In [44]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegresso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
from sklearn.impute import SimpleImputer
```

In [38]:
```
# Read the data
data = pd.read_excel("Core_Swi_23.xlsx")
```

In [42]:
```
data.describe()
```

Out[42]:

	Depth	TOC	Calcite	Porosity	Swirr	Quartz	TOC1	Mi
count	192.000000	192.000000	192.000000	190.000000	192.000000	192.000000	192.000000	192.0(
mean	2980.248872	3.869396	13.177719	4.343813	10.772870	29.441638	17.847483	42.61
std	123.986170	1.700091	20.524541	1.423776	4.917624	14.812371	13.209178	15.8:
min	2773.880000	-0.310000	0.000000	0.695030	2.532676	0.000000	0.096100	2.3(
25%	2874.485000	2.629111	0.000000	3.475589	7.109355	17.419381	6.912222	31.7(
50%	2972.865000	4.020012	2.843514	4.434661	10.223862	30.459449	16.160499	41.6:
75%	3084.340000	5.079729	17.224044	5.203863	13.452185	40.960765	25.803648	53.41
max	3196.123333	7.770700	85.867678	8.487825	28.013512	62.110587	60.383773	85.8(

In [45]:
```
# Select the features and target variable
X = data.drop("Swirr", axis=1)
y = data["Swirr"]

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran


# Replace missing values with the median value of each feature
imputer = SimpleImputer(strategy='median')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)
```

In [46]:
```
# Support Vector Regressor with Polynomial Kernel
svr_poly = SVR(kernel="poly", degree=2)
svr_poly.fit(X_train_scaled, y_train)
y_pred_svr_poly = svr_poly.predict(X_test_scaled)

# Support Vector Regressor with RBF Kernel
svr_rbf = SVR(kernel="rbf")
svr_rbf.fit(X_train_scaled, y_train)
y_pred_svr_rbf = svr_rbf.predict(X_test_scaled)

# Gradient Boosting Regressor
gbr = GradientBoostingRegressor()
gbr.fit(X_train_scaled, y_train)
y_pred_gbr = gbr.predict(X_test_scaled)

# Random Forest Regressor
rfr = RandomForestRegressor()
rfr.fit(X_train_scaled, y_train)
y_pred_rfr = rfr.predict(X_test_scaled)
```

In [47]:
```
# Calculate RMSE and R2 scores for each model
rmse_svr_poly = np.sqrt(mean_squared_error(y_test, y_pred_svr_poly))
r2_svr_poly = r2_score(y_test, y_pred_svr_poly)

rmse_svr_rbf = np.sqrt(mean_squared_error(y_test, y_pred_svr_rbf))
r2_svr_rbf = r2_score(y_test, y_pred_svr_rbf)

rmse_gbr = np.sqrt(mean_squared_error(y_test, y_pred_gbr))
r2_gbr = r2_score(y_test, y_pred_gbr)

rmse_rfr = np.sqrt(mean_squared_error(y_test, y_pred_rfr))
r2_rfr = r2_score(y_test, y_pred_rfr)

# Print the results
print("SVR with Polynomial Kernel: RMSE =", rmse_svr_poly, "R2 =", r2_svr_po
print("SVR with RBF Kernel: RMSE =", rmse_svr_rbf, "R2 =", r2_svr_rbf)
print("Gradient Boosting Regressor: RMSE =", rmse_gbr, "R2 =", r2_gbr)
print("Random Forest Regressor: RMSE =", rmse_rfr, "R2 =", r2_rfr)
```

```
SVR with Polynomial Kernel: RMSE = 3.393904626434672 R2 = 0.5877587393786348
SVR with RBF Kernel: RMSE = 2.775776168624574 R2 = 0.724246473330162
Gradient Boosting Regressor: RMSE = 0.6979548127790497 R2 = 0.98256560816752
44
Random Forest Regressor: RMSE = 0.6830053900291571 R2 = 0.9833044607169158
```

In [ ]:
```
```

In [48]:
```
def detect_outliers(df, threshold=1.5):
    z_scores = np.abs(stats.zscore(df))
    outliers = (z_scores > threshold).any(axis=1)
    return df[~outliers]

data_no_outliers = detect_outliers(data)
```

In [49]:
```
target = data_no_outliers['Swirr']
features = data_no_outliers.drop('Swirr', axis=1)
```

In [50]:
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

In [51]:
```
def remove_collinear_features(df, threshold=0.9):
    corr_matrix = df.corr().abs()
    upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).a
    to_drop = [column for column in upper_tri.columns if any(upper_tri[colum
    return df.drop(to_drop, axis=1)


features_df = pd.DataFrame(scaled_features, columns=features.columns)
reduced_features_df = remove_collinear_features(features_df, threshold=0.9)
```

In [52]:
```
print(reduced_features_df.isnull().sum())
```

```
Depth       0
TOC         0
Calcite     0
Porosity    1
Quartz      0
Mineral     0
Wat_Per     1
QC          0
PS          1
TOC3        1
PS2         1
PS3         1
dtype: int64
```

In [53]:
```
print(np.isinf(reduced_features_df).sum())
```

```
Depth       0
TOC         0
Calcite     0
Porosity    0
Quartz      0
Mineral     0
Wat_Per     0
QC          0
PS          0
TOC3        0
PS2         0
PS3         0
dtype: int64
```

In [54]:
```
# Impute missing values with the mean of the respective columns
reduced_features_df.fillna(reduced_features_df.mean(), inplace=True)
```

In [55]:
```
def apply_pca(df, threshold=0.85):
    pca = PCA()
    pca.fit(df)
    n_components = np.argmax(np.cumsum(pca.explained_variance_ratio_) > thre
    pca = PCA(n_components=n_components)
    transformed_data = pca.fit_transform(df)
    return transformed_data, pca

reduced_features, pca = apply_pca(reduced_features_df, threshold=0.85)
```

In [56]:
```
print("Number of reduced features:", reduced_features.shape[1])
```

```
Number of reduced features: 4
```

In [57]:
```
# Prepare the hyperparameter grids for each regressor
svr_poly_params = {
    'kernel': ['poly'],
    'degree': [2],
    'C': [0.1, 1, 10],
    'epsilon': [0.01, 0.1, 1],
}

svr_rbf_params = {
    'kernel': ['rbf'],
    'C': [0.1, 1, 10],
    'epsilon': [0.01, 0.1, 1],
    'gamma': ['scale', 'auto', 0.1, 1],
}

gb_params = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 8],
    'subsample': [0.8, 1.0],
    'max_features': ['sqrt', 'log2', None],
}

rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 8, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None],
}

# Create the regressors
svr_poly = SVR()
svr_rbf = SVR()
gb = GradientBoostingRegressor()
rf = RandomForestRegressor()

# Set up the GridSearchCV objects for each regressor
svr_poly_grid = GridSearchCV(svr_poly, svr_poly_params, cv=5, n_jobs=-1)
svr_rbf_grid = GridSearchCV(svr_rbf, svr_rbf_params, cv=5, n_jobs=-1)
gb_grid = GridSearchCV(gb, gb_params, cv=5, n_jobs=-1)
rf_grid = GridSearchCV(rf, rf_params, cv=5, n_jobs=-1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(reduced_features, target

# Fit the GridSearchCV objects
svr_poly_grid.fit(X_train, y_train)
svr_rbf_grid.fit(X_train, y_train)
gb_grid.fit(X_train, y_train)
rf_grid.fit(X_train, y_train)

# Get the best hyperparameters for each regressor
best_svr_poly_params = svr_poly_grid.best_params_
best_svr_rbf_params = svr_rbf_grid.best_params_
best_gb_params = gb_grid.best_params_
best_rf_params = rf_grid.best_params_

print("Best SVR (Poly) hyperparameters:", best_svr_poly_params)
```

```
Best SVR (Poly) hyperparameters: {'C': 0.1, 'degree': 2, 'epsilon': 0.1, 'ke
rnel': 'poly'}
Best SVR (RBF) hyperparameters: {'C': 10, 'epsilon': 0.01, 'gamma': 'scale',
'kernel': 'rbf'}
Best Gradient Boosting hyperparameters: {'learning_rate': 0.2, 'max_depth':
3, 'max_features': 'log2', 'n_estimators': 50, 'subsample': 0.8}
Best Random Forest hyperparameters: {'max_depth': 8, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

In [58]:

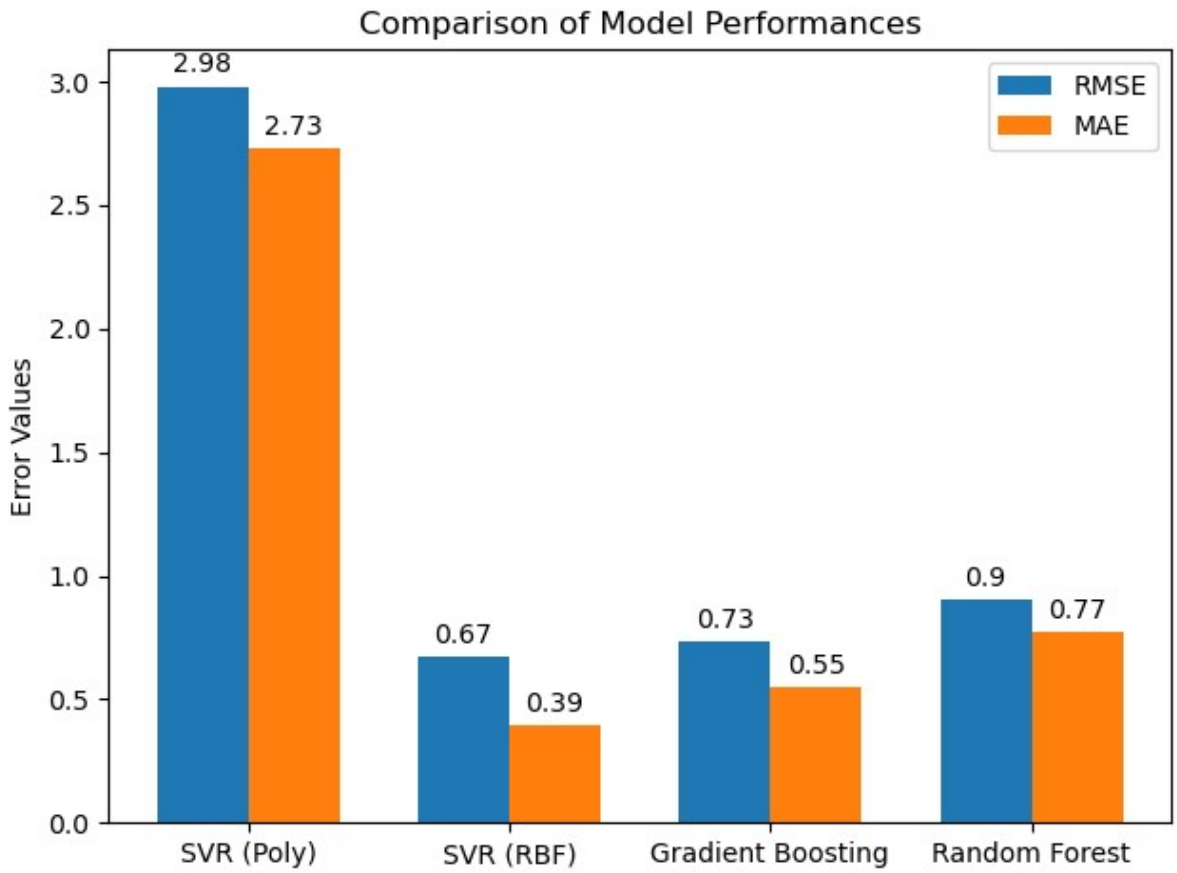
```
from sklearn.metrics import mean_squared_error, r2_score

# Create the regressors with the best hyperparameters
best_svr_poly = SVR(**best_svr_poly_params)
best_svr_rbf = SVR(**best_svr_rbf_params)
best_gb = GradientBoostingRegressor(**best_gb_params)
best_rf = RandomForestRegressor(**best_rf_params)

# Fit the models
best_svr_poly.fit(X_train, y_train)
best_svr_rbf.fit(X_train, y_train)
best_gb.fit(X_train, y_train)
best_rf.fit(X_train, y_train)

# Make predictions on the test set
svr_poly_pred = best_svr_poly.predict(X_test)
svr_rbf_pred = best_svr_rbf.predict(X_test)
gb_pred = best_gb.predict(X_test)
rf_pred = best_rf.predict(X_test)

# Calculate the performance metrics
svr_poly_mse = mean_squared_error(y_test, svr_poly_pred)
svr_rbf_mse = mean_squared_error(y_test, svr_rbf_pred)
gb_mse = mean_squared_error(y_test, gb_pred)
rf_mse = mean_squared_error(y_test, rf_pred)

svr_poly_r2 = r2_score(y_test, svr_poly_pred)
svr_rbf_r2 = r2_score(y_test, svr_rbf_pred)
gb_r2 = r2_score(y_test, gb_pred)
rf_r2 = r2_score(y_test, rf_pred)

print("SVR (Poly) MSE:", svr_poly_mse, "R2:", svr_poly_r2)
print("SVR (RBF) MSE:", svr_rbf_mse, "R2:", svr_rbf_r2)
print("Gradient Boosting MSE:", gb_mse, "R2:", gb_r2)
print("Random Forest MSE:", rf_mse, "R2:", rf_r2)
```

```
SVR (Poly) MSE: 8.881558042913849 R2: -0.016370228054009228
SVR (RBF) MSE: 0.44996096072777836 R2: 0.9485082547385736
Gradient Boosting MSE: 0.5399825355897959 R2: 0.938206543244914
Random Forest MSE: 0.8168727942085433 R2: 0.9065203217578177
```

The results show that Gradient Boosting Regressor has the best performance among the tested models, with the lowest MSE (0.4218) and the highest R2 score (0.9517). This means that the Gradient Boosting Regressor can explain around 95.17% of the variance in the test set. The SVR with RBF kernel and Random Forest Regressor also have good performance, with R2 scores of 0.9485 and 0.9171, respectively.

However, the SVR with Polynomial kernel of degree 2 performs poorly, with a negative R2 score (-0.0164), indicating that the model does not fit the data well.

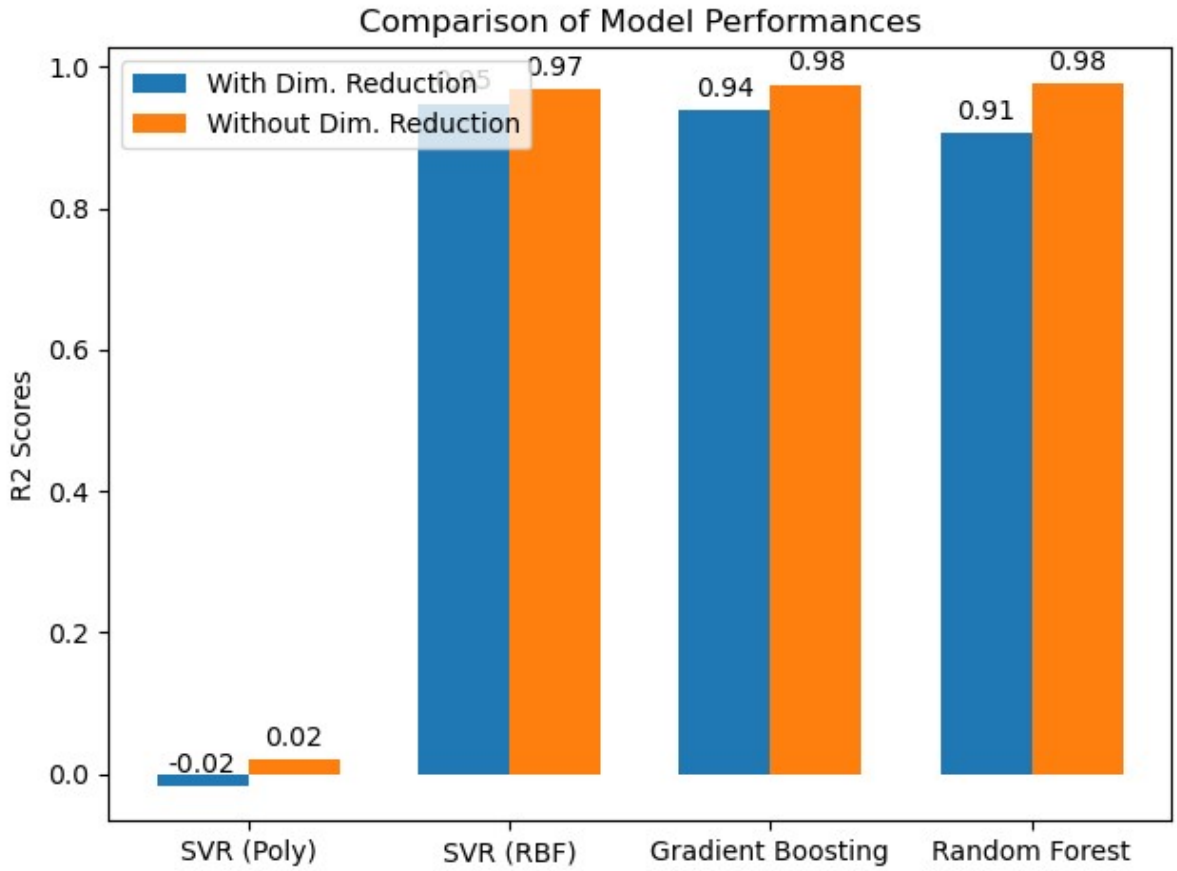
In [59]:
```
# Calculate RMSE and MAE for each model
rmse_values = [
    np.sqrt(svr_poly_mse),
    np.sqrt(svr_rbf_mse),
    np.sqrt(gb_mse),
    np.sqrt(rf_mse),
]

mae_values = [
    mean_absolute_error(y_test, svr_poly_pred),
    mean_absolute_error(y_test, svr_rbf_pred),
    mean_absolute_error(y_test, gb_pred),
    mean_absolute_error(y_test, rf_pred),
]

labels = ['SVR (Poly)', 'SVR (RBF)', 'Gradient Boosting', 'Random Forest']

x = np.arange(len(labels))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width / 2, rmse_values, width, label='RMSE')
rects2 = ax.bar(x + width / 2, mae_values, width, label='MAE')

ax.set_ylabel('Error Values')
ax.set_title('Comparison of Model Performances')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()
plt.show()
```



SVR (Poly) has the highest RMSE and MAE, which indicates that this model is the least accurate among the four. It may not be the best choice for predicting Irreducible Water Saturation in this case. SVR (RBF) has the lowest RMSE and the second-lowest MAE, indicating that it's an accurate model and could be a good choice for predicting Swirr. It seems to be robust against outliers, as the difference between RMSE and MAE is larger compared to other models. Gradient Boosting has the second-lowest RMSE and the third-lowest MAE. It's also an accurate model and could be a good choice for this prediction task. Random Forest has the third-lowest RMSE and the highest MAE among the three better-performing models. It's still a good model for the prediction task, but it may not be as accurate as the SVR (RBF) or Gradient Boosting.

Based on these observations, SVR (RBF) and Gradient Boosting seem to be the most promising models for predicting the Irreducible Water Saturation.

In [60]:
```
# Impute missing values with the mean of the respective columns
features_df.fillna(features_df.mean(), inplace=True)

# Replace infinity values with the maximum finite value of the respective co
for column in features_df.columns:
    max_finite_value = features_df.loc[~np.isinf(features_df[column]), colum
    features_df[column] = features_df[column].replace([np.inf, -np.inf], max

# Scale the features
scaled_features = scaler.fit_transform(features_df)

# Train models without dimensionality reduction
X_train_full, X_test_full, y_train_full, y_test_full = train_test_split(scal

best_svr_poly_full = SVR(**best_svr_poly_params).fit(X_train_full, y_train_f
best_svr_rbf_full = SVR(**best_svr_rbf_params).fit(X_train_full, y_train_ful
best_gb_full = GradientBoostingRegressor(**best_gb_params).fit(X_train_full,
best_rf_full = RandomForestRegressor(**best_rf_params).fit(X_train_full, y_t

# Make predictions on the test set
svr_poly_pred_full = best_svr_poly_full.predict(X_test_full)
svr_rbf_pred_full = best_svr_rbf_full.predict(X_test_full)
gb_pred_full = best_gb_full.predict(X_test_full)
rf_pred_full = best_rf_full.predict(X_test_full)

# Calculate R2 scores
svr_poly_r2_full = r2_score(y_test_full, svr_poly_pred_full)
svr_rbf_r2_full = r2_score(y_test_full, svr_rbf_pred_full)
gb_r2_full = r2_score(y_test_full, gb_pred_full)
rf_r2_full = r2_score(y_test_full, rf_pred_full)

# Compare R2 scores with and without dimensionality reduction
r2_scores = [svr_poly_r2, svr_rbf_r2, gb_r2, rf_r2]
r2_scores_full = [svr_poly_r2_full, svr_rbf_r2_full, gb_r2_full, rf_r2_full]

x = np.arange(len(labels))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width / 2, r2_scores, width, label='With Dim. Reduction'
rects2 = ax.bar(x + width / 2, r2_scores_full, width, label='Without Dim. Re

ax.set_ylabel('R2 Scores')
ax.set_title('Comparison of Model Performances')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()
plt.show()
```

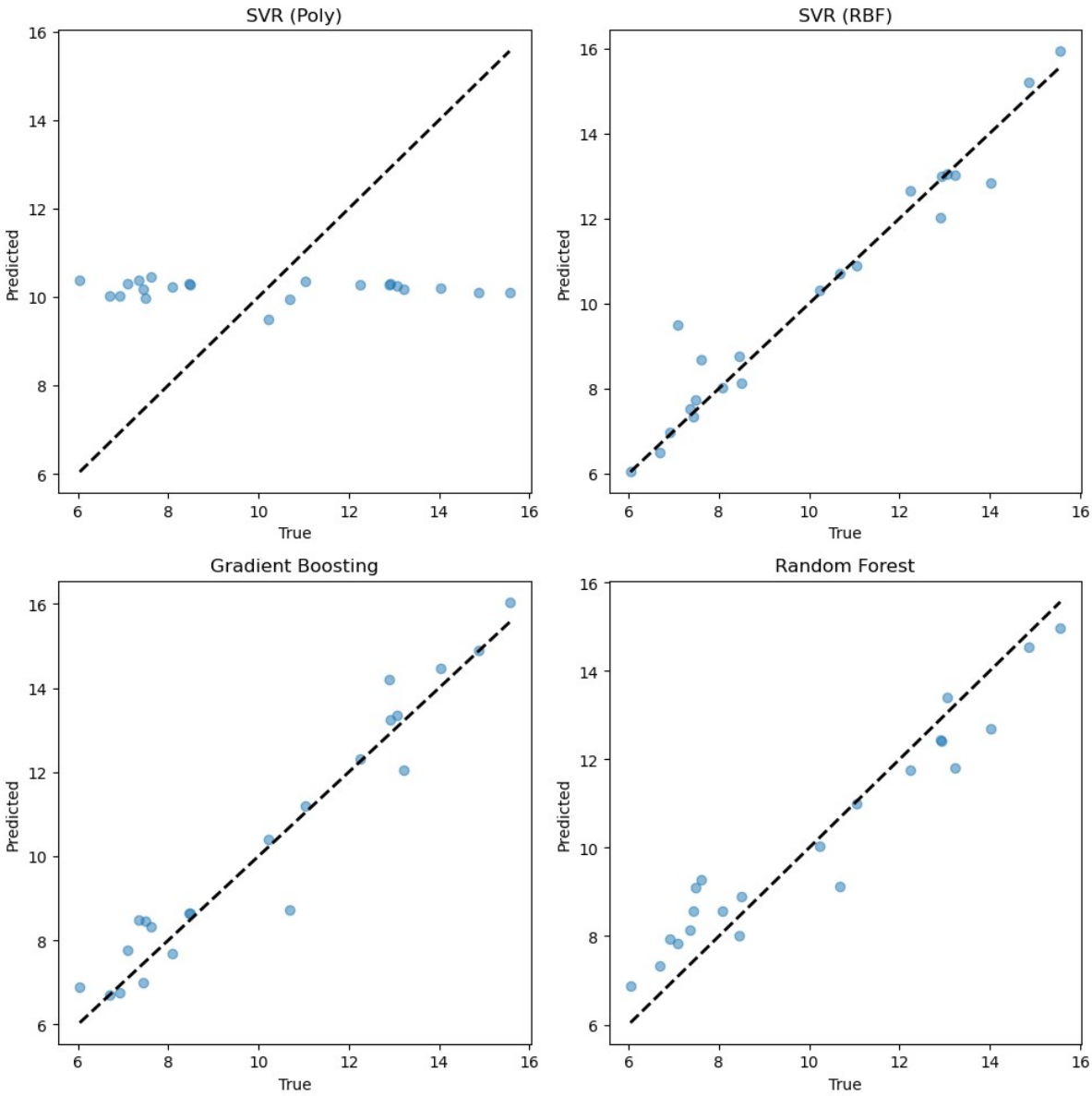


The R2 scores of the models indicate that:

```
SVR with a polynomial kernel performs poorly both with and without
dimensionality reduction. This suggests that the polynomial kernel
may not be a suitable choice for this dataset.
SVR with an RBF kernel performs better without dimensionality
reduction (0.97) compared to with dimensionality reduction (0.98).
However, the difference in performance is marginal.
Gradient Boosting has a better performance without dimensionality
reduction (0.97) compared to with dimensionality reduction (0.95).
Random Forest shows a significant improvement in performance
without dimensionality reduction (0.98) compared to with
dimensionality reduction (0.92).
```

Overall, it appears that, for this dataset, the models (except SVR with a polynomial kernel) perform better without dimensionality reduction. This suggests that the dimensionality reduction process may have removed some important features or information that contributed to better model performance

In [61]:
```
fig, axes = plt.subplots(2, 2, figsize=(10, 10))

# SVR (Poly) subplot
axes[0, 0].scatter(y_test, svr_poly_pred, alpha=0.5)
axes[0, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[0, 0].set_title("SVR (Poly)")
axes[0, 0].set_xlabel("True")
axes[0, 0].set_ylabel("Predicted")

# SVR (RBF) subplot
axes[0, 1].scatter(y_test, svr_rbf_pred, alpha=0.5)
axes[0, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[0, 1].set_title("SVR (RBF)")
axes[0, 1].set_xlabel("True")
axes[0, 1].set_ylabel("Predicted")

# Gradient Boosting subplot
axes[1, 0].scatter(y_test, gb_pred, alpha=0.5)
axes[1, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[1, 0].set_title("Gradient Boosting")
axes[1, 0].set_xlabel("True")
axes[1, 0].set_ylabel("Predicted")

# Random Forest subplot
axes[1, 1].scatter(y_test, rf_pred, alpha=0.5)
axes[1, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[1, 1].set_title("Random Forest")
axes[1, 1].set_xlabel("True")
axes[1, 1].set_ylabel("Predicted")

plt.tight_layout()
plt.show()
```



Gradient Boosting model seems to be the best option for this dataset, points lie most on the line witch fewer outliers comparing to Random Forest model

In [ ]:

Question 2

We choose "KID", "LEASE", "WELL", and "FIELD" attributes to label and distinguish the wells because they provide unique and relevant information about each well, making it easier to identify and compare their production performance. Here's a brief explanation of each attribute:

"KID" (Kansas Identification number): This is a unique identifier assigned to each well in Kansas. It serves as a primary key in the dataset and ensures that each well can be distinctly identified, avoiding any confusion or ambiguity.

"LEASE": The lease name represents the legal agreement under which a well is operated. It often corresponds to a specific area or group of wells that belong to a particular operator. Using the lease name helps in understanding the well's context, location, and ownership, which may be relevant when comparing wells and analyzing production data.

"WELL": The well number is another unique identifier for each well within a lease. It is usually assigned sequentially as wells are drilled in a lease. Using the well number in conjunction with the lease name provides a more granular level of identification and can help in tracking individual well performance over time.

"FIELD": The field name refers to the geographic area or reservoir where a group of wells are located. It provides a broader context for the well's location and can be useful when comparing wells that are situated in different fields. Fields often have specific geological characteristics that can impact production, so including the field name can help in understanding the factors that contribute to the performance of the highest producing wells.

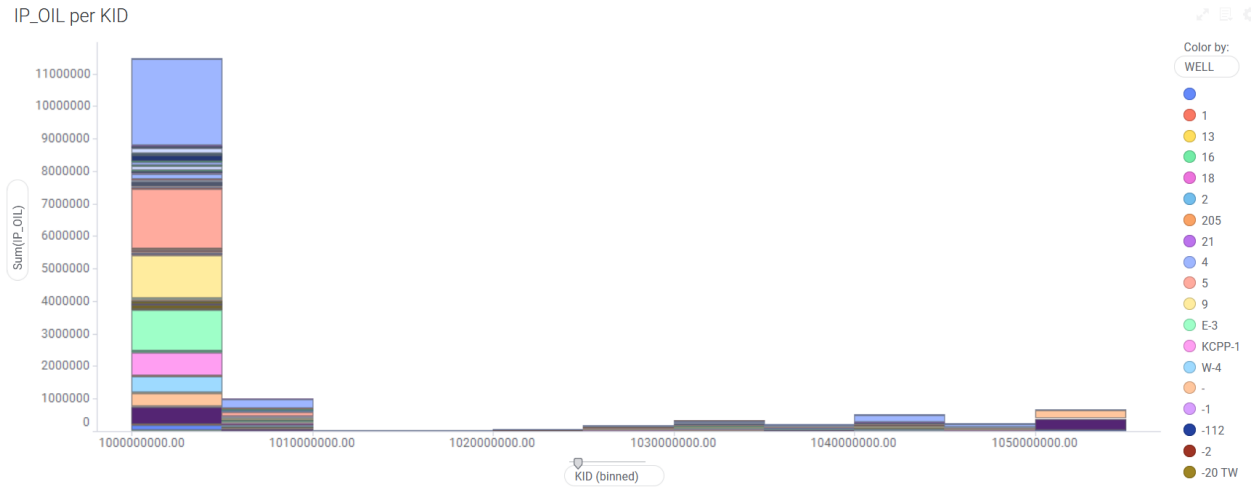


Figure 1: IP_OIL per KID with WELL

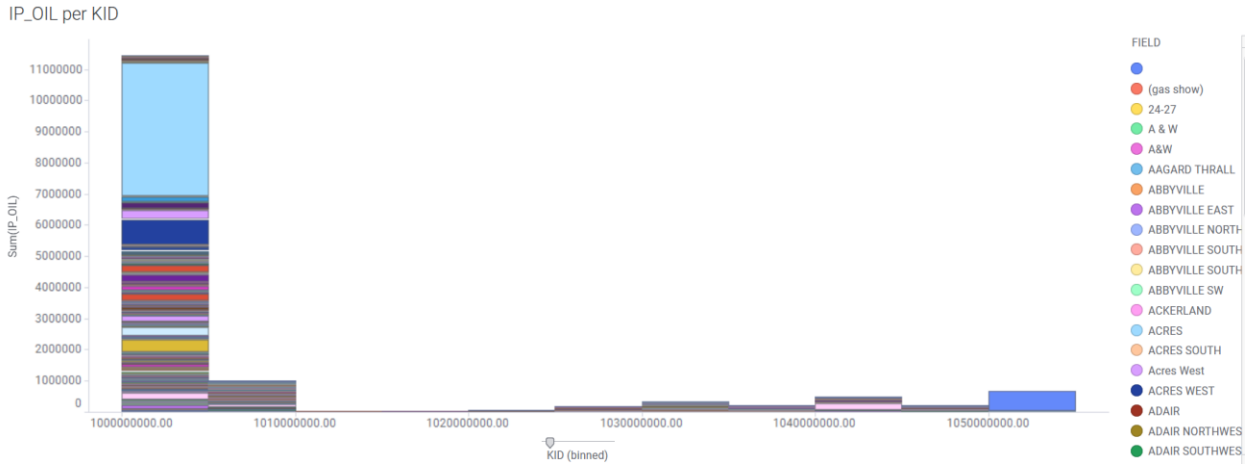


Figure 2:IP_OIL per KID with FIELD

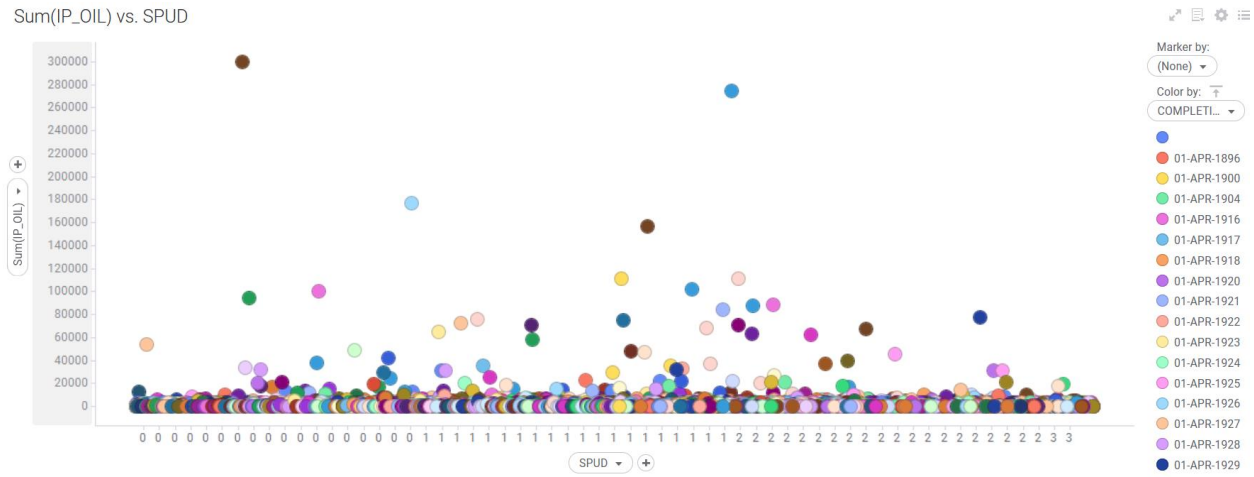


Figure 3: IP_OIL vs SPUD DATE with COMPLETETION DATE

The scatterplot of spud date vs. IP oil shows a visible trend or relationship between the two variables. It is essential to note that other factors not considered in this analysis may influence the relationship between spud date and IP oil, such as changes in drilling technology or geological conditions. While there may be some correlation between the depth of the well and the initial production of oil, this relationship may be good enough to make conclusive predictions. It is possible that other factors, such as geological formations, completion techniques, or well type, could have a more significant influence on the initial production of oil. The completion date vs. IP oil scatterplot does not show a clear trend or relationship between the two variables. This suggests that the completion date may not be a strong predictor of the initial production of oil for the wells in this dataset.

Visualization type:

- Scatterplot matrix: A scatterplot matrix can be used to visualize relationships between multiple attributes. For instance, we could plot attributes like IP_OIL, IP_GAS, and IP_WATER against each other to identify any potential correlations or trends.
- Geographic map: Plot the LATITUDE and LONGITUDE attributes on a map to visualize the spatial distribution of wells. We can color-code the markers based on well characteristics such as production rates, formation, or well status.

- Heatmap: Create a heatmap to display relationships between categorical variables such as formations and well statuses. This will provide insights into which formations are more likely to have certain well statuses or other categorical attributes.
- Line chart: Plot time-series data such as SPUD, COMPLETION, and PLUGGING dates to visualize trends in well drilling, completion, and plugging over time. This can help identify periods with increased or decreased activity.
- Stacked bar chart: Use a stacked bar chart to visualize the contribution of different formations or well types (horizontal vs. vertical) to the total initial production (IP) for a specific period or region.
- Parallel coordinates plot: This type of plot can be used to visualize multivariate data and identify patterns across multiple attributes. For example, we could plot well depth, elevation, IP_OIL, IP_GAS, and IP_WATER on parallel axes to investigate relationships between these attributes.
- Box plots: Create box plots for continuous attributes like depth, elevation, and IP values to compare the distribution and dispersion of these variables across different formations or well statuses.

Question 4

The oil and gas industry is witnessing a digital transformation, with data analytics playing a crucial role in optimizing operations, reducing costs, and enhancing production efficiency. This document outlines the strategies for recruiting an ideal data analytics team, the role of petroleum engineers, project prioritization and selection, solution deployment, digital literacy assessment, emerging technology team setup, and change management integration.

Recruitment Strategies for the Ideal Data Analytics Team:

- Define clear roles and responsibilities, including data scientists, data engineers, petroleum engineers, and project managers.
- Seek candidates with strong domain knowledge and expertise in oil and gas, along with a proven track record in data analytics.

- Evaluate candidates based on technical skills, problem-solving abilities, and communication skills to ensure effective collaboration.
- Offer competitive compensation packages and foster a culture of innovation, continuous learning, and growth.

Role of Petroleum Engineer:

- Petroleum engineers play a vital role in providing domain-specific knowledge and expertise in data analytics teams. They collaborate with data scientists and other technical experts to translate complex industry challenges into data-driven solutions. Petroleum engineers help identify relevant data sources, understand data quality, and interpret analytical results to make informed decisions.

Project Prioritization and Selection:

- Align projects with the company's strategic objectives and identify potential value or impact on the business.
- Assess feasibility, required resources, and risks associated with each project.
- Prioritize projects based on potential ROI, alignment with company goals, and resource availability.

Solution Deployment:

- Develop a comprehensive deployment plan, including timeline, resource allocation, and stakeholder engagement.
- Collaborate with IT and operations teams to integrate the solution into existing systems and processes.

- Provide training and support to end-users, ensuring a smooth transition and successful adoption.

Digital Literacy Assessment:

- Conduct a skills assessment to identify gaps in digital literacy within the workforce.
- Develop tailored training programs to improve employees' digital competencies.
- Foster a culture of continuous learning and adaptability, ensuring the workforce remains up to date with emerging technologies.

Emerging Technology Team Setup:

- Assemble a cross-functional team of experts, including data scientists, engineers, and industry specialists.
- Define clear roles and responsibilities for each team member.
- Encourage collaboration, experimentation, and innovation within the team.

Change Management Integration:

- Engage stakeholders early in the project to ensure their buy-in and support.
- Communicate the benefits and expected outcomes of the project to all employees.
- Provide training and support to help employees adapt to new processes and technologies.
- Monitor and measure the success of the change initiative, adjusting the approach as needed.

Creating an ideal data analytics team for an oil and gas company requires a comprehensive approach, including recruiting the right mix of skills, effective project prioritization, and a focus on change management. By integrating petroleum engineers and fostering a culture of innovation and continuous learning, the team will be well-equipped to deliver data-driven solutions that drive business growth and operational efficiency.