
plt.plot(df_prcp_lm['PRECIPITATION'], y_pred_prcp_lm, 'g')
plt.plot(df_prcp_um['PRECIPITATION'], y_pred_prcp_um, 'm')
plt.plot(df_prcp_u['PRECIPITATION'], y_pred_prcp_u, 'r')

### CI's
plt.fill_between(df_prcp_l['PRECIPITATION'], y_pred_prcp_l-ci_prcp_l,␣
↪y_pred_prcp_l+ci_prcp_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_prcp_lm['PRECIPITATION'], y_pred_prcp_lm-ci_prcp_lm,␣
↪y_pred_prcp_lm+ci_prcp_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_prcp_um['PRECIPITATION'], y_pred_prcp_um-ci_prcp_um,␣
↪y_pred_prcp_um+ci_prcp_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_prcp_u['PRECIPITATION'], y_pred_prcp_u-ci_prcp_u,␣
↪y_pred_prcp_u+ci_prcp_u, facecolor = 'r', alpha = 0.25)

#plt.scatter(df_prcp.PRECIPITATION, df_prcp.CUSTOMER_OUTAGE_HOURS)
plt.legend(labels = ['lower income', 'lower middle income', 'upper middle␣
↪income', 'upper income'])

plt.xlabel('PRECIPITATION (mm)')
plt.ylabel('CUSTOMER OUTAGE HOURS')


#### SNOW ####
df_snow_l = income_l[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_lm = income_lm[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_um = income_um[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_u = income_u[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
```

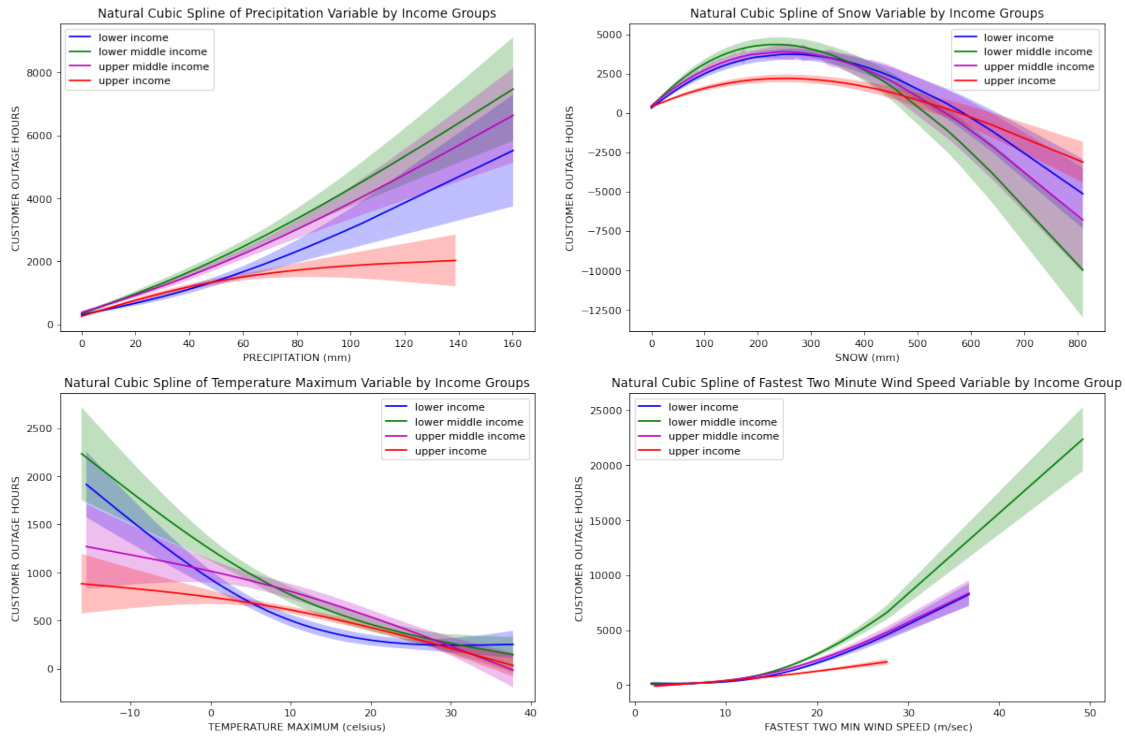
```python
#### creating confidence intervals
df_snow_l = df_snow_l.sample(n=20000, replace=False, random_state=75)
df_snow_lm = df_snow_lm.sample(n=20000, replace=False, random_state=75)
df_snow_um = df_snow_um.sample(n=20000, replace=False, random_state=75)
df_snow_u = df_snow_u.sample(n=20000, replace=False, random_state=75)

df_snow_l = df_snow_l.sort_values(by=['SNOW'])
df_snow_lm = df_snow_lm.sort_values(by=['SNOW'])
df_snow_um = df_snow_um.sort_values(by=['SNOW'])
df_snow_u = df_snow_u.sort_values(by=['SNOW'])


df_snow_l = df_snow_l.reset_index(drop = True)
df_snow_lm = df_snow_lm.reset_index(drop = True)
df_snow_um = df_snow_um.reset_index(drop = True)
df_snow_u = df_snow_u.reset_index(drop = True)


X_snow_l = pt.dmatrix('cr(SNOW, df=3)', df_snow_l)
X_snow_lm = pt.dmatrix('cr(SNOW, df=3)', df_snow_lm)
X_snow_um = pt.dmatrix('cr(SNOW, df=3)', df_snow_um)
X_snow_u = pt.dmatrix('cr(SNOW, df=3)', df_snow_u)


y_snow_l = np.asarray(df_snow_l['CUSTOMER_OUTAGE_HOURS'])
y_snow_lm = np.asarray(df_snow_lm['CUSTOMER_OUTAGE_HOURS'])
y_snow_um = np.asarray(df_snow_um['CUSTOMER_OUTAGE_HOURS'])
y_snow_u = np.asarray(df_snow_u['CUSTOMER_OUTAGE_HOURS'])


model_snow_l = sm.OLS(y_snow_l, X_snow_l).fit(disp=0)
model_snow_lm = sm.OLS(y_snow_lm, X_snow_lm).fit(disp=0)
model_snow_um = sm.OLS(y_snow_um, X_snow_um).fit(disp=0)
model_snow_u = sm.OLS(y_snow_u, X_snow_u).fit(disp=0)


y_pred_snow_l = model_snow_l.predict(X_snow_l)
y_pred_snow_lm = model_snow_lm.predict(X_snow_lm)
y_pred_snow_um = model_snow_um.predict(X_snow_um)
y_pred_snow_u = model_snow_u.predict(X_snow_u)

#### visualizing CI's
ci_snow_l = confidence_interval(X_snow_l, y_snow_l, y_pred_snow_l)
ci_snow_lm = confidence_interval(X_snow_lm, y_snow_lm, y_pred_snow_lm)
ci_snow_um = confidence_interval(X_snow_um, y_snow_um, y_pred_snow_um)
ci_snow_u = confidence_interval(X_snow_u, y_snow_u, y_pred_snow_u)
```

```
#plotting
plt.subplot(2, 2, 2) # row 2, col 2 index 2
plt.title('Natural Cubic Spline of Snow Variable by Income Groups')
plt.plot(df_snow_l['SNOW'], y_pred_snow_l, 'b')
plt.plot(df_snow_lm['SNOW'], y_pred_snow_lm, 'g')
plt.plot(df_snow_um['SNOW'], y_pred_snow_um, 'm')
plt.plot(df_snow_u['SNOW'], y_pred_snow_u, 'r')
plt.legend(labels = ['lower income', 'lower middle income', 'upper middle␣
↪income', 'upper income'])

### CI's
plt.fill_between(df_snow_l['SNOW'], y_pred_snow_l-ci_snow_l,␣
↪y_pred_snow_l+ci_snow_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_snow_lm['SNOW'], y_pred_snow_lm-ci_snow_lm,␣
↪y_pred_snow_lm+ci_snow_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_snow_um['SNOW'], y_pred_snow_um-ci_snow_um,␣
↪y_pred_snow_um+ci_snow_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_snow_u['SNOW'], y_pred_snow_u-ci_snow_u,␣
↪y_pred_snow_u+ci_snow_u, facecolor = 'r', alpha = 0.25)


#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)
plt.xlabel('SNOW (mm)')
plt.ylabel('CUSTOMER OUTAGE HOURS')


#### TEMP_MAX ####
df_tmax_l = income_l[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_lm = income_lm[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_um = income_um[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_u = income_u[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]

#### creating confidence intervals
df_tmax_l = df_tmax_l.sample(n=20000, replace=False, random_state=75)
df_tmax_lm = df_tmax_lm.sample(n=20000, replace=False, random_state=75)
df_tmax_um = df_tmax_um.sample(n=20000, replace=False, random_state=75)
df_tmax_u = df_tmax_u.sample(n=20000, replace=False, random_state=75)

df_tmax_l = df_tmax_l.sort_values(by=['TEMP_MAX'])
df_tmax_lm = df_tmax_lm.sort_values(by=['TEMP_MAX'])
df_tmax_um = df_tmax_um.sort_values(by=['TEMP_MAX'])
df_tmax_u = df_tmax_u.sort_values(by=['TEMP_MAX'])

df_tmax_l = df_tmax_l.reset_index(drop = True)
df_tmax_lm = df_tmax_lm.reset_index(drop = True)
df_tmax_um = df_tmax_um.reset_index(drop = True)
df_tmax_u = df_tmax_u.reset_index(drop = True)
```

```
X_tmax_l = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_l)
X_tmax_lm = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_lm)
X_tmax_um = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_um)
X_tmax_u = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_u)

y_tmax_l = np.asarray(df_tmax_l['CUSTOMER_OUTAGE_HOURS'])
y_tmax_lm = np.asarray(df_tmax_lm['CUSTOMER_OUTAGE_HOURS'])
y_tmax_um = np.asarray(df_tmax_um['CUSTOMER_OUTAGE_HOURS'])
y_tmax_u = np.asarray(df_tmax_u['CUSTOMER_OUTAGE_HOURS'])

model_tmax_l = sm.OLS(y_tmax_l, X_tmax_l).fit(disp=0)
model_tmax_lm = sm.OLS(y_tmax_lm, X_tmax_lm).fit(disp=0)
model_tmax_um = sm.OLS(y_tmax_um, X_tmax_um).fit(disp=0)
model_tmax_u = sm.OLS(y_tmax_u, X_tmax_u).fit(disp=0)

y_pred_tmax_l = model_tmax_l.predict(X_tmax_l)
y_pred_tmax_lm = model_tmax_lm.predict(X_tmax_lm)
y_pred_tmax_um = model_tmax_um.predict(X_tmax_um)
y_pred_tmax_u = model_tmax_u.predict(X_tmax_u)

#### visualizing CI's
ci_tmax_l = confidence_interval(X_tmax_l, y_tmax_l, y_pred_tmax_l)
ci_tmax_lm = confidence_interval(X_tmax_lm, y_tmax_lm, y_pred_tmax_lm)
ci_tmax_um = confidence_interval(X_tmax_um, y_tmax_um, y_pred_tmax_um)
ci_tmax_u = confidence_interval(X_tmax_u, y_tmax_u, y_pred_tmax_u)

plt.subplot(2, 2, 3) # row 2, col 2 index 3
plt.title('Natural Cubic Spline of Temperature Maximum Variable by Income␣
↪Groups')
plt.plot(df_tmax_l['TEMP_MAX'], y_pred_tmax_l, 'b')
plt.plot(df_tmax_lm['TEMP_MAX'], y_pred_tmax_lm, 'g')
plt.plot(df_tmax_um['TEMP_MAX'], y_pred_tmax_um, 'm')
plt.plot(df_tmax_u['TEMP_MAX'], y_pred_tmax_u, 'r')


#plt.scatter(df_tmax.TEMP_MAX, df_tmax.CUSTOMER_OUTAGE_HOURS)

### CI's
plt.fill_between(df_tmax_l['TEMP_MAX'], y_pred_tmax_l-ci_tmax_l,␣
↪y_pred_tmax_l+ci_tmax_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_tmax_lm['TEMP_MAX'], y_pred_tmax_lm-ci_tmax_lm,␣
↪y_pred_tmax_lm+ci_tmax_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_tmax_um['TEMP_MAX'], y_pred_tmax_um-ci_tmax_um,␣
↪y_pred_tmax_um+ci_tmax_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_tmax_u['TEMP_MAX'], y_pred_tmax_u-ci_tmax_u,␣
↪y_pred_tmax_u+ci_tmax_u, facecolor = 'r', alpha = 0.25)
```

```
plt.legend(labels = ['lower income', 'lower middle income', 'upper middle␣
 ↪income', 'upper income'])

plt.xlabel('TEMPERATURE MAXIMUM (celsius)')
plt.ylabel('CUSTOMER OUTAGE HOURS')


#### 2 min windspeed ####
df_2wnd_l = income_l[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_lm = income_lm[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_um = income_um[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_u = income_u[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]

#### creating confidence intervals
df_2wnd_l = df_2wnd_l.sample(n=20000, replace=False, random_state=75)
df_2wnd_lm = df_2wnd_lm.sample(n=20000, replace=False, random_state=75)
df_2wnd_um = df_2wnd_um.sample(n=20000, replace=False, random_state=75)
df_2wnd_u = df_2wnd_u.sample(n=20000, replace=False, random_state=75)

df_2wnd_l = df_2wnd_l.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_lm = df_2wnd_lm.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_um = df_2wnd_um.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_u = df_2wnd_u.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])

df_2wnd_l = df_2wnd_l.reset_index(drop = True)
df_2wnd_lm = df_2wnd_lm.reset_index(drop = True)
df_2wnd_um = df_2wnd_um.reset_index(drop = True)
df_2wnd_u = df_2wnd_u.reset_index(drop = True)

X_2wnd_l = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_l)
X_2wnd_lm = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_lm)
X_2wnd_um = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_um)
X_2wnd_u = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_u)

y_2wnd_l = np.asarray(df_2wnd_l['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_lm = np.asarray(df_2wnd_lm['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_um = np.asarray(df_2wnd_um['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_u = np.asarray(df_2wnd_u['CUSTOMER_OUTAGE_HOURS'])

model_2wnd_l = sm.OLS(y_2wnd_l, X_2wnd_l).fit(disp=0)
model_2wnd_lm = sm.OLS(y_2wnd_lm, X_2wnd_lm).fit(disp=0)
model_2wnd_um = sm.OLS(y_2wnd_um, X_2wnd_um).fit(disp=0)
model_2wnd_u = sm.OLS(y_2wnd_u, X_2wnd_u).fit(disp=0)

y_pred_2wnd_l = model_2wnd_l.predict(X_2wnd_l)
y_pred_2wnd_lm = model_2wnd_lm.predict(X_2wnd_lm)
```

```
y_pred_2wnd_um = model_2wnd_um.predict(X_2wnd_um)
y_pred_2wnd_u = model_2wnd_u.predict(X_2wnd_u)

#### visualizing CI's
ci_2wnd_l = confidence_interval(X_2wnd_l, y_2wnd_l, y_pred_2wnd_l)
ci_2wnd_lm = confidence_interval(X_2wnd_lm, y_2wnd_lm, y_pred_2wnd_lm)
ci_2wnd_um = confidence_interval(X_2wnd_um, y_2wnd_um, y_pred_2wnd_um)
ci_2wnd_u = confidence_interval(X_2wnd_u, y_2wnd_u, y_pred_2wnd_u)

plt.subplot(2, 2, 4) # row 2, col 2 index 4
plt.title('Natural Cubic Spline of Fastest Two Minute Wind Speed Variable by␣
↪Income Group')
plt.plot(df_2wnd_l['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_l, 'b')
plt.plot(df_2wnd_lm['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_lm, 'g')
plt.plot(df_2wnd_um['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_um, 'm')
plt.plot(df_2wnd_u['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_u, 'r')

### CI's
plt.fill_between(df_2wnd_l['FASTEST_TWO_MIN_WIND_SPEED'],␣
↪y_pred_2wnd_l-ci_2wnd_l, y_pred_2wnd_l+ci_2wnd_l, facecolor = 'b', alpha = 0.
↪25)
plt.fill_between(df_2wnd_lm['FASTEST_TWO_MIN_WIND_SPEED'],␣
↪y_pred_2wnd_lm-ci_2wnd_lm, y_pred_2wnd_lm+ci_2wnd_lm, facecolor = 'g', alpha␣
↪= 0.25)
plt.fill_between(df_2wnd_um['FASTEST_TWO_MIN_WIND_SPEED'],␣
↪y_pred_2wnd_um-ci_2wnd_um, y_pred_2wnd_um+ci_2wnd_um, facecolor = 'm', alpha␣
↪= 0.25)
plt.fill_between(df_2wnd_u['FASTEST_TWO_MIN_WIND_SPEED'],␣
↪y_pred_2wnd_u-ci_2wnd_u, y_pred_2wnd_u+ci_2wnd_u, facecolor = 'r', alpha = 0.
↪25)


#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)
plt.xlabel(' FASTEST TWO MIN WIND SPEED (m/sec)')
plt.ylabel('CUSTOMER OUTAGE HOURS')
plt.legend(labels = ['lower income', 'lower middle income', 'upper middle␣
↪income', 'upper income'])

plt.show()
```



[15]:

```
MSE_prcp_l = mean_squared_error(y_prcp_l, y_pred_prcp_l)
MSE_prcp_lm = mean_squared_error(y_prcp_lm, y_pred_prcp_lm)
MSE_prcp_um = mean_squared_error(y_prcp_um, y_pred_prcp_um)
MSE_prcp_u = mean_squared_error(y_prcp_u, y_pred_prcp_u)

print('LOWER INCOME MSE PRECIPITATION is: ', "{:.2f}".format(MSE_prcp_l))
print('LOWER MIDDLE INCOME MSE PRECIPITATION is: ', "{:.2f}".
 ↪format(MSE_prcp_lm))
print('UPPER MIDDLE INCOME MSE PRECIPITATION is: ', "{:.2f}".
 ↪format(MSE_prcp_um))
print('UPPER INCOME MSE PRECIPITATION is: ', "{:.2f}".format(MSE_prcp_u))

print()

MSE_snow_l = mean_squared_error(y_snow_l, y_pred_snow_l)
MSE_snow_lm = mean_squared_error(y_snow_lm, y_pred_snow_lm)
MSE_snow_um = mean_squared_error(y_snow_um, y_pred_snow_um)
MSE_snow_u = mean_squared_error(y_snow_u, y_pred_snow_u)

print('LOWER INCOME MSE SNOW is: ', "{:.2f}".format(MSE_snow_l))
print('LOWER MIDDLE INCOME MSE SNOW is: ', "{:.2f}".format(MSE_snow_lm))
print('UPPER MIDDLE INCOME MSE SNOW is: ', "{:.2f}".format(MSE_snow_um))
print('UPPER INCOME MSE SNOW is: ', "{:.2f}".format(MSE_snow_u))
```

```
print()

MSE_tmax_l = mean_squared_error(y_tmax_l, y_pred_tmax_l)
MSE_tmax_lm = mean_squared_error(y_tmax_lm, y_pred_tmax_lm)
MSE_tmax_um = mean_squared_error(y_tmax_um, y_pred_tmax_um)
MSE_tmax_u = mean_squared_error(y_tmax_u, y_pred_tmax_u)

print('LOWER INCOME MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_l))
print('LOWER MIDDLE INCOME MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_lm))
print('UPPER MIDDLE INCOME MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_um))
print('UPPER INCOME MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_u))

print()

MSE_2wnd_l = mean_squared_error(y_2wnd_l, y_pred_2wnd_l)
MSE_2wnd_lm = mean_squared_error(y_2wnd_lm, y_pred_2wnd_lm)
MSE_2wnd_um = mean_squared_error(y_2wnd_um, y_pred_2wnd_um)
MSE_2wnd_u = mean_squared_error(y_2wnd_u, y_pred_2wnd_u)

print('LOWER INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_l))
print('LOWER MIDDLE INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_lm))
print('UPPER MIDDLE INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_um))
print('UPPER INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_u))
```

```
LOWER INCOME MSE PRECIPITATION is:  8641877.60
LOWER MIDDLE INCOME MSE PRECIPITATION is:  15987647.48
UPPER MIDDLE INCOME MSE PRECIPITATION is:  12570721.14
UPPER INCOME MSE PRECIPITATION is:  5730930.63

LOWER INCOME MSE SNOW is:  8568799.87
LOWER MIDDLE INCOME MSE SNOW is:  16067842.76
UPPER MIDDLE INCOME MSE SNOW is:  12652864.80
UPPER INCOME MSE SNOW is:  5787813.47

LOWER INCOME MSE TEMP_MAX is:  8667576.20
LOWER MIDDLE INCOME MSE TEMP_MAX is:  16180469.22
UPPER MIDDLE INCOME MSE TEMP_MAX is:  12742515.00
UPPER INCOME MSE TEMP_MAX is:  5812322.00

LOWER INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is:  8505761.62
LOWER MIDDLE INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is:  15780393.90
UPPER MIDDLE INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is:  12448020.07
```

```
UPPER INCOME MSE FASTEST_TWO_MIN_WIND_SPEED is:  5725546.81
```

7 DIVERSITY

[16]:
```
diversity_l = data[data['DIVERSITY_GROUP']=='low diversity'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX', 'FASTEST_TWO_MIN_WIND_SPEED',␣
↪'CUSTOMER_OUTAGE_HOURS']]
diversity_lm = data[data['DIVERSITY_GROUP']=='low middle diversity'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX', 'FASTEST_TWO_MIN_WIND_SPEED',␣
↪'CUSTOMER_OUTAGE_HOURS']]
diversity_um = data[data['DIVERSITY_GROUP']=='high middle diversity'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX', 'FASTEST_TWO_MIN_WIND_SPEED',␣
↪'CUSTOMER_OUTAGE_HOURS']]
diversity_u = data[data['DIVERSITY_GROUP']=='high diversity'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX', 'FASTEST_TWO_MIN_WIND_SPEED',␣
↪'CUSTOMER_OUTAGE_HOURS']]
```

[17]:
```
plt.figure(figsize=(18, 12), dpi=80)
######## PRECIPITATION ########
df_prcp_l = diversity_l[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_prcp_lm = diversity_lm[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_prcp_um = diversity_um[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_prcp_u = diversity_u[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]

#### creating confidence intervals
df_prcp_l = df_prcp_l.sample(n=20000, replace=False, random_state=75)
df_prcp_lm = df_prcp_lm.sample(n=20000, replace=False, random_state=75)
df_prcp_um = df_prcp_um.sample(n=20000, replace=False, random_state=75)
df_prcp_u = df_prcp_u.sample(n=20000, replace=False, random_state=75)

df_prcp_l = df_prcp_l.sort_values(by=['PRECIPITATION'])
df_prcp_lm = df_prcp_lm.sort_values(by=['PRECIPITATION'])
df_prcp_um = df_prcp_um.sort_values(by=['PRECIPITATION'])
df_prcp_u = df_prcp_u.sort_values(by=['PRECIPITATION'])

df_prcp_l = df_prcp_l.reset_index(drop = True)
df_prcp_lm = df_prcp_lm.reset_index(drop = True)
df_prcp_um = df_prcp_um.reset_index(drop = True)
df_prcp_u = df_prcp_u.reset_index(drop = True)

X_prcp_l = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_l)
X_prcp_lm = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_lm)
X_prcp_um = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_um)
X_prcp_u = pt.dmatrix('cr(PRECIPITATION, df=3)', df_prcp_u)

y_prcp_l = np.asarray(df_prcp_l['CUSTOMER_OUTAGE_HOURS'])
y_prcp_lm = np.asarray(df_prcp_lm['CUSTOMER_OUTAGE_HOURS'])
```

```
y_prcp_um = np.asarray(df_prcp_um['CUSTOMER_OUTAGE_HOURS'])
y_prcp_u = np.asarray(df_prcp_u['CUSTOMER_OUTAGE_HOURS'])

model_prcp_l = sm.OLS(y_prcp_l, X_prcp_l).fit(disp=0)
model_prcp_lm = sm.OLS(y_prcp_lm, X_prcp_lm).fit(disp=0)
model_prcp_um = sm.OLS(y_prcp_um, X_prcp_um).fit(disp=0)
model_prcp_u = sm.OLS(y_prcp_u, X_prcp_u).fit(disp=0)

y_pred_prcp_l = model_prcp_l.predict(X_prcp_l)
y_pred_prcp_lm = model_prcp_lm.predict(X_prcp_lm)
y_pred_prcp_um = model_prcp_um.predict(X_prcp_um)
y_pred_prcp_u = model_prcp_u.predict(X_prcp_u)

#### visualizing CI's
ci_prcp_l = confidence_interval(X_prcp_l, y_prcp_l, y_pred_prcp_l)
ci_prcp_lm = confidence_interval(X_prcp_lm, y_prcp_lm, y_pred_prcp_lm)
ci_prcp_um = confidence_interval(X_prcp_um, y_prcp_um, y_pred_prcp_um)
ci_prcp_u = confidence_interval(X_prcp_u, y_prcp_u, y_pred_prcp_u)

plt.subplot(2, 2, 1) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Precipitation Variable by Diversity Groups')
plt.plot(df_prcp_l['PRECIPITATION'], y_pred_prcp_l, 'b')
plt.plot(df_prcp_lm['PRECIPITATION'], y_pred_prcp_lm, 'g')
plt.plot(df_prcp_um['PRECIPITATION'], y_pred_prcp_um, 'm')
plt.plot(df_prcp_u['PRECIPITATION'], y_pred_prcp_u, 'r')
#plt.scatter(df_prcp.PRECIPITATION, df_prcp.CUSTOMER_OUTAGE_HOURS)

### CI's
plt.fill_between(df_prcp_l['PRECIPITATION'], y_pred_prcp_l-ci_prcp_l,␣
 ↪y_pred_prcp_l+ci_prcp_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_prcp_lm['PRECIPITATION'], y_pred_prcp_lm-ci_prcp_lm,␣
 ↪y_pred_prcp_lm+ci_prcp_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_prcp_um['PRECIPITATION'], y_pred_prcp_um-ci_prcp_um,␣
 ↪y_pred_prcp_um+ci_prcp_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_prcp_u['PRECIPITATION'], y_pred_prcp_u-ci_prcp_u,␣
 ↪y_pred_prcp_u+ci_prcp_u, facecolor = 'r', alpha = 0.25)

plt.legend(labels = ['low diversity', 'low middle diversity', 'high middle␣
 ↪diversity', 'high diversity'], loc='upper left')
plt.xlabel('PRECIPITATION (mm)')
plt.ylabel('CUSTOMER OUTAGE HOURS')

