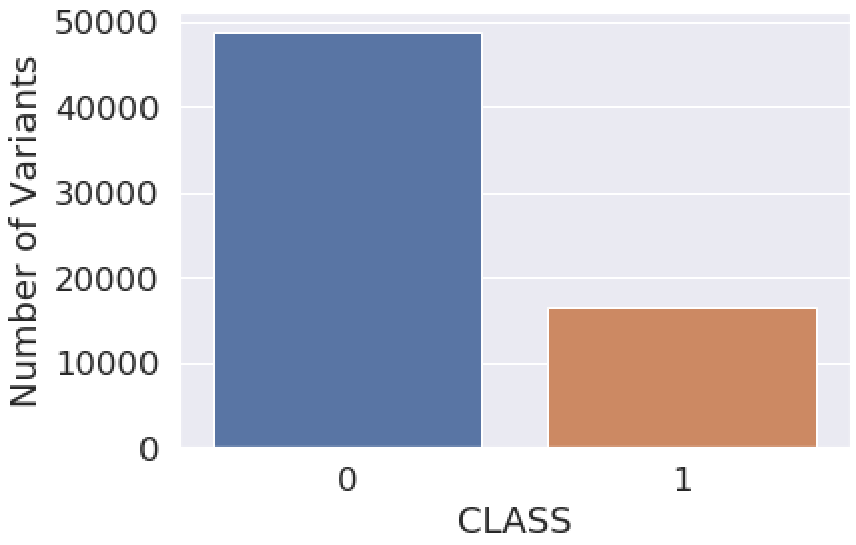
Explore some of the features in the Genetic Variant Classifications dataset. Each record represents a genetic "variant". For a more detailed description of the features please see the dataset (https://www.kaggle.com/kevinarvai/clinvar-conflicting). kaggle (https://www.kaggle.com/kevinarvai/clinvar-conflicting)

In [1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style(style='whitegrid')
sns.set(font_scale=1.5);

