SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

Technical Answers for Real World Problems (TARP)

(SWE1901)

Virtual Screening for Drug Discovery Using Neural Networks

Jth Component (Final Review)

Submitted by

Mounica K (20MIS0005)

Steve Mathew D A(20MIS0047)

Durga Shree N (20MIS0054)

Kuruva Naveen (20MIS0245)

R Nithyashree (20MIS0312)

TCC2

Submitted to

Prof. Ramkumar T

Virtual Screening for Drug Discovery Using Neural Networks

Objectives of the Project

- To accelerate the virtual screening process of drug discovery.
- To check the feasibility of converting the biological task of virtual screening into a computational problem.
- To experiment with neural network models and study their behavior in bioactivity prediction

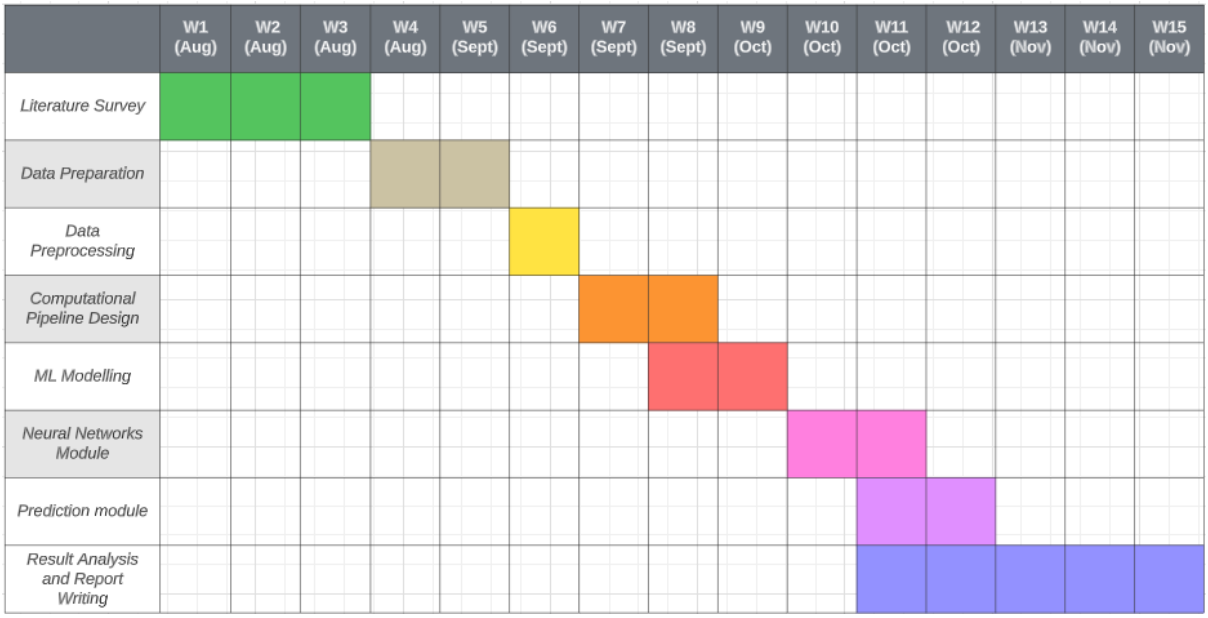
Problem statement

In the field of drug discovery, the conventional virtual screening process is time-consuming and resource-intensive. The current challenge is to expedite this process by exploring the feasibility of transforming the intricate biological task of virtual screening into a computationally efficient problem. Additionally, there is a need to investigate the application of neural network models to understand their behavior in bioactivity prediction. This study aims to address these challenges, ultimately paving the way for more efficient and effective drug discovery methods in the realm of computational biology.

Motivation in terms of technical societal/environmental/demographical perspective

Drug discovery is a very slow process spanning a minimum duration of 10-12 years before reaching clinical trials. We need to accelerate this process by simulating certain steps in drug discovery by using the potential of the advancement in deep learning algorithms. This will enhance the efficiency of the process and decrease the response time and cost.

Timeline of activities along with outcomes - Gantt Chart



Literature Review – Summary of literature studied / Gap identified

Paper	Attempt	Focus	Technology used	Outcome	Benchmark
Establishment and application of virtual screening model for anti-tuberculosis drugs based on graph neural network.	To address the urgent need for new and effective drugs to combat tuberculosis, especially in the face of drug-resistant strains.	Development and utilization of a virtual screening model for identifying potential anti-tuberculosis drug candidates.	Graph neural networks, curriculum learning, virtual screening techniques.	The successful development and validation of the GNN-MTB model for virtual screening of potential anti-tuberculosis drug candidates.	Superior predictive performance compared to other machine learning and graph neural network models

Paper	Attempt	Focus	Technology used	Outcome	Benchmark
Employing Molecular Conformations for Ligand-based Virtual Screening with Equivariant Graph Neural Network and Deep Multiple Instance Learning	To address the need to improve the accuracy of bioactivity prediction using molecular conformatio ns on Ligand-based Virtual Screening.	Ligand-based virtual screening (LBVS) is a method used in drug discovery to screen large compound databases and predict the bioactivity of molecules. Deep learning frameworks are being employed to improve LBVS by extracting intricate molecular structure representations.	EquiVS is designed using graph convolution network (GCN), equivariant graph neural network (EGNN), deep multiple instance learning layers (MIL) RDKit package is used to generate and optimize molecular confo rmers.	EquiVS outperformed others by demonstrating stability in its predictions. It had lower standard variations compared to other deep learning models indicating that its predictions were consistently relia ble	EquiVS achieved a significant relative improvement in performance comparedto one of the suboptimal baseline methods. The improvement was measured in terms of Means Squared Error, it was about 13.33%.

