K-Nearest Neighbors

By: Elsa Carlson, Jerad Ipsen, Shannon Bayless, Ryan-Arnold Gamilo

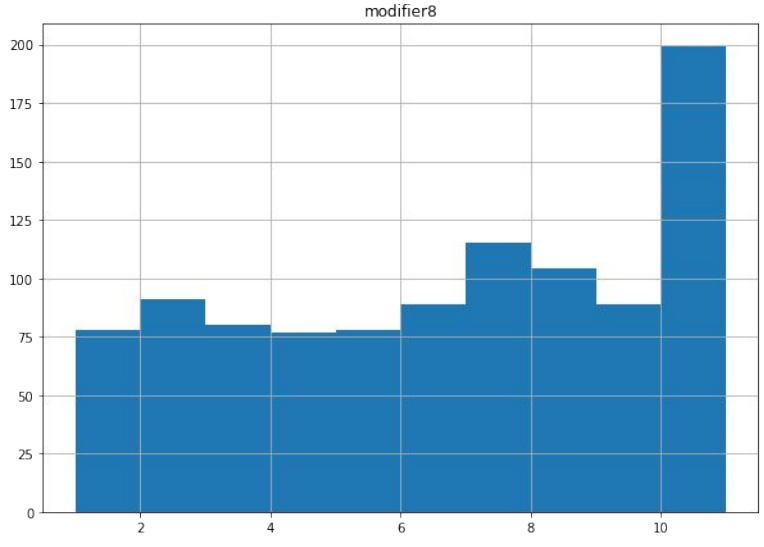
Dataset

- We chose to do a regression using KNN due to the datasets available
- Used the Dungeons & Dragons dataset from the Model Optimization Lesson
- Started with 1000 rows and 17 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   strength      1000 non-null   int64
 1   dexterity     1000 non-null   int64
 2   constitution  1000 non-null   int64
 3   wisdom        1000 non-null   int64
 4   intelligence  1000 non-null   int64
 5   charisma      1000 non-null   int64
 6   weight        1000 non-null   int64
 7   height        1000 non-null   int64