import pandas as pd
import re
```

In [2]:
```
df = pd.read_csv('/home/paperspace/Dropbox/input/kaggle/clinvar_conflicting.
```

The `CLASS` distribution is skewed a bit to the `0` class, meaning there are fewer variants with conflicting submissions.

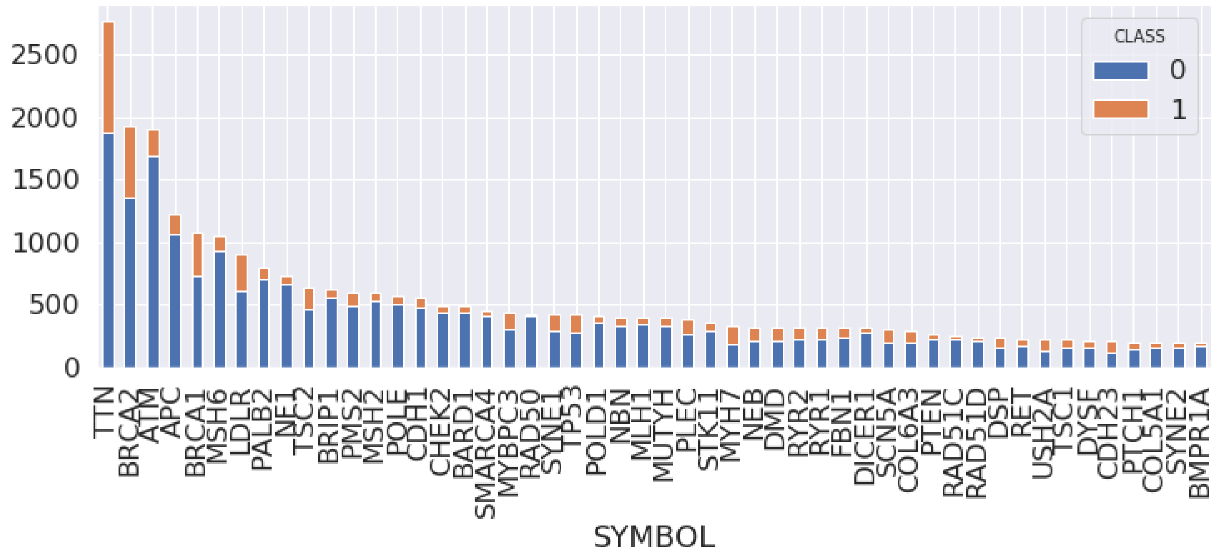
In [3]:
```
ax = sns.countplot(x="CLASS", data=df)
ax.set(xlabel='CLASS', ylabel='Number of Variants');
```



It's clear that conflicting variants are more common in some genes.

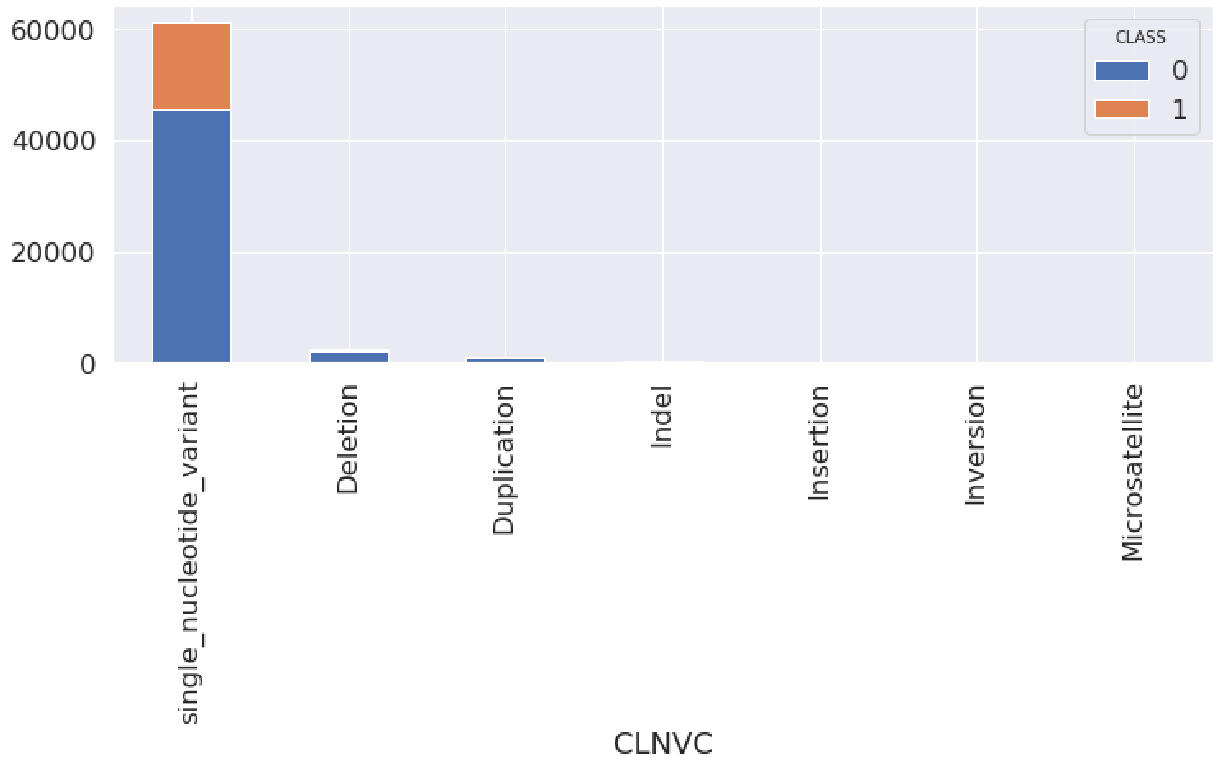
In [4]:
```
gene_ct = pd.crosstab(df.SYMBOL, df.CLASS, margins=True)
```

In [5]:
```
gene_ct = pd.crosstab(df.SYMBOL, df.CLASS, margins=True)
gene_ct.drop('All', axis=0, inplace=True)

# limit to the 50 most submitted genes for visualization
gene_ct = gene_ct.sort_values(by='All', ascending=False).head(50)
gene_ct.drop('All', axis=1, inplace=True)

gene_ct.plot.bar(stacked=True, figsize=(12, 4));
```



CLNVC (Variant Type)

In [6]:
```
vt_ct = pd.crosstab(df.CLNVC, df.CLASS, margins=True)
vt_ct.drop('All', axis=0, inplace=True)

# limit to the 50 most submitted genes for visualization
vt_ct = vt_ct.sort_values(by='All', ascending=False)
vt_ct.drop('All', axis=1, inplace=True)

