ALLSTATE CLAIMS SEVERITY

In [3]:
```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import math
from sklearn import neighbors, datasets, ensemble, cross_validation, uti
ls
from sklearn.metrics import mean_absolute_error
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
import statsmodels.api as sm
from sklearn.cross_validation import cross_val_score
from scipy import stats
```

EXPLORATORY DATA ANALYSIS

First, we will find how many unique values there are in each column containing "cat#", and list out in an array what those unique values are. Hopefully by these counts we are able to find some type of pattern!

In [4]:
```
#Read data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [18]:
```
#Find unique values
cat_names = list(train)
# for name in cat_names :
#     if name.find("cat") != -1 :
#         print(name, ":", train[name].unique(), "Total unique values:
 ", len(train[name].unique()))
# For now suppressing print because too long pdf
```

From this analysis we can speculate:

- columns "cat1" through "cat72" might be True/False or Yes/No
- columns "cat73" through "cat76" might be Low/Medium/High values, such as income class or the like with 3 categories
- columns "cat77" through "cat88" also seem to be category-based, with 4 categories each
- specifically for column 112, it may represent states as there is 51 unique values.

Now, let's also check some statistics about quantitative variables.

In [4]:
```
#Basic Statistics
train.describe()
```

Out[4]:

	id	cont1	cont2	cont3	cont4	co
count	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	18
mean	294135.982561	0.493861	0.507188	0.498918	0.491812	0.4
std	169336.084867	0.187640	0.207202	0.202105	0.211292	0.2
min	1.000000	0.000016	0.001149	0.002634	0.176921	0.2
25%	147748.250000	0.346090	0.358319	0.336963	0.327354	0.2
50%	294539.500000	0.475784	0.555782	0.527991	0.452887	0.4
75%	440680.500000	0.623912	0.681761	0.634224	0.652072	0.6
max	587633.000000	0.984975	0.862654	0.944251	0.954297	0.9

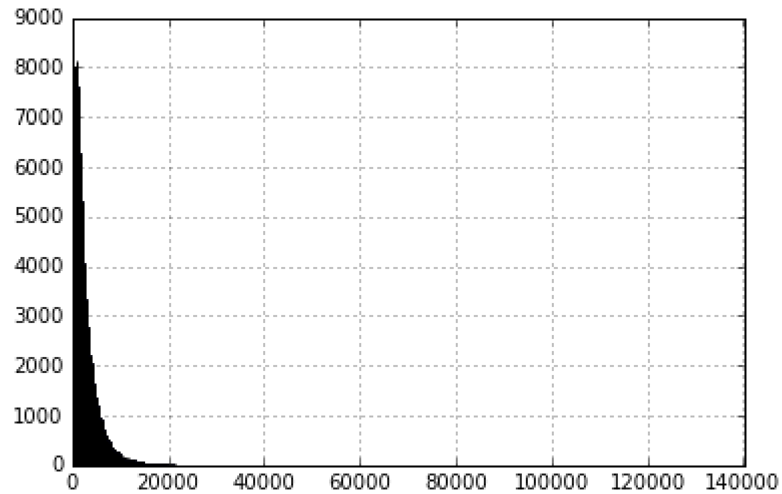
In [5]:
```
#Skewness of Continuous Variables
train.skew()
```

Out[5]:
```
id       -0.002155
cont1     0.516424
cont2    -0.310941
cont3    -0.010002
cont4     0.416096
cont5     0.681622
cont6     0.461214
cont7     0.826053
cont8     0.676634
cont9     1.072429
cont10    0.355001
cont11    0.280821
cont12    0.291992
cont13    0.380742
cont14    0.248674
loss      3.794958
dtype: float64
```

We can clearly see there is high skew factor in loss - let's check what the histogram of it looks like.

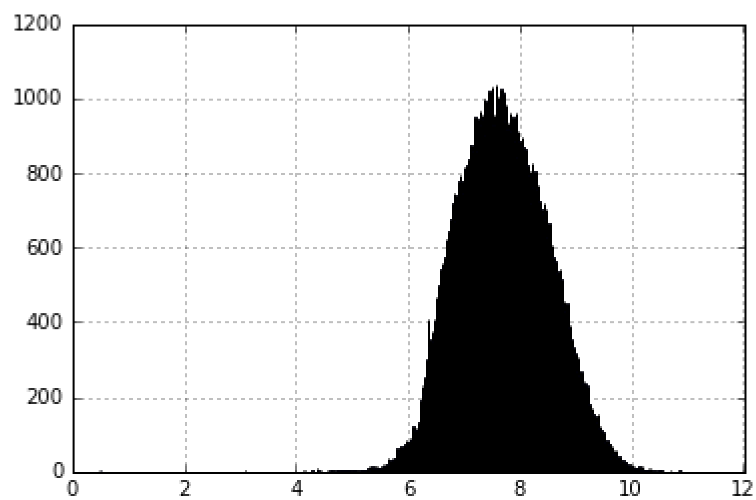
In [6]:
```
# histogram of loss
train['loss'].hist(bins=1000)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x118664898>



