In [1]:
```
import itertools
import matplotlib.pyplot as plt
import numpy
import pandas
import sklearn.neural_network as nn
import time
import sys
import gc
from scipy.stats import chi2
import Regression
import sklearn.metrics as metrics
import warnings
warnings.filterwarnings('ignore')
import itertools
import sklearn.neural_network as nn
```

Question 1

In [2]:
```
# Set some options for printing all the columns
numpy.set_printoptions(precision = 10, threshold = sys.maxsize)
numpy.set_printoptions(linewidth = numpy.inf)

pandas.set_option('display.max_columns', None)
pandas.set_option('display.expand_frame_repr', False)
pandas.set_option('max_colwidth', None)
#pandas.set_option('precision', 10)

pandas.options.display.float_format = '{:,.7}'.format
```

In [3]:
```
trainData = pandas.read_excel("Homeowner_Claim_History.xlsx")
```

(a) (10 points) Generate horizontal boxplots of Total Claim Amount in a Year grouped by each of the seven categorical predictors f_primary_age_tier, f_primary_gender, f_marital, f_residence_location, f_fire_alarm_type, f_mile_fire_station, and f_aoi_tier.

In [4]:

```python
catName = ["f_primary_age_tier","f_primary_gender","f_marital",
           "f_residence_location","f_fire_alarm_type",
           "f_mile_fire_station","f_aoi_tier"]
nPredictor = len(catName)
# Explore the categorical predictors using grouped boxplot

for X_name in catName:
    print(trainData[[X_name]].value_counts())
    fig, ax1 = plt.subplots(nrows = 1, ncols = 1, dpi = 100)
    trainData.boxplot(column = "amt_claims", by = X_name, ax = ax1, vert = False)
    ax1.set_xlabel('Total Claim Amount in a Year')
    ax1.xaxis.grid(True)
    ax1.invert_yaxis()
    plt.suptitle('')
    plt.title(X_name)
    plt.show()
```

```
f_primary_age_tier
28 - 37               9600
38 - 60               6951
21 - 27               5435
< 21                  2784
> 60                  2743
dtype: int64
```

(b) (10 points) For analyses, Severity will follow a Gamma distribution. Train a Gamma model with the logarithm link function. The target variable is Severity (use only positive and non-missing values for analyses). The predictors are the seven categorical predictors. The

model will include the Intercept term. Enter predictors into the model using the Forward Selection method. The entry threshold is 0.05. What is the estimate for the Shape parameter?

In [5]:

```
trainData["severity"] = trainData["amt_claims"] / trainData["num_claims"]
yName = "severity"
trainData = trainData[trainData["severity"]>0]
```

In [6]:

```
# Reorder the categories of the categorical variables in ascending frequency
for pred in catName:
    u = trainData[pred].astype('category').copy()
    u_freq = u.value_counts(ascending = True)
    trainData[pred] = u.cat.reorder_categories(list(u_freq.index)).copy()

# Generate a column of Intercept
X0_train = trainData[[yName]].copy()
X0_train.insert(0, 'Intercept', 1.0)
X0_train.drop(columns = [yName], inplace = True)

y_train = trainData[yName].copy()
```

In [7]:

```
maxIter = 20
tolS = 1e-7
stepSummary = pandas.DataFrame()

# Intercept only model
resultList = Regression.GammaModel (X0_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)

llk0 = resultList[3]
df0 = len(resultList[4])
stepSummary = stepSummary.append([['Intercept', ' ', df0, llk0, numpy.NaN, numpy.NaN, numpy.NaN]], ignore_index = Tru
stepSummary.columns = ['Predictor', 'Type', 'ModelDF', 'ModelLLK', 'DevChiSq', 'DevDF', 'DevSig']

print('======= Step Detail =======')
print('Step = ', 0)
print('Step Statistics:')
print(stepSummary)
```

```
======= Step Detail =======
Step =  0
Step Statistics:
   Predictor Type  ModelDF   ModelLLK  DevChiSq  DevDF  DevSig
0  Intercept             1 -99,940.91       NaN    NaN     NaN
```

(c) (10 points) Provide the Step Summary table. The table should contain (1) Step Number, (2) Model Degrees of Freedom, (3) Model Log-Likelihood, (4) Deviance Chi-Squares, (5) Deviance Degrees of Freedom, and (6) Deviance Significance. Show the Significance in .E7 scientific notation.

