前書き

このコンペは、ブラジルの第3位の保険会社（Porto Seguro）（https://en.wikipedia.org/wiki/Porto_Seguro_S.A）(https://en.wikipedia.org/wiki/Porto_Seguro_S.A）) が主催し、運転手が次の年に保険請求 を開始する確率を予測するものです*

このノートブックは、Pythonビジュアライゼーションライブラリ Plot.lyを使っていくつかのインタラクティブなチャートと競合データの分析を提供し、うまくいけば、他の人が取って複製することができるいくつかの洞察と美しいプロットをもたらすことを目指します。 Plot.lyは、ソフトウェア企業[Plotly]（https://plot.ly/）(https://plot.ly/）) が提供する主要製品の1つです。その会社はPython、R、Matlab、Node.jsなどの豊富なプログラミング言語およびツール群にAPIを提供するだけでなく、オンラインのグラフィカルおよび統計的な視覚化（チャートおよびダッシュボード）の提供に特化しています。

簡便性のために、このノートブックのさまざまなPlotlyプロットへのリンクを以下にリストしています：

- 単純な横棒グラフ - ターゲット変数の分布を検査するために使用されます
- 相関ヒートマッププロット - 異なる特徴間の相関関係を検査する
- 散布図 - ランダムフォレストとグラデーションブーストモデルで生成された機能の重要度を比較します。
- 縦棒グラフ - 下降順の一覧、さまざまな機能の重要性
- 3Dスキャッタプロット

このノートブックのテーマは以下のように簡単に要約できます：

1。データ品質チェック（＃品質） - すべての欠損値/ヌル値（-1の値）の可視化と評価

2。機能検査とフィルタリング - ターゲット変数に対する相関とフィーチャ相互情報プロット。バイナリ、カテゴリおよびその他の変数の検査。

3。学習モデルによる特徴重要度ランキング / nランダムフォレストとグラデーションブーストモデルを構築し、学習プロセスに基づいて機能をランク付けするのに役立ちます。

始めましょう

下記のカーネルは各種ライブラリーをインポートしている。

In [1]:

```
# 関連するPythonモジュールにロードしましょう
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を使用して提供されたトレーニングデータを読み込みましょう。

In [2]:

```
train = pd.read_csv("../input/train.csv")
train.head()
```

Out[2]:

	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind
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 [3]:

```
# trainデータセットに含まれる行と列の数を調べる
rows = train.shape[0]
columns = train.shape[1]
print("The train dataset contains {0} rows and {1} columns".format(rows, columns))
```

The train dataset contains 595212 rows and 59 columns

1. データ品質チェク

null値または欠損値のチェック

品質検査の一環として、次のようにtrainデータセットにヌル値があるかどうかを素早く見てみましょう。

In [4]:

```
# any()を2回適用してチェックすると、すべての列でisnullチェックが実行されます。
train.isnull().any().any()
```

Out[4]:

False

*null値チェックではFalseがリターンされますが、データが[-1の値は観測から欠落していることを示します]と記述されているため、このケースは閉じられているという意味ではありません。
（https：//www.kaggle.com/c/porto-seguro-safe-driver-prediction/data）。 したがって、ポルト・セグロは、データのすべてのnull値を-1の値でブランケット置換しただけです。 データの欠損値がどこにあるのか調べてみましょう。

ここで、どの列が値に-1を含んでいるかわかるので、たとえば、次のように、すべて-1のnullを最初にブランケット置換することができます。

下記のカーネルはtrain.csvの-1の値をNaNに置き換えています。

replace():https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.replace.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.replace.html)

In [5]:

```
train_copy = train
train_copy = train_copy.replace(-1, np.NaN)
```

次に、常駐Kagglerの[Aleksey Bilogur]が作成したデータセット内の欠損値を視覚化する上で最も有用で便利なツールである「Missingno」パッケージを使用することができます。 チェックしてください。

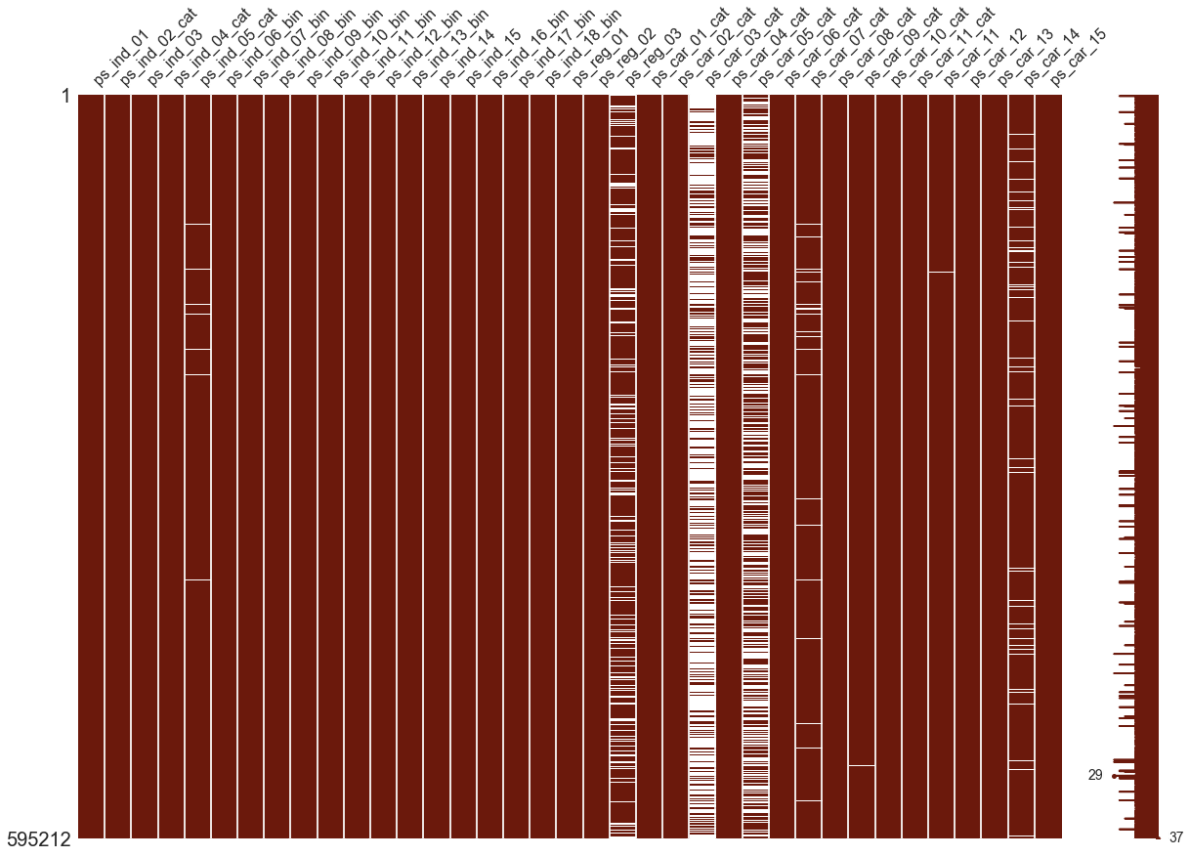
下記のカーネルはmissingno.matrixを使い各データにNULL値または欠損値がないか視覚的に表示しています。

In [6]:

```
import missingno as msno
#列ごとのNULL値または欠損値
msno.matrix(df=train_copy.iloc[:,2:39], figsize=(20, 14), color=(0.42, 0.1, 0.05))
```

Out[6]:

<matplotlib.axes._subplots.AxesSubplot at 0x105850c18>



見てわかるように、欠落している値は、視覚化すると明らかになります。空白の白いバンド（欠落しているデータ）が縦の濃い赤いバンド（重複していないデータ）に重なっていると、特定の列のデータの無効を反映します。この例では、59の全ての機能のうち7つの機能が実際にはnull値を含んでいることがわかります（コメント欄にJustin Nafeが正しく指摘しているように、実際には欠損値を持つ列が合計13あります）。これは、missingno matrix plot が約40個の奇妙な機能を1つのプロットに収めた後、おそらく一部の列が削除されること

によってのみ快適にフィットできるという事実によるもので、それゆえ残りの5つのnullカラムは除外されています。 すべてのnullを視覚化するには、データフレームをスライスする方法を調整するだけでなく、figsizeの引数を変更してみてください。

私たちが観察できる7つのnull列については、以下のようにリストされています。

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

欠損値の大部分は、_catで終わる列に現れます。実際にps_reg_03、ps_car_03_cat、ps_car_05_catという列にさらに注意する必要があります。白から暗のバンドの比率から明らかなように、これらの3つの列から値の大多数が欠落していることは非常に明白であり、したがって、ヌル場合の-1のブランケット置換はあまり良い戦略ではないかもしれません。

ターゲット変数の検査

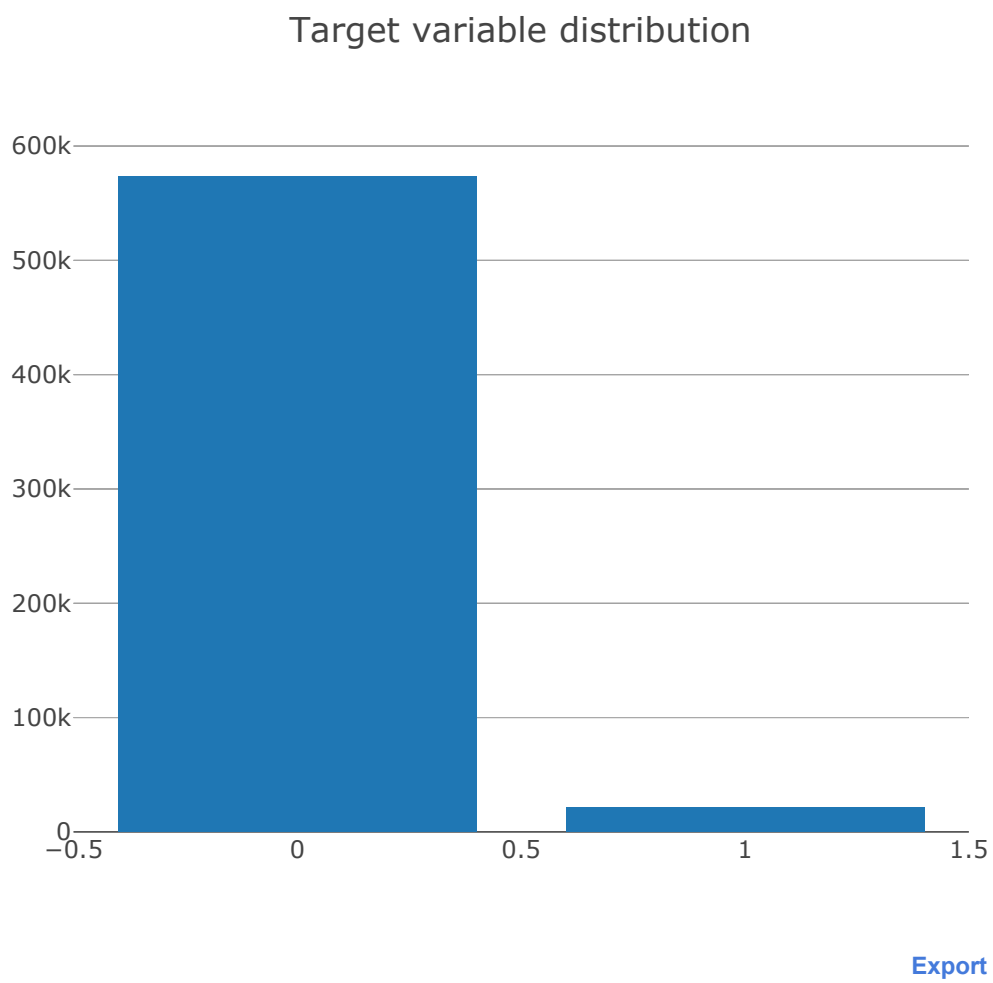
データ上で通常行われるもう1つの標準チェックは、この場合、カラムが便宜的に「target」と題されたターゲット変数に関するものです。 target値は、class/label/correctのあだ名でも与えられ、関数を学習するために与えられた（この例ではid列以外のすべてのtrainデータ）対応するデータとともに教師あり学習で使用されています。それは学習した関数が一般化でき、新しい目に見えないデータでうまく予測することを期待して、ターゲットにデータを最もよくマッピングする関数です。

下記のカーネルはの"target"のユニークな要素０と１をx軸にとり、y軸にそれぞれの総数をとりヒストグラムで表示しています。

In [7]:

```
print(train["target"].value_counts().values)

data = [go.Bar(
        x = train["target"].value_counts().index.values,
        y = train["target"].value_counts().values,
        text='Distribution of target variable'
    )]

layout = go.Layout(
    title='Target variable distribution'
)

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

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

[573518 21694]



うーん、target変数はむしろ不均衡なので、心に留めておくべきことかもしれません。 不均衡な目標は、かなり証明するでしょう。

データ型チェック

このチェックは、trainセットがどのような種類のデータ型で構成されているかを確認するために実行されます。整数値または文字列または浮動小数点数は、提供されたデータの概要を把握するためのものです。 Pythonシーケンスで一意の型の数を取得する1つのトリックは、Counterメソッドを使用することです。

Collections モジュールを次のようにインポートすると、

下記のカーネルはデータセットのデータタイプを算出し辞書型で返している。

Counter():https://docs.python.jp/3/library/collections.html#collections.Counter
(https://docs.python.jp/3/library/collections.html#collections.Counter)

In [8]:

```
Counter(train.dtypes.values)
```

Out[8]:

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

上記のようにtrainデータセットを構成する列は合計59個あり、このチェックからわかるように、フィーチャ/列は整数と浮動小数点の2つのデータ型のみで構成されています。

Porto Seguroは実際には、_bin、_cat、_regのような略語が末尾に付いたヘッダーをデータとして提供しています。_catはバイナリ機能を示し、_catはカテゴリー機能を示し、一方で残りの部分は連続的または序数的な特徴のいずれかであるという大まかな説明を与えています。ここでは、浮動小数点値（たぶん連続機能のみ）と整数データ型（バイナリ、カテゴリ、および序数機能）を見るだけでこれをさらに単純化します。

下記のカーネルはtrain_floatとtrain_intにそれぞれ「float64」「int64」をデータ型に持つカラムのリストを格納している。

select_dtypes(): データtypeに基づいて列をinclude/excludeするDataFrameのサブセットを返します。
https://pandas.pydata.org/pandasdocs/stable/generated/pandas.DataFrame.select_dtypes.html
(https://pandas.pydata.org/pandasdocs/stable/generated/pandas.DataFrame.select_dtypes.html)

In [9]:

```
train_float = train.select_dtypes(include=['float64'])
train_int = train.select_dtypes(include=['int64'])
```

相関プロット

まず、ある特徴が次の特徴とどの程度線型的に相関なのかを素早く見て、ここからいくつかの洞察を得ることを開始するために、線形相関プロットを生成してみましょう。 ここでは、シーボーンの統計的視覚化パッケージを使用して、相関係数のヒートマップをプロットします。 好都合なことに、Pandasのデータフレームには、ピアソン相関を計算するcorr()メソッドが組み込まれています。 また、シーボーンの相関プロットの呼び出し方法も便利です。 ちょうど"ヒートマップ"の文字通りに。