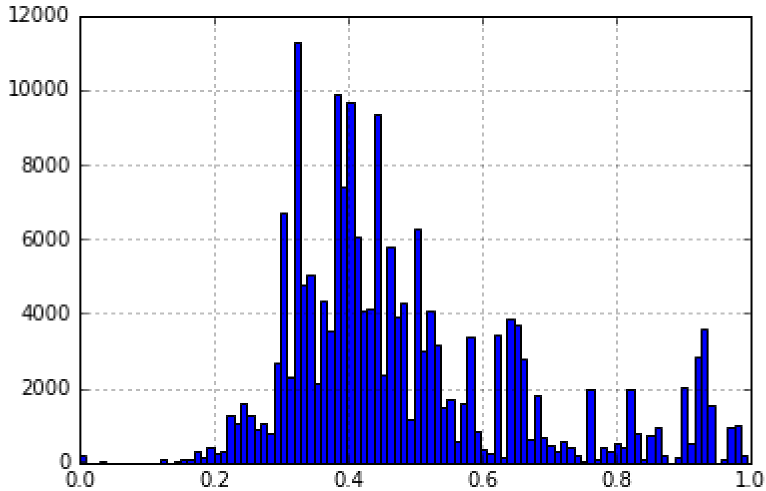
In [7]:
```
# looks not very normal. let's log(1+x) it
np.log1p(train['loss']).hist(bins=1000)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x120b71160>



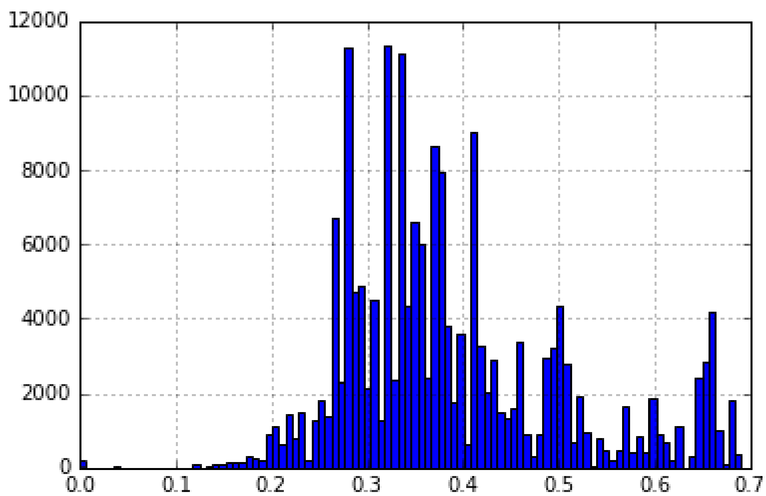
In [28]:
```
#How about cont9, the highest-skewed feature?
train['cont9'].hist(bins=100)
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x11fb25b38>