Bioactivity data collecting, integrating, filtering and processing workflow

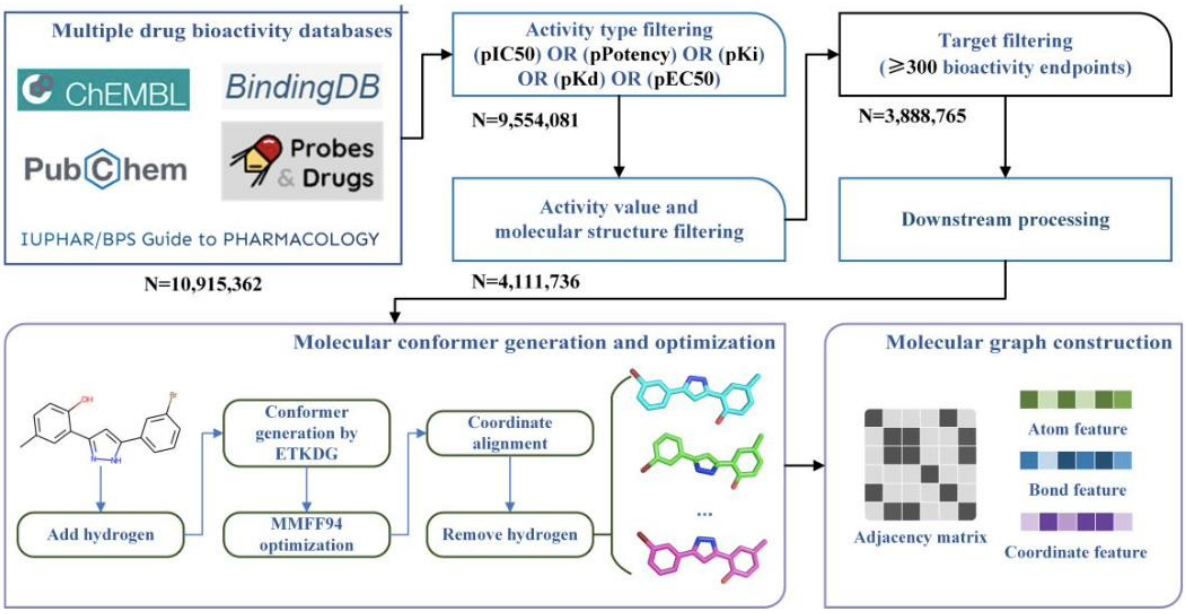
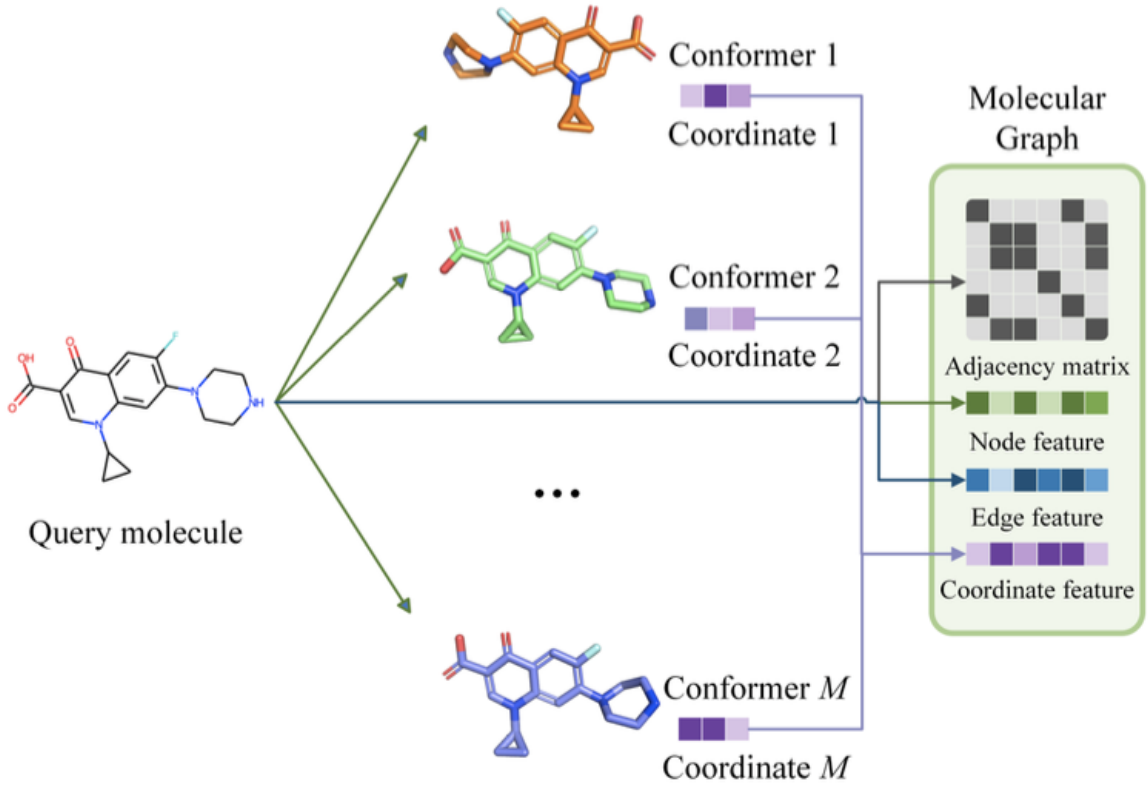
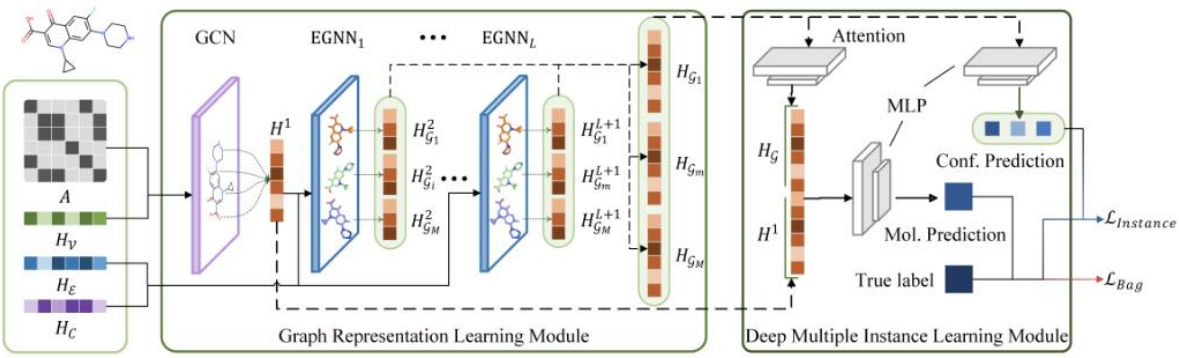


Diagram of constructing a molecular graph from the query molecule and its conformers



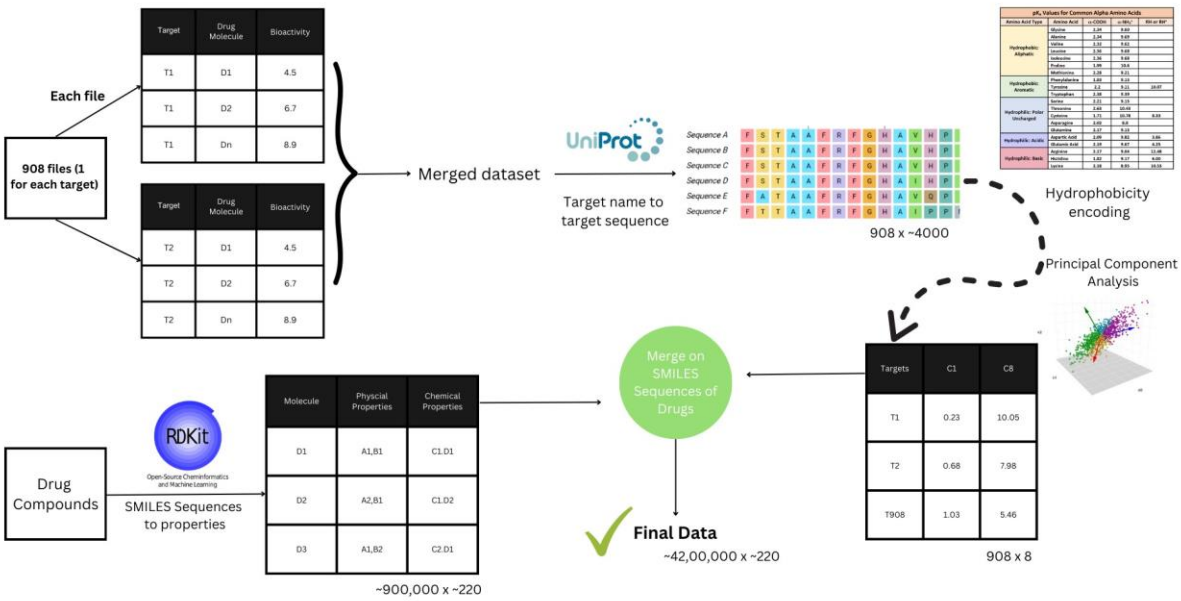
Architecture



Overcoming of limitations if any through the proposed work

- The dataset is huge with around 42 lakh rows. Performing dimensionality reduction through PCA makes the computational process effective and efficient.
- The idea of physical and chemical properties to represent compounds and encoding protein sequences using hydrophobicity to resolve the complicated learning process of proteins and drug compounds.