vt_ct.plot.bar(stacked=True, figsize=(12, 4));
```

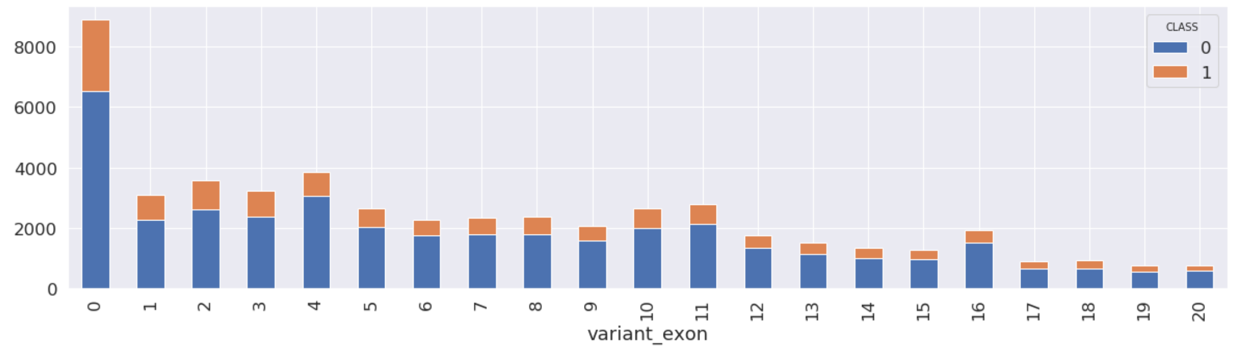


Exons are features of genes that map sequences nucleotides that encode functional parts of DNA. Genes have differing numbers of exons, some have few, some have many. Let's see if, regardless of gene, whether or not conflicting variants are enriched in a general exon location.

In [7]:
```
df.EXON.fillna('0', inplace=True)
df['variant_exon'] = df.EXON.apply(lambda x: [int(s) for s in re.findall(r'\
```

`variant_exon` = 0 represents that the variant is located in an **Intron**. Intron variants seem to be conflicting much more frequently than exon variants.

In [8]:
```
exondf = pd.crosstab(df['variant_exon'], df['CLASS'])
exondf.plot.bar(stacked=True, figsize=(20, 5));
plt.xlim(-0.5, 20.5);
```



Parse and encode the `MC` (molecular consequence) field

In [9]:
```
MC_list = df.MC.dropna().str.split(',').apply(lambda row: list((c.split('|')
MC_encoded = pd.get_dummies(MC_list.apply(pd.Series).stack()).sum(level=0)
MC_encoded = MC_encoded.reindex(index=MC_list.index)

