Setup

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
import pandas as pd
import json
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import statsmodels.api as sm

pd.options.display.max_columns = 40


f = open('/content/drive/MyDrive/ColabForHouzz/megamerge.txt', "r")

filedata = f.read()

filedata = filedata.replace("\n", "")
filedata = filedata.replace("][][", ", ")
filedata = filedata.replace("][", ", ")

filedata = filedata.replace(", , ", ", ")

k = json.loads(filedata)


df = pd.read_json(filedata)

print('un ', df['Product ID'].unique().shape)

df.drop_duplicates(inplace=True,keep='last')

print('df shape ', df.shape)

df = df.dropna(subset=['Size/Weight'])
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing i
  import pandas.util.testing as tm
un  (5185,)
df shape  (6512, 18)
```

```
df.head(2)
```

1 to 2 of 2 entries Filter

Price	Title ▲	Reviews	Product ID	Manufactured By	Sold By	Size/Weight	Color	
$1,686	Modern Aspen White Microfiber Leather Sofa	10 Reviews	14231908.0	Zuri Furniture	Zuri Furniture	W 80" / D 42" / H 32" / 125 lb.	White	NaN
	Sorrento Retro Upholstered Leather					W 70.59" /		

Feature Engineering and Filtering

Like what you see? Visit the data table notebook to learn more about interactive tables

```
tempDf = df.copy()

tempDf = tempDf.fillna(0)
#get price


#Feature Extraction

tempDf['Title'] = tempDf['Title'].str.lower()

tempDf['Materials'] = tempDf['Materials'].str.lower()

tempDf['Price'] = tempDf['Price'].str.replace(',', '')
tempDf['Price'] = tempDf['Price'].str.replace('$', '').astype(float)

#get width (?<=W )(\d{1,3})
tempDf['width'] = tempDf['Size/Weight'].str.extract(r'(?<=W )(\d{1,3})').astype(float)

#get depth (?<=D )(\d{1,3})
tempDf['depth'] = tempDf['Size/Weight'].str.extract(r'(?<=D )(\d{1,3})').astype(float)

#get height (?<=H )(\d{1,3})
tempDf['height'] = tempDf['Size/Weight'].str.extract(r'(?<=H )(\d{1,3})').astype(float)

#get weight (?<=\/ )(\d+)
tempDf['weight'] = tempDf['Size/Weight'].str.extract(r'(?<=\/ )(\d+)').astype(float)

#convert to numeric
#tempDf[['width','depth','height','weight','Price']] = tempDf[['width','depth','height','weight','Price']].apply(p


#calculate volume
tempDf['volume'] = tempDf['width'] * tempDf['depth'] * tempDf['height']

#Calculate density
tempDf['density'] = tempDf['weight'] / tempDf['volume']

#Get Reviews
tempDf['ReviewAmount'] = tempDf['Reviews'].str.extract(r'(\d+)',expand=False).astype(float)

tempDf['ReviewAmount'] = tempDf['ReviewAmount'].fillna(0)
```

```
tempDf = tempDf.dropna(subset=['volume','density','Price'])

tempDf['lengthofmaterials'] = tempDf['Materials'].str.len()
tempDf['lengthoftitle'] = tempDf['Title'].str.len()


#drop lower used cataorgiers?

a = tempDf.groupby('Category').count()
a = a[a['Price'] > a['Price'].mean()]

tempDf = tempDf[tempDf['Category'].isin(a.index)]

#drop lower used styles?

a = tempDf.groupby('Style').count()
a = a[a['Price'] > a['Price'].mean()]

tempDf = tempDf[tempDf['Style'].isin(a.index)]

#drop zero density, means widht, height, length or weight is zero

tempDf = tempDf[tempDf['width'] != 0]

tempDf = tempDf[tempDf['lengthoftitle'] > 5]

#everything

tempDf['everything'] = tempDf['Manufactured By'].astype(str)  + tempDf['Sold By'].astype(str)  + tempDf['Size/Weig

def myfunc(n):
  c =0
  for word in n['Title'].split():
    word = word.lower()
    if word in n['everything'].lower():
      c=c+1
  return c


tempDf['numt'] = tempDf.apply(myfunc,axis=1)

tempDf['numtitlewords'] = tempDf['Title'].str.count(' ')

tempDf['uniquewordsPercent'] =(tempDf['numtitlewords'] + 1 - tempDf['numt'] ) / tempDf['numtitlewords']
tempDf['uniquewords'] =(tempDf['numtitlewords'] + 1 - tempDf['numt'] )
#mention of mateirals in title

def myfunc2(n):
  c =0
  for word in n['Title'].split():
    word = word.lower()
    if word in n['Materials'].lower():
      c=c+1
  return c
tempDf['Materials'] = tempDf['Materials'].astype(str)
tempDf['mention_material'] = tempDf.apply(myfunc2,axis=1)


#Cataogry Encoding

#f = f.join(pd.get_dummies(f.Style, prefix='Style_'))
```

```
tempDf = tempDf.join(pd.get_dummies(tempDf['Category'],prefix='Type_'))

tempDf = tempDf.join(pd.get_dummies(tempDf['Style'],prefix='Style'))

#drop not needed columns

tempDf = tempDf.drop(columns=['Size','Weight','Product ID'])

#encode Assmebly yes no as 1 ,0

tempDf['Assembly Required'] = tempDf['Assembly Required'].str.replace('Yes', '1')
tempDf['Assembly Required'] = tempDf['Assembly Required'].str.replace('No', '0')
tempDf = tempDf.fillna(0)
tempDf['Assembly Required'] = tempDf['Assembly Required'].astype(int)

#Encode commerical grade
tempDf['Commercial-grade'] = tempDf['Commercial-grade'].str.replace('Yes','1')
tempDf = tempDf.fillna(0)
tempDf['Commercial-grade'] = tempDf['Commercial-grade'].astype(int)

tempNum = tempDf[tempDf.describe().columns]

#price * reviews number as popularity matrix? confirmed sales
#tempDf['confirmedSales'] = tempDf['ReviewAmount'] * tempDf['Price']

# tempDf.replace([np.inf, -np.inf], np.nan, inplace=True)
# tempDf = tempDf.dropna()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: FutureWarning: The default value of regex wi

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:45: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:46: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
```

Testing Part Of Speech Word Tokenizer