In [8]:

```
cName = catName.copy()
entryThreshold = 0.05

for step in range(nPredictor):
    enterName = ''
    stepDetail = pandas.DataFrame()

    # Enter the next predictor
    for X_name in cName:
        X_train = pandas.get_dummies(trainData[[X_name]])
        X_train = X0_train.join(X_train)
        resultList = Regression.GammaModel (X_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)
        llk1 = resultList[3]
        df1 = len(resultList[4])
        devChiSq = 2.0 * (llk1 - llk0)
        devDF = df1 - df0
        devSig = chi2.sf(devChiSq, devDF)
        stepDetail = stepDetail.append([[X_name, 'categorical', df1, llk1, devChiSq, devDF, devSig]], ignore_index =


    stepDetail.columns = ['Predictor', 'Type', 'ModelDF', 'ModelLLK', 'DevChiSq', 'DevDF', 'DevSig']

    # Find a predictor to enter, if any
    stepDetail.sort_values(by = ['DevSig', 'ModelLLK'], axis = 0, ascending = [True, False], inplace = True)
    enterRow = stepDetail.iloc[0].copy()
    minPValue = enterRow['DevSig']
    if (minPValue <= entryThreshold):
        stepSummary = stepSummary.append([enterRow], ignore_index = True)
        df0 = enterRow['ModelDF']
        llk0 = enterRow['ModelLLK']

        enterName = enterRow['Predictor']
        enterType = enterRow['Type']
        if (enterType == 'categorical'):
            X_train = pandas.get_dummies(trainData[[enterName]].astype('category'))
            X0_train = X0_train.join(X_train)
            cName.remove(enterName)

    else:
        break
```

```python
    # Print debugging output
    print('======= Step Detail =======')
    print('Step = ', step+1)
    print('Step Statistics:')
    print(stepDetail)
    print('Enter predictor = ', enterName)
    print('Minimum P-Value =', minPValue)
    print('\n')

# End of forward selection
print('======= Step Summary =======')
print(stepSummary)
```

```
======= Step Detail =======
Step =  1
Step Statistics:
             Predictor         Type  ModelDF    ModelLLK  DevChiSq  DevDF        DevSig
0    f_primary_age_tier  categorical        5 -99,738.72  404.3866      4 3.136693e-86
6            f_aoi_tier  categorical        5 -99,886.54   108.747      4 1.346463e-22
4     f_fire_alarm_type  categorical        3 -99,911.79  58.24039      2  2.25559e-13
5    f_mile_fire_station  categorical        4 -99,911.68  58.46986      3 1.247624e-12
3  f_residence_location  categorical        3 -99,921.65  38.52957      2 4.299427e-09
2             f_marital  categorical        3 -99,938.05  5.720389      2   0.05725763
1      f_primary_gender  categorical        2 -99,940.54 0.7429267      1    0.3887249
Enter predictor =  f_primary_age_tier
Minimum P-Value = 3.1366925636835666e-86


======= Step Detail =======
Step =  2
Step Statistics:
             Predictor         Type  ModelDF    ModelLLK  DevChiSq  DevDF        DevSig
5            f_aoi_tier  categorical        9  -99,679.7  118.0392      4 1.400917e-24
3     f_fire_alarm_type  categorical        7 -99,704.02  69.39581      2 8.528872e-16
4    f_mile_fire_station  categorical        8 -99,708.23  60.97377      3 3.640657e-13
2  f_residence_location  categorical        7 -99,718.89  39.64993      2 2.455426e-09
1             f_marital  categorical        7 -99,736.23  4.973669      2   0.08317283
0      f_primary_gender  categorical        6 -99,738.44 0.5629141      1    0.4530885
Enter predictor =  f_aoi_tier
Minimum P-Value = 1.4009169837558154e-24
```

```
======= Step Detail =======
Step =  3
Step Statistics:
              Predictor         Type  ModelDF   ModelLLK  DevChiSq  DevDF        DevSig
3     f_fire_alarm_type  categorical       11 -99,643.66  72.07999      2 2.228579e-16
4   f_mile_fire_station  categorical       12 -99,648.34  62.72641      3 1.536613e-13
2  f_residence_location  categorical       11 -99,659.33  40.74656      2 1.419048e-09
1             f_marital  categorical       11 -99,677.16  5.079599      2   0.07888221
0      f_primary_gender  categorical       10 -99,679.49 0.4105454      1    0.5216928
Enter predictor =  f_fire_alarm_type
Minimum P-Value = 2.228579143817684e-16


======= Step Detail =======
Step =  4
Step Statistics:
              Predictor         Type  ModelDF   ModelLLK  DevChiSq  DevDF        DevSig
3   f_mile_fire_station  categorical       14 -99,610.62  66.06938      3 2.962017e-14
2  f_residence_location  categorical       13 -99,622.45  42.42847      2  6.12034e-10
1             f_marital  categorical       13 -99,641.17  4.974828      2   0.08312465
0      f_primary_gender  categorical       12 -99,643.46 0.4034187      1     0.525329
Enter predictor =  f_mile_fire_station
Minimum P-Value = 2.962016608253086e-14


======= Step Detail =======
Step =  5
Step Statistics:
              Predictor         Type  ModelDF   ModelLLK  DevChiSq  DevDF        DevSig
2  f_residence_location  categorical       16 -99,585.69  49.87939      2 1.475124e-11
1             f_marital  categorical       16 -99,607.96  5.335108      2   0.06942182
0      f_primary_gender  categorical       15 -99,610.36 0.5343199      1    0.4647963
Enter predictor =  f_residence_location
Minimum P-Value = 1.4751239916365028e-11


======= Step Summary =======
              Predictor         Type  ModelDF   ModelLLK  DevChiSq  DevDF        DevSig
0             Intercept                     1 -99,940.91       NaN    NaN           NaN
1    f_primary_age_tier  categorical        5 -99,738.72  404.3866    4.0 3.136693e-86
2            f_aoi_tier  categorical        9  -99,679.7  118.0392    4.0 1.400917e-24
```

```
3     f_fire_alarm_type   categorical       11 -99,643.66  72.07999    2.0 2.228579e-16
4   f_mile_fire_station   categorical       14 -99,610.62  66.06938    3.0 2.962017e-14
5  f_residence_location   categorical       16 -99,585.69  49.87939    2.0 1.475124e-11
```

