PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

Copyright 2025 | RnD Pharma Plus | www.nghiencuuthuoc.com | Zalo: +84888999311 |

🧩 Topic: Ligand-based drug design using ChEMBL

Okay, I understand. As a Pharmaceutical Research and Development Specialist, I can guide you through your ChEMBL 35 data analysis project using RDKit within the AIMLops template. I'll focus on addressing the errors you encountered and provide a structured approach with SQL and Python code. We'll aim for a subset of 100 rows to keep the process manageable.

1. Analysis of the Model and Errors

a. SQL Error: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`

- **Problem:** This error indicates a type mismatch in your SQL query. The ~ operator in PostgreSQL is used for regular expression matching, but it expects a `TEXT` type on the left-hand side. Your `act.standard_value` column is likely a `NUMERIC` type.
- **Solution:** You need to cast the `standard_value` to `TEXT` before applying the regular expression.

b. Python Error: `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`

- **Problem:** Your scikit-learn version is outdated and doesn't support the `squared=False` parameter in `mean_squared_error`.
- **Solution:**
  1. **Update scikit-learn:** The preferred solution is to upgrade your scikit-learn version using pip: `pip install -U scikit-learn`
  2. **Alternative (if update is not possible):** Calculate the Root Mean Squared Error (RMSE) manually by taking the square root of the Mean Squared Error (MSE). This is functionally equivalent to using `squared=False`.

2. AIMLops Folder Structure (Assumed)

Based on your description, I'll assume the following basic AIMLops-inspired structure:

```
Project_Root/
├── data/
│   └── chembl_data.csv   # Output from SQL query
├── notebooks/
│   └── Topic_CheMBL_35_7_1_data_exploration.ipynb
│   └── Topic_CheMBL_35_7_2_model_building.ipynb
├── src/
│   └── (Optional: Python modules/scripts)
├── models/
│   └── (Optional: Saved model files)
└── README.md
```

3. SQL and Python Code

Here's the code, addressing the error and including filtering for 100 rows.

a. SQL (to be run in pgAdmin and saved as `chembl_data.csv` in the `data` folder)

```sql
-- File: chembl_query.sql
-- This SQL query extracts ChEMBL data and saves it as a CSV file.
```

```sql
-- Adjust the WHERE clause to match your specific criteria for selecting 100 rows.
-- This is a starting point; you may need to refine the criteria based on your
research question.

SELECT
    cmp.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value
FROM
    compound_structures cs
JOIN
    activities act ON cs.molregno = act.molregno
JOIN
    molecule_dictionary cmp ON cs.molregno = cmp.molregno
WHERE
    act.standard_type = 'IC50'  -- Example: Filter for IC50 values
    AND act.standard_units = 'nM' -- Example: Filter for nM units
    AND act.pchembl_value IS NOT NULL -- Ensure pChEMBL value exists
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Corrected: Cast to TEXT
for regex
ORDER BY
    act.pchembl_value DESC
LIMIT 100; -- Limit to 100 rows
```

Explanation:

- **`CAST(act.standard_value AS TEXT)`:** This is the critical fix. It converts the `standard_value` column to a `TEXT` type before the regular expression matching.
- **Regular Expression `^[0-9\.]+$`:** This ensures that the `standard_value` contains only numbers and decimal points.
- **`LIMIT 100`:** Restricts the output to 100 rows.
- **`WHERE` Clause:** I've added example filtering criteria (IC50, nM units, pChEMBL value exists). **You need to customize this based on your specific research question.** The `WHERE` clause is the key to selecting a representative sample.
- **Filename**: Save the output from pgAdmin as `chembl_data.csv` in the `data` directory. Make sure to select the correct CSV format in pgAdmin when exporting.

b. Python (Jupyter Notebook - `Topic_CheMBL_35_7_1_data_exploration.ipynb`)

```python
# File: Topic_CheMBL_35_7_1_data_exploration.ipynb

import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt
import seaborn as sns

# Define base path
base_path = os.path.dirname(os.getcwd()) # goes to the parent directory
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')
print(f"Data path: {data_path}")

# Load the data
try:
```

```
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL query and
saved the CSV file correctly.")
    exit()

print(df.head())
print(df.info())

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value', 'pchembl_value'])
df = df[df['standard_value'] > 0] #remove zero values

# Convert SMILES to Mol objects
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol'])

# Calculate Molecular Descriptors (Example: Molecular Weight)
df['mol_wt'] = df['mol'].apply(Descriptors.MolWt)

# Basic Data Exploration
print(df.describe())

# Visualizations (Example: pChEMBL Value Distribution)
plt.figure(figsize=(8, 6))
sns.histplot(df['pchembl_value'], kde=True)
plt.title('Distribution of pChEMBL Values')
plt.xlabel('pChEMBL Value')
plt.ylabel('Frequency')
plt.show()

# More Exploratory Analysis (customize based on your research question)
# Example: Scatter plot of Molecular Weight vs. pChEMBL Value
plt.figure(figsize=(8, 6))
sns.scatterplot(x='mol_wt', y='pchembl_value', data=df)
plt.title('Molecular Weight vs. pChEMBL Value')
plt.xlabel('Molecular Weight')
plt.ylabel('pChEMBL Value')
plt.show()

# Save processed data (optional)
processed_data_path = os.path.join(base_path, 'data', 'chembl_data_processed.csv')
df.to_csv(processed_data_path, index=False)
print(f"Processed data saved to: {processed_data_path}")
```

c. Python (Jupyter Notebook - `Topic_CheMBL_35_7_2_model_building.ipynb`)

```
# File: Topic_CheMBL_35_7_2_model_building.ipynb

import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import AllChem
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

```
# Define base path
base_path = os.path.dirname(os.getcwd())
data_path = os.path.join(base_path, 'data', 'chembl_data_processed.csv')

# Load the processed data
try:
    df = pd.read_csv(data_path)
    print("Processed data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the data
exploration notebook and saved the processed data.")
    exit()

# Feature Engineering: Calculate Morgan Fingerprints (ECFP4)
def calculate_morgan_fingerprint(mol):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048) # Radius 2,
2048 bits
    except:
        return None

df['fingerprint'] = df['mol'].apply(lambda x:
calculate_morgan_fingerprint(Chem.MolFromSmiles(x)))
df = df.dropna(subset=['fingerprint']) # Remove rows where fingerprint calculation
failed

# Convert fingerprints to numpy arrays
X = np.array([list(fp) for fp in df['fingerprint']])
y = df['pchembl_value']

# Data Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Model Training (Linear Regression Example)
model = LinearRegression()
model.fit(X_train, y_train)

# Model Prediction
y_pred = model.predict(X_test)

# Model Evaluation
mse = mean_squared_error(y_test, y_pred)
#rmse = np.sqrt(mse) # Equivalent to squared=False, if you can't update scikit-learn
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
#print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")

# Further Model Evaluation and Refinement (customize based on your needs)
# - Cross-validation
# - Hyperparameter tuning
# - Feature selection
```

4. Five Examples (Illustrative Data Points - Based on Assumed ChEMBL Data)

These examples are hypothetical, based on the types of data you might find in ChEMBL. **You would replace these with *actual* rows from your `chembl_data.csv` file for real-world analysis.** I'm focusing on the `canonical_smiles`, `standard_value`, and `pchembl_value` columns, as these are most relevant to modeling.

IMPORTANT: These are just examples; the actual values in your data will vary.

Example	canonical_smiles	standard_value (nM)	pchembl_value
1	COc1ccc2[nH]c(C(=O)Nc3ccc(Cl)cc3)cc2c1	50	7.30
2	CC(=O)Nc1ccc(Oc2ccccc2)cc1	250	6.60
3	CN1CCN(c2ccc(C(=O)Nc3ccc(F)cc3)cc2)CC1	10	8.00
4	O=C(Nc1ccccc1)c2cccnc2	1000	6.00
5	Clc1ccccc1Nc2c(C)nc(N)nc2N	75	7.12

How these examples relate to the code:

- **SQL:** The SQL query would retrieve rows *like* these (but likely with different values, given the `LIMIT 100`). The `WHERE` clause determines which compounds are selected.
- **Python (Data Exploration):** In the data exploration notebook, you'd see these compounds represented in the `df` DataFrame. You'd calculate descriptors (like molecular weight) and visualize the distribution of `pchembl_value`.
- **Python (Model Building):** In the model building notebook, the SMILES strings would be converted into Morgan fingerprints. The `pchembl_value` would be the target variable (`y`), and the fingerprints would be the features (`X`). The model would learn to predict `pchembl_value` based on the fingerprint.

5. Troubleshooting and Refinement

- **File Paths:** Double-check that your file paths in the Python code are correct relative to your AIMLops folder structure. Use `os.path.join` to construct paths. Print the `data_path` variable to verify its value.
- **SQL `WHERE` Clause:** The most important part of your SQL query is the `WHERE` clause. Experiment with different filtering criteria to select a relevant and representative subset of 100 compounds.
- **Data Cleaning:** Always handle missing values (`NaN`) and invalid data in your DataFrame.
- **Feature Engineering:** Explore different molecular descriptors and fingerprints to find features that are predictive of your target variable.
- **Model Selection:** Linear Regression is a simple starting point. Consider other models like Random Forests, Support Vector Machines, or neural networks.
- **Model Evaluation:** Use appropriate metrics to evaluate your model's performance (e.g., RMSE, R-squared, MAE). Consider cross-validation to get a more robust estimate of performance.

In summary:

1. **Fix the SQL error:** Use `CAST(act.standard_value AS TEXT)` for regex matching.
2. **Address the scikit-learn error:** Upgrade scikit-learn or calculate RMSE manually.
3. **Customize the SQL query:** Define a `WHERE` clause to select a relevant subset of 100 rows.
4. **Explore your data:** Use the data exploration notebook to understand your data and calculate molecular descriptors.
5. **Build and evaluate a model:** Use the model building notebook to train a model and assess its performance.
6. **Adapt examples**: Ensure the examples you analyse are relevant to your sample extracted.

Traduction Française:

1. Analyse du modèle et des erreurs

- **Erreur SQL :** `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`
    - **Problème :** Incompatibilité de type dans votre requête SQL. L'opérateur `~` attend un type `TEXT` à gauche.
    - **Solution :** Convertissez `standard_value` en `TEXT` avec `CAST(act.standard_value AS TEXT)`.
- **Erreur Python :** `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`
    - **Problème :** Version obsolète de scikit-learn.
    - **Solution :** Mettez à jour scikit-learn (`pip install -U scikit-learn`) ou calculez la RMSE manuellement : `rmse = np.sqrt(mse)`.

2. Structure des dossiers AIMLops (supposée)

(Voir la structure ci-dessus en anglais).

3. Code SQL et Python

(Voir le code complet ci-dessus en anglais). Les points clés :

- **SQL :** Utilisez `CAST(act.standard_value AS TEXT)` dans la requête. Personnalisez la clause `WHERE` pour sélectionner les 100 lignes pertinentes. Enregistrez le résultat au format CSV.
- **Python :** Utilisez `os.path.join` pour les chemins de fichiers. Chargez les données, nettoyez-les, calculez les descripteurs RDKit et construisez un modèle.

4. Cinq Exemples

(Voir les exemples hypothétiques ci-dessus en anglais). *Remplacez-les par des données réelles de votre fichier `chembl_data.csv`*.

5. Dépannage et Amélioration

- **Chemins de fichiers :** Vérifiez que les chemins dans Python sont corrects.
- **Clause `WHERE` SQL :** C'est la clé de la sélection des données.
- **Nettoyage des données :** Gérez les valeurs manquantes.
- **Ingénierie des caractéristiques :** Explorez différents descripteurs moléculaires.
- **Sélection du modèle :** Essayez différents modèles.
- **Évaluation du modèle :** Utilisez les métriques appropriées.

N'hésitez pas à me poser d'autres questions au fur et à mesure de votre progression. Good luck!