# Incorporate the transformed MC feature into the existing DataFrame
df = df.join(MC_encoded).drop(columns=['MC'])

# Transformed MC feature
MC_encoded.head()
```

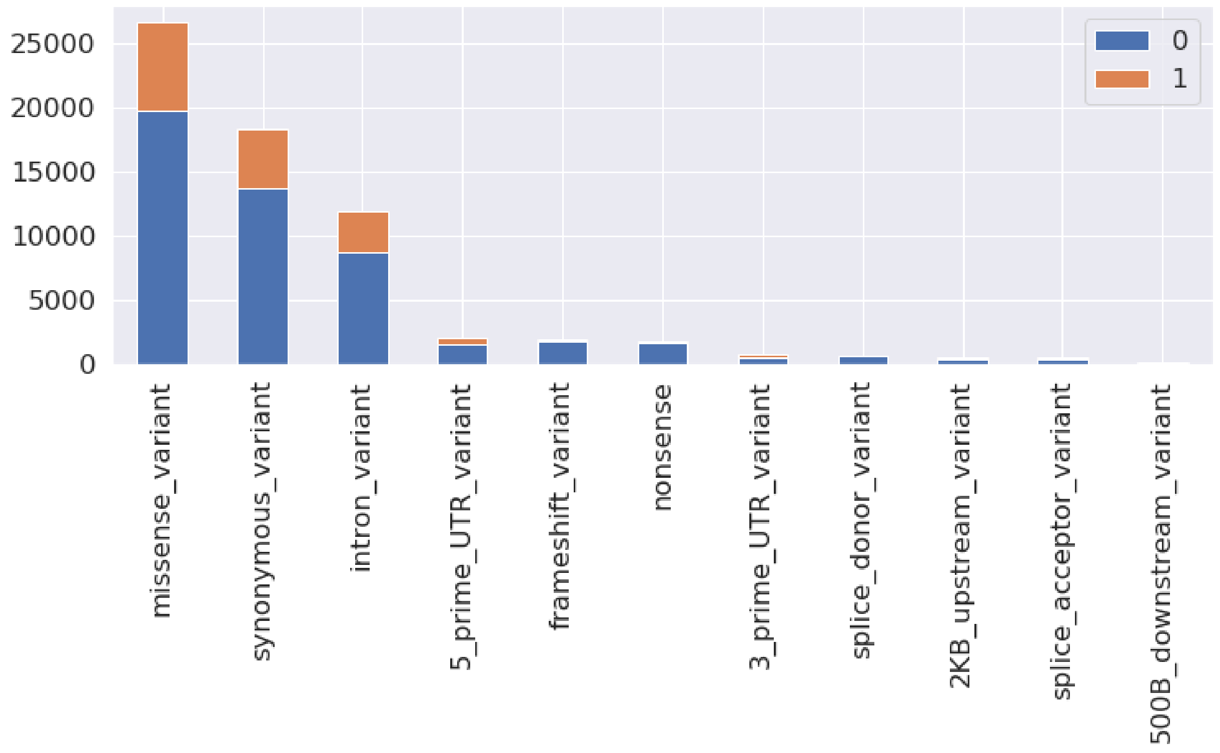
Out[9]:

| | 2KB_upstream_variant | 3_prime_UTR_variant | 500B_downstream_variant | 5_prime_UTR_variant | fram |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | |
| **1** | 0 | 0 | 0 | 0 | |
| **2** | 0 | 0 | 0 | 0 | |
| **3** | 0 | 0 | 0 | 0 | |
| **4** | 0 | 0 | 0 | 0 | |

Manually generate the `crosstab`, there is probably a faster method via `pandas`.

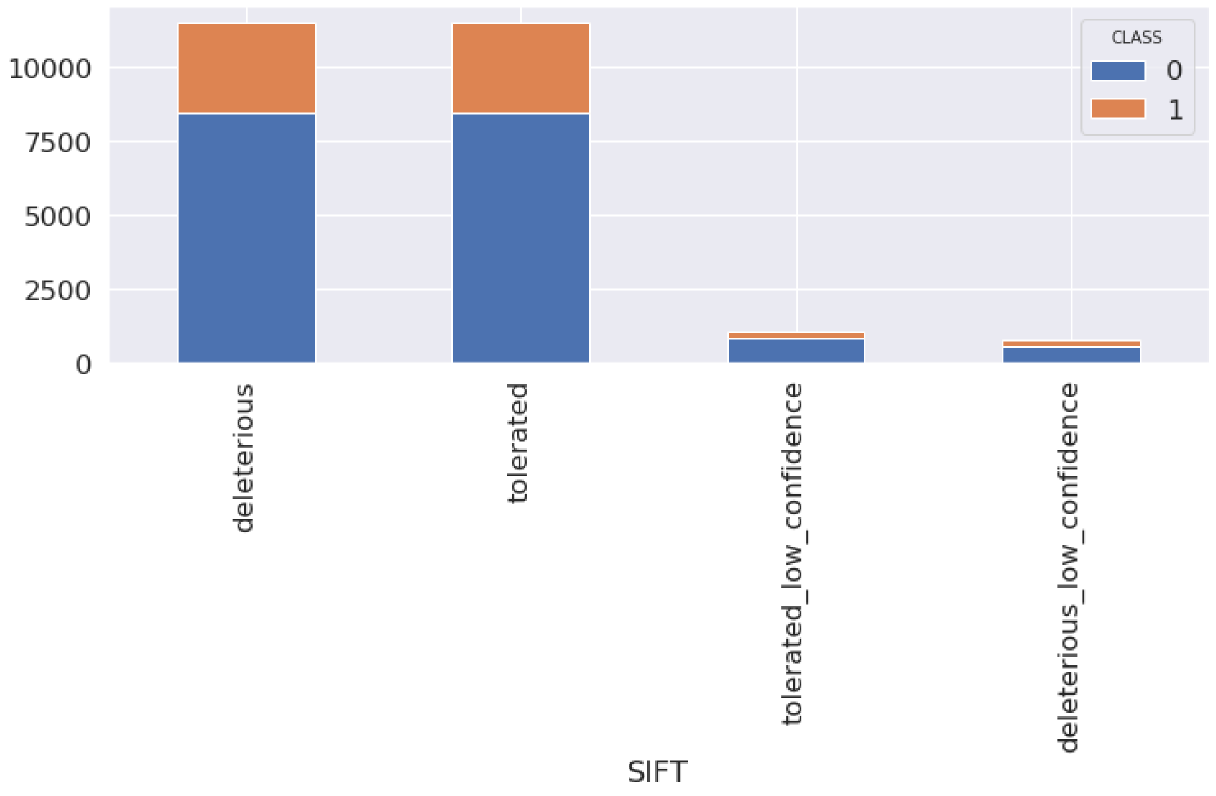
In [10]:
```
mccounts= {0: {},
           1: {},
           'All': {}
          }

for col in MC_encoded.columns:
    for class_ in [0, 1]:
        mccounts[class_][col] = df.loc[df['CLASS'] == class_][col].sum()

    mccounts['All'][col] = df[col].sum()

mc_ct = pd.DataFrame.from_dict(mccounts)

mc_ct_all = mc_ct.sum(axis=0)
mc_ct_all.name = 'All'
mc_ct = mc_ct.append(mc_ct_all, ignore_index=False)
```

In [11]:
```
mc_ct.drop('All', axis=0, inplace=True)

mc_ct = mc_ct.sort_values(by='All', ascending=False)
mc_ct.drop('All', axis=1, inplace=True)