In [29]:
```
#Try long(1+x) to see if there is major improvements?
np.log1p(train['cont9']).hist(bins=100)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1252c39b0>



Skew of the loss function was much improved through log(1+x)'ing it, but not cont9, the highest-skewed feature. So for now let's keep in mind to log(1+x) the loss variable (our response variable).

FEATURE ENGINEERING

First, let's do the simple things: Unskew loss:

In [5]:
```
train['loss'] = np.log1p(train['loss'])
```

Now, let us fill in the missing variables in continuous variables, with their means.

In [6]:
```
#Just checking: Are any values missing?
train.isnull().values.any()
```

Out[6]:
```
False
```

Since no values are missing, no need to fill in NAN's with means.

In [15]:
```
#No values missing, no need to do this:
#train = train.fillna(train.mean())
```

Let's see which variables are correlated/unnecessary variables and drop them:

In [44]:
```
#First only getting continuous variable columns
cont_columns = []

for i in train.columns:
    if train[i].dtype == 'float':
        cont_columns.append(i)
cont_columns
```

Out[44]:
```
['cont1',
 'cont2',
 'cont3',
 'cont4',
 'cont5',
 'cont6',
 'cont7',
 'cont8',
 'cont9',
 'cont10',
 'cont11',
 'cont12',
 'cont13',
 'cont14',
 'loss']
```

In [11]:
```
train.loc[:, cont_columns].corr()
```

Out[11]:

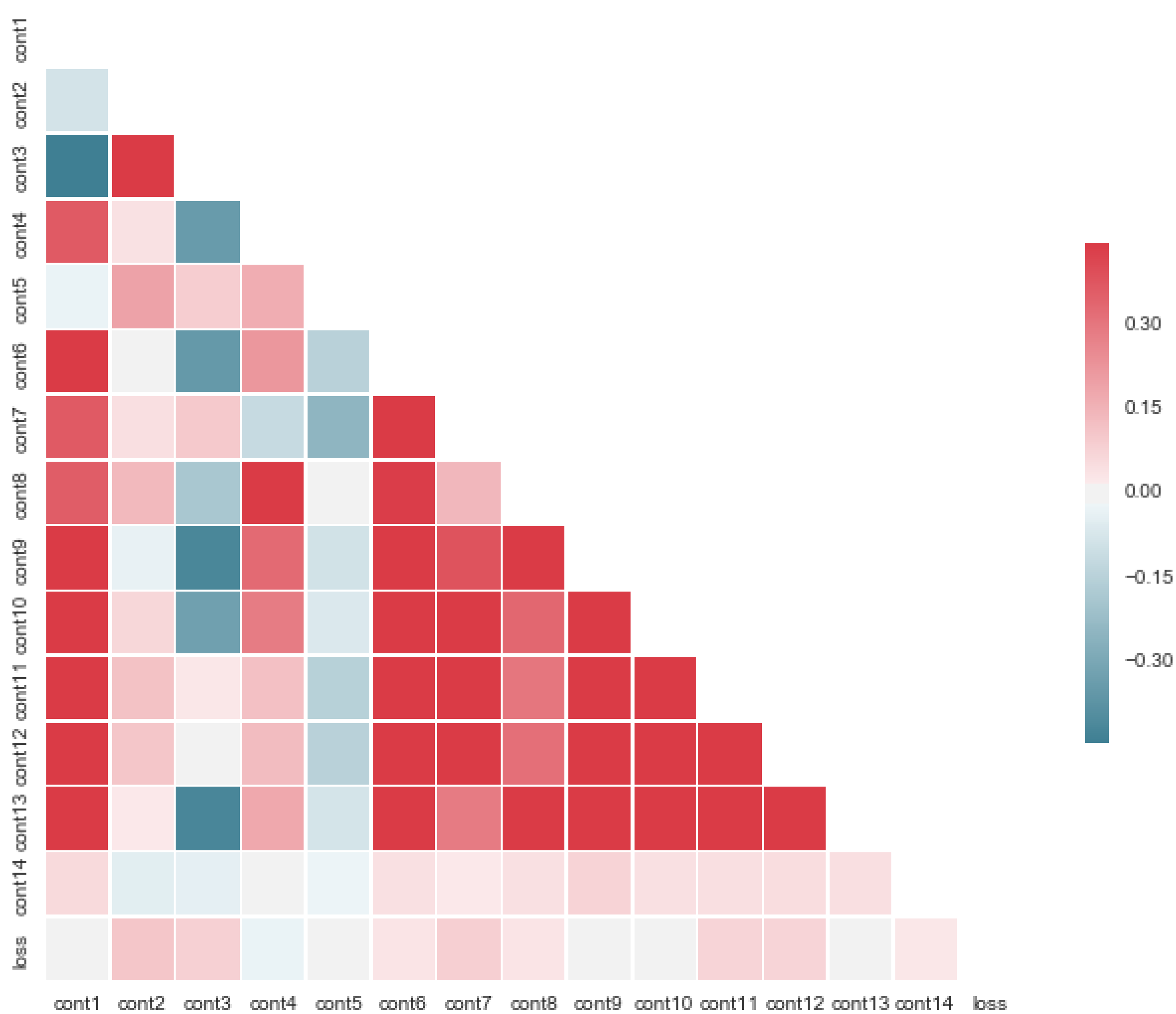
	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8
