# イントロダクション

このコンペは「ポルトセグロ」というブラジル保険会社の第３位が主催したものだ。

この会社は来年ドライバーが保証金請求をはじめる確率を予想したい。 このノートブックではインタラクティブなチャートやplotライブラリを使ったpythonでの視覚化方法を駆使してデータ分析します。

そしていくつかの洞察と他人が真似したくなる美しいプロットを導きます。

plot.lyはplotly社が作った主要ソフトの一つです。 この会社はオンラインでの統計データの可視化ツールに特化していて、さらにpython,R,Matlab, node.jsなどの広範囲な言語に対するapiも用意しています。

便利なものをかんたんに紹介します。

- シンプルな水平バープロット　ターゲットの分布を調べる
- 相関係数のヒートマップ　異なる特徴間の相関係数を調べる
- 散布図 ランダムフォレストや勾配ブースティングモデルで得られた重要な変数を比べる
- 垂直バープロット　重要な特徴を降順にならべる
- 3D散布図

このノートブックのテーマはかんたんにまとめると下記になる。

- データの質をチェック 欠損値(null -1)を可視化したり、評価したり
- 特徴を調べらり、フィルタリング　相関係数や特徴間の相互情報を目的変数に対してプロットしてみる。　変数を２値、カテゴリー、それ以外か調べたり
- モデル学習を通して得られた重要な特徴をランキング　　特徴をランク付けするのにランダムフォレストや勾配ブースティングは便利で学習プロセスにもとづいています。

やってみましょう！

In [2]:

```
# 関連モジュールをロードしましょう
# conda install plotlyをお忘れなく
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import warnings
from collections import Counter
from sklearn.feature_selection import mutual_info_classif
warnings.filterwarnings('ignore')
```

pandasを使って与えられたデータを読み込みましょう

In [3]:

```
# csvデータを読み込んで先頭から5件表示
data_dir = './competitions/porto-seguro-safe-driver-prediction/'
train = pd.read_csv(data_dir + 'train.csv')
train.head()
```

Out[3]:

| | id | target | ps_ind_01 | ps_ind_02_cat | ps_ind_03 | ps_ind_04_cat | ps_ind_05_cat | ps_ind_06_bin |
|---|---|---|---|---|---|---|---|---|
| **0** | 7 | 0 | 2 | 2 | 5 | 1 | 0 | 0 |
| **1** | 9 | 0 | 1 | 1 | 7 | 0 | 0 | 0 |
| **2** | 13 | 0 | 5 | 4 | 9 | 1 | 0 | 0 |
| **3** | 16 | 0 | 0 | 1 | 2 | 0 | 0 | 1 |
| **4** | 17 | 0 | 0 | 2 | 0 | 1 | 0 | 1 |

5 rows × 59 columns

In [4]:

```
# データの大きさを見てみましょう
rows = train.shape[0]
columns = train.shape[1]
print("トレーニングデータセットは{}行{}列です。".format(rows, columns))
```

トレーニングデータセットは595212行59列です。

## 1.データの質を確認する

### nullや欠損値を確認しましょう

質チェックとして、トレーニングデータセットにnull値があるかどうかをさっと見ていきます。

In [5]:

```
# isnullとanyの動作確認
df = pd.DataFrame([[1, 2, 3],[np.nan,5, 6]])
```

In [6]:

```
# isnull()でNanかどうかをTrue Falseで返す
df.isnull()
```

Out[6]:

| | 0 | 1 | 2 |
|---|---|---|---|
| **0** | False | False | False |
| **1** | True | False | False |

In [7]:

```
# any()でTrueがある行はTrueを返す
df.isnull().any()
```

Out[7]:

```
0    True
1   False
2   False
dtype: bool
```

In [8]:

```
# 2回で全体としてNanがあるか確認できる
df.isnull().any().any()
```

Out[8]:

```
True
```

In [9]:

```
# any()2回で行、列
train.isnull().any().any()
```

Out[9]:

```
False
```

これよりNan値はない。 nullチェックはFalseだったけど、この結果が本当にこのケースでは-1を欠損値にしているためnull値がないと早とちりしたらダメだ。 だから-1のデータをnull値への置き換えを実施しましょう。 もう一度確認しよう。

-1が含まれているかみて、下記のようにして-1をnullにかんたんに置換できるよ。

In [10]:

```
train_copy = train
train_copy = train_copy.replace(-1, np.NaN) # train_copyは-1をnp.Nanに置き換える。
```

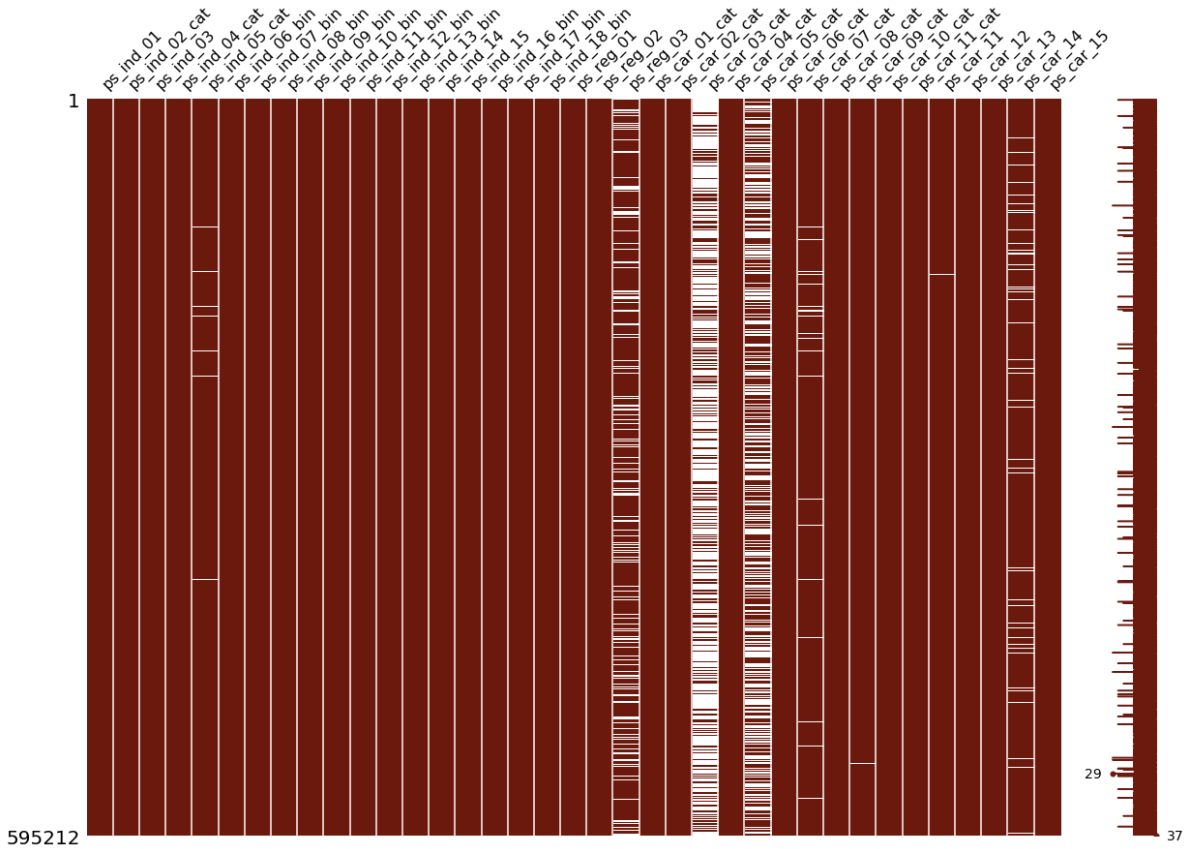
次にカグラーが作ったmissingnoというパッケージがデータセットの欠損値を見るのにめっちゃ便利だからやってみよう。

In [11]:

```
import missingno as msno
msno.matrix(df=train_copy.iloc[:, 2:39], figsize=(20, 14), color=(0.42, 0.1, 0.05))
# 欠損値の部分は白い線として表示される
```

Out[11]:

<matplotlib.axes._subplots.AxesSubplot at 0x1079a8128>




見ていると、白いところが欠損値で、縦に赤い帯になっていれば欠損値がないことがひと目で分かるね！ この場合、全体５９の内７つの項目で欠損値があることが分かります。だけどコメントでJustin Nafeが指摘しているようにホントは１３あるようです。 これはmissingnoの行列はおおよそ40奇数？の特徴を１度に表示して何個かは弾かれているかもしれない。だからのこりの5つのnull値を含んだものも取り除かれたんだ。 すべてのnull値を見るために、画面サイズ引数をかえて引き裂かれたデータを引っ張ってみましょう。 観測できた7つのnull値を含んだ項目を下に挙げておきましょう。

**ps_ind_05_cat | ps_reg_03 | ps_car_03_cat | ps_car_05_cat | ps_car_07_cat | ps_car_09_cat | ps_car_14**

null値のある項目の殆どの文字の後ろに_catがついています。s_reg_03, ps_car_03_cat,ps_car_05_catについて補足説明しておきましょう。 暗い帯の白い部分の比率をみるとわかりますが、この３つはnull値のほうがメジャーで-1に置換するという方法があまりいい戦略ではないかもしれません。

### 目的変数の調査

データを導く他の普通の方法は目的変数に関することです。この場合では項目に親切に'ターゲット'ってついていますね。目的変数は「分類/ラベル/正誤」などの答えで構成され、 まだ見ていないデータを上手く予測しできるような予測モデルを学習するために ターゲット変数を除いたトレーニングセットの変数を与えられた教師あり学習モデルが使われる。

In [12]:

```
# graphオブジェクトとして、目的変数とその数とタイトルを保存
data = [go.Bar(
            x = train["target"].value_counts().index.values,
            y = train['target'].value_counts().values,
            text = '目的変数の分布')]

# plotlyで描画する
layout = go.Layout(title='目的変数の分布')

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='basic-bar')
```

うーんデータがかなり偏っていることを心に留めておこう。 偏ったターゲットはすばらしいことだと後に分かるよ。

## データ型のチェック

このチェックはトレーニングデータセットの型を確認すること。 整数、文字、実数など。 重複なしのユニークな型を数えるのにいい方法がある。**Collections**モジュールをインポートしていれば、

In [13]:

```
# データ型の合計数や構成がひと目で分かる
Counter(train.dtypes.values)
```

Out[13]:

```
Counter({dtype('int64'): 49, dtype('float64'): 10})
```

上で分かる通り、全５９項目構成がわかり、int64とfloat64しかないことが分かります。

他に注意すべきこととして、ポルトセグロ社は項目に_binや_cat、_regをつけてくれて、_binは２値データ、catはカテゴリーデータ、それ以外と教えてくれる。 ここでさらに実数値（連続値をもつ特徴）と整数値(２値データ、カテゴリーデータ、ordinalな特徴に)とシンプルにしてみましょう。

In [14]:

```
# 実数型と整数型の項目をそれぞれ変数に格納
train_float = train.select_dtypes(include=['float64'])
train_int = train.select_dtypes(include=['int64'])
```

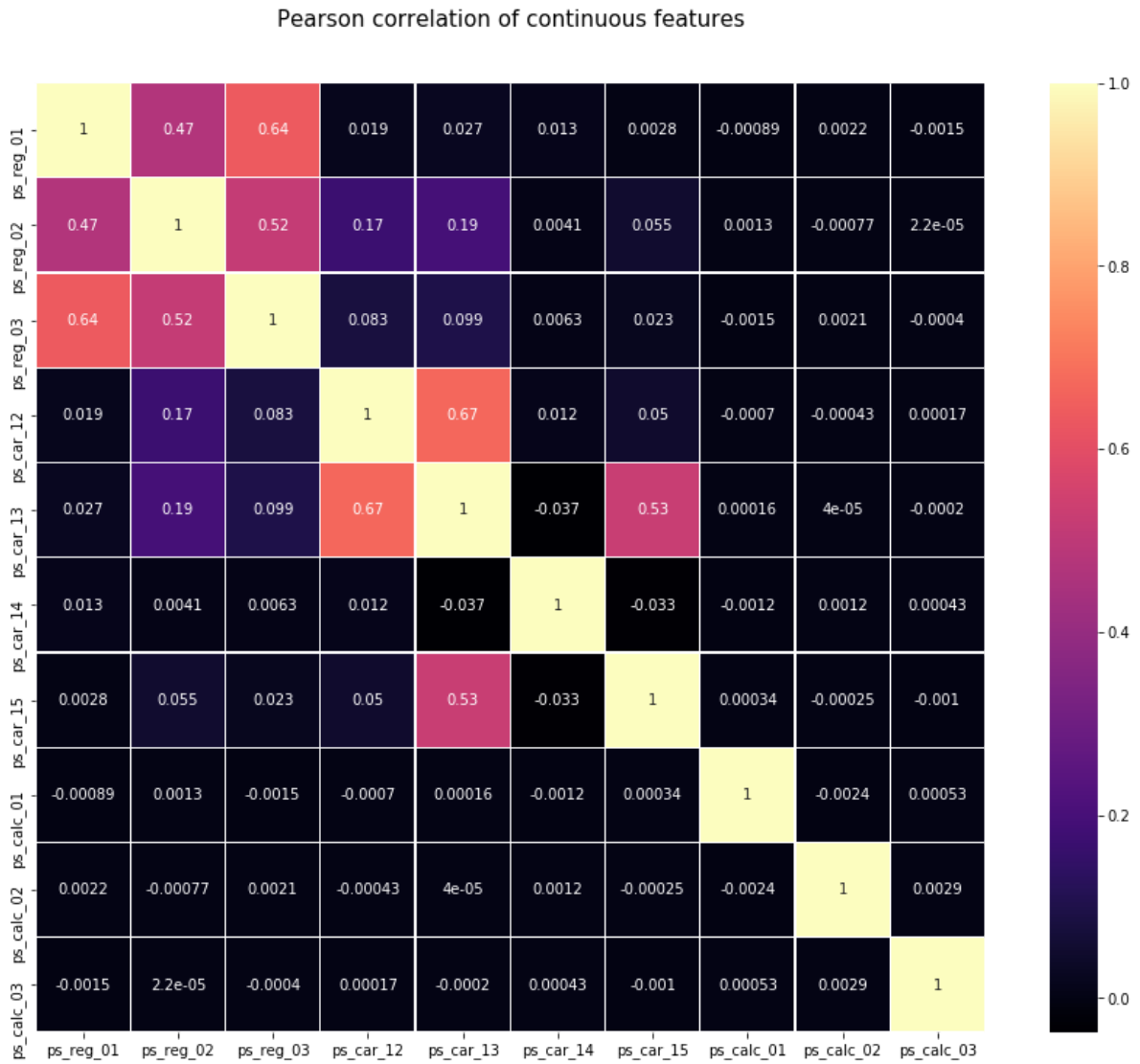
# 相関係数のプロット

手始めに、特徴が他のものとどう関連しているか、またもしかしたらそこから新しい気付きを得られるために相関係数をプロットしてみましょう。 この際、seabornをつかって相関係数をヒットマップでぷろっとします。 都合のいいことに、pandasのデータフレームにはcorr()関数が備わっているので、ピアソン相互相関係数を計算できます。またseabornの相関プロットも便利です。 文字通り"heatmap"で使えます。

**実数値の特徴の相関係数**

In [15]:

```
# カラーマップの設定 マグマ
colormap = plt.cm.magma
plt.figure(figsize=(16, 12))
plt.title('Pearson correlation of continuous features', y=1.05, size=15)
sns.heatmap(train_float.corr(), linewidths=0.1, vmax=1.0, square=True,
        cmap=colormap, linecolor='white', annot=True);
# linecolor= マス目の線を白色に、annot=True数値を表示
```




相関係数プロットから、だいたいが０で他と無相関だと分かります。 これはかなり興味深い観測結果で、これ以降のさらなる調査をwarrantします。 さしあたって、正の相関があるペアは下記だ

**(ps_reg_01, ps_reg_03)**

**(ps_reg_02, ps_reg_03)**

**(ps_car_12, ps_car_13)**

**(ps_car_13, ps_car_15)** ※1,2はだめ？

**整数値の特徴の相関関係**

整数型の項目に対して、Plotlyに切り替えて相関係数のヒートマップを表示してみましょう。 前のplotlyプロットほとんど同じように、ヒートマップを作るために、go.Heatmapを呼び出してみましょう。ここで３つの軸に値をセットする必要があり、x,yに項目名、z軸に相関係数値を。 カラースケール属性は決められたワードを入れてね。 （今回はGreysを使うけど、自分でPortlandやViridisも試してみて）

In [16]:

```
# Seabornでやるとすればこうかな?
#train_int = train_int.drop(["id", "target"], axis=1)
# colormap = plt.cm.bone
# plt.figure(figsize=(21,16))
# plt.title('Pearson correlation of categorical features', y=1.05, size=15)
# sns.heatmap(train_cat.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor=

# plotly流のheatmapの表示方法
#ピポットしなくても3変数を指定するだけで表示できる
#まマウスカーソルを合わせると情報を表示してくれる
#使い方は深入りしないが下記のサイトは参考になりそう。
# https://images.plot.ly/plotly-documentation/images/python_cheat_sheet.pdf
# https://qiita.com/ogrew/items/e6ba517cec9fb5161cb6
# https://qiita.com/inoory/items/12028af62018bf367722
data = [
    go.Heatmap(
        z = train_int.corr().values,
        x = train_int.columns.values,
        y = train_int.columns.values,
        colorscale = 'Viridis',
        reversescale = False,
        text = True,
        opacity = 1.0)
]

layout = go.Layout(
    title='Peason Correlation  of Integer-type features',
    xaxis = dict(ticks='', nticks=26),
    yaxis = dict(ticks=''),
    width = 900, height = 700)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='labelled-heatmap')
# 相関のあるところにカーソルをもっていくと項目情報が表示されるので分かりやすい
```

同様に、ほとんどが線形相関がないのがわかって、かなり多くが0だと明らかになったね。 これはかなり有益で、とくにもし主成分分析などはっきりと相関関係を必要とするような次元圧縮をやろうとするときにね。

***負の相関関係の特徴***

ps_ind_06_bin, ps_ind_07_bin, ps_ind_08_bin, ps_ind_09_bin

特筆すべき点は最初のヌル値の分析のときにps*car*-3_catとps_car_05_catは多くの欠損値かヌル値を含んでいたことです。 それゆえこれらの特徴が強い正の相関があっても不思議はない。 たとえ一方が裏に潜んだ真実を反映していないかもしれない。

# 相互情報プロット

相互情報は対応する特徴やターゲット変数間の相互情報を調べる別の有益なツールです。

分類問題に対して、sklearnのmutual_info_classifを呼んでサクッと依存関係を調べられます。 （0が無関係で1にいくほど強い相関です） それゆえこれは特徴内に含まれているかもしれない目的変数との多くの情報へのアイディアになります。

mutual_infi_classif関数のsklearn実装はK近傍法から推測されたエントロピーに基づいたノンパラメトリックな方法になっています。 くわしくは公式ホームページを見てね。

In [17]:

```
# 相互情報量を表示する。
mf = mutual_info_classif(train_float.values, train.target.values, n_neighbors=3, random_state=17)
print(mf)
```

```
[0.01402035 0.00431986 0.0055185  0.00778454 0.00157233 0.00197537
 0.01226    0.00553038 0.00545101 0.00562139]
```

# ２値特徴の調査

ほかに調査したいかもしれないデータの観点は２値しか取らない項目（たとえば0,1のどちらか）だろう。 なりゆきでちょうど２値データをすべて保管しているから、垂直なplotlyバープロットをやってみよう。

In [18]:

```
bin_col = [col for col in train.columns if '_bin' in col] # リスト内包表記 _binのつく項目だけをbin_colに格納
zero_list = []
one_list = []
# 各項目ごとに0,1の数を集計
for col in bin_col:
    zero_list.append((train[col]==0).sum())
    one_list.append((train[col]==1).sum())
```

In [19]:

```
# 2値を視覚的にわかりやすく表示する
trace1 = go.Bar(
    x=bin_col,
    y=zero_list,
    name='Zero count'
)
trace2 = go.Bar(
    x=bin_col,
    y=one_list,
    name='One count'
)

data = [trace1, trace2]
layout = go.Layout(
    barmode='stack',
    title='Count of 1 and 0 in binary variables'
)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='stacked-bar')
# すごく分かりやすい!!
```

ここで４つの特徴がみられる。 ps_ind_10_bin,ps_ind_11_bin,ps_ind_12_bin,ps_ind_13_binはほぼゼロしかないね。 この情報はこれらの特徴が有用なのかターゲット変数に対して大した情報がふくまれていないかという疑問を招きます。

# カテゴリーや順序のある特徴の調査

まず後ろに_catがついた特徴を見ていきましょう。

# ランダムフォレストで得る重要な変数

まずランダムフォレストを使ってみて、学習が終わったあとの特徴のランニングを見てみましょう。 これはアンサンブル学習をつかった手早い方法です。 （集計されたブートストラップの下で弱意思決定木学習者のアンサンブル） 有用な特徴を見つけるのにたいへんなパラメータチューニングは必要ないし、目的変数の不均衡さに対しても堅牢です。 ランダムフォレストを呼んでみよう

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=150, max_depth=8, min_samples_leaf=4, max_features
                            n_jobs=-1, random_state=0)
rf.fit(train.drop(['id', 'target'], axis=1), train.target)
features = train.drop(['id', 'target'], axis=1).columns.values
print("----- Training Done -----")
```

----- Training Done -----

### Plot.lyの特徴の重要度の散布図

ランダムフォレストで学習して、feature*importances*属性を呼んで特徴の重要度のリストを得て、 Plotlyの散布図でプロットしましょう。

Scatterコマンドを呼んで、前のPlotlyプロットごとに、xy軸を呼び出さなければならない。 しかしながら散布図で注意すべきことははマーカー属性です。 定義したマーカー属性はプロット点の大きさ、色をコントロールできます。

In [22]:

```
# 散布図
trace = go.Scatter(
    y = rf.feature_importances_,
    x = features,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref =1,
        size = 13,
        # size = rf.feature_importances_,
        #color = np.random.randn(500), # set color equal to a variable
        color = rf.feature_importances_,
        colorscale='Portland',
        showscale=True
    ),
    text = features
)
data = [trace]

layout = go.Layout(
    autosize = True,
    title = 'Random Forest Feature Importance',
    hovermode='closest',
    xaxis= dict(
        ticklen=5,
        showgrid=False,
        zeroline=False,
        showline=False
    ),
    yaxis=dict(
        title = 'Freature Importantce',
        showgrid=False,
        zeroline=False,
        ticklen=5,
        gridwidth=2
    ),
    showlegend = False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='scatter2010')
```

さらに重要さの順番で並べ替えて表示できます。 下のように高いものから低い順でバープロットしてみます。

In [23]:

```
# 特徴を重要度でソートしてリストに格納する
x, y = (list(x) for x in zip(*sorted(zip(rf.feature_importances_, features), reverse=False)))

# 横のバープロット
trace2 = go.Bar(
    x=x,
    y=y,
    marker=dict(
        color=x,
        colorscale = 'Viridis',
        reversescale = True
    ),
    name='Random Forest Feature importance',
    orientation='h', # これで縦のバープロットにする
)
layout = dict(
    title='Barplot of Feature importances',
    width = 900, height = 2000,
    yaxis=dict(
        showgrid=False,
        showline=False,
        showticklabels=True,
        # domain=[0, 0.85],
    ))

fig1 = go.Figure(data=[trace2])
fig1['layout'].update(layout)
py.iplot(fig1, filename='plots')
```

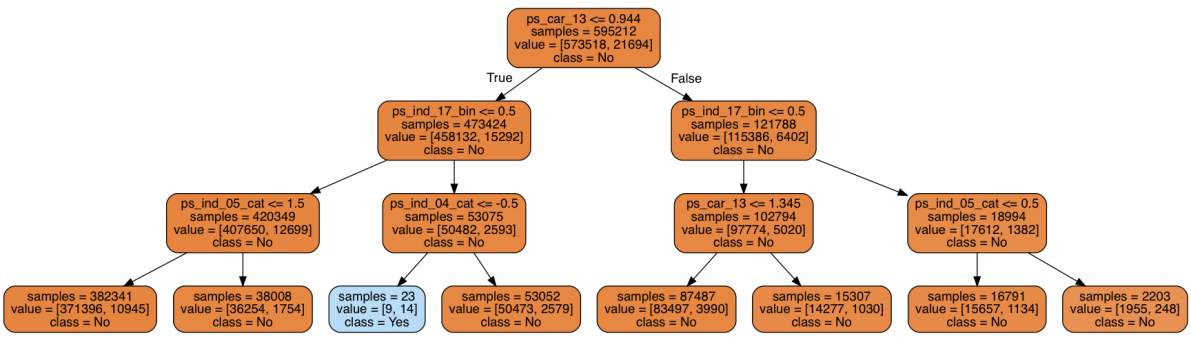
**決定木の可視化**

面白いテクニックとして決定木の枝を可視化することがよくあります。 簡素化のために、深さ３で学習したので３レベルの深さの枝までしか見れませんし、sklearnのexport_graphvisのグラフ可視化属性をエクスポートしたり、このノートブックで見せたイメージをインポートしたりエクスポートできます。

In [34]:

```
from sklearn import tree
from IPython.display import Image as PImage
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont
import re # 正規表現

decision_tree = tree.DecisionTreeClassifier(max_depth = 3)
decision_tree.fit(train.drop(['id', 'target'], axis=1), train.target)

# 学習モデルを.dotファイルにエクスポート
with open("tree1.dot", 'w') as f:
    f = tree.export_graphviz(decision_tree,
                             out_file=f,
                             max_depth = 4,
                             impurity = False,
                             feature_names = train.drop(['id', 'target'], axis=1).columns.values,
                             class_names = ['No', 'Yes'],
                             rounded = True,
                             filled = True)

# .dotを.pngファイルに変換
# brew install graphvisする必要あり
# https://qiita.com/umasoya/items/c1c4e583f393b27ebb7a
check_call(['dot', '-Tpng', 'tree1.dot', '-o', 'tree1.png'])

# Annotating chart with PIL
img = Image.open("tree1.png")
draw = ImageDraw.Draw(img)
img.save('sample-out.png')
PImage("sample-out.png", )
# カレントディレクトリに.dotと.pngファイルが生成されることを確認
```

Out[34]:




# 勾配ブースティングを使った特徴の重要度

好奇心のために、別のモデルで学習してみましょう。 ここでは勾配ブースティングの分類器を使います。 勾配ブースティングは段階を踏んで進み、それぞれの段階で回帰木は損失関数の勾配が与えられます。（sklearn実装では デビアンスがデフォルトです。）

In [24]:

```
from sklearn.ensemble import GradientBoostingClassifier
# 勾配ブースティング
gb = GradientBoostingClassifier(n_estimators=100, max_depth=3, min_samples_leaf=4,
                                max_features=0.2, random_state=0)
# モデル学習
gb.fit(train.drop(['id', 'target'], axis=1), train.target)
# idと目的変数を除いた項目名を特徴として保持
features = train.drop(['id', 'target'], axis=1).columns.values
print('------Training Done --------')
```

```
------Training Done --------
```

In [25]:

```
# 特徴とその重要度を散布図としてプロットする
trace = go.Scatter(
    y = gb.feature_importances_,
    x = features,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 13,
        # size = rf.feature_importances_,
        # color = np.random.randn(500), # set color equal to a variable
        color = gb.feature_importances_,
        colorscale='Portland',
        showscale=True
    ),
    text = features
)

data = [trace]

layout = go.Layout(
    autosize = True,
    title='Gradient Boosting Machine Feature Importance',
    hovermode='closest',
    xaxis=dict(
        ticklen =5,
        showgrid=False,
        zeroline=False,
        showline=False
    ),
    yaxis=dict(
        title ='Freature Importance',
        showgrid = False,
        zeroline=False,
        ticklen=5,
        gridwidth=2
    ),
    showlegend=False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='scatter2010')
```

In [26]:

```
# さきほどと同じように重要度順にバープロットしてみる。
x, y = (list(x) for x in zip(*sorted(zip(gb.feature_importances_, features),
                             reverse = False)))
trace2 = go.Bar(
    x=x,
    y=y,
    marker=dict(
        color=x,
        colorscale = 'Viridis',
        reversescale = True
    ),
    name="Gradient Boosting Classifer Feature importance",
    orientation='h',
)

layout =dict(
    title='Barplot of Feature importances',
    width = 900, height =2000,
    yaxis = dict(
        showgrid=False,
        showline=False,
        showticklabels=True,
    ))
fig1 = go.Figure(data=[trace2])
fig1['layout'].update(layout)
py.iplot(fig1, filename='plots')
```

興味深いことに、ランダムフォレストと勾配ブースティングどちらも最も重要な特徴は同じで**ps_car_13**のようですね。

この特徴はかなりさらなる調査が必要なのでより深く調べることを検討してください。

# 結論

これまで広範囲に渡る調査をやってきました。ヌル値とデータの質の調査、特徴間の相関関係を調べ、ラ特徴の分布の幾つかを調べ、ンダムフォレストと勾配ブースティングという２つの学習モデルを行ってみた。そしてモデルが重要だと示す特徴を特定できた！

# *続く..*