######## SNOW #######
df_snow_l = diversity_l[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_lm = diversity_lm[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_um = diversity_um[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_u = diversity_u[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
```

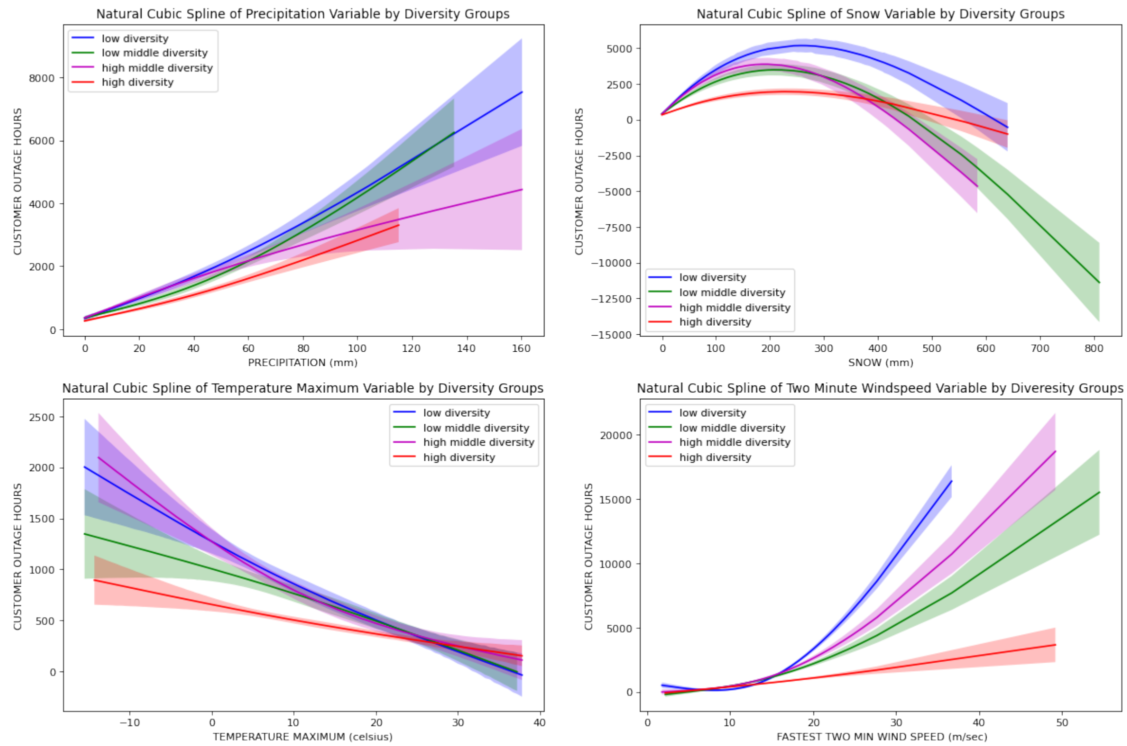
```
#### creating confidence intervals
df_snow_l = df_snow_l.sample(n=20000, replace=False, random_state=75)
df_snow_lm = df_snow_lm.sample(n=20000, replace=False, random_state=75)
df_snow_um = df_snow_um.sample(n=20000, replace=False, random_state=75)
df_snow_u = df_snow_u.sample(n=20000, replace=False, random_state=75)

df_snow_l = df_snow_l.sort_values(by=['SNOW'])
df_snow_lm = df_snow_lm.sort_values(by=['SNOW'])
df_snow_um = df_snow_um.sort_values(by=['SNOW'])
df_snow_u = df_snow_u.sort_values(by=['SNOW'])

df_snow_l = df_snow_l.reset_index(drop = True)
df_snow_lm = df_snow_lm.reset_index(drop = True)
df_snow_um = df_snow_um.reset_index(drop = True)
df_snow_u = df_snow_u.reset_index(drop = True)

X_snow_l = pt.dmatrix('cr(SNOW, df=3)', df_snow_l)
X_snow_lm = pt.dmatrix('cr(SNOW, df=3)', df_snow_lm)
X_snow_um = pt.dmatrix('cr(SNOW, df=3)', df_snow_um)
X_snow_u = pt.dmatrix('cr(SNOW, df=3)', df_snow_u)

y_snow_l = np.asarray(df_snow_l['CUSTOMER_OUTAGE_HOURS'])
y_snow_lm = np.asarray(df_snow_lm['CUSTOMER_OUTAGE_HOURS'])
y_snow_um = np.asarray(df_snow_um['CUSTOMER_OUTAGE_HOURS'])
y_snow_u = np.asarray(df_snow_u['CUSTOMER_OUTAGE_HOURS'])

model_snow_l = sm.OLS(y_snow_l, X_snow_l).fit(disp=0)
model_snow_lm = sm.OLS(y_snow_lm, X_snow_lm).fit(disp=0)
model_snow_um = sm.OLS(y_snow_um, X_snow_um).fit(disp=0)
model_snow_u = sm.OLS(y_snow_u, X_snow_u).fit(disp=0)

y_pred_snow_l = model_snow_l.predict(X_snow_l)
y_pred_snow_lm = model_snow_lm.predict(X_snow_lm)
y_pred_snow_um = model_snow_um.predict(X_snow_um)
y_pred_snow_u = model_snow_u.predict(X_snow_u)

#### visualizing CI's
ci_snow_l = confidence_interval(X_snow_l, y_snow_l, y_pred_snow_l)
ci_snow_lm = confidence_interval(X_snow_lm, y_snow_lm, y_pred_snow_lm)
ci_snow_um = confidence_interval(X_snow_um, y_snow_um, y_pred_snow_um)
ci_snow_u = confidence_interval(X_snow_u, y_snow_u, y_pred_snow_u)

plt.subplot(2, 2, 2) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Snow Variable by Diversity Groups')
plt.plot(df_snow_l['SNOW'], y_pred_snow_l, 'b')
plt.plot(df_snow_lm['SNOW'], y_pred_snow_lm, 'g')
```

```
plt.plot(df_snow_um['SNOW'], y_pred_snow_um, 'm')
plt.plot(df_snow_u['SNOW'], y_pred_snow_u, 'r')
plt.legend(labels = ['low diversity', 'low middle diversity', 'high middle␣
↪diversity', 'high diversity'], loc='lower left')

#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)

### CI's
plt.fill_between(df_snow_l['SNOW'], y_pred_snow_l-ci_snow_l,␣
↪y_pred_snow_l+ci_snow_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_snow_lm['SNOW'], y_pred_snow_lm-ci_snow_lm,␣
↪y_pred_snow_lm+ci_snow_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_snow_um['SNOW'], y_pred_snow_um-ci_snow_um,␣
↪y_pred_snow_um+ci_snow_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_snow_u['SNOW'], y_pred_snow_u-ci_snow_u,␣
↪y_pred_snow_u+ci_snow_u, facecolor = 'r', alpha = 0.25)

plt.xlabel('SNOW (mm)')
plt.ylabel('CUSTOMER OUTAGE HOURS')

################ T MAX ########
df_tmax_l = diversity_l[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_lm = diversity_lm[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_um = diversity_um[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_u = diversity_u[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]

#### creating confidence intervals
df_tmax_l = df_tmax_l.sample(n=20000, replace=False, random_state=75)
df_tmax_lm = df_tmax_lm.sample(n=20000, replace=False, random_state=75)
df_tmax_um = df_tmax_um.sample(n=20000, replace=False, random_state=75)
df_tmax_u = df_tmax_u.sample(n=20000, replace=False, random_state=75)

df_tmax_l = df_tmax_l.sort_values(by=['TEMP_MAX'])
df_tmax_lm = df_tmax_lm.sort_values(by=['TEMP_MAX'])
df_tmax_um = df_tmax_um.sort_values(by=['TEMP_MAX'])
df_tmax_u = df_tmax_u.sort_values(by=['TEMP_MAX'])

df_tmax_l = df_tmax_l.reset_index(drop = True)
df_tmax_lm = df_tmax_lm.reset_index(drop = True)
df_tmax_um = df_tmax_um.reset_index(drop = True)
df_tmax_u = df_tmax_u.reset_index(drop = True)

X_tmax_l = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_l)
X_tmax_lm = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_lm)
X_tmax_um = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_um)
X_tmax_u = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_u)
```

```
y_tmax_l = np.asarray(df_tmax_l['CUSTOMER_OUTAGE_HOURS'])
y_tmax_lm = np.asarray(df_tmax_lm['CUSTOMER_OUTAGE_HOURS'])
y_tmax_um = np.asarray(df_tmax_um['CUSTOMER_OUTAGE_HOURS'])
y_tmax_u = np.asarray(df_tmax_u['CUSTOMER_OUTAGE_HOURS'])

model_tmax_l = sm.OLS(y_tmax_l, X_tmax_l).fit(disp=0)
model_tmax_lm = sm.OLS(y_tmax_lm, X_tmax_lm).fit(disp=0)
model_tmax_um = sm.OLS(y_tmax_um, X_tmax_um).fit(disp=0)
model_tmax_u = sm.OLS(y_tmax_u, X_tmax_u).fit(disp=0)

y_pred_tmax_l = model_tmax_l.predict(X_tmax_l)
y_pred_tmax_lm = model_tmax_lm.predict(X_tmax_lm)
y_pred_tmax_um = model_tmax_um.predict(X_tmax_um)
y_pred_tmax_u = model_tmax_u.predict(X_tmax_u)

#### visualizing CI's
ci_tmax_l = confidence_interval(X_tmax_l, y_tmax_l, y_pred_tmax_l)
ci_tmax_lm = confidence_interval(X_tmax_lm, y_tmax_lm, y_pred_tmax_lm)
ci_tmax_um = confidence_interval(X_tmax_um, y_tmax_um, y_pred_tmax_um)
ci_tmax_u = confidence_interval(X_tmax_u, y_tmax_u, y_pred_tmax_u)

plt.subplot(2, 2, 3) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Temperature Maximum Variable by Diversity␣
↪Groups')
plt.plot(df_tmax_l['TEMP_MAX'], y_pred_tmax_l, 'b')
plt.plot(df_tmax_lm['TEMP_MAX'], y_pred_tmax_lm, 'g')
plt.plot(df_tmax_um['TEMP_MAX'], y_pred_tmax_um, 'm')
plt.plot(df_tmax_u['TEMP_MAX'], y_pred_tmax_u, 'r')
#plt.scatter(df_tmax.TEMP_MAX, df_tmax.CUSTOMER_OUTAGE_HOURS)

### CI's
plt.fill_between(df_tmax_l['TEMP_MAX'], y_pred_tmax_l-ci_tmax_l,␣
↪y_pred_tmax_l+ci_tmax_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_tmax_lm['TEMP_MAX'], y_pred_tmax_lm-ci_tmax_lm,␣
↪y_pred_tmax_lm+ci_tmax_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_tmax_um['TEMP_MAX'], y_pred_tmax_um-ci_tmax_um,␣
↪y_pred_tmax_um+ci_tmax_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_tmax_u['TEMP_MAX'], y_pred_tmax_u-ci_tmax_u,␣
↪y_pred_tmax_u+ci_tmax_u, facecolor = 'r', alpha = 0.25)

plt.legend(labels = ['low diversity', 'low middle diversity', 'high middle␣
↪diversity', 'high diversity'])
plt.xlabel('TEMPERATURE MAXIMUM (celsius)')
plt.ylabel('CUSTOMER OUTAGE HOURS')
```

```
############# 2 min windspeed ##########

df_2wnd_l = diversity_l[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_lm = diversity_lm[['CUSTOMER_OUTAGE_HOURS',␣
↪'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_um = diversity_um[['CUSTOMER_OUTAGE_HOURS',␣
↪'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_u = diversity_u[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]

#### creating confidence intervals
df_2wnd_l = df_2wnd_l.sample(n=20000, replace=False, random_state=75)
df_2wnd_lm = df_2wnd_lm.sample(n=20000, replace=False, random_state=75)
df_2wnd_um = df_2wnd_um.sample(n=20000, replace=False, random_state=75)
df_2wnd_u = df_2wnd_u.sample(n=20000, replace=False, random_state=75)

df_2wnd_l = df_2wnd_l.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_lm = df_2wnd_lm.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_um = df_2wnd_um.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_u = df_2wnd_u.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])

df_2wnd_l = df_2wnd_l.reset_index(drop = True)
df_2wnd_lm = df_2wnd_lm.reset_index(drop = True)
df_2wnd_um = df_2wnd_um.reset_index(drop = True)
df_2wnd_u = df_2wnd_u.reset_index(drop = True)

X_2wnd_l = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_l)
X_2wnd_lm = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_lm)
X_2wnd_um = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_um)
X_2wnd_u = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_u)

y_2wnd_l = np.asarray(df_2wnd_l['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_lm = np.asarray(df_2wnd_lm['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_um = np.asarray(df_2wnd_um['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_u = np.asarray(df_2wnd_u['CUSTOMER_OUTAGE_HOURS'])

model_2wnd_l = sm.OLS(y_2wnd_l, X_2wnd_l).fit(disp=0)
model_2wnd_lm = sm.OLS(y_2wnd_lm, X_2wnd_lm).fit(disp=0)
model_2wnd_um = sm.OLS(y_2wnd_um, X_2wnd_um).fit(disp=0)
model_2wnd_u = sm.OLS(y_2wnd_u, X_2wnd_u).fit(disp=0)

y_pred_2wnd_l = model_2wnd_l.predict(X_2wnd_l)
y_pred_2wnd_lm = model_2wnd_lm.predict(X_2wnd_lm)
y_pred_2wnd_um = model_2wnd_um.predict(X_2wnd_um)
y_pred_2wnd_u = model_2wnd_u.predict(X_2wnd_u)