mc_ct.plot.bar(stacked=True, figsize=(12, 4));
```

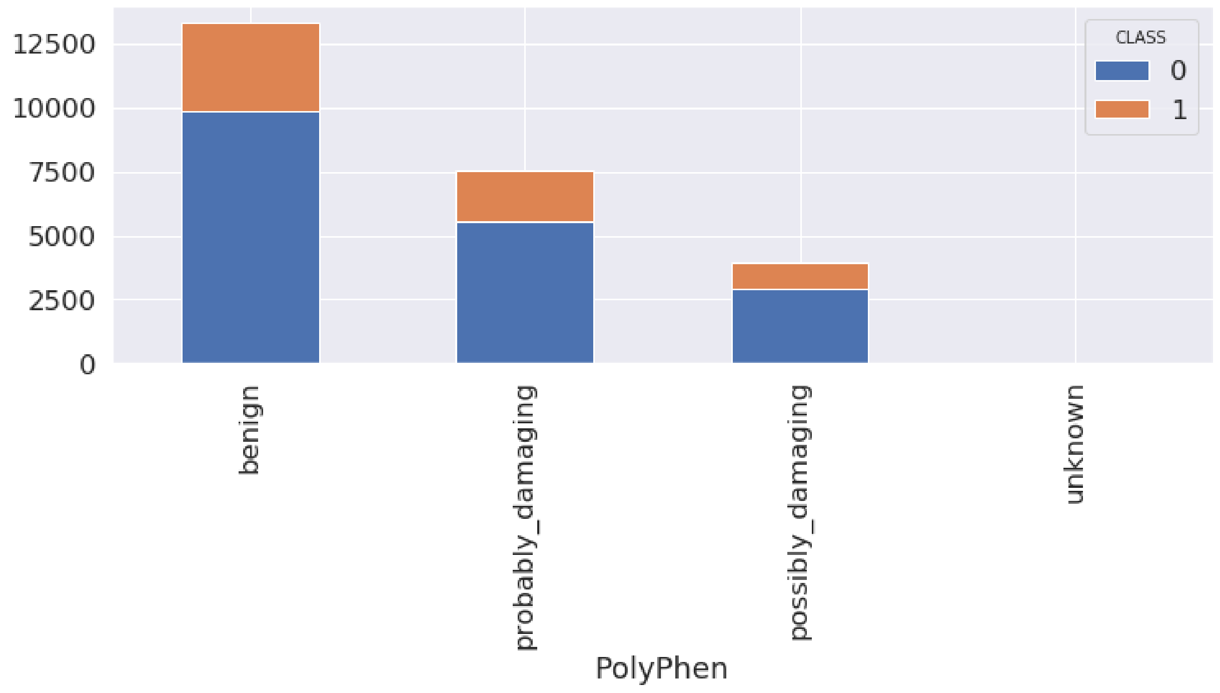


Results from `SIFT` and `PolyPhen` software that predict the severity of a variant, in-silico.

In [12]:
```
sift_ct = pd.crosstab(df.SIFT, df.CLASS, margins=True)
sift_ct.drop('All', axis=0, inplace=True)

# limit to the 50 most submitted genes for visualization
sift_ct = sift_ct.sort_values(by='All', ascending=False)
sift_ct.drop('All', axis=1, inplace=True)

sift_ct.plot.bar(stacked=True, figsize=(12, 4));
```



```
In [13]: pp_ct = pd.crosstab(df.PolyPhen, df.CLASS, margins=True)
         pp_ct.drop('All', axis=0, inplace=True)

         # limit to the 50 most submitted genes for visualization
         pp_ct = pp_ct.sort_values(by='All', ascending=False)
         pp_ct.drop('All', axis=1, inplace=True)

         pp_ct.plot.bar(stacked=True, figsize=(12, 4));
```



Encode `SIFT` and `PolyPhen`

```
In [14]: df = pd.get_dummies(df, columns=['SIFT', 'PolyPhen'])
```

## Correlation for categorical featuress by way of chi-square test

```
In [28]: from itertools import combinations
         from scipy.stats import chi2_contingency
```

```
In [29]: # select a few categorical features
         categoricals_index = pd.MultiIndex.from_tuples(combinations(['CHROM', 'REF'
         categoricals_corr = pd.DataFrame(categoricals_index, columns=['cols'])
```

```
In [30]: def chisq_of_df_cols(row):
             c1, c2 = row[0], row[1]
             groupsizes = df.groupby([c1, c2]).size()
             ctsum = groupsizes.unstack(c1)
             # fillna(0) is necessary to remove any NAs which will cause exceptions
             return chi2_contingency(ctsum.fillna(0))[1]
```

```
In [31]: categoricals_corr[ 'chi2_p'] =  categoricals_corr.cols.apply(chisq_of_df_co
```

In [32]:
```
categoricals_corr
```
Out[32]:

| | cols | chi2_p |
|---|---|---|
| **0** | (CHROM, REF) | 9.025884e-07 |
| **1** | (CHROM, ALT) | 8.443776e-01 |
| **2** | (CHROM, IMPACT) | 3.496264e-311 |
| **3** | (CHROM, Consequence) | 0.000000e+00 |
| **4** | (CHROM, SYMBOL) | 0.000000e+00 |
| **5** | (CHROM, CLASS) | 4.060535e-46 |
| **6** | (REF, ALT) | 0.000000e+00 |
| **7** | (REF, IMPACT) | 0.000000e+00 |
| **8** | (REF, Consequence) | 0.000000e+00 |
| **9** | (REF, SYMBOL) | 1.000000e+00 |
| **10** | (REF, CLASS) | 3.457498e-03 |
| **11** | (ALT, IMPACT) | 0.000000e+00 |
| **12** | (ALT, Consequence) | 0.000000e+00 |
| **13** | (ALT, SYMBOL) | 0.000000e+00 |
| **14** | (ALT, CLASS) | 3.671461e-02 |
| **15** | (IMPACT, Consequence) | 0.000000e+00 |
| **16** | (IMPACT, SYMBOL) | 0.000000e+00 |
| **17** | (IMPACT, CLASS) | 1.856664e-191 |
| **18** | (Consequence, SYMBOL) | 0.000000e+00 |
| **19** | (Consequence, CLASS) | 3.122902e-211 |
| **20** | (SYMBOL, CLASS) | 0.000000e+00 |

In [33]:
```
categoricals_corr.index = categoricals_index
categoricals_corr = categoricals_corr.chi2_p.unstack()
```

I trid plotting a heatmap with `-np.log(p)` but it didn't look good as a visualization.

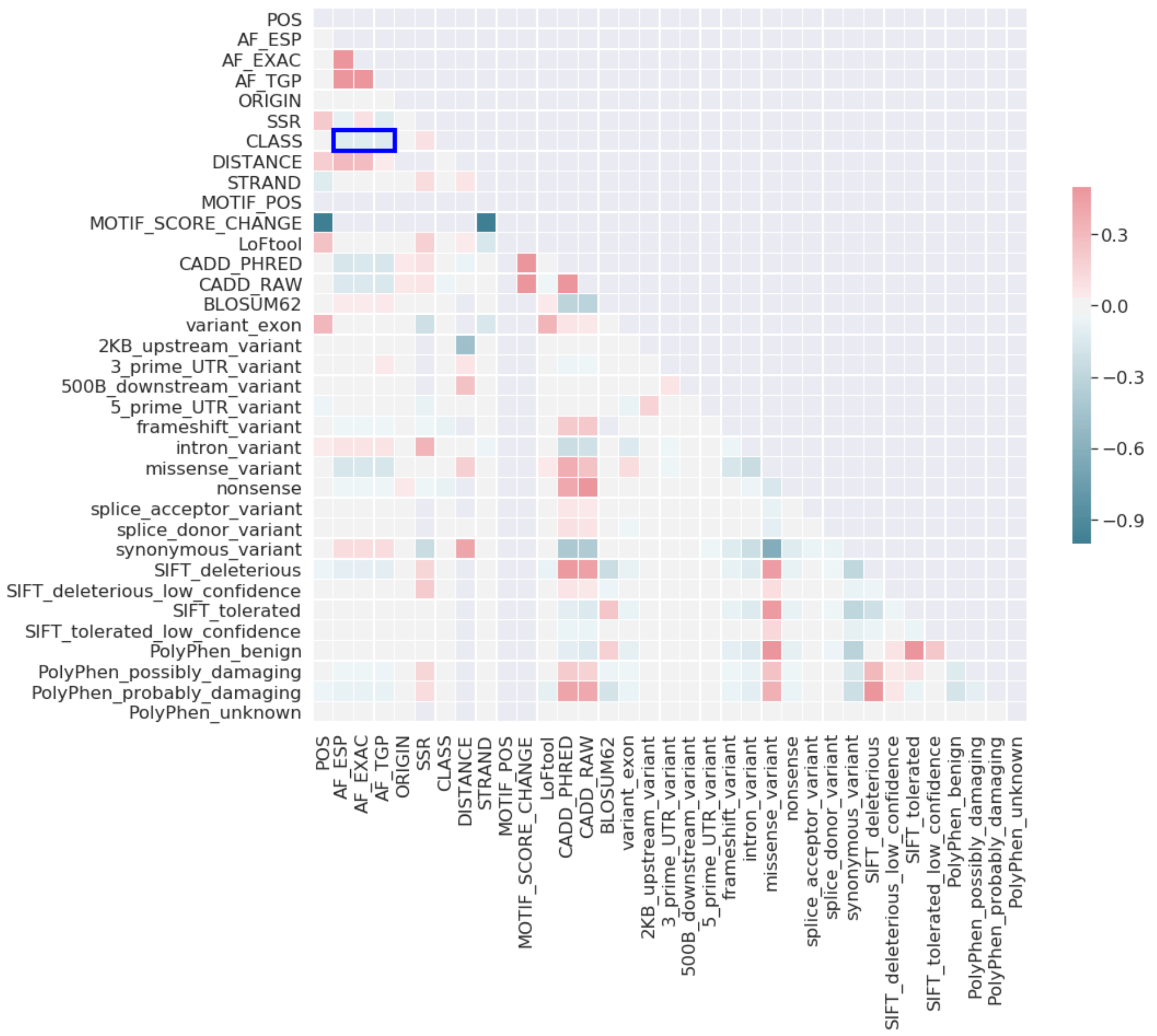
In [34]:
```
categoricals_corr
```

Out[34]:

| | ALT | CLASS | Consequence | IMPACT | REF | SYMBOL |
|---|---|---|---|---|---|---|
| **ALT** | NaN | 3.671461e-02 | 0.0 | 0.000000e+00 | NaN | 0.0 |
| **CHROM** | 0.844378 | 4.060535e-46 | 0.0 | 3.496264e-311 | 9.025884e-07 | 0.0 |
| **Consequence** | NaN | 3.122902e-211 | NaN | NaN | NaN | 0.0 |
| **IMPACT** | NaN | 1.856664e-191 | 0.0 | NaN | NaN | 0.0 |
| **REF** | 0.000000 | 3.457498e-03 | 0.0 | 0.000000e+00 | NaN | 1.0 |
| **SYMBOL** | NaN | 0.000000e+00 | NaN | NaN | NaN | NaN |

The dark blue box in in the heatmap highlights the negatvie correlation with the **allele frequency** features. Commomn alleles are less likely to pathogenic (cause disease), therefore most labs agree they should be benign.

In [35]:
```
corr = df.select_dtypes(exclude='object').corr()

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

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(15, 12));

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

# Draw the heatmap with the mask and correct aspect ratio
g = sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.5, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5});


from matplotlib.patches import Rectangle

g.add_patch(Rectangle((1, 6), 3, 1, fill=False, edgecolor='blue', lw=4));
```



[One of the ways variants can be classified is by the amount (and type) of sequence change (https://www.ebi.ac.uk/training/online/course/human-genetic-variation-i-introduction/what-genetic-variation/types-genetic-variation)](https://www.ebi.ac.uk/training/online/course/human-genetic-variation-i-introduction/what-genetic-variation/types-genetic-variation).
A substitution of a nucleotide (letter) is considered a single nucleotide variant (SNV), these are sometimes referred to as single nucleotide polymorphisms (SNP).
When one or more nucleotides are inserted or deleted the variant is considered an insertion or deletion. Therefore, if the length of `REF` or `ALT` is >1 then the variant can be considered an Insertion or Deletion (indel), otherwise it can be considered a SNV.

In [36]:
```
snvs = df.loc[(df.REF.str.len()==1) & (df.ALT.str.len()==1)]
indels = df.loc[(df.REF.str.len()>1) | (df.ALT.str.len()>1)]
```

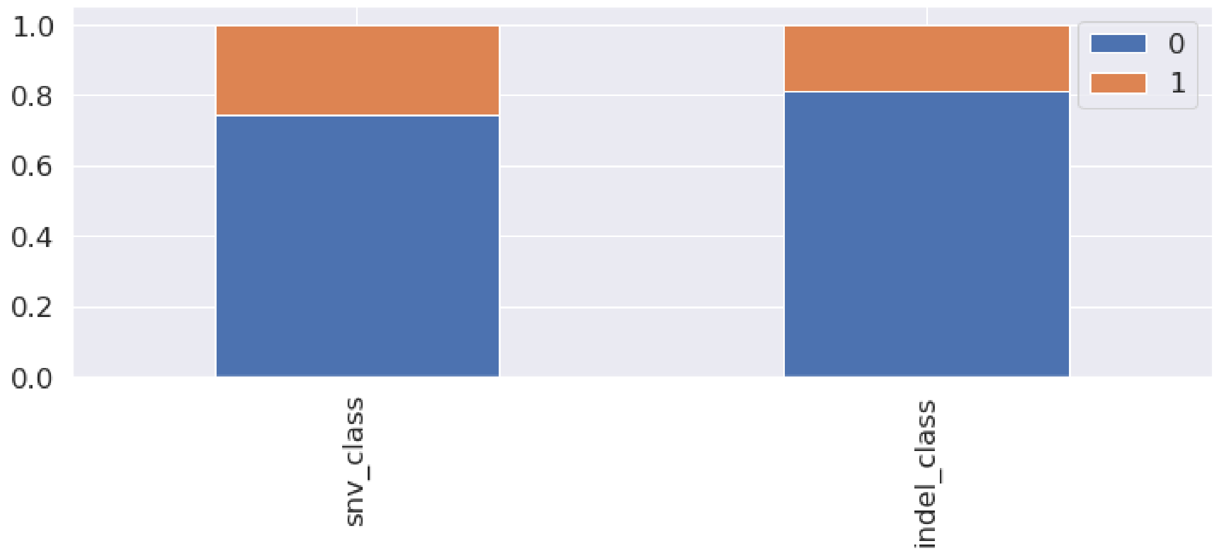
In [37]:
```
len(df) == (len(snvs) + len(indels))
```

Out[37]:
```
True
```

## SNVs are more likely to be conflicting than Indels

In [38]:
```
snp_indel = pd.concat([snvs.CLASS.value_counts(normalize=True).rename('snv_
                       indels.CLASS.value_counts(normalize=True).rename('in
                      axis=1).T
```

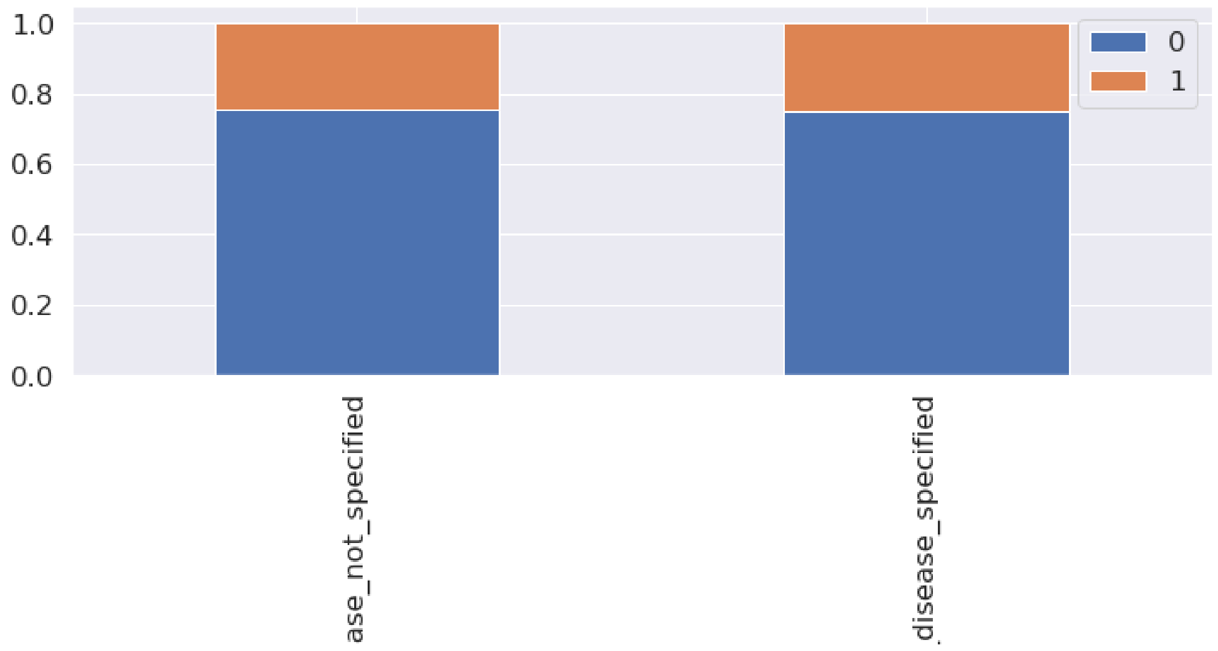
In [39]:
```
snp_indel.plot.bar(stacked=True, figsize=(12, 4));
```



CLNDN are lists of diseases associated with the variant. It may be beneficial to treat both `not_specified` and/or `not_provided` as the same category..

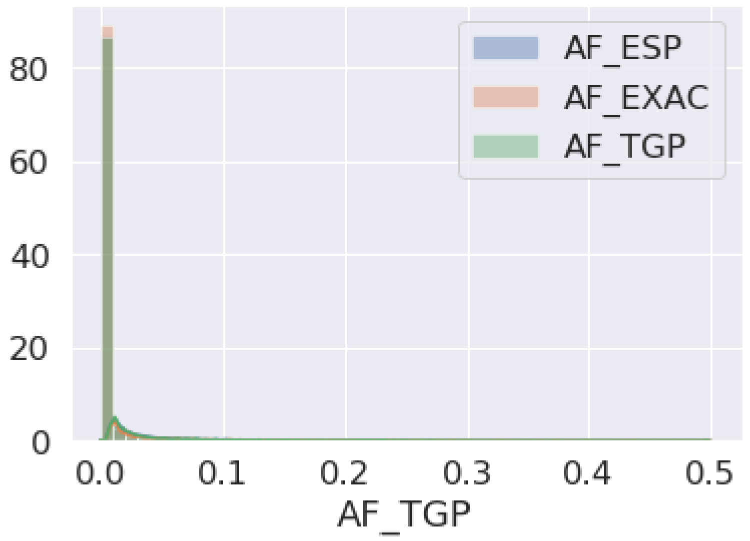
In [40]:
```
clndn = pd.concat([df.CLASS.loc[(df.CLNDN=='not_specified') | (df.CLNDN=='n
                   df.CLASS.loc[(df.CLNDN!='not_specified') | (df.CLNDN
                  axis=1).T
```

In [41]:
```
clndn.plot.bar(stacked=True, figsize=(12, 4));
```



**# most AF values are very low**

In [42]:
```
sns.distplot(df.AF_ESP, label="AF_ESP")
sns.distplot(df.AF_EXAC, label="AF_EXAC")
sns.distplot(df.AF_TGP, label="AF_TGP")
plt.legend();
```


AF_TGP
AF_ESP
AF_EXAC
AF_TGP