cont1	1.000000	-0.085180	-0.445431	0.367549	-0.025230	0.758315	0.367384	0.361
cont2	-0.085180	1.000000	0.455861	0.038693	0.191427	0.015864	0.048187	0.137
cont3	-0.445431	0.455861	1.000000	-0.341633	0.089417	-0.349278	0.097516	-0.185
cont4	0.367549	0.038693	-0.341633	1.000000	0.163748	0.220932	-0.115064	0.528
cont5	-0.025230	0.191427	0.089417	0.163748	1.000000	-0.149810	-0.249344	0.009
cont6	0.758315	0.015864	-0.349278	0.220932	-0.149810	1.000000	0.658918	0.437
cont7	0.367384	0.048187	0.097516	-0.115064	-0.249344	0.658918	1.000000	0.142
cont8	0.361163	0.137468	-0.185432	0.528740	0.009015	0.437437	0.142042	1.000
cont9	0.929912	-0.032729	-0.417054	0.328961	-0.088202	0.797544	0.384343	0.452
cont10	0.808551	0.063526	-0.325562	0.283294	-0.064967	0.883351	0.492621	0.336
cont11	0.596090	0.116824	0.025271	0.120927	-0.151548	0.773745	0.747108	0.302
cont12	0.614225	0.106250	0.006111	0.130453	-0.148217	0.785144	0.742712	0.315
cont13	0.534850	0.023335	-0.418203	0.179342	-0.082915	0.815091	0.288395	0.476
cont14	0.056688	-0.045584	-0.039592	0.017445	-0.021638	0.042178	0.022286	0.043
loss	-0.007335	0.104666	0.081548	-0.027523	-0.014958	0.031517	0.085095	0.032

Correlated Variables:

- cont1/cont9
- cont1/cont10
- cont6 with a lot of variables, just remove
- cont7 with cont11/12
- cont9 with cont10
- cont10 with a lotta variables
- cont 11 with cont12
- cont6 with cont13

In [6]:
```
#This code was taken and modified from Kaggle user "denoiser"'s kernel
 "Simple EDA - feature transformations"