 8   modifier1     1000 non-null   int64
 9   modifier2     1000 non-null   int64
 10  modifier3     1000 non-null   int64
 11  modifier4     1000 non-null   int64
 12  modifier5     1000 non-null   int64
 13  modifier6     1000 non-null   int64
 14  modifier7     1000 non-null   int64
 15  modifier8     1000 non-null   int64
 16  score         1000 non-null   float64
dtypes: float64(1), int64(16)
memory usage: 132.9 KB
```

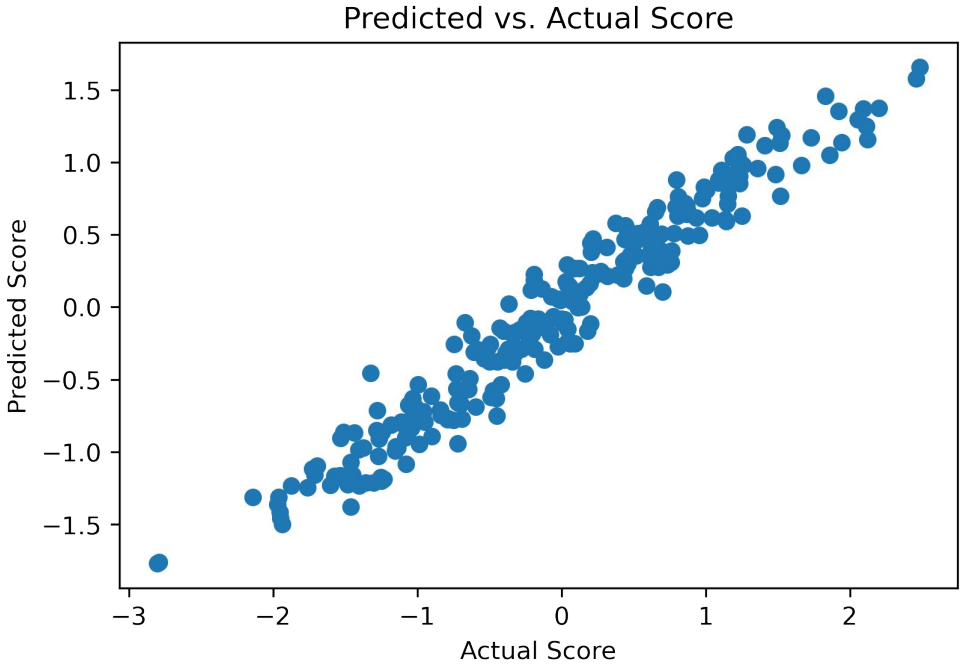
Data Processing

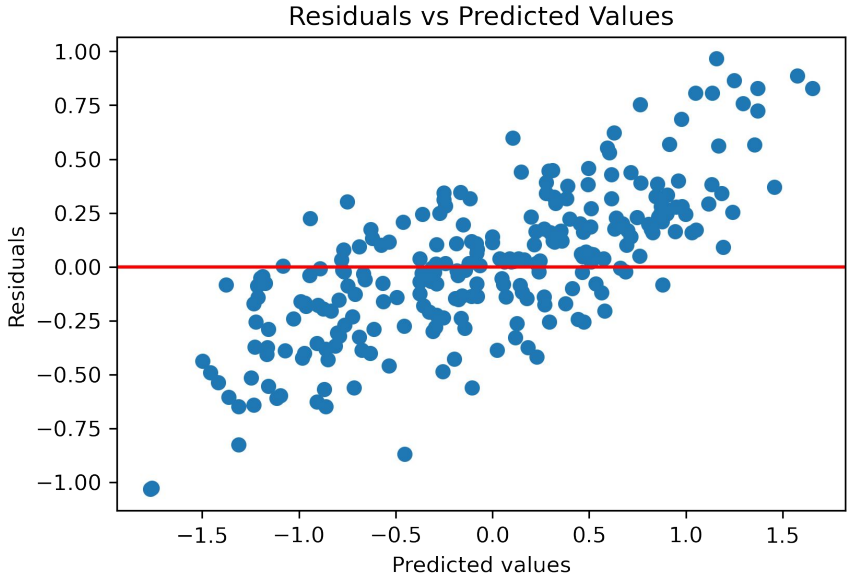
- No nulls or zero values
- All data types were what they were expected to be
- Data distributed fairly normally with no outliers (except for modifier8 but that was fine)
- Not much cleaning necessary, but we did standardize the data



Initial Model

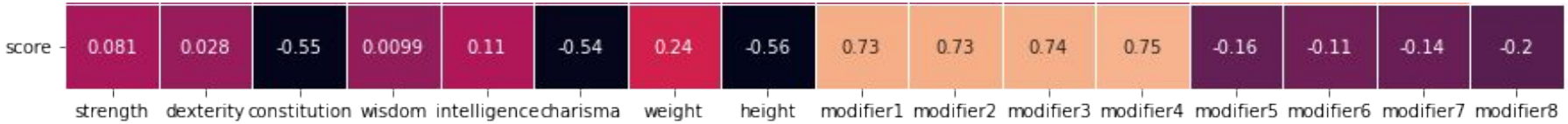
Original R^2: 0.889





Feature Engineering

- Dropped non-correlated columns (|n| < 0.4)
- Dropped columns that were highly correlated with each other
    - Kept height & mod4



Hyperparameter tuning: k & distance metric

```
y = df_scaled_dropped['score']
X = df_scaled_dropped.drop(columns = 'score')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
knn = KNeighborsRegressor()

k_range = list(range(1,100))
d_range = ['minkowski','manhattan','euclidean']
param_grid = dict(n_neighbors=k_range,metric=d_range)
grid = GridSearchCV(knn, param_grid,scoring='r2')

grid_search = grid.fit(X_train,y_train)
print(grid_search.best_params_)
```

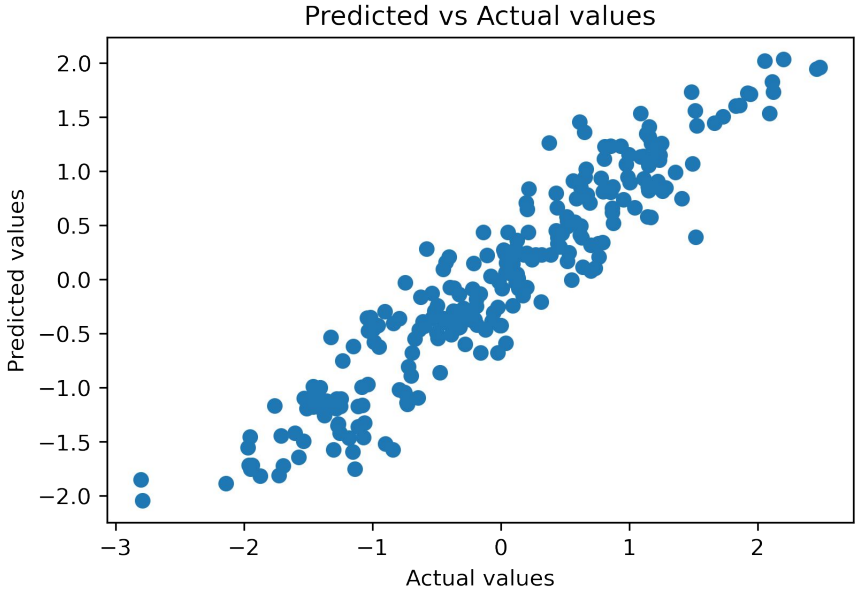
✓ 10.4s

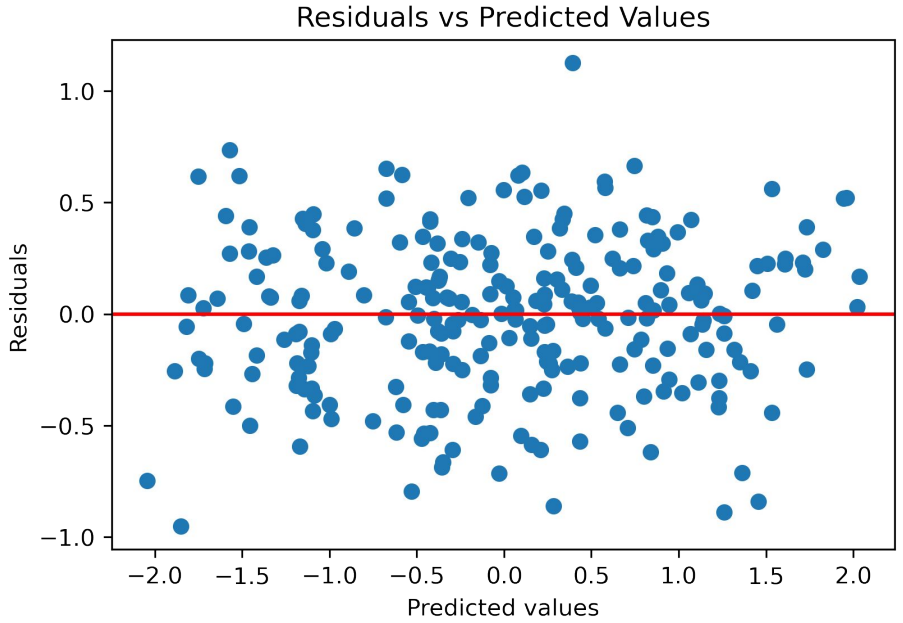
```
{'metric': 'minkowski', 'n_neighbors': 16}
```

Final Model

Initial R^2: 0.885

Final R^2: 0.886

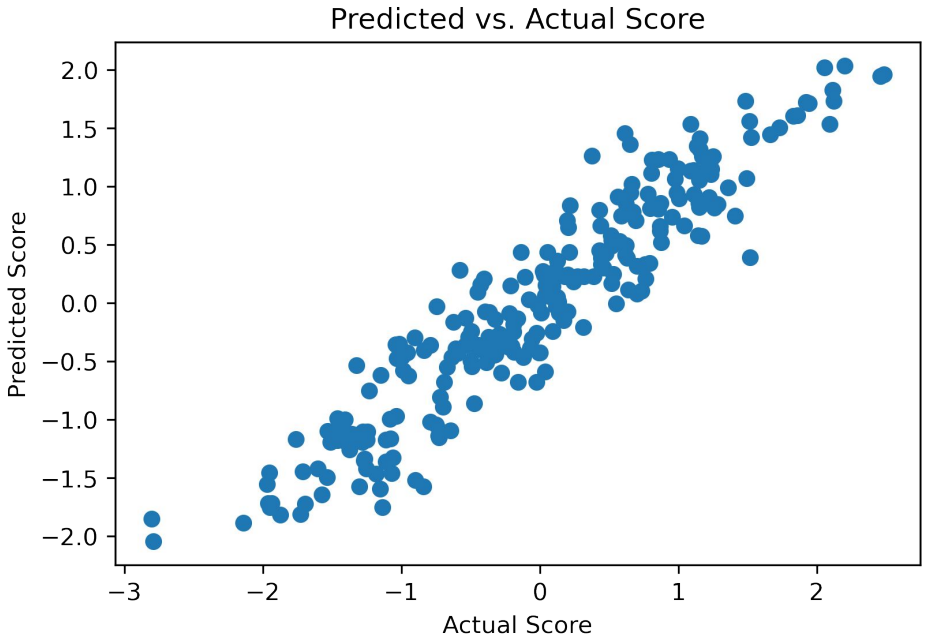


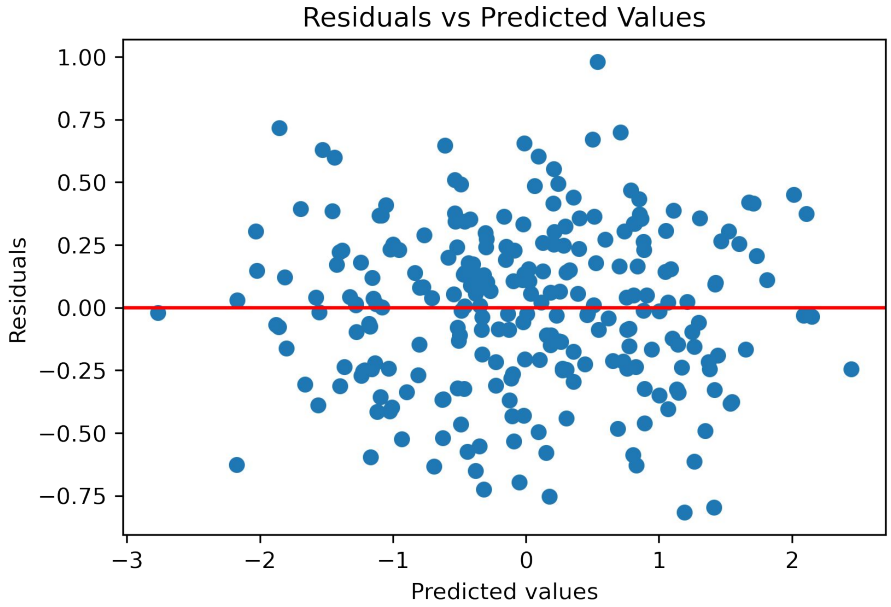


Comparison to LASSO

Initial R^2: 0.999

Final R^2: 0.901





Conclusion

- Optimized & tuned KNN regression performs worse than other regression algorithms
- Potential improvements:
  - Increase correlation threshold
  - Combine features rather than dropping them altogether