(d) (10 points) Assess the final model goodness-of-fit using (1) Root Mean Squared Error, (2) Relative Error, (3) Mean Absolute Proportion Error, and (4) Pearson Correlation. What are the values of these metrics?

In [9]:
```python
# Final model
resultList = Regression.GammaModel (X0_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)
```

In [10]:
```python
y_pred = resultList[6]

# Simple Residual
y_simple_residual = y_train - y_pred

# R-Squared
corr_matrix = numpy.corrcoef(y_train, y_pred)
rsq = corr_matrix[0,1] ** 2

# Pearson Residual
y_pearson_residual = y_simple_residual / numpy.sqrt(y_pred)

# Deviance Residual
r_vec = y_train / y_pred
di_2 = 2* (r_vec - numpy.log(r_vec) - 1)
y_deviance_residual = numpy.where(y_simple_residual > 0, 1.0, -1.0) * numpy.sqrt(di_2)
```

In [11]:
```python
# shape parameter is alpha
alpha = resultList[7]
alpha
```

Out[11]:
```
2.1725703079719096
```

In [12]:

```
# Root Mean Squared Error
mse = numpy.mean(numpy.power(y_simple_residual, 2))
rmse = numpy.sqrt(mse)
rmse
```

Out[12]:

```
942.2956575358173
```

In [13]:

```
# Relative Error
relerr = mse / numpy.var(y_train, ddof = 0)
relerr
```

Out[13]:

```
0.9468732570788471
```

In [14]:

```
# Mean Absolute Proportion Error
ape = numpy.abs(y_simple_residual) / y_train
mape = numpy.mean(ape)
mape
```

Out[14]:

```
1.207372070588855
```

In [15]:

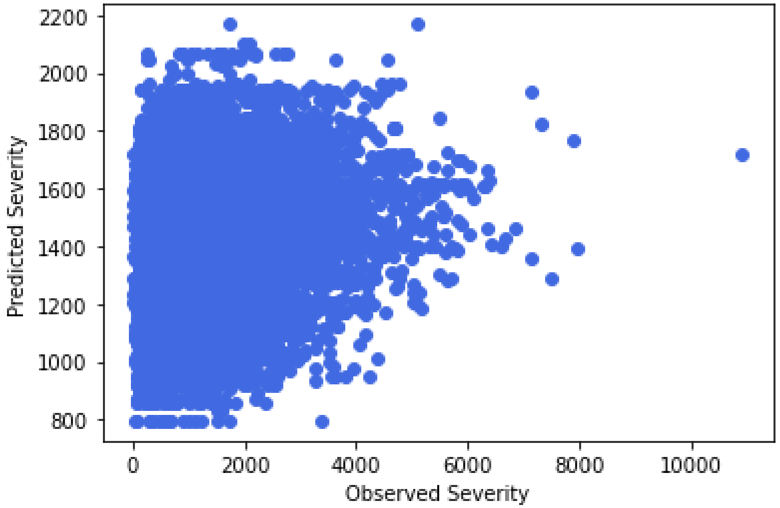
```
# Pearson Correlation
corr_matrix[0,1]
```

Out[15]:

```
0.2305203761686082
```

In [16]:

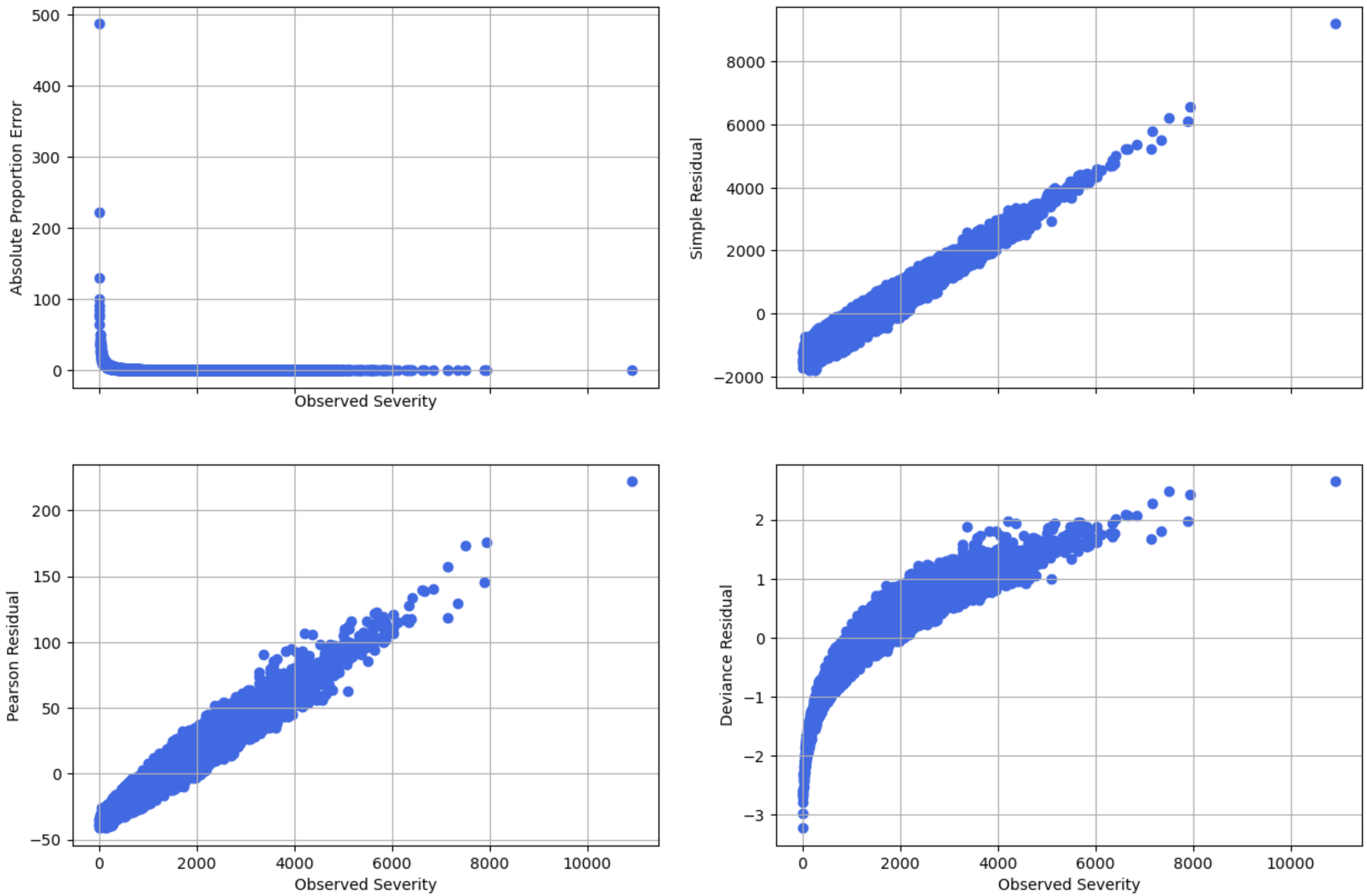
```
y_train = trainData[yName].copy()

# Plot predicted severity versus observed severity
plt.scatter(y_train, y_pred, c = 'royalblue', marker = 'o')
plt.xlabel('Observed Severity')
plt.ylabel('Predicted Severity')
plt.show()

# Based on scatter plots above, we can clearly see some points are out of the group.
# One extreme poor prediction point will be the rightmost point.
print(y_simple_residual[y_simple_residual > 8000])
```



```
21047   9,202.26
dtype: float64
```

In [17]:

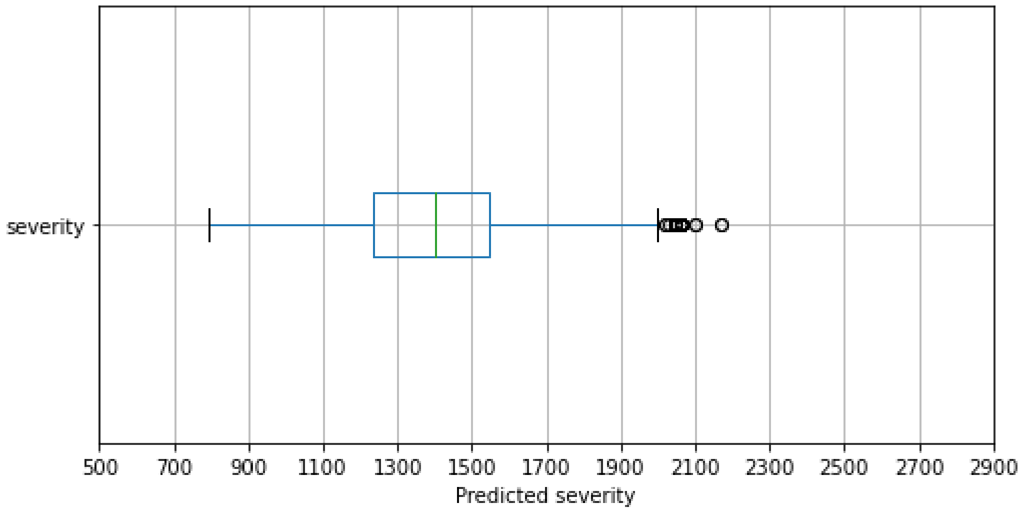
```
fig, ((ax0, ax1), (ax2, ax3)) = plt.subplots(nrows = 2, ncols = 2, dpi = 100, sharex = True,
                                             figsize = (15,10))

# Mean Absolute Proportion Error
ape = numpy.abs(y_simple_residual) / y_train
mape = numpy.mean(ape)
ax0.scatter(y_train, ape, c = 'royalblue', marker = 'o')
ax0.set_xlabel('Observed Severity')
ax0.set_ylabel('Absolute Proportion Error')
ax0.xaxis.grid(True)
ax0.yaxis.grid(True)

# Plot simple residuals versus observed severity
ax1.scatter(y_train, y_simple_residual, c = 'royalblue', marker = 'o')
ax1.set_xlabel('')
ax1.set_ylabel('Simple Residual')
ax1.xaxis.grid(True)
ax1.yaxis.grid(True)

# Plot Pearson residuals versus observed severity
ax2.scatter(y_train, y_pearson_residual, c = 'royalblue', marker = 'o')
ax2.set_xlabel('Observed Severity')
ax2.set_ylabel('Pearson Residual')
ax2.xaxis.grid(True)
ax2.yaxis.grid(True)

# Plot deviance residuals versus observed severity
ax3.scatter(y_train, y_deviance_residual, c = 'royalblue', marker = 'o')
ax3.set_xlabel('Observed Severity')
ax3.set_ylabel('Deviance Residual')
ax3.xaxis.grid(True)
ax3.yaxis.grid(True)

plt.show()


#For the deviance residual plot, there are some points above the arc, which is poorly predicted points.
```



Absolute Proportion Error
Simple Residual
Pearson Residual
Deviance Residual
Observed Severity
Observed Severity
Observed Severity

In [18]:

```
# Identify extremely high predictions
plotData = pandas.DataFrame(y_pred, columns = [yName])
plotData.boxplot(column = yName,vert = False, figsize = (8,4))
plt.xlabel('Predicted severity')
plt.ylabel('')
plt.xticks(numpy.arange(500, 3000, 200))

plt.suptitle('')
plt.title('')
plt.grid(True)
plt.show()

# Based on boxplot, breakdown by 2100, two extreme outliers are these following points
print(y_pred[y_pred > 2100])
```




```
3734    2,100.571
```

```
5802     2,100.571
13726     2,169.18
21664     2,169.18
dtype: float64
```

Question 2

In [19]:

```
trainData = pandas.read_excel("Homeowner_Claim_History.xlsx")

catName = ["f_primary_age_tier","f_primary_gender","f_marital",
           "f_residence_location","f_fire_alarm_type",
           "f_mile_fire_station","f_aoi_tier"]
#intName = []

trainData["severity"] = trainData["amt_claims"] / trainData["num_claims"]
trainData = trainData[trainData["severity"]>0]
yName = "severity"
y = trainData[yName]

n_sample = trainData.shape[0]

X = pandas.get_dummies(trainData[catName].astype('category'))
#X = X.join(trainData[intName])
```

(a) (20 points). Train a Multi-Layer Perceptron neural network. The target variable is Severity (use only positive and non-missing values for analyses). The predictors are the seven categorical predictors. Perform a naïve grid search to select the best network structure. For each Hyperbolic Tangent and Rectified Linear Unit activation function, try the number of layers from 1 to 10, the common number of neurons per layer from 1 to 5. Provide a table that shows your grid search results. The table should contain (1) the activation function type, (2) the number of layers, (3) the common number of neurons per layer, (4) the total number of neurons, and (5) the mean absolute proportion error.

In [20]:

```
# Grid Search for the best neural network architecture
result = pandas.DataFrame()
actFunc = ['relu','tanh']
nLayer = range(1,11,1)
nHiddenNeuron = range(1,6,1)
```

