Aromatase Drug Discovery Project

By: Gowrav Mannem
Published: July 9th, 2022
GitHub Link: https://github.com/gowravmannem/Aromatase-Drug-Discovery
Adopted from Chanin Nantasenamat's (Dataprofessor) Youtube Tutorial

Background

According to the National Breast Cancer Foundation, 1 in 8 women in the US will develop Breast Cancer. Clearly, breast cancer is a problem that continues to plague society. Aromatase is an enzyme related to the production of estrogen, the female hormone. It converts androstenedione and estrone into estrogen. Blocking this enzyme from performing its task is one the prominent methods to treat and manage breast cancer.

Goal

Aromatase Inhibitors is a popular way to mitigate the risk of developing breast cancer. The goal is to build an accurate ML model that when given the SMILES (simplified molecular-input line-entry system) notation of a compound will output the predicted pIC50 value or the potency of a substance in inhibiting a particular biological function. The pIC50 value, in our use case, will show how good a substance is at inhibiting the aromatase enzyme. Hopefully, this model will aid efforts to combat breast cancer.

Project Breakdown

1. Procure and Clean Data
2. Chemical Space Analysis on Lipinski Descriptors
3. Calculate Fingerprint Descriptors with PaDEL
4. Comparing various Regression Models
5. Building Model
6.

Step 1: Procure and Clean Data

I used the ChEMBL database to obtain my data regarding aromatase. ChEMBL is curated and run by the European Bioinformatics Institute so the data I pulled from it is accurate and reliable. I also loaded the pandas library to clean the data from ChEMBL.

```
[ ]  import pandas as pd
     from chembl_webresource_client.new_client import new_client
```

Once I loaded the ChEMBL Database onto my workspace on Google Colab, I loaded the Aromatase Homo Sapien dataset onto my workspace. I then selected all data points that have a 'standard_type' of 'IC50'. This value represents the potency of a substance in inhibiting Aromatase activity. Without this value type, the data is useless to our cause.

In this Aromatase IC50 data set, I deleted all points without 'standard_value' and 'canonical_smiles". These two columns represent key data for our model. From this dataset, I selected the 'mol_cid', 'canonical_smiles' (chemical structure of compound), and 'standard_value' (potency of compound) and put them in a new dataset.

Using the 'standard_value' column, I organized the data points into 3 categories of bioactivity: active, intermediate, and inactive. I added this column to the dataset.

```
[ ]  # We are classfying and isolating each datapoint's standard value into 3 classes
     # inactice, active, and intermediate
     # standad_value > 10000 => "inactive"
     # standard_value <= 1000 => "active"
     # else => intermediate
     bioactivity_class=[]
     for i in aromatase_df_clean.standard_value:
       if i>10000:
         bioactivity_class.append("inactive")
       elif i <=1000:
         bioactivity_class.append("active")
       else:
         bioactivity_class.append("intermediate")
```

The final resulting dataset looks like this:

	molecule_chembl_id	canonical_smiles	bioactivity_class	standard_value
0	CHEMBL341591	CC12CCC(O)CC1=CCC1C2CCC2(C)C(CC3CN3)CCC12	intermediate	7100.00
1	CHEMBL2111947	C[C@]12CC[C@H]3[C@@H](CC=C4C[C@@H](O)CC[C@@]43...	inactive	50000.00
2	CHEMBL431859	CCn1c(C(c2ccc(F)cc2)n2ccnc2)c(C)c2cc(Br)ccc21	active	238.00
3	CHEMBL113637	CCn1cc(C(c2ccc(F)cc2)n2ccnc2)c2ccccc21	active	57.00
4	CHEMBL112021	Clc1ccccc1Cn1cc(Cn2ccnc2)c2ccccc21	active	54.00
...	...	...	...	...
2813	CHEMBL4755831	Nc1ccc(C(Cn2ccnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	active	5.24
2814	CHEMBL4750835	Nc1ccc(C(Cn2cncn2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	active	104.00
2815	CHEMBL4745681	Nc1ccc(C(Cn2cnnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	active	17.70
2816	CHEMBL4778401	Oc1ccc(C(=C(Cn2ccnc2)c2cccc(O)c2)c2ccc(O)cc2)cc1	active	60.40
2817	CHEMBL4761359	Nc1cccc(C(Cn2ccnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)c1	active	439.00

2818 rows × 4 columns

I have 2818 data points. This is plenty to make an accurate model.

Step 2: Chemical Space Analysis on Lipinski Descriptors

Before I can analyze the Lipinski Descriptors, I must first generate the Lipinski Descriptors for each compound in our dataset. In order to generate these descriptors, I need to import Conda and RDKit. RDKit is a chemical informatics library that will help us to calculate the descriptors based on the 'chemical_smiles' or chemical structure of compounds. Conda will help us bring RDKit onto our workspace.

```
! wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.8.2-Linux-x86_64.sh
! chmod +x Miniconda3-py37_4.8.2-Linux-x86_64.sh
! bash ./Miniconda3-py37_4.8.2-Linux-x86_64.sh -b -f -p /usr/local
! conda install -c rdkit rdkit -y
import sys
sys.path.append('/usr/local/lib/python3.7/site-packages/')
```

Using Lipinski's rule I can evaluate the drug likeness of a compound. This is based on the Absorption, Distribution, Metabolism and Excretion or the pharmacokinetic profile of a compound.

The Lipinski's Rule states the following:
1) Molecular Weight < 500 Dalton
2) Octanol-water partition coefficient (LogP) < 5
3) Hydrogen bond donors < 5
4)Hydrogen bond acceptors < 10

These descriptors represent the characteristics of a compound based on its structure.

```
def lipinski(smiles, verbose=False):

    moldata= []
    for elem in smiles:
        mol=Chem.MolFromSmiles(elem)
        moldata.append(mol)

    baseData= np.arange(1,1)
    i=0
    for mol in moldata:

        desc_MolWt = Descriptors.MolWt(mol)
        desc_MolLogP = Descriptors.MolLogP(mol)
        desc_NumHDonors = Lipinski.NumHDonors(mol)
        desc_NumHAcceptors = Lipinski.NumHAcceptors(mol)

        row = np.array([desc_MolWt,
                        desc_MolLogP,
                        desc_NumHDonors,
                        desc_NumHAcceptors])

        if(i==0):
            baseData=row
        else:
            baseData=np.vstack([baseData, row])
        i=i+1

    columnNames=["MW","LogP","NumHDonors","NumHAcceptors"]
    descriptors = pd.DataFrame(data=baseData,columns=columnNames)

    return descriptors
```

The resulting dataset looks like this:

	MW	LogP	NumHDonors	NumHAcceptors
0	329.528	4.28820	2.0	2.0
1	315.501	3.89810	2.0	2.0
2	412.306	5.70542	0.0	3.0
3	319.383	4.63450	0.0	3.0
4	321.811	4.58780	0.0	3.0
...	...	...	...	...
2813	383.451	4.53580	3.0	5.0
2814	384.439	3.93080	3.0	6.0
2815	384.439	3.93080	3.0	6.0
2816	384.435	4.65920	3.0	5.0
2817	383.451	4.53580	3.0	5.0

2818 rows × 4 columns

MW —> Molecular Weight in Daltons
LogP —> Solubility
NumHDonors —> Number of Hydrogen donors
NumHAcceptors —> Number of Hydrogen Acceptors

I then converted all IC50 'standard_value' to pIC50. This takes the -log of ('standard_value' x 10^-9). This makes the distribution of 'standard_value' more uniform, which is ideal when building a model. I want the response variable to be unbiased in order to build an accurate model.

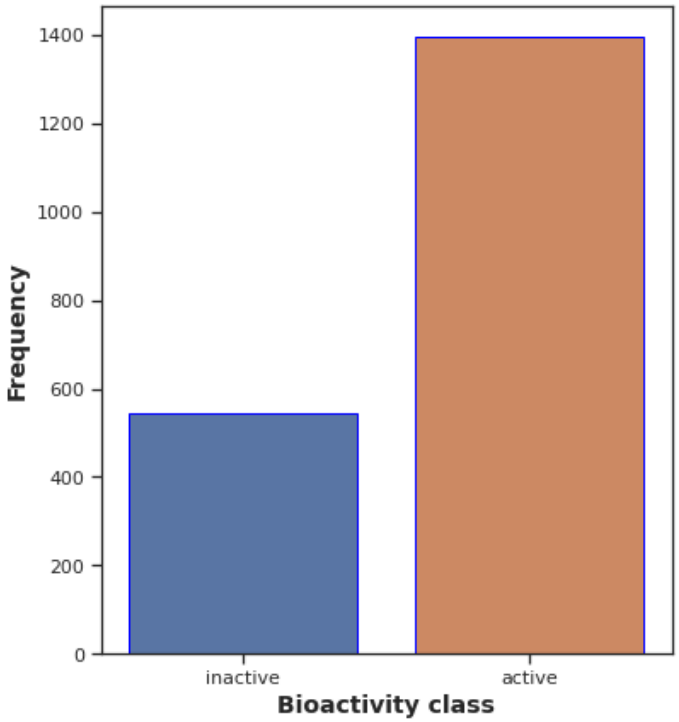
```
import numpy as np

def pIC50(input):
    pIC50 = []

    for i in input['standard_value_norm']:
        molar = i*(10**-9) # Converts nM to M
        pIC50.append(float(-np.log10(molar)))

    input['pIC50'] = pIC50
    x = input.drop('standard_value_norm', 1)

    return x
```

I also need to get rid of all data points with a bioactivity_class of 'intermediate'. This will take out the gray space between inactive and active classes, making our model capable of determining the active compounds clearly. The resulting dataset looks like this:

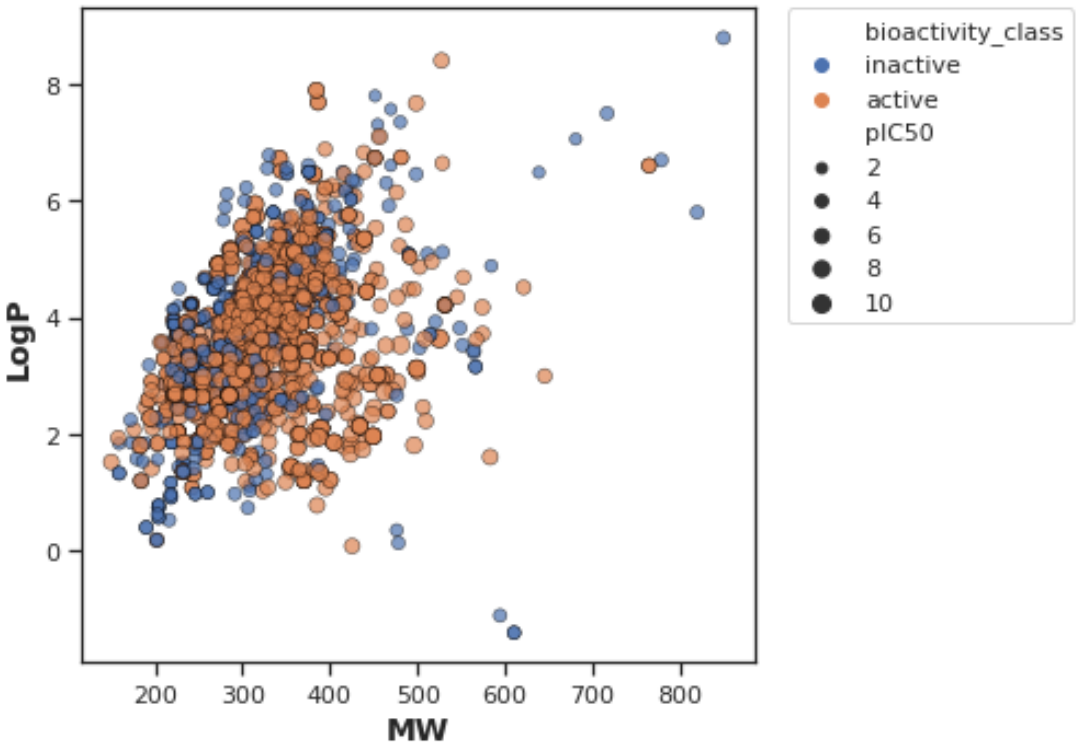
	molecule_chembl_id	canonical_smiles	bioactivity_class	MW	LogP	NumHDonors	NumHAcceptors	pIC50
1	CHEMBL2111947	C[C@]12CC[C@H]3[C@@H](CC=C4C[C@@H](O)CC[C@@]43...	inactive	315.501	3.89810	2.0	2.0	4.301030
2	CHEMBL431859	CCn1c(C(c2ccc(F)cc2)n2ccnc2)c(C)c2cc(Br)ccc21	active	412.306	5.70542	0.0	3.0	6.623423
3	CHEMBL113637	CCn1cc(C(c2ccc(F)cc2)n2ccnc2)c2ccccc21	active	319.383	4.63450	0.0	3.0	7.244125
4	CHEMBL112021	Clc1ccccc1Cn1cc(Cn2ccnc2)c2ccccc21	active	321.811	4.58780	0.0	3.0	7.267606
6	CHEMBL41761	CCn1ccc2cc(C(c3ccc(F)cc3)n3ccnc3)ccc21	active	319.383	4.63450	0.0	3.0	7.387216
...	...	...	...	...	...	...	...	...
2813	CHEMBL4755831	Nc1ccc(C(Cn2ccnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	active	383.451	4.53580	3.0	5.0	8.280669
2814	CHEMBL4750835	Nc1ccc(C(Cn2cncn2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	active	384.439	3.93080	3.0	6.0	6.982967
2815	CHEMBL4745681	Nc1ccc(C(Cn2cnnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	active	384.439	3.93080	3.0	6.0	7.752027
2816	CHEMBL4778401	Oc1ccc(C(=C(Cn2ccnc2)c2cccc(O)c2)c2ccc(O)cc2)cc1	active	384.435	4.65920	3.0	5.0	7.218963
2817	CHEMBL4761359	Nc1cccc(C(Cn2ccnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)c1	active	383.451	4.53580	3.0	5.0	6.357535

1940 rows × 8 columns

Now that I have our clean dataset, I can perform chemical space analysis on this data. I need to first import Seaborn and matplotlib.



This bar plot shows us that I only have inactive and active bioactivity classes. The number of active compounds in our dataset is more than twice the number of inactive compounds.

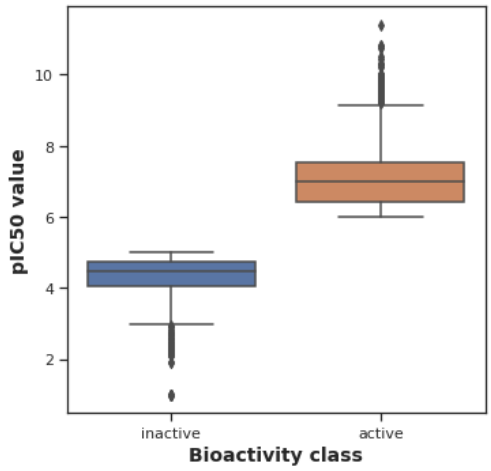


LogP represents the solubility of the compound in water and MW stands for Molecular Weight. There is an overall positive trend between MW and LogP in both active and inactive classes. There are also some clear outliers in our data. I will have to take them out in Step 4.

I need to explore the statistical relevance of the features of our dataset. To do this, I will be using the Mann-Whitney U-test. This test will help us determine if the feature has a significant difference between active and inactive compounds.

```python
def mannwhitney(descriptor, verbose=False):
  # https://machinelearningmastery.com/nonparametric-statistical-significance-tests-in-python/
  from numpy.random import seed
  from numpy.random import randn
  from scipy.stats import mannwhitneyu

# seed the random number generator
  seed(1)

# actives and inactives
  selection = [descriptor, 'bioactivity_class']
  df = aromatase_2class[selection]
  active = df[df.bioactivity_class == 'active']
  active = active[descriptor]

  selection = [descriptor, 'bioactivity_class']
  df = aromatase_2class[selection]
  inactive = df[df.bioactivity_class == 'inactive']
  inactive = inactive[descriptor]

# compare samples
  stat, p = mannwhitneyu(active, inactive)
  #print('Statistics=%.3f, p=%.3f' % (stat, p))

# interpret
  alpha = 0.05
  if p > alpha:
    interpretation = 'Same distribution (fail to reject H0)'
  else:
    interpretation = 'Different distribution (reject H0)'

  results = pd.DataFrame({'Descriptor':descriptor,
                          'Statistics':stat,
                          'p':p,
                          'alpha':alpha,
                          'Interpretation':interpretation}, index=[0])
  filename = 'mannwhitneyu_' + descriptor + '.csv'
  results.to_csv(filename)

  return results
```

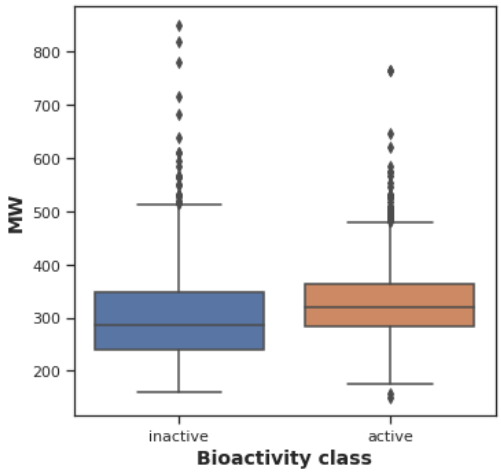
The null hypothesis is that there isn't a significant difference between the particular features of active and inactive compounds. If I use features that don't have significant differences, the model will not be able to clearly differentiate active and inactive compounds.




```
mannwhitney('pIC50')
```

	Descriptor	Statistics	p	alpha	Interpretation
0	pIC50	0.0	1.501908e-257	0.05	Different distribution (reject H0)

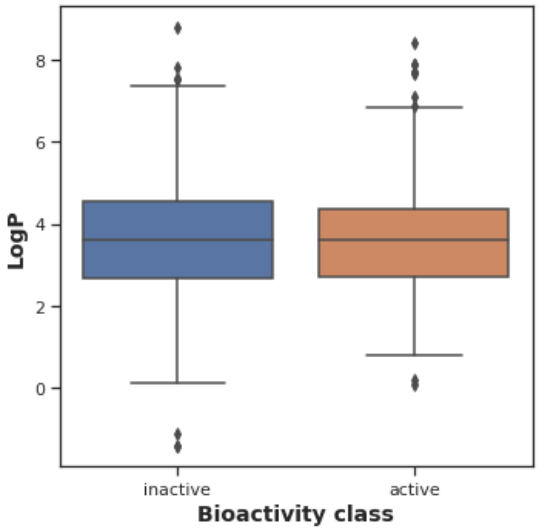
As expected, there is a strong statistical difference between the pIC50 values of active and inactive classes. I classified the pIC50 values based on standard_value.




```
mannwhitney('MW')
```

	Descriptor	Statistics	p	alpha	Interpretation
0	MW	300591.0	4.705227e-13	0.05	Different distribution (reject H0)

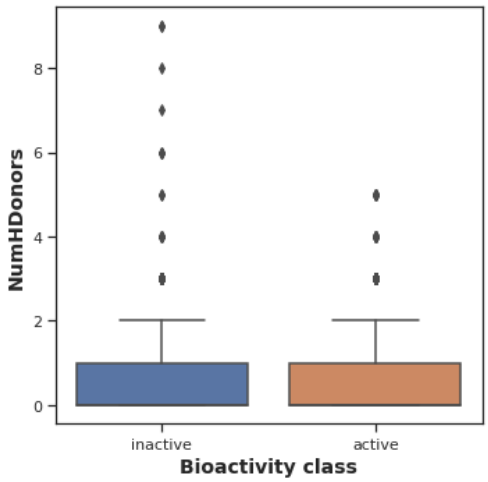
There is a strong statistical difference between inactive compounds and active compounds molecular Weight. This is a good feature.




```
mannwhitney('LogP')
```

	Descriptor	Statistics	p	alpha	Interpretation
0	LogP	377106.0	0.40707	0.05	Same distribution (fail to reject H0)

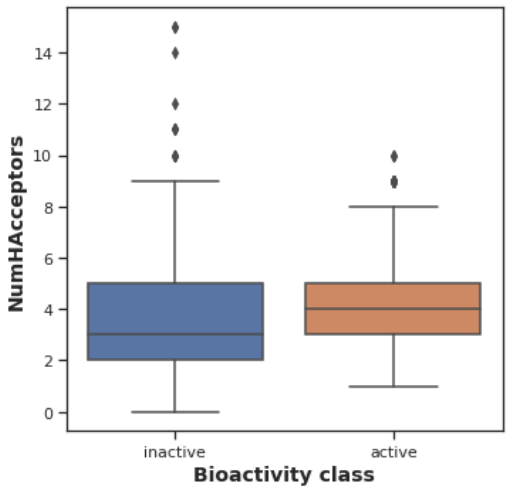
There isn't strong statistical evidence between the solubility of inactive and active class. This is not a good feature to use.




```
mannwhitney('NumHDonors')
```

	Descriptor	Statistics	p	alpha	Interpretation
0	NumHDonors	328006.0	4.488631e-08	0.05	Different distribution (reject H0)

I have a statistically significant difference between the number of Hydrogen Donors between active and inactive compounds. This is a viable feature




```
mannwhitney('NumHAcceptors')
```

	Descriptor	Statistics	p	alpha	Interpretation
0	NumHAcceptors	310464.0	9.606932e-11	0.05	Different distribution (reject H0)

I have a statistically significant difference between the number of Hydrogen Acceptors between active and inactive compounds. This is a viable feature to use.

Lipinski Descriptor Name	P-value	Mann-Whitney U-test Result	Viable in Model?
Molecular light	4.7e-13	Reject H0	Yes
LogP	0.41	Fail to Reject H0	No
# of H donors	4.49e-8	Reject H0	Yes
# of H acceptors	9.61e-11	Reject H0	Yes

Step 3: Calculate Fingerprint Descriptors with PaDEL

ML models work best when the inputs are numbers. Our canonical SMILES notation is a string, making it difficult for the model to accurately predict potency levels. I need to convert this notation to numbers.

I can do this through PaDEL, a software that calculates the 797 descriptors from SMILES Notation. In order to make it interpretable for the computer, PaDEL will one-hot code all 797 descriptors.

	canonical_smiles	molecule_chembl_id
0	CC12CCC(O)CC1=CCC1C2CCC2(C)C(CC3CN3)CCC12	CHEMBL341591
1	C[C@]12CC[C@H]3[C@@H](CC=C4C[C@@H](O)CC[C@@]43...	CHEMBL2111947
2	CCn1c(C(c2ccc(F)cc2)n2ccnc2)c(C)c2cc(Br)ccc21	CHEMBL431859
3	CCn1cc(C(c2ccc(F)cc2)n2ccnc2)c2ccccc21	CHEMBL113637
4	Clc1ccccc1Cn1cc(Cn2ccnc2)c2ccccc21	CHEMBL112021
...	...	...
2812	Nc1ccc(C(Cn2ccnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	CHEMBL4755831
2813	Nc1ccc(C(Cn2cncn2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	CHEMBL4750835
2814	Nc1ccc(C(Cn2cnnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)cc1	CHEMBL4745681
2815	Oc1ccc(C(=C(Cn2ccnc2)c2cccc(O)c2)c2ccc(O)cc2)cc1	CHEMBL4778401
2816	Nc1cccc(C(Cn2ccnc2)=C(c2ccc(O)cc2)c2ccc(O)cc2)c1	CHEMBL4761359

2817 rows × 2 columns

I start with a dataset filled with SMILES Notation. The model can't interpret this.

	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	PubchemFP8	PubchemFP9
0	1	1	1	1	0	0	0	0	0	1
1	1	1	1	1	0	0	0	0	0	1
2	1	1	1	0	0	0	0	0	0	1
3	1	1	1	0	0	0	0	0	0	1
4	1	1	1	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...
2812	1	1	1	0	0	0	0	0	0	1
2813	1	1	1	0	0	0	0	0	0	1
2814	1	1	1	0	0	0	0	0	0	1
2815	1	1	1	0	0	0	0	0	0	1
2816	1	1	1	0	0	0	0	0	0	1

2817 rows × 882 columns

Now, the model can interpret the chemical structure, the primary feature in our model.

Step 4: Comparing various Regression Models

Now that the data is ready to go, I need to determine which model is best suited for our data. To do this, I used lazypredict, a modeling python library, to find the right model. Lazypredict runs 42 ML models at the same time on the same train and test data and gives you the most accurate model.

I loaded up all the data I needed onto my workspace. I combined the fingerprint dataset from part 3 with the lipinski descriptor dataset from part 4. I made sure not to use the LogP descriptor data as it's not a good feature to differentiate active and inactive compounds.

```
[14] aromatase_fp_df = pd.read_csv('aromatase_3class_pIC50_pubchem_fp.csv')
```

```
[15] aromatase_lipinski = pd.read_csv('aromatase_3class_pIC50.csv')
```

```
[26] # assigning X and Y values
     aromatase_fp = aromatase_fp_df.drop('pIC50', axis=1)
     aromatase_3des = aromatase_lipinski[["MW","NumHDonors","NumHAcceptors"]]
     aromatase_X = pd.concat([aromatase_fp, aromatase_3des], axis=1).reindex(aromatase_fp.index)
     aromatase_Y = aromatase_fp_df.pIC50
```

After that, I took out all the low variance features so that the model can accurately determine the difference between active and active compounds. I then split the data in a 4:1 ratio. 80% of our data will help train our model. The other 20% will be used to test and rate that model.

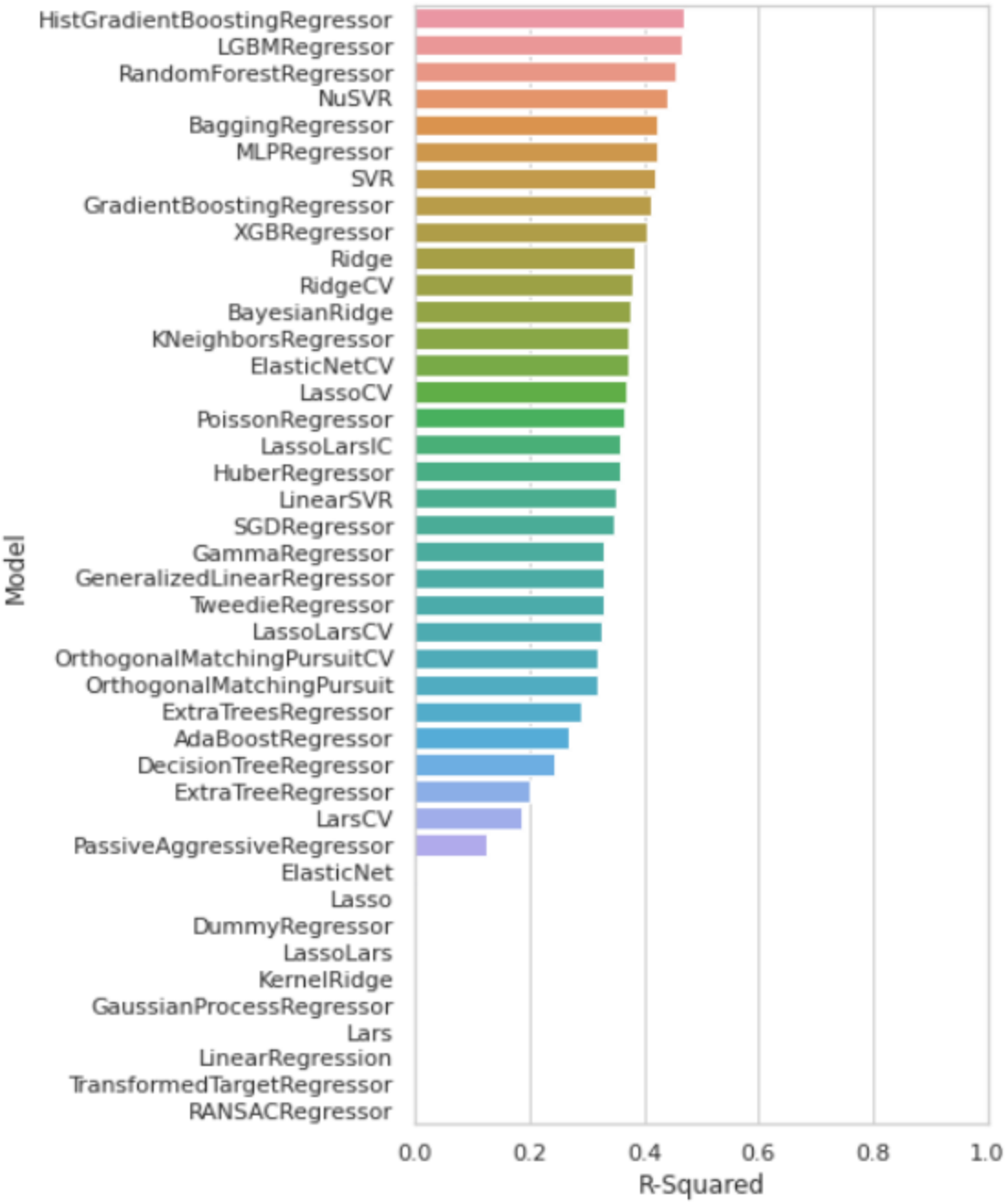
```
# Removing low variance features
from sklearn.feature_selection import VarianceThreshold
selection = VarianceThreshold(threshold=(.8 * (1 - .8)))
aromatase_X = selection.fit_transform(aromatase_X)
aromatase_X.shape
```

```
(2817, 155)
```

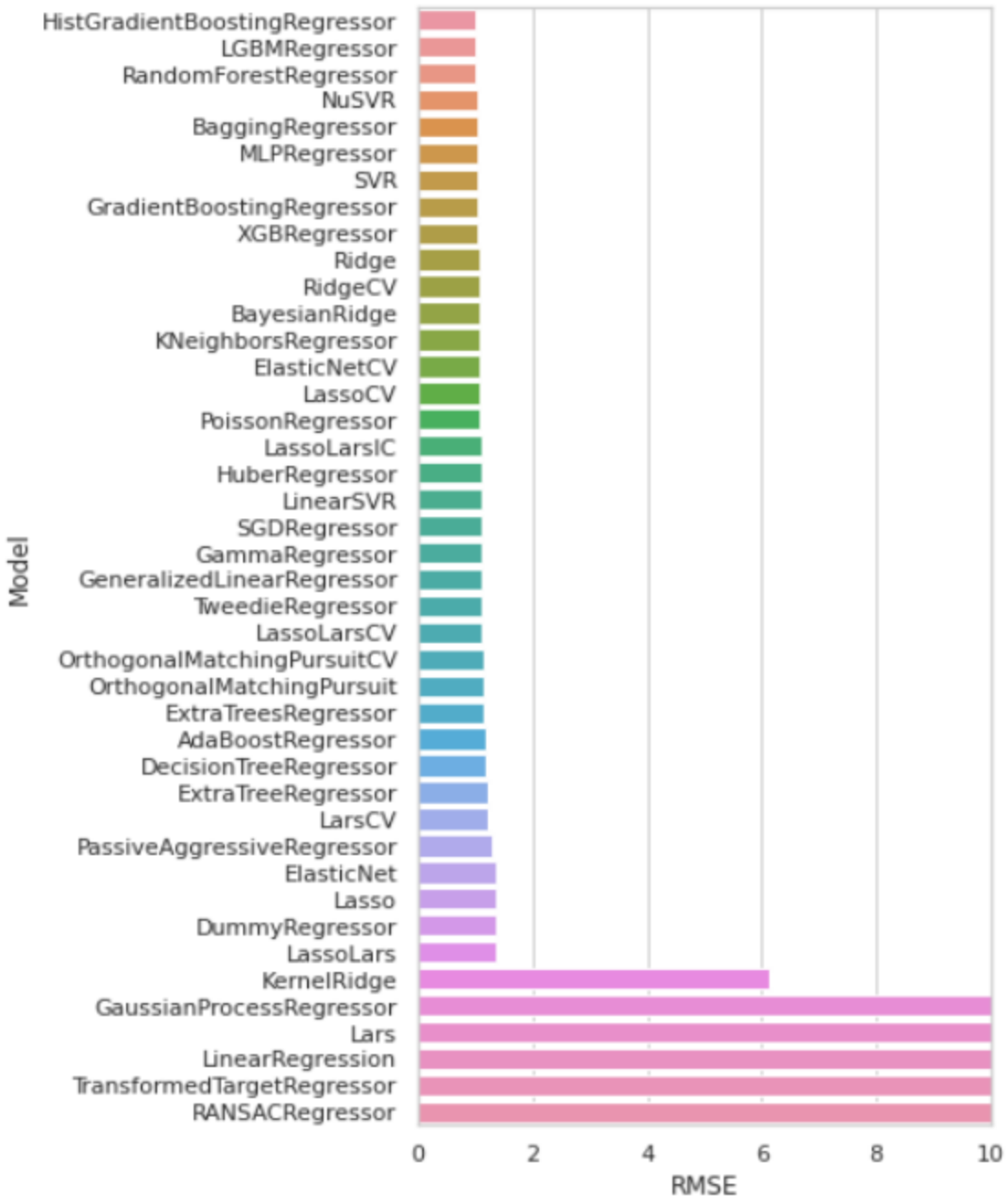
```
[29] # Data splitting with 4:1 ratio
     aromatase_X_train, aromatase_X_test, aromatase_Y_train, aromatase_Y_test = train_test_split(aromatase_X, aromatase_Y, test_size=0.2)
```

This is where lazypredict helps out. Instead of running 20 different models individually and finding the best one, I can run 42 of them through lazypredict and pick the best one. This saves a ton of time and effort.

Then I can generate graphs that compare the performance of the 42 models on multiple factors like R-squared, RMSE, and Calculation Time.



The R-squared value is a measure of how accurate the model is. A higher R-squared value is preferred. HistGradientBoostingRegressor has the highest R-squared at 0.47.



The RMsE is a measure of error between predicted and actual data. A lower RMSE is preferred.
Again, HistGradientBoostingRegressor is the best model as it has the lowest RMSE.

In a professional setting calculation time may be an important factor. But in our case, it really doesn't make a difference as all these models complete generation with the span of seconds.

The clear choice for the model that is best suited for our data is HistGradientBoostingRegressor.
It had the highest R-squared value and lowest RMSE value.

Step 5: Building HistGradientBoostingRegressor Model

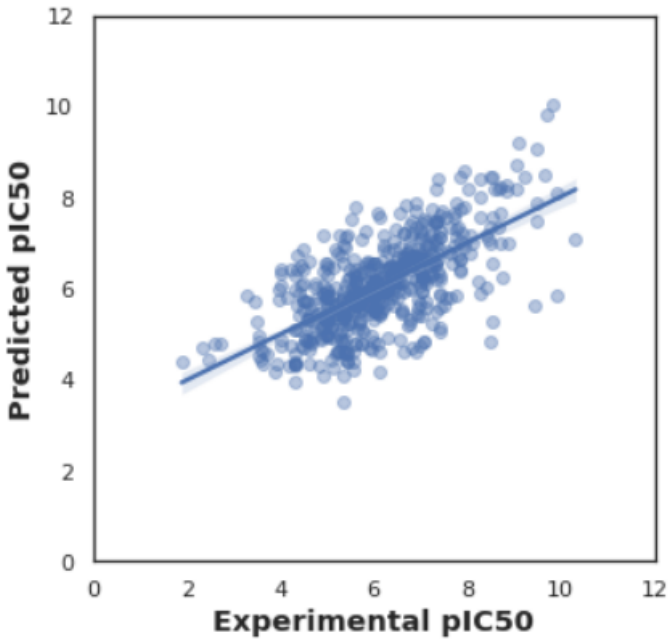
I repeated the initial steps from part 4. I combined the two datasets of fingerprint and Lipinski descriptor data. I then deleted all the low variance features of the dataset. I split the dataset in a 80:20 ratio. 80% is dedicated to training the model. The other 20% id used to test the model.

I found that the HistGradientBoostingRegressor is the most accurate model for our data. So, I built a model accordingly.

```
model = HistGradientBoostingRegressor()
model.fit(aromatase_X_train, aromatase_Y_train)
aromatase_pred_r2 = model.score(aromatase_X_test, aromatase_Y_test)
aromatase_pred_r2
```

```
0.43147222280140884
```

I created a graph to compare the predicted pIC50 values given by the model when test data was inputted against the actual test pIC50 values. We can see that the model is accurate.



Works Cited

https://www.nationalbreastcancer.org/breast-cancer-facts

https://www.cancer.org/cancer/breast-cancer/risk-and-prevention/aromatase-inhibitors-for-lowering-breast-cancer-risk.html

https://github.com/dataprofessor