# Analysis in Medicare Provider Utilization and Payment Data: From the Prospectives of Average Difference between Submitted and Charged Medicare Amount from Physician in California

## Data Input

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:

```
data = pd.read_csv("medical_insurance.csv")
data.head()
```

```
/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2728: DtypeWarning: Columns (10
) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

| | National Provider Identifier | Last Name/Organization Name of the Provider | First Name of the Provider | Middle Initial of the Provider | Credentials of the Provider | Gender of the Provider | Entity Type of the Provider | Street Address 1 of the Provider | Street Address 2 of the Provider | City of Provi |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1003000126 | ENKESHAFI | ARDALAN | NaN | M.D. | M | I | 900 SETON DR | NaN | CUMBERL/ |
| **1** | 1003000126 | ENKESHAFI | ARDALAN | NaN | M.D. | M | I | 900 SETON DR | NaN | CUMBERL/ |
| **2** | 1003000126 | ENKESHAFI | ARDALAN | NaN | M.D. | M | I | 900 SETON DR | NaN | CUMBERL/ |
| **3** | 1003000126 | ENKESHAFI | ARDALAN | NaN | M.D. | M | I | 900 SETON DR | NaN | CUMBERL/ |
| **4** | 1003000126 | ENKESHAFI | ARDALAN | NaN | M.D. | M | I | 900 SETON DR | NaN | CUMBERL/ |

5 rows × 26 columns

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9497892 entries, 0 to 9497891
Data columns (total 26 columns):
National Provider Identifier                            int64
Last Name/Organization Name of the Provider             object
First Name of the Provider                              object
Middle Initial of the Provider                          object
Credentials of the Provider                             object
Gender of the Provider                                  object
Entity Type of the Provider                             object
Street Address 1 of the Provider                        object
Street Address 2 of the Provider                        object
City of the Provider                                    object
Zip Code of the Provider                                object
State Code of the Provider                              object
Country Code of the Provider                            object
Provider Type                                           object
Medicare Participation Indicator                        object
Place of Service                                        object
HCPCS Code                                              object
HCPCS Description                                       object
HCPCS Drug Indicator                                    object
Number of Services                                      float64
Number of Medicare Beneficiaries                        float64
Number of Distinct Medicare Beneficiary/Per Day Services    float64
Average Medicare Allowed Amount                         float64
Average Submitted Charge Amount                         float64
Average Medicare Payment Amount                         float64
Average Medicare Standardized Amount                    float64
dtypes: float64(7), int64(1), object(18)
memory usage: 1.8+ GB
```

In [4]:

```
# subset dataset to only Californa data
data_ca = data.loc[data['State Code of the Provider']=='CA']
display(data_ca.shape)
```

```
(731564, 26)
```

We can see that number of samples are reduced from 9+ millions to 700 thousands. Let's start to dive more into the dataset

# Exploratory Data Analysis

## Missing Value

Since there's small portion of missing values in the remaining columns, these missing values are just dropped out.

In [5]:

```
# function to calculate missing value percentage
def get_missing_percentage(column):
    num = column.isnull().sum()
    total_n = len(column)
    return round(num/total_n, 2)
```

In [6]:

```
data_ca_drop = data_ca.drop(['Middle Initial of the Provider', 'National Provider Identifier','First Na
me of the Provider','Credentials of the Provider','Street Address 2 of the Provider','State Code of the
Provider','Country Code of the Provider'], axis=1)
```

In [7]:

```
print (data_ca['Gender of the Provider'].describe())
```

```
count     691100
unique         2
top            M
freq      515106
Name: Gender of the Provider, dtype: object
```

Name: Gender of the Provider, dtype: object

we can see that gender in male accounts for almost 74% of the data. It can be inferred from the fact that generally there are more male physicians (provider) than female physicians.

In [8]:

```
data_ca_drop_null = data_ca_drop[data_ca_drop.columns[data_ca_drop.isnull().any()].tolist()]
get_missing_percentage(data_ca_drop_null)
```

Out[8]:

```
Last Name/Organization Name of the Provider    0.00
Gender of the Provider                         0.06
dtype: float64
```

Since there's only missing value in gender of the provider with missing at around 6%, therefore missing values were dropped.

In [9]:

```
data_ca_drop = data_ca_drop.dropna()
data_ca_drop.isnull().sum()
```

Out[9]:

```
Last Name/Organization Name of the Provider                 0
Gender of the Provider                                      0
Entity Type of the Provider                                 0
Street Address 1 of the Provider                            0
City of the Provider                                        0
Zip Code of the Provider                                    0
Provider Type                                               0
Medicare Participation Indicator                            0
Place of Service                                            0
HCPCS Code                                                  0
HCPCS Description                                           0
HCPCS Drug Indicator                                        0
Number of Services                                          0
Number of Medicare Beneficiaries                            0
Number of Distinct Medicare Beneficiary/Per Day Services    0
Average Medicare Allowed Amount                             0
Average Submitted Charge Amount                             0
Average Medicare Payment Amount                             0
Average Medicare Standardized Amount                        0
dtype: int64
```

Now we don't have any missing value in the dataframe, let's look at more details in this data. First, let's create a column to show the difference between "Avergae submitted Charged Amount" and "Average Medicare Allowed Amount"

In [10]:

```
data_ca_drop['Average Medicare Difference'] = data_ca_drop['Average Submitted Charge Amount']- data_ca_
drop['Average Medicare Allowed Amount']
```

## Subset dataframe based on Provider Type with Top 5 highest Average Medicare Difference

Let's focus on the top 5 provider types that have the most average medicare difference.

In [11]:

```
data_pt_amaa = data_ca_drop.groupby('Provider Type')['Average Medicare Difference'].mean().reset_index(
).sort_values('Average Medicare Difference', ascending = False).head(5)
data_pt_amaa
```

Out[11]:

| | Provider Type | Average Medicare Difference |
|---|---|---|
| **75** | Thoracic Surgery | 1192.946294 |
| **43** | Neurosurgery | 1149.126776 |
| **6** | Cardiac Surgery | 1146.053486 |

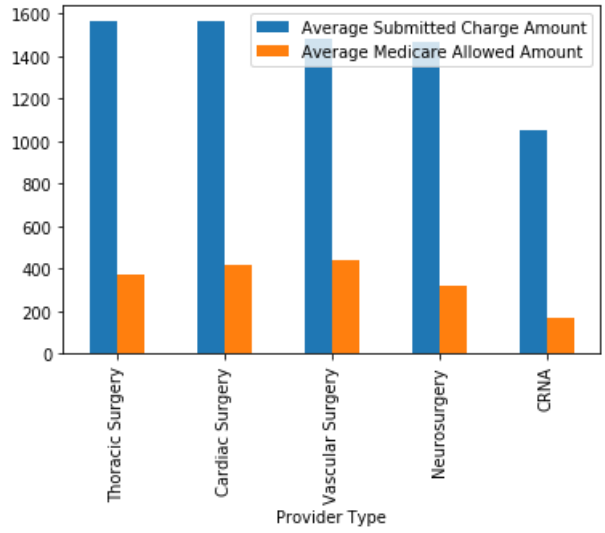
| | Provider Type | Average Medicare Difference |
|---|---|---|
| 79 | Vascular Surgery | 1042.064660 |
| 4 | CRNA | 882.815789 |

In [12]:

```
provider_list = ['Thoracic Surgery','Neurosurgery','Cardiac Surgery', 'Vascular Surgery', 'CRNA']
data_ca_drop_pro = data_ca_drop.loc[data_ca_drop['Provider Type'].isin(provider_list)]

# group by provider type and plot bar plot
data_ca_drop_pro.groupby('Provider Type')['Average Submitted Charge Amount','Average Medicare Allowed A
mount'].mean().sort_values(by = 'Average Submitted Charge Amount', ascending = False).plot(kind = 'bar'
)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c29b978>
```




In [13]:

```
data_ca_drop_pro.groupby('Provider Type')['Average Medicare Difference'].mean().sort_values(ascending =
False)
```

Out[13]:

```
Provider Type
Thoracic Surgery    1192.946294
Neurosurgery        1149.126776
Cardiac Surgery     1146.053486
Vascular Surgery    1042.064660
CRNA                 882.815789
Name: Average Medicare Difference, dtype: float64
```

In [14]:

```
### Top 5 Average Medicare Difference Cities
```

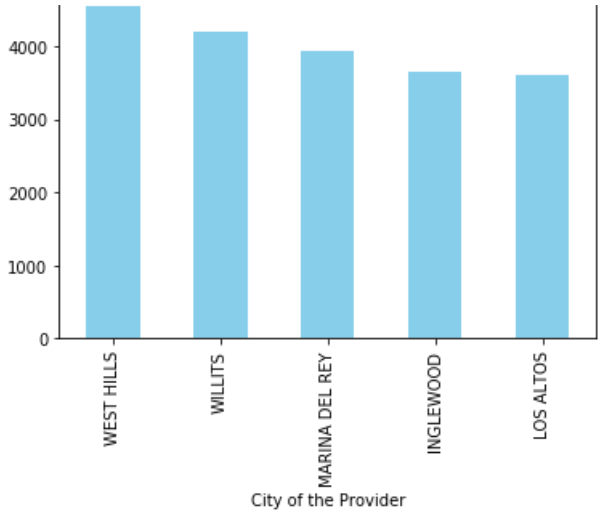
We may also curious about what would average medicare difference varies among cities, here, we extract top 5 cities that have most average medicare difference:

In [15]:

```
data_ca_drop_pro.groupby('City of the Provider')['Average Medicare Difference'].mean().sort_values(asce
nding = False).head(5).plot(kind = 'bar', color = 'skyblue')
data_ca_drop_pro.groupby('City of the Provider')['Average Medicare Difference'].mean().sort_values(asce
nding = False).head(5)
```

Out[15]:

```
City of the Provider
WEST HILLS         4544.530351
WILLITS            4189.692122
MARINA DEL REY     3933.719806
INGLEWOOD          3641.784906
LOS ALTOS          3608.698382
Name: Average Medicare Difference, dtype: float64
```




We can see that West Hills has the most difference in average medicare amount, with around $4500 Now, let's subset dataframe based on top 5 provider types.