High-level conceptual framework of the proposed work



Brief description of the modules involved in the conceptual design

<u>Molecule-target bioactivity data</u>

ChEMBL ID	PubChem ID	IUPHAR ID	Target	Activity ty	Assay type	Unit	Activity check annotation	Ligand names	Structure check (Tanimoto)	Source	Final Activity	SMILES	Conformer_path
CHEMBL2112990			a-431	pIC50	cell-based	neg. log	only 1 data point	pd135683	1 structure	pd	5.5	CCOc1cc2ncc(C#N)c	../data/molecule_structure/279621.sdf
CHEMBL1090090			a-431	pIC50	cell-based	neg. log	only 1 data point	vx-702	1 structure	pd	4.1	NC(=O)c1ccc(N(C(N	../data/molecule_structure/34936.sdf
CHEMBL2134202			a-431	pIC50	cell-based	neg. log	only 1 data point	pf-4708671	1 structure	pd	3.9	CCc1cncnc1N1CCN(	../data/molecule_structure/931691.sdf
CHEMBL2137530			a-431	pIC50	cell-based	neg. log	only 1 data point	serdemetan	1 structure	pd	4.6	c1ccc2c(CCNc3ccc(N	../data/molecule_structure/931694.sdf
CHEMBL2138625			a-431	pIC50	cell-based	neg. log	only 1 data point	av-412	1 structure	pd	7	C=CC(=O)Nc1cc2c(N	../data/molecule_structure/84735.sdf
CHEMBL2165191			a-431	pIC50	cell-based	neg. log	only 1 data point	azd-6482	1 structure	pd	5	Cc1cc([C@@H](C)N	../data/molecule_structure/95261.sdf
CHEMBL217354			a-431	pIC50	cell-based	neg. log	only 1 data point	tw-37	1 structure	pd	5.8	CC(C)c1ccccc1Cc1cc(	../data/molecule_structure/85522.sdf
CHEMBL2178352			a-431	pIC50	cell-based	neg. log	only 1 data point	pd085873	1 structure	pd	6.9	CN1CCN(c2ccc(Nc3r	../data/molecule_structure/931729.sdf
CHEMBL2180203			a-431	pEC50	cell-based	neg. log	only 1 data point	pd134620	1 structure	pd	5	C=CC(=O)N1CCC[C@	../data/molecule_structure/279223.sdf
CHEMBL2180204			a-431	pEC50	cell-based	neg. log	only 1 data point	pd137482	1 structure	pd	4.8	C=CC(=O)N1CCC[C@	../data/molecule_structure/280700.sdf
CHEMBL1090771			a-431	pIC50	cell-based	neg. log	only 1 data point	avagacestat	1 structure	pd	3.9	NC(=O)[C@@H](CC	../data/molecule_structure/153555.sdf
CHEMBL2206431			a-431	pIC50	cell-based	neg. log	only 1 data point	arglabin	1 structure	pd	5.4	C=C1C(=O)O[C@H]2	../data/molecule_structure/931760.sdf
CHEMBL221137			a-431	pIC50	cell-based	neg. log	only 1 data point	embelin	1 structure	pd	5.2	CCCCCCCCCCCCC1=C	../data/molecule_structure/94872.sdf
CHEMBL222102			a-431	pIC50	cell-based	neg. log	only 1 data point	ku-55933	1 structure	pd	3.6	O=c1cc(-c2cccc3c2S(	../data/molecule_structure/38080.sdf
CHEMBL228043			a-431	pIC50	cell-based	neg. log	only 1 data point	lfm-a13	1 structure	pd	3.7	C/C(O)=C(\C#N)C(=	../data/molecule_structure/103136.sdf
CHEMBL230006			a-431	pIC50	cell-based	neg. log	only 1 data point	enoxolone	1 structure	pd	4.1	CC1(C)[C@@H](O)C	../data/molecule_structure/84782.sdf
CHEMBL1091644			a-431	pIC50	cell-based	neg. log	only 1 data point	linsitinib	1 structure	pd	3.9	C[C@]1(O)c[C@@H	../data/molecule_structure/418924.sdf
CHEMBL2335230			a-431	pIC50	cell-based	neg. log	only 1 data point	furvina	1 structure	pd	4.6	O=[N+]([O-])/C(Br)=	../data/molecule_structure/931836.sdf
CHEMBL2363137			a-431	pIC50	cell-based	neg. log	only 1 data point	fh535	1 structure	pd	5.7	Cc1cc([N+](=O)[O-])	../data/molecule_structure/931844.sdf
CHEMBL240954			a-431	pIC50	cell-based	neg. log	only 1 data point	sl 0101-1	1 structure	pd	3.1	CC(=O)O[C@H]1[C@	../data/molecule_structure/116424.sdf
CHEMBL2419760			a-431	pIC50	cell-based	neg. log	only 1 data point	pd084292	1 structure	pd	6.1	c1ccc(-c2cccc(Nc3nc	../data/molecule_structure/129453.sdf

1. Target dataset preprocessing
    - The dataset acquired has 908 targets.
    - Each target has been tested against relevant drug compounds and their bioactivities are recorded with each file concentrating on one target.
    - All the files are combined to produce a unified dataset containing all the targets.
    - The sequences of targets are queried from UniProt.
    - Targets are generally proteins.
    - The protein sequences are made up of 20 different amino acids.
    - Each amino acid of a sequence is placed in a different column.
    - To obtain a numerical representation, the amino acids are encoded using their corresponding hydrophobicity
    - Each protein is about 4000 amino acids long.
    - In order to reduce the dimensions, PCA is applied which results in 8 columns that capture 95% of variance.

Sequence retrieval from UniProt

Ligand names	Source	Final Activity	SMILES	Sequence
6-amino-4-(4-(dimethylamino)phenyl)-3-p-tolyl-...	chembl, pc	5.5	Cc1ccc(-c2[nH]nc3c2C(c2ccc(N(C)C)cc2)C(C#N)C(=...	MDLEGDRNGGAKKKNFFKLNNKSEKDKKEKKPTVSVFSMFRYSNWL...
benzoic acid ;,2r,4s,5r,6s,7s,9r,12r)-4,5,12...	chembl, pc	5.6	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	MDLEGDRNGGAKKKNFFKLNNKSEKDKKEKKPTVSVFSMFRYSNWL...
benzoic acid ;,2r,4s,5r,6s,7s,9r,12r)-4,7,12...	chembl, pc	5.2	CC(=O)OC[C@]12C(OC(C)=O)C(=O)C3[C@@H](OC(C)=O)...	MDLEGDRNGGAKKKNFFKLNNKSEKDKKEKKPTVSVFSMFRYSNWL...
benzoic acid (1s,2r,4s,5r,6s,7s,9r,12r)-4,12-d...	chembl, pc	4.7	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	MDLEGDRNGGAKKKNFFKLNNKSEKDKKEKKPTVSVFSMFRYSNWL...
n-cyano-n-methyl-n-(2-[[(5-methyl-1h-imidazol-...	chembl, pc, pd	4.3	CN=C(NC#N)NCCSCc1[nH]cnc1C	MDLEGDRNGGAKKKNFFKLNNKSEKDKKEKKPTVSVFSMFRYSNWL...
...	...	...	...	...
1-(1-(tert-utylcarbamoyl)piperidin-4-yl)-n-(3...	chembl	5.6	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	MPDPAAHLPFFYGSISRAEAEEHLKLAGMADGLFLLRQCLRSLGGY...
(1-(1-(tert-utylcarbamoyl)piperidin-4-yl)-n-(...	chembl	5.7	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	MPDPAAHLPFFYGSISRAEAEEHLKLAGMADGLFLLRQCLRSLGGY...
1-(1-(tert-utylcarbamoyl)piperidin-4-yl)-n-(3...	chembl	7.3	CC(C)(C)NC(=O)N1CCC(n2nc(C(=O)Nc3ccc(NC(=O)c4c...	MPDPAAHLPFFYGSISRAEAEEHLKLAGMADGLFLLRQCLRSLGGY...
1-(4-((cis)-3-(4-amino-5-(2-fluoro-5-(((s)-tet...	chembl, pc	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cc(OC[C@@H]5...	MPDPAAHLPFFYGSISRAEAEEHLKLAGMADGLFLLRQCLRSLGGY...
1-(4-((cis)-3-(4-amino-5-(2-fluoro-3-(((s)-tet...	chembl, pc	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cccc(OC[C@@H...	MPDPAAHLPFFYGSISRAEAEEHLKLAGMADGLFLLRQCLRSLGGY...

2. Drug dataset preprocessing
    - Drug compounds are represented in terms of SMILE sequences.
    - RDKit is used to obtain the physical and chemical properties of compounds.

Properties obtained from RDKit

	Unnamed: 0	Molecular Formula	Number of Atoms	MaxAbsEStateIndex	MaxEStateIndex	MinAbsEStateIndex	MinEStateIndex
0	Cc1ccc(O)c(-c2cc(-c3cccc(Br)c3)[nH]n2)c1	C16H13BrN2O	20	9.950946	9.950946	0.238789	0.238789
1	CN[C@@H]1CCc2[nH]c3ccc(C(N)=O)cc3c2C1	C14H17N3O	18	11.254750	11.254750	0.367754	-0.367754
2	O=[N+]([O-])c1cccc2[nH]nnc12	C6H4N4O2	12	10.436759	10.436759	0.026620	-0.481296
3	O=c1[nH]c(-c2ccccn2)cc2ccccc12	C14H10N2O	17	11.889521	11.889521	0.079858	-0.079858
4	O=C(Nc1ccc(Oc2ccccc2)cc1)N1CCN(c2ncnc3[nH]ncc2...	C22H21N7O2	31	12.682108	12.682108	0.118034	-0.118034
...	...	...	...	...	...	...	...
972220	C/C=C/Cn1cc(-c2cccc(C(=O)N(C)C)c2)c2cc[nH]c2c1=O	C20H21N3O2	25	12.580483	12.580483	0.046852	-0.053359
972221	CC(C)CC(NC(=O)N1CCOCC1)C(=O)N[C@@H](C=CS(=O)(=...	C28H37N3O5S	37	13.360850	13.360850	0.164360	-3.666956
972222	O=C1NC(=O)C(=Cc2cccc(C(F)(F)F)c2)S1	C11H6F3NO2S	18	12.460871	12.460871	0.082708	-4.435105
972223	O=C1NC(=O)C(=Cc2ccc3nccnc3c2)S1	C12H7N3O2S	18	11.403616	11.403616	0.340897	-0.357183
972224	COC1=CC=CC(C)C(=O)Nc2cc(O)c(OC)c(c2O)C=C(C)C[C...	C29H40N2O9	40	12.914890	12.914890	0.013579	-1.014084

972225 rows × 212 columns

3. Merging the target and drug datasets
    - Both the drug and the target datasets contain SMILE sequences.
    - Based on this common column, the drug dataset along with the physical and chemical properties of drug compounds and the protein dataset along with the encoded and reduced protein sequences are merged.

Final dataset

```
data
```

	Target	Activity type	Final Activity	SMILES	0	1	2	3	4	5	...	fr_sulfi
0	abcb1	0	5.5	Cc1ccc(-c2[nH]nc3c2C(c2ccc(N(C)C)cc2)C(C#N)C(=...	0.026644	0.055052	-0.046658	0.036779	0.004133	0.037435	...	
1	abcb1	3	5.6	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.026644	0.055052	-0.046658	0.036779	0.004133	0.037435	...	
2	abcb1	3	5.2	CC(=O)OC[C@]12C(OC(C)=O)C(=O)C3[C@@H](OC(C)=O)...	0.026644	0.055052	-0.046658	0.036779	0.004133	0.037435	...	
3	abcb1	3	4.7	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.026644	0.055052	-0.046658	0.036779	0.004133	0.037435	...	
4	abcb1	1	4.3	CN=C(NC#N)NCCSCc1[nH]cnc1C	0.026644	0.055052	-0.046658	0.036779	0.004133	0.037435	...	
...	...	...	...	...	...	...	...	...	...	...	...	
4238247	zap70	0	5.6	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.035618	0.013987	0.038120	-0.021800	-0.033059	-0.022727	...	
4238248	zap70	0	5.7	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.035618	0.013987	0.038120	-0.021800	-0.033059	-0.022727	...	
4238249	zap70	0	7.3	CC(C)(C)NC(=O)N1CCC(n2nc(C(=O)Nc3ccc(NC(=O)c4c...	0.035618	0.013987	0.038120	-0.021800	-0.033059	-0.022727	...	
4238250	zap70	1	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cc(OC[C@@H]5...	0.035618	0.013987	0.038120	-0.021800	-0.033059	-0.022727	...	
4238251	zap70	1	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cccc(OC[C@@H...	0.035618	0.013987	0.038120	-0.021800	-0.033059	-0.022727	...	

4238252 rows × 222 columns

4. Training
    - Random Forest and Neural Networks are implemented for training.
5. Prediction and Evaluation
    - Models predict the bioactivity values on unseen data.
    - The performance is evaluated using Mean Squared Error.

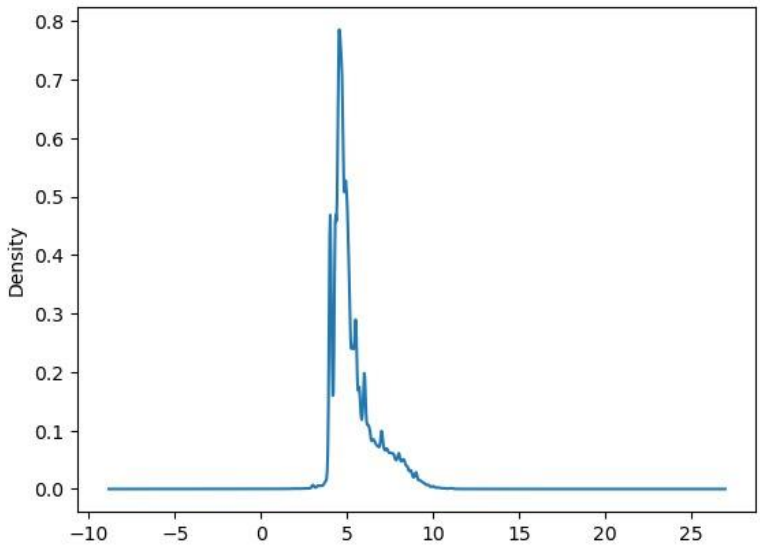
Data processing standards implemented

- Principal Component Analysis (PCA) has been used to perform dimensionality reduction.

Observations made

The distribution of bioactivity values

```
In [29]: data['Final Activity'].plot(kind='kde')
Out[29]: <Axes: ylabel='Density'>
```

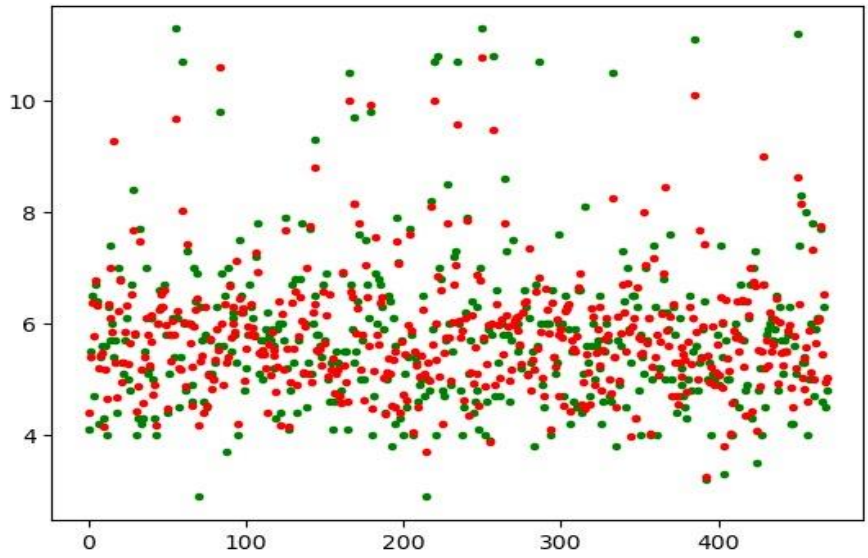


Detailed description of the work carried out in terms of innovation/novelty

- **<u>Random Forest Regressor</u>**
  - Considering a single target out of 908 to test the performance of a random forest regressor where green dots represent the actual bioactivity value whereas red dots represent the predicted bioactivity value.

```
import matplotlib.pyplot as plt


plt.plot(y_test_df['Final Activity'], 'g.', y_test_df['Predicted'], 'r.')
plt.show()
```



- Prediction values and actual values from random forest regressor

```
In [72]: y_test_df['Predicted']=y_preds
         y_test_df
```

Out[72]:

	Final Activity	Predicted
0	4.1	4.413000
1	5.5	5.420000
2	6.5	6.390000
3	4.7	5.785526
4	6.7	6.782000
...	...	...
465	7.7	7.765636
466	4.6	5.452333
467	6.3	6.531758
468	4.5	4.964000
469	4.8	5.041583