In [*]:

```
combList = itertools.product(actFunc, nLayer, nHiddenNeuron)

for comb in combList:
    time_begin = time.time()
    actFunc = comb[0]
    nLayer = comb[1]
    nHiddenNeuron = comb[2]
    ntotalNeuron = nLayer * nHiddenNeuron

    nnObj = nn.MLPRegressor(hidden_layer_sizes = (nHiddenNeuron,)*nLayer,
                            activation = actFunc, verbose = False,
                            max_iter = 3000, random_state = 31010)
    thisFit = nnObj.fit(X, y)
    y_pred = nnObj.predict(X)

    y_simple_residual = y - y_pred

    # Root Mean Squared Error
    mse = numpy.mean(numpy.power(y_simple_residual, 2))
    rmse = numpy.sqrt(mse)

    # Relative Error
    relative_error = mse / numpy.var(y, ddof = 0)


    # Mean Absolute Proportion Error
    ape = numpy.abs(y_simple_residual) / y
    mape = numpy.mean(ape)

    # pearson corr
    pearson = numpy.corrcoef(y, y_pred)[0,1]

    elapsed_time = time.time() - time_begin
    result = result.append([[actFunc, nLayer, nHiddenNeuron,ntotalNeuron, elapsed_time, rmse, relative_error, mape, p

result.columns = ['Activation Function', 'nLayer','nHiddenNeuron','ntotalNeuron','Elapsed Time', "RMSE", "Relative Er
```

I just set max_iter to 3000 since my laptop is too slow and old. I set max_iter to 20000 and run for 10hrs but

I just set max_iter to 5000 since my laptop is too slow and old. I set max_iter to 20000 and ran for 10hrs but still not complete. So, I know this answer may not valid since it's not converge, but I can do nothing about it...But I believe my method is correct.

(b) (10 points) Recommend the best network structure which yields the lowest Mean Absolute Proportion Error. In the case of ties, choose the network with a fewer total number of neurons.

In [50]:

```
# lowest MAPE is: relu, nlayer=8, nHiddenNeuron=4
result.sort_values("MAPE").head(5)
```

Out[50]:

	Activation Function	nLayer	nHiddenNeuron	ntotalNeuron	Elapsed Time	RMSE	Relative Error	MAPE	Pearson Correlation
38	relu	8	4	32	1.968249	959.9008	0.9825851	1.042598	0.2303976
28	relu	6	4	24	2.962625	958.6042	0.9799324	1.055405	0.2309077
14	relu	3	5	15	0.7113676	952.797	0.9680956	1.070993	0.2312666
19	relu	4	5	20	0.5345666	949.9618	0.9623427	1.093491	0.2305496
39	relu	8	5	40	2.115183	948.2011	0.9587787	1.103594	0.2314461

(c) (10 points) Assess the final model goodness-of-fit using (1) Root Mean Squared Error, (2) Relative Error, (3) Mean Absolute Proportion Error, and (4) Pearson Correlation. What are the values of these metrics?

In [65]:

```
result = pandas.DataFrame()
time_begin = time.time()
actFunc = 'relu'
nLayer = 8
nHiddenNeuron = 4
ntotalNeuron = 8*4

nnObj = nn.MLPRegressor(hidden_layer_sizes = (nHiddenNeuron,)*nLayer,
                        activation = actFunc, verbose = False, learning_rate_init = 0.1,
                        max_iter = 3000, random_state = 31010)
thisFit = nnObj.fit(X, y)
y_pred = nnObj.predict(X)

y_simple_residual = y - y_pred

# Root Mean Squared Error
mse = numpy.mean(numpy.power(y_simple_residual, 2))
rmse = numpy.sqrt(mse)

# Relative Error
relative_error = mse / numpy.var(y, ddof = 0)

# Mean Absolute Proportion Error
ape = numpy.abs(y_simple_residual) / y
mape = numpy.mean(ape)

# pearson corr
pearson = numpy.corrcoef(y, y_pred)[0,1]

elapsed_time = time.time() - time_begin
result = result.append([[actFunc, nLayer, nHiddenNeuron,ntotalNeuron, elapsed_time, rmse, relative_error, mape, pears

result.columns = ['Activation Function', 'nLayer','nHiddenNeuron','ntotalNeuron','Elapsed Time', "RMSE", "Relative Er
```

In [66]:
```
result
```

Out[66]:

	Activation Function	nLayer	nHiddenNeuron	ntotalNeuron	Elapsed Time	RMSE	Relative Error	MAPE	Pearson Correlation
0	relu	8	4	32	0.8808901	959.9008	0.9825851	1.042598	0.2303976

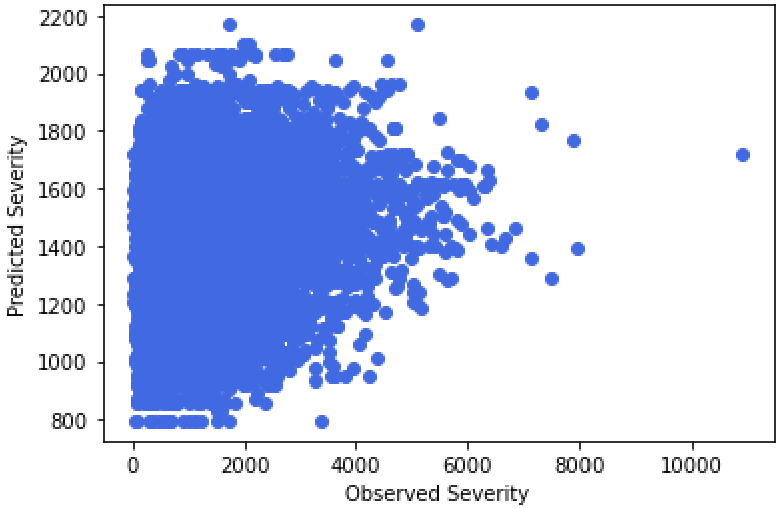
I just set max_iter to 3000 since my laptop is too slow and old. I set max_iter to 20000 and run for 10hrs but still not complete. So, I know this answer may not valid since it's not converge, but I can do nothing about it...But I believe my method is correct.

(d) (10 points) Identify any poorly predicted observations. First, plot the predicted versus the observed Severity. Second, together in a single chart frame, plot the Simple Residuals, the Pearson Residuals, and the Absolute Proportion Errors versus the observed Severity. Label the axes of these two charts accordingly. To receive full credits, generate your charts with proper dimensions (e.g., length and width) and resolution (e.g., dpi).

In [ ]:
```
# R-Squared
corr_matrix = numpy.corrcoef(y_train, y_pred)
rsq = corr_matrix[0,1] ** 2

# Pearson Residual
y_pearson_residual = y_simple_residual / numpy.sqrt(y_pred)

# Deviance Residual
r_vec = y_train / y_pred
di_2 = 2* (r_vec - numpy.log(r_vec) - 1)
y_deviance_residual = numpy.where(y_simple_residual > 0, 1.0, -1.0) * numpy.sqrt(di_2)
```

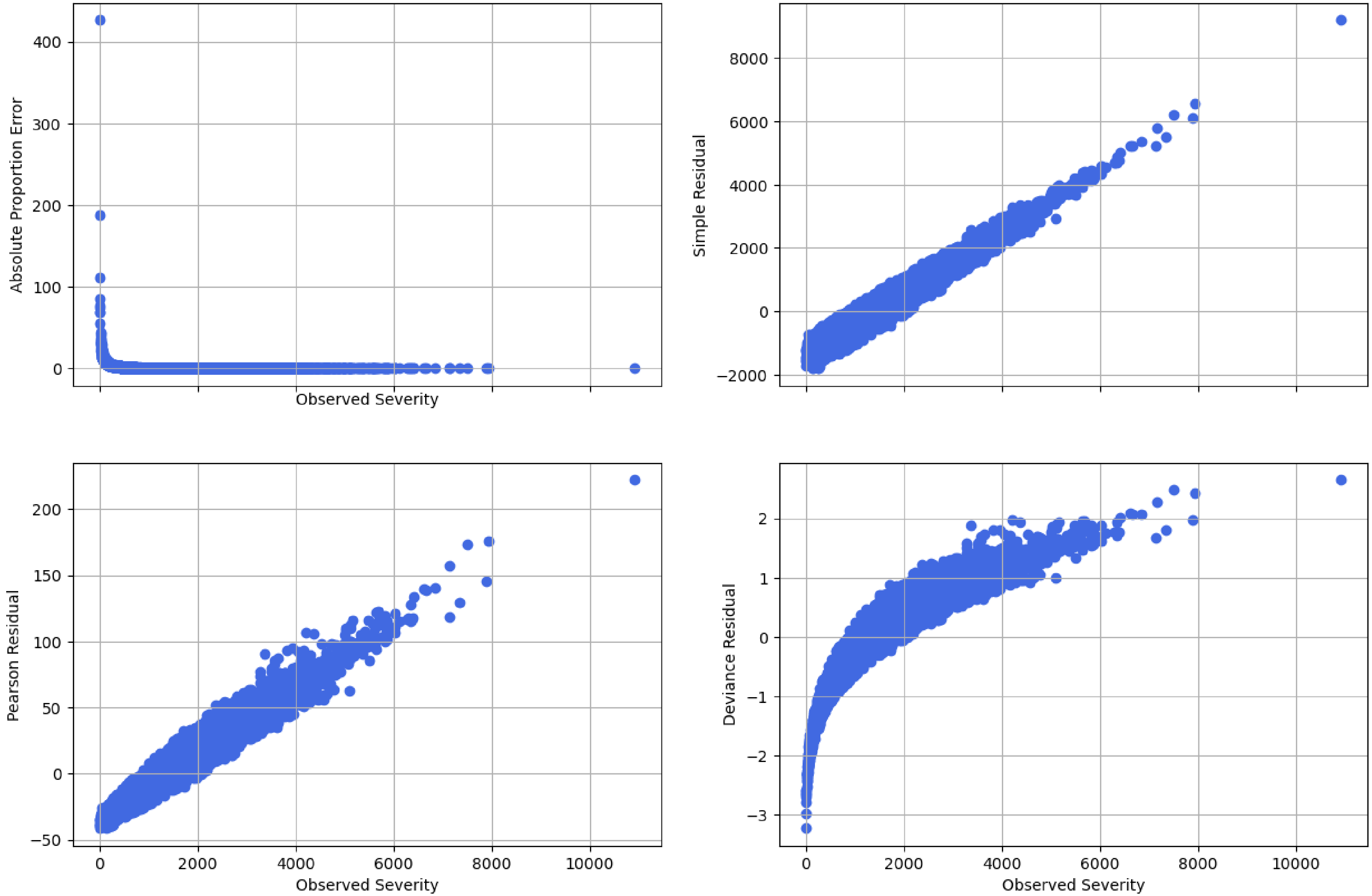
In [106]:

```
y_train = trainData[yName].copy()

# Plot predicted severity versus observed severity
plt.scatter(y_train, y_pred, c = 'royalblue', marker = 'o')
plt.xlabel('Observed Severity')
plt.ylabel('Predicted Severity')
plt.show()
```



In [97]:

```
fig, ((ax0, ax1), (ax2, ax3)) = plt.subplots(nrows = 2, ncols = 2, dpi = 100, sharex = True,
                                             figsize = (15,10))

# Mean Absolute Proportion Error
ape = numpy.abs(y_simple_residual) / y_train
mape = numpy.mean(ape)
ax0.scatter(y_train, ape, c = 'royalblue', marker = 'o')
ax0.set_xlabel('Observed Severity')
ax0.set_ylabel('Absolute Proportion Error')
ax0.xaxis.grid(True)
ax0.yaxis.grid(True)

# Plot simple residuals versus observed sale price
ax1.scatter(y_train, y_simple_residual, c = 'royalblue', marker = 'o')
ax1.set_xlabel('')
ax1.set_ylabel('Simple Residual')
ax1.xaxis.grid(True)
ax1.yaxis.grid(True)

# Plot Pearson residuals versus observed sale price
ax2.scatter(y_train, y_pearson_residual, c = 'royalblue', marker = 'o')
ax2.set_xlabel('Observed Severity')
ax2.set_ylabel('Pearson Residual')
ax2.xaxis.grid(True)
ax2.yaxis.grid(True)

# Plot deviance residuals versus observed sale price
ax3.scatter(y_train, y_deviance_residual, c = 'royalblue', marker = 'o')
ax3.set_xlabel('Observed Severity')
ax3.set_ylabel('Deviance Residual')
ax3.xaxis.grid(True)
ax3.yaxis.grid(True)

plt.show()
```

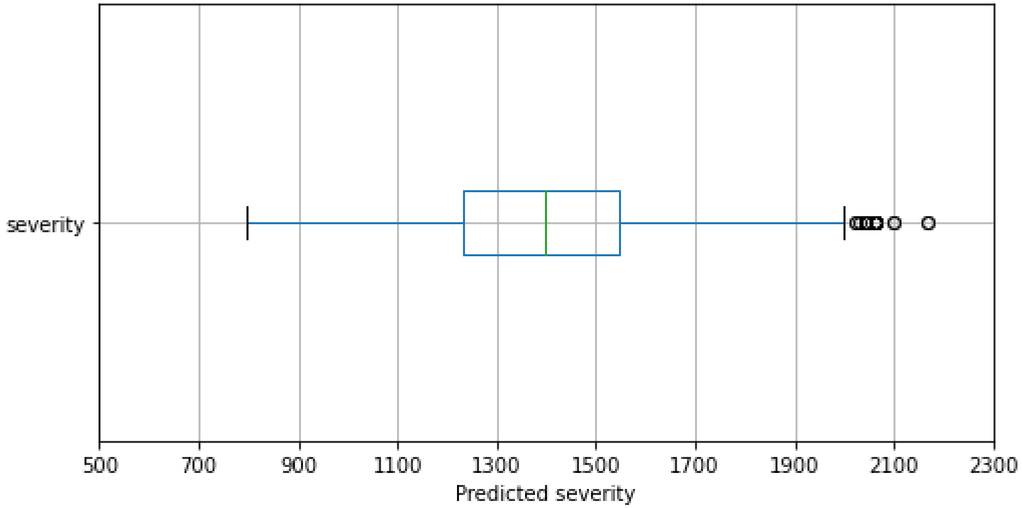


In [*]:

```
#For the deviance residual plot, there are some points above the arc, which is poorly predicted points.
```

In [111]:

```
plotData = pandas.DataFrame(y_pred, columns = [yName])
plotData.boxplot(column = yName,vert = False, figsize = (8,4))
plt.xlabel('Predicted severity')
plt.ylabel('')
plt.xticks(numpy.arange(500, 2500, 200))

plt.suptitle('')
plt.title('')
plt.grid(True)
plt.show()
```



In [112]:

```
# Based on boxplot, breakdown by 2100, two extreme outliers are these following points
print(y_pred[y_pred > 2100])
```

```
3734    2,100.571
5802    2,100.571
13726    2,169.18
21664    2,169.18
dtype: float64
```

In [ ]: