In [3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sco
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.impute import SimpleImputer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import joblib
import kagglehub
```

In [4]:
```
# Download dataset
path = kagglehub.dataset_download("sohier/calcofi")
df = pd.read_csv(f"{path}/bottle.csv")
```

```
<ipython-input-4-18d9a3d4457d>:3: DtypeWarning: Columns (47,73) have mixed types. Specify
dtype option on import or set low_memory=False.
  df = pd.read_csv(f"{path}/bottle.csv")
```

In [5]:
```
# Data Exploration and Understanding
print("Initial dataset shape:", df.shape)
print("Columns:", df.columns)
```

```
Initial dataset shape: (864863, 74)
Columns: Index(['Cst_Cnt', 'Btl_Cnt', 'Sta_ID', 'Depth_ID', 'Depthm', 'T_degC',
       'Salnty', 'O2ml_L', 'STheta', 'O2Sat', 'Oxy_µmol/Kg', 'BtlNum',
       'RecInd', 'T_prec', 'T_qual', 'S_prec', 'S_qual', 'P_qual', 'O_qual',
       'SThtaq', 'O2Satq', 'ChlorA', 'Chlqua', 'Phaeop', 'Phaqua', 'PO4uM',
       'PO4q', 'SiO3uM', 'SiO3qu', 'NO2uM', 'NO2q', 'NO3uM', 'NO3q', 'NH3uM',
       'NH3q', 'C14As1', 'C14A1p', 'C14A1q', 'C14As2', 'C14A2p', 'C14A2q',
       'DarkAs', 'DarkAp', 'DarkAq', 'MeanAs', 'MeanAp', 'MeanAq', 'IncTim',
       'LightP', 'R_Depth', 'R_TEMP', 'R_POTEMP', 'R_SALINITY', 'R_SIGMA',
       'R_SVA', 'R_DYNHT', 'R_O2', 'R_O2Sat', 'R_SIO3', 'R_PO4', 'R_NO3',
       'R_NO2', 'R_NH4', 'R_CHLA', 'R_PHAEO', 'R_PRES', 'R_SAMP', 'DIC1',
       'DIC2', 'TA1', 'TA2', 'pH2', 'pH1', 'DIC Quality Comment'],
      dtype='object')
```

In [6]:
```
# Limit to a subset of the data
df = df.iloc[:100000]
```

In [7]:
```
print(df.describe())  # Summary Statistics
print(df.isnull().sum())  # Missing Values Check
```

```
             Cst_Cnt        Btl_Cnt         Depthm        T_degC  \
count  100000.000000  100000.000000  100000.000000  97895.000000
mean     1651.227600   50000.500000     369.328580      9.098469
std       968.719887   28867.657797     401.557767      4.482319
min         1.000000       1.000000       0.000000      1.540000
25%       811.000000   25000.750000      73.000000      5.310000
50%      1616.000000   50000.500000     200.000000      8.330000
75%      2508.000000   75000.250000     600.000000     12.010000
```

```
max       3383.000000  100000.000000    3762.000000     28.540000

              Salnty         O2ml_L         STheta          O2Sat    Oxy_µmol/Kg  \
count  74360.000000  86693.000000  73611.000000  67049.000000  67049.000000
mean      33.883527      2.732315     26.116003     45.908333    122.053198
std        0.531202      2.117817      1.096576     37.187560     92.870150
min       30.250000      0.040000     21.713000      0.700000      2.174110
25%       33.510000      0.610000     25.188000      8.800000     27.396530
50%       33.997500      2.250000     26.438000     35.400000    102.687300
75%       34.340000      5.040000     27.061000     87.400000    222.970600
max       35.280000      7.680000     28.083000    132.200000    334.977800

       BtlNum  ...  R_CHLA  R_PHAEO         R_PRES  R_SAMP  DIC1  DIC2  TA1  \
count     0.0  ...     0.0      0.0  100000.000000     0.0   0.0   0.0  0.0
mean      NaN  ...     NaN      NaN     372.258680     NaN   NaN   NaN  NaN
std       NaN  ...     NaN      NaN     405.587004     NaN   NaN   NaN  NaN
min       NaN  ...     NaN      NaN       0.000000     NaN   NaN   NaN  NaN
25%       NaN  ...     NaN      NaN      74.000000     NaN   NaN   NaN  NaN
50%       NaN  ...     NaN      NaN     201.000000     NaN   NaN   NaN  NaN
75%       NaN  ...     NaN      NaN     604.000000     NaN   NaN   NaN  NaN
max       NaN  ...     NaN      NaN    3818.000000     NaN   NaN   NaN  NaN

       TA2  pH2  pH1
count  0.0  0.0  0.0
mean   NaN  NaN  NaN
std    NaN  NaN  NaN
min    NaN  NaN  NaN
25%    NaN  NaN  NaN
50%    NaN  NaN  NaN
75%    NaN  NaN  NaN
max    NaN  NaN  NaN

[8 rows x 70 columns]
Cst_Cnt                     0
Btl_Cnt                     0
Sta_ID                      0
Depth_ID                    0
Depthm                      0
                        ...
TA1                    100000
TA2                    100000
pH2                    100000
pH1                    100000
DIC Quality Comment    100000
Length: 74, dtype: int64
```

In [8]:

```
# Check how many rows are missing in critical columns
print("Missing values in critical columns before cleaning:")
print(df[['T_degC', 'Depthm', 'Salnty', 'O2ml_L']].isnull().sum())
```

```
Missing values in critical columns before cleaning:
T_degC     2105
Depthm        0
Salnty    25640
O2ml_L    13307
dtype: int64
```

In [9]:

```
#Handle missing values by filling missing values in critical columns with t
df[['T_degC', 'Depthm', 'Salnty', 'O2ml_L']] = df[['T_degC', 'Depthm', 'Sal
```

In [10]:

```
# Check if we still have any data after cleaning
print("Dataset shape after cleaning:", df.shape)
```

```
Dataset shape after cleaning: (100000, 74)
```

In [11]:

```
# Remove non-numeric columns and duplicates
df_cleaned = df.select_dtypes(include=[np.number])
df_cleaned = df_cleaned.drop_duplicates()
```

```python
# Remove columns with all missing values before imputation
df_cleaned = df_cleaned.dropna(axis=1, how='all')
```

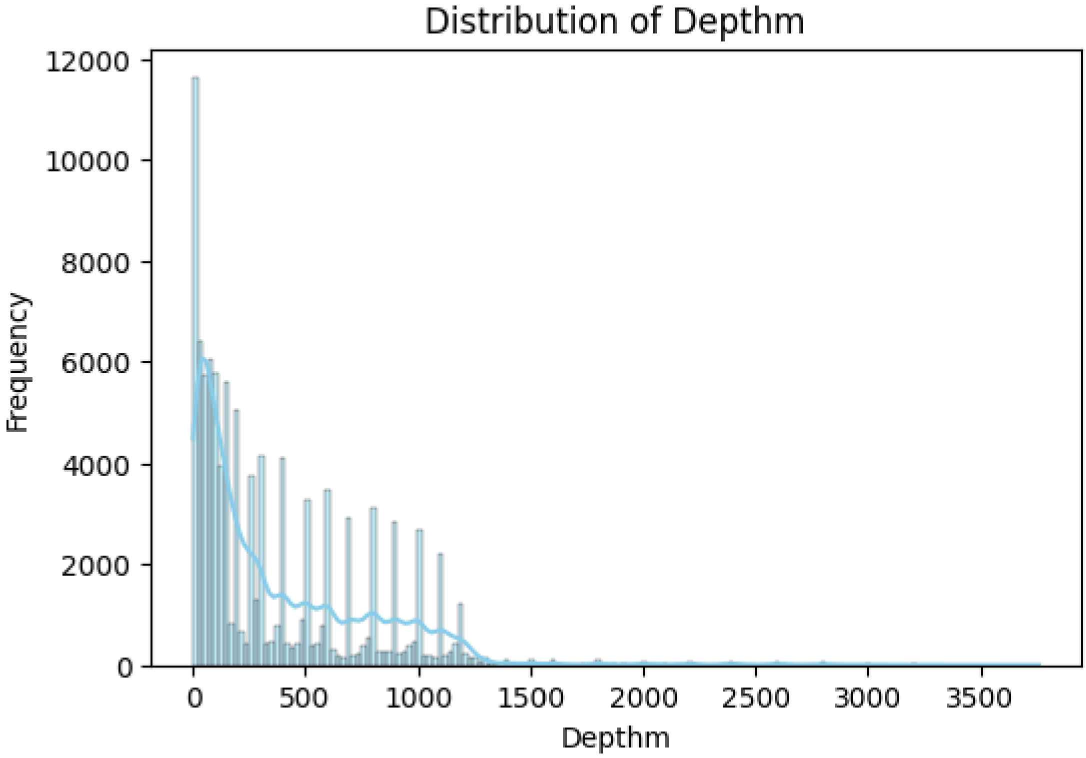
In [12]:

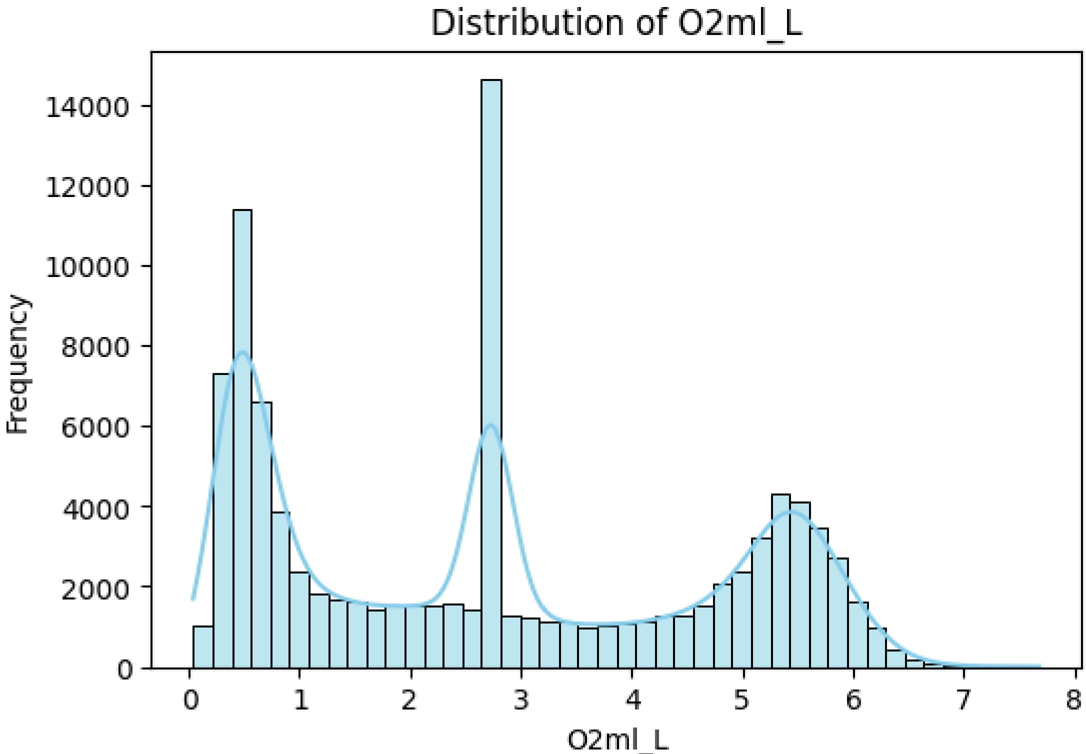
```python
# Feature Selection
target = 'T_degC'
X = df_cleaned.drop(columns=[target])
y = df_cleaned[target]

