Predicting Shirt Size, with Data!

Presented by:
Zack Godwin

What’s the big deal?

Buying clothes online has never been easier than it is today. Businesses that provide free return shipping are faced with financial loss when the customer doesn’t get the right fit.

Providing the customer with all the tools necessary to get the right fit the first time, could save money and provide a unique service to the customer.

Data Science + Apparel = $$$

The Research Question:

- Can we predict chest circumference, a common measure of shirt size, using other body measurements?

The Dataset

- The dataset contained body measurement data and some demographic-related features for people age < 1 - 20 years old.
- 3,900 observations
- 122 variables

	WEIGHT	STATURE	ERECT SITTING HEIGHT	MAXIMUM HIP BREADTH (SEATED)	BUTTOCK-KNEE LENGTH	KNEE HEIGHT	HEAD CIRCUMFERENCE	HEAD BREADTH	SHOULDER BREADTH
12	499.0	1578.0	838.0	330.0	518.0	488.0	532.0	154.0	396.0
13	558.0	1618.0	826.0	325.0	568.0	511.0	549.0	152.0	402.0
15	400.0	1468.0	766.0	297.0	509.0	452.0	512.0	145.0	353.0

https://raw.githubusercontent.com/Padam-0/cluster_t-shirt_sizing/master/data.csv

Data Cleaning

In [2]:

```python
# Cleaning the features
df = pd.read_csv('https://raw.githubusercontent.com/Padam-0/cluster_t-shirt_sizing/master/data.csv')

remove_cols = []

# I want to get rid of the columns with more than 2000 Nulls,
# because incomplete data won't be useful.
for i in df.columns:
    if 3900 - df.loc[:,i].astype(bool).sum() > 2000:
        remove_cols.append(i)

df2 = df.drop(remove_cols, axis = 'columns')

# I don't need these categorical variables
demographic_attributes = ['AGE IN YEARS', 'LOCATION',
                          'BIRTH DATE', 'MEASUREMENT DATE', 'MEASUREMENT SET TP',
                          'MEASURER NUMBER', 'COMPUTER NUMBER', 'RACE', 'GRADE LEVEL',
                          'HANDEDNESS', 'NUMBER OF BROTHERS', 'NUMBER OF SISTERS', 'TWIN',
                          'BIRTH ORDER', 'MOTHERS OCCUPATION', 'FATHERS OCCUPATION',
                          'MOTHERS EDUCATION', 'FATHERS EDUCATION', 'YEARS IN COMMUNITY',
                          'ANTHROPOMETER NO', 'CALIPER NO', 'GIRTH NO', 'PERSON #']

df2 = df2.drop(demographic_attributes, axis = 'columns')

# Recode Gender Variable
def binary_gender(gender):
    if gender == 1:
        return 0
    else:
        return 1

df2 = df2.replace(0, np.nan)
df2 = df2.dropna()

df2['gender_recode'] = df2['SEX'].apply(binary_gender)

df2 = df2.drop(['SEX'], axis = 'columns')

df2.head()
```

Creating Shirt Size Categories

```
In [4]:
df_men = df2[(df2.gender_recode == 0) & \
                    (df2['CHEST CIRCUMFERENCE'] >= 640)]
```

```
In [5]:
df_women = df2[(df2.gender_recode == 1) & \
                          (df2['CHEST CIRCUMFERENCE'] >= 640)]
```

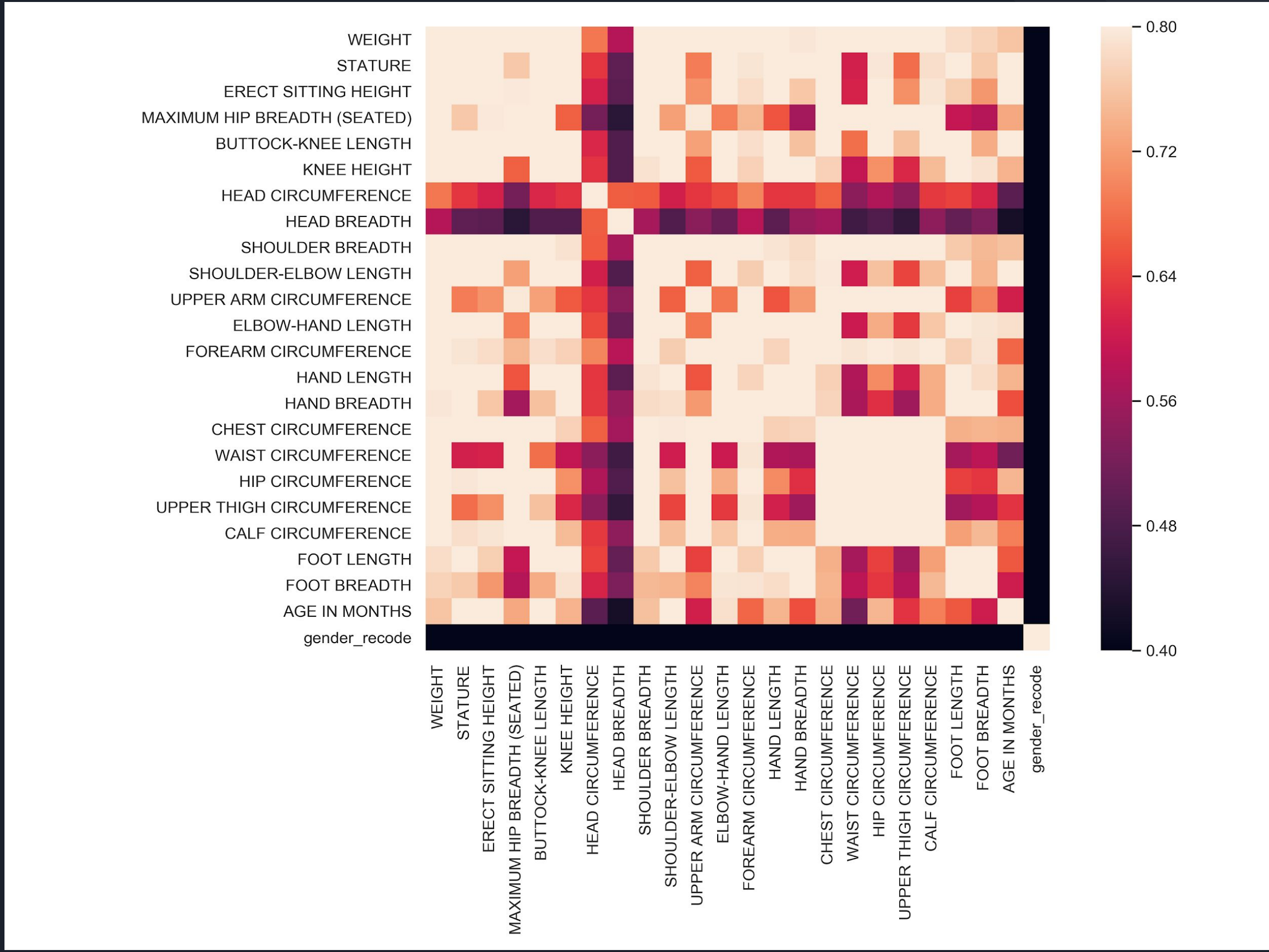
```
In [41]:
#Adding size catergory.
def cat_size_men(size):
    if 660 > size and size >= 640:
        return "YM X Small"
    elif 690 > size and size >= 660:
        return "YM Small"
    elif 750 > size and size >= 690:
        return "YM Medium"
    elif 820 > size and size >= 750:
        return "YM Large"
    elif 890 > size and size >= 820:
        return "YM Extra Large"
    elif 960 > size and size >= 880:
        return "AM Small"
    elif 1040 > size and size >= 960:
        return "AM Medium"
    elif 1120 > size and size >= 1040:
        return "AM Large"
    elif 1240 > size and size >= 1120:
        return "AM Extra Large"
    elif 1360 > size and size >= 1240:
        return "AM XX Large"
    elif 1480 > size and size >= 1360:
        return "AM XXX Large"
    elif 1600 > size and size >= 1480:
        return "AM XXXX Large"
    else:
        return "Other"
pd.options.mode.chained_assignment = None  # default='warn'
df_men['cat_size'] = df_men.loc[:,'CHEST CIRCUMFERENCE'].apply(cat_size_men)
df_men.head()
```

```
In [42]:
def cat_size_women(size):
    if 660 > size and size >= 640:
        return "YW X Small"
    elif 690 > size and size >= 660:
        return "YW Small"
    elif 750 > size and size >= 690:
        return "YW Medium"
    elif 780 > size and size >= 750:
        return "YW Large"
    elif 830 > size and size >= 780:
        return "AW Extra Small"
    elif 900 > size and size >= 830:
        return "AW Small"
    elif 970 > size and size >= 900:
        return "AW Medium"
    elif 1040 > size and size >= 970:
        return "AW Large"
    elif 1140 > size and size >= 1040:
        return "AW X Large"
    elif 1240 > size and size >= 1140:
        return "AW XX Large"
    else:
        return "Other"

df_women['cat_size'] = df_women.loc[:,'CHEST CIRCUMFERENCE'].apply(cat_size_women)
df_women.head()
```

```
In [64]:
frames = [df_women, df_men]

df_full = pd.concat(frames)
df_full.cat_size.unique()
```

```
Out[64]: array(['YW Large', 'AW Small', 'YW Medium', 'AW Extra Small', 'YW Small',
               'AW Large', 'AW Medium', 'YW X Small', 'AW X Large', 'YM X Small',
               'YM Medium', 'YM Small', 'YM Large', 'YM Extra Large', 'AM Medium',
               'AM Small', 'AM Large', 'AM Extra Large'], dtype=object)
```

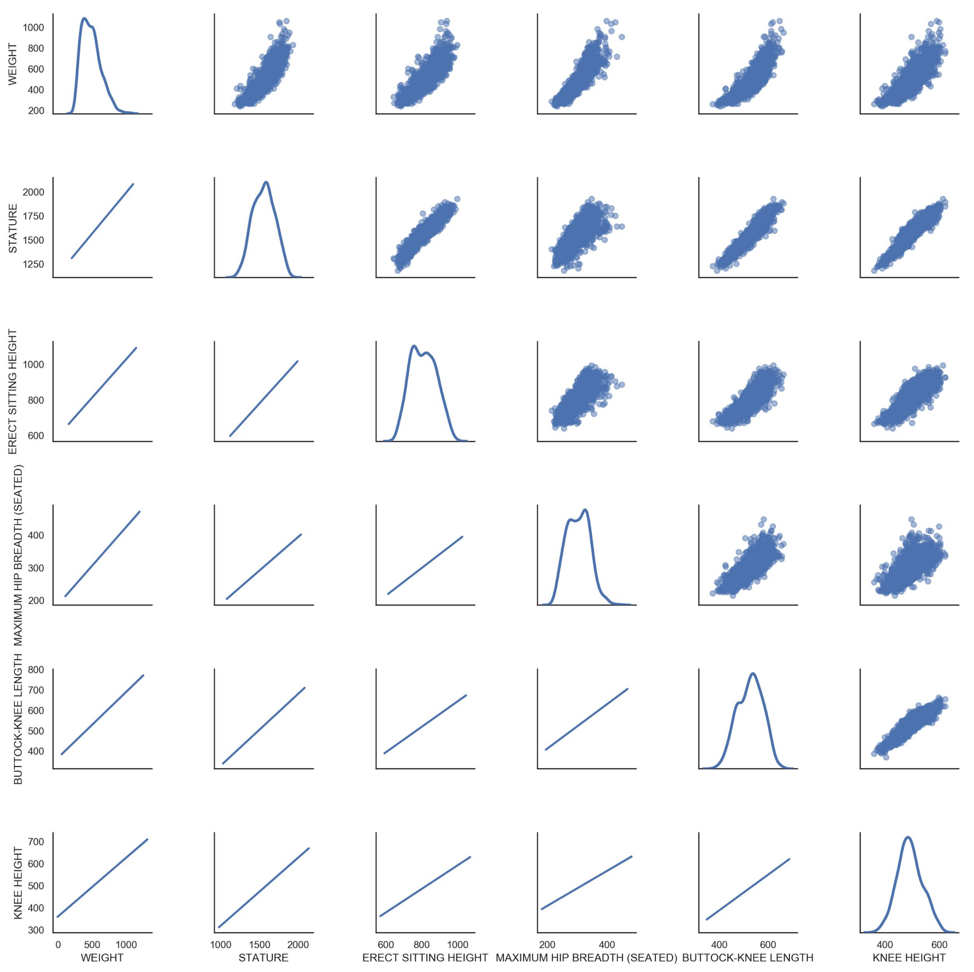
Feature correlations:



- We have very strongly correlated variables, and few weakly correlated.

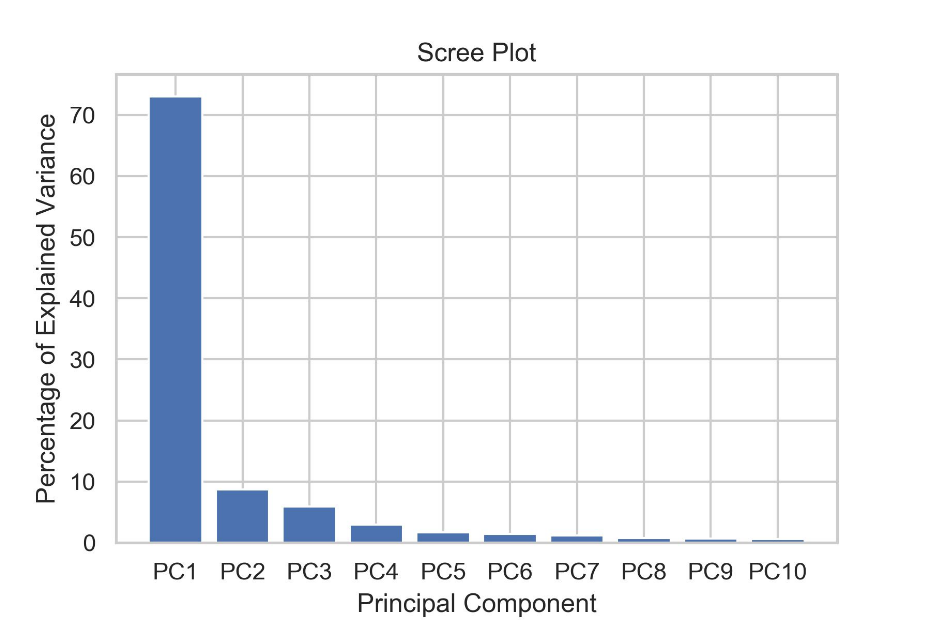
- We may assume linearity in body measurement variables, but to confirm we will examine a few.

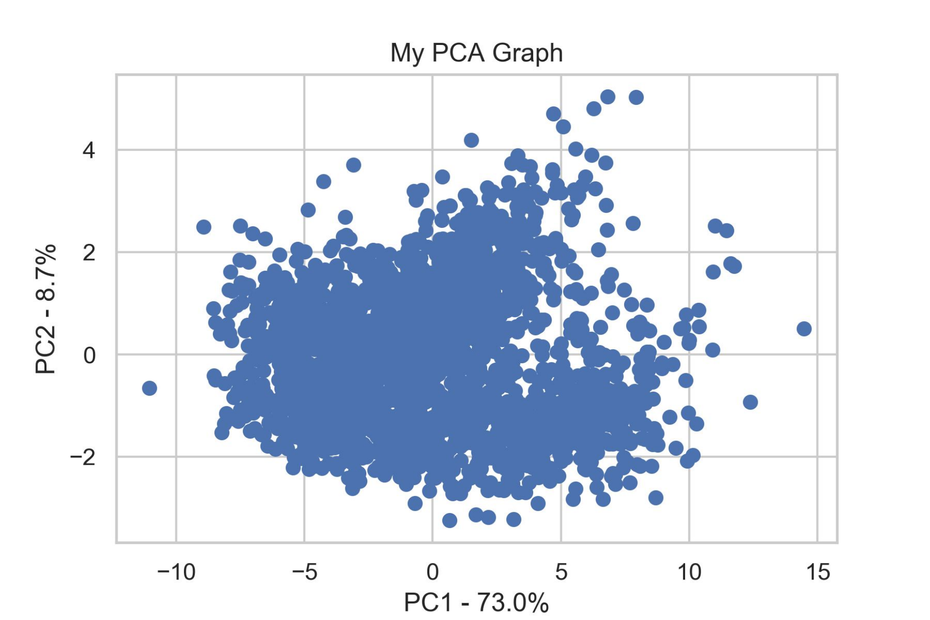
Feature exploration:



- **Looking at just a few features, we see several linear relationships.**
- **To simplify the model, I first performed Principal Component Analysis.**

PCA



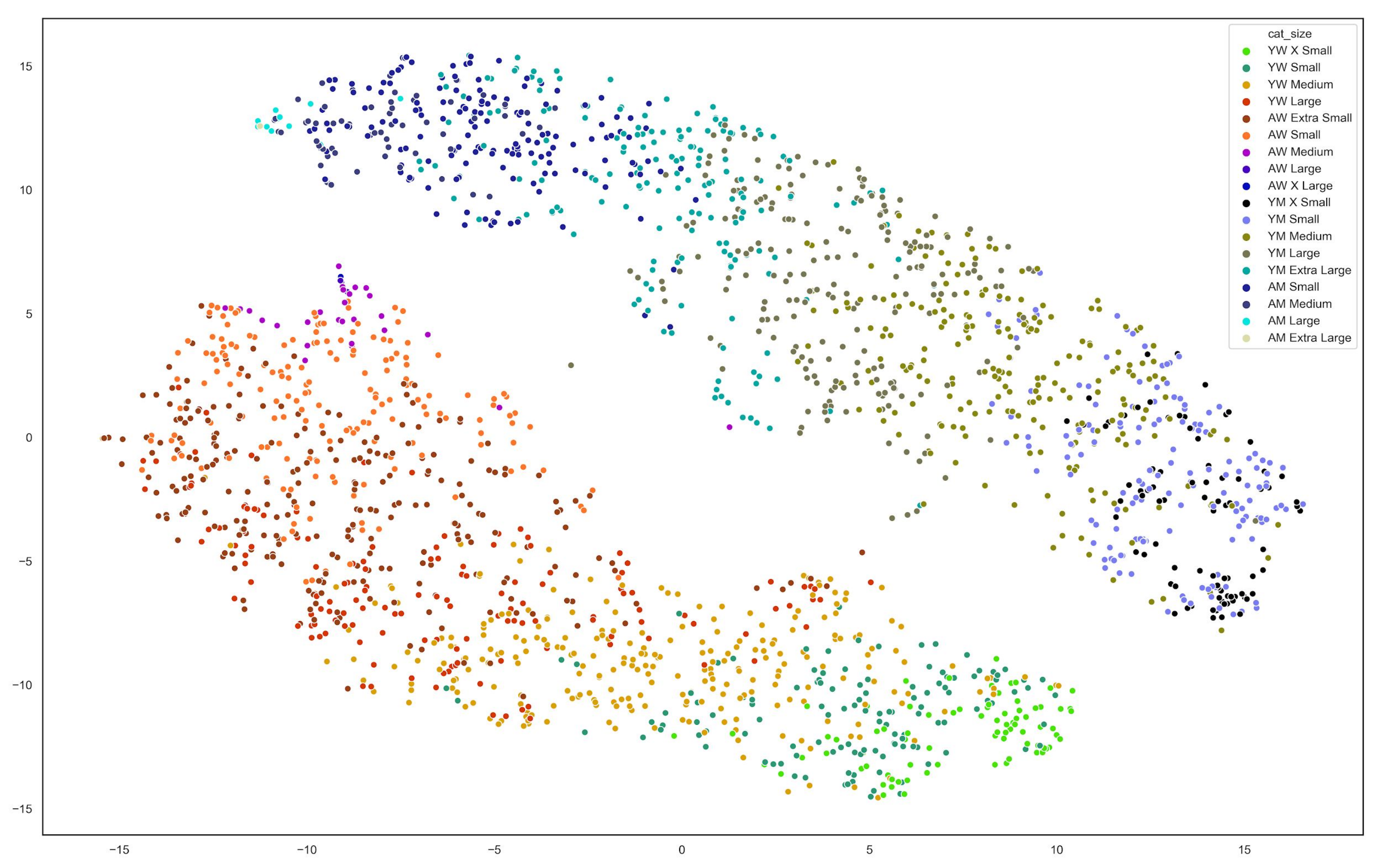


- PCA reduced 24 features down to 10 principal components.
- We'll only use PC1 and PC2.

TSNE

To visualize how the observations cluster relative to the shirt size category I created I used a t-distributed stochastic neighbor embedding algorithm.

We see clustering by gender, and clustering by size in a gradient pattern from smallest to largest!



Model Selection

OLS models trained with 80/20 split.

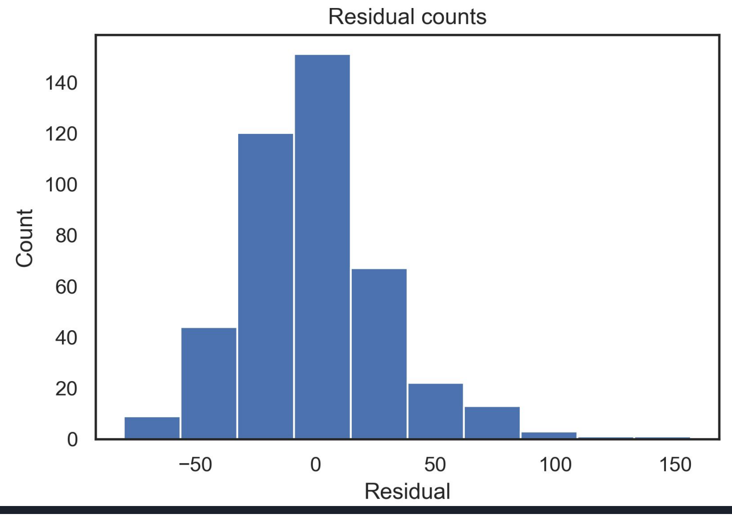
Target = Chest Circumference

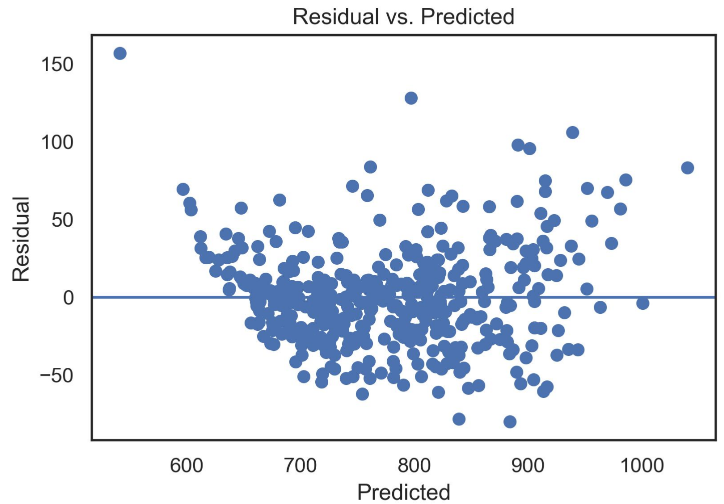
Success = High R-Squared value and a high mean CV with low standard deviation.

Linear Regression on Principal Components

R-squared:
0.890

Mean (SD) Cross Validation Score:
0.94 (+/- 0.01)





Regression Model Scoreboard

Regression Model	R-Squared	Mean CV (+/- SD)
PCA Linear OLS	0.890	0.94 (+/- 0.01)

Is PCA the best method?

➔ I used sklearn’s variance threshold to remove features with low variance.

In [28]:

```
from sklearn.feature_selection import VarianceThreshold


def variance_threshold_selector(data, threshold):
    selector = VarianceThreshold(threshold)
    selector.fit(data)
    return data[data.columns[selector.get_support(indices=True)]]
```

In [29]:

```
variance_threshold = variance_threshold_selector(
    X_scaled_df, threshold=(.8 * (1 - .8)))
```

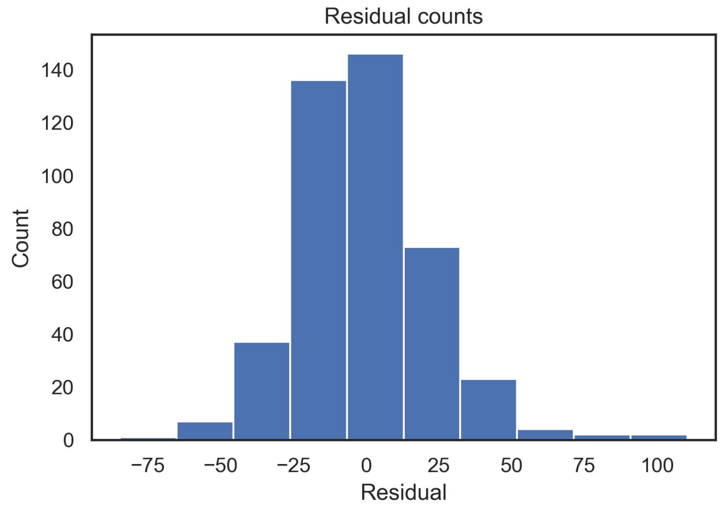
Linear Regression on Variance Threshold

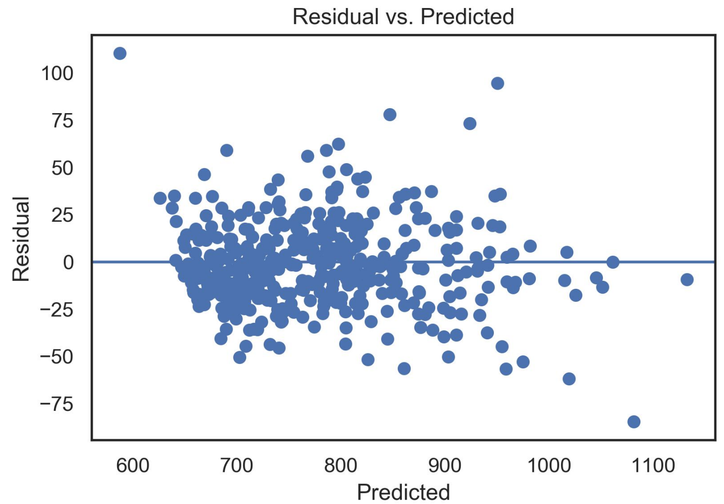
R-squared:

0.945

Mean (SD) Cross Validation Score:

0.94 (+/- 0.01)





Regression Model Scoreboard

Regression Model	R-Squared	Mean CV (+/- SD)
PCA Linear OLS	0.890	0.94 (+/- 0.01)
Variance Threshold OLS	**0.945**	**0.94 (+/- 0.01)**

Is OLS the best model?

- ➔ I compared the results of the OLS models to Ridge and Lasso regression models.
- ➔ These models trained on the whole data, and did their our feature management.

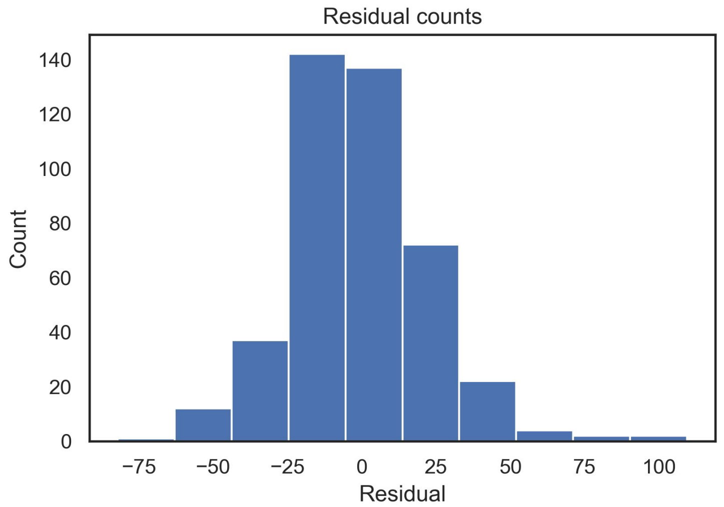
Lasso Regression

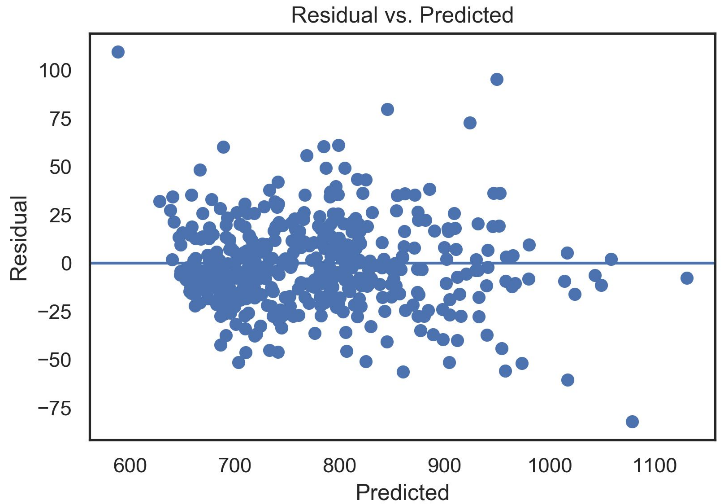
R-squared:

0.945

Mean (SD) Cross Validation Score:

0.93 (+/- 0.05)

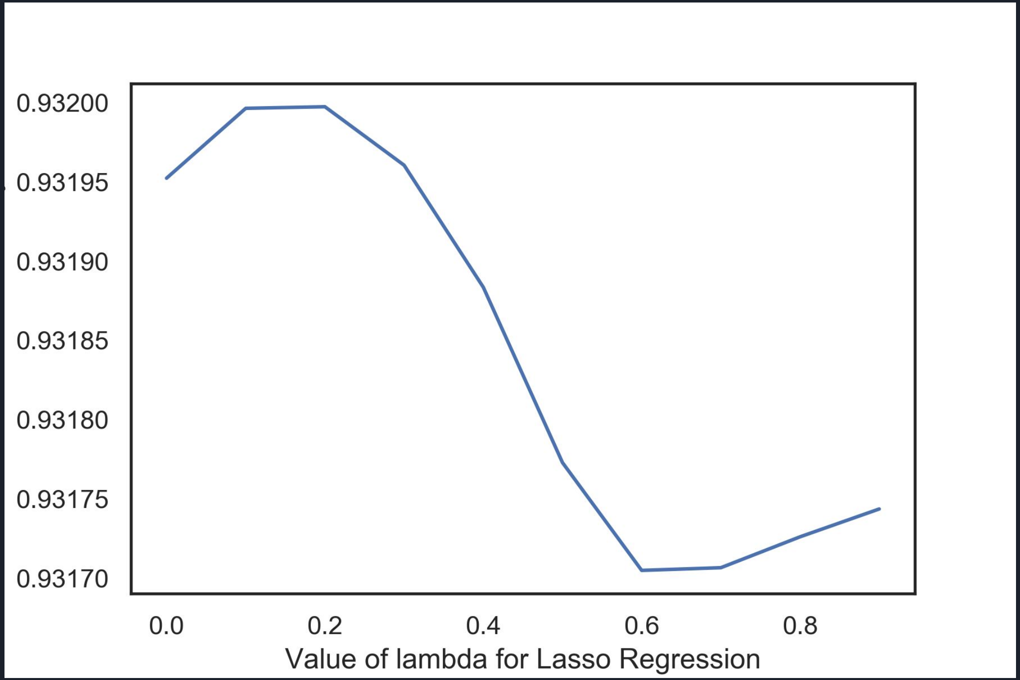




Regression Model Scoreboard

Regression Model	**R-Squared**	**Mean CV (+/- SD)**
PCA Linear OLS	0.890	0.94 (+/- 0.01)
Variance Threshold OLS	**0.945**	**0.94 (+/- 0.01)**
Lasso Regression	0.889	0.93 (+/- 0.05)

Finding Lambda...



... 0.2 should work just fine.

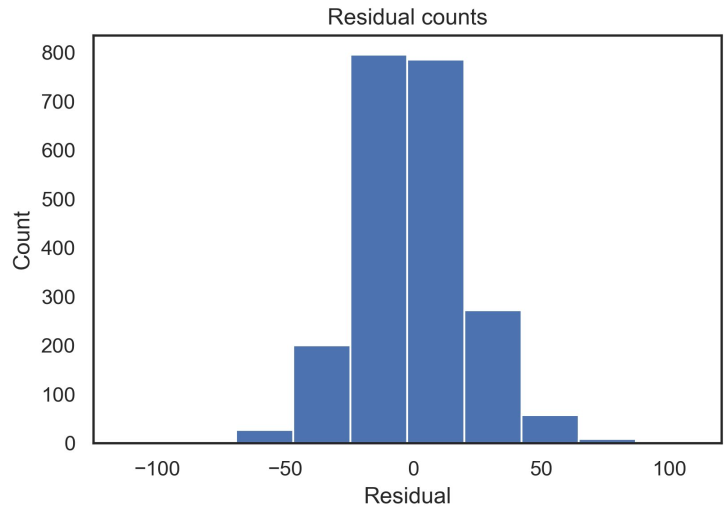
Tuned lambda Lasso Regression

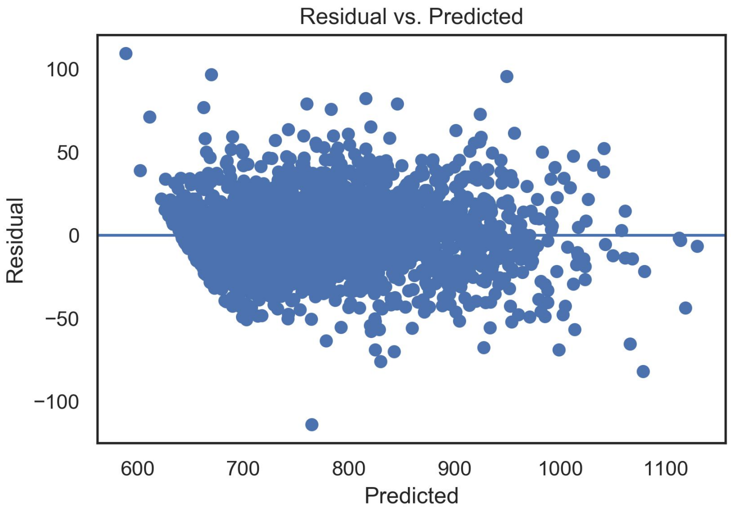
R-squared:

0.944

Mean (SD) Cross Validation Score:

0.94 (+/- 0.01)

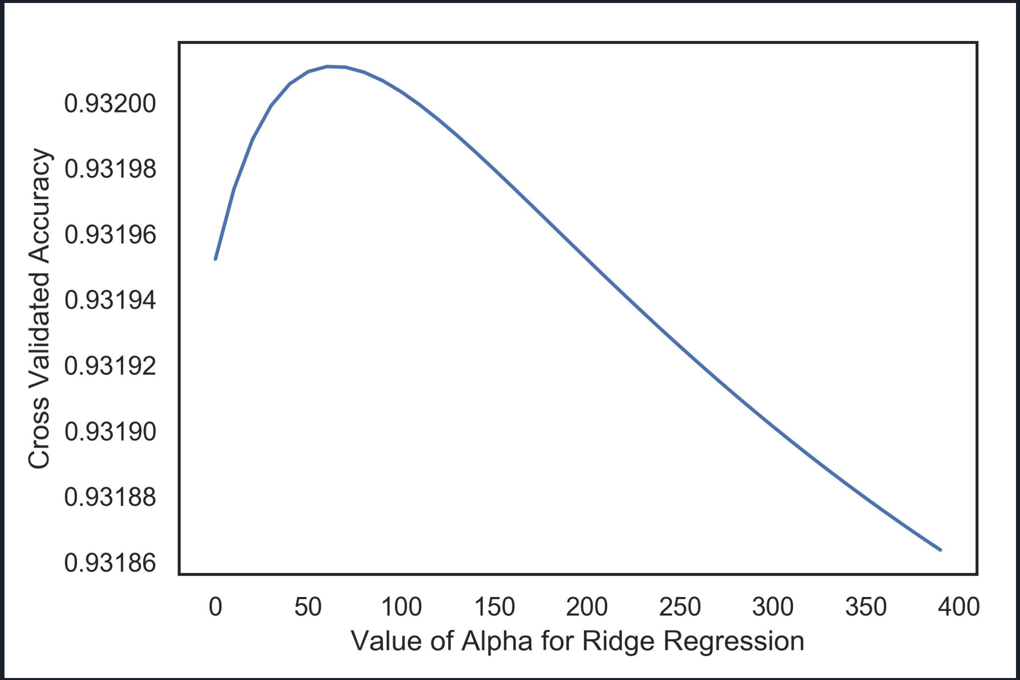




Regression Model Scoreboard

Regression Model	**R-Squared**	**Mean CV (+/- SD)**
PCA Linear OLS	0.890	0.94 (+/- 0.01)
Variance Threshold OLS	**0.945**	**0.94 (+/- 0.01)**
Lasso Regression	0.889	0.93 (+/- 0.05)
Tuned Lambda Lasso	0.944	0.94 (+/- 0.01)

Finding Alpha...

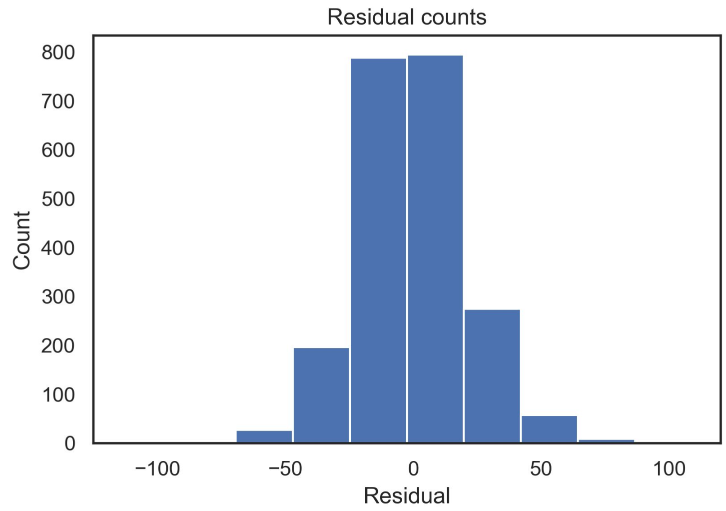


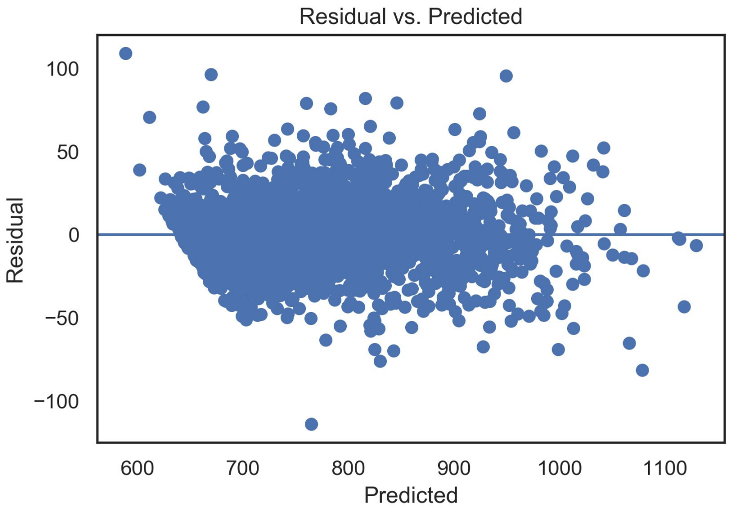
... 60 should work just fine.

Tuned Ridge Regression

R-squared:
0.944

Mean (SD) Cross Validation Score:
0.94 (+/- 0.01)





Regression Model Scoreboard

Regression Model	**R-Squared**	**Mean CV (+/- SD)**
PCA Linear OLS	0.890	0.94 (+/- 0.01)
Variance Threshold OLS	**0.945**	**0.94 (+/- 0.01)**
Lasso Regression	0.889	0.93 (+/- 0.05)
Tuned Lambda Lasso	0.944	0.94 (+/- 0.01)
Tuned Ridge Regression	0.944	0.94 (+/- 0.01)

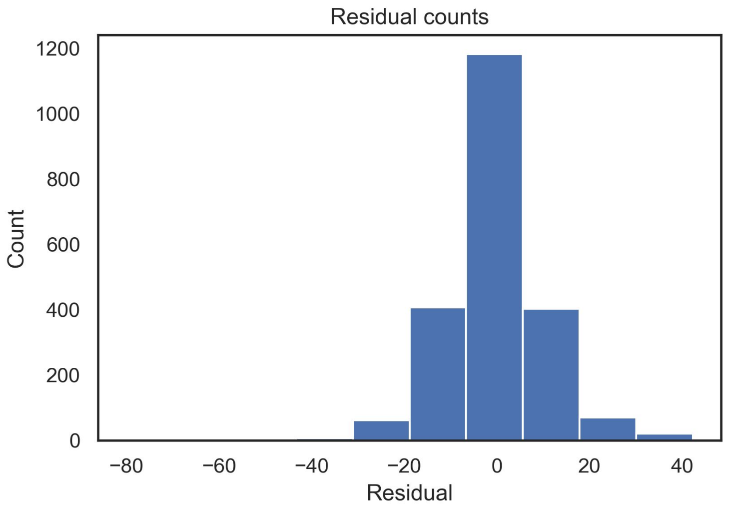
Random Forest Regression

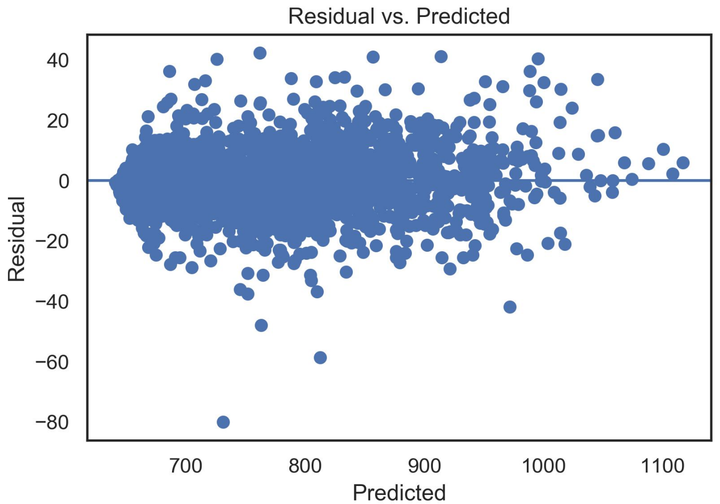
R-squared:

0.987

Mean (SD) Cross Validation Score:

0.93 (+/- 0.01)





Regression Model Scoreboard

Regression Model	R-Squared	Mean CV (+/- SD)
PCA Linear OLS	0.890	0.94 (+/- 0.01)
Variance Threshold OLS	**0.945**	**0.94 (+/- 0.01)**
Lasso Regression	0.889	0.93 (+/- 0.05)
Tuned Lambda Lasso	0.944	0.94 (+/- 0.01)
Tuned Ridge Regression	0.944	0.94 (+/- 0.01)
Random Forest	0.979	0.88 (+/- 0.01)

Tuning Hyperparameters: Random Forest

In [42]:
```python
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True, False],
    'max_depth': [80, 90, 100, 110],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
```

In [43]:
```python
rfr = ensemble.RandomForestRegressor()
rfrfit = rfr.fit(X, y)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rfrfit, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X, y)
grid_search.best_params_
```

```
Fitting 3 folds for each of 288 candidates, totalling 864 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  6.6min
[Parallel(n_jobs=-1)]: Done 864 out of 864 | elapsed:  9.9min finished
```

Out[43]:
```
{'bootstrap': True,
 'max_depth': 110,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 200}
```

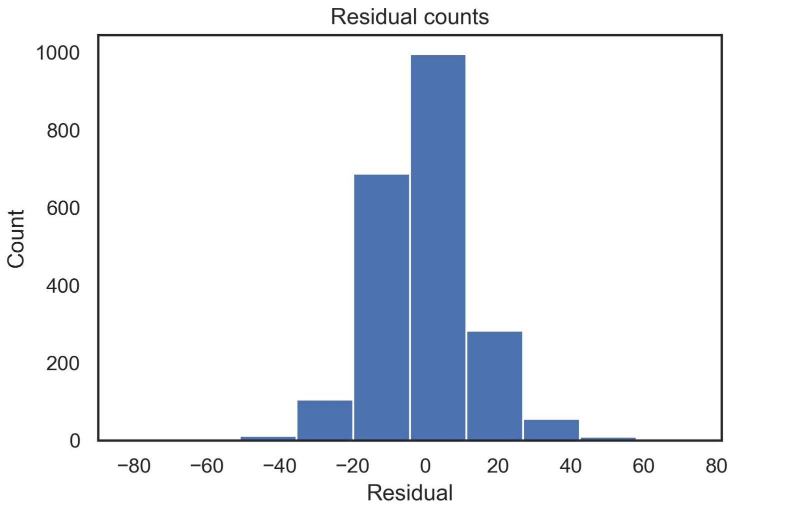
Tuned Hyperparameters Random Forest Regression

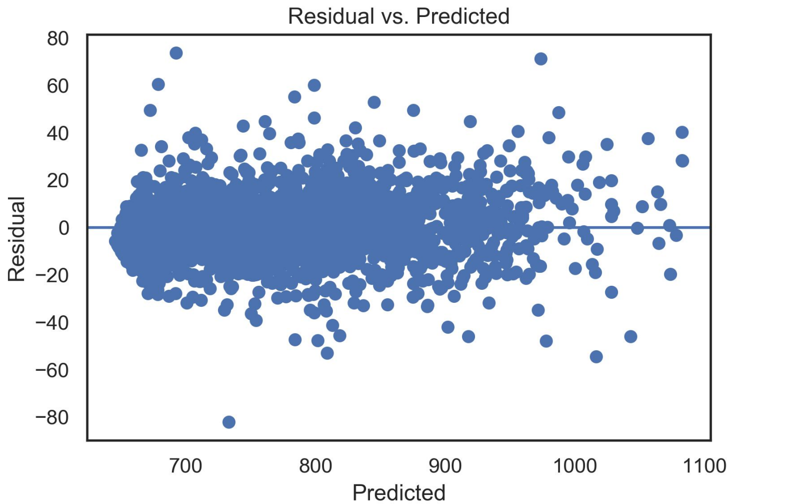
R-squared:

0.978

Mean (SD) Cross Validation Score:

0.94 (+/- 0.01)





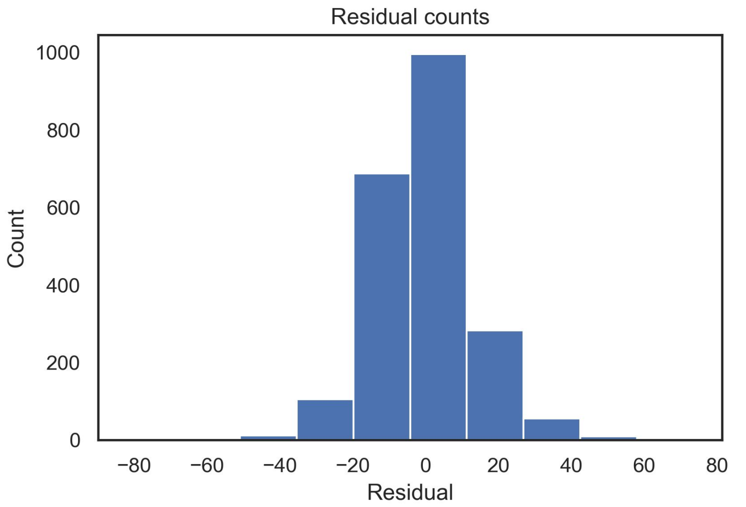
Regression Model Scoreboard

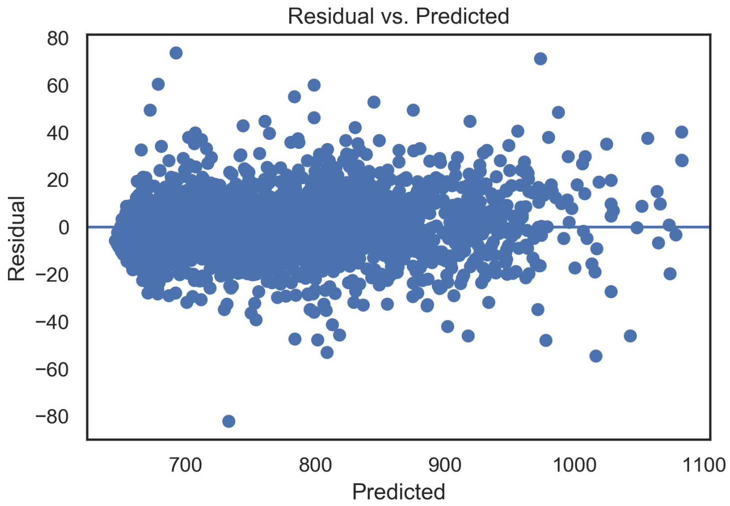
Regression Model	**R-Squared**	**Mean CV (+/- SD)**
PCA Linear OLS	0.890	0.94 (+/- 0.01)
Variance Threshold OLS	0.945	0.94 (+/- 0.01)
Lasso Regression	0.889	0.93 (+/- 0.05)
Tuned Lambda Lasso	0.944	0.94 (+/- 0.01)
Tuned Ridge Regression	0.944	0.94 (+/- 0.01)
Random Forest	0.979	0.88 (+/- 0.01)
Tuned Random Forest	**0.978**	**0.94 (+/- 0.01)**

Conclusion

➔ **The Tuned-Hyperparameters Random Forest Regression** best met my success definition for this data:

- **High R2 = 0.978**
- **High Mean CV of 0.94**
- **Low CV (SD) of (+/- 0.01)**





Questions?