浮動小数点機能の相関

下記のカーネルはtrainの浮動小数点を持つデータ同士の相関係数を出してヒートマップで表示している。

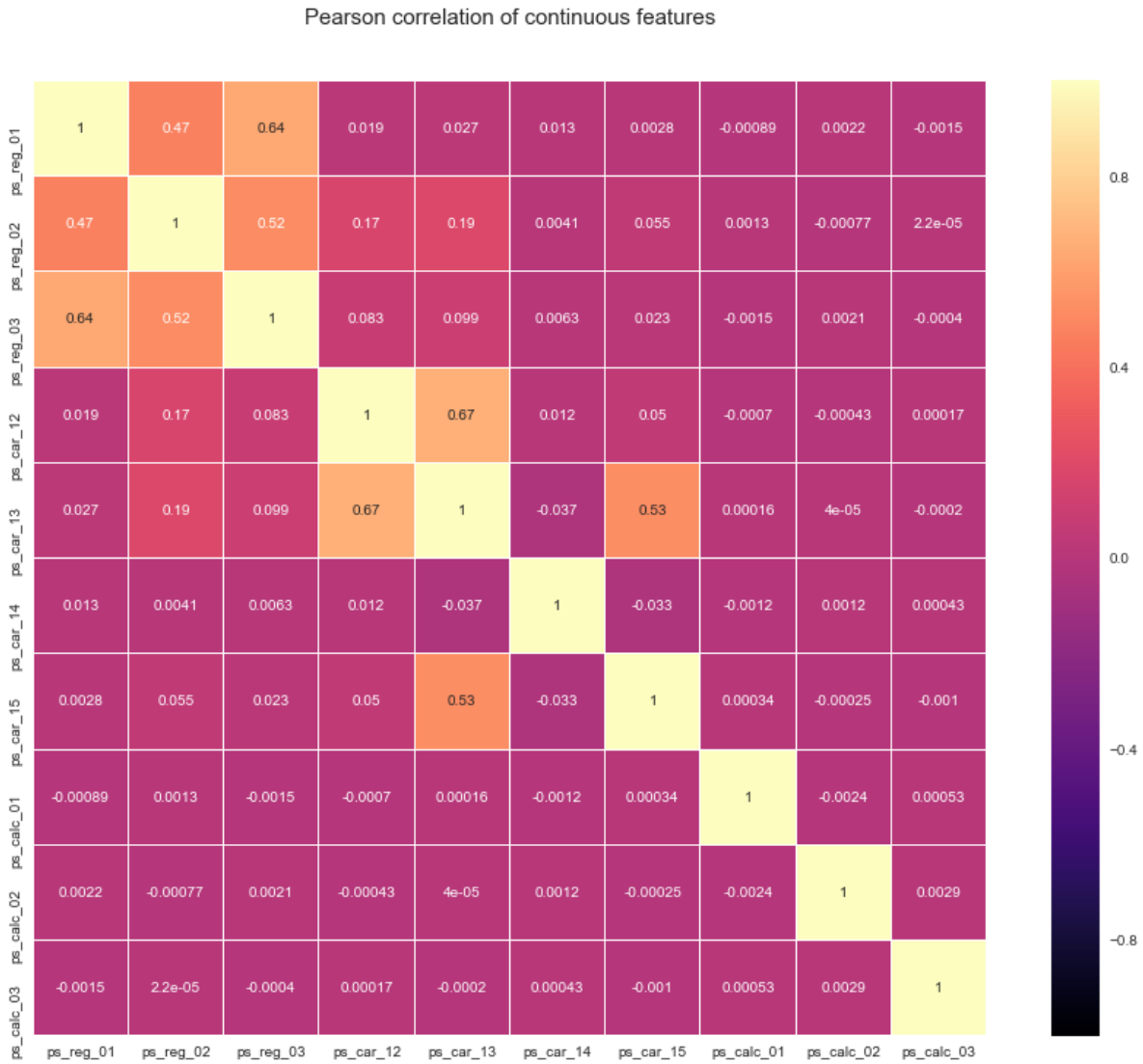
plt.cm.magma：カラーマップをmagmaに設定

In [10]:

```
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)
```

Out[10]:

<matplotlib.axes._subplots.AxesSubplot at 0x11cb7cbe0>



相関プロットから、大部分のは互いにゼロまたは非相関を表示することがわかります。 これは私たちのさらなる調査を後で保証することになる、非常に興味深い観察です。 現時点では、正の線形相関を示す対のフィーチャが次のようにリストされています。

（ps_reg_01、ps_reg_03）

（ps_reg_02、ps_reg_03）

（ps_car_12、ps_car_13）

（ps_car_13、ps_car_15）

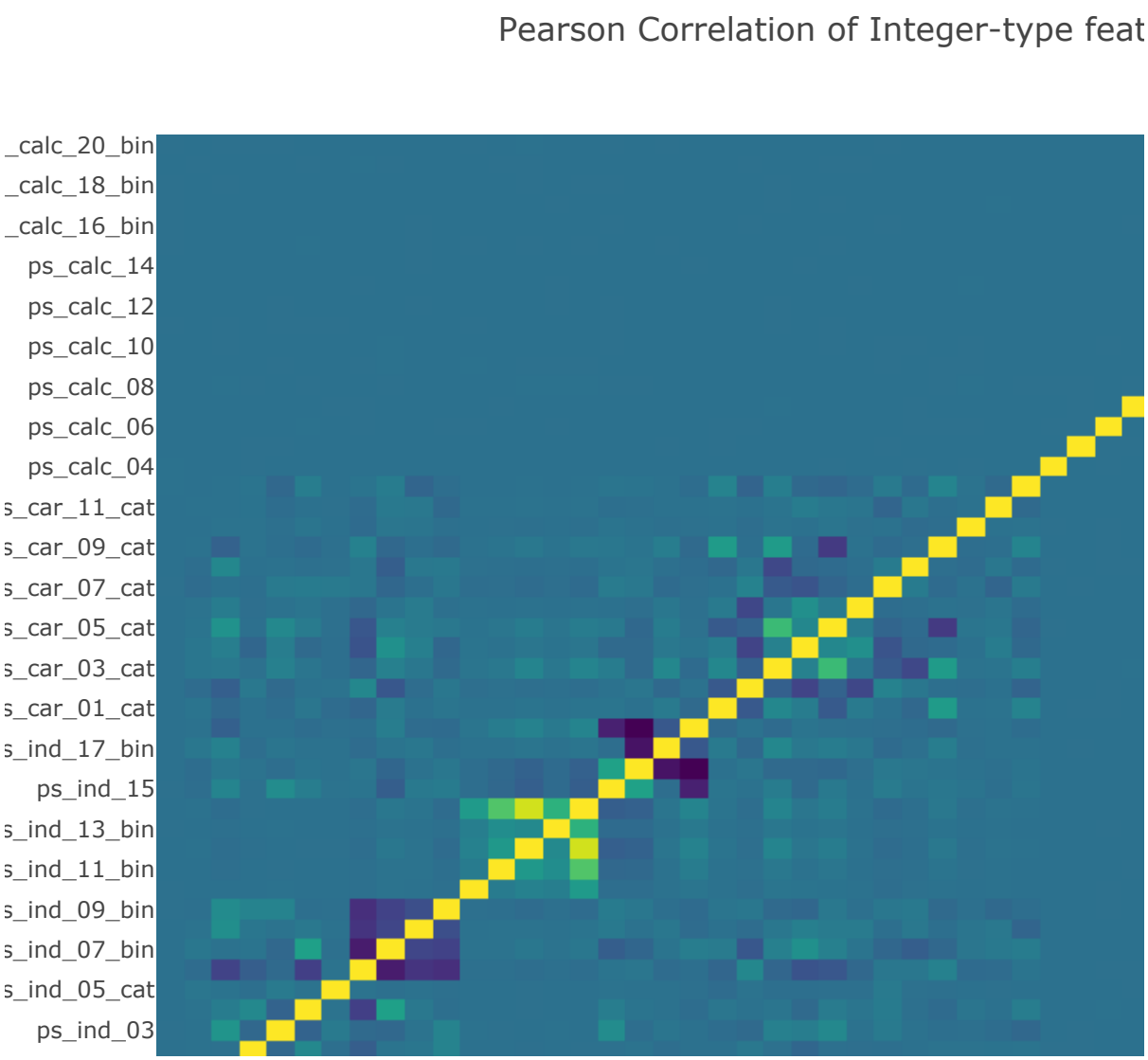
整数特徴の相関

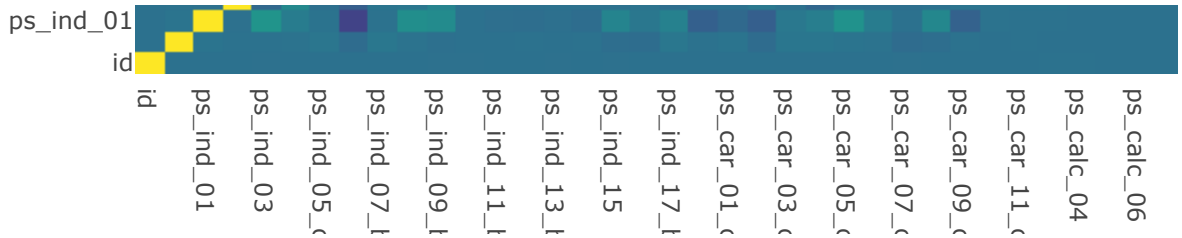
int型の列については、相関係数のヒートマップをインタラクティブに生成する方法を示すため、Plotlyライブラリを使用することに切り替えましょう。 以前のPlotlyプロットと同様に、単に "go.Heatmap"を呼び出すことでヒートマップオブジェクトを生成します。 ここでは、3つの異なる軸に値を提供しなければなりません。ここでは、x軸とy軸が列名を取り込み、相関値はz軸によって与えられます。 colorscale属性は、ヒートマップに表示されるさまざまなカラーパレットに対応するキーワードを取得します。この例では、私はGraysのカラースケールを使用しています（他のものにはPortlandとViridisが含まれます）。

下記のカーネルはz値にtrain_intの相関係数、x,y値にtrain_intのカラム値をとりヒートマップを表示している。

In [11]:

```
#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='whit
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='Pearson Correlation of Integer-type features',
    xaxis = dict(ticks='', nticks=36),
    yaxis = dict(ticks='' ),
    width = 900, height = 700)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='labelled-heatmap')
```





上図を見ると非常にほとんどの要素同士の相関がないことがわかります。

同様に相関プロットでは0値のセルが非常に多く観測されていることから、線型相関のない膨大な数の列があることがわかります。 これは私たちにとって非常に有用な観測であり、特に主成分分析（PCA）などの次元削減変換を実行しようとしている場合はある程度の相関は必要です。興味のある機能は以下の通りです。

否定相関機能：ps_ind_06_bin、ps_ind_07_bin、ps_ind_08_bin、ps_ind_09_bin

注目すべき興味深い側面は、私たちの初期の無効解析では、ps_car_03_catとps_car_05_catに多くの欠損値またはnull値が含まれていることが分かりました。したがって、これらの両方の特徴がこれに基づいてお互いに強い正の線形相関をしめしているのは驚くべきことではありませんが、データの根底にある真実を反映していないかもしれません。

相互情報量プロット

相互情報量は目標変数とそれに対して計算される対応する特徴との間の相互情報量を検査することを可能にするもう一つの有用なツールである。分類問題では、 二つのランダム変数と０（ランダム変数がお互いに独立している）から高い値（従属関係を示す）までの範囲を測定するsklearnのmutual_info_classifメソッドを都合よく呼び出すことができます。 したがって、これはtargetからどれだけ多く情報が特徴の中に含まれるのかというアイデアを私たちに与えるのに役立ちます。 mutual_info_classif関数のsklearnの実装は、「k-最近傍距離からのエントロピー推定に基づくノンパラメトリックな手法に依存している」と述べていますが、より詳しくはSklearn公式ページで見ることができますlink here (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html#sklearn.feature_selection.

下記のカーネルはsklearnライブラリーのmutual_info_classifメソッドでパラメーターのXにtrain_float.values,Yにtrain.target.values、n_neighbors=3、andom_state=17として離散目標変数の相互情報量を表示している。

values : https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.values.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.values.html)

In [12]:

```
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]
```

バイナリ機能検査

検査する可能性のあるデータの別の側面は、バイナリ値のみを含む列です。つまり、値が1または0の2つの値のどちらか一方しか取らない場合です。次に、これらのバイナリ値を含むすべての列を格納し、 これらのバイナリ値の縦の棒グラフプロットは次のようになります。

下記のカーネルはバイナリ値のみを持つ列をbin_colに格納。そのうち値が０のみを持つ列をzero_listに、１のみを持つ列をone_listに配列として格納している。

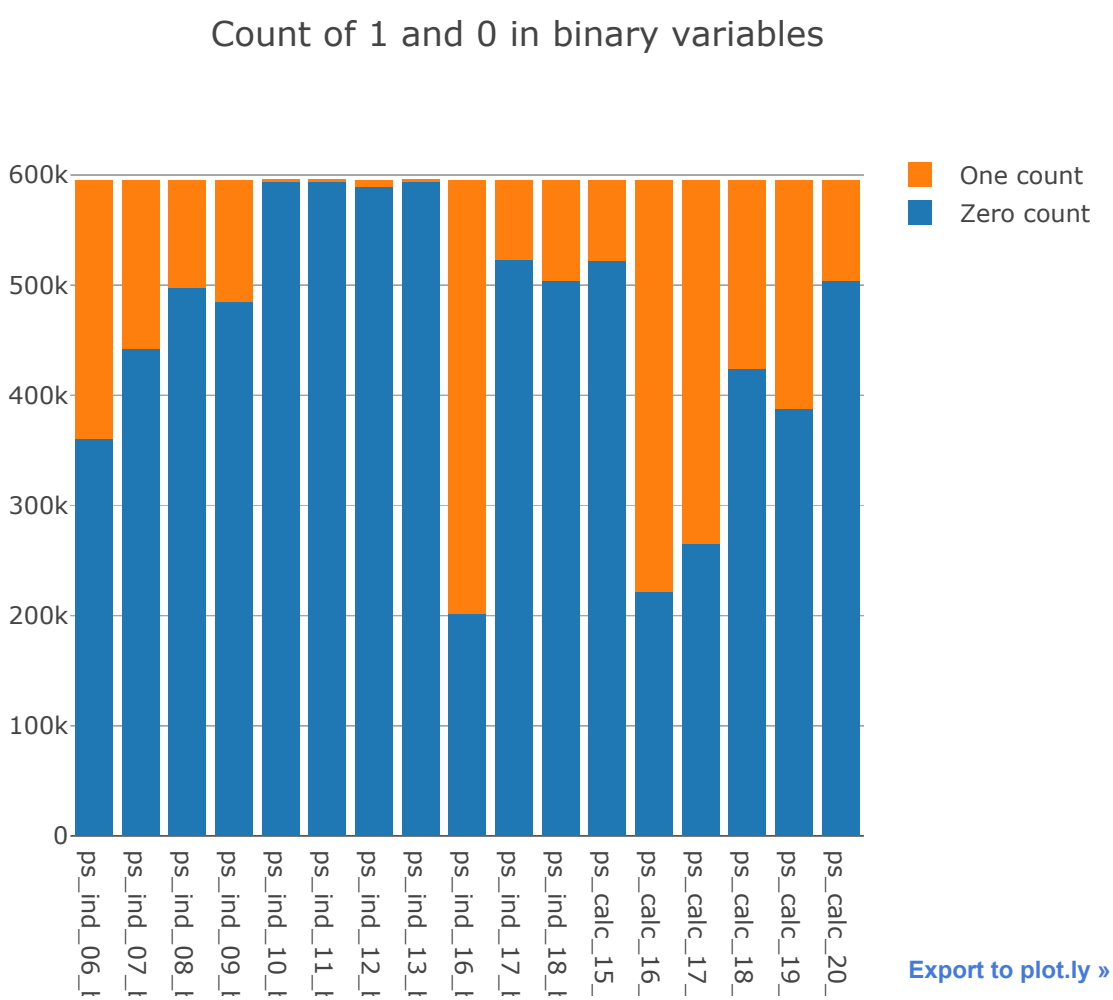
In [13]:

```
bin_col = [col for col in train.columns if '_bin' in col]
zero_list = []
one_list = []
for col in bin_col:
    zero_list.append((train[col]==0).sum())
    one_list.append((train[col]==1).sum())
```

下記のカーネルは上で作成した二つの配列を使って積み上げ棒グラフを作成・表示している。

In [14]:

```
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')
```



ここでは、完全にゼロによって支配されている、ps_ind_10_bin、ps_ind_11_bin、ps_ind_12_bin、ps_ind_13_binの4つの特徴が存在することがわかります。 これは、targetに対する他のクラスに関する十分な情報を含んでいないので、これらの特徴が全て有用であるかどうかという問題を招きます。

類別的および順序的な特徴検査

最初に、接尾辞 "_cat"のようにカテゴリに分類される機能を見てみましょう。

ランダムフォレストによる特徴重要度

ここでは、ランダムフォレスト分類機で学習したデータを適合させるランダムフォレストモデルを実装し、モデルが学習を終えた後に特徴のランキングを調べます。 これは有用な特徴重要度を得るために多くのパラメータ調整を必要とせず、targetの不均衡に対してかなり堅牢なアンサンブルモデル（ブートストラップ・アグリゲーティングの下で適用された弱決定木学習のアンサンブル）を使用する迅速な方法です。 私たちはランダムフォレストを以下のように呼んでいます：

下記のカーネルはランダムフォレスト分類器を使いモデルの学習を行なっている。

RandomForestClassifier:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

以下はモデルのパラメーターです。

n_estimators：生成する木の数
max_depth：ツリーの最大深度
min_samples_leaf：葉ノードに必要なサンプルの最小数
max_features：最適な分割を探す際に考慮する機能の数
n_jobs：fitと予測の両方を同時に実行するジョブの数
random_state：intの場合、random_stateは乱数ジェネレータによって使用されるシードです

In [15]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=150, max_depth=8, min_samples_leaf=4, max_features=0.2
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散布図の特徴量の重要度

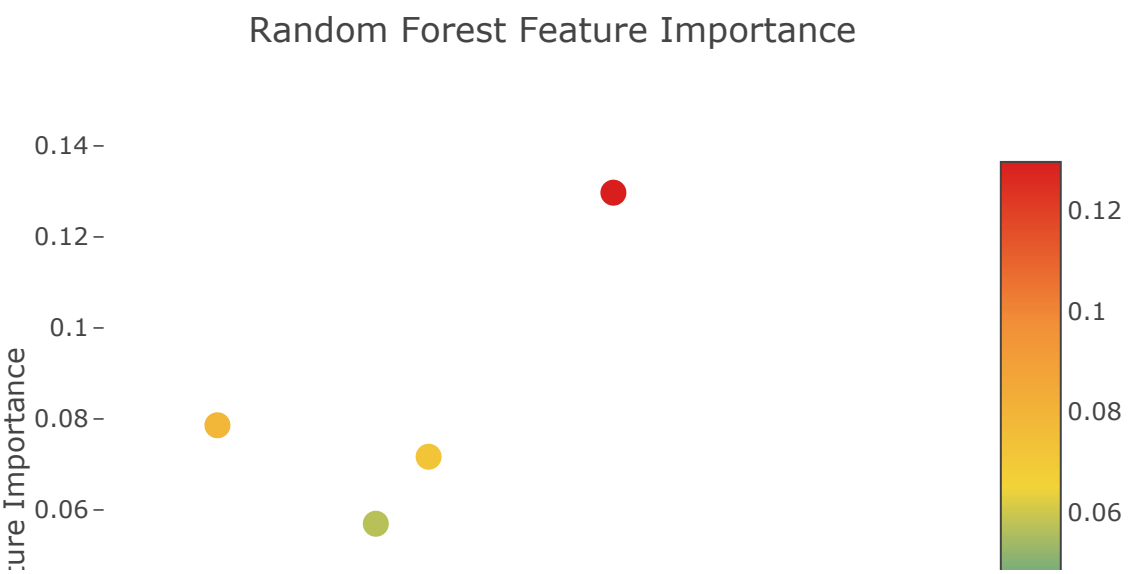
ランダムフォレストを学習した後、属性「feature*importances*」を呼び出して、次のPlotlyプロットの散布図をプロットして、特徴の重要度のリストを取得できます。

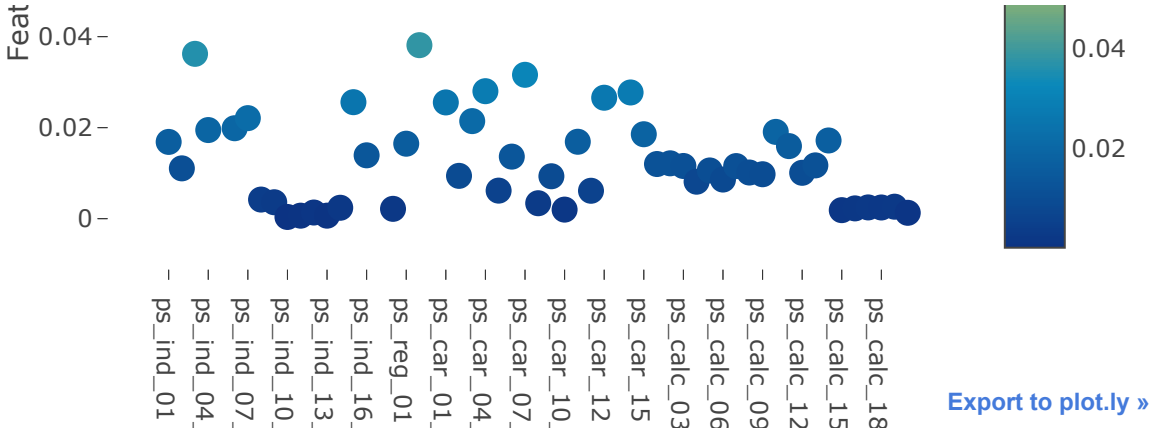
ここでは、Scatterコマンドを呼び出し、前のそれぞれのPlotlyプロットと同様に、y軸とx軸を定義する必要があります。 しかし、散布図で注意を払うことはマーカー属性です。 埋め込まれた散布図を定義し、それによってサイズ、色、スケール制御するのはマーカー属性です。

下記のカーネルはy軸に学習モデルの特徴重要度を設定し、x軸を各特徴として散布図を作成表示している。

In [16]:

```
# 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= 'Feature Importance',
        showgrid=False,
        zeroline=False,
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')
```





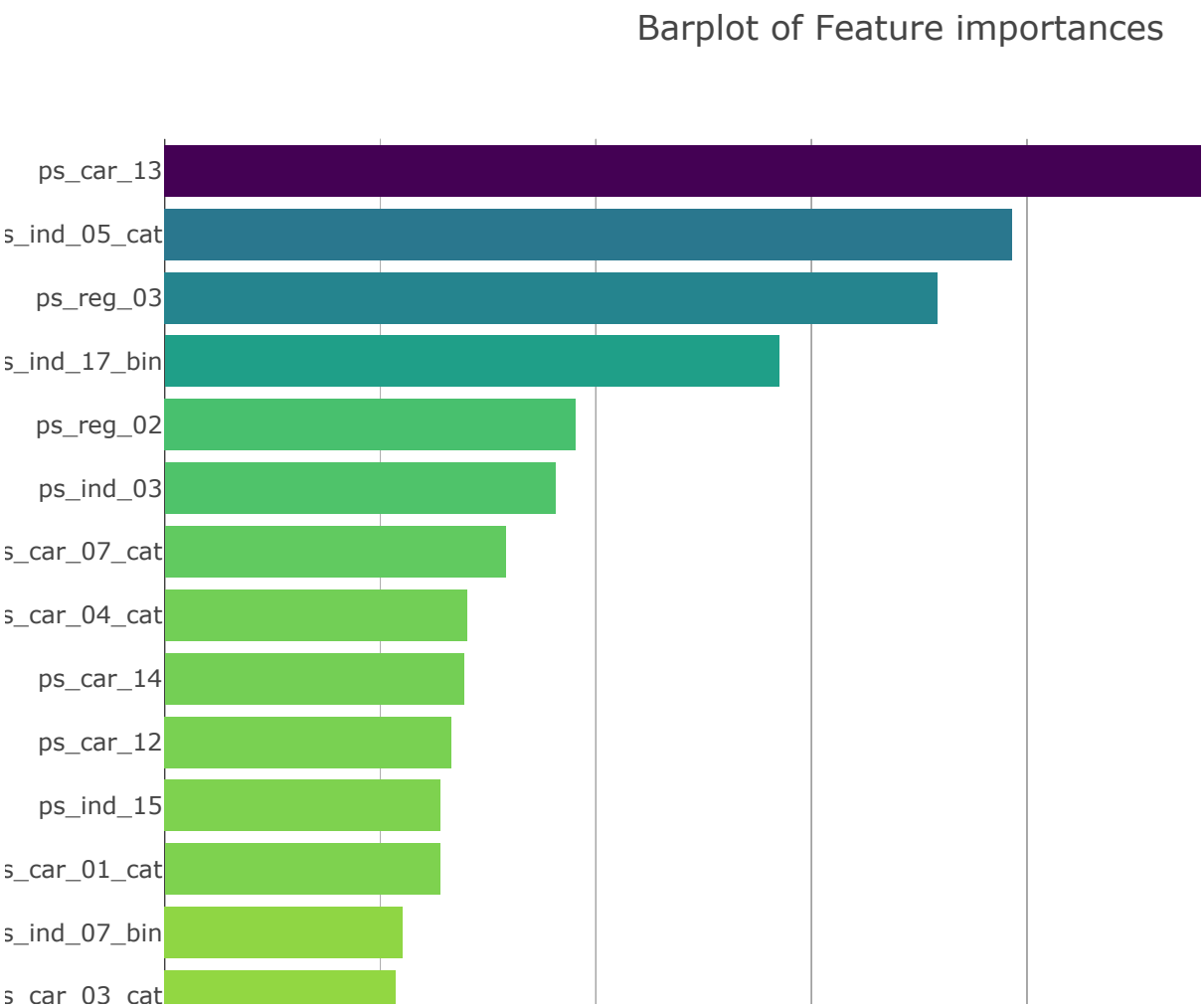
さらに、重要度の高い順にランク付けされたすべての機能のソートされたリストを、次のような同じプロットの棒グラフを使用して最高から最低まで表示することもできます。

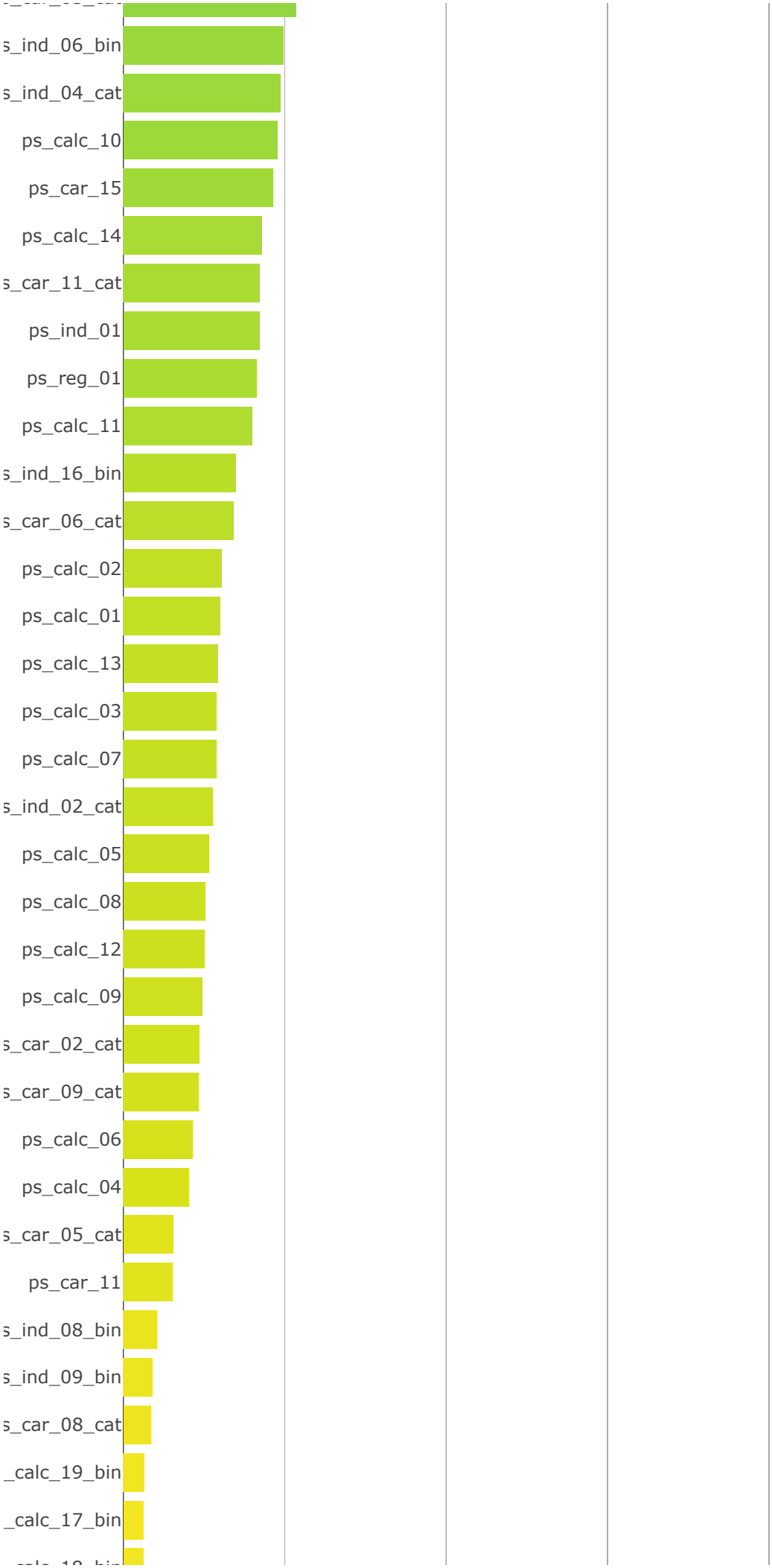
下記のカーネルx軸を各特徴の重要度、y軸を各特徴として棒グラフを表示している。

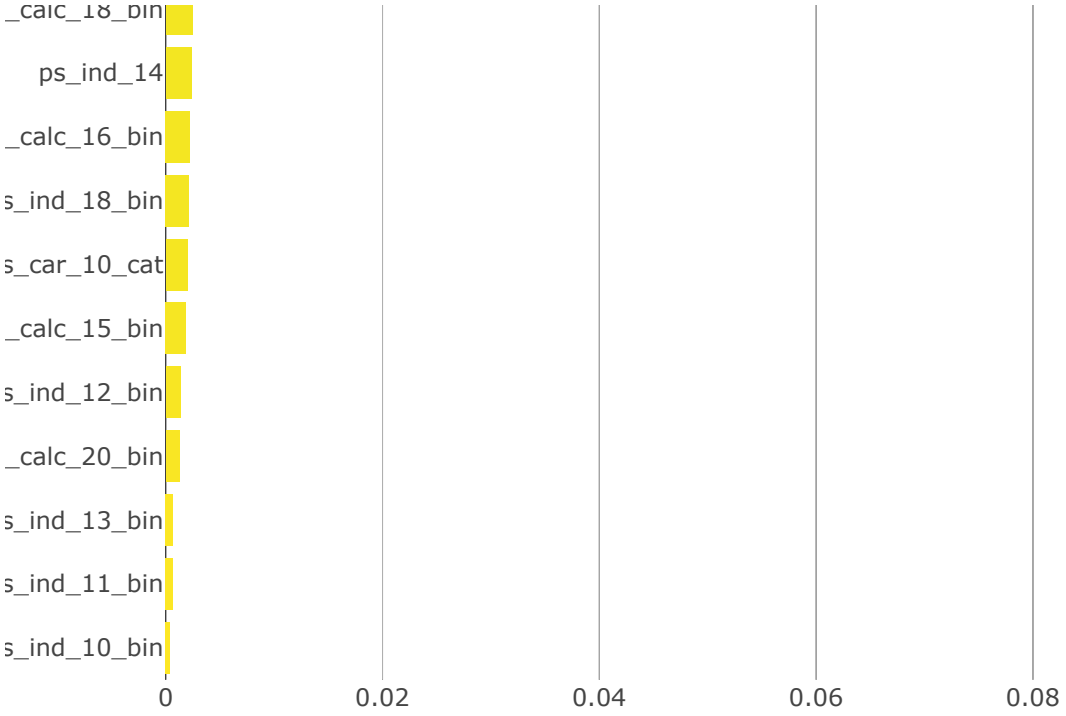
zip: https://docs.python.jp/3/library/functions.html#zip (https://docs.python.jp/3/library/functions.html#zip)

In [17]:

```
x, y = (list(x) for x in zip(*sorted(zip(rf.feature_importances_, features),
                                     reverse = False)))
trace2 = go.Bar(
    x=x ,
    y=y,
    marker=dict(
        color=x,
        colorscale = 'Viridis',
        reversescale = True
    ),
    name='Random Forest Feature importance',
    orientation='h',
)

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')
```




s_ind_06_bin
s_ind_04_cat
ps_calc_10
ps_car_15
ps_calc_14
s_car_11_cat
ps_ind_01
ps_reg_01
ps_calc_11
s_ind_16_bin
s_car_06_cat
ps_calc_02
ps_calc_01
ps_calc_13
ps_calc_03
ps_calc_07
s_ind_02_cat
ps_calc_05
ps_calc_08
ps_calc_12
ps_calc_09
s_car_02_cat
s_car_09_cat
ps_calc_06
ps_calc_04
s_car_05_cat
ps_car_11
s_ind_08_bin
s_ind_09_bin
s_car_08_cat
_calc_19_bin
_calc_17_bin



決定木の可視化

常に使用される他の興味深いトリックやテクニックの1つは、モデルによって作られた木の枝や意思決定を視覚化することです。 簡単にするために、私は決定木（max_depth = 3）に合わせ、したがって決定ブランチでレベル３しか見ることができず、sklearnの "export_graphviz"でグラフ可視化属性のエキスポートを用いて、それから ノートブックで視覚化のためにツリーイメージをエクスポートおよびインポートします。

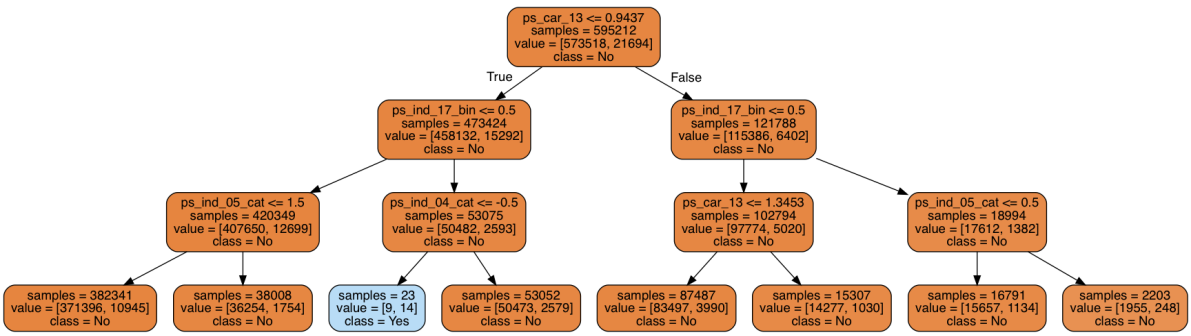
下記のカーネルは深さ３の決定木を可視化している。

DecisionTreeClassifier：http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

In [18]:

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

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 fileとしてエキスポート
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、.pngをWebノートブックに表示できるように変換する
check_call(['dot','-Tpng','tree1.dot','-o','tree1.png'])

# PILによるチャートの注釈
img = Image.open("tree1.png")
draw = ImageDraw.Draw(img)
img.save('sample-out.png')
PImage("sample-out.png",)
```

Out[18]:



Gradient Boostingモデルによる特徴重要度

好奇心のために、我々は特徴重要度をえるために別の学習方法を試してみましょう。 今回は、学習データに合わせるためGradient Boosting分類器を使用します。 Gradient Boostingは、各ステージで回帰木が損失関数の傾き（Sklearnの実装ではデフォルト値である）に当てはまるように段階的な方法で進行します。

以下のカーネルは上でRandom Forestを実装して識別した特徴重要度を、今度はGradient Boosting分類器を使い同様の手順で特徴重要度を識別している。

GradientBoostingClassifier : http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)

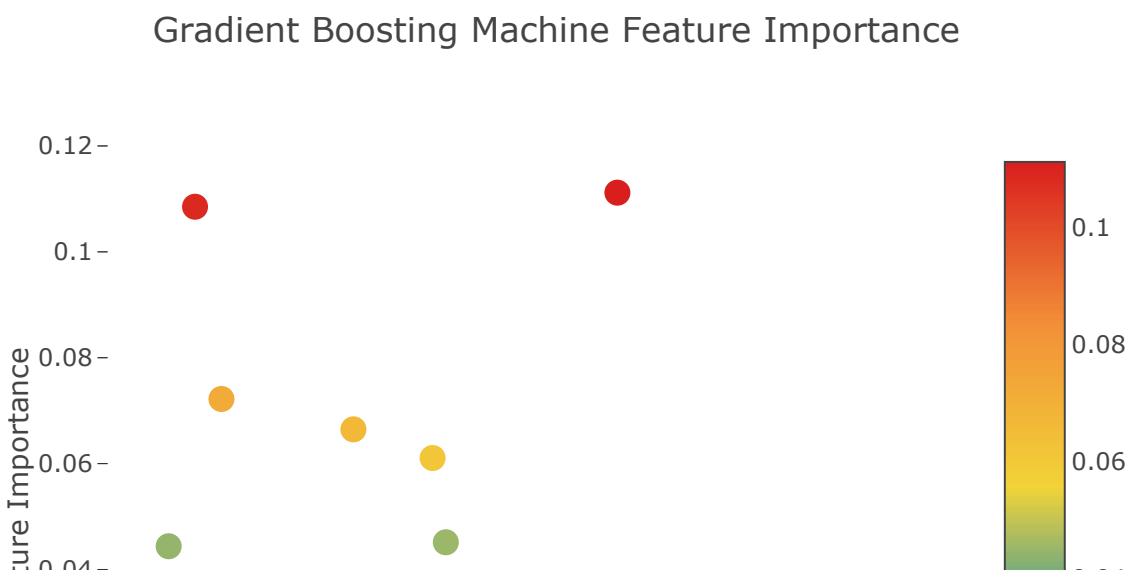
In [19]:

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=100, max_depth=3, min_samples_leaf=4, max_features=
gb.fit(train.drop(['id', 'target'],axis=1), train.target)
features = train.drop(['id', 'target'],axis=1).columns.values
print("----- Training Done -----")
```

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

In [20]:

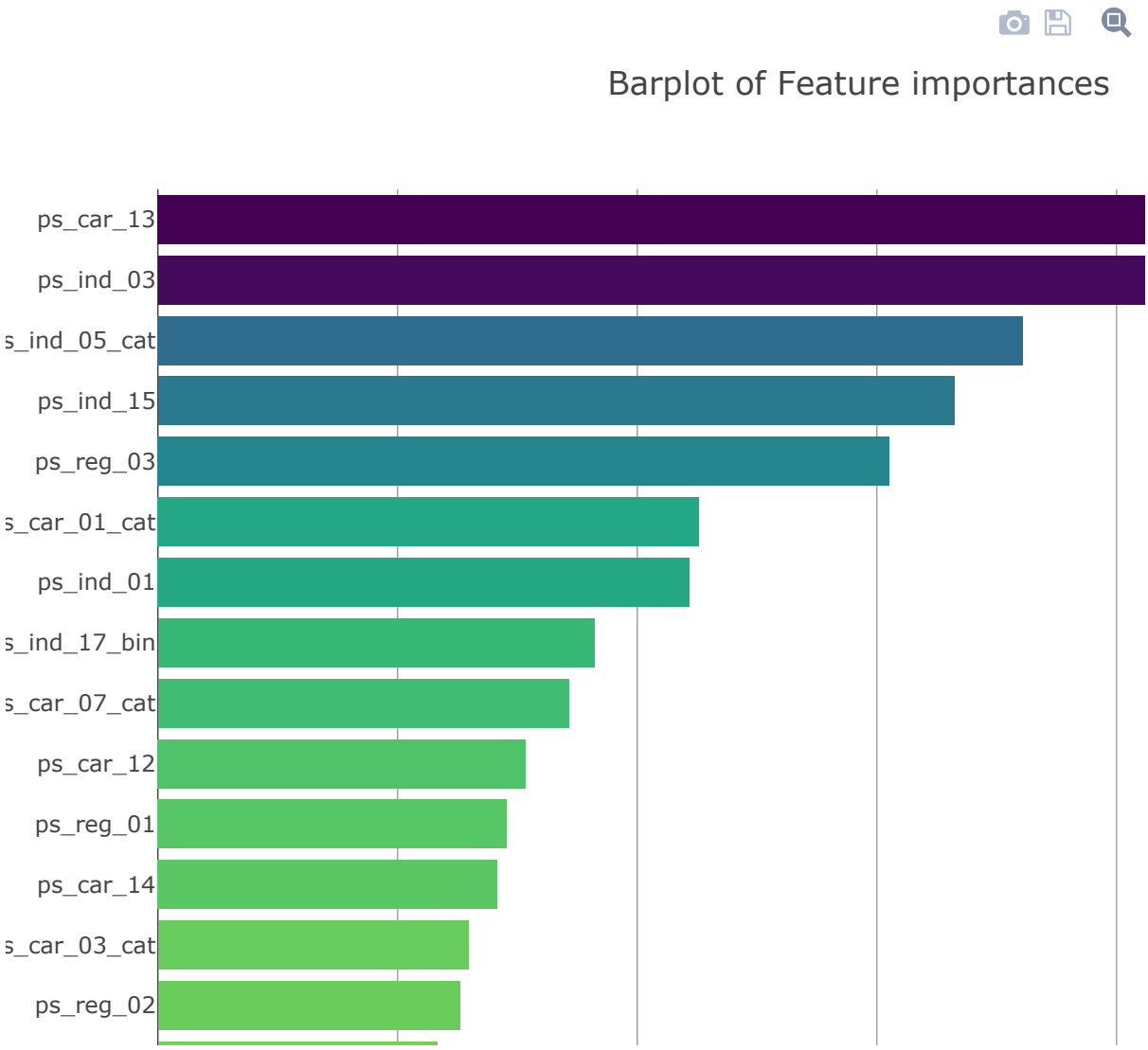
```
# Scatter plot
trace = go.Scatter(
    y = gb.feature_importances_,
    x = features,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 13,
        #size= rf.feature_importances_,
        #color = np.random.randn(500), #set color equal to a variable
        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= 'Feature Importance',
        showgrid=False,
        zeroline=False,
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')
```

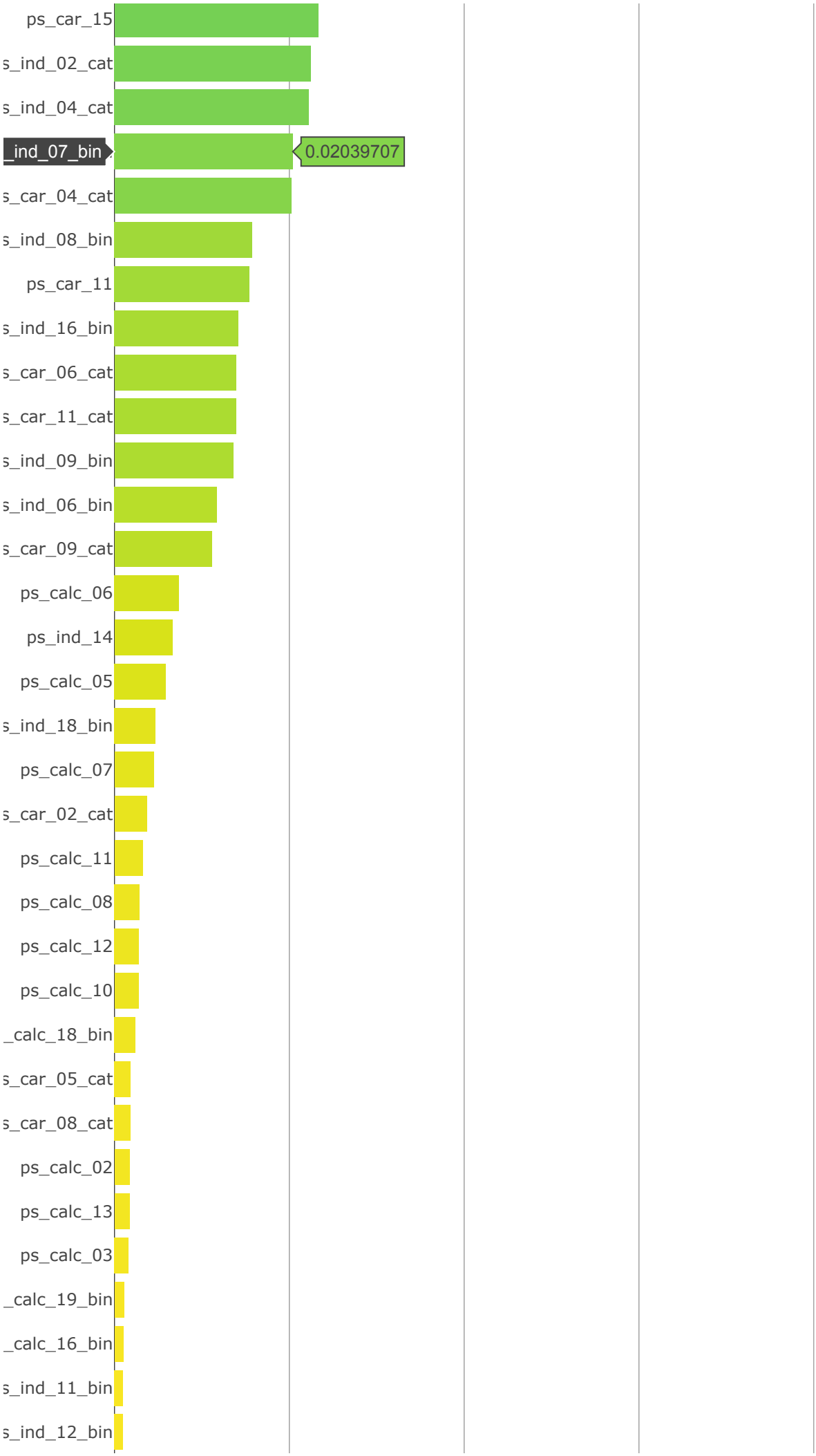


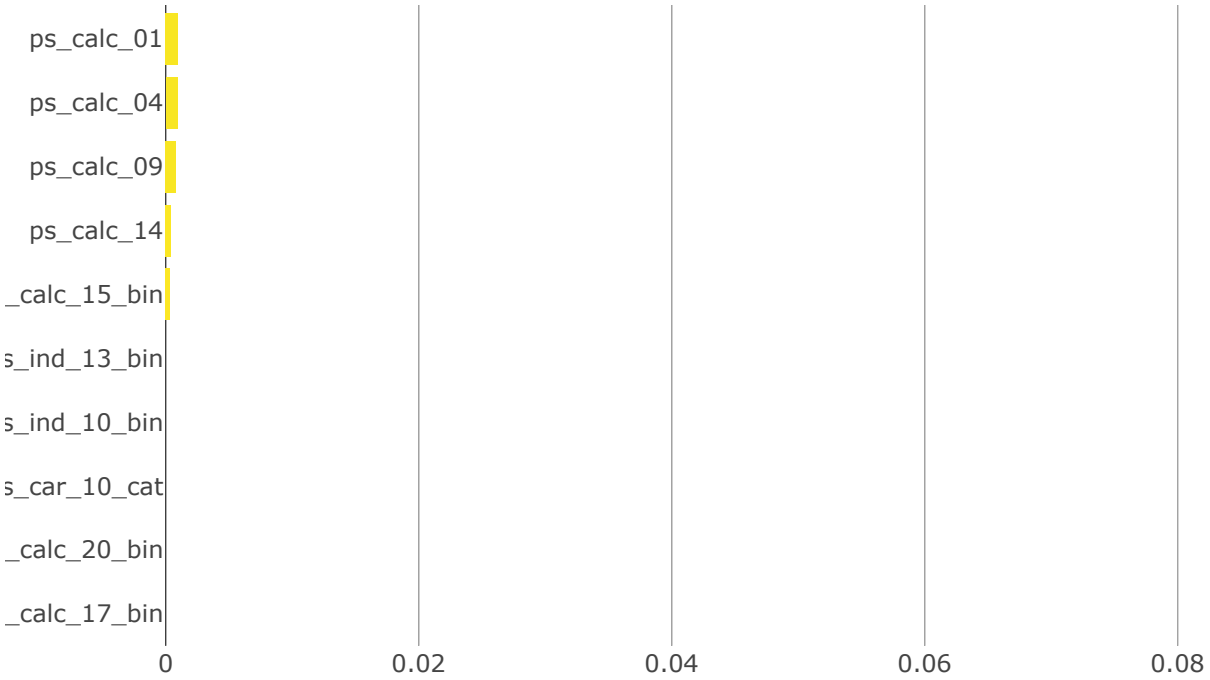
Feat
0.04
0.02
0
ps_ind_01
ps_ind_04
ps_ind_07
ps_ind_10
ps_ind_13
ps_ind_16
ps_reg_01
ps_car_01
ps_car_04
ps_car_07
ps_car_10
ps_car_12
ps_car_15
ps_calc_03
ps_calc_06
ps_calc_09
ps_calc_12
ps_calc_15
ps_calc_18
0.04
0.02
0
Export to plot.ly »

In [21]:

```
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')
```




ps_car_15
s_ind_02_cat
s_ind_04_cat
_ind_07_bin
0.02039707
s_car_04_cat
s_ind_08_bin
ps_car_11
s_ind_16_bin
s_car_06_cat
s_car_11_cat
s_ind_09_bin
s_ind_06_bin
s_car_09_cat
ps_calc_06
ps_ind_14
ps_calc_05
s_ind_18_bin
ps_calc_07
s_car_02_cat
ps_calc_11
ps_calc_08
ps_calc_12
ps_calc_10
_calc_18_bin
s_car_05_cat
s_car_08_cat
ps_calc_02
ps_calc_13
ps_calc_03
_calc_19_bin
_calc_16_bin
s_ind_11_bin
s_ind_12_bin



興味深いことに、ランダムフォレストとグラジエントブースト学習モデルの両方で、両モデルが選んだ最も重要な特徴は、列： **ps_car_13** であったことがわかります。 この特定の特徴は詳細な調査を必要としますので、深く掘り下げて検討してください。

結論

我々は、null値とデータ品質を検査することでPorto Seguroデータセットをかなり広範に調査し、特徴間の線形相関を調べ、特徴分布のいくつかを検査し、モデルが重要と考える特徴を識別することができる、いくつかの学習モデル（ランダムフォレストとグラジエントブースト分類器）を実装することにより、いくつかの機能分布を点検しました。

To Be Continued..

つづく