## ZIPCODE

This section of code is referred from: https://www.christianpeccei.com/zipmap/

In [16]:

```
def read_ascii_boundary(filestem):
    '''
    Reads polygon data from an ASCII boundary file.
    Returns a dictionary with polygon IDs for keys. The value for each
    key is another dictionary with three keys:
    'name' - the name of the polygon
    'polygon' - list of (longitude, latitude) pairs defining the main
    polygon boundary
    'exclusions' - list of lists of (lon, lat) pairs for any exclusions in
    the main polygon
    '''
    metadata_file = filestem + 'a.dat'
    data_file = filestem + '.dat'
    # Read metadata
    lines = [line.strip().strip('"') for line in open(metadata_file)]
    polygon_ids = lines[::6]
    polygon_names = lines[2::6]
    polygon_data = {}
    for polygon_id, polygon_name in zip(polygon_ids, polygon_names):
        # Initialize entry with name of polygon.
        # In this case the polygon_name will be the 5-digit ZIP code.
        polygon_data[polygon_id] = {'name': polygon_name}
    del polygon_data['0']
    # Read lon and lat.
    f = open(data_file)
    for line in f:
        fields = line.split()
        if len(fields) == 3:
            # Initialize new polygon
            polygon_id = fields[0]
            polygon_data[polygon_id]['polygon'] = []
            polygon_data[polygon_id]['exclusions'] = []
        elif len(fields) == 1:
            # -99999 denotes the start of a new sub-polygon
            if fields[0] == '-99999':
                polygon_data[polygon_id]['exclusions'].append([])
        else:
            # Add lon/lat pair to main polygon or exclusion
            lon = float(fields[0])
            lat = float(fields[1])
            if polygon_data[polygon_id]['exclusions']:
                polygon_data[polygon_id]['exclusions'][-1].append((lon, lat))
            else:
                polygon_data[polygon_id]['polygon'].append((lon, lat))
    return polygon_data
```

In [17]:

```
#data_ca_drop_pro['Zip Code of the Provider'] = data_ca_drop_pro['Zip Code of the Provider'].astype('str')
data_ca_drop_pro['Zip Code of the Provider'] = data_ca_drop_pro['Zip Code of the Provider'].apply(lambda x: str(x)[:5])
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

In [18]:

```
# group by Zip Code of the Provider
provider_zipcode = data_ca_drop_pro.groupby('Zip Code of the Provider')['Average Medicare Difference'].mean().sort_values(ascending = False)
```

In [19]:

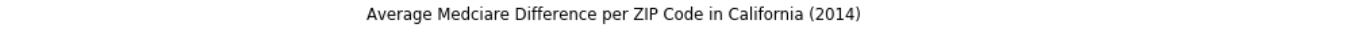
```
from pylab import *
avg_med_diff = {}

# Add data for each ZIP code
for i in range(provider_zipcode.shape[0]):
    avg_med_diff[provider_zipcode.index[i]] = provider_zipcode[i]
max_avg_med_diff = max(avg_med_diff.values())
```

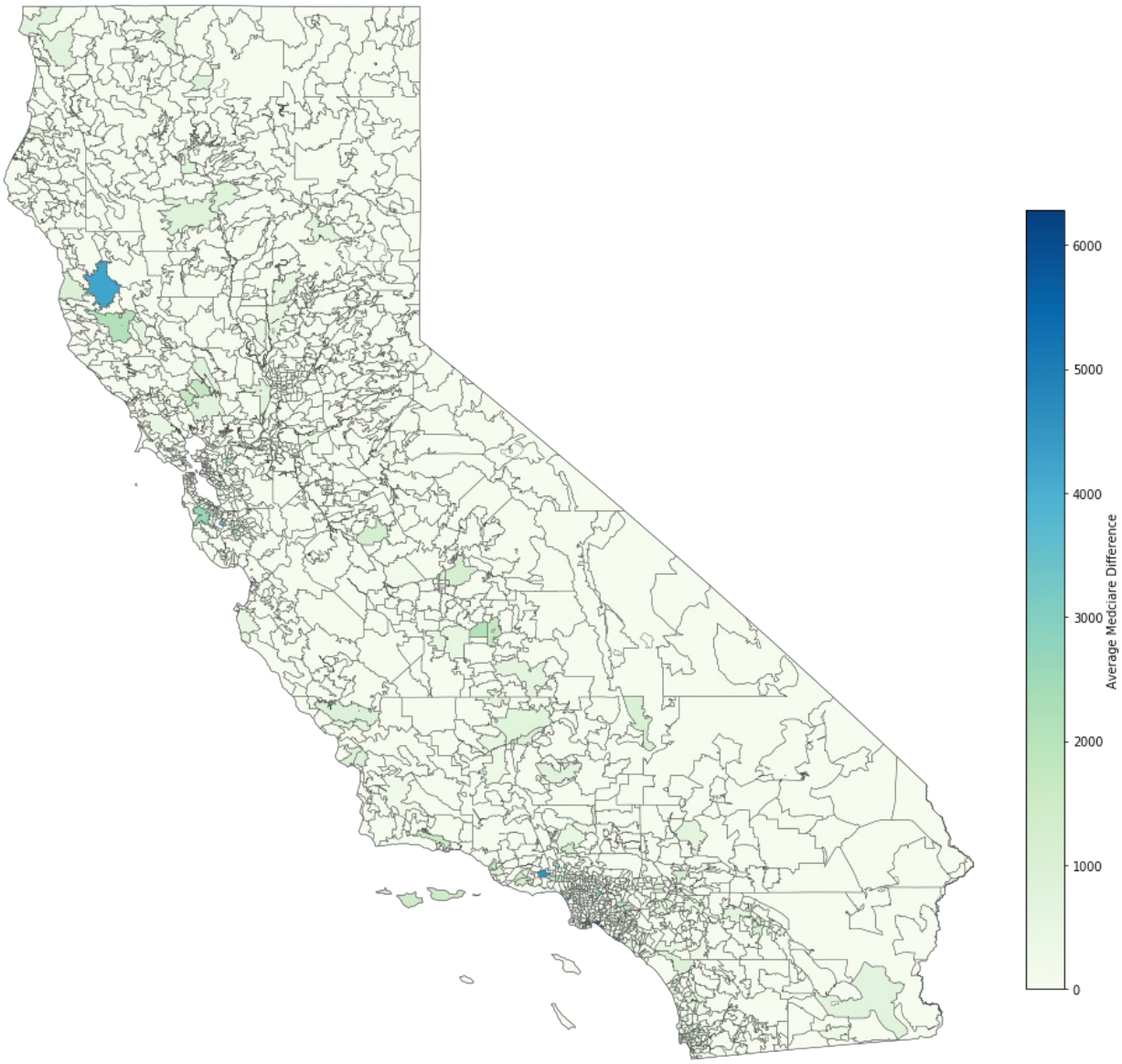
In [20]:

```
# Read in ZIP code boundaries for California
d = read_ascii_boundary('zip5/zt06_d00')
```

In [21]:

```
# Create figure and two axes: one to hold the map and one to hold
# the colorbar
figure(figsize=(15, 15), dpi=70)
map_axis = axes([0.0, 0.0, 0.8, 0.9])
cb_axis = axes([0.83, 0.1, 0.03, 0.6])

# Define colormap to color the ZIP codes.
# You can try changing this to cm.Blues or any other colormap
# to get a different effect
cmap = cm.GnBu

# Create the map axis
axes(map_axis)
axis([-125, -114, 32, 42.5])
gca().set_axis_off()

# Loop over the ZIP codes in the boundary file
for polygon_id in d:
    polygon_data = array(d[polygon_id]['polygon'])
    zipcode = d[polygon_id]['name']
    avg_med_diff_mean = avg_med_diff[zipcode] if zipcode in avg_med_diff else 0.

    # Define the color for the ZIP code
    fc = cmap(avg_med_diff_mean/max_avg_med_diff)

    # Draw the ZIP code
    patch = Polygon(array(polygon_data), facecolor=fc,
        edgecolor=(.3, .3, .3, 1), linewidth=.4)
    gca().add_patch(patch)
title('Average Medciare Difference per ZIP Code in California (2014)')

# Draw colorbar
cb = mpl.colorbar.ColorbarBase(cb_axis, cmap=cmap,
    norm = mpl.colors.Normalize(vmin=0, vmax=max_avg_med_diff))
cb.set_label('Average Medciare Difference')
```






In [22]:

```
# group by city of the provider and zipcode of the provider
data_ca_drop_pro.groupby(['City of the Provider','Zip Code of the Provider'])['Average Medicare Difference'].mean().sort_values(ascending = False).head(5)
```

Out[22]:

```
City of the Provider  Zip Code of the Provider
SAN DIEGO             92093                       17248.377187
LONG BEACH            90803                        6280.067273
WEST HILLS            91307                        4544.530351
WILLITS               95490                        4189.692122
MARINA DEL REY        90292                        3933.719806
Name: Average Medicare Difference, dtype: float64
```

## Procedures in each Provider Type

In [23]:

```
#subset dataframe based on provider type
data_crna = data_ca_drop_pro.loc[data_ca_drop_pro['Provider Type']=='CRNA']
data_vas = data_ca_drop_pro.loc[data_ca_drop_pro['Provider Type']=='Vascular Surgery']
data_cardiac = data_ca_drop_pro.loc[data_ca_drop_pro['Provider Type']=='Cardiac Surgery']
data_thora = data_ca_drop_pro.loc[data_ca_drop_pro['Provider Type']=='Thoracic Surgery']
data_neuro = data_ca_drop_pro.loc[data_ca_drop_pro['Provider Type']=='Neurosurgery']
```

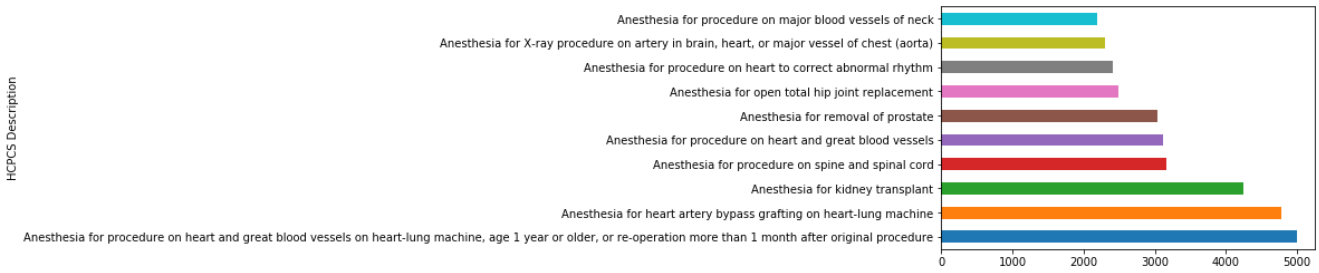
## Top 10 Procedures in CRNA

In [24]:

```
data_crna.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = False).head(10).plot(kind ="barh")
data_crna.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = False).head(10)
```

Out[24]:

```
HCPCS Description
Anesthesia for procedure on heart and great blood vessels on heart-lung machine, age 1 year or older, o
r re-operation more than 1 month after original procedure    5002.128809
Anesthesia for heart artery bypass grafting on heart-lung machine
4772.314838
Anesthesia for kidney transplant
4252.883077
Anesthesia for procedure on spine and spinal cord
3170.178024
Anesthesia for procedure on heart and great blood vessels
3118.022305
Anesthesia for removal of prostate
3040.789101
Anesthesia for open total hip joint replacement
2489.418778
Anesthesia for procedure on heart to correct abnormal rhythm
2407.988031
Anesthesia for X-ray procedure on artery in brain, heart, or major vessel of chest (aorta)
2299.052803
Anesthesia for procedure on major blood vessels of neck
2191.525714
Name: Average Medicare Difference, dtype: float64
```




It seems that anesthesia for procedures on hear and great blood vessel cost has the most differences in average medicare amount, which is around $5000 USD. The procedure includes heart-lung usage, re-operation after original procedures. Further details about the frequency of these re-operating procedures and risk factors that contirbute to the re-operation can be discussed. But these questions will be left opened in the project.

## Top 10 Vascular Surgery

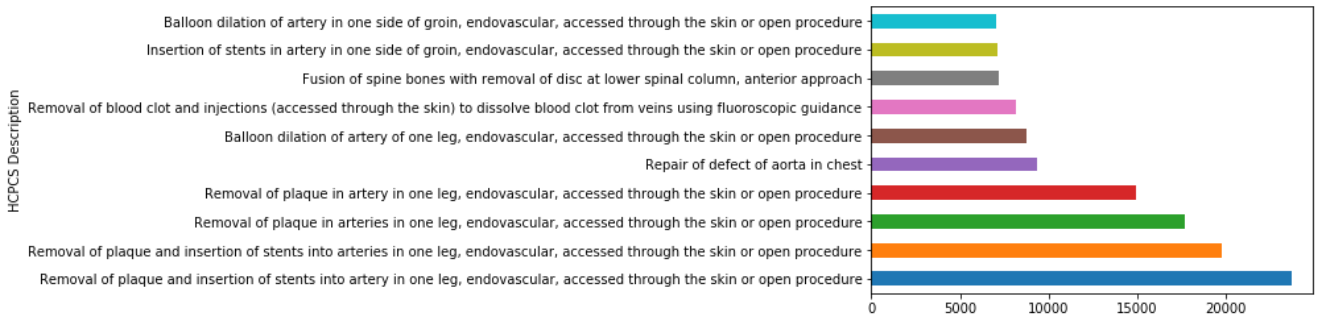
In [25]:

```
data_vas.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = False).head(10).plot(kind ="barh")
data_vas.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = False).head(10)
```

Out[25]:

```
HCPCS Description
Removal of plaque and insertion of stents into artery in one leg, endovascular, accessed through the sk
in or open procedure       23740.022667
Removal of plaque and insertion of stents into arteries in one leg, endovascular, accessed through the
skin or open procedure     19768.319407
Removal of plaque in arteries in one leg, endovascular, accessed through the skin or open procedure
17688.368755
Removal of plaque in artery in one leg, endovascular, accessed through the skin or open procedure
14947.196645
Repair of defect of aorta in chest
9370.670909
Balloon dilation of artery of one leg, endovascular, accessed through the skin or open procedure
8738.647956
Removal of blood clot and injections (accessed through the skin) to dissolve blood clot from veins usin
g fluoroscopic guidance     8160.098750
Fusion of spine bones with removal of disc at lower spinal column, anterior approach
```

```
7164.818288
Insertion of stents in artery in one side of groin, endovascular, accessed through the skin or open pro
cedure                          7086.245425
Balloon dilation of artery in one side of groin, endovascular, accessed through the skin or open proced
ure                             7069.189548
Name: Average Medicare Difference, dtype: float64
```




It seems that cost in removal of plaque and insection of stents into artery has the most differences in average medicare amount, which is $23700 USD, followed by the procedures involving removal of plaque and insection of stents into arteriers. We can see that removel of plaque and insection of stents account for top 4 average medicare amount difference procedures in Vascular Surgery.

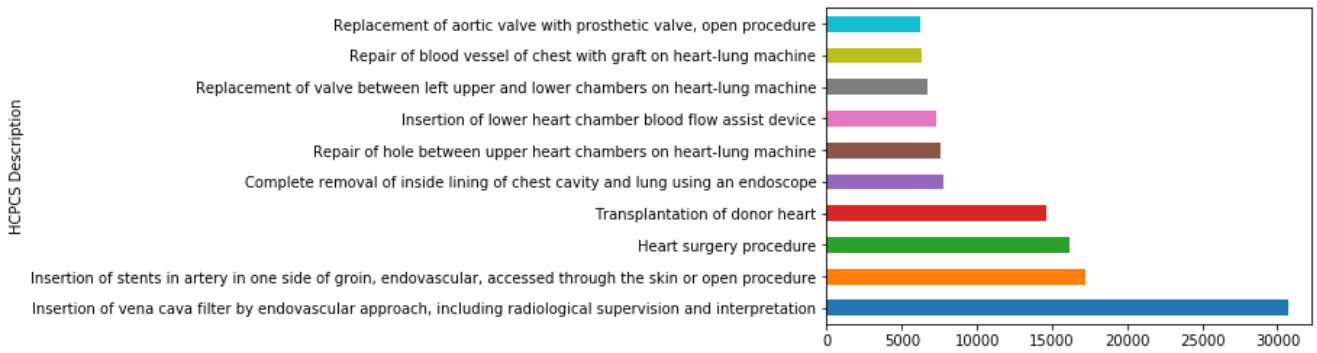
## Top 10 Cardiac Surgery

In [26]:

```
data_cardiac.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending =
False).head(10).plot(kind ="barh")
data_cardiac.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending =
False).head(10)
```

Out[26]:

```
HCPCS Description
Insertion of vena cava filter by endovascular approach, including radiological supervision and interpre
tation    30757.930833
Insertion of stents in artery in one side of groin, endovascular, accessed through the skin or open pro
cedure    17229.114615
Heart surgery procedure
16159.088667
Transplantation of donor heart
14659.190000
Complete removal of inside lining of chest cavity and lung using an endoscope
7801.992500
Repair of hole between upper heart chambers on heart-lung machine
7540.277238
Insertion of lower heart chamber blood flow assist device
7303.768788
Replacement of valve between left upper and lower chambers on heart-lung machine
6728.743013
Repair of blood vessel of chest with graft on heart-lung machine
6352.985000
Replacement of aortic valve with prosthetic valve, open procedure
6235.826208
Name: Average Medicare Difference, dtype: float64
```




The bar chart shows that insection of vena cava by endovascular approach has the most difference in average medicare amount in cardiac surgery, which is $30700 USD.

cardiac surgery, which is $[illegible] USD.

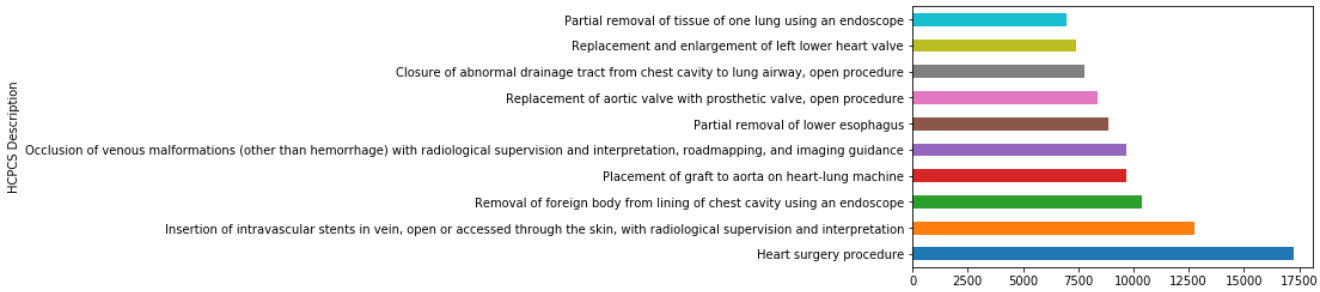
## Top 10 Thoracic Surgery

In [27]:

```
data_thora.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = F
alse).head(10).plot(kind ="barh")
data_thora.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = F
alse).head(10)
```

Out[27]:

```
HCPCS Description
Heart surgery procedure
17248.377187
Insertion of intravascular stents in vein, open or accessed through the skin, with radiological supervi
sion and interpretation                    12753.594117
Removal of foreign body from lining of chest cavity using an endoscope
10403.839235
Placement of graft to aorta on heart-lung machine
9683.436511
Occlusion of venous malformations (other than hemorrhage) with radiological supervision and interpretat
ion, roadmapping, and imaging guidance     9681.330000
Partial removal of lower esophagus
8841.641538
Replacement of aortic valve with prosthetic valve, open procedure
8396.858084
Closure of abnormal drainage tract from chest cavity to lung airway, open procedure
7774.056667
Replacement and enlargement of left lower heart valve
7431.038889
Partial removal of tissue of one lung using an endoscope
6982.266265
Name: Average Medicare Difference, dtype: float64
```




The procedures that involves heart surgery shows the most difference between submitted charged amount and allowed medicare amount, which is around $17200 USD.

## Top 10 Neurosurgery

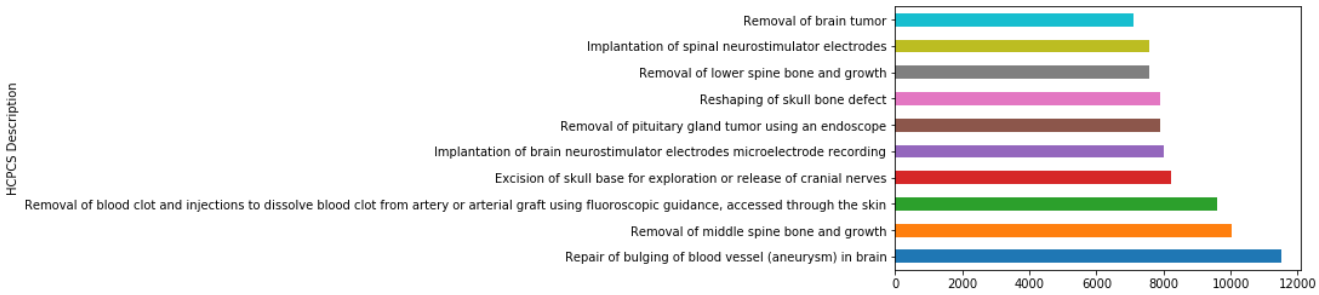
In [28]:

```
data_neuro.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = F
alse).head(10).plot(kind ="barh")
data_neuro.groupby('HCPCS Description')['Average Medicare Difference'].mean().sort_values(ascending = F
alse).head(10)
```

Out[28]:

```
HCPCS Description
Repair of bulging of blood vessel (aneurysm) in brain
11521.145454
Removal of middle spine bone and growth
10031.506923
Removal of blood clot and injections to dissolve blood clot from artery or arterial graft using fluoros
copic guidance, accessed through the skin     9610.130833
Excision of skull base for exploration or release of cranial nerves
8217.257069
Implantation of brain neurostimulator electrodes microelectrode recording
8025.686634
Removal of pituitary gland tumor using an endoscope
7914.086006
Reshaping of skull bone defect
```

```
7901.622421
Removal of lower spine bone and growth
7581.249946
Implantation of spinal neurostimulator electrodes
7562.391111
Removal of brain tumor
7122.118343
Name: Average Medicare Difference, dtype: float64
```




The largest difference in neruosurgery is the procedure of repairing of bulging of blood vessel in brain, which is around 11520 USD.

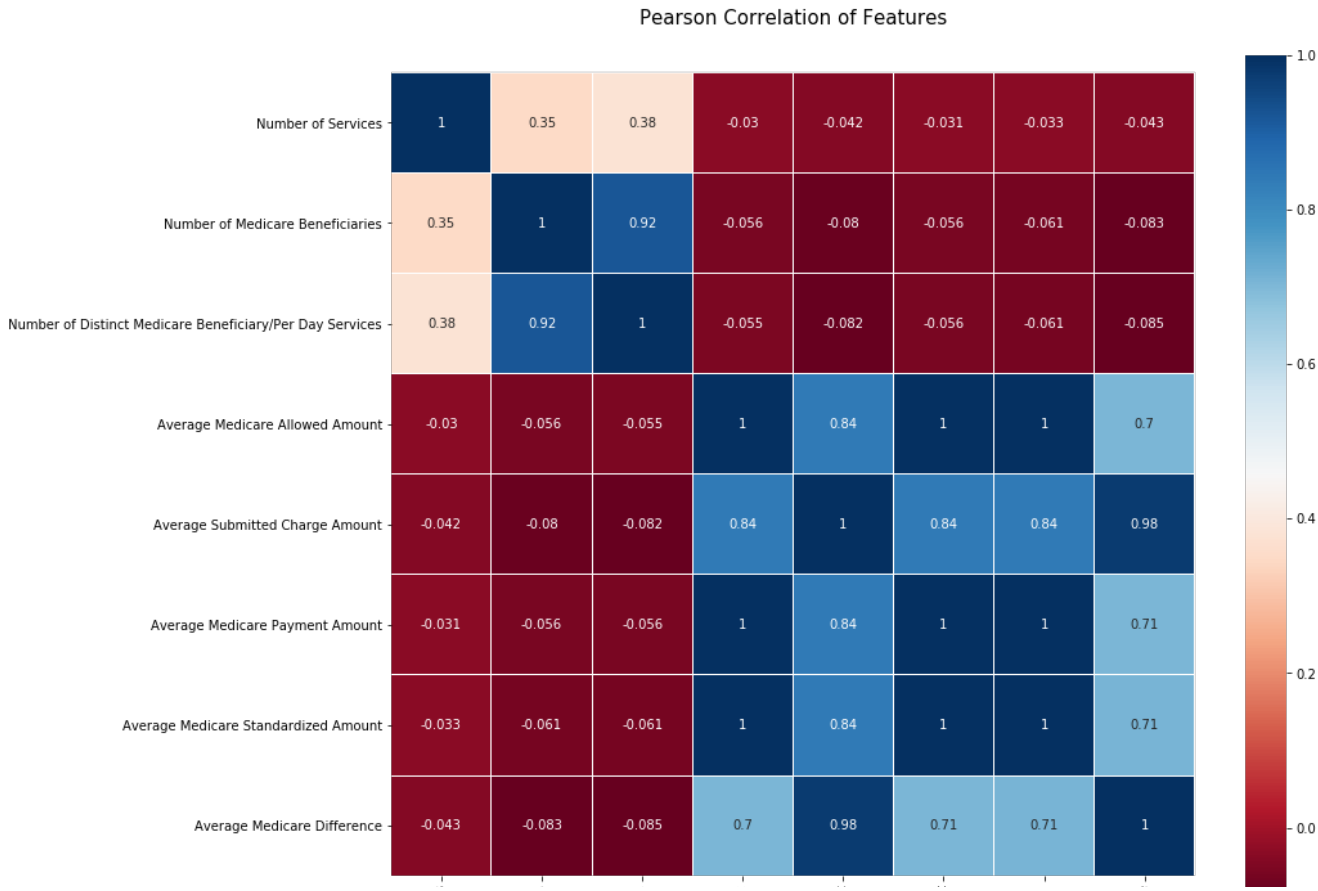
# Feature Exploration

## Correlation Matrix

To investigate what factors would contribute to major impact on the dependent variable - average medicare difference, and understand if independent variables have correlations, correlation matrix is applied.

In [29]:

```
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(data_ca_drop_pro.corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
plt.savefig("corrmatirx.png")
```




We can see that Average Medicare Allowed Amount, Average Submitted Charge Amount, Average Medicare Payment Amount,and Average Medicare Standardized Amount have higher correlation with Average Medicare Difference. We will focus on visualizing these features.

## Feature Transformation

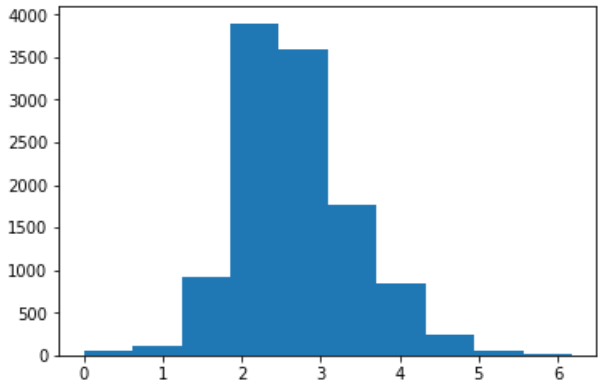
### Average Medicare Difference

In [30]:

```
plt.hist(pow(data_ca_drop_pro['Average Medicare Difference'], 1/6))
```

Out[30]:

```
(array([  45.,  101.,  918., 3897., 3586., 1767.,  847.,  232.,   62.,
         20.]),
 array([0.        , 0.618311  , 1.236622  , 1.85493301, 2.47324401,
        3.09155501, 3.70986601, 4.32817702, 4.94648802, 5.56479902,
        6.18311002]),
 <a list of 10 Patch objects>)
```



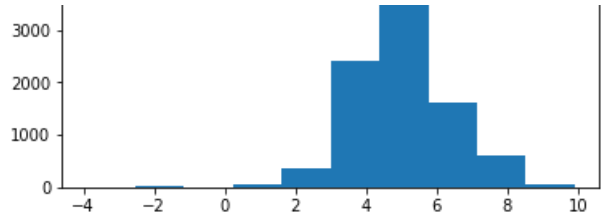

### Average Medicare Allowed Amount

In [31]:

```
# log transformation of Average Medicare Allowed Amount
plt.hist(np.log(data_ca_drop_pro['Average Medicare Allowed Amount']))
```

Out[31]:

```
(array([1.000e+00, 3.600e+01, 9.000e+00, 4.000e+01, 3.650e+02, 2.414e+03,
        6.335e+03, 1.611e+03, 5.960e+02, 6.800e+01]),
 array([-3.91202301, -2.53018528, -1.14834756,  0.23349017,  1.61532789,
         2.99716562,  4.37900334,  5.76084107,  7.14267879,  8.52451652,
         9.90635424]),
 <a list of 10 Patch objects>)
```

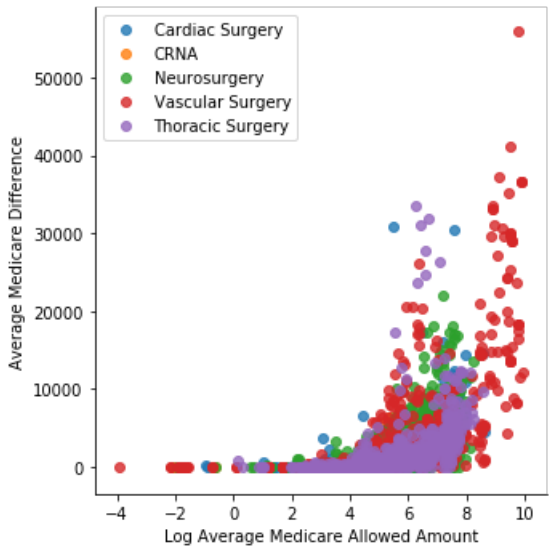



In [32]:

```
# copy df
data_pro_copy = data_ca_drop_pro.copy()
# log transformation of
data_pro_copy['Log Average Medicare Allowed Amount'] = np.log(data_pro_copy['Average Medicare Allowed A
mount'])
sns.lmplot(x = 'Log Average Medicare Allowed Amount', y='Average Medicare Difference', data = data_pro_
copy, fit_reg = False, hue = 'Provider Type', legend = False)
plt.legend(loc='upper left')
```

Out[32]:

```
<matplotlib.legend.Legend at 0x1a19b03c18>
```




We could see there's more variations in vascular surgery. With more allowed average medicare amoumt, there's a increasing trend in variations in average medicare difference and the amount of average medicare difference.

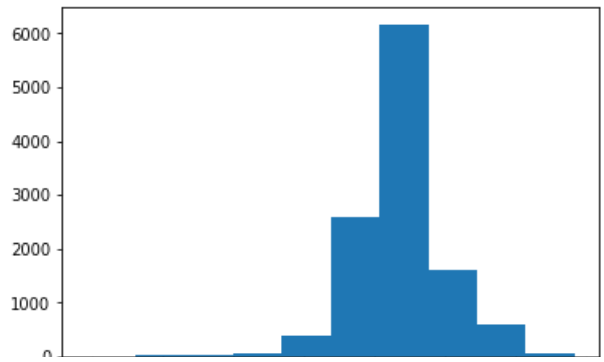
#### Average Medicare Payment Amount

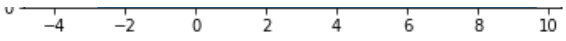
In [33]:

```
plt.hist(np.log(data_ca_drop_pro['Average Medicare Payment Amount']))
```

Out[33]:

```
(array([1.000e+00, 3.600e+01, 9.000e+00, 4.000e+01, 3.720e+02, 2.581e+03,
        6.172e+03, 1.600e+03, 5.960e+02, 6.800e+01]),
 array([-4.15197368e+00, -2.77047552e+00, -1.38897737e+00, -7.47922062e-03,
         1.37401893e+00,  2.75551708e+00,  4.13701524e+00,  5.51851339e+00,
         6.90001154e+00,  8.28150969e+00,  9.66300784e+00]),
 <a list of 10 Patch objects>)
```



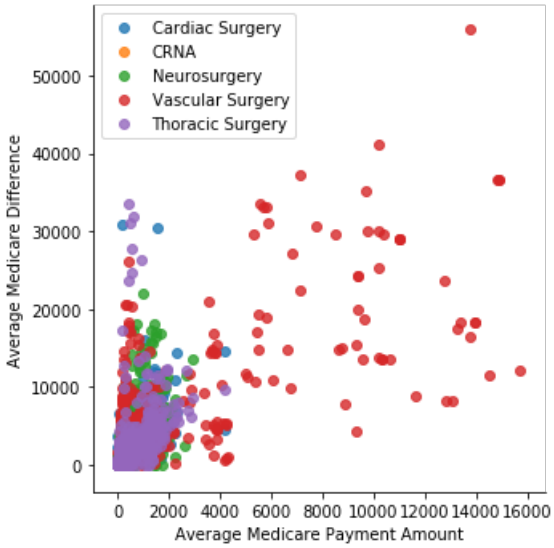



In [34]:

```
# scatter plot
sns.lmplot(x = 'Average Medicare Payment Amount', y='Average Medicare Difference', data = data_ca_drop_
pro, fit_reg = False, hue = 'Provider Type', legend = False)
plt.legend(loc='upper left')
```

Out[34]:

```
<matplotlib.legend.Legend at 0x1a1af55198>
```




The scatter plot showed that vascular surgery varied the most in average medicare payment amount that medicare covered after coinsurance amount deducted compared to other surgery, and it can be visualized that there's slightly positive correlation between average medicare payment amount and average medicare difference.

## Creating Dummy Variables

Now, I want to know that what factors would affect the average submitted charge amount. To do so, I need to get dummies for each categorical value to save space and ease computatinal complexity.

In [35]:

```
# convert categorical variable to dummy variable
data_ca_drop_dummy = pd.get_dummies(data_ca_drop_pro[['Zip Code of the Provider','Entity Type of the Pr
ovider','Provider Type','Medicare Participation Indicator','HCPCS Description','HCPCS Drug Indicator','
City of the Provider']])
```

In [36]:

```
# build continuous variable dataframe
data_ca_drop_continue = data_ca_drop_pro[['Number of Services','Number of Medicare Beneficiaries','Numb
er of Distinct Medicare Beneficiary/Per Day Services','Average Medicare Payment Amount','Average Medica
re Standardized Amount','Average Medicare Difference']]
```

In [37]:

```
# normalize continuous dataframe
data_ca_drop_continue = (data_ca_drop_continue-data_ca_drop_continue.min())/(data_ca_drop_continue.max(
)-data_ca_drop_continue.min())
```

In [38]:

```
# concat binary dataframe and continuous dataframe
data_ca_drop_dummy = pd.concat([data_ca_drop_continue,data_ca_drop_dummy], axis = 1)
```

In [39]:

```
from sklearn.preprocessing import StandardScaler
# run lineaqr regression model
data_ca_drop_dummy.reset_index(drop = True)
X = data_ca_drop_dummy['Average Medicare Difference']
```

```
Y = data_ca_drop_dummy['Average Medicare Difference']
X = data_ca_drop_dummy.drop(['Average Medicare Difference'], axis = 1)
```

In [40]:

```
from sklearn.model_selection import train_test_split
```

In [41]:

```
# split data into training set and testing set
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

## Linear Regression Model as a baseline model

In [42]:

```
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.metrics import mean_squared_error, r2_score
```

In [43]:

```
# train baseline model: linear regression
model = linear_model.LinearRegression()
model.fit(x_train, y_train)
y_pred_lr = model.predict(x_test)
print ("R Squared: ", model.score(x_test,y_test))
print("Residual sum of squares: %.2f"
              % np.mean((model.predict(x_test) - y_test) ** 2))
print ("Linear Regression MSE: %.4f"%mean_squared_error(y_test, y_pred_lr))
```

```
R Squared:  -4.722544063284281e+21
Residual sum of squares: 10821449170476201984.00
Linear Regression MSE: 10821449170476197888.0000
```

## Elastic Regression Model

In [44]:

```
cv_enet = ElasticNetCV(l1_ratio = np.linspace(0.1,1,40.), cv = 10, eps = 0.001, n_alphas = 100, fit_int
ercept = True, normalize = True, max_iter = 2000)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning: object of type <cla
ss 'float'> cannot be safely interpreted as an integer.
  """Entry point for launching an IPython kernel.
```

In [45]:

```
cv_enet.fit(x_train, y_train)
```

Out[45]:

```
ElasticNetCV(alphas=None, copy_X=True, cv=10, eps=0.001, fit_intercept=True,
       l1_ratio=array([0.1    , 0.12308, 0.14615, 0.16923, 0.19231, 0.21538, 0.23846,
       0.26154, 0.28462, 0.30769, 0.33077, 0.35385, 0.37692, 0.4    ,
       0.42308, 0.44615, 0.46923, 0.49231, 0.51538, 0.53846, 0.56154,
       0.58462, 0.60769, 0.63077, 0.65385, 0.67692, 0.7    , 0.72308,
       0.74615, 0.76923, 0.79231, 0.81538, 0.83846, 0.86154, 0.88462,
       0.90769, 0.93077, 0.95385, 0.97692, 1.     ]),
       max_iter=2000, n_alphas=100, n_jobs=1, normalize=True,
       positive=False, precompute='auto', random_state=None,
       selection='cyclic', tol=0.0001, verbose=0)
```

In [46]:

```
print ("optimal l1_ratio: %.3f"%cv_enet.l1_ratio_)
print ("optimal alpha: %.6f"%cv_enet.alpha_)
print ("number of iterations : %d"%cv_enet.n_iter_)
```

```
optimal l1_ratio: 0.100
optimal alpha: 0.000009
number of iterations : 42
```

The l1 ratio is 0.1, which means ridge regression accounts majority part in the elastic net. This is reasonable because we may have collinearity and ridge solve collinearity issue better than lasso regressor does.

In [47]:

```
# train elastic net model
net_model = ElasticNet(l1_ratio = cv_enet.l1_ratio_, alpha = cv_enet.alpha_, max_iter = cv_enet.n_iter_
, fit_intercept = True, normalize = True)
```

In [48]:

```
net_model.fit(x_train, y_train)
```

Out[48]:

```
ElasticNet(alpha=9.335318823677131e-06, copy_X=True, fit_intercept=True,
      l1_ratio=0.1, max_iter=42, normalize=True, positive=False,
      precompute=False, random_state=None, selection='cyclic', tol=0.0001,
      warm_start=False)
```

In [49]:

```
# MSE
y_pred_net = net_model.predict(x_test)
print("Elastic Net MSE: {}".format(mean_squared_error(y_test, y_pred_net)))

# R squared
print ("Elastic Net R-Squared: {}".format(r2_score(y_test, y_pred_net)))
```

```
Elastic Net MSE: 0.000778326776968653
Elastic Net R-Squared: 0.6603336168785282
```

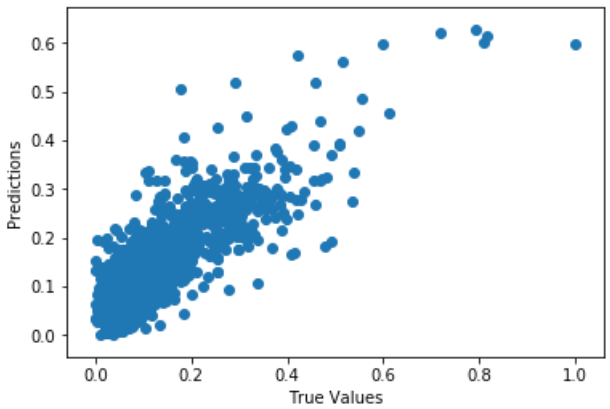
In [50]:

```
plt.scatter(np.sqrt(y_test), np.sqrt(y_pred_net))
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: RuntimeWarning: invalid value encounter
ed in sqrt
  """Entry point for launching an IPython kernel.
```

Out[50]:

```
Text(0,0.5,'Predictions')
```




## Feature Importance from Elastic Net

In [51]:

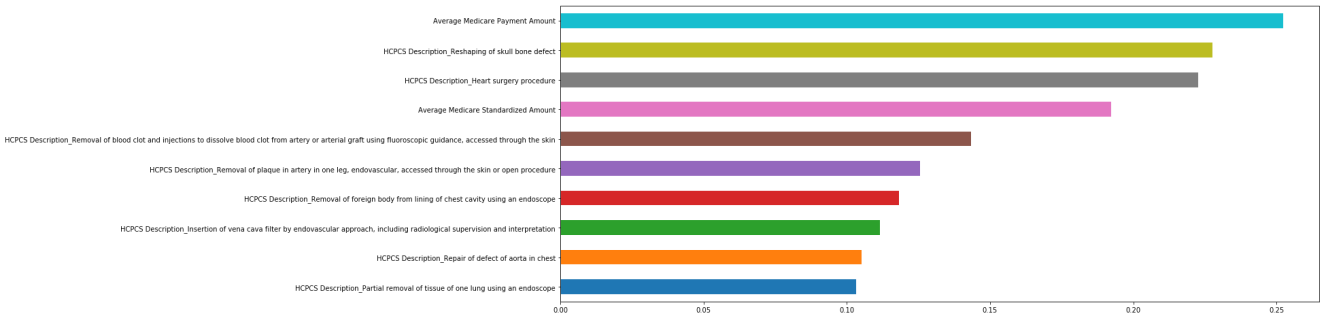
```
def get_feature_importance(x_train, model):

    feature_importance = pd.Series(index = x_train.columns, data = np.abs(model.coef_))
    selected_features = (feature_importance>0).sum()
    print('{0:d} features, reduction of {1:2.2f}%'.format(
        selected_features,(1-selected_features/len(feature_importance))*100))
    feature_importance.sort_values().tail(10).plot(kind = 'barh', figsize = (20,8))
```

In [52]:

```
get_feature_importance(x_train, net_model)
```

```
get_feature_importance(x_train, net_model)
```

```
684 features, reduction of 44.70%
```




# Random Forest Training and Prediction

In [53]:

```
from sklearn.ensemble import RandomForestRegressor

# random forest model
rf = RandomForestRegressor(n_estimators = 200, min_samples_leaf = 2, min_samples_split = 15)

# train random forest regressor
rf_model = rf.fit(x_train, y_train)

# Use the forest's predict method on the test data
y_pred_rf = rf.predict(x_test)
```

In [54]:

```
# MSE
print("Random Forest MSE: {}".format(mean_squared_error(y_test, y_pred_rf)))

# R squared
print ("Random Forest R-Squared: {}".format(r2_score(y_test, y_pred_rf)))
```

```
Random Forest MSE: 0.0007623087960162973
Random Forest R-Squared: 0.6673239579743151
```

## Feature Importance from Random Forest

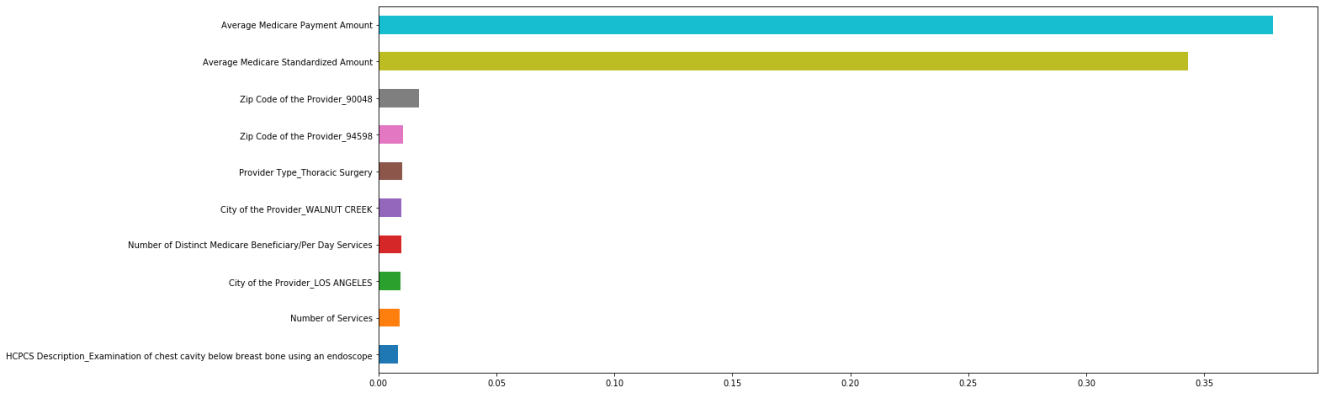
In [55]:

```
# feature importance
feature_importance = pd.Series(index = x_train.columns, data = np.abs(rf_model.feature_importances_))
selected_features = (feature_importance>0).sum()
print('{0:d} features, reduction of {1:2.2f}%'.format(
    selected_features,(1-selected_features/len(feature_importance))*100))
feature_importance.sort_values().tail(10).plot(kind = 'barh', figsize = (20,8))
```

```
745 features, reduction of 39.77%
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1dda6710>
```



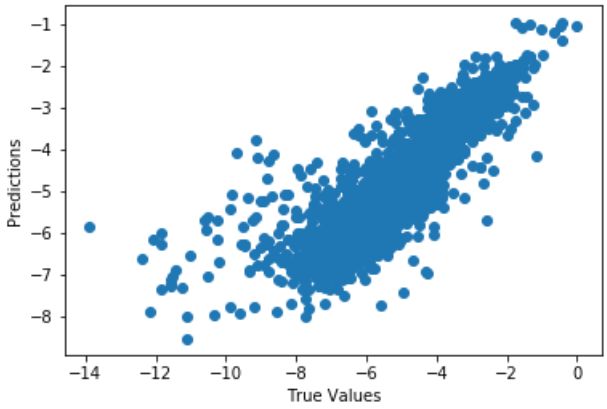

In [56]:

```
# scatter plot
plt.scatter(np.log(y_test), np.log(y_pred_rf))
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encounte
red in log
  """Entry point for launching an IPython kernel.
```

