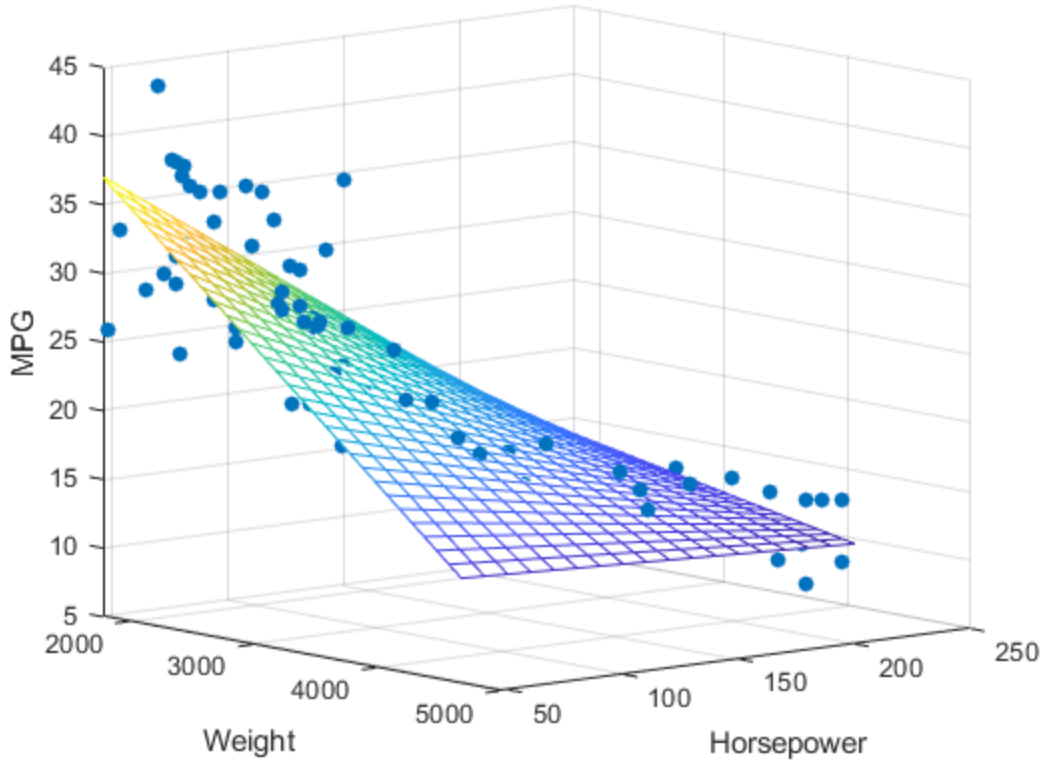
1. Introduction

In this data exploration, we will be using a dataset to try to predict the value of a house using various regression techniques. What features will be important when it comes to predicting the value of a house? The living space? How many bathrooms there are? Who knows? (we will!) The source of difficulty of this project is in the large number of variables, as well as what kind of model to use with them. There are a whopping 79 variables to work with here, so minimizing covariance, and trying to maximize the performance of the model are the name of the game.

2: Regression?

Linear regression is the process of predicting one variable in terms of one or multiple other variables. A line is drawn with a slope corresponding with each relevant variable involved in the regression, then the distance from that line to each data point is calculated, and then squared. This is known as r^2 and is the primary means of evaluation that will be used in this assignment. There are different ways to do this process. Ridge regression and principal component analysis are the other two methods that will be used. Some preparations need to be made when working with linear data: A linear relationship between the input variable and the output variable, non-noisy data, non-redundant variables, and normal distributions.



3. Experiments:

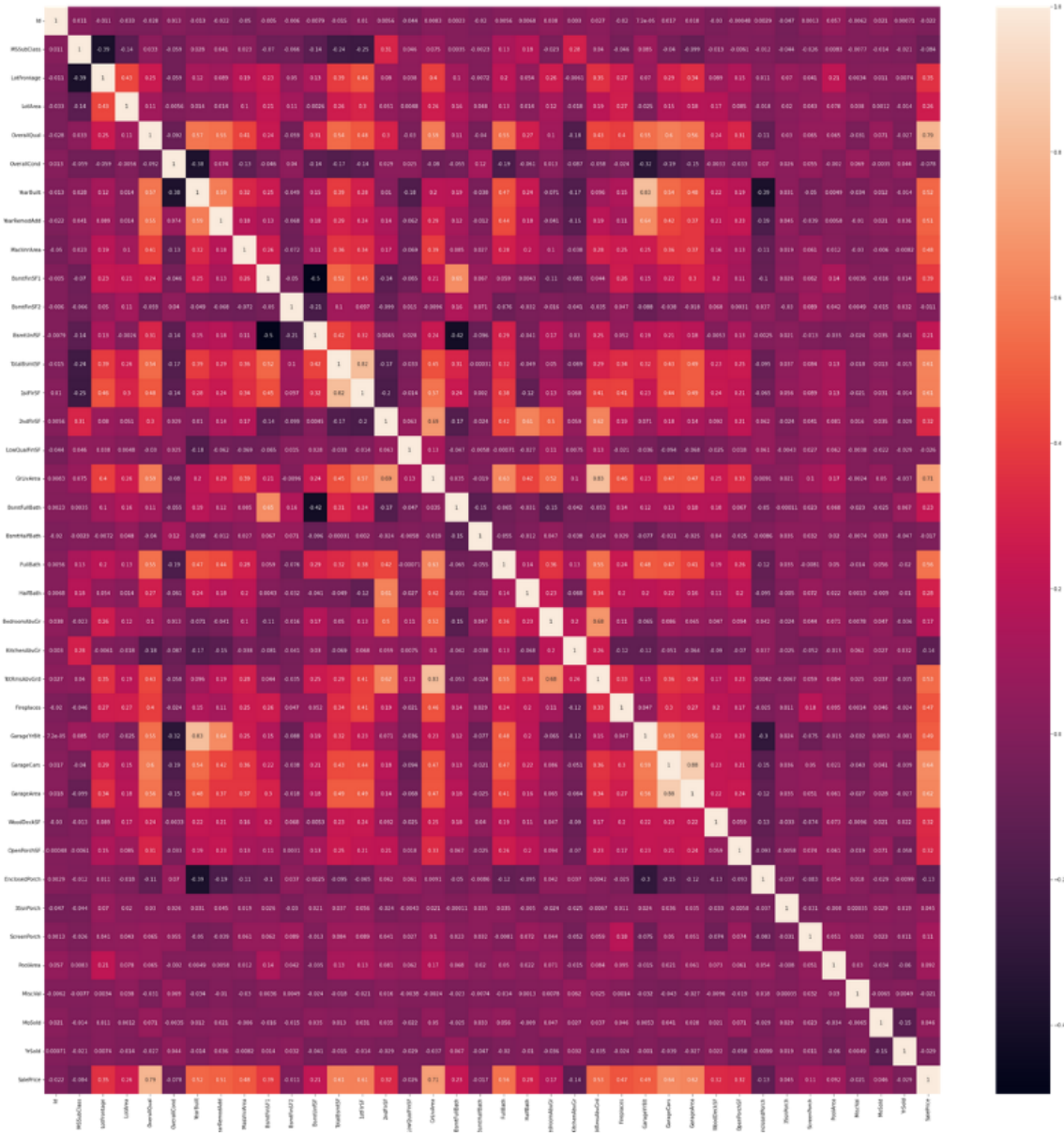
Experiment 1:

Preprocessing:

The first step in my preprocessing procession was to get a really broad overview of my variables and their relationship to sales price. My first goal was to find highly correlated variables, and my second was to find highly collinear variables.

In [5]:
```
#broad overview of correlations
corr = df_train.corr()
plt.figure(figsize = (40,40))
sns.heatmap(corr, annot = True)
```

Out[5]: <AxesSubplot:>

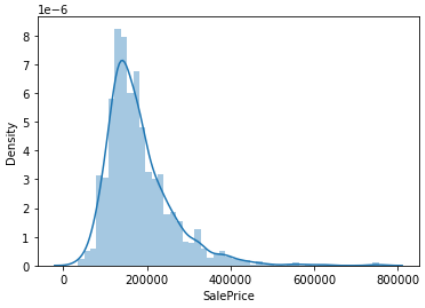


These are some of the variables that were highly collinear that I decided to drop, along with ID, which was obviously not very helpful.

```
#bad or colinear variables

#drop id because it has no use
df_train.drop("Id", axis = 1, inplace = True)

#drop garage GarageYrBlt because it is redundant with year built
df_train.drop("GarageYrBlt", axis = 1, inplace = True)

#drop Garagearea because it is colinear with GarageCars
df_train.drop("GarageArea", axis = 1, inplace = True)

#drop 1stFlrSF because it is colinear with 1stFlrSF
df_train.drop("1stFlrSF", axis = 1, inplace = True)
```

I then wanted to start normalizing variables, especially sale price. Here is what the ‘sale price’ variable looked before applying a log transformation to it.

```
In [9]: #not normal
        sns.distplot(df_train['SalePrice']);
```



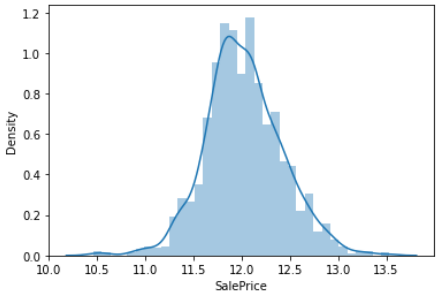

```
n [18]: print("Skewness: %f" % df_train['SalePrice'].skew()) # how symmetric it  is: quite normal but skewed left
        print("Kurtosis: %f" % df_train['SalePrice'].kurt()) #how unnormal it is, how long the tails are: long.

        Skewness: 0.121347
        Kurtosis: 0.809519
```

Wow! Much better now! Normal variables play much nicer with linear regression. I also did some other preprocessing in this image. Firstly I split the variables into categorical and numerical, then I used the median for null variables for the numerical, and just dropped the ones for numerical. I then converted the categorical variables to dummy variables to make them usable in regression. I applied a logarithmic transformation to the rest of the

numeric variables, assuming that the majority of them were skewed in a similar way.

```
In [10]: #numerical replace with median
         train_num = train_num.fillna(train_num.median())

         #apply log transformation to numerical
         train_num = np.log1p(train_num)


         #drop catagorical nulls
         train_cat = train_cat.dropna()


         train_cat = pd.get_dummies(train_cat)

         df_train = pd.concat([train_num, train_cat], axis = 1)
```

```
In [11]: sns.distplot(df_train['SalePrice']);
```



I went ahead and put the categorical variables back together and threw a linear regression at it. R Squared of .879 isn't too shabby if you ask me.

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.879
Model:	OLS	**Adj. R-squared:**	0.876
Method:	Least Squares	**F-statistic:**	312.8
Date:	Tue, 22 Mar 2022	**Prob (F-statistic):**	0.00
Time:	19:27:06	**Log-Likelihood:**	808.02
No. Observations:	1460	**AIC:**	-1548.
Df Residuals:	1426	**BIC:**	-1368.
Df Model:	33		
Covariance Type:	nonrobust		

Experiment 2:

I wanted to try to use ridge regression on it, since there are a lot of variables that are collinear. My hope was that the regular model would overfit, so using a ridge regression might be able to have slightly better results. The benefit of ridge regression is that it penalizes variables that have a large impact, so that other, previously undervalued variables will get boosted. This may be beneficial for this dataset since there are so many variables that do have some influence, while there are also a couple that have a lot of influence. If it was more beneficial for the previously undervalued variables to be boosted, it will perform better.

In [27]:
```
#Experiment 2 Ridge
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
Ridge_pipe = Pipeline([
    ('StandardScaler', StandardScaler()),
    ('Ridge', Ridge())
]).fit(X_train, y_train)

ridge_y_pred = Ridge_pipe.predict(X_test)


print(f"Ridge R^2: {r2_score(y_test, ridge_y_pred):.5f}")
```

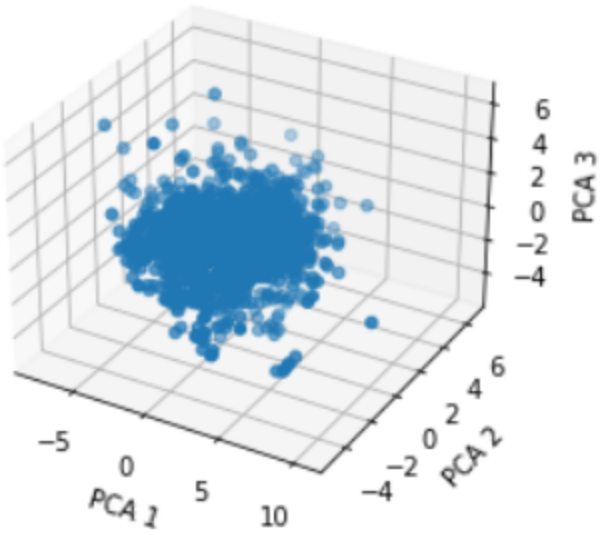
```
Ridge R^2: 0.88080
```

It did have slightly better results! Yay. The difference is pretty minimal sadly. But a higher number is a higher number.

Experiment 3: PCA

Admittedly, I went into principal component analysis hoping for a performance increase, but the end result of it was actually the exact same score. It is still very interesting to see all the 70 or so dimensions compressed down into just three. PCA is less so a way to improve performance than a way to reduce down dimensions into a set of latent variables that explain the data with minimal collinearity. The Strength of PCA in this context is its ability to reduce down the dataset's variable count without the loss of accuracy, but its biggest weakness is its lack of interpretability. What each of the columns actually represent are extremely ambiguous. I kind of wanted to do a higher dimensionality, since I expected for there to be more than 3 relevant variables ideally, but 3 is the most I could visualize reasonably. I wanted to do about 7 originally.

```
:
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(projection = '3d')

x = pca_df[0]
y = pca_df[1]
z = pca_df[2]

ax.set_xlabel("PCA 1")
ax.set_ylabel("PCA 2")
ax.set_zlabel("PCA 3")

ax.scatter(x, y, z)

plt.show()
```



4. Conclusion

Data exploration is a deliberate process, and oftentimes, the difficulty comes from putting data into a form where it is useful. For me, that meant removing nonsense and collinear variables,

transforming numerical variables using log transformations, and dealing with nulls appropriately, as well as the use of dummy variables. Different models certainly also had an influence, but cleaning and transforming the data made a huge difference.

In terms of models, I wish I knew more models to experiment with! It appears like there are a lot of regression techniques to try out, some of which would have likely performed better than what I used. Apparently lasso outperforms ridge, for example. But there are a lot of different models to try out. Very exciting for future projects I may undertake when my data science skills are better.