# Compute the correlation matrix
corr = train[cont_columns].corr()

sns.set(style="white")

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x293ab2554a8>



By the looks of the previous two analyses, the most uncorrelated 'cont' variables that would serve as better predictors are:

- cont2, cont3, cont4, cont5, and cont14

In [7]:
```
#Knock out the overly correlated columns
unwanted_list = ['id', 'cont1','cont6','cont7','cont8','cont9','cont10',
 'cont11','cont12','cont13']
train.drop(unwanted_list, axis = 1, inplace = True)
```

In [8]:
```
#For some reason the above code produces an error, but the action is don
e here anyways,
#as verified by taking a peek at the new training dataset
train.head(3)
```

Out[8]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cat113	cat114	cat115	cat11(
0	1	A	B	A	B	A	A	A	A	B	...	S	A	O	LB
1	2	A	B	A	A	A	A	A	A	B	...	BM	A	O	DP
2	5	A	B	A	A	B	A	A	A	B	...	AF	A	I	GK

3 rows × 123 columns

We took a look at the recommended transformations on categorical variables given in the kernel "Simple EDA - feature transformations"by user 'denoiser', and applied these transformations on the training set in the cont categories that we kept.

In [8]:
```
#Apply appropriate transformations on the columns we want -- gosh, I sou
nd like a commercial lol


train['cont2'] = np.tan(train['cont2'])
train['cont4'] = stats.boxcox(train['cont4'])[0]
train['cont5'] = stats.boxcox(train['cont5'])[0]
train.head(3)
```

Out[8]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	...	cat113	cat114	cat115	ca
0	A	B	A	B	A	A	A	A	B	A	...	S	A	O	LE
1	A	B	A	A	A	A	A	A	B	B	...	BM	A	O	DI
2	A	B	A	A	B	A	A	A	B	B	...	AF	A	I	GI

3 rows × 122 columns

Next, let's create one-hot encodings for categorical variables.

In [9]:
```
#drop_first = True to remove perfect multicollinearity
train = pd.get_dummies(train, drop_first=True)
```

PREPARING DATA

In [10]:
```
X = train.drop('loss', axis = 1)
Y = train['loss']
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, r
andom_state=123)
X_train.shape
```

Out[10]:
```
(150654, 1028)
```

LINEAR REGRESSION

In [15]:
```
from sklearn import linear_model
```

In [62]:
```
#Just checking the shape of train and test shape for validation
print("Local train shape: ", X_train.shape)
print("Local test shape: ", X_val.shape)
```

```
Local train shape:  (150654, 1038)
Local test shape:  (37664, 1038)
```

Now let us run the model!

In [17]:
```
clf = linear_model.LinearRegression()
clf.fit(X_train, Y_train)
```

Out[17]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=F
alse)
```

Let's cross validate the model and see if it's robust.

In [41]:
```
#Function for cross-validation scores
def cv_stats(cv_score):
    """ Returns the mean and standard deviation in a readable format"""
    mean = np.mean(cv_score)
    std = np.std(cv_score)
    return mean, std
```

In [45]:
```
clf_score = cross_val_score(clf, X_train, Y_train, cv = 5)
cv_stats(clf_score)
```

Out[45]:
```
(0.51487530645278956, 0.0039174138628955387)
```

Let's try running this model for local_test or local validation.

In [18]:
```
local_y_pred = clf.predict(X_val)
```

Converting log(1+x)'ed scores back into actual loss response variables to check for MAE.

In [22]:
```
local_lin_mae = mean_absolute_error(np.expm1(Y_val), np.expm1(local_y_pr
ed))
local_lin_mae
```

Out[22]:
```
1244.5386469308219
```

RANDOM FOREST

Linear Regression ain't good enough, so Random Forest it is: recall that from cat112, we believe that the data can be separated by state (51, including DC). So let's set the n_estimators to 51.

In [1]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [11]:
```
rf = RandomForestRegressor(n_estimators=51, random_state = 123, n_jobs =
 3)
rf.fit(X_train,Y_train)
preds = rf.predict(X_val)
preds[0:5]
```

Out[11]:
```
array([ 6.93813926,  7.81986002,  7.46552968,  8.12533434,  9.1696390
3])
```

In [12]:
```
local_rf_mae = mean_absolute_error(np.expm1(Y_val), np.expm1(preds))
local_rf_mae
```

Out[12]:
```
1216.4064324458941
```

MAKING A SUBMISSION

Feature Engineering Test Data:

In [14]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [15]:
```
#Re-preparing data for global/total dataset
Y_train = np.log1p(train['loss'])
Y_train.head()
```

Out[15]:
```
0    7.702637
1    7.158203
2    8.008396
3    6.846784
4    7.924742
Name: loss, dtype: float64
```

In [16]:
```
data = pd.concat([train, test], axis = 0)
ID = test['id']
unwanted_list = ['id', 'cont1','cont6','cont7','cont8','cont9','cont10',
 'cont11','cont12','cont13']
data.drop(unwanted_list, axis = 1, inplace = True)
data['cont2'] = np.tan(data['cont2'])
data['cont4'] = stats.boxcox(data['cont4'])[0]
data['cont5'] = stats.boxcox(data['cont5'])[0]
data = pd.get_dummies(data, drop_first=True)
data.shape
```

Out[16]:
```
(313864, 1066)
```

In [17]:
```
X_train = data[pd.notnull(data['loss'])].drop('loss', axis = 1)
X_train.shape
```

Out[17]:
```
(188318, 1065)
```

Training Model with all train.csv datapoints:

In [19]:
```
best_model = RandomForestRegressor(n_estimators = 51, random_state =
123, n_jobs = 3)
best_model.fit(X_train,Y_train)
```

Out[19]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None, min_samples_leaf=
1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=51, n_jobs=3, oob_score=False, random_state=12
3,
           verbose=0, warm_start=False)
```

Making Submission:

In [35]:
```
X_test = data[pd.isnull(data['loss'])].drop('loss', axis = 1)
predictions = np.expm1(best_model.predict(X_test))
predictions[0:5]
```

Out[35]:
```
array([ 1982.601879  ,  2311.00095181,  9406.93242042,  5280.8919217 ,
         718.73366417])
```

In [40]:
```
with open("submission_new.csv", "w") as subfile:
    subfile.write("id,loss\n")
    for i, pred in enumerate(list(predictions)):
        subfile.write("%s,%s\n"%(ID[i],pred))
```

SIDE: XGBOOST

In [39]:
```
from xgboost import XGBRegressor
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call l
ast)
<ipython-input-39-b25935c1568f> in <module>()
----> 1 from xgboost import XGBRegressor

ImportError: No module named 'xgboost'
```

In [38]:
```
xgb = XGBRegressor(n_estimators= 51, seed = 124)
xgb.fit(X_train,Y_train)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-38-74e691acf296> in <module>()
----> 1 xgb = XGBRegressor(n_estimators= 51, seed = 124)
      2 xgb.fit(X_train,Y_train)

NameError: name 'XGBRegressor' is not defined
```

In [ ]:
```
xgb_preds = np.expm1(xgb.predict(test))
xgb_preds[0:5]
```

In [ ]:
```
with open("xgb_submission.csv", "w") as subfile:
    subfile.write("id,loss\n")
    for i, pred in enumerate(list(predictions)):
        subfile.write("%s,%s\n"%(ID[i],pred))
```

MODEL INTERPRETATION

Linear Regression

For the Linear Regression Model, we tried to remove as much correlated variables as possible and convert categorical data into dummie, one-hot encoding format so we could regress it properly. However, it did not give us as great of prediction results as we expected, probably because it is one of the simpler machine learning models. Thus, we turned to Random Forests.

Random Forest

We decided to chose n_estimators as 51, mainly because we believed that from cat112, there were 51 unique values, and thus believed the data was split into 51 parts, namely - 51 states. Thus, the individual "trees" would be individual states' data, and within those trees the Gini index would hopefully be further minimized than a random number of trees. Intuitively, we believed that perhaps a larger number of trees could give us better predictions results, but decided to stick with 51 because of computing issues as well as avoiding the possibility of overfitting. Note that we used a RandomForestRegressor instead of the Classifier that we saw in class, because this inherently is a regression, not a classification problem. Furthermore, we intended to use XGBoost as well in case our prediction results were not low enough, but we fortunately got a score of 1194, and thus did not need it (and as a side note: we also ran into installation compatability issues with python 3.5 and conda for xgboost package installation).

Some feature importance on the Random Forest Model:

In [ ]:
```
fimps = DataFrame({"fimps":
best_model.feature_importances_},index=test.columns.values[0:])
fimps.plot(kind='bar')
```

Though we can see these features and their importances, it was diffcult actually trying to understand what they mean because the data and the columns that described the data, were very minimal and we believed it was somewhat pointless to discover which feature were more important, since we do not know what the features are themselves.

Sources/References:

General Reference on Running Code/Running Different Models: https://www.kaggle.com/sharmasanthosh/allstate-claims-severity/exploratory-study-on-ml-algorithms (https://www.kaggle.com/sharmasanthosh/allstate-claims-severity/exploratory-study-on-ml-algorithms)

Thinking about the Random Forest Trees in as 51 States: https://www.kaggle.com/dmi3kno/allstate-claims-severity/all-the-allstate-states-eda (https://www.kaggle.com/dmi3kno/allstate-claims-severity/all-the-allstate-states-eda)

EDA and Feature Engineering of Continuous Variables: https://www.kaggle.com/snmateen/allstate-claims-severity/simple-eda-feature-transformations (https://www.kaggle.com/snmateen/allstate-claims-severity/simple-eda-feature-transformations)

In [ ]: