PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Merging ChEMBL with chemical vendor data

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit, with a specific focus on the `Topic_CheMBL_35_90` topic. You're facing issues with SQL operators and scikit-learn version compatibility, and you want to follow an AIMLops-inspired folder structure. Let's break down the analysis, provide the code, and address the errors.

1. Analysis Model

The core of your analysis will likely involve:

- **Data Extraction (SQL):** Fetching relevant data from the Chembl 35 database. This will involve querying specific tables (e.g., `activities`, `molecule_dictionary`, etc.) based on your `Topic_CheMBL_35_90` topic. You'll be filtering and limiting the data to 100 rows to manage the processing load.
- **Data Preprocessing (Python/RDKit):** Cleaning, transforming, and preparing the data for analysis. This involves handling missing values, converting SMILES strings to RDKit molecules, and potentially calculating molecular descriptors.
- **Exploratory Data Analysis (Python):** Exploring the data to understand its distribution, relationships between variables, and potential insights. This will involve using libraries like Pandas, Matplotlib, and Seaborn for visualization and statistical analysis.
- **Modeling (Python):** Developing a predictive model to analyze activities. Possible models include:
    - **Regression Models:** If your activity data is continuous (e.g., IC50 values), you might use linear regression, support vector regression (SVR), or random forest regression.
    - **Classification Models:** If your activity data is categorical (e.g., active/inactive), you might use logistic regression, support vector machines (SVM), or random forest classification.
- **Evaluation (Python):** Assessing the performance of your model using appropriate metrics (e.g., R-squared, RMSE for regression; accuracy, precision, recall for classification).

2. Folder Structure (AIMLops Inspired)

Given your description, let's define the folder structure. I'll assume a simplified structure for this example. Adjust as needed.

```
project_root/
├── data/             # CSV files extracted from Chembl
├── notebooks/        # Jupyter notebooks
│   ├── Topic_CheMBL_35_90_1_data_extraction.ipynb
│   ├── Topic_CheMBL_35_90_2_data_preprocessing.ipynb
│   ├── Topic_CheMBL_35_90_3_eda.ipynb
│   ├── Topic_CheMBL_35_90_4_modeling.ipynb
│   └── Topic_CheMBL_35_90_5_evaluation.ipynb
├── sql/              # SQL scripts
│   └── Topic_CheMBL_35_90_extraction.sql
└── README.md         # Project description
```

3. Code (SQL & Python)

Here's the SQL and Python code, addressing the errors and filtering to 100 rows.

3.1 SQL Code (`sql/Topic_CheMBL_35_90_extraction.sql`)

```
-- SQL script to extract activity data from ChEMBL 35, limited to 100 rows
-- based on Topic_CheMBL_35_90

-- Adjust this WHERE clause according to your 'Topic_CheMBL_35_90' criteria
-- This is a placeholder - you MUST replace it with your actual filtering logic.
-- Example:  Targeting a specific target protein.  Replace 'CHEMBL205' with your
actual target chembl_id
-- Example targeting: CHEMBL205
SELECT
    md.chembl_id,
    act.standard_value,
    act.standard_units,
    act.standard_type
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL205' -- Filter by the target chembl_id for
TOPIC_CheMBL_35_90
    AND act.standard_relation = '='
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'
    AND act.standard_type = 'IC50'
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Corrected operator for numeric
check

LIMIT 100;
```

Explanation of SQL Correction:

- **`act.standard_value::text ~ '^[0-9\.]+$'`**: The error `ERROR: operator does not exist: numeric ~ unknown` occurs because you are trying to use the regular expression operator `~` directly on a numeric column. The corrected code casts the `standard_value` to text (`::text`) before applying the regular expression, allowing the check to work correctly. The regular expression `^[0-9\.]+$` ensures that the value contains only digits and periods.

3.2 Python Code (Jupyter Notebooks)

Here's the Python code, split across several notebooks to align with your structure.

Notebook 1: `notebooks/Topic_CheMBL_35_90_1_data_extraction.ipynb`

```
import os
import psycopg2
import pandas as pd

# Database credentials
db_ip = '192.168.206.136'
db_user = 'rd'
db_pass = 'rd'
db_name = 'chembl_35'

# Output file path
base_path = os.getcwd()  # Get current working directory, which should be
'project_root'
data_path = os.path.join(base_path, 'data', 'chembl_35_data.csv')
```

```python
sql_path = os.path.join(base_path, 'sql', 'Topic_CheMBL_35_90_extraction.sql')


# Function to execute SQL query and save to CSV
def extract_data(sql_file, csv_file, db_host, db_name, db_user, db_pass):
    try:
        # Read SQL query from file
        with open(sql_file, 'r') as f:
            sql_query = f.read()

        # Connect to PostgreSQL
        conn = psycopg2.connect(host=db_host, database=db_name, user=db_user,
password=db_pass)
        cur = conn.cursor()

        # Execute the query
        cur.execute(sql_query)

        # Fetch the results
        results = cur.fetchall()

        # Get column names
        column_names = [desc[0] for desc in cur.description]

        # Convert results to a Pandas DataFrame
        df = pd.DataFrame(results, columns=column_names)

        # Save DataFrame to CSV
        df.to_csv(csv_file, index=False)

        print(f"Data extracted and saved to {csv_file}")

    except Exception as e:
        print(f"Error: {e}")

    finally:
        if conn:
            cur.close()
            conn.close()

# Run the extraction
extract_data(sql_path, data_path, db_ip, db_name, db_user, db_pass)
```

Notebook 2: notebooks/Topic_CheMBL_35_90_2_data_preprocessing.ipynb

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

# Input CSV file path
base_path = os.getcwd()
data_path = os.path.join(base_path, 'data', 'chembl_35_data.csv')

# Load the data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the data
extraction notebook first.")
    exit()

# Data Cleaning and Transformation
def preprocess_data(df):
    # 1. Handle Missing Values (if any) - Replace with median for numeric columns
    for col in df.columns:
        if df[col].dtype in ['int64', 'float64']:
            df[col] = df[col].fillna(df[col].median())  # Or another strategy

    # 2. Convert ChEMBL ID to SMILES (This requires another query or a lookup table -
placeholder)
    # This assumes you have a way to get SMILES from ChEMBL ID.
    # For demonstration, I'll create fake SMILES.  REPLACE THIS WITH REAL LOOKUP.
    smiles_list = ['CC(=O)Oc1ccccc1C(=O)O' for _ in range(len(df))]  # Dummy SMILES
    df['smiles'] = smiles_list

    # 3. Convert SMILES to RDKit Mol objects
    df['mol'] = df['smiles'].apply(lambda x: Chem.MolFromSmiles(x))

    # 4. Remove rows with invalid molecules
    df = df[df['mol'].notna()]

    # 5. Standardize Activity Values (e.g., convert all to pIC50 if necessary)
    # Assuming you want to convert IC50 to pIC50.  This part is crucial and needs
adjustment based on your activity data
    # and topic
    df = df[df['standard_value'].notna()]
    df['pIC50'] = -np.log10(df['standard_value'].astype(float) * 1e-9)  # Convert IC50
in nM to pIC50

    return df

df = preprocess_data(df.copy()) # Work on a copy to avoid modifying the original
DataFrame

print(df.head())
print(df.dtypes)
```

Notebook 3: notebooks/Topic_CheMBL_35_90_3_eda.ipynb

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the preprocessed data
base_path = os.getcwd()
data_path = os.path.join(base_path, 'data', 'chembl_35_data.csv')

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the data
extraction notebook first.")
    exit()

# Basic EDA
print(df.describe())
```

```
# Distribution of pIC50 values
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'], kde=True)
plt.title('Distribution of pIC50 Values')
plt.xlabel('pIC50')
plt.ylabel('Frequency')
plt.show()

# Example:  Correlation between molecular weight and pIC50 (requires descriptor
calculation in preprocessing)
# from rdkit.Chem import Descriptors  # Import inside this notebook to avoid circular
dependencies
# df['mol_wt'] = df['mol'].apply(Descriptors.MolWt)  # Requires 'mol' column from
previous notebook

# plt.figure(figsize=(8, 6))
# sns.scatterplot(x='mol_wt', y='pIC50', data=df)
# plt.title('Molecular Weight vs. pIC50')
# plt.xlabel('Molecular Weight')
# plt.ylabel('pIC50')
# plt.show()


#  Add more EDA based on your specific questions and topic.
```

Notebook 4: notebooks/Topic_CheMBL_35_90_4_modeling.ipynb

```
import os
import pandas as pd
from rdkit.Chem import AllChem
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from joblib import dump

# Load the preprocessed data
base_path = os.getcwd()
data_path = os.path.join(base_path, 'data', 'chembl_35_data.csv')

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the data
extraction notebook first.")
    exit()

# Feature Generation (using RDKit)
def generate_descriptors(mol):
    fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048) #EFCF6 fingerprints
with radius 2
    return list(fp)

df['descriptors'] = df['mol'].apply(generate_descriptors)

# Prepare data for modeling
X = list(df['descriptors'])
y = df['pIC50']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```python
random_state=42)

# Train a RandomForestRegressor model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Save the model
model_path = os.path.join(base_path, 'models', 'random_forest_model.joblib')
os.makedirs(os.path.join(base_path, 'models'), exist_ok=True)  # Create the 'models'
directory if it doesn't exist
dump(model, model_path)
print(f"Model saved to {model_path}")
```

Notebook 5: `notebooks/Topic_CheMBL_35_90_5_evaluation.ipynb`

```python
import os
import pandas as pd
from joblib import load
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

# Load the preprocessed data
base_path = os.getcwd()
data_path = os.path.join(base_path, 'data', 'chembl_35_data.csv')
model_path = os.path.join(base_path, 'models', 'random_forest_model.joblib')

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the data
extraction notebook first.")
    exit()

# Load the model
try:
    model = load(model_path)
except FileNotFoundError:
    print(f"Error: Model not found at {model_path}. Make sure you ran the modeling
notebook first.")
    exit()


# Prepare data for evaluation
X = list(df['descriptors'])
y = df['pIC50']

# Split data into training and testing sets (consistent split is important for proper
evaluation)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  # Same random_state as training!


# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')


# Plot predicted vs. actual values
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel('Actual pIC50')
plt.ylabel('Predicted pIC50')
plt.title('Actual vs. Predicted pIC50 Values')
plt.show()

# Residual plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.xlabel('Predicted pIC50')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')  # Add a horizontal line at y=0
plt.show()
```

Explanation of Python Code:

- **Data Loading:** The code loads the CSV file you create from the SQL query. It uses `os.path.join` to correctly construct file paths based on your folder structure.
- **RDKit Integration:** It uses RDKit to convert SMILES strings to RDKit molecule objects (`Chem.MolFromSmiles`).
- **Descriptor Generation**: Generates Molecular descriptors using RDKit
- **Model Training & Prediction:** Trains and evaluates a Random Forest Regressor model using sklearn
- **pIC50 Calculation:** Converts IC50 values to pIC50 values. **Important:** Adapt this part to your specific activity data.
- **Error Handling:** Includes `try...except` blocks to gracefully handle potential errors like file not found.
- **File Paths:** Uses `os.path.join` to create platform-independent file paths.
- **Scikit-learn:** It utilizes scikit-learn for model building and evaluation.
- **Model Saving:** Saves the trained model using `joblib` for later use.

Addressing the `squared=False` error:

The error "old scikit-learn version does not support parameters squared=False in the mean_squared_error function" means you are using an older version of scikit-learn. The `squared` parameter was introduced in a later version.

Solution:

1. **Update scikit-learn:** The best solution is to update scikit-learn to the latest version (or at least a version that supports `squared=False`). You can do this using pip:

   ```
   pip install --upgrade scikit-learn
   ```

2. **Alternative (if you cannot update):** If updating is not possible (e.g., due to environment constraints), you can calculate the RMSE manually:

   ```
   from sklearn.metrics import mean_squared_error
   import numpy as np

   # Calculate MSE
   mse = mean_squared_error(y_test, y_pred)

   # Calculate RMSE (Root Mean Squared Error)
   rmse = np.sqrt(mse)

   print(f'Root Mean Squared Error: {rmse}')  # Print RMSE instead of MSE if you need it
   ```

4. Examples for Topic_CheMBL_35_90

Here are five examples of how you might define your `Topic_CheMBL_35_90` and the corresponding SQL adjustments:

1. **Topic:** Compounds active against a specific target protein (e.g., EGFR).

   - **SQL (Adjusted `WHERE` clause):**

     ```
     WHERE
         td.chembl_id = 'CHEMBL205' -- EGFR
         AND act.standard_relation = '='
         AND act.standard_value IS NOT NULL
         AND act.standard_units = 'nM'
         AND act.standard_type = 'IC50'
         AND act.standard_value::text ~ '^[0-9\.]+$'
     LIMIT 100;
     ```

2. **Topic:** Compounds with high binding affinity to a specific target (e.g., Ki < 100 nM for DHFR).

   - **SQL (Adjusted `WHERE` clause):**

     ```
     WHERE
         td.chembl_id = 'CHEMBL134'  -- DHFR
         AND act.standard_relation = '='
         AND act.standard_value IS NOT NULL
         AND act.standard_units = 'nM'
         AND act.standard_type = 'Ki'
         AND act.standard_value::text ~ '^[0-9\.]+$'
         AND act.standard_value <= 100
     LIMIT 100;
     ```

3. **Topic:** Compounds that inhibit a specific enzyme (e.g., Acetylcholinesterase) with high potency (IC50 < 50 nM).

   - **SQL (Adjusted `WHERE` clause):**

     ```
     WHERE
         td.chembl_id = 'CHEMBL232' -- Acetylcholinesterase
         AND act.standard_relation = '='
         AND act.standard_value IS NOT NULL
     ```

```
    AND act.standard_units = 'nM'
    AND act.standard_type = 'IC50'
    AND act.standard_value::text ~ '^[0-9\.]+$'
    AND act.standard_value <= 50
LIMIT 100;
```

4. **Topic:** Selective inhibitors of a particular kinase (e.g., VEGFR2) compared to other kinases.

   - **SQL (More complex, requires subqueries or joins to compare activities against multiple targets):**

```
-- This is a simplified example; a full implementation would need to compare activities
-- across multiple target tables and involve subqueries or more complex joins.
SELECT
    md.chembl_id,
    act.standard_value,
    act.standard_units,
    act.standard_type
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL302'  -- VEGFR2
    AND act.standard_relation = '='
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'
    AND act.standard_type = 'IC50'
    AND act.standard_value::text ~ '^[0-9\.]+$'
--  AND <Add logic to compare VEGFR2 activity to activity against other kinases>
LIMIT 100;
```

5. **Topic:** Compounds with activity against a specific disease target (e.g., Alzheimer's Disease, requires linking to disease data in ChEMBL). This is often less direct and requires careful analysis of target-disease relationships.

   - **SQL (Requires understanding of ChEMBL's target-disease relationships. This is a placeholder.):**

```
--  This requires knowledge of how ChEMBL links targets to diseases.
--  The following is a very generic placeholder and WILL NOT WORK without modification.
SELECT
    md.chembl_id,
    act.standard_value,
    act.standard_units,
    act.standard_type
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
--  JOIN <Other tables to link targets to Alzheimer's Disease>
WHERE
    -- <Filtering based on disease association - NEEDS SPECIFIC CHemBL
```

```
TABLES/FIELDS>
    act.standard_relation = '='
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'
    AND act.standard_type = 'IC50'
    AND act.standard_value::text ~ '^[0-9\.]+$'
LIMIT 100;
```

Important Notes:

- **Replace Placeholders:** You **MUST** replace the placeholder `WHERE` clauses in the SQL code with the actual logic for your `Topic_CheMBL_35_90`. The examples above are just illustrations.
- **SMILES Lookup:** The Python code assumes you have a way to get SMILES strings from ChEMBL IDs. This might involve querying the `molecule_dictionary` table or using a separate lookup table.
- **Activity Standardization:** The activity standardization (e.g., converting to pIC50) is crucial. Adapt the code based on the specific activity types (IC50, Ki, EC50, etc.) and units in your data.
- **RDKit Installation:** Ensure you have RDKit installed: `conda install -c conda-forge rdkit` or `pip install rdkit`.
- **Error Handling:** The code includes basic error handling. Add more robust error handling as needed.
- **Dependencies:** Make sure you have all the necessary Python libraries installed: `pandas`, `psycopg2`, `rdkit`, `scikit-learn`, `matplotlib`, `seaborn`, `joblib`.
- **Model Selection**: Adjust your model based on your specific data and objectives. Consider other regression or classification models. Experiment with different hyperparameters.
- **Feature Engineering**: Create new relevant features for your model.

In Summary

This provides a comprehensive guide to your project, addressing the errors, organizing the code, and providing examples. Remember to adapt the code to your specific `Topic_CheMBL_35_90` and data characteristics. Good luck!

Traducción al Español (Spanish Translation)

1. Análisis del Modelo (Spanish: Análisis del Modelo)

El núcleo de tu análisis probablemente involucrará:

- **Extracción de Datos (SQL):** Obtener datos relevantes de la base de datos Chembl 35. Esto implicará consultar tablas específicas (ej., `activities`, `molecule_dictionary`, etc.) basadas en tu tema `Topic_CheMBL_35_90`. Estarás filtrando y limitando los datos a 100 filas para manejar la carga de procesamiento.
- **Preprocesamiento de Datos (Python/RDKit):** Limpiar, transformar y preparar los datos para el análisis. Esto implica manejar valores faltantes, convertir cadenas SMILES a moléculas RDKit y potencialmente calcular descriptores moleculares.
- **Análisis Exploratorio de Datos (Python):** Explorar los datos para comprender su distribución, las relaciones entre las variables y las posibles ideas. Esto implicará el uso de bibliotecas como Pandas, Matplotlib y Seaborn para la visualización y el análisis estadístico.
- **Modelado (Python):** Desarrollar un modelo predictivo para analizar las actividades. Los modelos posibles incluyen:
    - **Modelos de Regresión:** Si tus datos de actividad son continuos (ej., valores IC50), podrías usar regresión lineal, regresión de vector de soporte (SVR) o regresión de bosque aleatorio.

    - **Modelos de Clasificación:** Si tus datos de actividad son categóricos (ej., activo/inactivo), podrías usar regresión logística, máquinas de vector de soporte (SVM) o clasificación de bosque aleatorio.
- **Evaluación (Python):** Evaluar el rendimiento de tu modelo utilizando métricas apropiadas (ej., R-cuadrado, RMSE para regresión; precisión, exactitud, recall para clasificación).

2. Estructura de Carpetas (Inspirada en AIMLops) (Spanish: Estructura de Carpetas)

Dada tu descripción, definamos la estructura de carpetas. Asumiré una estructura simplificada para este ejemplo. Ajusta según sea necesario. (See the folder structure above in English)

3. Código (SQL & Python) (Spanish: Código)

Aquí está el código SQL y Python, abordando los errores y filtrando a 100 filas. (See the code above in English, paying attention to the SQL correction and the Python explanations).

Explicación de la Corrección de SQL (Spanish: Explicación de la Corrección de SQL):

El error `ERROR: operator does not exist: numeric ~ unknown` ocurre porque estás intentando usar el operador de expresión regular ~ directamente en una columna numérica. El código corregido convierte el `standard_value` a texto (`::text`) antes de aplicar la expresión regular, permitiendo que la comprobación funcione correctamente. La expresión regular `^[0-9\.]+$` asegura que el valor contenga sólo dígitos y puntos.

Abordando el error `squared=False` (Spanish: Abordando el error `squared=False`):

El error "old scikit-learn version does not support parameters squared=False in the mean_squared_error function" significa que estás utilizando una versión anterior de scikit-learn. El parámetro `squared` se introdujo en una versión posterior. (See the solution above in English)

4. Ejemplos para Topic_CheMBL_35_90 (Spanish: Ejemplos para Topic_CheMBL_35_90)

Aquí hay cinco ejemplos de cómo podrías definir tu `Topic_CheMBL_35_90` y los ajustes SQL correspondientes: (See the examples above in English, remember to translate the descriptions to Spanish if needed for clarity in your documentation).

Notas Importantes (Spanish: Notas Importantes):

- **Reemplaza los Marcadores de Posición:** Debes **REEMPLAZAR** las cláusulas `WHERE` de marcador de posición en el código SQL con la lógica real para tu `Topic_CheMBL_35_90`. Los ejemplos anteriores son sólo ilustraciones.
- **Búsqueda de SMILES:** El código Python asume que tienes una forma de obtener cadenas SMILES de los ID de ChEMBL. Esto podría implicar consultar la tabla `molecule_dictionary` o usar una tabla de búsqueda separada.
- **Estandarización de la Actividad:** La estandarización de la actividad (ej., conversión a pIC50) es crucial. Adapta el código en función de los tipos de actividad específicos (IC50, Ki, EC50, etc.) y las unidades en tus datos.
- **Instalación de RDKit:** Asegúrate de tener RDKit instalado: `conda install -c conda-forge rdkit` o `pip install rdkit`.
- **Manejo de Errores:** El código incluye un manejo de errores básico. Añade un manejo de errores más robusto según sea necesario.
- **Dependencias:** Asegúrate de tener todas las bibliotecas de Python necesarias instaladas: `pandas`, `psycopg2`, `rdkit`, `scikit-learn`, `matplotlib`, `seaborn`, `joblib`.
- **Selección de Modelo**: Ajusta tu modelo según tus datos y objetivos específicos. Considera otros modelos de regresión o clasificación. Experimenta con diferentes hiperparámetros.
- **Ingeniería de Características**: Crea nuevas características relevantes para tu modelo.

En Resumen (Spanish: En Resumen)

Esto proporciona una guía completa para tu proyecto, abordando los errores, organizando el código y proporcionando ejemplos. Recuerda adaptar el código a tu `Topic_CheMBL_35_90` específico y a las características de los datos. ¡Buena suerte!