B.TECH INDEPENDENT PROJECT(IP)

Membrane Permeability Prediction using Machine Learning Algorithms

Suraj Kumar Jha - 2021209, Rajat jaiswal - 2021184, Tarun Bansal - 2021210

Introduction

The aim of this project is to predict membrane permeability using machine learning algorithms and comparing the score on various models to find which is the best model for given datasets of SMILES and logP or Energy Gap, thereby enhancing our understanding of biological concepts. Cell Membrane permeability prediction is crucial in pharmaceutical research for assessing the ability of drugs to cross biological membranes based on different factors associated with the physico-chemical properties of the materials used in that drug, impacting drug efficacy and toxicity.

Objective

The objective of this study is to compare the results of predictive models for membrane permeability of chemicals by utilizing machine learning techniques, based on their unique SMILE structures and other physico-chemical properties. We aim to collect and organize data for various chemicals, including their structural and physico-chemical characteristics, and train a machine learning model to identify patterns that correlate with membrane permeability. By comparing the model's predictions based on SMILE structures alone, physico-chemical properties alone, and a combination of both, we will determine the most accurate method for predicting membrane permeability.

Methodology

Data Collection and Exploration:

- Curated datasets from public repositories like github and research papers like pubmed and google scholar.

```
# Columns format:
# 1) SMILES string.
# 2) Martini unimer representation.
# 3) Delta_G_water/octanol predicted from ALOGPS [kcal/mol].
# 4) Delta_G_water/octanol of the Martini unimer [kcal/mol].
# 5) apKa. NOT indicates that the compound has a apKa higher than 20.
# 6) bpKa. NOT indicates that the compound has a bpKa lower than -10.
# 7) Acidity of the compound: acidic (A), basic (B), zwitterionic (Z), neutral (N).
#    We consider as neutral all compounds presenting an apKa larger than 16
#    and a bpKa smaller than -2.
# 8) Logarithm of the permeability coefficient log_10(P). NOT indicates the
#    impossibility of predicting the permeability coefficient of zwitterionic
#    compounds that map to coarse-grained unimers.
#
#################################################################################################


CC C5 -1.977 -1.656 NOT NOT N 0.269
CN P2 1.455 0.920 NOT 10.08 B -4.686
CO P3 1.894 2.106 15.78 -2.46 A -2.865
CF Na -0.549 -0.595 NOT NOT N -1.005
FF N0 -1.304 -1.009 NOT NOT N -0.191
C=C N0 -1.235 -1.009 NOT NOT N -0.191
C=O P2 0.947 0.920 NOT -8.09 N -2.012
O=O P3 1.784 2.106 NOT NOT N -2.865
C#C P1 0.041 0.540 NOT NOT N -1.574
CCC C3 -3.006 -2.930 NOT NOT N 0.023
CCN P1 0.275 0.540 NOT 10.23 B -4.391
CCO P1 0.549 0.540 16.47 -2.16 N -1.574
CCF C5 -1.386 -1.656 NOT NOT N 0.269
```

Ln 11, Col 38 Spaces: 4 UTF-8 CRLF Plain Text

Since, parameters other than SMILES and logP were not essential from a permeability point of view. Hence removed the rest of the columns and turned the existing text file into csv file to handle it better.

test > Codes > Lazy Predict > data_ip.csv > data

```
SMILES,Column8
CC,0.269
CN,-4.686
CO,-2.865
CF,-1.005
FF,-0.191
C=C,-0.191
C=O,-2.012
O=O,-2.865
C#C,-1.574
```

- Explored public databases like Kaggle and literature sources such as PubMed for relevant datasets and research articles.

test > Codes > Lazy Predict > input_data.csv > data

```
SMILES,Energygap
Cc1ccc(cc1)C(F)(F)F,197.7494212
OC(=O)CCCCl,247.4939422
CC(C)(Oc1ccc(CCNC(=O)c2ccc(Cl)cc2)cc1)C(=O)O,164.7123274
Nc1ccc(Cl)c(Cl)c1,169.0277068
C[C@@H](CCO)CCC=C(C)C,209.5698082
OC(C=C)C=C,210.9791935
O=N(=O)c1ccc(Cl)c(c1)N(=O)=O,168.833179
O=CC1CCCCC1,213.4515789
COc1ccc(cc1)C#C,171.1919853
Brc1cncnc1,188.3449438
COc1ccc(cc1[C@]12C[C@H]3C[C@H](C[C@H](C3)C1)C2)c1ccc2cc(ccc2c1)C(=O)O,139.4927407
N[C@@H](CC(=O)O)C(=O)O,218.3248138
CCC#CCC,230.1295131
Cc1cccc2cc[nH]c12,170.4672124
CC(=O)c1cccc(C)c1,174.692858
Nc1ccccc1C(=O)c1ccccc1,142.0234845
N#Cc1sccc1,175.4414763
Clc1ccccc1C(=O)c1ccccc1,168.8990674
CC(=C)C(=O)N,207.8416485
```

Feature Engineering:

- Identified key features influencing membrane permeability, including chemical nature, SMILE structure, and physico-chemical properties.
  `canonical_smiles(['C=CCC'])` and `canonical_smiles(['CCC=C'])`
  For both, Output will be `['C=CCC']` as both of them are representations of the same molecule.

- Utilized RDKit and Mordred libraries to calculate molecular descriptors.

```
%pip install rdkit-pypi
%pip install mordred
```

- Molecular descriptors are a number of numerical features from the SMILES string which comes from physico-chemical properties like molecular weights, number of valence electrons, the magnitude of partial charges, number of bonds, number of atoms of a particular element, etc.
- Molecular descriptors can be very useful as predictive features for models. In the RDKit package, there are 200 such descriptors which can automatically be generated using smiles.

- Generated fingerprints (e.g., Morgan Fingerprints) as descriptors for predictive modeling.

```
def morgan_fpts(data):

    Morgan_fpts = []

    for i in data:

        mol = Chem.MolFromSmiles(i)

        fpts =  AllChem.GetMorganFingerprintAsBitVect(mol,2,2048)

        mfpts = np.array(fpts)

        Morgan_fpts.append(mfpts)

    return np.array(Morgan_fpts)
```

Model Development:

- Explored various machine learning algorithms such as LASSO regression and Multi-Layer Perceptron (MLP) to analyze the result of cell membrane permeability prediction.
  The multi-layer perceptron (MLP) is another artificial neural network process containing a number of layers.
- Implemented regularization techniques to reduce model complexity and overfitting.
  Regularization techniques prevent overfitting in machine learning models and improve their generalization ability. They can help reduce a model's complexity by penalizing parameters that are too large and encouraging models to use simpler and more interpretable structures.
- Used Lazy Predict tool for evaluating model accuracy and computational efficiency.
- With the help of Rdkit libraries, more than 200 descriptors can be found based on SMILE structure of the molecule, for example molecular weight, heavy atomic molecular weight or exact molecular weight.
- Using Mordred, we can obtain descriptors more than 1800 based on SMILES structure.

- RDKit

	MaxEStateIndex	MinEStateIndex	MaxAbsEStateIndex	MinAbsEStateIndex	qed	MolWt	HeavyAtomMolWt	ExactMolWt
0	12.550510	-5.076351	12.550510	1.008796	0.546828	160.138	153.082	160.049985
1	10.676844	-3.333333	10.676844	1.840718	0.569323	122.551	115.495	122.013457
2	13.050084	-4.111425	13.050084	0.722809	0.790287	361.825	341.665	361.108086
3	7.402685	-0.449630	7.402685	0.074321	0.582519	162.019	156.979	160.979905
4	8.095237	-4.484184	8.095237	1.886963	0.606746	156.269	136.109	156.151415
...	...	...	...	...	...	...	...	...
2868	9.505488	-3.873136	9.505488	0.973292	0.373065	398.426	374.234	398.135324
2869	7.651157	-0.170718	7.651157	0.017477	0.625891	213.099	208.059	211.929533
2870	13.017078	-3.813937	13.017078	0.325694	0.823664	247.294	230.158	247.120843
2871	7.592407	-0.348333	7.592407	0.030556	0.519376	134.203	128.155	134.019021
2872	7.522616	-2.919676	7.522616	0.821250	0.557704	122.167	112.087	122.073165

873 rows × 208 columns

- Mordred

mordred_descriptors

Python

	ABC	ABCGG	nAcid	nBase	SpAbs_A	SpMax_A	SpDiam_A	SpAD_A	SpMAD_A	LogEE_A	...	SRW10	TSRW10	MW	AMW	WPath	WPol	Zagreb1	Zagreb2	mZagreb1	mZagreb2
0	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	12.527341	2.311476	4.622953	12.527341	1.138849	3.302522	...	9.182249	41.326257	160.049985	8.891666	152	13	54.0	59.0	5.284722	2.333333
1	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	1	0	7.727407	1.931852	3.863703	7.727407	1.103915	2.752227	...	7.321850	31.336140	122.013457	8.715247	52	4	24.0	22.0	3.861111	1.833333
2	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	1	0	30.648742	2.324224	4.648448	30.648742	1.225950	4.118873	...	9.931735	59.295845	361.108086	8.024624	1882	35	124.0	139.0	9.729167	5.486111
3	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	10.792280	2.245827	4.491654	10.792280	1.199142	3.099448	...	8.806724	37.839725	160.979905	11.498565	84	10	42.0	46.0	4.083333	2.027778
4	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	12.133645	2.047810	4.095621	12.133645	1.103059	3.219224	...	8.131825	38.565088	156.151415	5.037142	194	9	42.0	41.0	5.472222	2.750000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2868	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	38.253985	2.374938	4.749876	38.253985	1.366214	4.258939	...	10.073357	62.971588	398.135324	7.656449	2014	41	142.0	164.0	6.166667	6.361111
2869	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	13.098358	2.369838	4.633950	13.098358	1.309836	3.261311	...	9.161465	53.745115	211.929533	14.128636	105	12	52.0	61.0	2.833333	2.222222
2870	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	22.978744	2.442763	4.885526	22.978744	1.276597	3.800055	...	9.763593	50.871918	247.120843	7.060596	574	28	88.0	103.0	6.055556	4.277778
2871	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	12.170709	2.322596	4.516123	12.170709	1.352301	3.160409	...	8.914761	51.887188	134.019021	8.934601	79	9	46.0	53.0	1.972222	2.027778
2872	module 'numpy' has no attribute 'float'.\n'np....	module 'numpy' has no attribute 'float'.\n'np....	0	0	10.891331	2.263821	4.527642	10.891331	1.210148	3.099901	...	8.876126	37.975562	122.073165	6.424903	82	11	42.0	47.0	4.083333	2.055556

2873 rows × 1826 columns

Biological Insights:

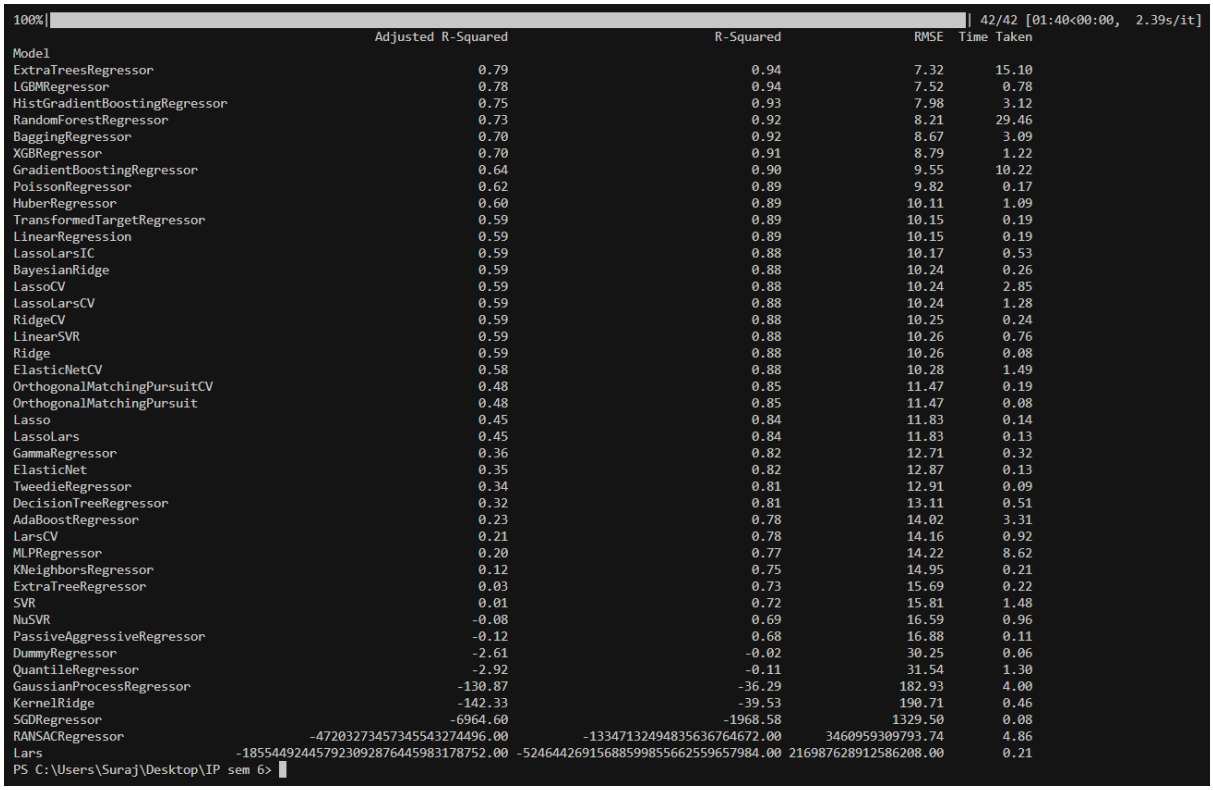
- Explored cell membrane structure, permeability factors, and experimental methods.
- Identified main physio-chemical properties determining permeability and molecular descriptors as predictive features.

test > Codes > Lazy Predict > descriptors.csv > data

```
MaxEStateIndex,MinEStateIndex,MaxAbsEStateIndex,MinAbsEStateIndex,qed,MolWt,HeavyAtomMolWt,ExactMolWt,NumValenceElectrons,NumRadicalElectrons,MaxPartialChar
12.550509731670445,-5.076351095993956,12.550509731670445,1.008796296296296,0.5468276913734111,160.13800000000006,153.082,160.049984884,60,0,0.41590983407335
10.676844135802469,-3.3333333333333333,10.676844135802469,1.8407175925925923,0.5693226475084426,122.55099999999996,115.49499999999999,122.013457144,42,0,0.30
13.050084195647448,-4.111425039348887,13.050084195647448,0.7228089497915431,0.7902868665254312,361.82499999999976,341.66500000000013,361.1080858000005,132,0
7.402685185185845,-0.4496296296296294,7.402685185185845,0.07432098765432071,0.5825188519698148,162.01900000000003,156.97899999999998,160.97990452,48,0,0.1
8.095237150415722,-4.484183673469385,8.095237150415722,1.8869631571764514,0.6067463726043824,156.26900000000018,136.10899999999998,156.15141526,66,0,0.2100
7.261574074074074,-2.7384259259259256,7.261574074074074,1.0606481481481478,0.4886128495679416,84.11799999999997,76.054,84.057514876,34,0,0.21137876964578545
10.514890400604687,-1.1190123456790118,10.514890400604687,0.7448148148148144,0.5431088805091585,202.55300000000003,199.529,201.978134256,68,0,0.29424789610
11.281771541950114,-3.7525462962962965,11.281771541950114,2.1567129629629633,0.4722882253981138,112.17199999999993,100.07599999999998,112.088815004,46,0,0.1
7.544490740740741,-2.867958120748 3,7.544490740740741,0.2658408919123205,0.5280532292060911,132.162,124.09799999999997,132.057514876,50,0,0.12422420457248952
7.063217592592593,-0.2933333333333333,7.063217592592593,0.12037037037037002,0.5685144631214077,158.98600000000002,155.962,157.947960196,36,0,0.11485330870864
12.312143119113511,-4.614573505216474,12.312143119113511,0.8776765511007489,0.5197041292773107,412.52899999999966,384.3050000000002,412.20384475600076,158,0
10.954357993197279,-3.5174768518518515,10.954357993197279,0.7492129629629631,0.4520212550353777,133.103,126.04699999999997,133.037507704,52,0,0.32057810707
7.042361111111111,-3.0507723922902503,7.042361111111111,1.5315972222222222,0.390833250197772,82.14599999999994,72.06599999999999,82.07825032,34,0,0.039942671
7.689907407407407,-2.7550925925925925,7.689907407407407,0.2434027777777772,0.5649684553619617,131.178,122.10599999999998,131.073499288,50,0,0.16593394953863
11.719839380196522,-3.1591317082388 53,11.719839380196522,0.8730631141345426,0.5379444707224508,134.17800000000003,124.09799999999997,134.07316494,52,0,0.159
12.856447467876036,-1.2976851851851852,12.856447467876036,0.18016298185941038,0.5933842323034868,197.23700000000005,186.14899999999994,197.084063972,74,0,0.
8.329953703703703,-0.1513425925925922,8.329953703703703,0.012453703703703356,0.496284724723427,109.15299999999998,106.12899999999999,108.998620096,34,0,0.11
12.66089191232048,-1.1629629629629628,12.66089191232048,0.5714710884353738,0.7018029704350378,216.66700000000006,207.59499999999997,216.034192588,74,0,0.194
10.824189814814815,-2.9342592592592602,10.824189814814815,0.467476851851851 7,0.4489458287260874,85.10599999999997,78.05,85.052763844,34,0,0.2432329843139914
```

Results and Analysis:

- Predicted membrane permeability using machine learning models.
- Analyzed model performance metrics such as R-squared, adjusted R-squared, RMSE, and computational time.



```
100%|████████████████████████████████████████████████████████████| 42/42 [01:40<00:00,  2.39s/it]
                                                  Adjusted R-Squared                  R-Squared                 RMSE  Time Taken
Model
ExtraTreesRegressor                                             0.79                       0.94                 7.32       15.10
LGBMRegressor                                                   0.78                       0.94                 7.52        0.78
HistGradientBoostingRegressor                                   0.75                       0.93                 7.98        3.12
RandomForestRegressor                                           0.73                       0.92                 8.21       29.46
BaggingRegressor                                                0.70                       0.92                 8.67        3.09
XGBRegressor                                                    0.70                       0.91                 8.79        1.22
GradientBoostingRegressor                                       0.64                       0.90                 9.55       10.22
PoissonRegressor                                                0.62                       0.89                 9.82        0.17
HuberRegressor                                                  0.60                       0.89                10.11        1.09
TransformedTargetRegressor                                      0.59                       0.89                10.15        0.19
LinearRegression                                                0.59                       0.89                10.15        0.19
LassoLarsIC                                                     0.59                       0.88                10.17        0.53
BayesianRidge                                                   0.59                       0.88                10.24        0.26
LassoCV                                                         0.59                       0.88                10.24        2.85
LassoLarsCV                                                     0.59                       0.88                10.24        1.28
RidgeCV                                                         0.59                       0.88                10.25        0.24
LinearSVR                                                       0.59                       0.88                10.26        0.76
Ridge                                                           0.59                       0.88                10.26        0.08
ElasticNetCV                                                    0.58                       0.88                10.28        1.49
OrthogonalMatchingPursuitCV                                     0.48                       0.85                11.47        0.19
OrthogonalMatchingPursuit                                       0.48                       0.85                11.47        0.08
Lasso                                                           0.45                       0.84                11.83        0.14
LassoLars                                                       0.45                       0.84                11.83        0.13
GammaRegressor                                                  0.36                       0.82                12.71        0.32
ElasticNet                                                      0.35                       0.82                12.87        0.13
TweedieRegressor                                                0.34                       0.81                12.91        0.09
DecisionTreeRegressor                                           0.32                       0.81                13.11        0.51
AdaBoostRegressor                                               0.23                       0.78                14.02        3.31
LarsCV                                                          0.21                       0.78                14.16        0.92
MLPRegressor                                                    0.20                       0.77                14.22        8.62
KNeighborsRegressor                                             0.12                       0.75                14.95        0.21
ExtraTreeRegressor                                              0.03                       0.73                15.69        0.22
SVR                                                             0.01                       0.72                15.81        1.48
NuSVR                                                          -0.08                       0.69                16.59        0.96
PassiveAggressiveRegressor                                     -0.12                       0.68                16.88        0.11
DummyRegressor                                                 -2.61                      -0.02                30.25        0.06
QuantileRegressor                                              -2.92                      -0.11                31.54        1.30
GaussianProcessRegressor                                     -130.87                      -36.29               182.93        4.00
KernelRidge                                                  -142.33                      -39.53               190.71        0.46
SGDRegressor                                                 -6964.60                    -1968.58              1329.50        0.08
RANSACRegressor                  -472032734573455432744960.00          -133471324948356367646720.00      3460959309793.74        4.86
Lars                  -1855449244579230928764459831787520.00 -52464426915688599855662559657984.00 216987628912586208.00        0.21
PS C:\Users\Suraj\Desktop\IP sem 6>
```

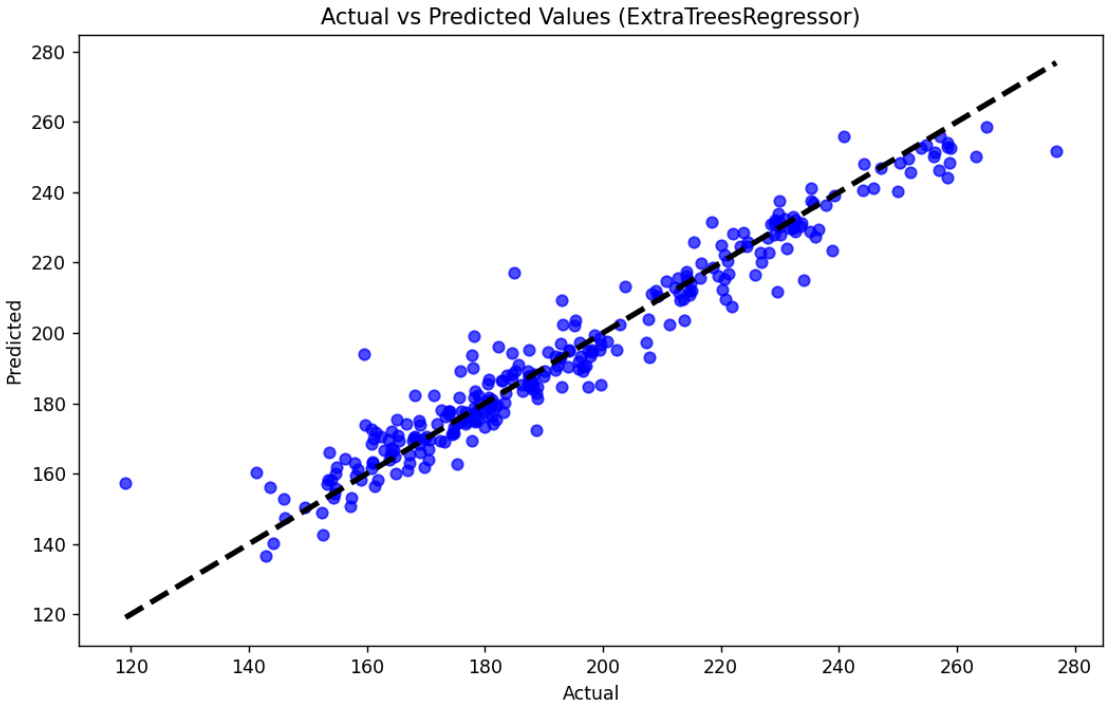
- Based on the results provided, the ExtraTreesRegressor is the best performing model, with the highest Adjusted R-Squared (0.79) and R-Squared (0.94) values, and a relatively low RMSE (7.32). Additionally, it has a reasonable computation time (15.10 seconds).

overview and analysis of the top performing models listed, highlighting their strengths, weaknesses, and typical use cases.

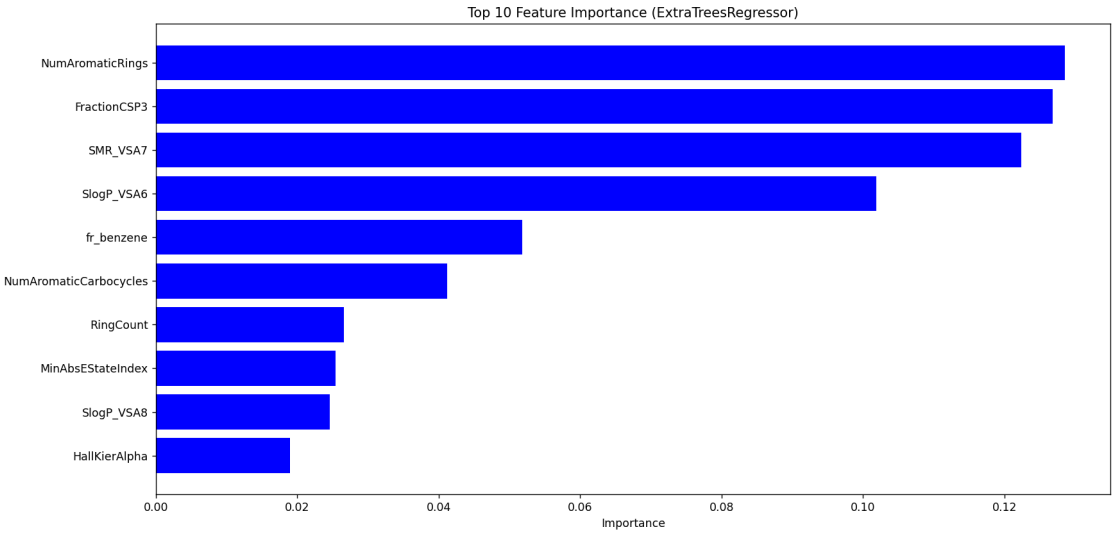
1. ExtraTreesRegressor

- Theoretical Details:
    - ➢ Type: Ensemble learning method
    - ➢ Algorithm: Uses multiple decision trees to build a model (similar to Random Forest).
    - ➢ Working: Each tree is trained on a random subset of the data, and splits in each tree are determined by random subsets of features.
- Strengths:
    - ➢ Reduces overfitting due to averaging multiple trees.
    - ➢ Handles high-dimensional data well.
    - ➢ Robust to noisy data and outliers.
- Weaknesses:
    - ➢ Computationally intensive, especially with a large number of trees.
    - ➢ Can be less interpretable than single decision trees.
    - ➢ Typical Use Cases: Any regression task where model performance is more critical than model interpretability.
- Analysis:
    - ➢ Performance: Highest R-Squared (0.94) and Adjusted R-Squared (0.79), indicating excellent fit to the data.
    - ➢ RMSE: 7.32, which is low, suggesting accurate predictions.
    - ➢ Time Taken: 15.10 seconds, which is relatively high but reasonable given the model's complexity.
    - ➢ Scatter Plot: Actual vs Predicted Values

- Scatter Plot: Actual vs Predicted Values



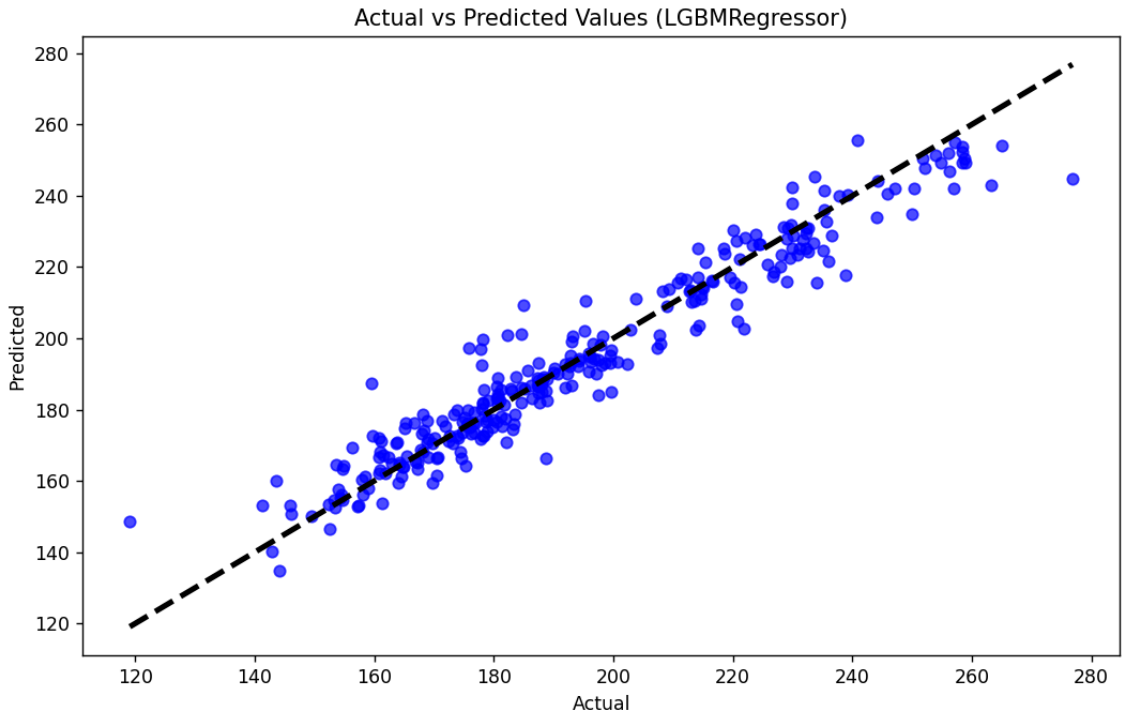
- Feature importance plot



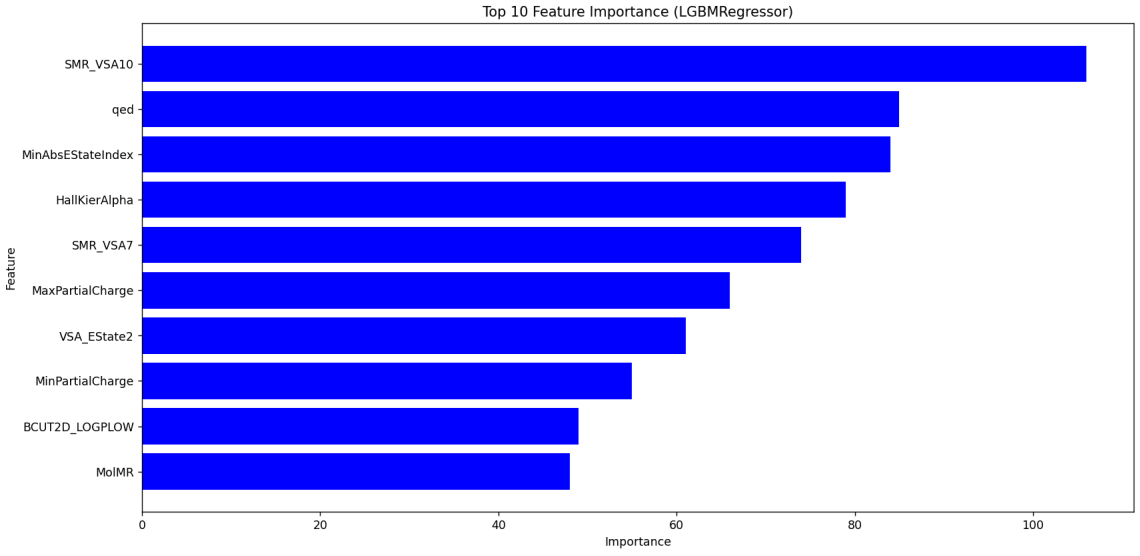
2. LGBMRegressor (LightGBM)

- Theoretical Details:
  - ➢ Type: Gradient Boosting Framework
  - ➢ Algorithm: Uses tree-based learning algorithms, optimized for faster training and lower memory usage.
  - ➢ Working: Builds trees leaf-wise (best-first), which tends to converge faster and achieve higher accuracy.
- Strengths:
  - ➢ Fast training speed and high efficiency.
  - ➢ Capable of handling large datasets.
  - ➢ Supports parallel and GPU learning.
- Weaknesses:
  - ➢ Sensitivity to hyperparameters.
  - ➢ Can overfit if not properly tuned.
  - ➢ Typical Use Cases: Large datasets, high-dimensional data, and when training speed is critical.
- Analysis:
  - ➢ Performance: High R-Squared (0.94) and Adjusted R-Squared (0.78).
  - ➢ RMSE: 7.52, slightly higher than ExtraTrees but still low.
  - ➢ Time Taken: 0.78 seconds, significantly faster, making it highly efficient.

- Scatter Plot: Actual vs Predicted Values



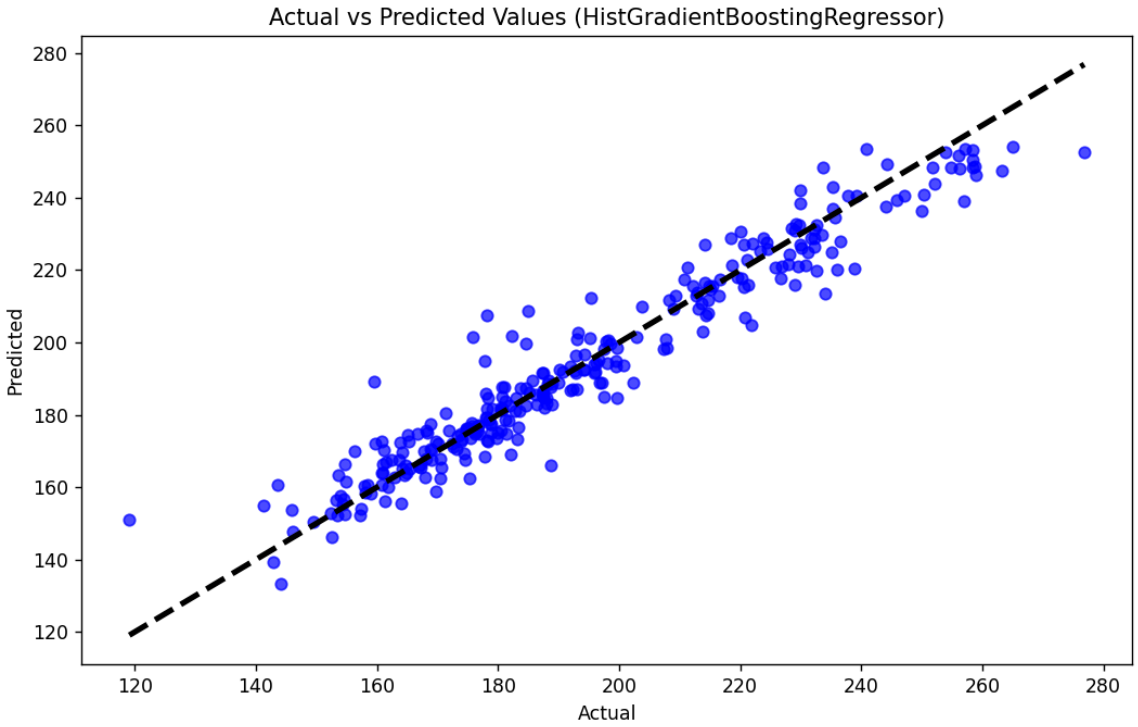
- Feature importance plot



3. HistGradientBoostingRegressor

- Theoretical Details:
  - ➢ Type: Gradient Boosting variant
  - ➢ Algorithm: Uses histogram-based gradient boosting for faster computations.
  - ➢ Working: Discretized continuous features into bins, reducing the number of splitting points.
- Strengths:
  - ➢ Faster training time with large datasets.
  - ➢ Efficient memory usage.
  - ➢ Can handle categorical features directly.
- Weaknesses:
  - ➢ Can be less interpretable than simpler models.
  - ➢ Still requires careful tuning of hyperparameters.
  - ➢ Typical Use Cases: Large datasets, especially when speed and memory efficiency are crucial.
- Analysis:
  - ➢ Performance: R-Squared (0.93) and Adjusted R-Squared (0.75), indicating very good fit.
  - ➢ RMSE: 7.98, higher than ExtraTrees and LGBM but still reasonable.
  - ➢ Time Taken: 3.12 seconds, a good balance between speed and performance.

- Scatter Plot: Actual vs Predicted Values



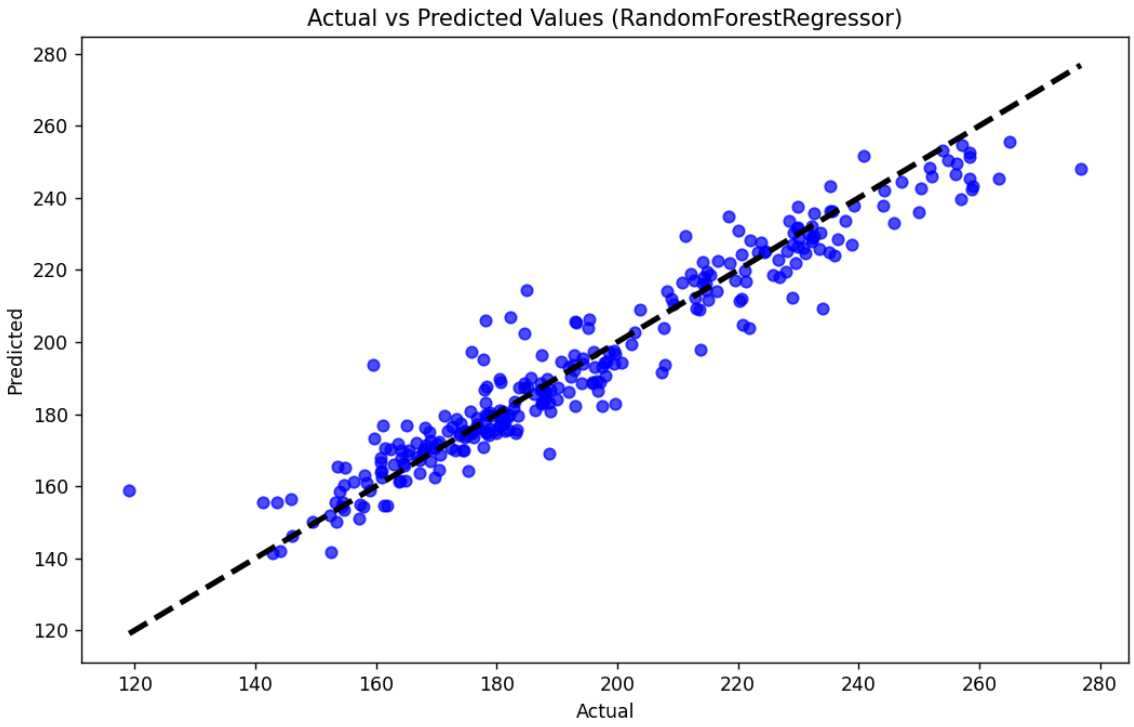
- Feature importance plot

  HistGradientBoostingRegressor does not support feature importances.

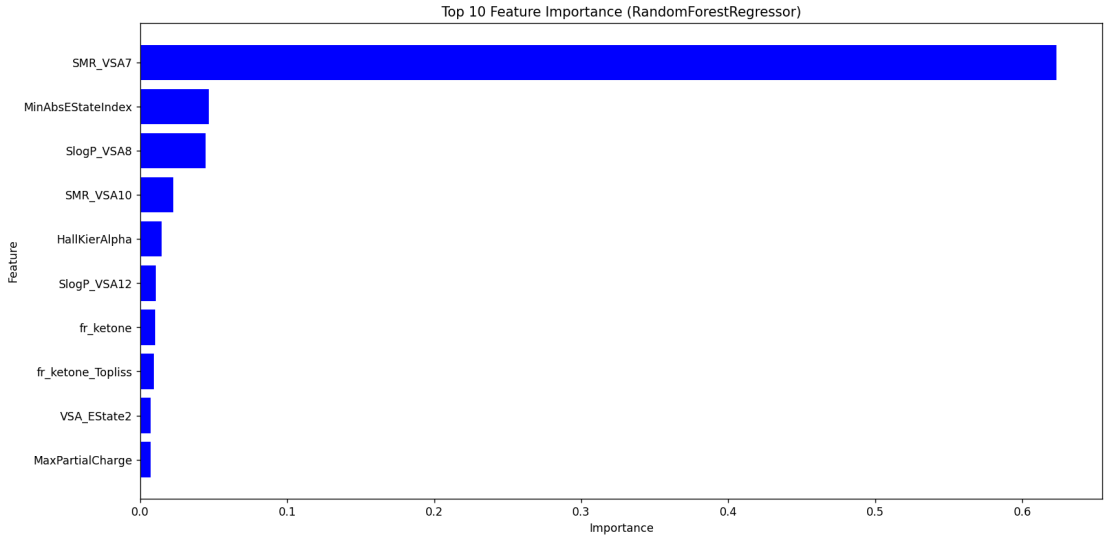
4. RandomForestRegressor

- Theoretical Details:
  - ➢ Type: Ensemble learning method
  - ➢ Algorithm: Constructs a multitude of decision trees during training and outputs the average prediction.
  - ➢ Working: Each tree is trained on a bootstrap sample of the data and considers a random subset of features for splitting.
- Strengths:
  - ➢ Reduces overfitting compared to single decision trees.
  - ➢ Easy to parallelize.
  - ➢ Robust to outliers and noise.

- Weaknesses:
    - ➢ Computationally expensive with large datasets and many trees.
    - ➢ Less interpretable than single decision trees.
    - ➢ Typical Use Cases: General-purpose regression tasks where accuracy is important.
- Analysis:
    - ➢ Performance: R-Squared (0.92) and Adjusted R-Squared (0.73).
    - ➢ RMSE: 8.21, which is slightly higher but still acceptable.
    - ➢ Time Taken: 29.46 seconds, the longest among the top models, indicating a trade-off between time and performance.
- Scatter Plot: Actual vs Predicted Values



- Feature importance plot



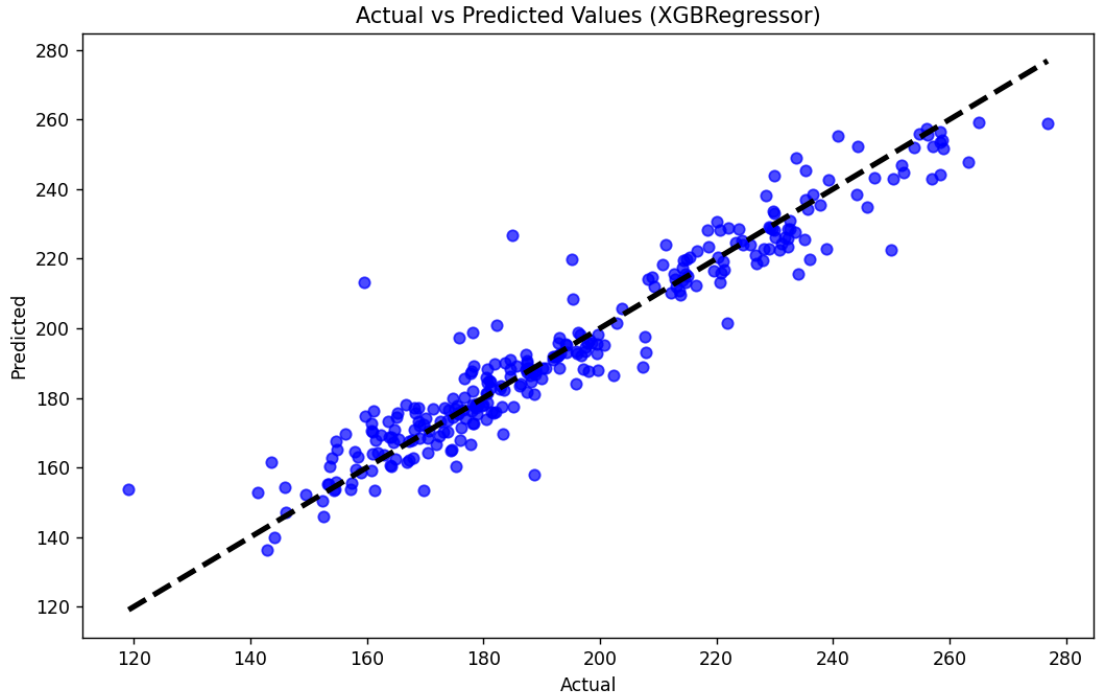
5. BaggingRegressor

- Theoretical Details:
  - ➢ Type: Ensemble learning method
  - ➢ Algorithm: Uses bootstrap aggregating (bagging) to reduce variance.
  - ➢ Working: Trains multiple instances of a base estimator on different subsets of the training data and averages their predictions.
- Strengths:
  - ➢ Reduces overfitting.
  - ➢ Improves stability and accuracy.
- Weaknesses:
  - ➢ Can be computationally expensive.
  - ➢ Dependent on the performance of the base estimator.
  - ➢ Typical Use Cases: Situations where reducing variance is crucial and the base estimator is prone to overfitting.
- Analysis:
  - ➢ Performance: R-Squared (0.92) and Adjusted R-Squared (0.70).
  - ➢ RMSE: 8.67, higher than the top models but still reasonable.
  - ➢ Time Taken: 3.09 seconds, fairly efficient.

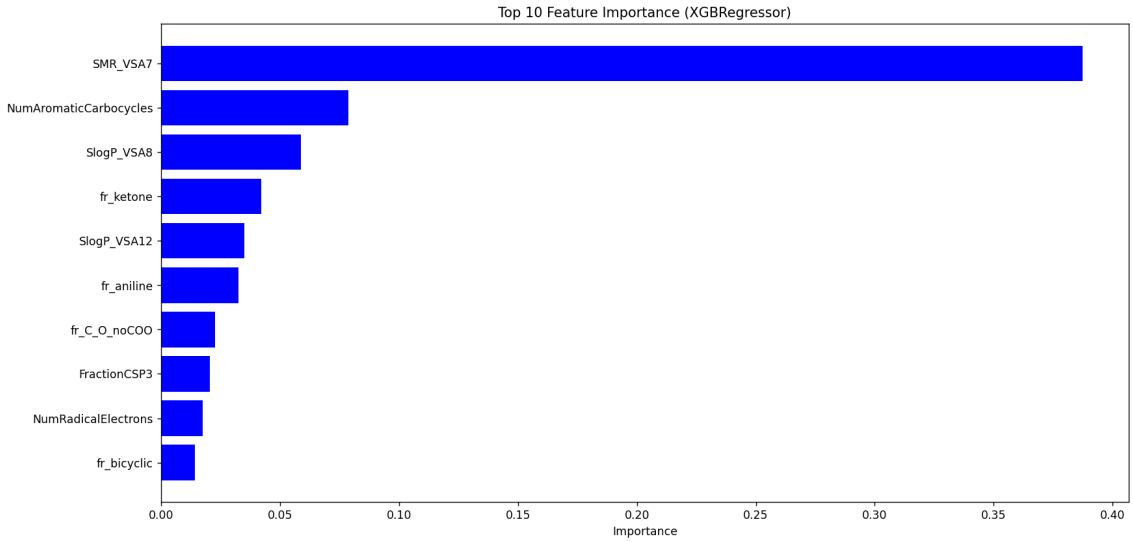
6. XGBRegressor (XGBoost)

- Theoretical Details:
  - ➢ Type: Gradient Boosting Framework
  - ➢ Algorithm: Implements gradient boosting with decision trees.
  - ➢ Working: Optimized for speed and performance using techniques like tree pruning, parallel processing, and handling sparse data.
- Strengths:
  - ➢ High performance and accuracy.
  - ➢ Regularization to prevent overfitting.
  - ➢ Flexibility to handle different types of data.
- Weaknesses:
  - ➢ Complexity in hyperparameter tuning.
  - ➢ Higher memory usage.
  - ➢ Typical Use Cases: Competitions and applications where prediction accuracy is critical.
- Analysis:
  - ➢ Performance: R-Squared (0.91) and Adjusted R-Squared (0.70).
  - ➢ RMSE: 8.79, indicating good accuracy.
  - ➢ Time Taken: 1.22 seconds, fast and efficient.

- Scatter Plot: Actual vs Predicted Values



- Feature importance plot



Summary

- The top models each have unique strengths and trade-offs:
    - ExtraTreesRegressor: Best overall performance but computationally expensive.
    - LGBMRegressor: High performance and extremely efficient.
    - HistGradientBoostingRegressor: Balanced performance with good speed.
    - RandomForestRegressor: Robust with good performance but slowest.
    - BaggingRegressor: Effective variance reduction, efficient.
    - XGBRegressor: High accuracy, efficient, but complex to tune.

Choosing the right model depends on the specific requirements of the task, including the need for speed, interpretability, and handling large datasets.

Code Snippets

Canonical SMILES:

```
def canonical_smiles(smiles):

    mols = [Chem.MolFromSmiles(smi) for smi in smiles]

    smiles = [Chem.MolToSmiles(mol) for mol in mols]

    return smiles
```

Calculating Molecular Descriptors using RDKit:

```
def RDkit_descriptors(smiles):

    mols = [Chem.MolFromSmiles(i) for i in smiles]

    calc = MoleculeDescriptors.MolecularDescriptorCalculator([x[0] for x
in Descriptors._descList])

    desc_names = calc.GetDescriptorNames()



    Mol_descriptors =[]

    for mol in mols:

        mol=Chem.AddHs(mol)

        descriptors = calc.CalcDescriptors(mol)

        Mol_descriptors.append(descriptors)

    return Mol_descriptors, desc_names
```

Calculating Molecular Descriptors using Mordred:

```
def All_Mordred_descriptors(data):

    calc = Calculator(descriptors, ignore_3D=False)

    mols = [Chem.MolFromSmiles(smi) for smi in data]



    # pandas df

    df = calc.pandas(mols)

    return df
```

Predicting with Lazy Predict:

```
from lazypredict.Supervised import LazyRegressor

from sklearn.utils import shuffle

import pandas as pd



# Load dataset

df1 = pd.read_csv('input_data.csv')

df2 = pd.read_csv("descriptors.csv")



X = df2.iloc[1:, :]  # Features

y = df1.iloc[1:, -1]  # Target variable
```

```
# Shuffle dataset
X, y = shuffle(X, y, random_state=42)


# Split data into training and testing sets
offset = int(X.shape[0] * 0.9)
X_train, y_train = X[:offset], y[:offset]
X_test, y_test = X[offset:], y[offset:]


# Initialize LazyRegressor and fit models
reg = LazyRegressor(verbose=0, ignore_warnings=False, custom_metric=None)
models, predictions = reg.fit(X_train, X_test, y_train, y_test)


print(models)
```

Conclusion

Through this project, we successfully developed a predictive model for membrane permeability using machine learning algorithms. By leveraging molecular descriptors and biological insights, our model demonstrates promising accuracy in predicting membrane permeability, thereby contributing to drug discovery and development processes.

Future Work

- Explore additional machine learning algorithms and ensemble techniques for improved predictive performance.
- Incorporate advanced molecular descriptors and biological features for enhanced model robustness.
- Validate the model on diverse datasets and conduct rigorous external validation for real-world applicability.

Acknowledgments

We acknowledge the support of our mentors and the availability of open-source datasets and libraries that facilitated our research and experimentation.

References

Kumari C, Abulaish M, Subbarao N. Exploring Molecular Descriptors and Fingerprints to Predict mTOR Kinase Inhibitors using Machine Learning Techniques. IEEE/ACM Trans Comput Biol Bioinform. 2021 Sep-Oct;18(5):1902-1913. doi: 10.1109/TCBB.2020.2964203. Epub 2021 Oct 7. PMID: 31905145.

Rezaee, Reza, and Jamiu Ekundayo. 2022. "Permeability Prediction Using Machine Learning Methods for the CO2 Injectivity of the Precipice Sandstone in Surat Basin, Australia" Energies 15, no. 6: 2053. https://doi.org/10.3390/en15062053

Tian, Jianwei & Qi, Chongchong & Sun, Yingfeng & Yaseen, Zaher & Phạm, Thai. (2021). Permeability prediction of porous media using a combination of computational fluid dynamics and hybrid machine learning methods. Engineering with Computers. 37. 10.1007/s00366-020-01012-z.