#### visualizing CI's
ci_2wnd_l = confidence_interval(X_2wnd_l, y_2wnd_l, y_pred_2wnd_l)
```

```
ci_2wnd_lm = confidence_interval(X_2wnd_lm, y_2wnd_lm, y_pred_2wnd_lm)
ci_2wnd_um = confidence_interval(X_2wnd_um, y_2wnd_um, y_pred_2wnd_um)
ci_2wnd_u = confidence_interval(X_2wnd_u, y_2wnd_u, y_pred_2wnd_u)

plt.subplot(2, 2, 4) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Two Minute Windspeed Variable by Diveresity␣
 ↪Groups')
plt.plot(df_2wnd_l['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_l, 'b')
plt.plot(df_2wnd_lm['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_lm, 'g')
plt.plot(df_2wnd_um['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_um, 'm')
plt.plot(df_2wnd_u['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_u, 'r')
#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)

### CI's
plt.fill_between(df_2wnd_l['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_l-ci_2wnd_l, y_pred_2wnd_l+ci_2wnd_l, facecolor = 'b', alpha = 0.
 ↪25)
plt.fill_between(df_2wnd_lm['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_lm-ci_2wnd_lm, y_pred_2wnd_lm+ci_2wnd_lm, facecolor = 'g', alpha␣
 ↪= 0.25)
plt.fill_between(df_2wnd_um['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_um-ci_2wnd_um, y_pred_2wnd_um+ci_2wnd_um, facecolor = 'm', alpha␣
 ↪= 0.25)
plt.fill_between(df_2wnd_u['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_u-ci_2wnd_u, y_pred_2wnd_u+ci_2wnd_u, facecolor = 'r', alpha = 0.
 ↪25)

plt.xlabel(' FASTEST TWO MIN WIND SPEED (m/sec)')
plt.ylabel('CUSTOMER OUTAGE HOURS')
plt.legend(labels = ['low diversity', 'low middle diversity', 'high middle␣
 ↪diversity', 'high diversity'])



plt.show()
```



[18]:

```
MSE_prcp_l = mean_squared_error(y_prcp_l, y_pred_prcp_l)
MSE_prcp_lm = mean_squared_error(y_prcp_lm, y_pred_prcp_lm)
MSE_prcp_um = mean_squared_error(y_prcp_um, y_pred_prcp_um)
MSE_prcp_u = mean_squared_error(y_prcp_u, y_pred_prcp_u)

print('LOWER DIVERSITY MSE PRECIPITATION is: ', "{:.2f}".format(MSE_prcp_l))
print('LOWER MIDDLE DIVERSITY MSE PRECIPITATION is: ', "{:.2f}".
 ↪format(MSE_prcp_lm))
print('UPPER DIVERSITY MSE PRECIPITATION is: ', "{:.2f}".format(MSE_prcp_um))
print('UPPER MIDDLE DIVERSITY MSE PRECIPITATION is: ', "{:.2f}".
 ↪format(MSE_prcp_u))

print()

MSE_snow_l = mean_squared_error(y_snow_l, y_pred_snow_l)
MSE_snow_lm = mean_squared_error(y_snow_lm, y_pred_snow_lm)
MSE_snow_um = mean_squared_error(y_snow_um, y_pred_snow_um)
MSE_snow_u = mean_squared_error(y_snow_u, y_pred_snow_u)

print('LOWER DIVERSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_l))
print('LOWER MIDDLE DIVERSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_lm))
print('UPPER MIDDLE DIVERSITYMSE SNOW is: ', "{:.2f}".format(MSE_snow_um))
print('UPPER DIVERSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_u))
```

```
print()

MSE_tmax_l = mean_squared_error(y_tmax_l, y_pred_tmax_l)
MSE_tmax_lm = mean_squared_error(y_tmax_lm, y_pred_tmax_lm)
MSE_tmax_um = mean_squared_error(y_tmax_um, y_pred_tmax_um)
MSE_tmax_u = mean_squared_error(y_tmax_u, y_pred_tmax_u)

print('LOWER DIVERSITY MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_l))
print('LOWER MIDDLE DIVERSITY MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_lm))
print('UPPER MIDDLE DIVERSITY MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_um))
print('UPPER MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_u))

print()

MSE_2wnd_l = mean_squared_error(y_2wnd_l, y_pred_2wnd_l)
MSE_2wnd_lm = mean_squared_error(y_2wnd_lm, y_pred_2wnd_lm)
MSE_2wnd_um = mean_squared_error(y_2wnd_um, y_pred_2wnd_um)
MSE_2wnd_u = mean_squared_error(y_2wnd_u, y_pred_2wnd_u)

print('LOWER DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_l))
print('LOWER MIDDLE DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_lm))
print('UPPER MIDDLE DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_um))
print('UPPER DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_u))
```

```
LOWER DIVERSITY MSE PRECIPITATION is:  15968231.67
LOWER MIDDLE DIVERSITY MSE PRECIPITATION is:  13798459.94
UPPER DIVERSITY MSE PRECIPITATION is:  15914845.68
UPPER MIDDLE DIVERSITY MSE PRECIPITATION is:  4598342.42

LOWER DIVERSITY MSE SNOW is:  15982293.79
LOWER MIDDLE DIVERSITY MSE SNOW is:  13883164.39
UPPER MIDDLE DIVERSITYMSE SNOW is:  15914630.34
UPPER DIVERSITY MSE SNOW is:  4654937.14

LOWER DIVERSITY MSE TEMP_MAX is:  16176116.25
LOWER MIDDLE DIVERSITY MSE TEMP_MAX is:  13982188.23
UPPER MIDDLE DIVERSITY MSE TEMP_MAX is:  16001096.32
UPPER MSE TEMP_MAX is:  4679671.67

LOWER DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is:  15421005.83
LOWER MIDDLE DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is:  13697676.95
UPPER MIDDLE DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is:  15701788.62
```

```
UPPER DIVERSITY MSE FASTEST_TWO_MIN_WIND_SPEED is:  4619300.91
```

8 DENSITY

[19]:
```
density_l = data[data['DENSITY_GROUP']=='low density'].loc[:, ['PRECIPITATION',␣
↪'SNOW', 'TEMP_MAX',
                                                                               ␣
↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]

density_lm = data[data['DENSITY_GROUP']=='low middle density'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX',
                                                                               ␣
↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]

density_um = data[data['DENSITY_GROUP']=='high middle density'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX',
                                                                               ␣
↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]

density_u = data[data['DENSITY_GROUP']=='high density'].loc[:,␣
↪['PRECIPITATION', 'SNOW', 'TEMP_MAX',
                                                                               ␣
↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]
```

[20]:
```
plt.figure(figsize=(18, 12), dpi=80)
### PRECIPITATION ####
df_precipitation_l = density_l[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_precipitation_lm = density_lm[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_precipitation_um = density_um[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]
df_precipitation_u = density_u[['CUSTOMER_OUTAGE_HOURS', 'PRECIPITATION']]

#### creating confidence intervals
df_precipitation_l = df_precipitation_l.sample(n=20000, replace=False,␣
↪random_state=75)
df_precipitation_lm = df_precipitation_lm.sample(n=20000, replace=False,␣
↪random_state=75)
df_precipitation_um = df_precipitation_um.sample(n=20000, replace=False,␣
↪random_state=75)
df_precipitation_u = df_precipitation_u.sample(n=20000, replace=False,␣
↪random_state=75)

df_precipitation_l = df_precipitation_l.sort_values(by=['PRECIPITATION'])
df_precipitation_lm = df_precipitation_lm.sort_values(by=['PRECIPITATION'])
df_precipitation_um = df_precipitation_um.sort_values(by=['PRECIPITATION'])
df_precipitation_u = df_precipitation_u.sort_values(by=['PRECIPITATION'])
```

```
df_precipitation_l = df_precipitation_l.reset_index(drop = True)
df_precipitation_lm = df_precipitation_lm.reset_index(drop = True)
df_precipitation_um = df_precipitation_um.reset_index(drop = True)
df_precipitation_u = df_precipitation_u.reset_index(drop = True)

X_precipitation_l = pt.dmatrix('cr(PRECIPITATION, df=3)', df_precipitation_l)
X_precipitation_lm = pt.dmatrix('cr(PRECIPITATION, df=3)', df_precipitation_lm)
X_precipitation_um = pt.dmatrix('cr(PRECIPITATION, df=3)', df_precipitation_um)
X_precipitation_u = pt.dmatrix('cr(PRECIPITATION, df=3)', df_precipitation_u)

y_precipitation_l = np.asarray(df_precipitation_l['CUSTOMER_OUTAGE_HOURS'])
y_precipitation_lm = np.asarray(df_precipitation_lm['CUSTOMER_OUTAGE_HOURS'])
y_precipitation_um = np.asarray(df_precipitation_um['CUSTOMER_OUTAGE_HOURS'])
y_precipitation_u = np.asarray(df_precipitation_u['CUSTOMER_OUTAGE_HOURS'])

model_precipitation_l = sm.OLS(y_precipitation_l, X_precipitation_l).fit(disp=0)
model_precipitation_lm = sm.OLS(y_precipitation_lm, X_precipitation_lm).
 ↪fit(disp=0)
model_precipitation_um = sm.OLS(y_precipitation_um, X_precipitation_um).
 ↪fit(disp=0)
model_precipitation_u = sm.OLS(y_precipitation_u, X_precipitation_u).fit(disp=0)

y_pred_precipitation_l = model_precipitation_l.predict(X_precipitation_l)
y_pred_precipitation_lm = model_precipitation_lm.predict(X_precipitation_lm)
y_pred_precipitation_um = model_precipitation_um.predict(X_precipitation_um)
y_pred_precipitation_u = model_precipitation_u.predict(X_precipitation_u)

#### visualizing CI's
ci_precipitation_l = confidence_interval(X_precipitation_l, y_precipitation_l,␣
 ↪y_pred_precipitation_l)
ci_precipitation_lm = confidence_interval(X_precipitation_lm,␣
 ↪y_precipitation_lm, y_pred_precipitation_lm)
ci_precipitation_um = confidence_interval(X_precipitation_um,␣
 ↪y_precipitation_um, y_pred_precipitation_um)
ci_precipitation_u = confidence_interval(X_precipitation_u, y_precipitation_u,␣
 ↪y_pred_precipitation_u)

###
plt.subplot(2, 2, 1) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Precipitation Variable by Density Groups')
plt.plot(df_precipitation_l['PRECIPITATION'], y_pred_precipitation_l, 'b')
plt.plot(df_precipitation_lm['PRECIPITATION'], y_pred_precipitation_lm, 'g')
plt.plot(df_precipitation_um['PRECIPITATION'], y_pred_precipitation_um, 'm')
plt.plot(df_precipitation_u['PRECIPITATION'], y_pred_precipitation_u, 'r')
```

```
### CI's
plt.fill_between(df_precipitation_l['PRECIPITATION'],␣
 ↪y_pred_precipitation_l-ci_precipitation_l,␣
 ↪y_pred_precipitation_l+ci_precipitation_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_precipitation_lm['PRECIPITATION'],␣
 ↪y_pred_precipitation_lm-ci_precipitation_lm,␣
 ↪y_pred_precipitation_lm+ci_precipitation_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_precipitation_um['PRECIPITATION'],␣
 ↪y_pred_precipitation_um-ci_precipitation_um,␣
 ↪y_pred_precipitation_um+ci_precipitation_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_precipitation_u['PRECIPITATION'],␣
 ↪y_pred_precipitation_u-ci_precipitation_u,␣
 ↪y_pred_precipitation_u+ci_precipitation_u, facecolor = 'r', alpha = 0.25)

#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)
plt.legend(labels = ['low density', 'low middle density', 'high middle␣
 ↪density', 'high density'])
plt.xlabel('PRECIPITATION (mm)')
plt.ylabel('CUSTOMER OUTAGE HOURS')

#### SNOW ####

df_snow_l = density_l[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_lm = density_lm[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_um = density_um[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]
df_snow_u = density_u[['CUSTOMER_OUTAGE_HOURS', 'SNOW']]

#### creating confidence intervals
df_snow_l = df_snow_l.sample(n=20000, replace=False, random_state=75)
df_snow_lm = df_snow_lm.sample(n=20000, replace=False, random_state=75)
df_snow_um = df_snow_um.sample(n=20000, replace=False, random_state=75)
df_snow_u = df_snow_u.sample(n=20000, replace=False, random_state=75)

df_snow_l = df_snow_l.sort_values(by=['SNOW'])
df_snow_lm = df_snow_lm.sort_values(by=['SNOW'])
df_snow_um = df_snow_um.sort_values(by=['SNOW'])
df_snow_u = df_snow_u.sort_values(by=['SNOW'])

df_snow_l = df_snow_l.reset_index(drop = True)
df_snow_lm = df_snow_lm.reset_index(drop = True)
df_snow_um = df_snow_um.reset_index(drop = True)
df_snow_u = df_snow_u.reset_index(drop = True)

X_snow_l = pt.dmatrix('cr(SNOW, df=3)', df_snow_l)
X_snow_lm = pt.dmatrix('cr(SNOW, df=3)', df_snow_lm)
X_snow_um = pt.dmatrix('cr(SNOW, df=3)', df_snow_um)
X_snow_u = pt.dmatrix('cr(SNOW, df=3)', df_snow_u)
```

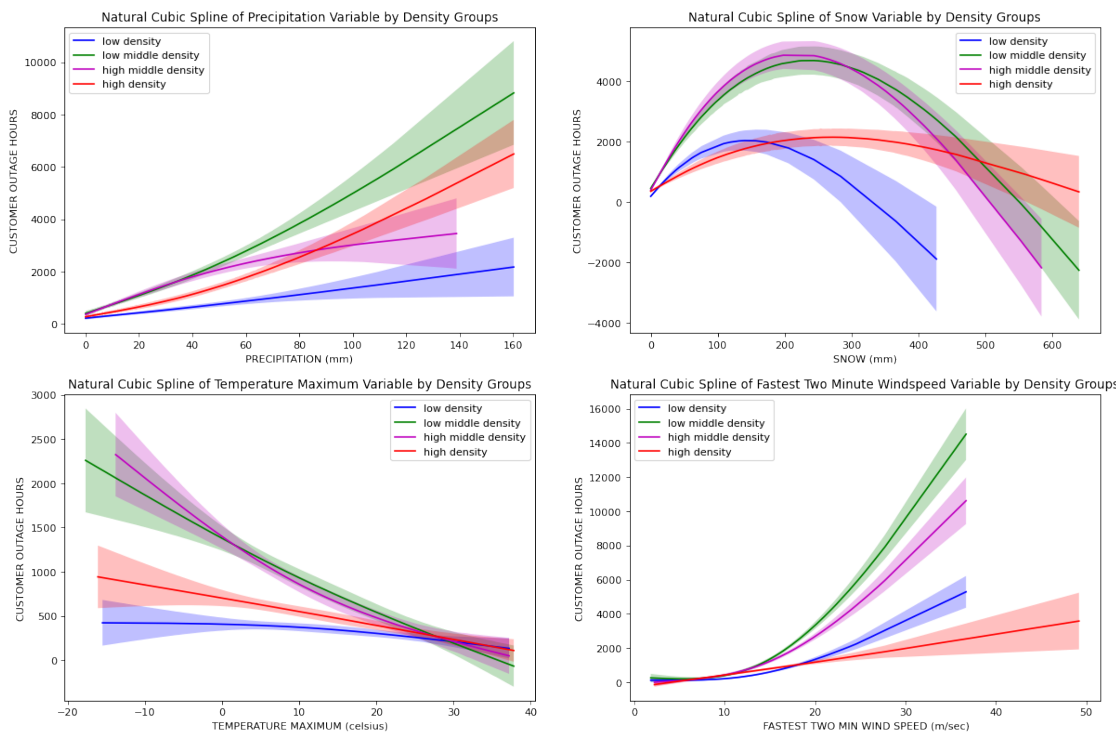
```
y_snow_l = np.asarray(df_snow_l['CUSTOMER_OUTAGE_HOURS'])
y_snow_lm = np.asarray(df_snow_lm['CUSTOMER_OUTAGE_HOURS'])
y_snow_um = np.asarray(df_snow_um['CUSTOMER_OUTAGE_HOURS'])
y_snow_u = np.asarray(df_snow_u['CUSTOMER_OUTAGE_HOURS'])

model_snow_l = sm.OLS(y_snow_l, X_snow_l).fit(disp=0)
model_snow_lm = sm.OLS(y_snow_lm, X_snow_lm).fit(disp=0)
model_snow_um = sm.OLS(y_snow_um, X_snow_um).fit(disp=0)
model_snow_u = sm.OLS(y_snow_u, X_snow_u).fit(disp=0)

y_pred_snow_l = model_snow_l.predict(X_snow_l)
y_pred_snow_lm = model_snow_lm.predict(X_snow_lm)
y_pred_snow_um = model_snow_um.predict(X_snow_um)
y_pred_snow_u = model_snow_u.predict(X_snow_u)

#### visualizing CI's
ci_snow_l = confidence_interval(X_snow_l, y_snow_l, y_pred_snow_l)
ci_snow_lm = confidence_interval(X_snow_lm, y_snow_lm, y_pred_snow_lm)
ci_snow_um = confidence_interval(X_snow_um, y_snow_um, y_pred_snow_um)
ci_snow_u = confidence_interval(X_snow_u, y_snow_u, y_pred_snow_u)

###
plt.subplot(2, 2, 2) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Snow Variable by Density Groups')
plt.plot(df_snow_l['SNOW'], y_pred_snow_l, 'b')
plt.plot(df_snow_lm['SNOW'], y_pred_snow_lm, 'g')
plt.plot(df_snow_um['SNOW'], y_pred_snow_um, 'm')
plt.plot(df_snow_u['SNOW'], y_pred_snow_u, 'r')

### CI's
plt.fill_between(df_snow_l['SNOW'], y_pred_snow_l-ci_snow_l,␣
 ↪y_pred_snow_l+ci_snow_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_snow_lm['SNOW'], y_pred_snow_lm-ci_snow_lm,␣
 ↪y_pred_snow_lm+ci_snow_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_snow_um['SNOW'], y_pred_snow_um-ci_snow_um,␣
 ↪y_pred_snow_um+ci_snow_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_snow_u['SNOW'], y_pred_snow_u-ci_snow_u,␣
 ↪y_pred_snow_u+ci_snow_u, facecolor = 'r', alpha = 0.25)

#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)
plt.legend(labels = ['low density', 'low middle density', 'high middle␣
 ↪density', 'high density'])
plt.xlabel('SNOW (mm)')
plt.ylabel('CUSTOMER OUTAGE HOURS')
```

```python
### T MAX ###
df_tmax_l = density_l[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_lm = density_lm[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_um = density_um[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]
df_tmax_u = density_u[['CUSTOMER_OUTAGE_HOURS', 'TEMP_MAX']]

#### creating confidence intervals
df_tmax_l = df_tmax_l.sample(n=20000, replace=False, random_state=75)
df_tmax_lm = df_tmax_lm.sample(n=20000, replace=False, random_state=75)
df_tmax_um = df_tmax_um.sample(n=20000, replace=False, random_state=75)
df_tmax_u = df_tmax_u.sample(n=20000, replace=False, random_state=75)

df_tmax_l = df_tmax_l.sort_values(by=['TEMP_MAX'])
df_tmax_lm = df_tmax_lm.sort_values(by=['TEMP_MAX'])
df_tmax_um = df_tmax_um.sort_values(by=['TEMP_MAX'])
df_tmax_u = df_tmax_u.sort_values(by=['TEMP_MAX'])

df_tmax_l = df_tmax_l.reset_index(drop = True)
df_tmax_lm = df_tmax_lm.reset_index(drop = True)
df_tmax_um = df_tmax_um.reset_index(drop = True)
df_tmax_u = df_tmax_u.reset_index(drop = True)

X_tmax_l = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_l)
X_tmax_lm = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_lm)
X_tmax_um = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_um)
X_tmax_u = pt.dmatrix('cr(TEMP_MAX, df=3)', df_tmax_u)

y_tmax_l = np.asarray(df_tmax_l['CUSTOMER_OUTAGE_HOURS'])
y_tmax_lm = np.asarray(df_tmax_lm['CUSTOMER_OUTAGE_HOURS'])
y_tmax_um = np.asarray(df_tmax_um['CUSTOMER_OUTAGE_HOURS'])
y_tmax_u = np.asarray(df_tmax_u['CUSTOMER_OUTAGE_HOURS'])

model_tmax_l = sm.OLS(y_tmax_l, X_tmax_l).fit(disp=0)
model_tmax_lm = sm.OLS(y_tmax_lm, X_tmax_lm).fit(disp=0)
model_tmax_um = sm.OLS(y_tmax_um, X_tmax_um).fit(disp=0)
model_tmax_u = sm.OLS(y_tmax_u, X_tmax_u).fit(disp=0)

y_pred_tmax_l = model_tmax_l.predict(X_tmax_l)
y_pred_tmax_lm = model_tmax_lm.predict(X_tmax_lm)
y_pred_tmax_um = model_tmax_um.predict(X_tmax_um)
y_pred_tmax_u = model_tmax_u.predict(X_tmax_u)


#### visualizing CI's
ci_tmax_l = confidence_interval(X_tmax_l, y_tmax_l, y_pred_tmax_l)
ci_tmax_lm = confidence_interval(X_tmax_lm, y_tmax_lm, y_pred_tmax_lm)
ci_tmax_um = confidence_interval(X_tmax_um, y_tmax_um, y_pred_tmax_um)
```

```
ci_tmax_u = confidence_interval(X_tmax_u, y_tmax_u, y_pred_tmax_u)

###
plt.subplot(2, 2, 3) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Temperature Maximum Variable by Density␣
↪Groups')
plt.plot(df_tmax_l['TEMP_MAX'], y_pred_tmax_l, 'b')
plt.plot(df_tmax_lm['TEMP_MAX'], y_pred_tmax_lm, 'g')
plt.plot(df_tmax_um['TEMP_MAX'], y_pred_tmax_um, 'm')
plt.plot(df_tmax_u['TEMP_MAX'], y_pred_tmax_u, 'r')

### CI's
plt.fill_between(df_tmax_l['TEMP_MAX'], y_pred_tmax_l-ci_tmax_l,␣
↪y_pred_tmax_l+ci_tmax_l, facecolor = 'b', alpha = 0.25)
plt.fill_between(df_tmax_lm['TEMP_MAX'], y_pred_tmax_lm-ci_tmax_lm,␣
↪y_pred_tmax_lm+ci_tmax_lm, facecolor = 'g', alpha = 0.25)
plt.fill_between(df_tmax_um['TEMP_MAX'], y_pred_tmax_um-ci_tmax_um,␣
↪y_pred_tmax_um+ci_tmax_um, facecolor = 'm', alpha = 0.25)
plt.fill_between(df_tmax_u['TEMP_MAX'], y_pred_tmax_u-ci_tmax_u,␣
↪y_pred_tmax_u+ci_tmax_u, facecolor = 'r', alpha = 0.25)

#plt.scatter(df_tmax.TEMP_MAX, df_tmax.CUSTOMER_OUTAGE_HOURS)
plt.legend(labels = ['low density', 'low middle density', 'high middle␣
↪density', 'high density'])
plt.xlabel('TEMPERATURE MAXIMUM (celsius)')
plt.ylabel('CUSTOMER OUTAGE HOURS')

### 2 MIN WIND SPEED ##########################

df_2wnd_l = density_l[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_lm = density_lm[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_um = density_um[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]
df_2wnd_u = density_u[['CUSTOMER_OUTAGE_HOURS', 'FASTEST_TWO_MIN_WIND_SPEED']]

#### creating confidence intervals
df_2wnd_l = df_2wnd_l.sample(n=20000, replace=False, random_state=75)
df_2wnd_lm = df_2wnd_lm.sample(n=20000, replace=False, random_state=75)
df_2wnd_um = df_2wnd_um.sample(n=20000, replace=False, random_state=75)
df_2wnd_u = df_2wnd_u.sample(n=20000, replace=False, random_state=75)

df_2wnd_l = df_2wnd_l.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_lm = df_2wnd_lm.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_um = df_2wnd_um.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
df_2wnd_u = df_2wnd_u.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])

df_2wnd_l = df_2wnd_l.reset_index(drop = True)
```

```
df_2wnd_lm = df_2wnd_lm.reset_index(drop = True)
df_2wnd_um = df_2wnd_um.reset_index(drop = True)
df_2wnd_u = df_2wnd_u.reset_index(drop = True)

X_2wnd_l = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_l)
X_2wnd_lm = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_lm)
X_2wnd_um = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_um)
X_2wnd_u = pt.dmatrix('cr(FASTEST_TWO_MIN_WIND_SPEED, df=3)', df_2wnd_u)

y_2wnd_l = np.asarray(df_2wnd_l['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_lm = np.asarray(df_2wnd_lm['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_um = np.asarray(df_2wnd_um['CUSTOMER_OUTAGE_HOURS'])
y_2wnd_u = np.asarray(df_2wnd_u['CUSTOMER_OUTAGE_HOURS'])

model_2wnd_l = sm.OLS(y_2wnd_l, X_2wnd_l).fit(disp=0)
model_2wnd_lm = sm.OLS(y_2wnd_lm, X_2wnd_lm).fit(disp=0)
model_2wnd_um = sm.OLS(y_2wnd_um, X_2wnd_um).fit(disp=0)
model_2wnd_u = sm.OLS(y_2wnd_u, X_2wnd_u).fit(disp=0)

y_pred_2wnd_l = model_2wnd_l.predict(X_2wnd_l)
y_pred_2wnd_lm = model_2wnd_lm.predict(X_2wnd_lm)
y_pred_2wnd_um = model_2wnd_um.predict(X_2wnd_um)
y_pred_2wnd_u = model_2wnd_u.predict(X_2wnd_u)

#### visualizing CI's
ci_2wnd_l = confidence_interval(X_2wnd_l, y_2wnd_l, y_pred_2wnd_l)
ci_2wnd_lm = confidence_interval(X_2wnd_lm, y_2wnd_lm, y_pred_2wnd_lm)
ci_2wnd_um = confidence_interval(X_2wnd_um, y_2wnd_um, y_pred_2wnd_um)
ci_2wnd_u = confidence_interval(X_2wnd_u, y_2wnd_u, y_pred_2wnd_u)

###
plt.subplot(2, 2, 4) # row 1, col 2 index 1
plt.title('Natural Cubic Spline of Fastest Two Minute Windspeed Variable by␣
 ↪Density Groups')
plt.plot(df_2wnd_l['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_l, 'b')
plt.plot(df_2wnd_lm['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_lm, 'g')
plt.plot(df_2wnd_um['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_um, 'm')
plt.plot(df_2wnd_u['FASTEST_TWO_MIN_WIND_SPEED'], y_pred_2wnd_u, 'r')

### CI's
plt.fill_between(df_2wnd_l['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_l-ci_2wnd_l, y_pred_2wnd_l+ci_2wnd_l, facecolor = 'b', alpha = 0.
 ↪25)
plt.fill_between(df_2wnd_lm['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_lm-ci_2wnd_lm, y_pred_2wnd_lm+ci_2wnd_lm, facecolor = 'g', alpha␣
 ↪= 0.25)
```

```
plt.fill_between(df_2wnd_um['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_um-ci_2wnd_um, y_pred_2wnd_um+ci_2wnd_um, facecolor = 'm', alpha␣
 ↪= 0.25)
plt.fill_between(df_2wnd_u['FASTEST_TWO_MIN_WIND_SPEED'],␣
 ↪y_pred_2wnd_u-ci_2wnd_u, y_pred_2wnd_u+ci_2wnd_u, facecolor = 'r', alpha = 0.
 ↪25)

#plt.scatter(df_snow.SNOW, df_snow.CUSTOMER_OUTAGE_HOURS)
plt.xlabel(' FASTEST TWO MIN WIND SPEED (m/sec)')
plt.ylabel('CUSTOMER OUTAGE HOURS')
plt.legend(labels = ['low density', 'low middle density', 'high middle␣
 ↪density', 'high density'])


plt.show()
```



[21]:
```
MSE_precipitation_l = mean_squared_error(y_precipitation_l,␣
 ↪y_pred_precipitation_l)
MSE_precipitation_lm = mean_squared_error(y_precipitation_lm,␣
 ↪y_pred_precipitation_lm)
MSE_precipitation_um = mean_squared_error(y_precipitation_um,␣
 ↪y_pred_precipitation_um)
```

```
MSE_precipitation_u = mean_squared_error(y_precipitation_u,␣
↪y_pred_precipitation_u)

print('LOWER DENSITY MSE PRECIPITATION is: ', "{:.2f}".
↪format(MSE_precipitation_l))
print('LOWER MIDDLE DENSITY MSE PRECIPITATION is: ', "{:.2f}".
↪format(MSE_precipitation_lm))
print('UPPER MIDDLE DENSITY MSE PRECIPITATION is: ', "{:.2f}".
↪format(MSE_precipitation_um))
print('UPPER DENSITY MSE PRECIPITATION is: ', "{:.2f}".
↪format(MSE_precipitation_u))

print()

MSE_snow_l = mean_squared_error(y_snow_l, y_pred_snow_l)
MSE_snow_lm = mean_squared_error(y_snow_lm, y_pred_snow_lm)
MSE_snow_um = mean_squared_error(y_snow_um, y_pred_snow_um)
MSE_snow_u = mean_squared_error(y_snow_u, y_pred_snow_u)

print('LOWER DENSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_l))
print('LOWER MIDDLE DENSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_lm))
print('UPPER MIDDLE DENSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_um))
print('UPPER DENSITY MSE SNOW is: ', "{:.2f}".format(MSE_snow_u))

print()

MSE_tmax_l = mean_squared_error(y_tmax_l, y_pred_tmax_l)
MSE_tmax_lm = mean_squared_error(y_tmax_lm, y_pred_tmax_lm)
MSE_tmax_um = mean_squared_error(y_tmax_um, y_pred_tmax_um)
MSE_tmax_u = mean_squared_error(y_tmax_u, y_pred_tmax_u)

print('LOWER DENSITY MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_l))
print('LOWER MIDDLE DENSITY MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_lm))
print('UPPER MIDDLE DENSITY MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_um))
print('UPPER MSE TEMP_MAX is: ', "{:.2f}".format(MSE_tmax_u))

print()

MSE_2wnd_l = mean_squared_error(y_2wnd_l, y_pred_2wnd_l)
MSE_2wnd_lm = mean_squared_error(y_2wnd_lm, y_pred_2wnd_lm)
MSE_2wnd_um = mean_squared_error(y_2wnd_um, y_pred_2wnd_um)
MSE_2wnd_u = mean_squared_error(y_2wnd_u, y_pred_2wnd_u)

print('LOWER MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".format(MSE_2wnd_l))
print('LOWER MIDDLE MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
↪format(MSE_2wnd_lm))
```

```
print('UPPER MIDDLE MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".
 ↪format(MSE_2wnd_um))
print('UPPER MSE FASTEST_TWO_MIN_WIND_SPEED is: ', "{:.2f}".format(MSE_2wnd_u))
```

```
LOWER DENSITY MSE PRECIPITATION is:  5043162.21
LOWER MIDDLE DENSITY MSE PRECIPITATION is:  19283525.06
UPPER MIDDLE DENSITY MSE PRECIPITATION is:  18257062.31
UPPER DENSITY MSE PRECIPITATION is:  7958820.29

LOWER DENSITY MSE SNOW is:  5019858.99
LOWER MIDDLE DENSITY MSE SNOW is:  19361368.64
UPPER MIDDLE DENSITY MSE SNOW is:  18211749.66
UPPER DENSITY MSE SNOW is:  8049556.84

LOWER DENSITY MSE TEMP_MAX is:  5063108.86
LOWER MIDDLE DENSITY MSE TEMP_MAX is:  19502177.13
UPPER MIDDLE DENSITY MSE TEMP_MAX is:  18369236.52
UPPER MSE TEMP_MAX is:  8075670.84

LOWER MSE FASTEST_TWO_MIN_WIND_SPEED is:  4988974.39
LOWER MIDDLE MSE FASTEST_TWO_MIN_WIND_SPEED is:  18940623.87
UPPER MIDDLE MSE FASTEST_TWO_MIN_WIND_SPEED is:  17964986.41
UPPER MSE FASTEST_TWO_MIN_WIND_SPEED is:  8003044.90
```

9 GAMs

[22]:
```
# Read in data and create dataframe with relevant predictor and response␣
 ↪variables
data = pd.read_csv('Electricity_Reliability_Dataset.csv')
data_vars = data[['PRECIPITATION', 'SNOW', 'TEMP_MAX',␣
 ↪'FASTEST_TWO_MIN_WIND_SPEED', 'CUSTOMER_OUTAGE_HOURS']]
```

[23]:
```
# Function to find the degrees of freedom for the optimal basic spline using␣
 ↪K-fold cross validation
def spline_optimal(df, k, degree, degree_of_freedom_max, predictor_name,␣
 ↪response_name):
    mses = pd.DataFrame()
    fold = 0
    # Run K-fold cross validation
    kf = KFold(n_splits=k, shuffle=True, random_state=0)
    for train_index, val_index in kf.split(df):
        df_train = np.asarray(df)[train_index]
        df_train = pd.DataFrame(df_train, columns = df.columns)
        df_val = np.asarray(df)[val_index]
        df_val = pd.DataFrame(df_val, columns = df.columns)
        # Cross validate on degrees of freedom / number of knots
        MSE_array = []
```

```
        for deg_of_freedom in range(degree + 1, degree_of_freedom_max + 1):
            string = 'bs('+predictor_name+', df='+str(deg_of_freedom)+',␣
↪degree='+str(degree)+', include_intercept=True)'
            X_train = pt.dmatrix(string, df_train)
            X_val = pt.dmatrix(string, df_val)
            model = linear_model.LinearRegression().fit(X_train, np.
↪asarray(df_train[response_name]))
            y_pred = model.predict(X_val)
            MSE = mean_squared_error(np.asarray(df_val[response_name]), y_pred)
            MSE_array.append(MSE)
        mses[fold] = MSE_array
        fold = fold+1
    dof = np.arange(degree+1, degree_of_freedom_max+1)
    # Find the average and standard deviation of the MSEs
    mses['mses_ave'] = mses.mean(axis=1)
    mses['mses_std'] = mses.std(axis=1)
    # Determine the minimum average MSE and the polynomial order where it occurs
    MSE_ave_min = min(mses.mses_ave)
    degree_of_freedom_MSE_ave_min = mses['mses_ave'].idxmin()+degree+1
    # Find one standard error limit
    one_std_err_limit = MSE_ave_min + mses['mses_std'][mses['mses_ave'].
↪idxmin()]
    MSE_ave_min_one_std_err = 0
    for item in mses['mses_ave']:
        if item <= one_std_err_limit:
            MSE_ave_min_one_std_err = item
            break
    degree_of_freedom_one_std_err = mses.index[mses['mses_ave'] ==␣
↪MSE_ave_min_one_std_err]+degree+1
    return MSE_ave_min, degree_of_freedom_MSE_ave_min,␣
↪degree_of_freedom_one_std_err[0]
```

[24]:

```
# Function to find the degrees of freedom for the optimal natural cubic spline␣
↪using K-fold cross validation
def spline_optimal_cubic(df, k, degree, degree_of_freedom_max, predictor_name,␣
↪response_name):
    mses = pd.DataFrame()
    fold = 0
    kf = KFold(n_splits=k, shuffle=True, random_state=0)
    for train_index, val_index in kf.split(df):
        df_train = np.asarray(df)[train_index]
        df_train = pd.DataFrame(df_train, columns = df.columns)
        df_val = np.asarray(df)[val_index]
        df_val = pd.DataFrame(df_val, columns = df.columns)
        MSE_array = []
        for deg_of_freedom in range(degree + 1, degree_of_freedom_max + 1):
            string = 'cr('+predictor_name+', df='+str(deg_of_freedom)+')'
```

```
            X_train = pt.dmatrix(string, df_train)
            X_val = pt.dmatrix(string, df_val)
            model = linear_model.LinearRegression().fit(X_train, np.
 ↪asarray(df_train[response_name]))
            y_pred = model.predict(X_val)
            MSE = mean_squared_error(np.asarray(df_val[response_name]), y_pred)
            MSE_array.append(MSE)
        mses[fold] = MSE_array
        fold = fold+1
    dof = np.arange(degree+1, degree_of_freedom_max+1)
    mses['mses_ave'] = mses.mean(axis=1)
    mses['mses_std'] = mses.std(axis=1)
    MSE_ave_min = min(mses.mses_ave)
    degree_of_freedom_MSE_ave_min = mses['mses_ave'].idxmin()+degree+1
    one_std_err_limit = MSE_ave_min + mses['mses_std'][mses['mses_ave'].
 ↪idxmin()]
    MSE_ave_min_one_std_err = 0
    for item in mses['mses_ave']:
        if item <= one_std_err_limit:
            MSE_ave_min_one_std_err = item
            break
    degree_of_freedom_one_std_err = mses.index[mses['mses_ave'] ==␣
 ↪MSE_ave_min_one_std_err]+degree+1
    return MSE_ave_min, degree_of_freedom_MSE_ave_min,␣
 ↪degree_of_freedom_one_std_err[0]
```

[25]:

```
# Create an array with the optimal degrees of freedom for each variable using␣
 ↪natural cubic splines
k = 5
degree = 3
degree_of_freedom_max = 10
response_name = 'CUSTOMER_OUTAGE_HOURS'

predictors = data_vars.drop(['CUSTOMER_OUTAGE_HOURS'], axis=1).columns

optimal_dof_cubic = []
for predictor in predictors:
    predictor_name = predictor
    df = data_vars[['CUSTOMER_OUTAGE_HOURS', predictor_name]]
    MSE_ave_min, degree_of_freedom_MSE_ave_min, dof_ose =␣
 ↪spline_optimal_cubic(df,
                                                                                ␣
 ↪k,degree,
                                                                                ␣
 ↪degree_of_freedom_max,
```

```
                                                                                    ␣
↪predictor_name,
                                                                                    ␣
↪response_name)
    optimal_dof_cubic.append(dof_ose)
```

[26]:
```
# Create dataframe with optimal degrees of freedom for each natural cubic spline
degrees_of_freedom = pd.DataFrame(data = [optimal_dof_cubic], columns =␣
↪list(predictors))
```

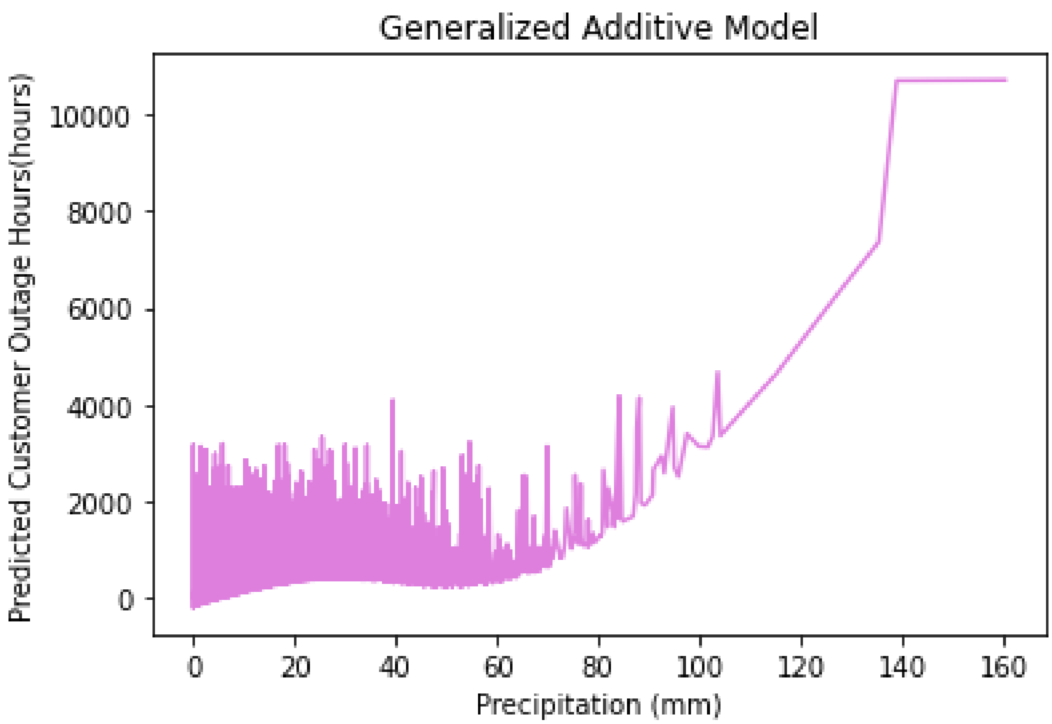
[27]:
```
# Create function to build GAM using natural cubic splines
def GAMs_with_CI(df, response_variable, k, degrees_of_freedom):
    l = list(df.columns)
    l.remove(response_variable)
    string = ''
    for feature in l:
        string = string + 'cr(' + feature + ', df='+␣
↪str(int(degrees_of_freedom[feature])) + ') +'
    string = string[:-1]
    mses = pd.DataFrame()
    MSE_array = []
    fold = 0
    kf = KFold(n_splits=k, shuffle=True)
    for train_index, val_index in kf.split(df):
        df_train = np.asarray(df)[train_index]
        df_train = pd.DataFrame(df_train, columns = df.columns)
        df_val = np.asarray(df)[val_index]
        df_val = pd.DataFrame(df_val, columns = df.columns)
        X_train = pt.dmatrix(string, df_train)
        X_val = pt.dmatrix(string, df_val)
        model = sm.OLS(df_train[response_variable], X_train).fit(disp=0)
        y_pred = model.predict(X_val)
        MSE = mean_squared_error(df_val[response_variable], y_pred)
        MSE_array.append(MSE)
    MSE_ave = np.mean(MSE_array)
    X = pt.dmatrix(string, df)
    y = np.asarray(df[response_variable])
    model = sm.OLS(y, X).fit(disp=0)
    y_pred = model.predict(X)
    return MSE_ave, y_pred
```

[28]:
```
# GAM created
response_variable = 'CUSTOMER_OUTAGE_HOURS'
k = 5

[MSE_ave, y_pred]= GAMs_with_CI(data_vars, response_variable, k,␣
↪degrees_of_freedom)
```

[29]:
```
# Define y variable
y = np.array(data['CUSTOMER_OUTAGE_HOURS'])
```

[30]:
```
# Plot of GAM 1
response_variable = 'CUSTOMER_OUTAGE_HOURS'
k = 5

data_vars = data_vars.sort_values(by=['PRECIPITATION'])
data_vars = data_vars.reset_index(drop = True)

[MSE_ave, y_pred]= GAMs_with_CI(data_vars, response_variable, k,␣
↪degrees_of_freedom)

plt.plot(data_vars['PRECIPITATION'], y_pred, 'm', alpha = 0.5)
plt.title('Generalized Additive Model');
plt.xlabel('Precipitation (mm)')
plt.ylabel('Predicted Customer Outage Hours(hours)')
#plt.savefig('GAMs_prec.png')
```

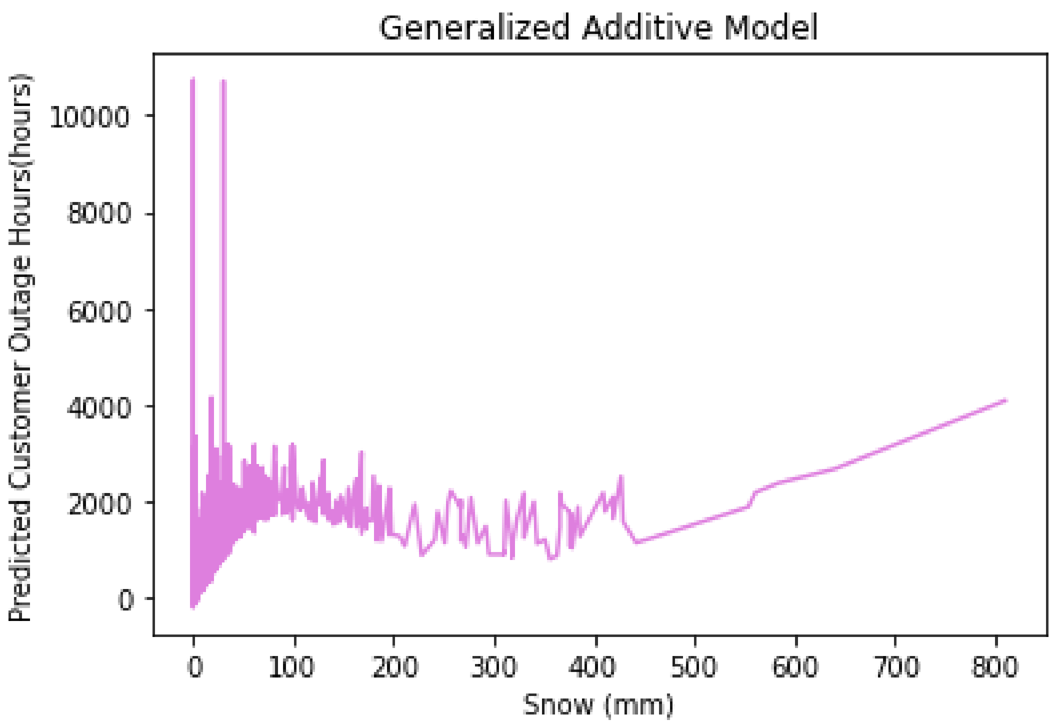
[30]: Text(0, 0.5, 'Predicted Customer Outage Hours(hours)')



[31]:
```
# Plot of GAM 2
response_variable = 'CUSTOMER_OUTAGE_HOURS'
```

```
k = 5

data_vars = data_vars.sort_values(by=['SNOW'])
data_vars = data_vars.reset_index(drop = True)

[MSE_ave, y_pred]= GAMs_with_CI(data_vars, response_variable, k,␣
 ↪degrees_of_freedom)

plt.plot(data_vars['SNOW'], y_pred, 'm', alpha = 0.5)
plt.title('Generalized Additive Model');
plt.xlabel('Snow (mm)')
plt.ylabel('Predicted Customer Outage Hours(hours)')
#plt.savefig('GAMs_snow.png')
```

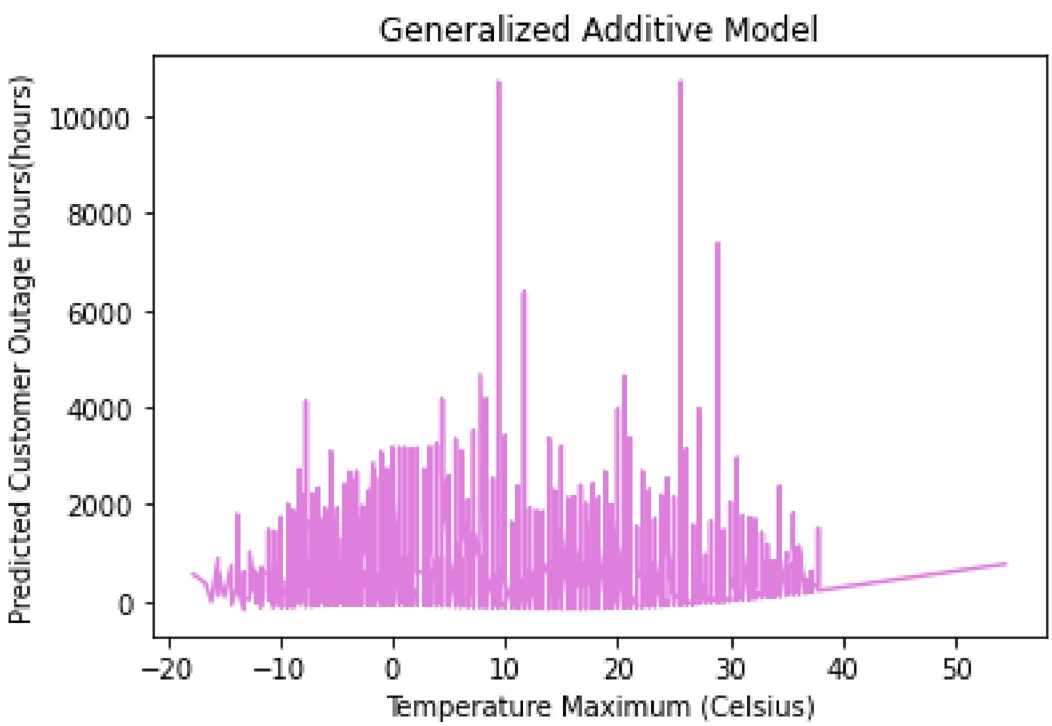
[31]: Text(0, 0.5, 'Predicted Customer Outage Hours(hours)')



[32]:
```
# Plot of GAM 3
response_variable = 'CUSTOMER_OUTAGE_HOURS'
k = 5

data_vars = data_vars.sort_values(by=['TEMP_MAX'])
data_vars = data_vars.reset_index(drop = True)
```

```
[MSE_ave, y_pred]= GAMs_with_CI(data_vars, response_variable, k,␣
 ↪degrees_of_freedom)

plt.plot(data_vars['TEMP_MAX'], y_pred, 'm', alpha = 0.5)
plt.title('Generalized Additive Model');
plt.xlabel('Temperature Maximum (Celsius)')
plt.ylabel('Predicted Customer Outage Hours(hours)')
#plt.savefig('GAMs_tmax.png')
```

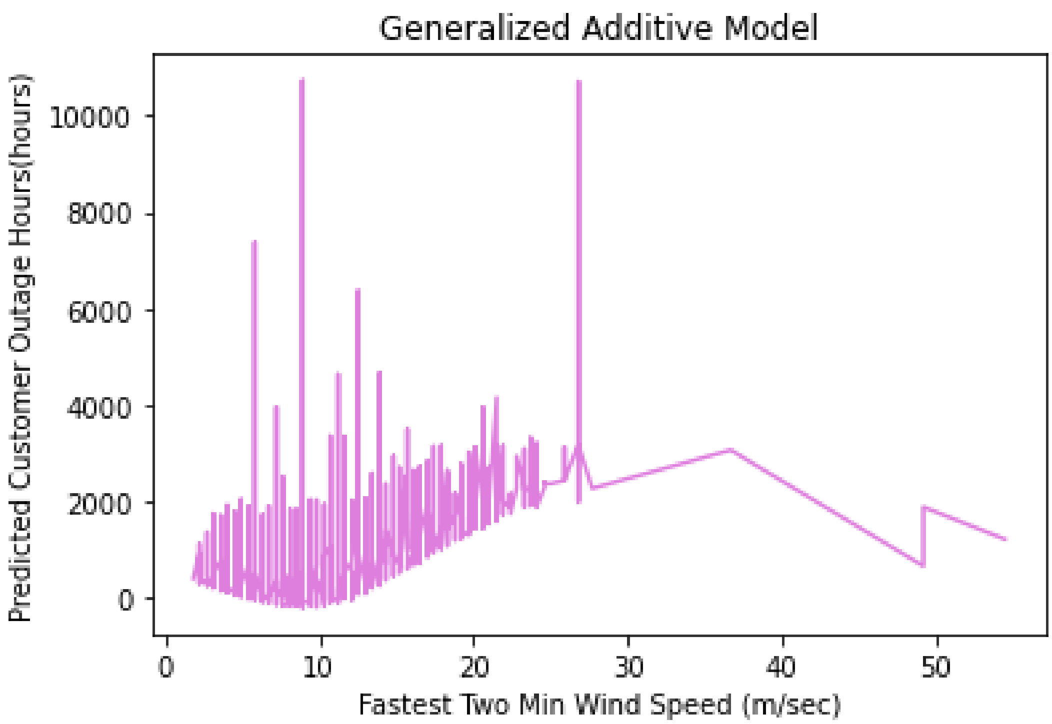
[32]: Text(0, 0.5, 'Predicted Customer Outage Hours(hours)')



[33]:
```
# Plot of GAM 4
response_variable = 'CUSTOMER_OUTAGE_HOURS'
k = 5

data_vars = data_vars.sort_values(by=['FASTEST_TWO_MIN_WIND_SPEED'])
data_vars = data_vars.reset_index(drop = True)

[MSE_ave, y_pred]= GAMs_with_CI(data_vars, response_variable, k,␣
 ↪degrees_of_freedom)

plt.plot(data_vars['FASTEST_TWO_MIN_WIND_SPEED'], y_pred, 'm', alpha = 0.5)
plt.title('Generalized Additive Model');
```

```
plt.xlabel('Fastest Two Min Wind Speed (m/sec)')
plt.ylabel('Predicted Customer Outage Hours(hours)')
#plt.savefig('GAMs_2wnd.png')
```

[33]: Text(0, 0.5, 'Predicted Customer Outage Hours(hours)')

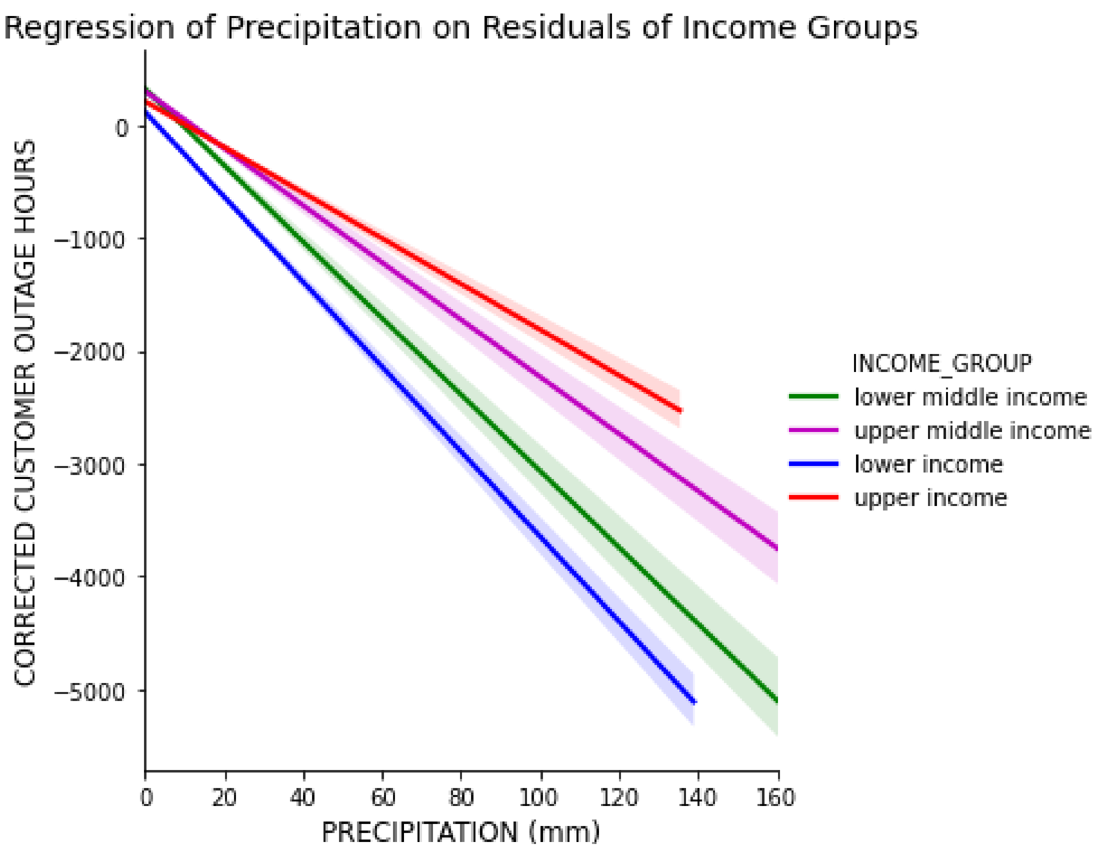


[34]:
```
# Create residuals and missing items from data_vars dataframe
data_vars['RESIDUALS'] = y-y_pred
data_vars['INCOME_GROUP'] = data['INCOME_GROUP']
data_vars['DENSITY_GROUP'] = data['DENSITY_GROUP']
data_vars['DIVERSITY_GROUP'] = data['DIVERSITY_GROUP']
```

[35]:
```
# CI function
def confidence_interval(X, y, y_pred):
    mse = np.sum(np.square(y_pred - y)) / y.size
    cov = mse * np.linalg.inv(X.T @ X)
    var_f = np.diagonal((X @ cov) @ X.T)
    se = np.sqrt(abs(var_f))
    conf_int = 2*se
    return conf_int
```

[36]:
```
# Residuals 1
```

```
sns.lmplot(x="PRECIPITATION", y="RESIDUALS", hue="INCOME_GROUP",␣
 ↪data=data_vars,scatter=False, ci=95,palette=['g', 'm','b','r'])
plt.title('Regression of Precipitation on Residuals of Income␣
 ↪Groups',fontsize=14);
plt.xlabel('PRECIPITATION (mm)',fontsize = 12)
plt.ylabel('CORRECTED CUSTOMER OUTAGE HOURS',fontsize = 12)
plt.savefig('Income_GAMS_residuals.png',bbox_inches="tight")
plt.show()
```

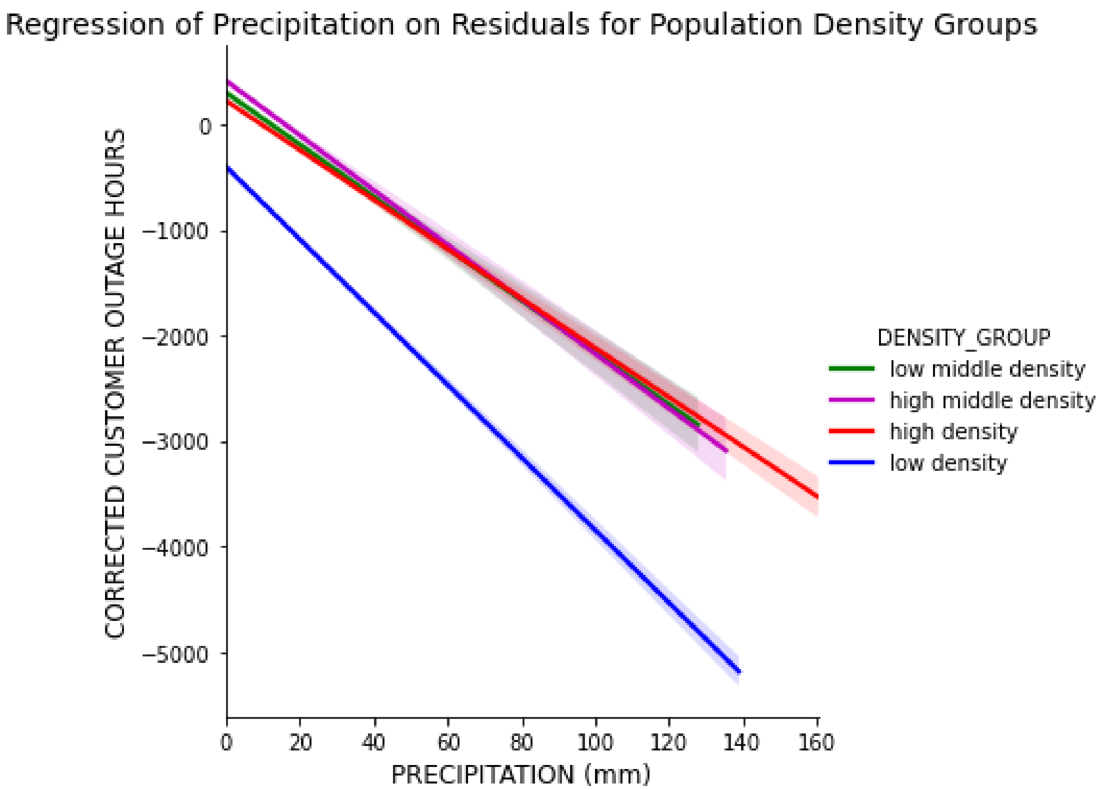


[37]:
```
# apply linear regression using numpy
def linReg(x, y):
    A = np.vstack([x, np.ones(len(x))]).T
    slope, intercept = np.linalg.lstsq(A, y, rcond=None)[0]
    return pd.Series({'slope':slope})
res = data_vars.groupby('INCOME_GROUP').apply(lambda x: linReg(x.PRECIPITATION,␣
 ↪x.RESIDUALS))
print(res)
```

```
                              slope
INCOME_GROUP
```

```
lower income         -37.637471
lower middle income -33.876327
upper income         -20.232288
upper middle income -25.323095
```

[38]:
```python
# Residuals 2
sns.lmplot(x="PRECIPITATION", y="RESIDUALS", hue="DENSITY_GROUP",␣
 ↪data=data_vars,scatter=False, ci=95,palette=['g', 'm','r','b']);
plt.title('Regression of Precipitation on Residuals for Population Density␣
 ↪Groups',fontsize=14);
plt.xlabel('PRECIPITATION (mm)',fontsize = 12)
plt.ylabel('CORRECTED CUSTOMER OUTAGE HOURS',fontsize = 12)
plt.savefig('population_density_GAMS_residuals.png',bbox_inches="tight")
plt.show()
```

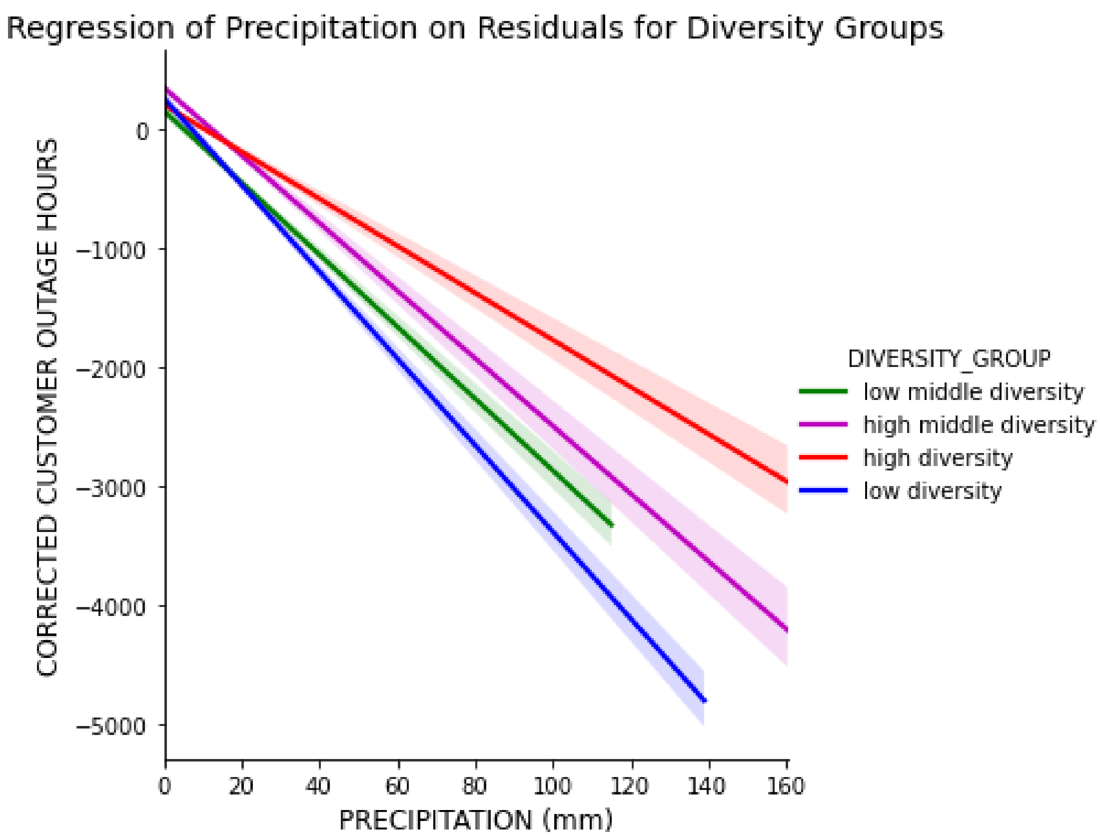


[39]:
```python
# apply linear regression using numpy
def linReg(x, y):
    A = np.vstack([x, np.ones(len(x))]).T
    slope, intercept = np.linalg.lstsq(A, y, rcond=None)[0]
    return pd.Series({'slope':slope})
res = data_vars.groupby('DENSITY_GROUP').apply(lambda x: linReg(x.
 ↪PRECIPITATION, x.RESIDUALS))
```

```
print(res)
```

```
                           slope
DENSITY_GROUP
high density         -23.426289
high middle density  -25.890853
low density          -34.453866
low middle density   -24.635492
```

[40]:
```
# Residuals 3
sns.lmplot(x="PRECIPITATION", y="RESIDUALS", hue="DIVERSITY_GROUP",␣
 ↪data=data_vars,scatter=False, ci=95,palette=['g', 'm','r','b']);
plt.title('Regression of Precipitation on Residuals for Diversity␣
 ↪Groups',fontsize = 14);
plt.xlabel('PRECIPITATION (mm)',fontsize = 12)
plt.ylabel('CORRECTED CUSTOMER OUTAGE HOURS',fontsize = 12)
plt.savefig('diveristy_GAMS_residuals.png',bbox_inches="tight")
plt.show()
```



[41]:

```
# apply linear regresion using numpy
def linReg(x, y):
    A = np.vstack([x, np.ones(len(x))]).T
    slope, intercept = np.linalg.lstsq(A, y, rcond=None)[0]
    return pd.Series({'slope':slope})
res = data_vars.groupby('DIVERSITY_GROUP').apply(lambda x: linReg(x.
 ↪PRECIPITATION, x.RESIDUALS))
print(res)
```

```
                           slope
DIVERSITY_GROUP
high diversity        -19.795598
high middle diversity -28.451736
low diversity         -36.474908
low middle diversity  -30.272812
```