```
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

def wordFunc(n):
  return nltk.pos_tag(word_tokenize(n['Title']))

def countNoun(n):
  c = 0
  for z in n['pos']:
    if(z[1] == 'JJ'): ##is noun
      c = c + 1
  return c
```

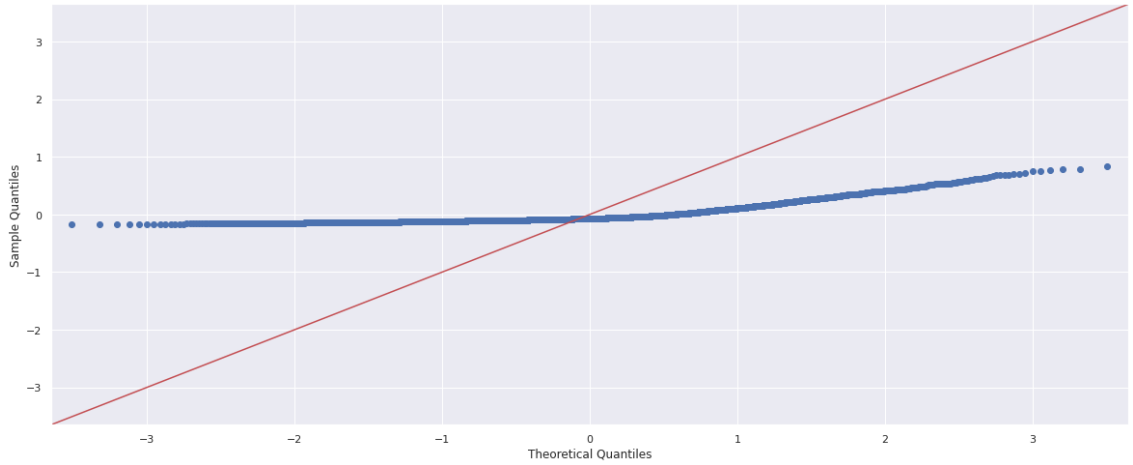
```
k = pd.DataFrame()

#k['tokens'] = word_tokenize(a['Title'].str)
#k['pos'] = nltk.pos_tag(text)

tempDf['pos'] = tempDf.apply(wordFunc,axis=1)
tempDf['jjcount'] = tempDf.apply(countNoun,axis=1)
tempDf['jjpercent'] = tempDf['nncount'] / (tempDf['numtitlewords'] +1)
```

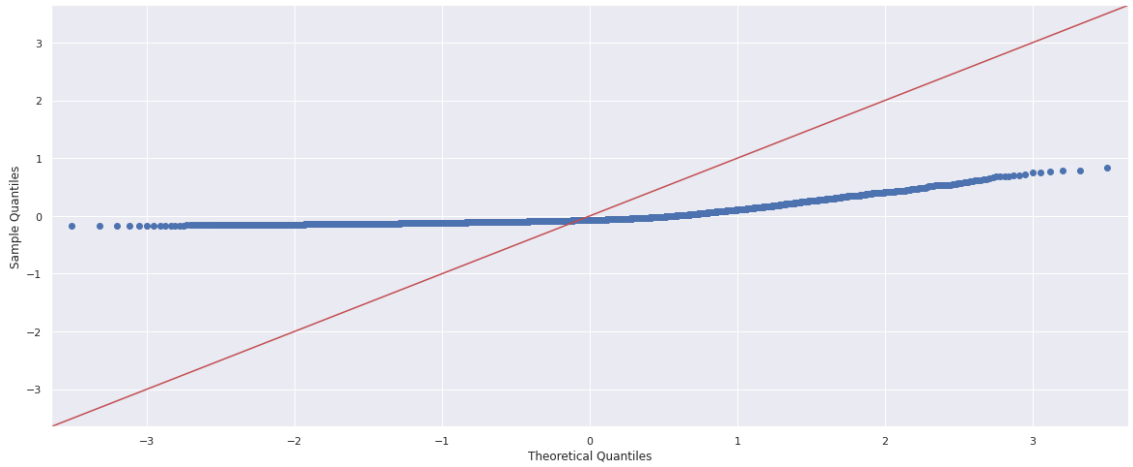
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Data Exploration

QQPLOT

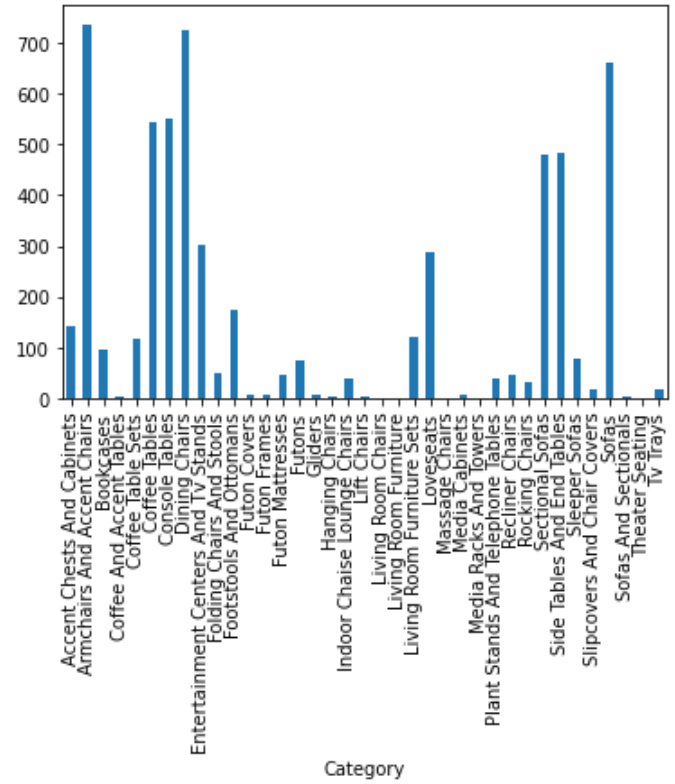
```
#from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
normi = preprocessing.Normalizer()

scaled = scaler.fit_transform(tempNum)
#print(type(scaled[:,0]))

scaled = normi.fit_transform(scaled)
sm.qqplot(scaled[:,0], line ='45')
```





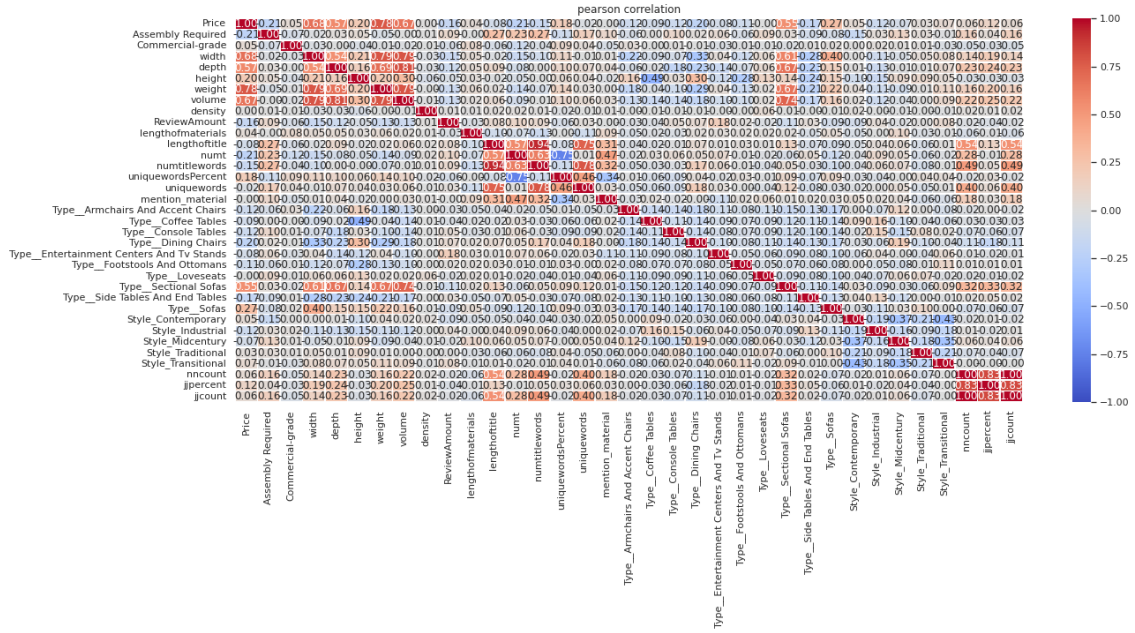

```
print('number of categories: ', len(df['Category'].unique()))
df.groupby(by='Category').count()['Price'].plot(kind='bar')
```

```
number of categories:  37
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc3af05310>
```



```
#Correlation Matrix
corr_matrix = tempDf[tempDf.describe().columns]
for col in corr_matrix.columns:
    if corr_matrix[col].dtype == "O":
        corr_matrix[col] = corr_matrix[col].factorize(sort=True)[0]
corr_matrix = corr_matrix.corr(method="pearson")
sns.set(rc = {'figure.figsize':(20,8)})
sns.heatmap(corr_matrix, vmin=-1., vmax=1., annot=True, fmt='.2f', cmap="coolwarm", cbar=True, linewidths=0.5)
plt.title("pearson correlation")
```

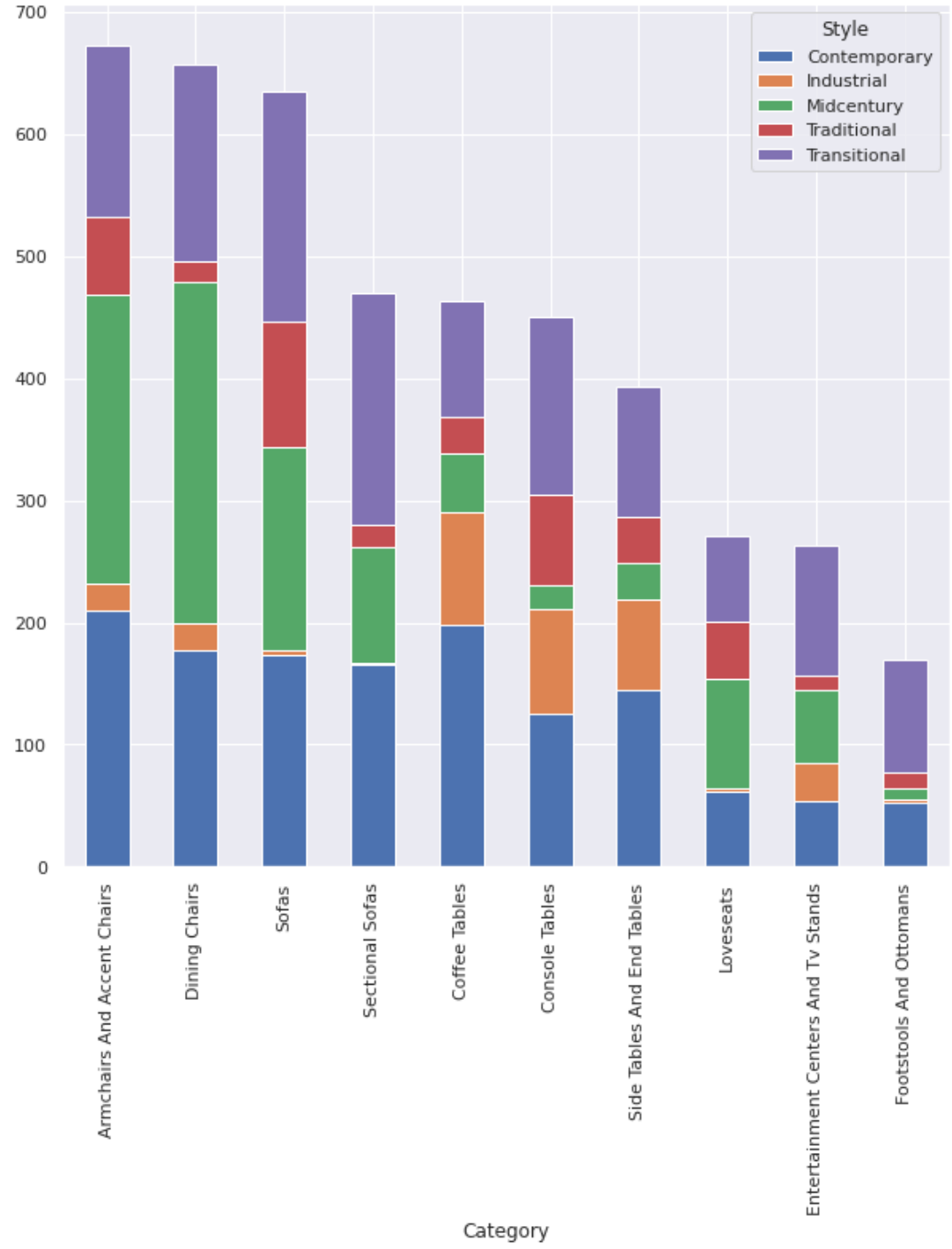
```
Text(0.5, 1.0, 'pearson correlation')
```




```
eazyDf = tempDf[['Category','Price','Style']]
e = eazyDf.groupby(['Category','Style']).size().reset_index(name='Count')
e = pd.pivot(e, index='Category',columns='Style')

e.columns = e.columns.droplevel()


e['sum'] = e.sum(axis=1)
e = e.sort_values(by=['sum'],ascending=False )
e = e.drop(columns=['sum'])
```

```
e.plot(kind='bar',figsize=(10,10), stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc3ad58790>
```

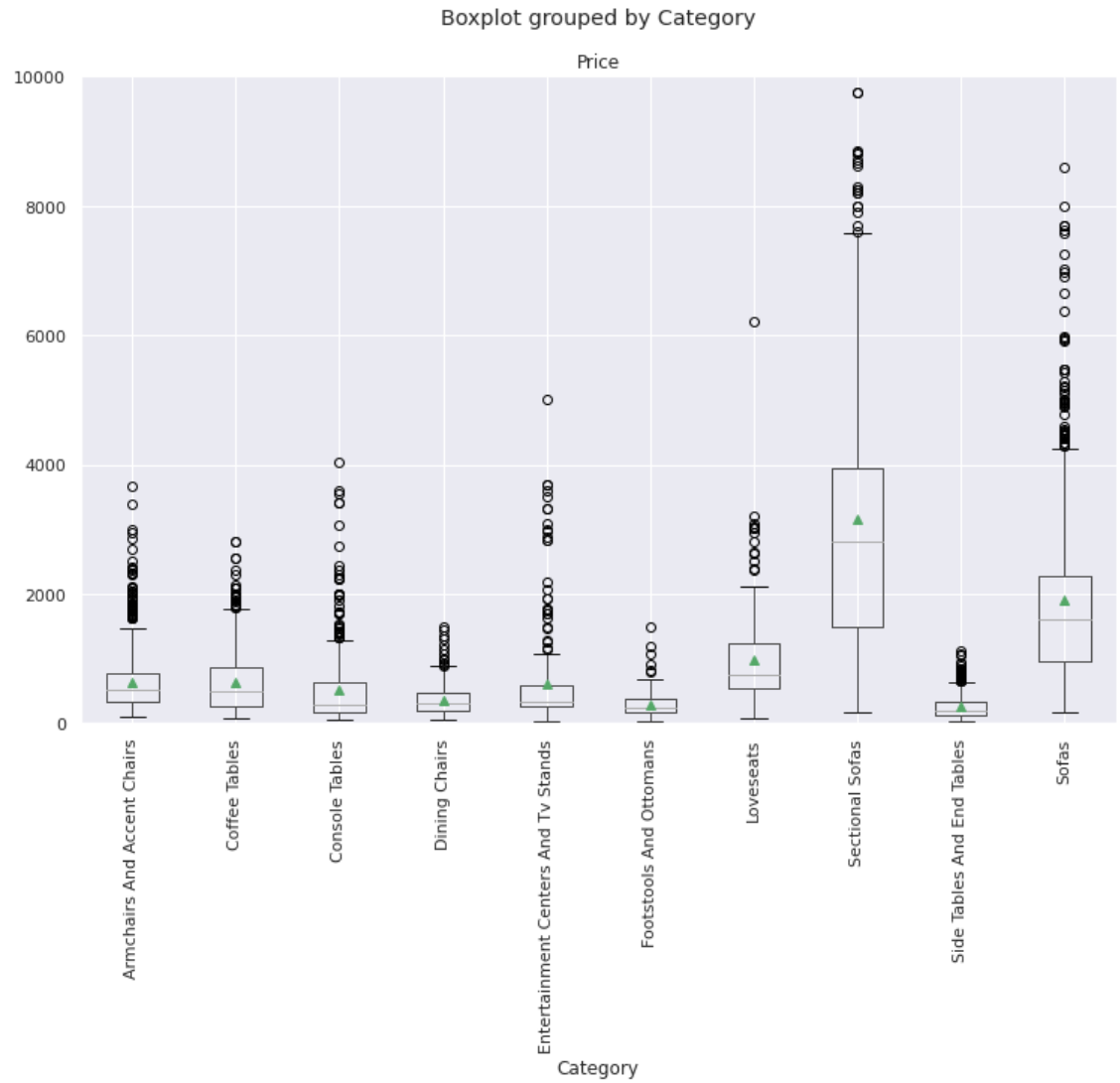



```
tempDf.groupby("Category")['density'].describe().sort_values(by=['count'],ascending=False)
```

Category	count	mean	std	min	25%	50%	75%	
Armchairs And Accent Chairs	673.0	0.016369	0.120895	0.000077	0.001139	0.001417	0.001796	1.

```
#kdf = tempDf[tempDf['Category'] == 'Coffee Tables']

kdf = tempDf

b = kdf.boxplot('Price', by='Category', figsize=(12, 8), rot=90,showmeans=True)
plt.ylim(0,10000)
#plt.title('Filtered to Sofas')
```

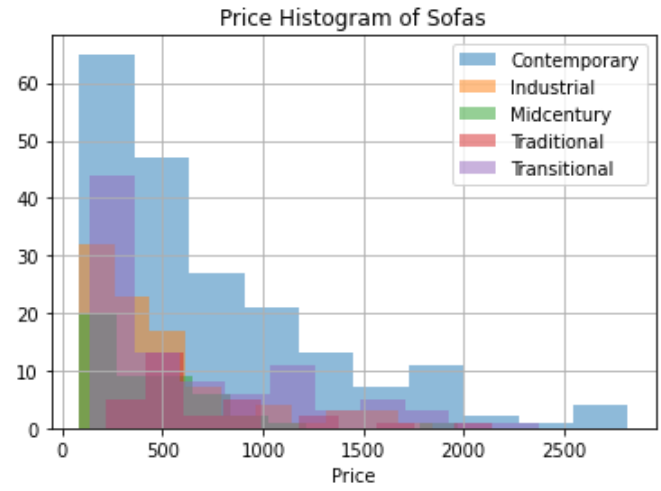
```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleD
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
(0.0, 10000.0)
```




```
kdf.groupby(by='Style')['Price'].hist(alpha=0.5,legend=True)
plt.xlabel('Price')
```

```
plt.title('Price Histogram of Sofas')
#kdf.hist(column='Price',by='Style',bins=10)
```

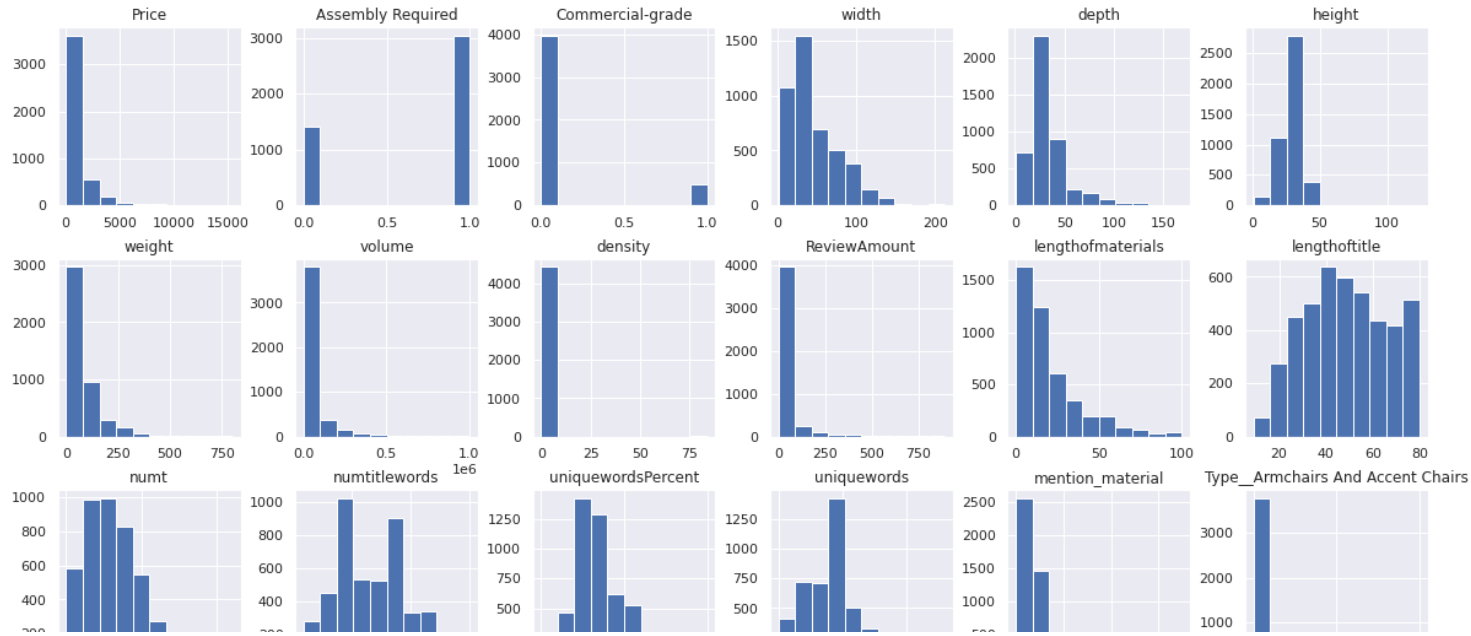
```
Text(0.5, 1.0, 'Price Histogram of Sofas')
```



Histogram

```
tempDf.hist(figsize=(20,20))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe9a03d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe85dfd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe784110>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe7a0610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe744b10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe77afd0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe73c5d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe6f1a10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe6f1a50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe6b4050>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe620990>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe5d7e90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe59b3d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe5528d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe509dd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe4ce310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe482810>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe4b8d10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe47c250>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe430750>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe3e8c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe3ac190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe363690>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe319b90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe2de0d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe2935d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe24aad0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe274b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe1c2510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe1fba10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe1aff10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe174450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe129950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe0e0e50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe0a3390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fbe05a890>]],
      dtype=object)
```




```
snsDf = tempDf[['Price','weight','width','height','depth','ReviewAmount','lengthofmaterials','lengthoftitle','numt
```

```
pd.plotting.scatter_matrix(snsDf)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc923d410>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8d7cf90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8ce3f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8c6a650>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8c80d10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8c35190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8be8710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8b9eb50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8b9eb90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8b631d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8a4cad0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8a03fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc89c6550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc89fba50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc89b1f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc89774d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc892c9d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc88e2ed0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc88a7410>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc885b910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8812e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc87d4350>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc878d850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8743d50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8706290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc873d790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc86f1c90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc86b61d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc866b6d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8623bd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc85e7110>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc855d610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8513b10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc84cbfd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc848c550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8443a50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8479f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc843d490>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc83f3990>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc83a9e90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc836b3d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc82e28d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc8298dd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc821b310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc81d3810>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc818ad10>,
```

Anova Table

```
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f6fc80b56d0>
```

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols('Price ~ C(Category) + C(Style) + C(Category):C(Style)',data=tempDf).fit()
aov_table = sm.stats.anova_lm(model, typ=2)
aov_table
```

	sum_sq	df	F	PR(>F)

VIF for multicolinearity

```
from IPython.core.display import Video

from statsmodels.stats.outliers_influence import variance_inflation_factor

X_variables = tempNum.drop(columns=['Price'])

vif_data = pd.DataFrame()
vif_data["feature"] = X_variables.columns
vif_data['VIF'] = [variance_inflation_factor(X_variables.values, i) for i in range(len(X_variables.columns))]

#vif_data = vif_data[vif_data['VIF'] != math.inf]
vif_data.sort_values(by='VIF',ascending=False)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/stats/outliers_influence.py:1
  vif = 1. / (1. - r_squared_i)
```

	feature	VIF
30	Style_Transitional	inf
25	Type__Sofas	inf
23	Type__Sectional Sofas	inf
22	Type__Loveseats	inf
21	Type__Footstools And Ottomans	inf
20	Type__Entertainment Centers And Tv Stands	inf
17	Type__Coffee Tables	inf

Model Building

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn import model_selection, preprocessing, feature_selection, ensemble, linear_model, metrics, decomposit


numDf = tempDf[tempDf.describe().columns]


#numDf = tempDf[['lengthoftitle','Assembly Required','ReviewAmount','numt','density','Price','Type__Sofas']]

#numDf = numDf.drop(columns=['weight','width','volume','density','depth','height'])

numDf = tempDf[['Price','weight','volume','Assembly Required','ReviewAmount']]

numDf =(numDf-numDf.mean())/numDf.std()

numDf['Category'] = tempDf['Category']

dtf_train, dtf_test = model_selection.train_test_split(numDf, test_size=0.3, random_state=15)

X_train = dtf_train.drop(columns=["Price",'Category'], axis=1).values
y_train = dtf_train["Price"].values

X_test = dtf_test.drop(columns=["Price",'Category'], axis=1).values
y_test = dtf_test["Price"].values

model = RandomForestRegressor(min_samples_split =4)
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(model.score(X_test,y_test))

feature_names = dtf_train.drop(columns=["Price",'Category'], axis=1).columns

importances = model.feature_importances_
```

```
##Extra Cross Validation

X = numDf.drop(columns=["Price",'Category'], axis=1).values
y = numDf["Price"].values

scores = model_selection.cross_val_score(model,X,y)
scores

print('Mean R2: %.5f (%.3f)' % (np.mean(scores), np.std(scores)))

     0.7479240410622003
     Mean R2: 0.74226 (0.015)
```

Test from each cateogry

```
for cat in np.sort(tempDf['Category'].unique()):
  #print('cat: ', cat)
  newTest = dtf_test[dtf_test['Category'] == cat]
  X_test = newTest.drop(columns=["Price",'Category'], axis=1).values
  y_test = newTest["Price"].values
  predictions = model.predict(X_test)
  #print('MSE :', metrics.mean_squared_error(y_test,predictions))
  #print('MAE :', metrics.mean_absolute_percentage_error(y_test,predictions))
  #print("RMSE : ", metrics.mean_squared_error(y_test,predictions))
  print(model.score(X_test,y_test))

     0.3369165488706969
     0.21181279285371157
     0.49204336228664236
     0.16388771881108533
     0.8483752956111448
     -0.16734053538607752
     0.021605972957003394
     0.29709827744102346
     0.1489265326426219
     0.5351444716371143
```

Lasso Regression

```
from sklearn.linear_model import LassoCV
from sklearn.datasets import make_regression
from sklearn import metrics


model = LassoCV(cv=5, random_state=42).fit(X_train, y_train)

predictions = model.predict(X_test)

print('Regular R2 alpha: ', model.alpha_)
print(model.score(X_test,y_test))
print('MSE : ' ,metrics.mean_squared_error(y_test,predictions))
print('MAE : ' ,metrics.mean_absolute_percentage_error(y_test,predictions))




lasso =linear_model.Lasso(alpha=0.001472480754465545)
```

```
lasso.fit(X_train,y_train)

importances = np.abs(lasso.coef_)

print('lasso score: ',lasso.score(X_test,y_test))
```

```
Regular R2 alpha:  0.0007529494878879277
0.6295564419960942
MSE :  0.44177980343875783
MAE :  1.4189923429867801
lasso score:  0.6294358259860766
```

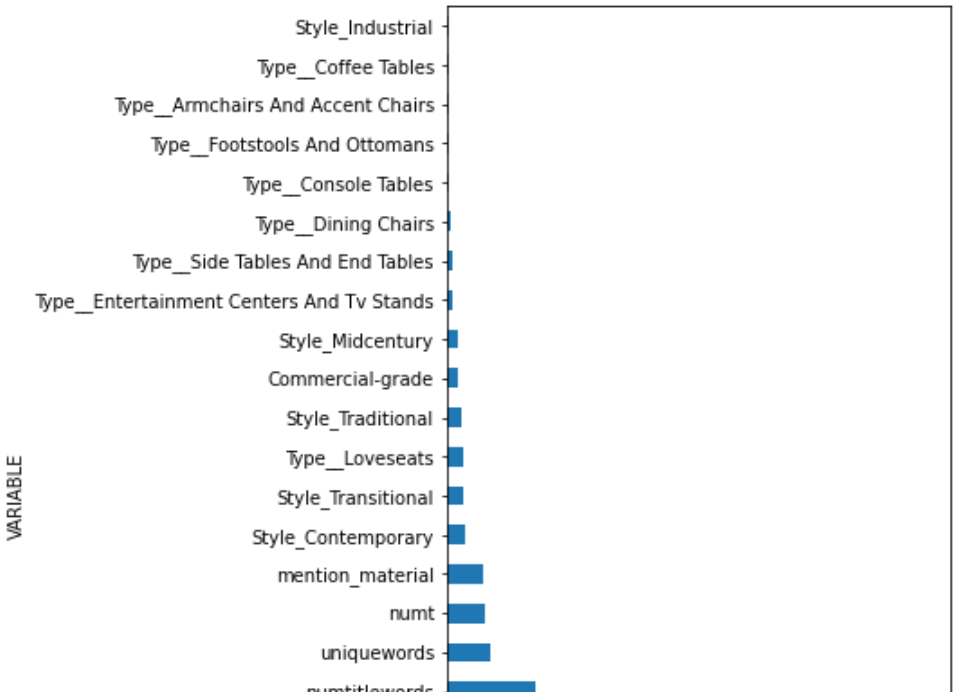
```
def r2_score_adj(estimator, X, y):

    y_pred = estimator.predict(X)

    n = len(y)
    k = np.count_nonzero(estimator.coef_ > 0)



    r2 = metrics.r2_score(y, y_pred)

    r2_adjusted = 1 - ( ( (1 - r2)*(n - 1) )/ (n - k - 1) )

    return r2_adjusted

model =linear_model.Lasso()

gcv = model_selection.GridSearchCV(estimator=model,  param_grid={'alpha': np.arange(0.009,0.02,0.00001)},scoring=r
gcv.fit(X_test,y_test)


print('Adjustd R2: ', gcv.best_params_,' ',gcv.best_score_)
print(gcv.best_params_)
```

```
Adjustd R2:  {'alpha': 0.017279999999999664}   0.08335753276571234
{'alpha': 0.017279999999999664}
```

Print Feature Importance

```
dtf_importances = pd.DataFrame({"IMPORTANCE":importances, "VARIABLE":feature_names}).sort_values("IMPORTANCE", asc
dtf_importances = dtf_importances.set_index("VARIABLE")

dtf_importances['IMPORTANCE'].plot(kind='barh',figsize=(5,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35c91c9710>
```



Train model for each cateogry

```
from sklearn.linear_model import LinearRegression



#numDf['Category'] = tempDf['Category']

for cat, df_cat in numDf.groupby('Category'):

  X = df_cat.drop(["Price",'Category'], axis=1).values

  y = df_cat["Price"].values

  model = LinearRegression()

  scores = model_selection.cross_val_score(model, X, y, scoring='r2', cv=5)
  #scores = absolute(scores)
  print('Mean R2: %.3f (%.3f) (%s)' % (np.mean(scores), np.std(scores), cat))



numDf = tempDf[tempDf.describe().columns]
numDf['Category'] = tempDf['Category']

feature_names = numDf.drop(["Price",'Category'], axis=1).columns
```

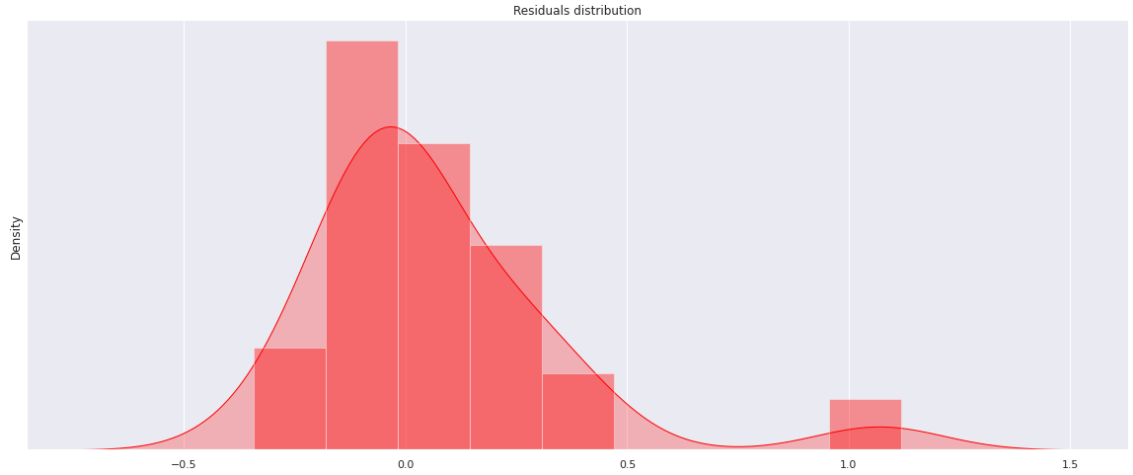
```
Mean R2: 0.397 (0.061) (Armchairs And Accent Chairs)
Mean R2: 0.535 (0.057) (Coffee Tables)
Mean R2: 0.597 (0.109) (Console Tables)
Mean R2: 0.050 (0.143) (Dining Chairs)
Mean R2: 0.619 (0.209) (Entertainment Centers And Tv Stands)
Mean R2: 0.394 (0.301) (Footstools And Ottomans)
Mean R2: 0.291 (0.205) (Loveseats)
Mean R2: 0.365 (0.118) (Sectional Sofas)
Mean R2: 0.402 (0.093) (Side Tables And End Tables)
Mean R2: 0.407 (0.110) (Sofas)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
```

Residual Analysis

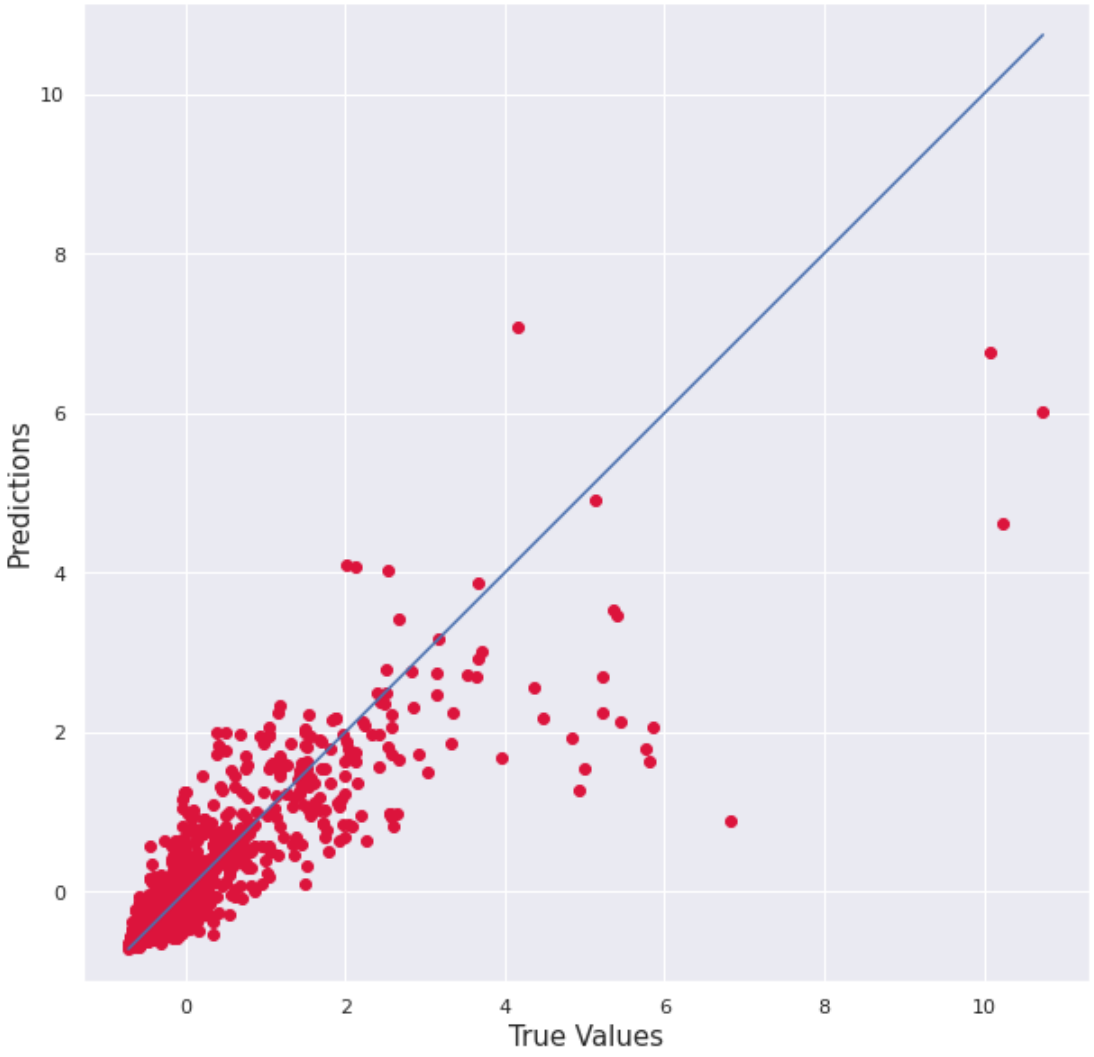
```
residuals = y_test - predictions

resDf = pd.DataFrame()
resDf['residuals'] = residuals
resDf["StdResidual"] = (resDf["residuals"] - resDf['residuals'].mean())/resDf['residuals'].std()

fig, ax = plt.subplots()
sns.distplot(residuals, color="red", hist=True, kde=True, kde_kws={"shade":True}, ax=ax)
ax.grid(True)
ax.set(yticks=[], yticklabels=[], title="Residuals distribution")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarnir
  warnings.warn(msg, FutureWarning)
```



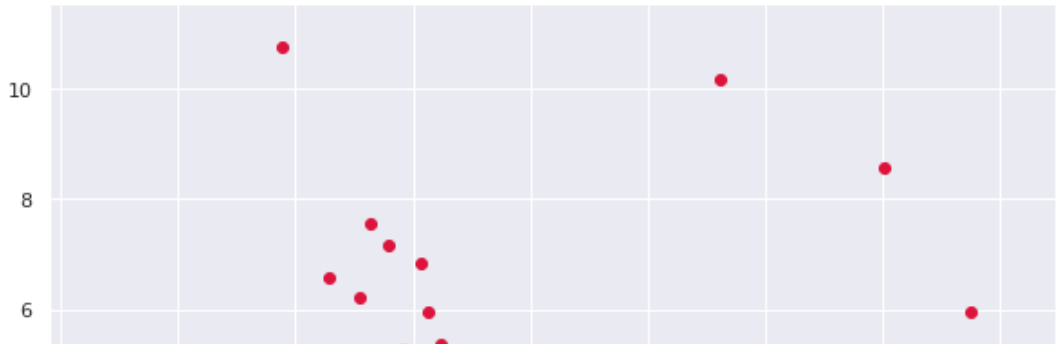

```
plt.figure(figsize=(10,10))
plt.scatter(y_test, predictions, c='crimson')
#plt.yscale('log')
#plt.xscale('log')

p1 = max(max(predictions), max(y_test))
p2 = min(min(predictions), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
```

```
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



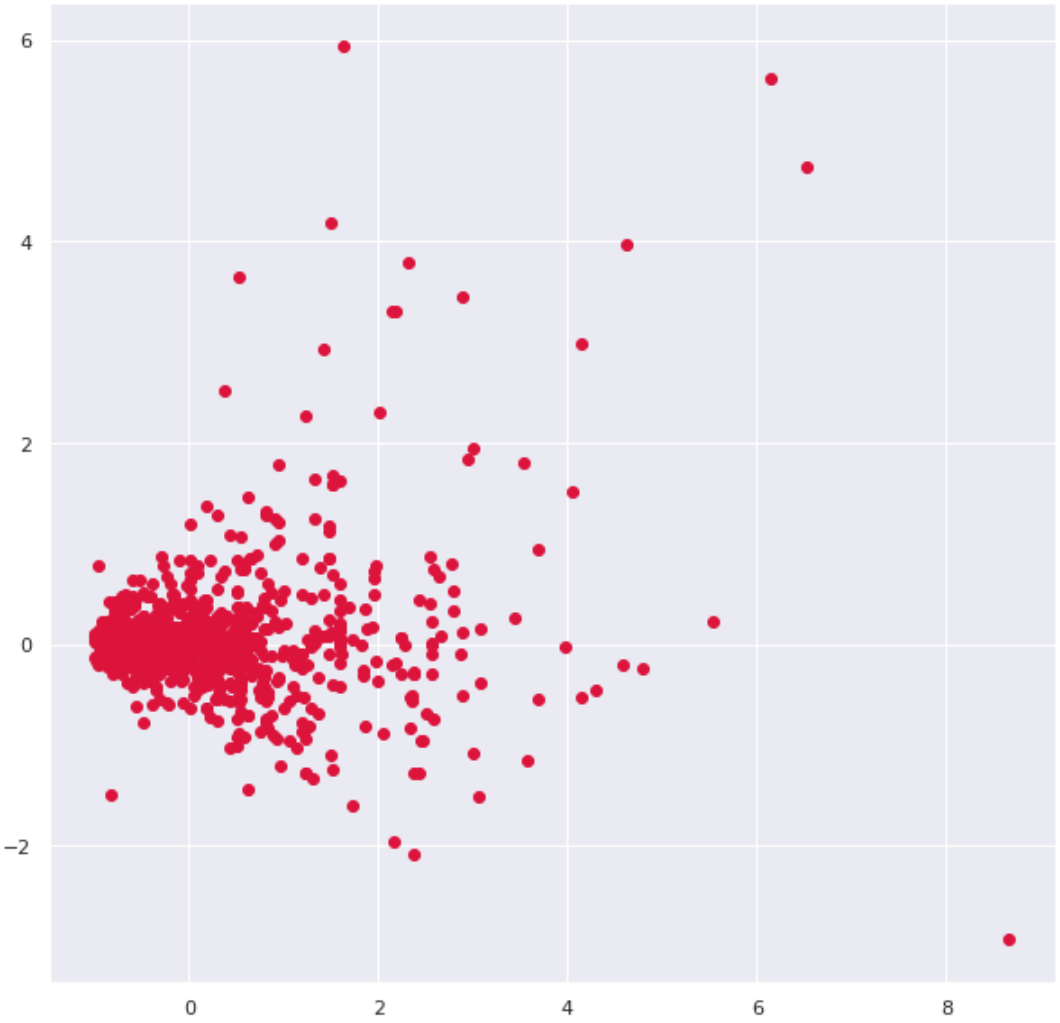

```
plt.figure(figsize=(10,10))
plt.scatter(predictions, resDf["StdResidual"],  c='crimson')
```

```
<matplotlib.collections.PathCollection at 0x7fbaff138b90>
```




```
print(type(X_test))

plt.figure(figsize=(10,10))
plt.scatter(X_test[:,0], residuals, c='crimson')
```

```
<class 'numpy.ndarray'>
<matplotlib.collections.PathCollection at 0x7fbaff0b62d0>
```




```
sns.residplot(x='volume', y='Price', data=numDf,lowess=True,robust=True)
#sns.regplot(x='weight', y='Price', data=tempDf)
plt.show()
```