Out[56]:

```
Text(0,0.5,'Predictions')
```




# eXtreme Gradient Boosting

In [57]:

```
#!pip install xgboost
from xgboost import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

In [58]:

```
print (x_train.shape)
print(y_train.shape)
```

```
(9180, 1237)
(9180,)
```

In [59]:

```
# train xgboosting regressor
params = {'min_child_weight':[4,5], 'gamma':[i/10.0 for i in range(3,6)],  'subsample':[i/10.0 for i in
range(6,11)],
'colsample_bytree':[i/10.0 for i in range(6,11)], 'max_depth': [2,3,4]}
xgb = XGBRegressor(colsample_bytree=0.2, gamma=0.0,
                                learning_rate=0.05, max_depth=6,
                                min_child_weight=1.5, n_estimators=200,
                                reg_alpha=0.9, reg_lambda=0.6,
                                subsample=0.2,seed=42)
xgb_grid = GridSearchCV(xgb, params)
xgb.fit(x_train,y_train)
y_pred_xgb = xgb.predict(x_test)
```

In [60]:

```
# MSE
print("Xgboost MSE: {}".format(mean_squared_error(y_test, y_pred_xgb)))

# R squared
print ("Xgboost R-Squared: {}".format(r2_score(y_test, y_pred_xgb)))
```

```
Xgboost MSE: 0.0008630282118585724
Xgboost R-Squared: 0.6233694125294127
```

In [61]:

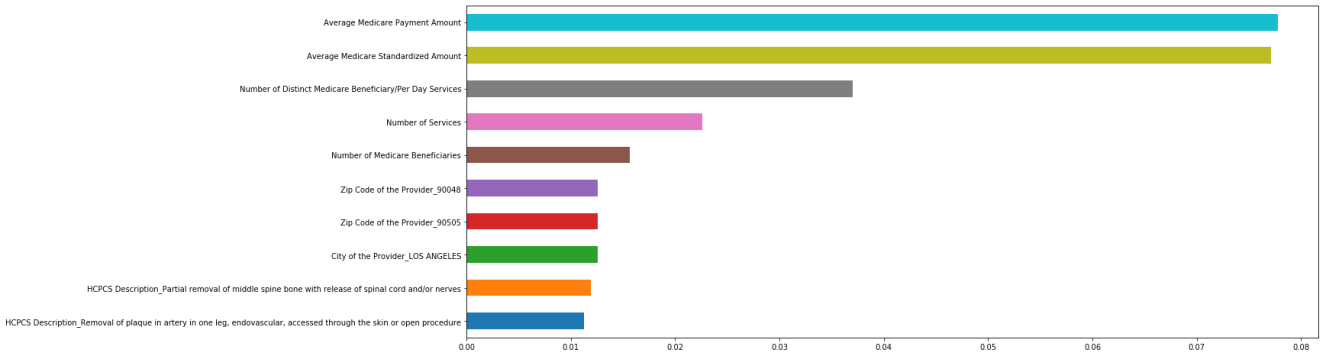
```
# feature importance
feature_importance = pd.Series(index = x_train.columns, data = np.abs(xgb.feature_importances_))
selected_features = (feature_importance>0).sum()
print('{0:d} features, reduction of {1:2.2f}%'.format(
```

```
    selected_features,(1-selected_features/len(feature_importance))*100))
feature_importance.sort_values().tail(10).plot(kind = 'barh', figsize = (20,8))
```

```
318 features, reduction of 74.29%
```

Out[61]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a18d097b8>
```

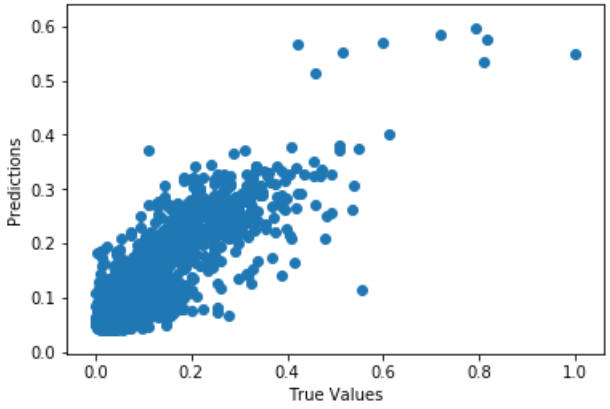



In [62]:

```
# plot prediciton values vs true values
plt.scatter(np.sqrt(y_test), np.sqrt(y_pred_xgb))
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[62]:

```
Text(0,0.5,'Predictions')
```




In [63]:

```
# import MLP classifier
from sklearn.neural_network import MLPRegressor

# initialize MLP classifier
mlp = MLPRegressor(hidden_layer_sizes=(30,30,30))

# train MLP classifier
mlp.fit(x_train,y_train)

# predict
y_pred_mlp = mlp.predict(x_test)
```

In [64]:

```
# MSE
print("Neural Network MSE: {}".format(mean_squared_error(y_test, y_pred_mlp)))

# R squared
print ("Neural Network R-Squared: {}".format(r2_score(y_test, y_pred_mlp)))
```

```
Neural Network MSE: 0.000680317208914271
Neural Network R-Squared: 0.703105568811091
```

In [65]:

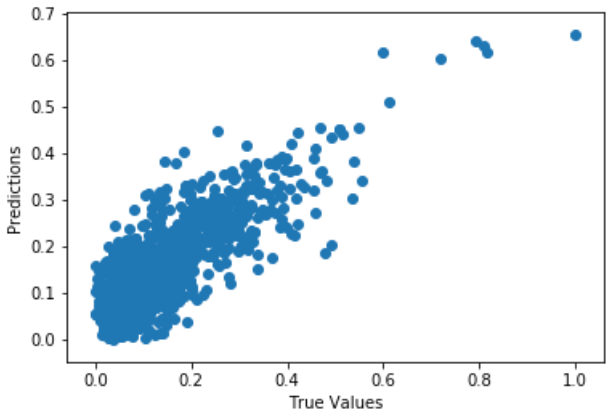
```
# plot prediciton values vs true values
plt.scatter(np.sqrt(y_test), np.sqrt(y_pred_mlp))
```

```
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: invalid value encounter
ed in sqrt
```

Out[65]:

```
Text(0,0.5,'Predictions')
```




# Stacking

In [116]:

```
from sklearn.model_selection import StratifiedKFold
from xgboost.sklearn import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn.cross_validation import KFold
import plotly.graph_objs as go
import scipy.optimize as spo
import plotly.offline as py
py.init_notebook_mode(connected=True)
```

In [67]:

```
# x's dimension and y's dimension
print (x_train.shape)
print(y_train.shape)
```

```
(9180, 1237)
(9180,)
```

In [68]:

```
ntrain = x_train.shape[0]
ntest = x_test.shape[0]

SEED = 43
kf = KFold(ntrain, n_folds= 10, random_state=SEED)
```

In [72]:

```
def get_train_test_per_model(clf, x_train, y_train, x_test):
    train = np.zeros((ntrain,))
    test = np.zeros((ntest,))
    test_skf = np.empty((10, ntest))

    for i, (train_index, test_index) in enumerate(kf):
        x_tr = x_train.iloc[train_index]
        y_tr = y_train.iloc[train_index]
        x_te = x_train.iloc[test_index]

        clf.fit(x_tr, y_tr)

        train[test_index] = clf.predict(x_te)
        test_skf[i, :] = clf.predict(x_test)
```

```
    test[:] = test_skf.mean(axis=0)
    return train.reshape(-1, 1), test.reshape(-1, 1)
```

In [73]:

```
# train all models
rf_train_stack, rf_test_stack = get_train_test_per_model(rf, x_train, y_train, x_test) # Random Forest
xgb_train_stack, xgb_test_stack = get_train_test_per_model(xgb,x_train, y_train, x_test) # Xgboost
mlp_train_stack, mlp_test_stack = get_train_test_per_model(mlp, x_train, y_train, x_test) # Neural Neto
rk
```

In [74]:

```
predictions_train_stack = pd.DataFrame( {'RandomForest': rf_train_stack.ravel(),
      'GradientBoost': xgb_train_stack.ravel(),
      'NeuralNetowrk': mlp_train_stack.ravel()
    })
```

In [75]:

```
# referred from: https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python
heatmap = [
    go.Heatmap(
        z= predictions_train_stack.astype(float).corr().values ,
        x= predictions_train_stack.columns.values,
        y= predictions_train_stack.columns.values,
          colorscale='YlGnBu',
            showscale=True,
            reversescale = True
    )
]
py.iplot(heatmap, filename='labeled-heatmap')
```

In [119]:

```
### Do combinations of models (stacking)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
from sklearn import cross_validation

#define parameter for stacking
model_list = ['XGBoost','MLP','RandomForest']
model_len = len(model_list)
```

```
opt_dict = {}

# Change df to np.array
X1 = X
X2 = X
X3 = X
y = Y

pred= np.zeros((3,9180))
pred_xgb = pd.Series([])
pred_mlp = pd.Series([])
pred_rf = pd.Series([])
# Do 10-fold train & evaluate
cv = cross_validation.KFold(len(x_train), n_folds=10, shuffle=False, random_state=None)

for traincv, testcv in cv:

    X_train1, X_test1= (X1.iloc[traincv], X1.iloc[testcv])
    X_train2, X_test2= (X2.iloc[traincv], X2.iloc[testcv])
    X_train3, X_test3 = (X3.iloc[traincv], X3.iloc[testcv])
    y_train1, y_test1= (y.iloc[traincv], y.iloc[testcv])

    # XGB
    xgb= XGBRegressor(colsample_bytree=0.2, gamma=0.0,
                            learning_rate=0.05, max_depth=8,
                            min_child_weight=1.5, n_estimators=200,
                            reg_alpha=0.9, reg_lambda=0.6,
                            subsample=0.2,seed=42)
    xgb.fit(X_train1,y_train1)
    pred_xgb_temp = xgb.predict(X_test1)
    pred_xgb = pd.concat([pred_xgb,pd.Series(pred_xgb_temp)], axis = 1)

    # MLP classifier
    mlp = MLPRegressor(hidden_layer_sizes=(30,30,30))
    mlp.fit(X_train2,y_train1)
    pred_mlp_temp = mlp.predict(X_test2)
    pred_mlp = pd.concat([pred_mlp,pd.Series(pred_mlp_temp)], axis = 1)

    #random forest regressor
    rf = RandomForestRegressor(n_estimators = 200, min_samples_leaf = 2, min_samples_split = 15)
    rf_model = rf.fit(X_train3, y_train1)
    pred_rf_temp = rf.predict(X_test1)
    pred_rf = pd.concat([pred_rf,pd.Series(pred_rf_temp)], axis = 1)
```

In [123]:

```
#average result from cross validation by row
xgb_result = pred_xgb.mean(axis = 1)
mlp_result = pred_mlp.mean(axis = 1)
rf_result = pred_rf.mean(axis = 1)

#combine three model
pred_combine = pd.concat([xgb_result, mlp_result, rf_result], axis = 1)
pred_combine.columns = ['xgb','mlp','rf']
```

In [151]:

```
def get_optimal_MSE(allocs,pred_df, eps=1e-15):
    pred_df = np.clip(pred_df, eps, 1 - eps)
    pred_df_w = np.sum(pred_df*allocs,axis = 1)
    return mean_squared_error(y_test1, pred_df_w)
```

In [152]:

```
model_list = ['XGBoost','MLP','RandomForest']
model_len = len(model_list)
init_vals= [1.0 / model_len] * model_len
cons = ({'type': 'ineq', 'fun': lambda x: 1.0-np.sum(x)})
bnds = [(0.0, 1.0)] * model_len

#optimized allocations
opts = spo.minimize(get_optimal_MSE, init_vals, args = (pred_combine,),method='SLSQP', bounds=bnds, con
straints=cons, options = {'disp':True})
opt_allocs = opts.x
```

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 0.0019119187971855295
            Iterations: 1
            Function evaluations: 5
            Gradient evaluations: 1
```

In [153]:

```
# optimal wieghts for each learner
display(opt_allocs)

# optimal result
display(opts)
```

```
array([0.33333333, 0.33333333, 0.33333333])
```

```
     fun: 0.0019119187971855295
     jac: array([-2.25775148e-05,  2.54604238e-05, -6.28369744e-05])
 message: 'Optimization terminated successfully.'
    nfev: 5
     nit: 1
    njev: 1
  status: 0
 success: True
       x: array([0.33333333, 0.33333333, 0.33333333])
```

In [154]:

```
y_stack_pred = 0.33333333*mlp.predict(x_test) + 0.33333333*xgb.predict(x_test) + 0.33333333*rf.predict
(x_test)
```

In [155]:

```
# MSE
print("Stacking MSE: {}".format(mean_squared_error(y_test, y_stack_pred)))

# R squared
print ("Stacking R-Squared: {}".format(r2_score(y_test, y_stack_pred)))
```

```
Stacking MSE: 0.000732264623199042
Stacking R-Squared: 0.6804354116936113
```

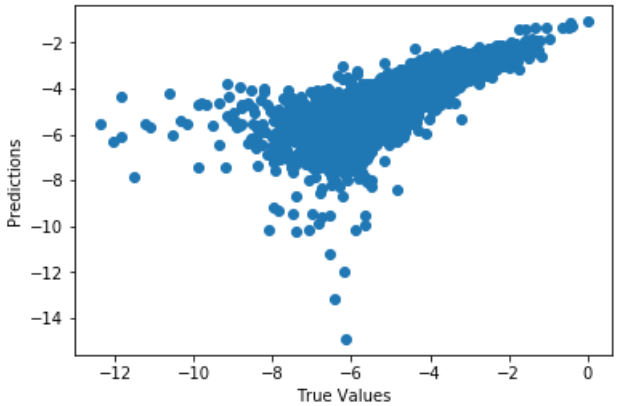
In [158]:

```
# plot prediciton values vs true values
plt.scatter(np.log(y_test), np.log(y_stack_pred))
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning:

divide by zero encountered in log

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning:

invalid value encountered in log
```

Out[158]:

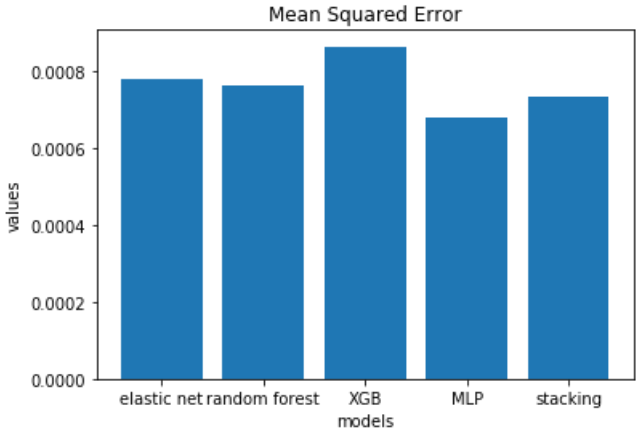
```
Text(0,0.5,'Predictions')
```



## Comparison between Models

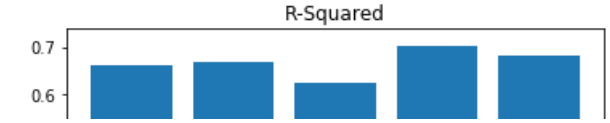
In [162]:

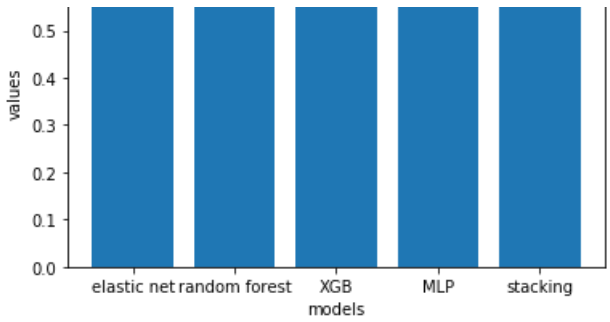
```
# Mean Squared Error
model_mse = [
              mean_squared_error(y_test, y_pred_net),\
              mean_squared_error(y_test, y_pred_rf),\
              mean_squared_error(y_test, y_pred_xgb),\
              mean_squared_error(y_test, y_pred_mlp),\
              mean_squared_error(y_test, y_stack_pred)]
bar_cat = ('elastic net','random forest', 'XGB','MLP', 'stacking')
y_pos = np.arange(len(bar_cat))

# Create bars
plt.bar(y_pos, model_mse)

# Create names on the x-axis
plt.xticks(y_pos, bar_cat)

# Add title and axis names
plt.title('Mean Squared Error')
plt.xlabel('models')
plt.ylabel('values')

# Show graphic
plt.show()
```




In [165]:

```
# R-Squared
model_r_squared = [
              r2_score(y_test, y_pred_net),\
              r2_score(y_test, y_pred_rf),\
              r2_score(y_test, y_pred_xgb),\
              r2_score(y_test, y_pred_mlp),\
              r2_score(y_test, y_stack_pred)]
bar_cat_r = ('elastic net','random forest', 'XGB','MLP', 'stacking')
y_pos_r = np.arange(len(bar_cat_r))

# Create bars
plt.bar(y_pos_r, model_r_squared)

# Names on x-axis
plt.xticks(y_pos_r, bar_cat_r)

# title and axis names
plt.title('R-Squared')
plt.xlabel('models')
plt.ylabel('values')

# Show graphic
plt.show()
```






The result showded that in addtion to baseline model linear regression, the rest five models significantly reduced mean squared error, at the same time, R-Sqaured significantly were increased among the rest five models. Among the five models, MLP model showed the lowest mean squared error and highest R-Squared. It is intersted to note that MLP perform the same or even slightly better than the stacking model.

Noted that stacking model was built based on the mean value from cross validation and the training set and testing set were resampled, it is unlikely that stacking model did not perform the best because of overfitting issue. Instead, it may be explained by the fact that the model may be suboptimal. It can be inferred that neural network performed the best because random forest regressor and xgboosting regressor did not fit to the optimal parameters, grid search can be implemented to get the optimal value and increase performace in stacking model.

Also, noted that random forest regressor also outperformed xgbkosting regressor, it is very likely that xgboosting regressor did not for to the optimal values. It is also possible that neural newtwork may better predict numerical value compared to xgboosting and random forest in this case. For the future work, optimal parameters should be achieved. In addition to MSE and R-Squared, other analysis can be also conducted to accurately define what is the "best model".