470 rows × 2 columns

- Mean Squared Error from the random forest regressor: 0.63

➢ **<u>Feed Forward Neural Network</u>**

- Prediction on one target using two hidden-layered neural network (128, 64 neurons)

```
In [24]: X_test_data=pd.DataFrame()
         y_test_data=pd.DataFrame()
         for i in unique_targets:
             print(i)
             curr = i
             curr_data = data[data['Target']==curr]
             curr_data = curr_data.drop(['SMILES','Target'],axis=1)
             curr_data=curr_data.dropna()
             X = curr_data.drop('Final Activity',axis = 1)
             y = curr_data['Final Activity']
             if len(X)!=0:
                 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,
                 X_test_data=pd.concat([X_test_data,X_test], ignore_index=True)
                 y_test_data=pd.concat([y_test_data,y_test], ignore_index=True)
                 training_models(X_train, y_train)
             else:
                 print(i,"has zero samples")
```

```
13/13 [==============================] - 0s 525us/step - loss: 1.1515
Epoch 92/100
13/13 [==============================] - 0s 2ms/step - loss: 1.1474
Epoch 93/100
13/13 [==============================] - 0s 2ms/step - loss: 1.1528
Epoch 94/100
13/13 [==============================] - 0s 2ms/step - loss: 1.1431
Epoch 95/100
13/13 [==============================] - 0s 1ms/step - loss: 1.1562
Epoch 96/100
13/13 [==============================] - 0s 1ms/step - loss: 1.1526
Epoch 97/100
13/13 [==============================] - 0s 1ms/step - loss: 1.1502
Epoch 98/100
13/13 [==============================] - 0s 2ms/step - loss: 1.1316
Epoch 99/100
13/13 [==============================] - 0s 1ms/step - loss: 1.1579
Epoch 100/100
13/13 [==============================] - 0s 1ms/step - loss: 1.1438
```

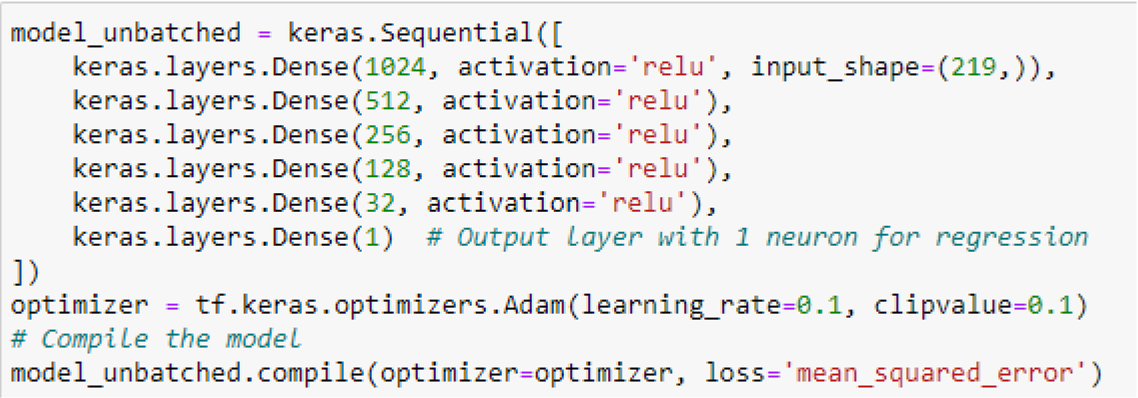
```
In [28]: loss = model.evaluate(X_test_data, y_test_data)
         print(f"Test loss: {loss:.4f}")

         # Make predictions
         y_pred = model.predict(X_test)
```

```
26021/26021 [==============================] - 24s 915us/step - loss: 1.5110
Test loss: 1.5110
4/4 [==============================] - 0s 7ms/step
```

- The Mean Squared Error on bioactivity values using a two-hidden layered feedforward neural network is 1.51

➢ Applying neural network on the entire dataset with the architecture

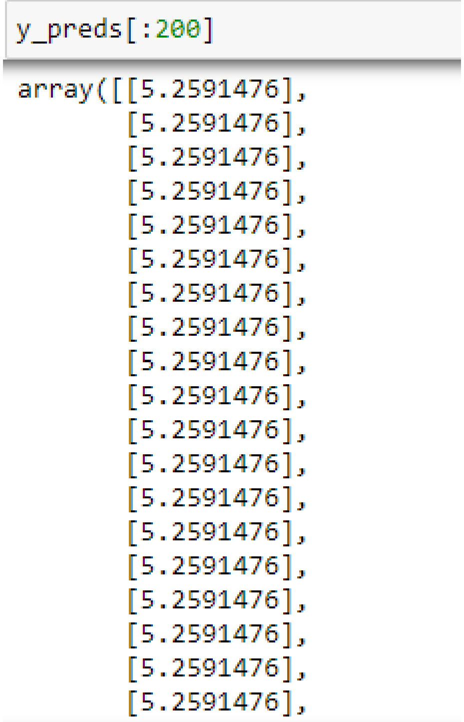


```
model_unbatched = keras.Sequential([
    keras.layers.Dense(1024, activation='relu', input_shape=(219,)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output layer with 1 neuron for regression
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.1, clipvalue=0.1)
# Compile the model
model_unbatched.compile(optimizer=optimizer, loss='mean_squared_error')
```

Two training methods employed:

  - Training the model in terms of batches where each batch contains records from a target
  - Training the model on the whole dataset

- Prediction from the new neural network architecture on the entire dataset.



```
y_preds[:200]
```

```
array([[5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
       [5.2591476],
```

- The Mean Squared Error in bioactivity values on a train-test-split containing the entire dataset: 1.4
- The model is settling at a constant value that results in minimal loss.

Verification and validation of results / Accuracy Evaluation

- The process of encoding proteins with amino acid **hydrophobicity values** and drug molecules with their **physical and chemical properties**

  – feasibility verified for bioactivity prediction

- Neural Networks are underfitting irrespective of the complexity of the model.

- Random Forest Regressor showed decent performance in bioactivity prediction on individual targets.

The outcome obtained – Paper/ Product / Solution Framework

The outcome obtained is a solution framework to explore the various possibilities of predicting the bioactivity value and hence accelerating the process of virtual screening for drug discovery. The work can be extended into a research paper in the future following the future work plan.

Future work

- To identify the biological significance – neural networks are generally converging to ~5.3 (bioactivity)

- Working with GNNs for bioactivity prediction.

  ✓ Nodes – Drugs and Targets

  ✓ Edges – Bioactivity values

  ✓ Properties of Nodes – Properties of Drugs / Targets

In [ ]:
```
import pandas as pd
import os
```

In [ ]:
```
df=[]
```

In [ ]:
```
task_list = os.listdir(path=os.path.join("C:\\Users\\admin\\Documents\\Dr S
```

In [ ]:
```
not_include=["lncap",
"mcf7",
"ht-29",
"mrc5",
"hipk",
"nci-h460",
"mda-mb-231",
"mv4-11",
"ccrf-cem",
"hl-60",
"m28b",
"hct-116",
"sw-620",
"du-145",
"k562",
"rock",
"brsk",
"m67a",
"type1",
"mskb",
"mlk",
"ck1-g",
"thp-1",
"hepg2",
"mapkapk",
"rskb"]
```

In [ ]:
```
for i in range(len(task_list)):
    if task_list[i].split(".csv")[0] not in not_include:
        d=pd.read_csv("C:\\Users\\admin\\Documents\\Dr Swarna Priya R M\\Vi
        df.append(d)
```

In [ ]:
```
merged_targets=pd.concat(df, ignore_index=True)
```

In [ ]:
```
len(merged_targets)
```

In [ ]:
```
merged_targets
```

In [ ]:
```
merged_targets = merged_targets.drop(['ChEMBL ID','PubChem ID', 'IUPHAR ID'
```

In [ ]:
```
merged_targets.to_csv("merged_targets.csv")
```

In [ ]:
```
molecules=pd.read_csv("molecules9l.csv")
```

In [ ]:
```
protein_sequences=pd.read_excel("protein_sequences.xlsx")
```

In [ ]:
```
protein_sequences=protein_sequences[["From","Sequence"]].copy()
```

In [ ]:
```
merged_targets=pd.merge(merged_targets, protein_sequences, left_on='Target'
```

In [ ]:
```
merged_targets
```

In [ ]:
```
merged_targets.to_csv("merged_targets_with_sequences.csv")
```

In [ ]:
```
molecules
```

In [ ]:
```
merged_targets=pd.merge(merged_targets, molecules, left_on='SMILES', right_
```

In [ ]:
```
merged_targets
```

In [ ]:
```
merged_targets.to_csv("master_data.csv")
```

In [ ]:
```
del merged_targets["Sequence"]
```

In [ ]:
```
protein_sequences
```

In [ ]:
```
#vocabulary
aa = ["L", "I", "N", "G", "V", "E", "P", "H", "K", "A", "Y", "W", "Q", "M",

#encoding
aa_encode = [
    0.0000,0.0000,0.0036,0.0050,0.0057,0.0058,0.0198,0.0242,0.0371,0.0373,
    0.0516,0.0548,0.0761,0.0823,0.0829,0.0829,0.0941,0.0954,0.0956,0.1263
]
```

In [ ]:
```
d={}
max=4128
for i in range(len(protein_sequences)):
    key=protein_sequences.iloc[i,0]
    value=list(protein_sequences.iloc[i,1])
    value=[aa_encode[aa.index(i)] for i in value]
    padding=4128-len(value)
    value+=[-1]*padding
    d[key]=value
```

In [ ]:
```
proteins=pd.DataFrame.from_dict(d).T
```

In [ ]:
```
proteins
```

In [ ]:
```
l=["ep"+str(i) for i in range(4128)]
```

In [ ]:
```
proteins.columns=l
```

In [ ]:
```
proteins
```

In [ ]:
```
proteins.to_csv("encoded_protein_sequences.csv")
```

In [ ]:
```
proteins_encoded=pd.read_csv("encoded_protein_sequences.csv")
```

In [ ]:
```
merged_targets
```

In [ ]:
```
merged_targets=merged_targets[:500000]
```

In [ ]:
```
merged_targets=pd.merge(merged_targets, proteins_encoded, left_on='Target',
```

In [ ]:
```
merged_targets.columns
```

In [ ]:
```
merged_targets.to_csv("master_data5l.csv")
```

In [ ]:
```
merged_targets=merged_targets.drop(["Target","Activity type", "Assay type",
```

In [ ]:
```
y = merged_targets['Final Activity'].copy()
merged_targets = merged_targets.drop('Final Activity', axis = 1)
```

In [1]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
```

In [2]:
```
import pandas as pd
```

In [3]:
```
merged_targets = pd.read_csv('master_data5l.csv')
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_14500\87676818.py:1: DtypeWarn
ing: Columns (5) have mixed types. Specify dtype option on import or set l
ow_memory=False.
  merged_targets = pd.read_csv('master_data5l.csv')
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(merged_targets,y,test_s
dt = DecisionTreeRegressor()
dt.fit(X_train,y_train)
```

In [ ]:
```
dt.score(X_test,y_test)
```

In [ ]:
```
merged_targets
```

In [ ]:

In [1]:
```
import pandas as pd
```

In [2]:
```
data=pd.read_csv("merged_targets_with_sequences.csv")
```

In [3]:
```
data
```

Out[3]:

	Unnamed: 0	Target	Activity type	Assay type	Unit	Ligand names	Source
0	0	abcb1	pEC50	functional	neg. log	6-amino-4-(4-(dimethylamino)phenyl)-3-p-tolyl-...	chembl, pc
1	1	abcb1	pKi	functional	neg. log	benzoic acid (1s,2r,4s,5r,6s,7s,9r,12r)-4,5,12...	chembl, pc
2	2	abcb1	pKi	functional	neg. log	benzoic acid (1s,2r,4s,5r,6s,7s,9r,12r)-4,7,12...	chembl, pc
3	3	abcb1	pKi	functional	neg. log	benzoic acid (1s,2r,4s,5r,6s,7s,9r,12r)-4,12-d...	chembl, pc
4	4	abcb1	pIC50	cell-based	neg. log	n-cyano-n-methyl-n-(2-{[(5-methyl-1h-imidazol-...	chembl, pc, pd
...	...	...	...	...	...	...	...
4238247	4238247	zap70	pEC50	functional	neg. log	1-(1-(tert-butylcarbamoyl)piperidin-4-yl)-n-(3...	chembl
4238248	4238248	zap70	pEC50	functional	neg. log	(1-(1-(tert-butylcarbamoyl)piperidin-4-yl)-n-(...	chembl
4238249	4238249	zap70	pEC50	functional	neg. log	1-(1-(tert-butylcarbamoyl)piperidin-4-yl)-n-(3...	chembl
4238250	4238250	zap70	pIC50	cell-free	neg. log	1-(4-((cis)-3-(4-amino-5-(2-fluoro-5-(((s)-tet...	chembl, pc
4238251	4238251	zap70	pIC50	cell-free	neg. log	1-(4-((cis)-3-(4-amino-5-(2-fluoro-3-(((s)-tet...	chembl, pc

4238252 rows × 10 columns

In [4]:
```
protein_data = pd.read_csv('PCA_proteins.csv')
```

In [5]:
```
protein_data
```

Out[5]:

	Unnamed: 0	0	1	2	3	4	5	6	
0	0	0.036710	-0.017598	0.024183	0.029517	-0.002757	0.028334	-0.027612	-0
1	1	0.036637	-0.007323	0.036436	0.017652	-0.027056	0.027144	0.010581	-0
2	2	0.036469	-0.001257	0.038694	0.006287	-0.033097	0.013308	0.032031	-0
3	3	0.035673	0.013262	0.038426	-0.020545	-0.033981	-0.020505	0.048740	0
4	4	0.036042	0.006177	0.040331	-0.006633	-0.036317	-0.002856	0.045910	-0
...	...	...	...	...	...	...	...	...	
900	900	0.031461	0.048017	-0.008968	-0.027055	0.035067	0.011414	-0.039270	-0
901	901	0.035211	0.019805	0.035095	-0.030763	-0.026404	-0.032656	0.037356	0
902	902	0.036713	-0.012780	0.031458	0.025681	-0.015911	0.032001	-0.011634	-0
903	903	0.035888	-0.028561	-0.003884	0.021955	0.031551	-0.012047	-0.018488	0
904	904	0.036563	-0.004912	0.037794	0.013551	-0.030045	0.022486	0.019562	-0

905 rows × 10 columns

In [6]:
```
data=pd.merge(data, protein_data, left_on='Target', right_on='protein', how
```

In [7]:
```
data=data.drop('Sequence',axis = 1)
```

In [8]:
```
data=data.drop(['Unnamed: 0_x','Assay type','Unit','Ligand names','Source',
```

In [9]:
```
data
```

Out[9]:

	Target	Activity type	Final Activity	SMILES	
0	abcb1	pEC50	5.5	Cc1ccc(-c2[nH]nc3c2C(c2ccc(N(C)C)cc2)C(C#N)C(=...	0.0266
1	abcb1	pKi	5.6	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.0266
2	abcb1	pKi	5.2	CC(=O)OC[C@]12C(OC(C)=O)C(=O)C3[C@@H](OC(C)=O)...	0.0266
3	abcb1	pKi	4.7	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.0266
4	abcb1	pIC50	4.3	CN=C(NC#N)NCCSCc1[nH]cnc1C	0.0266
...	...	...	...	...	
4238247	zap70	pEC50	5.6	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.0356
4238248	zap70	pEC50	5.7	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.0356
4238249	zap70	pEC50	7.3	CC(C)(C)NC(=O)N1CCC(n2nc(C(=O)Nc3ccc(NC(=O)c4c...	0.0356
4238250	zap70	pIC50	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cc(OC[C@@H]5...	0.0356
4238251	zap70	pIC50	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cccc(OC[C@@H...	0.0356

4238252 rows × 12 columns

In [10]:
```
molecules = pd.read_csv('molecules9l.csv', low_memory=False)
```

In [11]:
```
import numpy as np
```

In [12]:
```
molecules = molecules[~molecules.iloc[:,2:212].apply(lambda row: row.apply(
```

In [13]:
```
molecules=molecules.dropna()
```

In [14]:
```
molecules=molecules.drop('Molecular Formula',axis = 1)
```

In [15]:
```
data=pd.merge(data, molecules, left_on='SMILES', right_on='Unnamed: 0', how
```

In [16]:
```
data
```

Out[16]:

	Target	Activity type	Final Activity	SMILES	
0	abcb1	pEC50	5.5	Cc1ccc(-c2[nH]nc3c2C(c2ccc(N(C)C)cc2)C(C#N)C(=...	0.0266
1	abcb1	pKi	5.6	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.0266
2	abcb1	pKi	5.2	CC(=O)OC[C@]12C(OC(C)=O)C(=O)C3[C@@H](OC(C)=O)...	0.0266
3	abcb1	pKi	4.7	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.0266
4	abcb1	pIC50	4.3	CN=C(NC#N)NCCSCc1[nH]cnc1C	0.0266
...	...	...	...	...	
4238247	zap70	pEC50	5.6	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.0356
4238248	zap70	pEC50	5.7	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.0356
4238249	zap70	pEC50	7.3	CC(C)(C)NC(=O)N1CCC(n2nc(C(=O)Nc3ccc(NC(=O)c4c...	0.0356
4238250	zap70	pIC50	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cc(OC[C@@H]5...	0.0356
4238251	zap70	pIC50	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cccc(OC[C@@H...	0.0356

4238252 rows × 222 columns

In [17]:
```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['Activity type']= label_encoder.fit_transform(data['Activity type'])
```

In [18]:
```
data
```

Out[18]:

	Target	Activity type	Final Activity	SMILES	
0	abcb1	0	5.5	Cc1ccc(-c2[nH]nc3c2C(c2ccc(N(C)C)cc2)C(C#N)C(=...	0.0266
1	abcb1	3	5.6	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.0266
2	abcb1	3	5.2	CC(=O)OC[C@]12C(OC(C)=O)C(=O)C3[C@@H](OC(C)=O)...	0.0266
3	abcb1	3	4.7	CC(=O)OC[C@]12C(OC(=O)c3ccccc3)C(=O)C3[C@@H](O...	0.0266
4	abcb1	1	4.3	CN=C(NC#N)NCCSCc1[nH]cnc1C	0.0266
...	...	...	...	...	
4238247	zap70	0	5.6	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.0356
4238248	zap70	0	5.7	CCc1cc(NC(=O)c2nn(C3CCN(C(=O)NC(C)(C)C)CC3)c3c...	0.0356
4238249	zap70	0	7.3	CC(C)(C)NC(=O)N1CCC(n2nc(C(=O)Nc3ccc(NC(=O)c4c...	0.0356
4238250	zap70	1	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cc(OC[C@@H]5...	0.0356
4238251	zap70	1	5.0	CC(=O)N1CCN([C@H]2C[C@@H](n3cc(-c4cccc(OC[C@@H...	0.0356

4238252 rows × 222 columns

In [19]:
```
unique_targets = list(data['Target'].unique())
```

In [20]:
```
unique_targets
```

Out[20]:
```
['abcb1',
 'abcb11',
 'abcc1',
 'abcc2',
 'abcc3',
 'abcc4',
 'abcc9',
 'abcg2',
 'abl1',
 'abl2',
 'acaca',
 'acacb',
 'ace',
 'ache',
 'ackr3',
 'acp1',
 'acss2',
 'adam17',
 'adamts4',
```

In [21]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
```

In [25]:
```
# Create a neural network model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(219,)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(1)  # Output layer with 1 neuron for regression
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.05, clipvalue=0.1)
# Compile the model
model.compile(optimizer=optimizer, loss='mean_squared_error')
```

In [26]:
```
def training_models(X_train, y_train):
    model.fit(X_train, y_train, epochs=100, batch_size=32)
```

In [27]:
```
X_test_data=pd.DataFrame()
y_test_data=pd.DataFrame()
for i in unique_targets:
    print(i)
    curr = i
    curr_data = data[data['Target']==curr]
    curr_data = curr_data.drop(['SMILES','Target'],axis=1)
    curr_data=curr_data.dropna()
    X = curr_data.drop('Final Activity',axis = 1)
    y = curr_data['Final Activity']
    if len(X)!=0:
        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =
        X_test_data=pd.concat([X_test_data,X_test], ignore_index=True)
        y_test_data=pd.concat([y_test_data,y_test], ignore_index=True)
        training_models(X_train, y_train)
    else:
        print(i,"has zero samples")
```

```
Epoch 19/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9107
Epoch 20/100
59/59 [==============================] - 0s 1ms/step - loss: 2.0225
Epoch 21/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9351
Epoch 22/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9804
Epoch 23/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9773
Epoch 24/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9158
Epoch 25/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9910
Epoch 26/100
59/59 [==============================] - 0s 1ms/step - loss: 1.8915
Epoch 27/100
59/59 [==============================] - 0s 1ms/step - loss: 1.9471
Epoch 28/100
59/59 [==============================] - 0s 1ms/step - loss: 1.8806
```

In [28]:
```
X_test_data.to_csv('X_test_data.csv')
y_test_data.to_csv('y_test_data.csv')
```

In [29]:
```
# serialize model to JSON
model_json = model.to_json()
with open("NN_model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

In [30]:
```
model.save("NN_model_keras.keras")
```

In [31]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
```

In [32]:
```
model_unbatched_2 = keras.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(219,)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output layer with 1 neuron for regression
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.05, clipvalue=0.1)
# Compile the model
model_unbatched_2.compile(optimizer=optimizer, loss='mean_squared_error')
```

In [33]:
```
data = data.drop(['SMILES','Target'],axis=1)
data=data.dropna()
```

In [34]:
```
X = data.drop('Final Activity',axis = 1)
y = data['Final Activity']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, ra
```

In [35]:
```
model_unbatched_2.fit(X_train, y_train, epochs=100, batch_size=64)
```

```
Epoch 22/100
52018/52018 [==============================] - 392s 8ms/step - loss: 1.
4762
Epoch 23/100
52018/52018 [==============================] - 392s 8ms/step - loss: 1.
4762
Epoch 24/100
52018/52018 [==============================] - 394s 8ms/step - loss: 1.
4763
Epoch 25/100
52018/52018 [==============================] - 396s 8ms/step - loss: 1.
4763
Epoch 26/100
52018/52018 [==============================] - 403s 8ms/step - loss: 1.
4763
Epoch 27/100
52018/52018 [==============================] - 394s 8ms/step - loss: 1.
4762
Epoch 28/100
52018/52018 [==============================] - 398s 8ms/step - loss: 1.
```

In [36]:
```
# serialize model to JSON
model_json = model_unbatched_2.to_json()
with open("NN_model_unbatched_2.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model_unbatched_2.save_weights("model_unbatched_2.h5")
print("Saved model to disk")
model_unbatched_2.save("NN_model_keras_unbatched_2.keras")
```

```
Saved model to disk
```

In [22]:
```
curr = 'abcb1'

curr_data = data[data['Target']==curr]
```

In [23]:
```
curr_data = curr_data.drop(['SMILES','Target'],axis=1)
```

In [25]:
```
rf = RandomForestRegressor()
```

In [26]:
```
curr_data=curr_data.dropna()
```

In [75]:
```
curr_data.shape
```

Out[75]:
```
(2348, 220)
```

In [27]:
```
X = curr_data.drop('Final Activity',axis = 1)
y = curr_data['Final Activity']
```

In [28]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, ra
```

In [29]:
```
rf.fit(X_train, y_train)
y_preds = rf.predict(X_test)
print(r2_score(y_test,y_preds))
print(mean_squared_error(y_test,y_preds))
```

```
0.6600727967007036
0.6148368120291415
```

In [69]:
```
y_test_df=pd.read_csv('y_test.csv')
```

In [72]:
```
y_test_df['Predicted']=y_preds
y_test_df
```

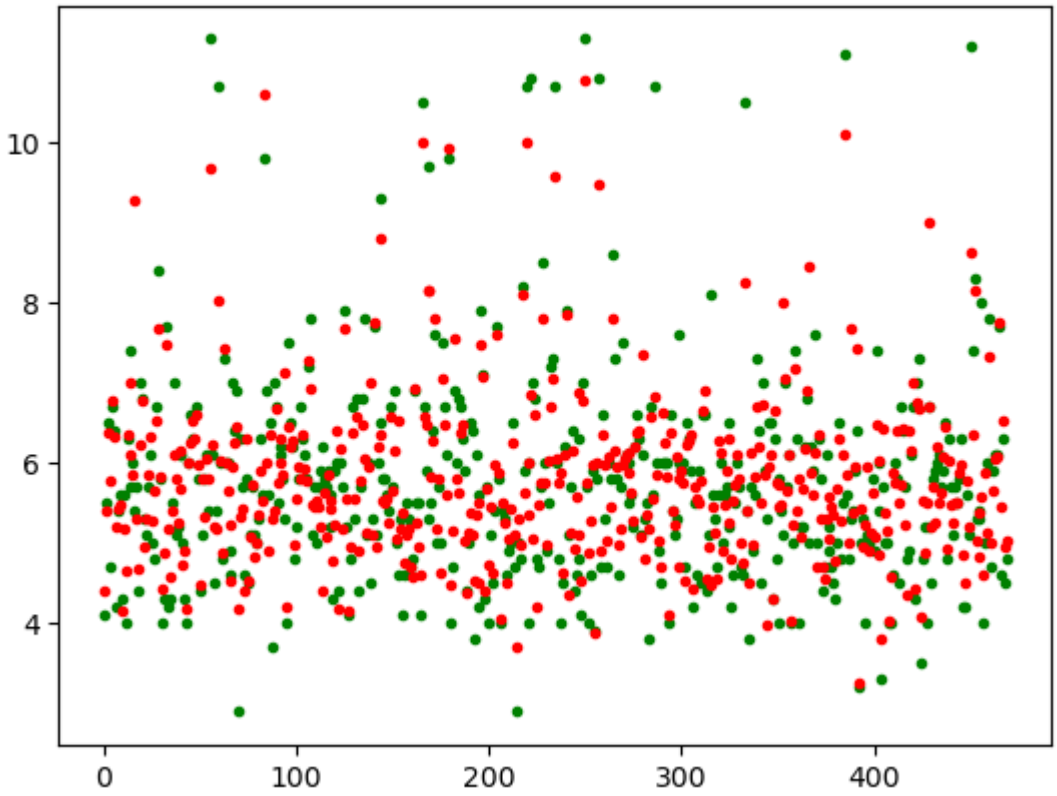
Out[72]:

	Final Activity	Predicted
0	4.1	4.413000
1	5.5	5.420000
2	6.5	6.390000
3	4.7	5.785526
4	6.7	6.782000
...	...	...
465	7.7	7.765636
466	4.6	5.452333
467	6.3	6.531758
468	4.5	4.964000
469	4.8	5.041583

470 rows × 2 columns

In [74]:
```
import matplotlib.pyplot as plt