# Impute missing values in features with the mean of each column
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Check if NaN values are handled after imputation
print(f"Missing values after imputation: {np.isnan(X_imputed).sum()}")  # S

# Select top 10 features using SelectKBest
selector = SelectKBest(score_func=f_regression, k=10)
X_selected = selector.fit_transform(X_imputed, y)

# Get selected feature names from the original dataframe columns
selected_features = np.array(X.columns)[selector.get_support()]
print("Selected Features:\n", selected_features)
```

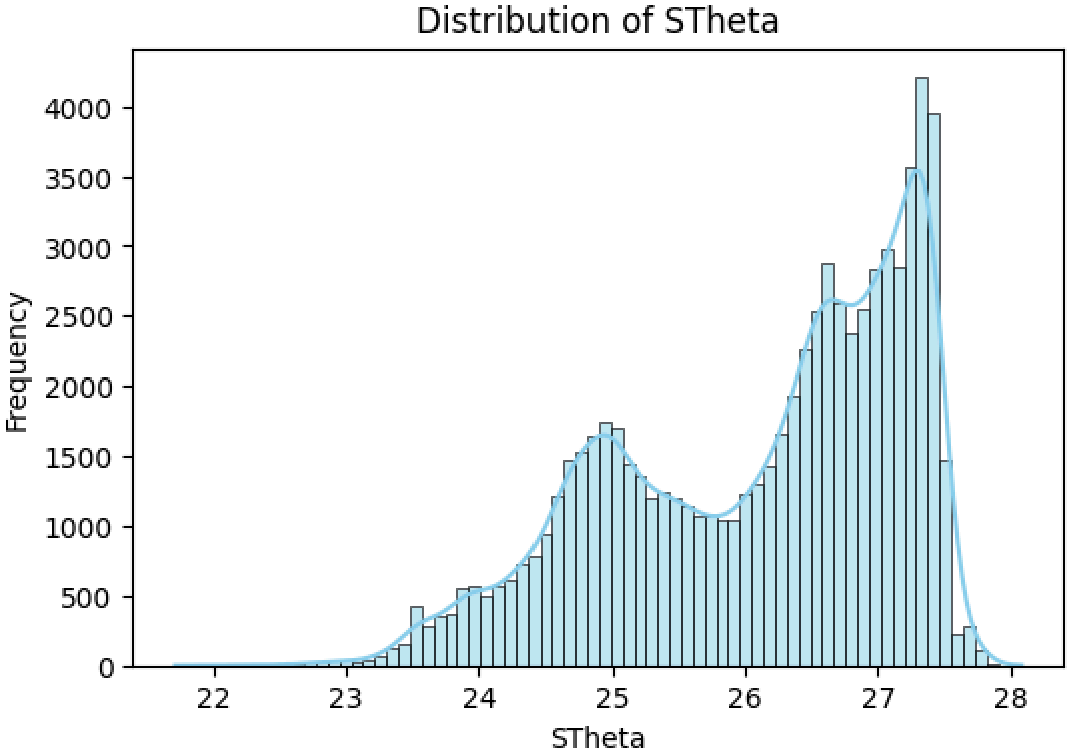
```
Missing values after imputation: 0
Selected Features:
 ['Depthm' 'O2ml_L' 'STheta' 'R_Depth' 'R_POTEMP' 'R_SIGMA' 'R_SVA'
 'R_DYNHT' 'R_O2' 'R_PRES']
```

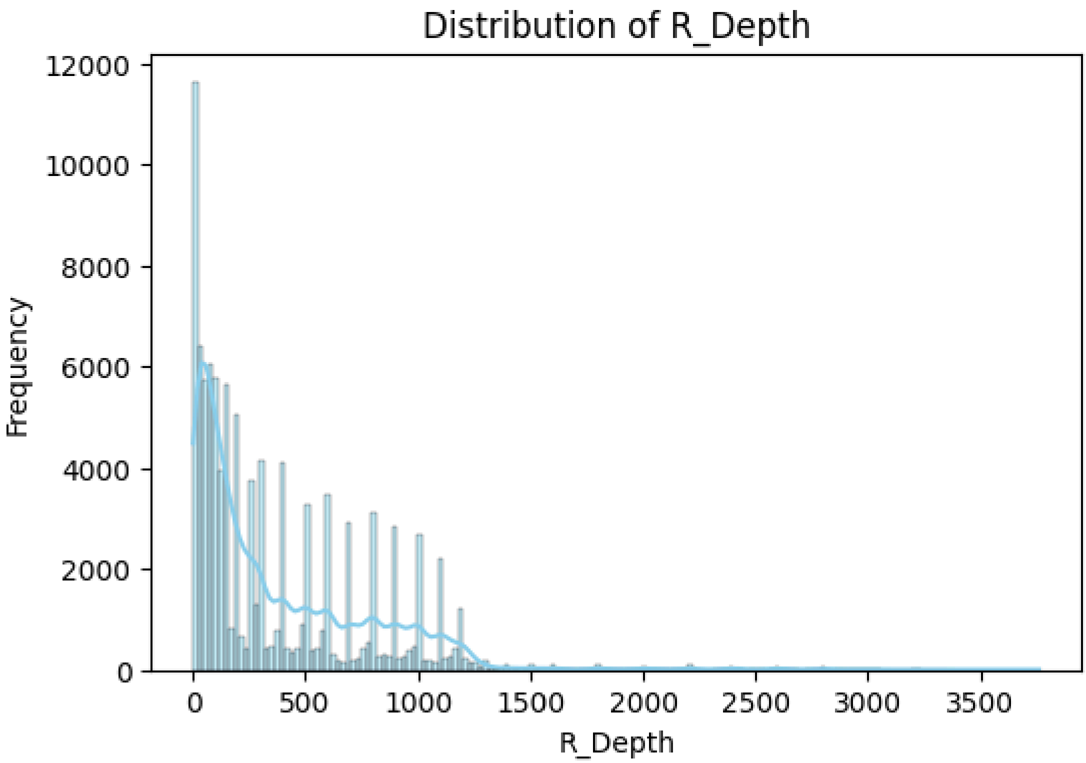
In [14]:

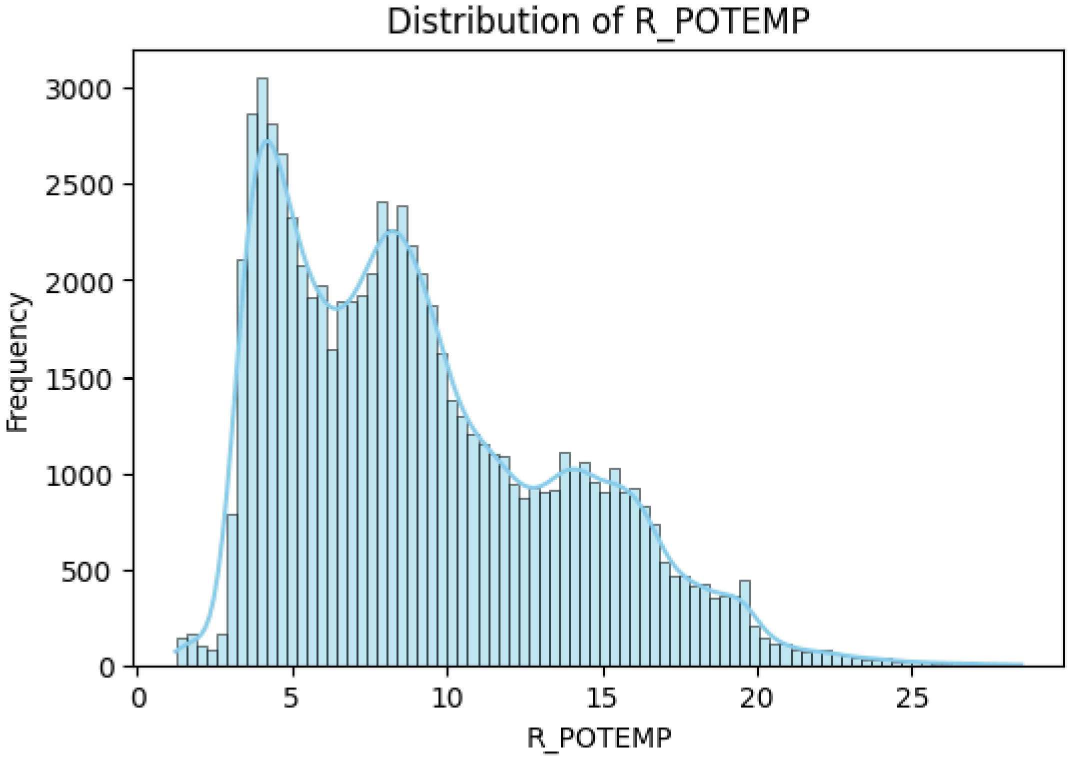
```python
# 1. Distribution Plots for Selected Features
import matplotlib.pyplot as plt
import seaborn as sns

for feature in selected_features:
    plt.figure(figsize=(6,4))
    sns.histplot(df[feature], kde=True, color='skyblue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()
```

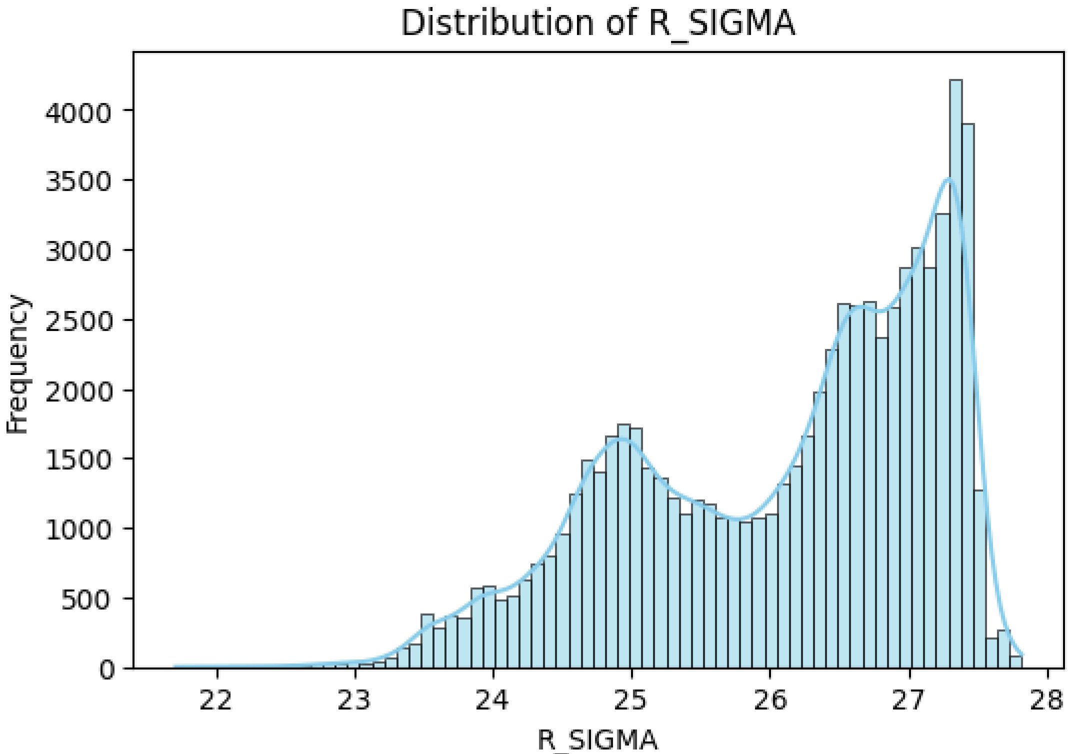

Distribution of Depthm
Frequency
Depthm
0
2000
4000
6000
8000
10000
12000
0
500
1000
1500
2000
2500
3000
3500

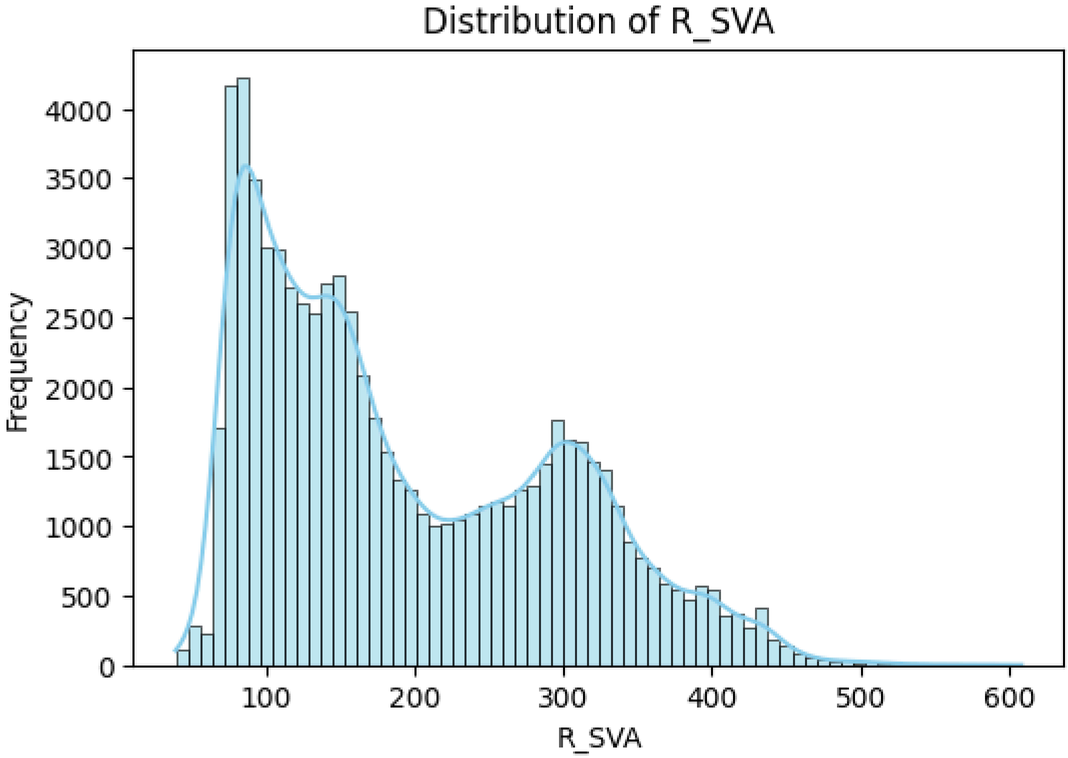

Distribution of O2ml_L
Frequency
O2ml_L
0
2000
4000
6000
8000
10000
12000
14000
0
1
2
3
4
5
6
7
8


Distribution of STheta
Frequency
STheta
0
500
1000
1500
2000
2500
3000
3500
4000
22
23
24
25
26
27
28

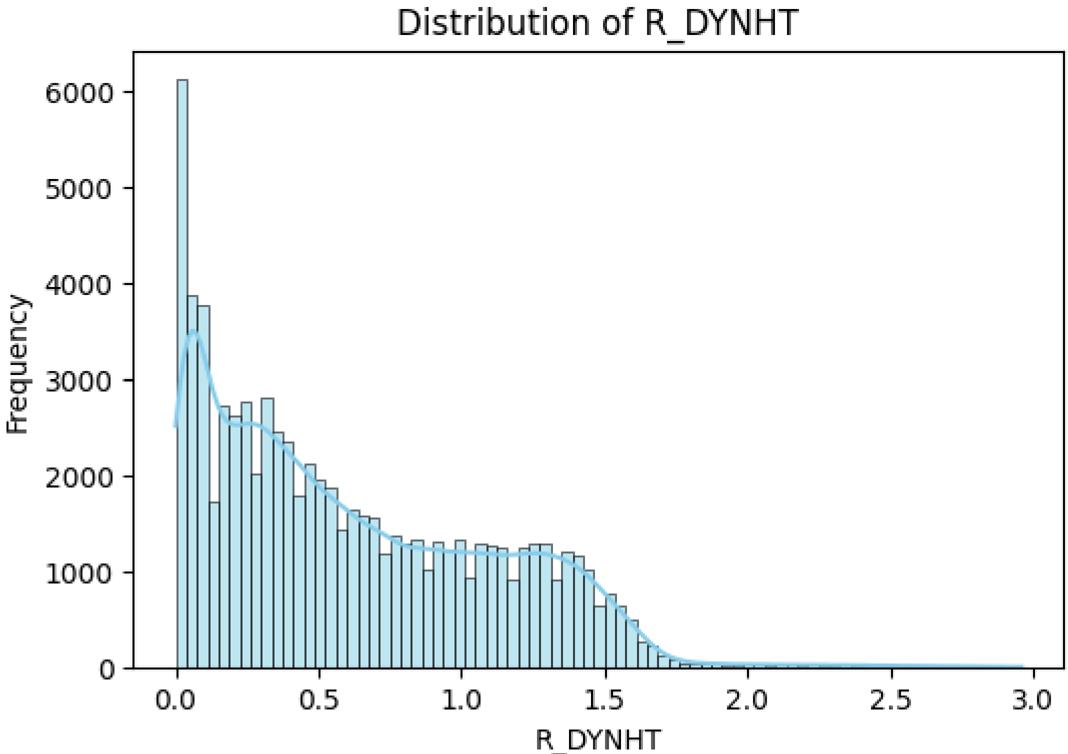

Distribution of R_Depth
Frequency
R_Depth
0
2000
4000
6000
8000
10000
12000
0
500
1000
1500
2000
2500
3000
3500


Distribution of R_POTEMP
Frequency
R_POTEMP
0
500
1000
1500
2000
2500
3000
0
5
10
15
20
25

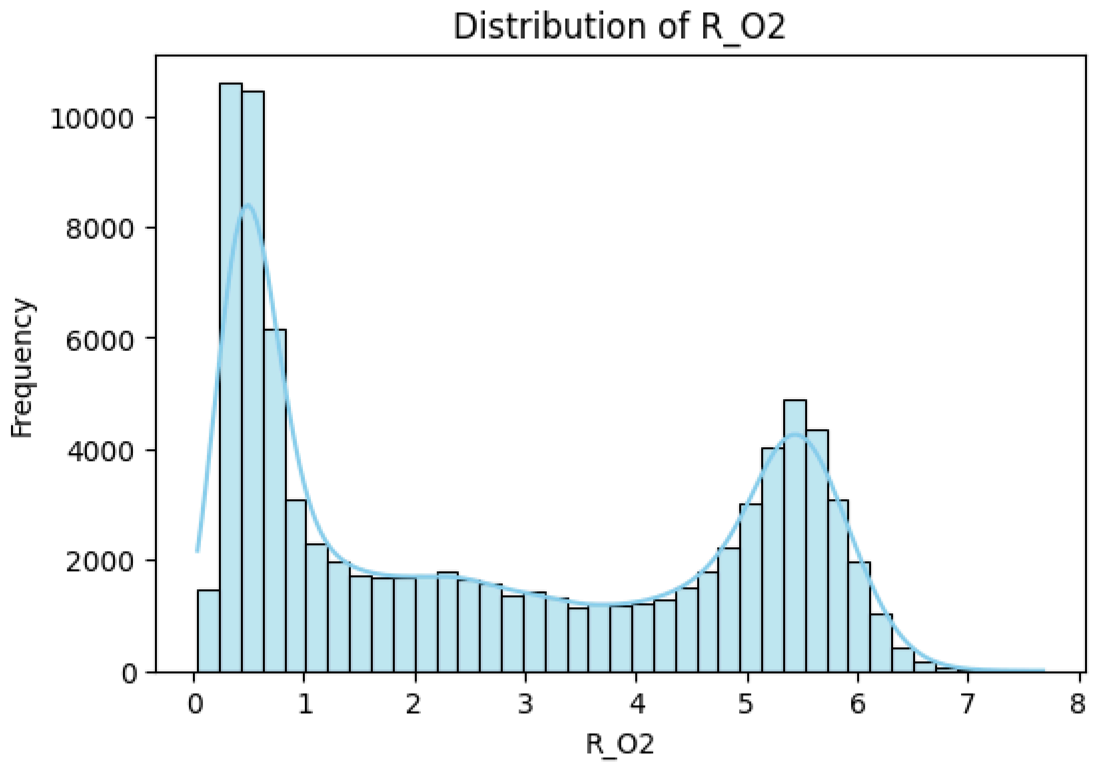

Distribution of R_SIGMA
Frequency
R_SIGMA
0
500
1000
1500
2000
2500
3000
3500
4000
22
23
24
25
26
27
28

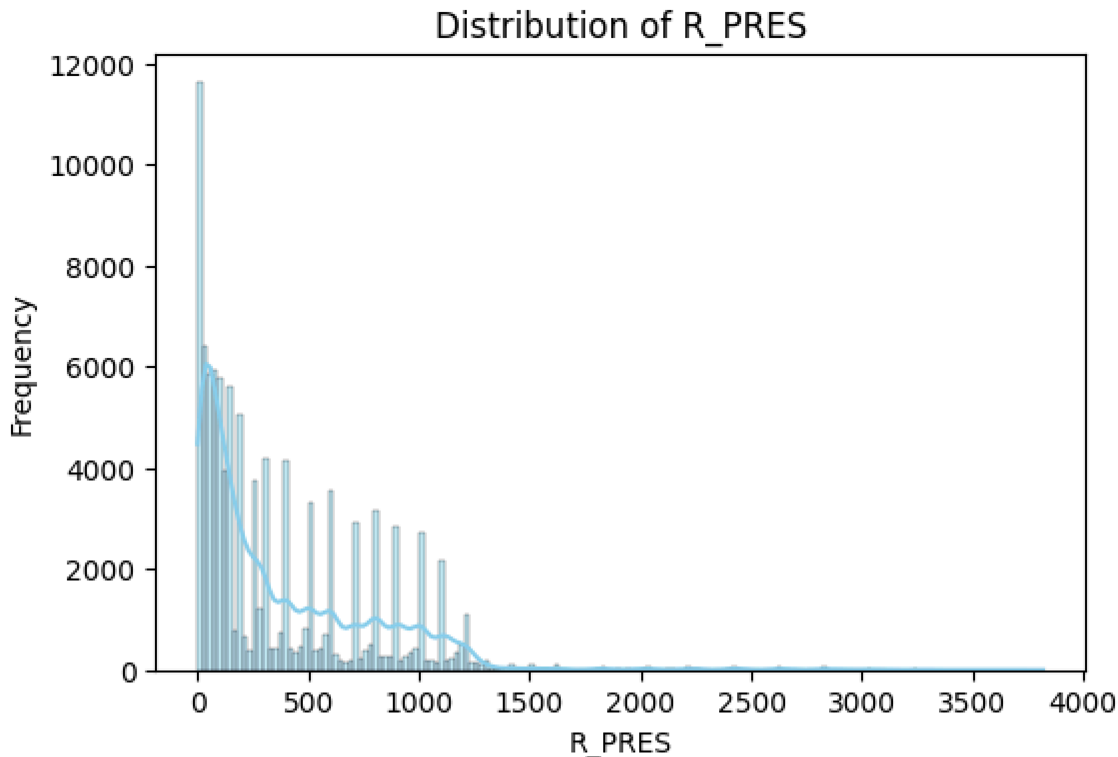


Distribution of R_SVA
Frequency
0
500
1000
1500
2000
2500
3000
3500
4000
100
200
300
400
500
600
R_SVA

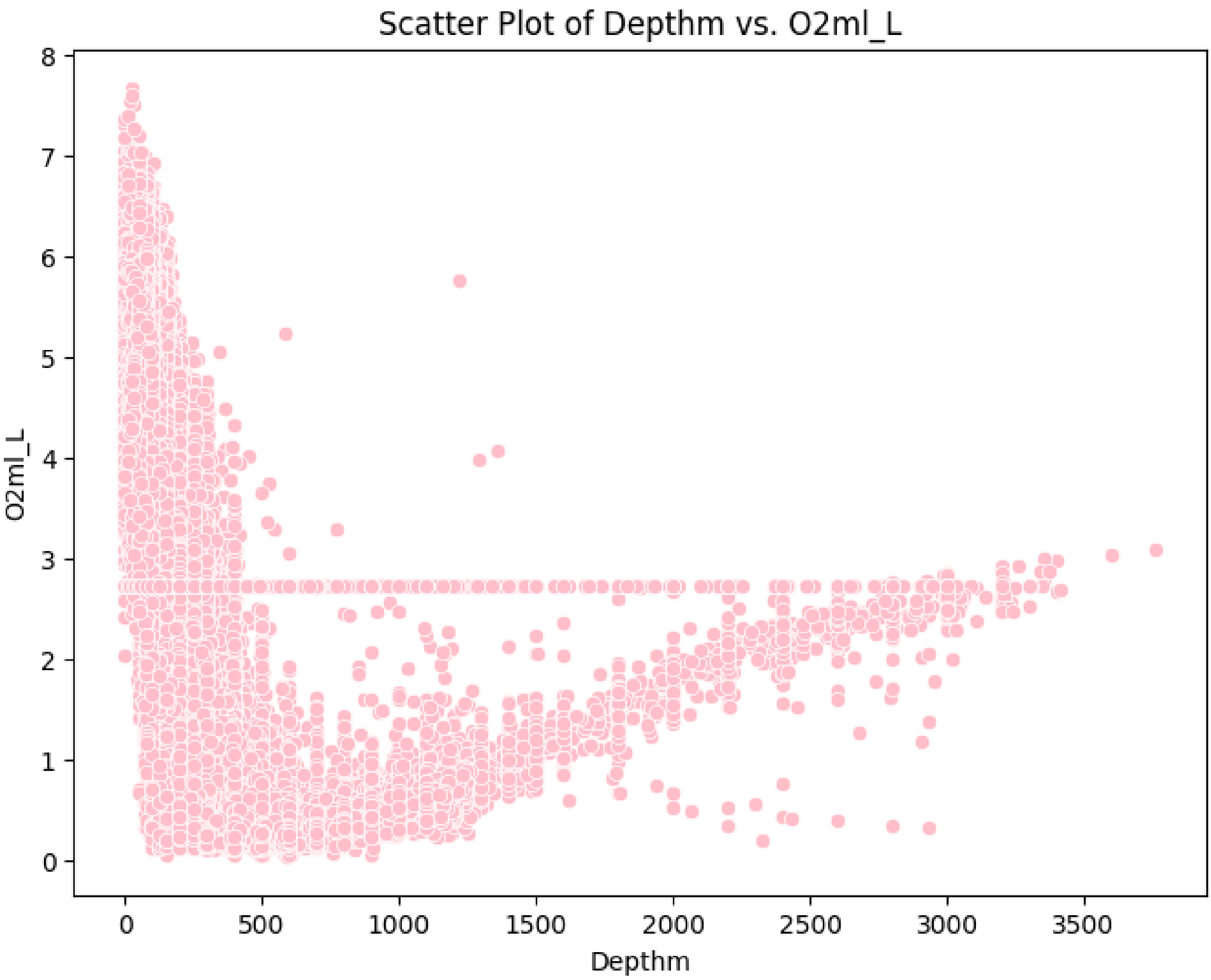


Distribution of R_DYNHT
Frequency
0
1000
2000
3000
4000
5000
6000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
R_DYNHT

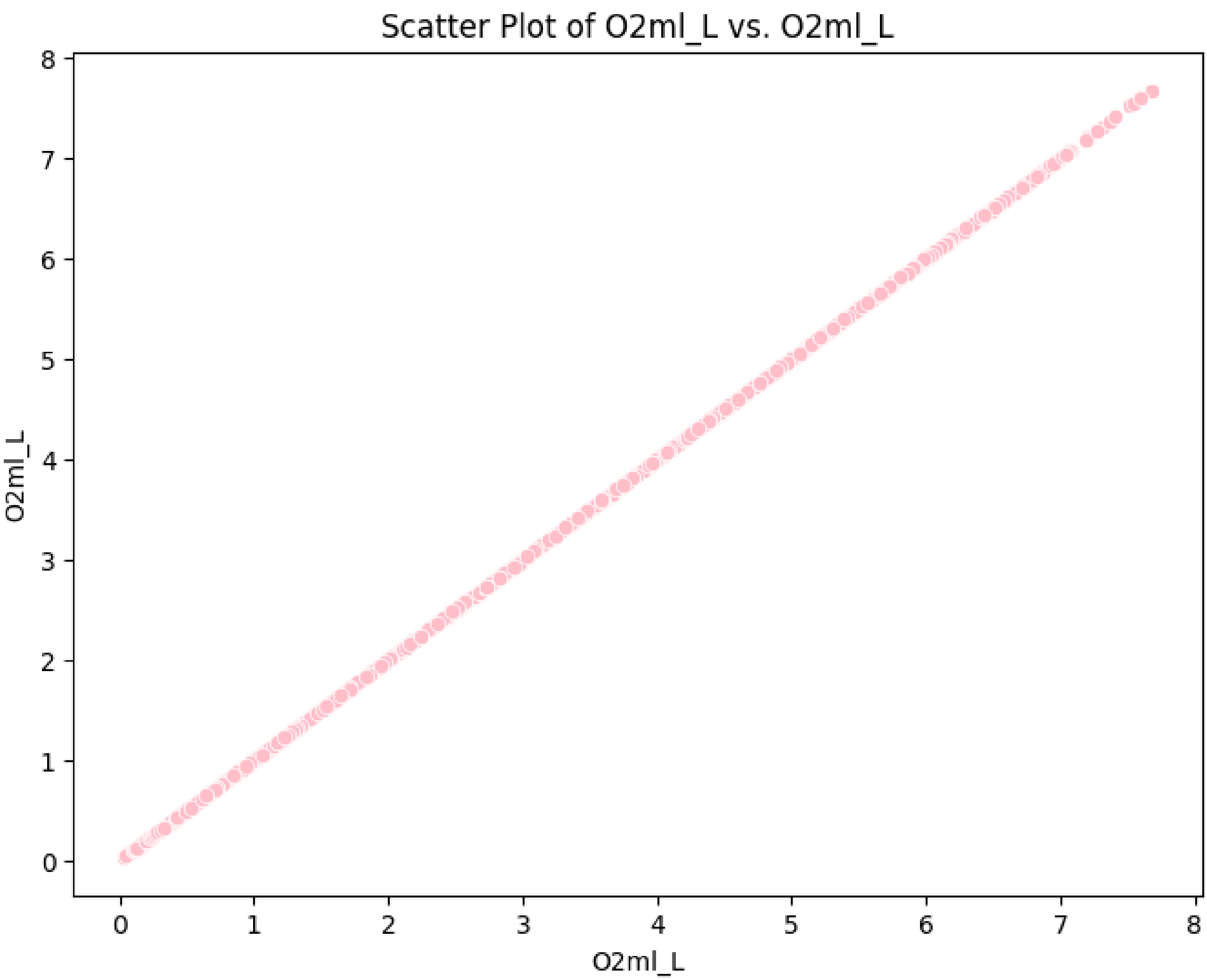




In [17]:
```
# Scatter Plots for Selected Features vs. Target Variable ('T_degC')
for feature in selected_features:
  plt.figure(figsize=(8,6))
  sns.scatterplot(x = df[feature], y = df['O2ml_L'],color='pink')
  plt.title(f'Scatter Plot of {feature} vs. O2ml_L')
  plt.xlabel(feature)
  plt.ylabel('O2ml_L')
  plt.show()
```

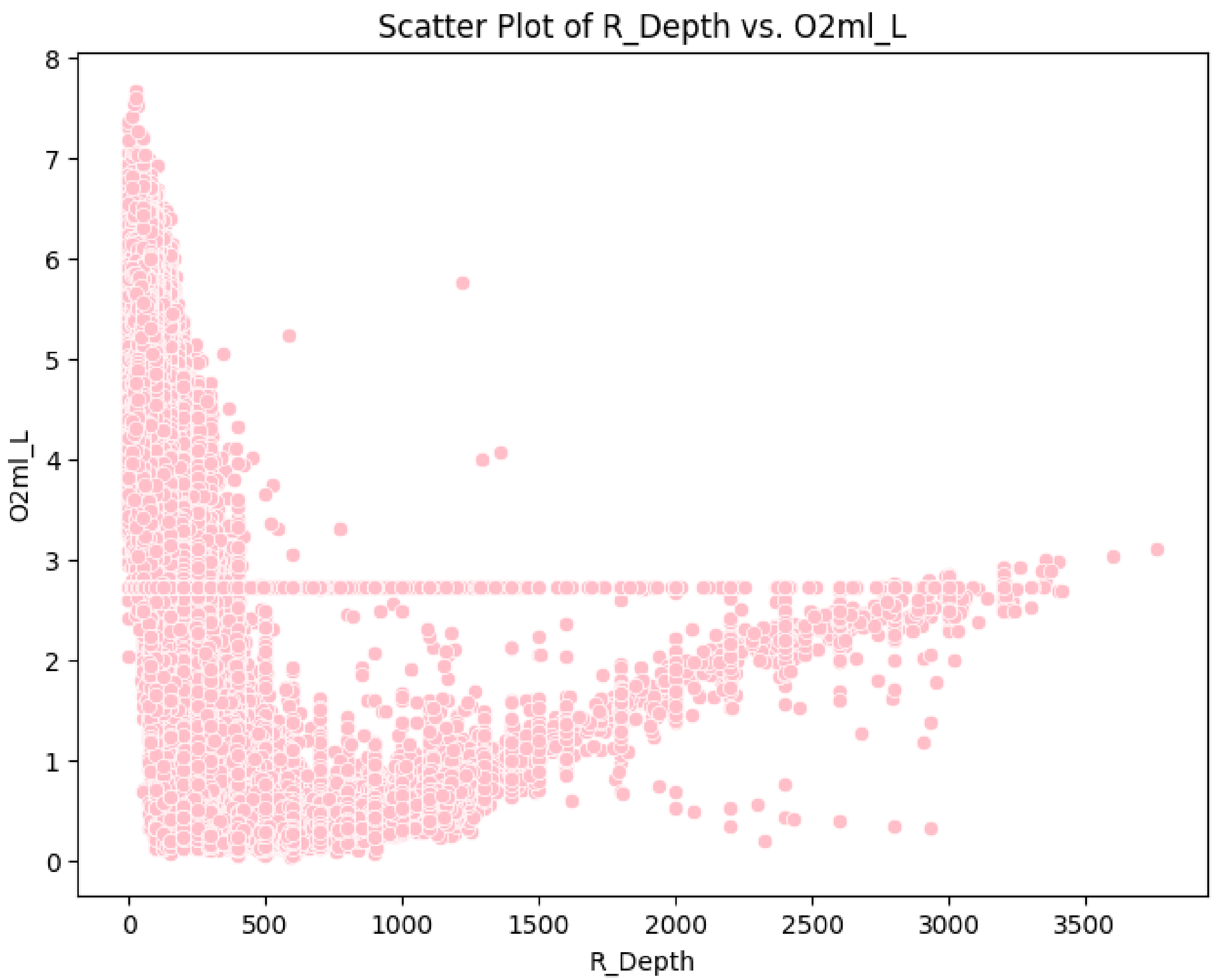


Scatter Plot of Depthm vs. O2ml_L
O2ml_L
Depthm
0
1
2
3
4
5
6
7
8
0
500
1000
1500
2000
2500
3000
3500

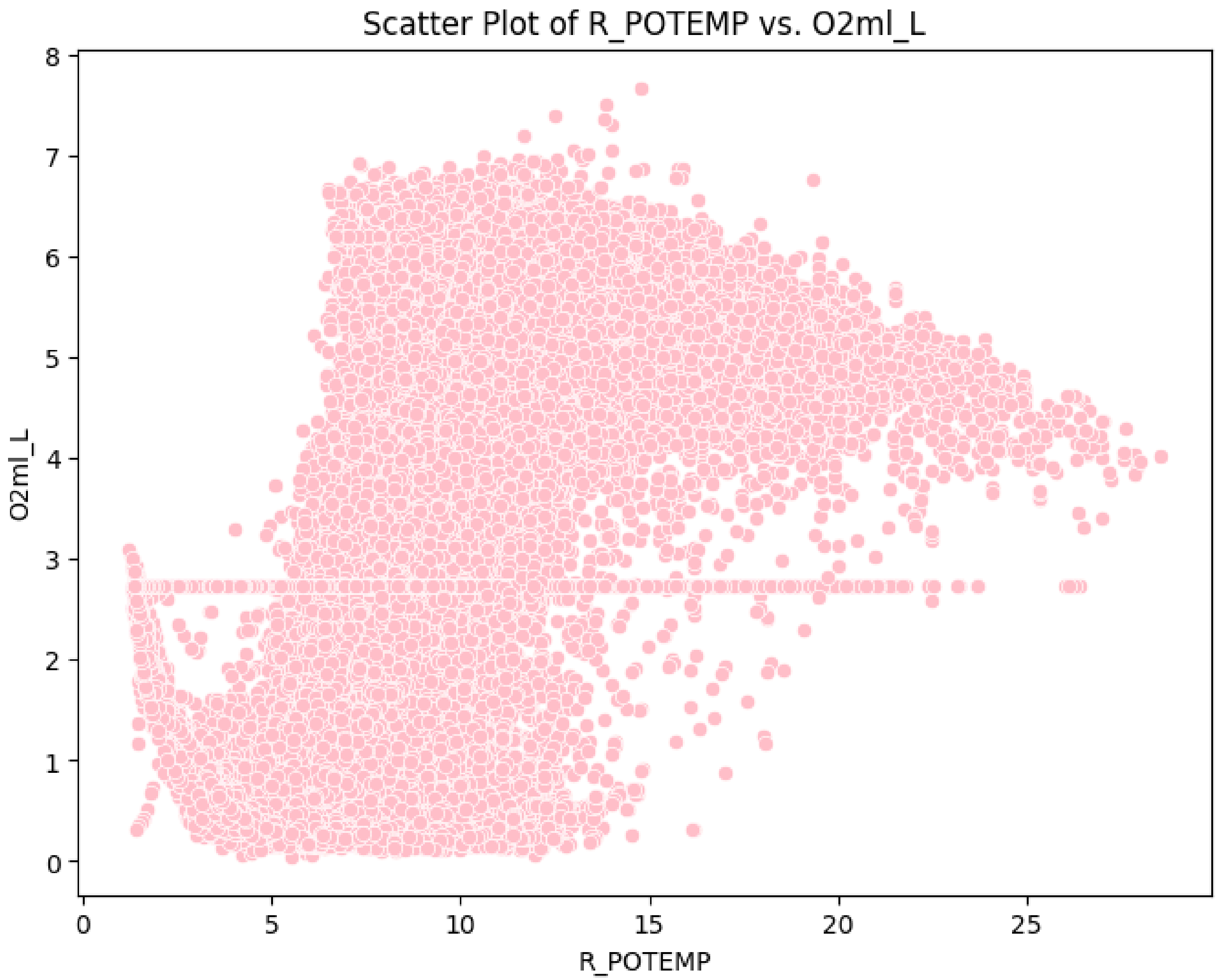


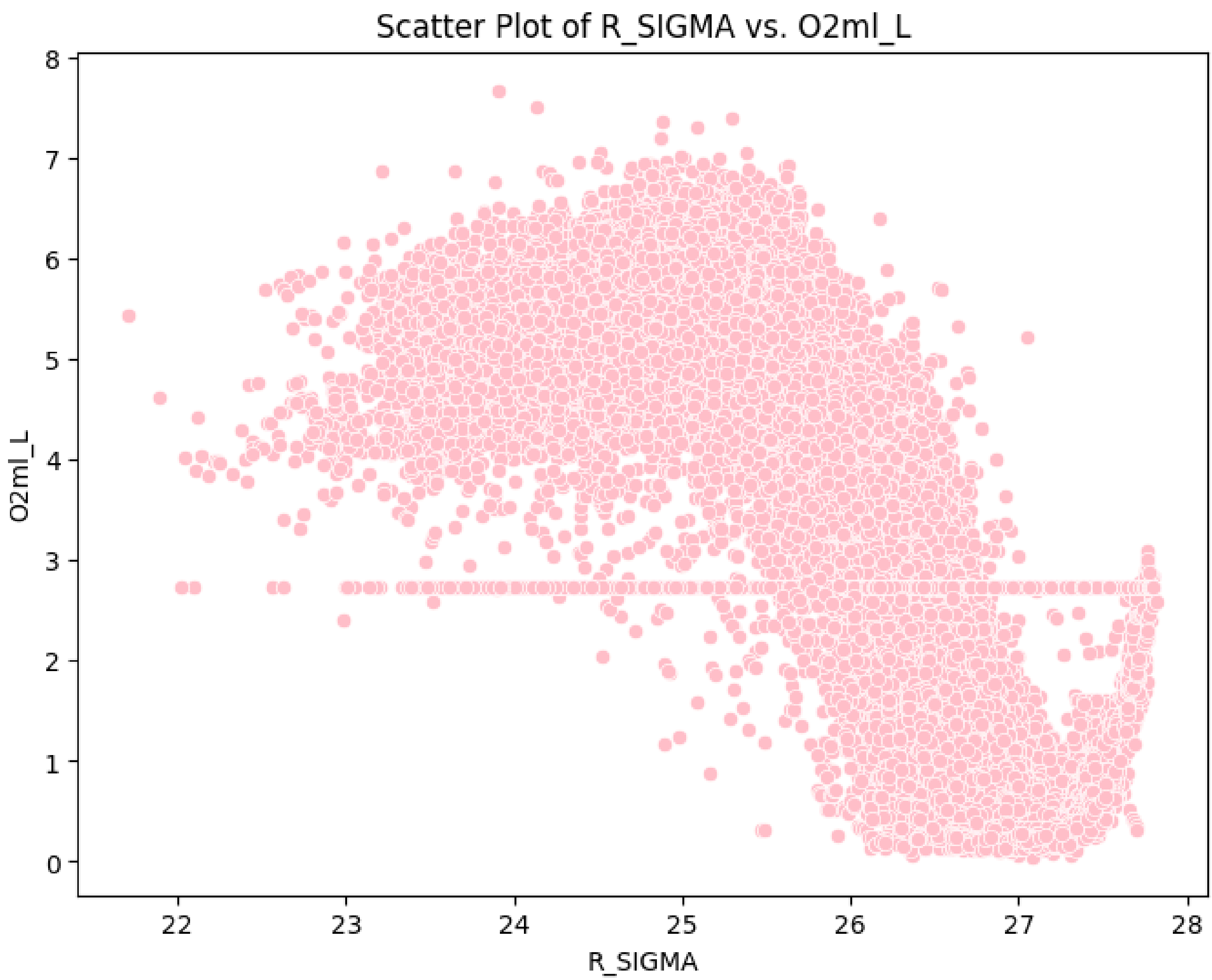
Scatter Plot of O2ml_L vs. O2ml_L
O2ml_L
O2ml_L
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8

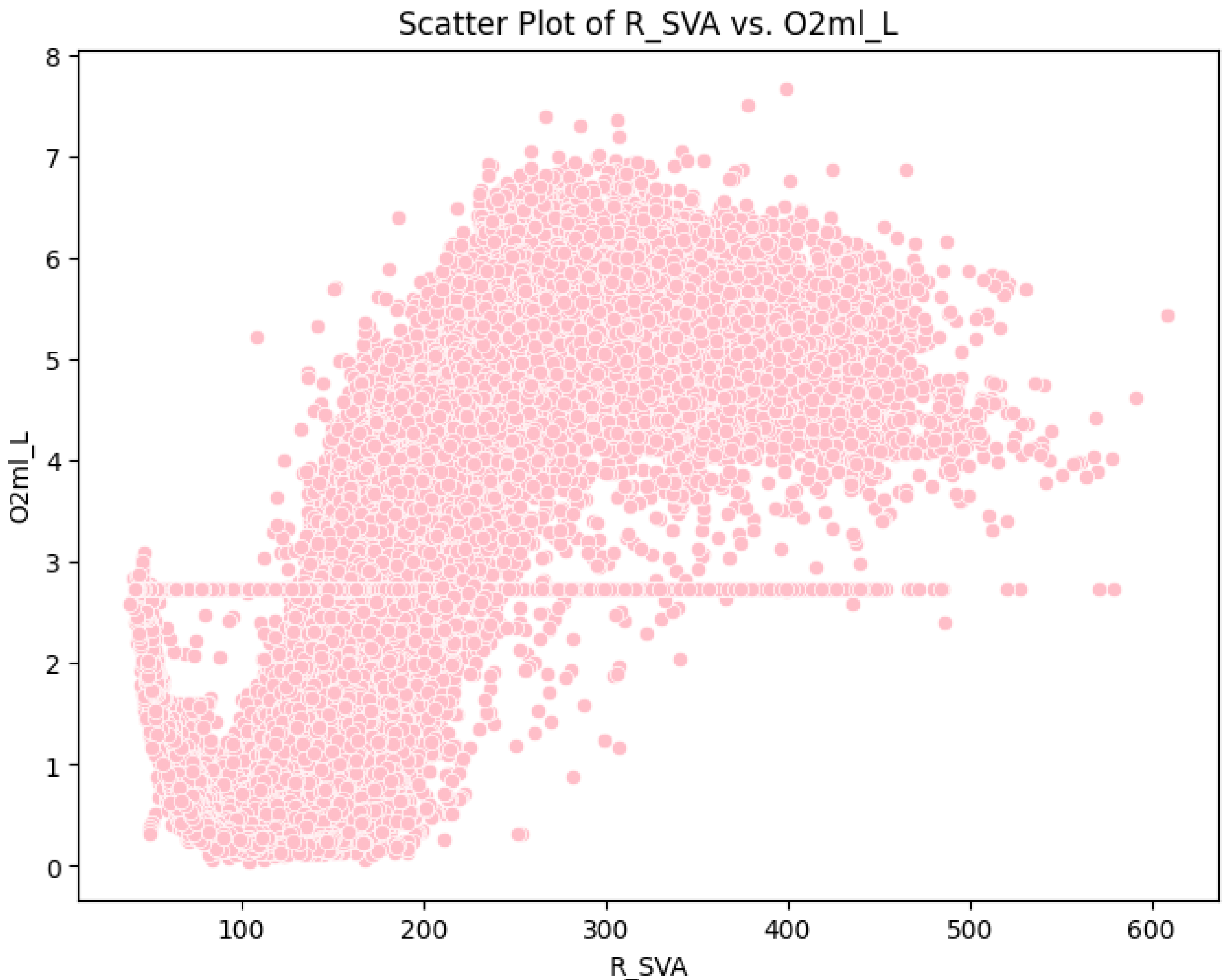
Scatter Plot of STheta vs. O2ml_L
O2ml_L
STheta
0
1
2
3
4
5
6
7
8
22
23
24
25
26
27
28

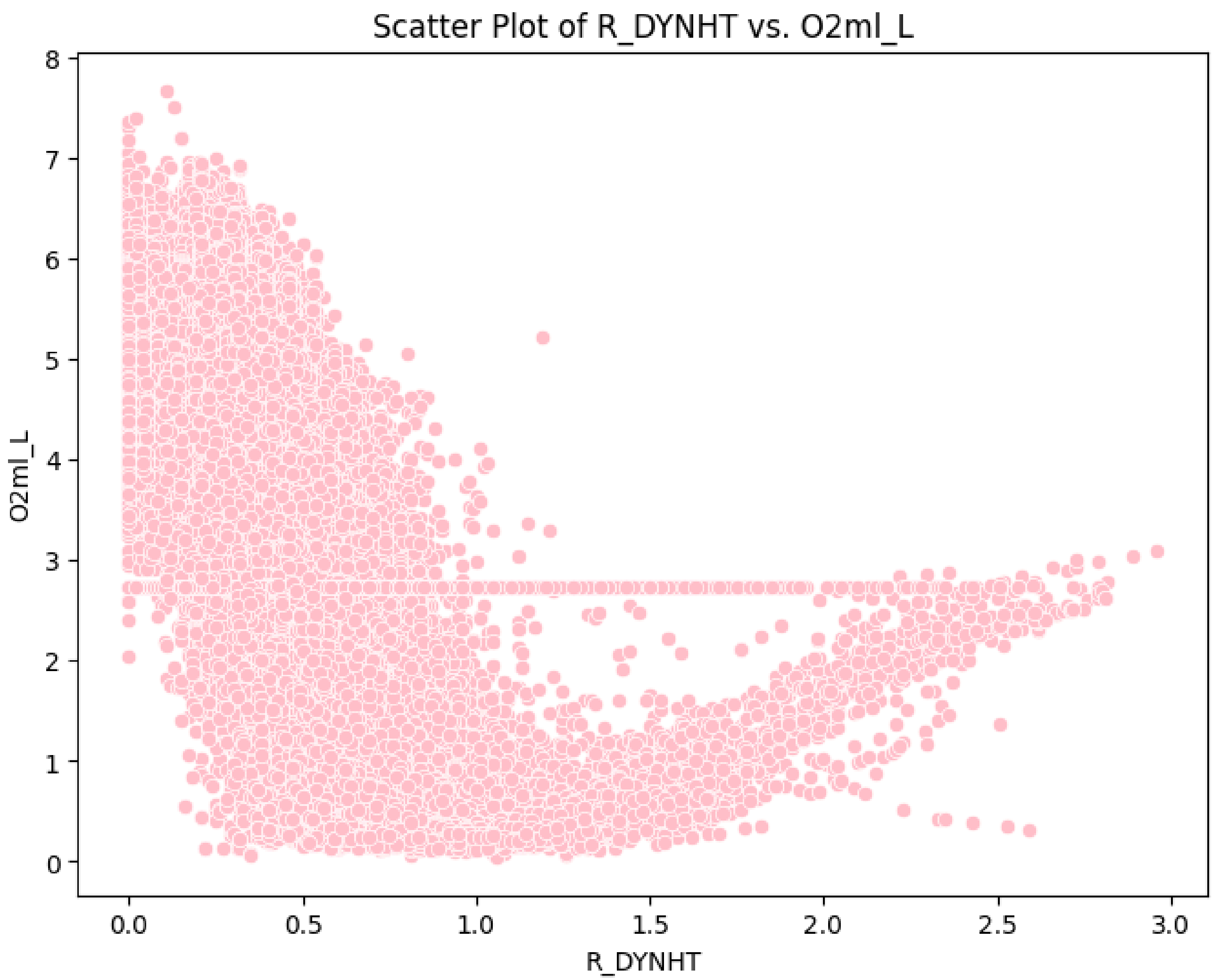


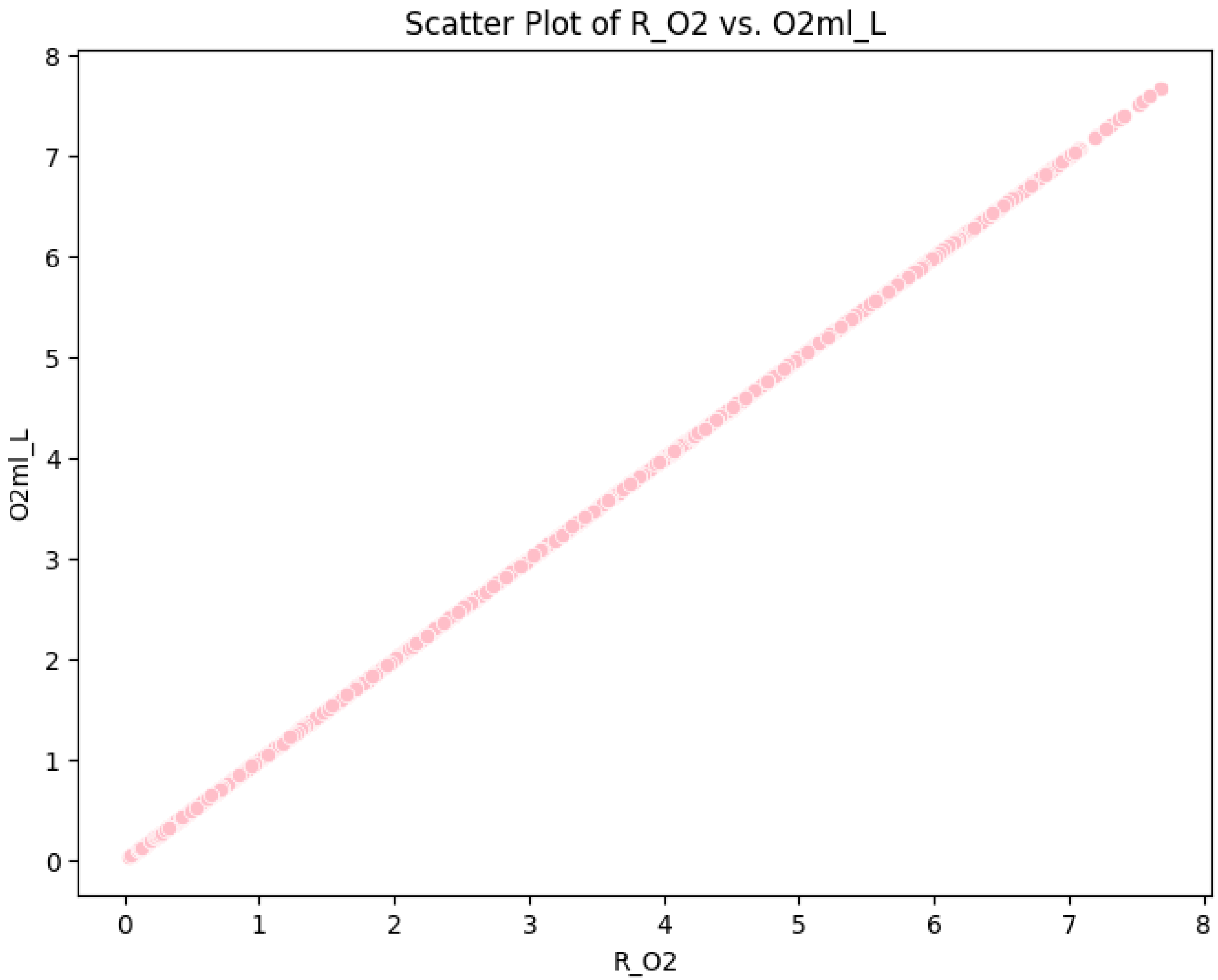
Scatter Plot of R_Depth vs. O2ml_L
O2ml_L
R_Depth
0
1
2
3
4
5
6
7
8
0
500
1000
1500
2000
2500
3000
3500

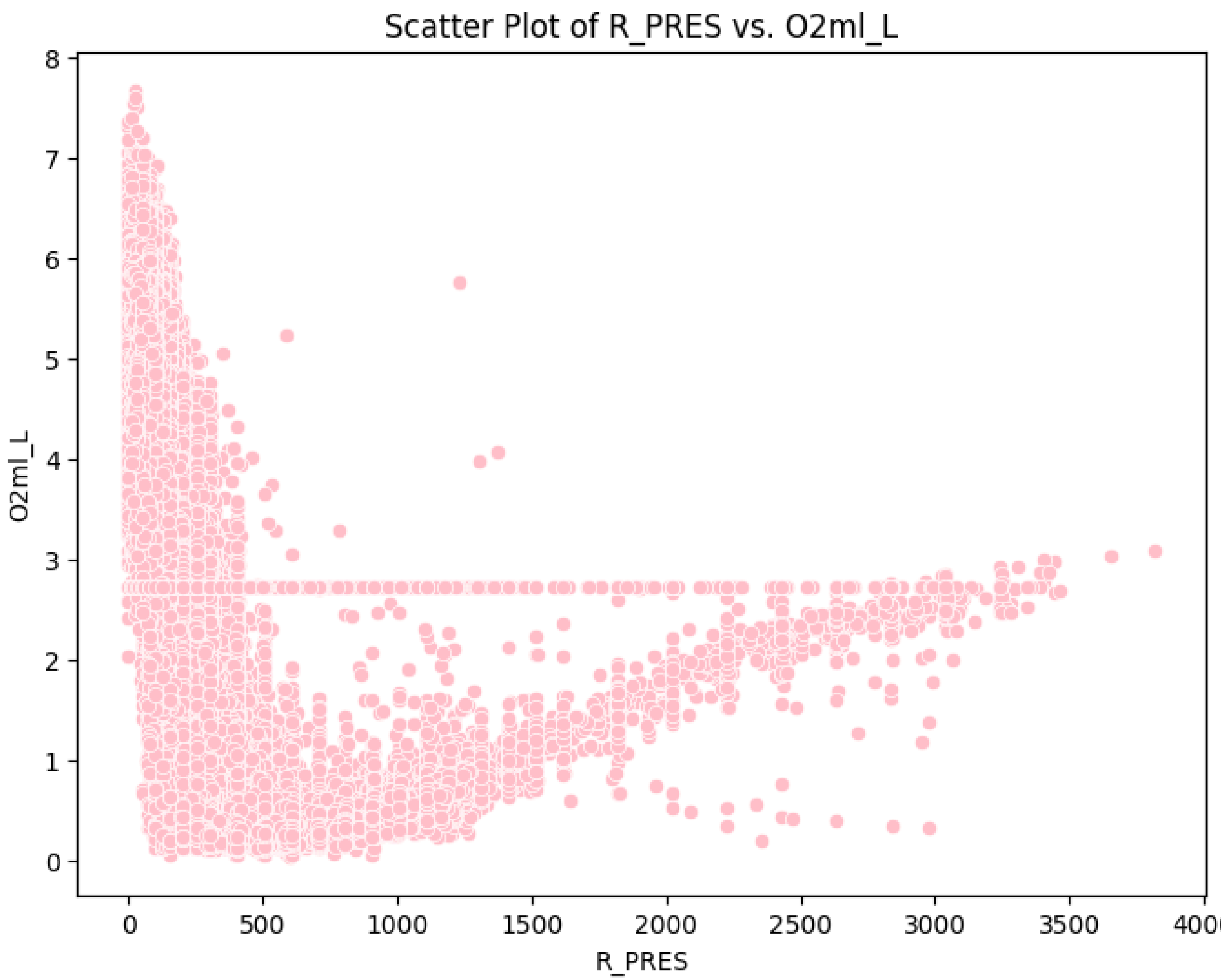

Scatter Plot of R_POTEMP vs. O2ml_L
O2ml_L
R_POTEMP
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25


Scatter Plot of R_SIGMA vs. O2ml_L
O2ml_L
R_SIGMA
0
1
2
3
4
5
6
7
8
22
23
24
25
26
27
28


Scatter Plot of R_SVA vs. O2ml_L
O2ml_L
R_SVA
0
1
2
3
4
5
6
7
8
100
200
300
400
500
600


Scatter Plot of R_DYNHT vs. O2ml_L
O2ml_L
R_DYNHT
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
2.0
2.5
3.0

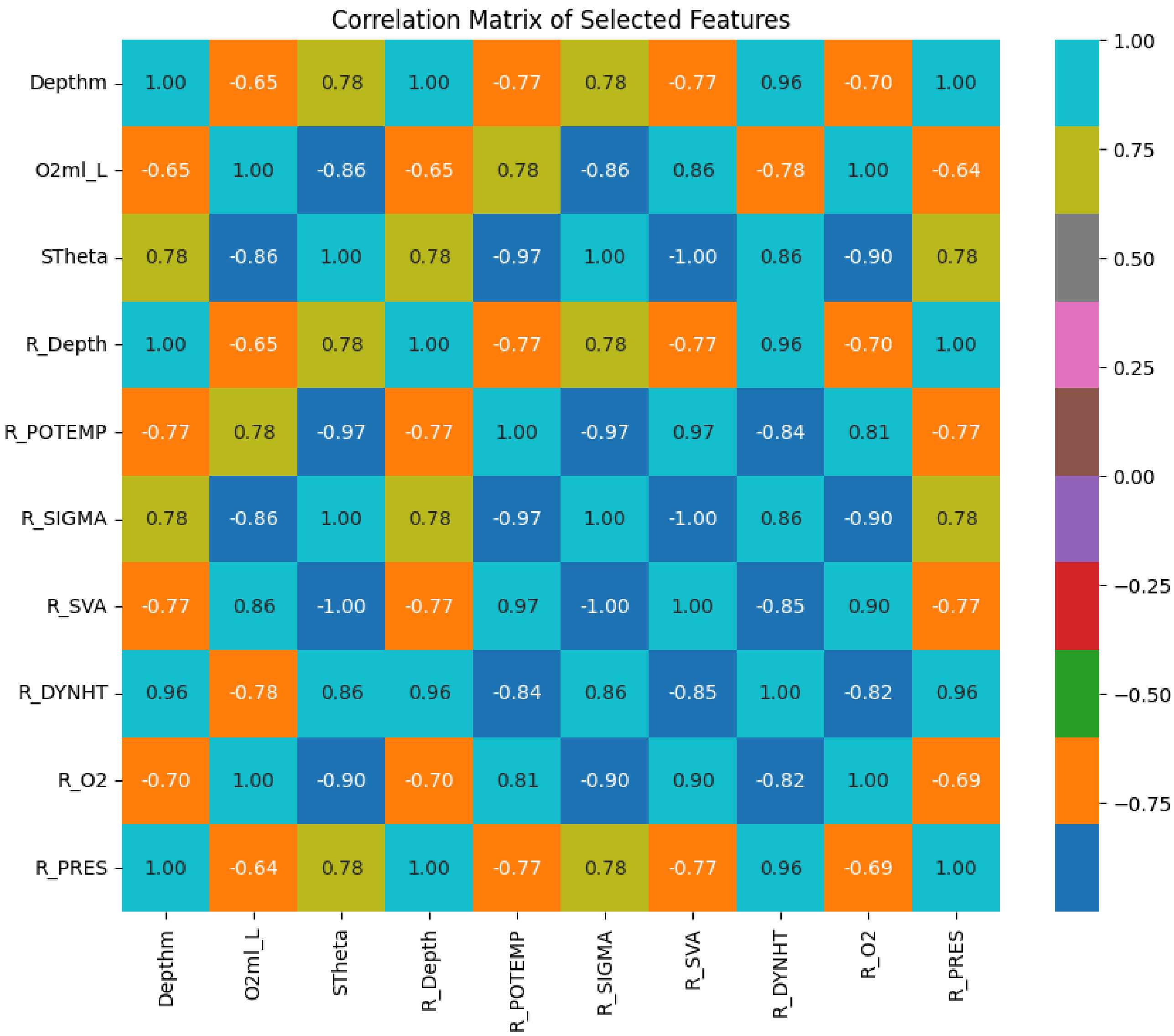




In [16]:

```
# Correlation Matrix Heatmap
plt.figure(figsize=(10, 8))
```

```
correlation_matrix = df[selected_features].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='tab10', fmt=".2f")
plt.title('Correlation Matrix of Selected Features')
plt.show()
```



In [18]:

```
# Data Split into Train and Test
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_siz

# Model Definitions
models = {
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'RandomForest': RandomForestRegressor(),
    'SVR': SVR(),
    'KNN': KNeighborsRegressor(),
    'DecisionTree': DecisionTreeRegressor()
}

# Model Training and Evaluation
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    results.append({
        'Model': name,
        'MSE': mse,
        'MAE': mae,
        'R2': r2
    })
```

In [20]:

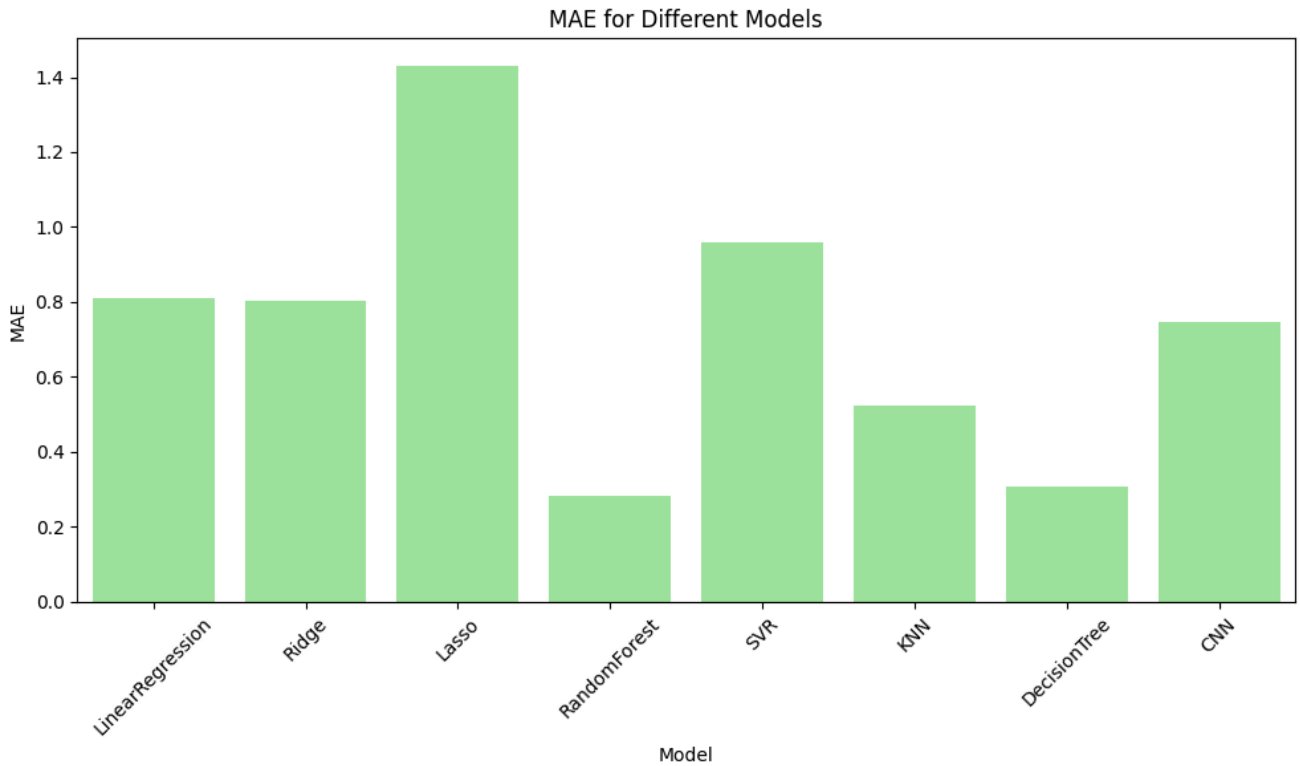
```
# Deep Learning Model (CNN for Regression)
def build_cnn(input_dim):
    model = Sequential()
    model.add(Dense(128, activation='relu', input_dim=input_dim))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1))  # Regression output
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['ma
    return model

cnn = build_cnn(X_train.shape[1])
cnn.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
y_pred_cnn = cnn.predict(X_test)

mse_cnn = mean_squared_error(y_test, y_pred_cnn)
mae_cnn = mean_absolute_error(y_test, y_pred_cnn)
r2_cnn = r2_score(y_test, y_pred_cnn)
results.append({
    'Model': 'CNN',
    'MSE': mse_cnn,
    'MAE': mae_cnn,
    'R2': r2_cnn
})

# Results Compilation
results_df = pd.DataFrame(results)
print(results_df)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do
not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[1m625/625[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step
              Model       MSE       MAE        R2
0  LinearRegression  1.658982  0.808742  0.914727
1             Ridge  1.665182  0.804111  0.914408
2             Lasso  3.881015  1.431757  0.800513
3      RandomForest  0.856572  0.282687  0.955972
4               SVR  2.525465  0.957615  0.870189
5               KNN  1.101703  0.523249  0.943372
6      DecisionTree  1.048364  0.306079  0.946113
7               CNN  1.254173  0.748133  0.935535
```

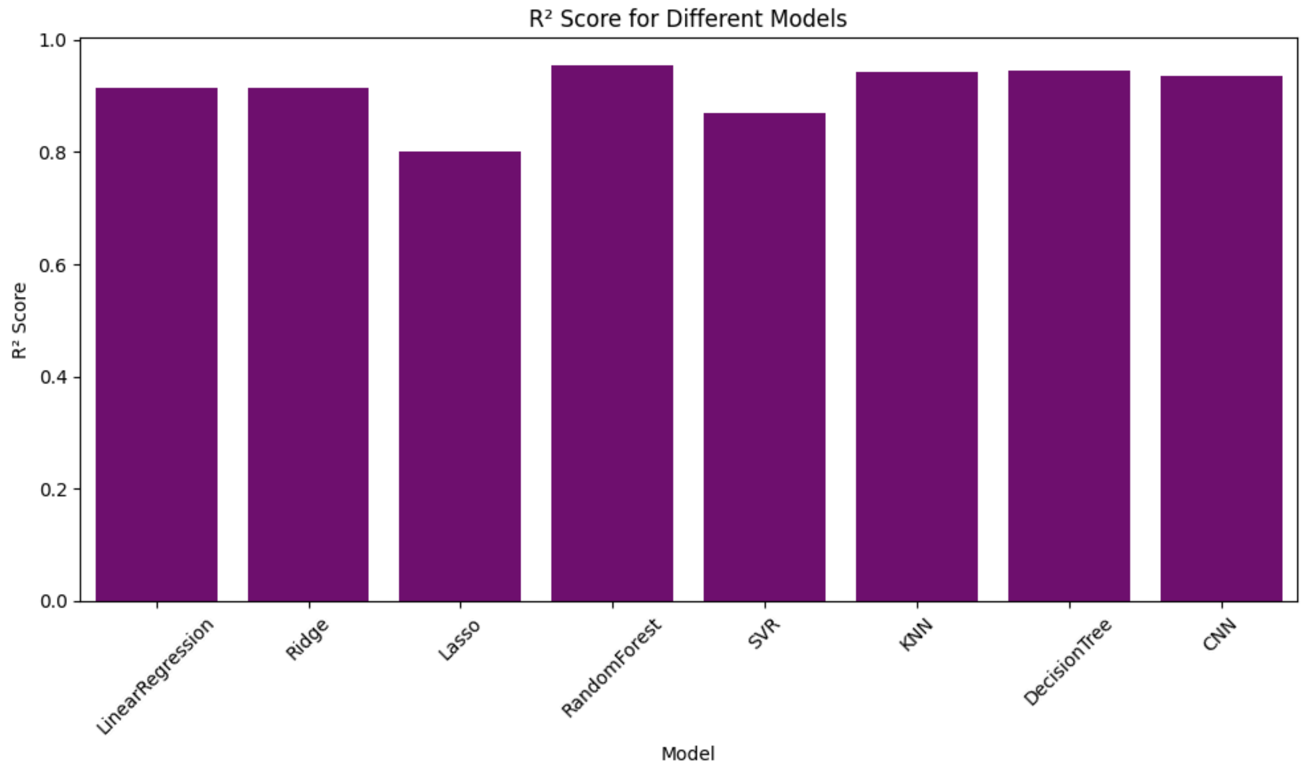
In [24]:

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

# Load the results from the CSV file
results_df.to_csv('model_results.csv', index=False)
results_df = pd.read_csv('model_results.csv')

# Accuracy Metrics Visualization
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='MAE', data=results_df,color='lightgreen')
plt.title('MAE for Different Models')
plt.xlabel('Model')
plt.ylabel('MAE')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [23]:

```
# Accuracy Metrics Visualization
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='R2', data=results_df,color='purple')
plt.title('R² Score for Different Models')
plt.xlabel('Model')
plt.ylabel('R² Score')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



R² Score for Different Models
1.0
0.8
0.6
0.4
0.2
0.0
R² Score
LinearRegression
Ridge
Lasso
RandomForest
SVR
KNN
DecisionTree
CNN
Model