plt.plot(y_test_df['Final Activity'], 'g.', y_test_df['Predicted'], 'r.')
plt.show()
```



In [89]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd


# Create a neural network model
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(219,)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output layer with 1 neuron for regression
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.1, clipvalue=0.1)
# Compile the model
model.compile(optimizer=optimizer, loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_t

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test loss: {loss:.4f}")

# Make predictions
y_pred = model.predict(X_test)

# Now, you can use y_pred for your regression predictions.
```

```
Epoch 1/100
59/59 [==============================] - 1s 4ms/step - loss: inf - val_
loss: 2.4718
Epoch 2/100
59/59 [==============================] - 0s 2ms/step - loss: 1.9835 - v
al_loss: 1.8269
Epoch 3/100
59/59 [==============================] - 0s 2ms/step - loss: 1.9070 - v
al_loss: 1.9367
Epoch 4/100
59/59 [==============================] - 0s 2ms/step - loss: 1.9262 - v
al_loss: 1.8109
Epoch 5/100
59/59 [==============================] - 0s 2ms/step - loss: 1.9189 - v
al_loss: 1.8090
Epoch 6/100
59/59 [==============================] - 0s 3ms/step - loss: 2.0237 - v
al_loss: 1.8183
Epoch 7/100
```

In [ ]:

In [ ]:
```
all_molecules_all_proteins=pd.read_csv("C:\\Users\\admin\\Documents\\SPRM\\
```

In [ ]:
```
all_molecules_all_proteins.head()
```

In [ ]:
```
encoded_protein_sequences=pd.read_csv("encoded_protein_sequences.csv")
```

In [ ]:
```
encoded_protein_sequences
```

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load your dataset into a Pandas DataFrame (replace 'data.csv' with your d
data = encoded_protein_sequences.drop('protein', axis=1)
data = data.T

# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Initialize PCA
pca = PCA()

# Fit PCA to the standardized data
pca.fit(data_scaled)

# Calculate the explained variance for each component
explained_variance = pca.explained_variance_ratio_

# Plot the scree plot
plt.plot(np.cumsum(explained_variance))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Scree Plot')
plt.show()

# Determine the number of components to retain
cumulative_variance_threshold = 0.95  # You can adjust this threshold as ne
num_components = np.argmax(np.cumsum(explained_variance) >= cumulative_vari

print(f'Number of components to retain for {cumulative_variance_threshold *

# Apply PCA with the chosen number of components
pca = PCA(n_components=num_components)
data_pca = pca.fit_transform(data_scaled)
```

In [ ]:
```
import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components=8)
PCA_encoded_proteins=pca.fit(data_scaled)
```

In [ ]:
```
pca_proteins_df = PCA_encoded_proteins.components_
pca_proteins_df = pd.DataFrame(pca_proteins_df,columns = data.columns)
pca_proteins_df=pca_proteins_df.T
```

In [ ]:
```
pca_proteins_df['protein'] = encoded_protein_sequences['protein']
pca_proteins_df.to_csv('PCA_proteins.csv')
```

In [ ]:
```
all_molecules_all_proteins=pd.merge(all_molecules_all_proteins, pca_protein
```

In [ ]:
```
all_molecules_all_proteins
```

In [ ]:
```
all_molecules_all_proteins=all_molecules_all_proteins.drop(['Unnamed: 0','T
```

In [ ]:
```
all_molecules_all_proteins=all_molecules_all_proteins.drop(['Molecular Form
```

In [ ]:
```
molecule_params.head()
```

In [ ]:
```
all_molecules_all_proteins
```

In [ ]:
```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
all_molecules_all_proteins['Activity type']= label_encoder.fit_transform(al

all_molecules_all_proteins['Activity type'].unique()
```

In [ ]:
```
all_molecules_all_proteins
```

In [ ]:
```
all_molecules_all_proteins = all_molecules_all_proteins[~all_molecules_all_
```

In [ ]:
```
molecule_params = all_molecules_all_proteins.iloc[:,[i for i in range(2,212
```

In [ ]:
```
molecule_params
```

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load your dataset into a Pandas DataFrame (replace 'data.csv' with your d
data1 = molecule_params
data1 = data1.T

# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled1 = scaler.fit_transform(data1)

# Initialize PCA
pca = PCA()

# Fit PCA to the standardized data
pca.fit(data_scaled1)

# Calculate the explained variance for each component
explained_variance = pca.explained_variance_ratio_

# Plot the scree plot
plt.plot(np.cumsum(explained_variance))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Scree Plot')
plt.show()

# Determine the number of components to retain
cumulative_variance_threshold = 0.95  # You can adjust this threshold as ne
num_components = np.argmax(np.cumsum(explained_variance) >= cumulative_vari

print(f'Number of components to retain for {cumulative_variance_threshold *
```

In [7]:
```
molecules_rdkit = molecules.iloc[:,2:212]
```

In [8]:
```
molecules_rdkit
```

Out[8]:

	Number of Atoms	MaxAbsEStateIndex	MaxEStateIndex	MinAbsEStateIndex	MinEStateIndex	
0	20	9.950946	9.950946	0.238789	0.238789	0
1	18	11.254750	11.254750	0.367754	-0.367754	0
2	12	10.436759	10.436759	0.026620	-0.481296	0
3	17	11.889521	11.889521	0.079858	-0.079858	0
4	31	12.682108	12.682108	0.118034	-0.118034	0
...	...	...	...	...	...	
972220	25	12.580483	12.580483	0.046852	-0.053359	0
972221	37	13.360850	13.360850	0.164360	-3.666956	0
972222	18	12.460871	12.460871	0.082708	-4.435105	0
972223	18	11.403616	11.403616	0.340897	-0.357183	0
972224	40	12.914890	12.914890	0.013579	-1.014084	0

971689 rows × 210 columns

In [ ]:
```
# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='binary_crossentropy', optimizer='rmsprop', metri
score = loaded_model.evaluate(X, Y, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

In [38]:
```
y_preds = model_unbatched_2.predict(X_test)
```

```
26009/26009 [==============================] - 38s 1ms/step
```

In [39]:
```
y_preds[:100]
```

```
Out[39]: array([[5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
               [5.3508368],
```

```
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368],
       [5.3508368]], dtype=float32)
```

Tuning

In [41]:
```
model_unbatched_2 = keras.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(219,)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output layer with 1 neuron for regression
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.05, clipvalue=0.1)
# Compile the model
model_unbatched_2.compile(optimizer=optimizer, loss='mean_squared_error')
```

Out[41]:
```
2186011    4.5
3068547    4.1
609471     4.4
1390927    5.1
3451646    5.8
          ...
2379211    4.8
3544486    4.3
2251914    4.3
2791392    4.9
2241939    4.3
Name: Final Activity, Length: 3329131, dtype: float64
```

In [ ]: