In [1]:
```
# Importation des bibliothèques nécessaires
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# 1. Charger les données à partir d'un fichier Excel
fichier_excel = 'BAISSER LE RENDEMENT.xlsx'
data = pd.read_excel(fichier_excel)
```

In [3]:
```
data.head()  # Affiche les 5 premières lignes du DataFrame
```

Out[3]:

	Date	CP DU GOM	FLASH DU KEROSENE	PC DU KEROSENE	Densité de Fond C01	Rendement du Bleed
0	2024-01-07	-16.0	27.0	-74.0	799.724410	48.142800
1	2024-01-08	-16.0	28.0	-74.0	782.912548	26.299940
2	2024-01-09	-34.0	21.0	-74.0	758.117468	28.945648
3	2024-01-10	5.0	48.0	-59.0	739.248178	11.380420
4	2024-01-11	2.0	22.0	-74.0	727.942915	78.500000

In [4]:
```
# Vérifier combien de valeurs manquantes il y a dans chaque colonne
print(data.isnull().sum())
```

```
Date                  0
CP DU GOM             5
FLASH DU KEROSENE     7
PC DU KEROSENE        6
Densité de Fond C01   0
Rendement du Bleed    0
dtype: int64
```

In [5]:
```
# Supprimer les lignes contenant des valeurs manquantes
data = data.dropna()
```

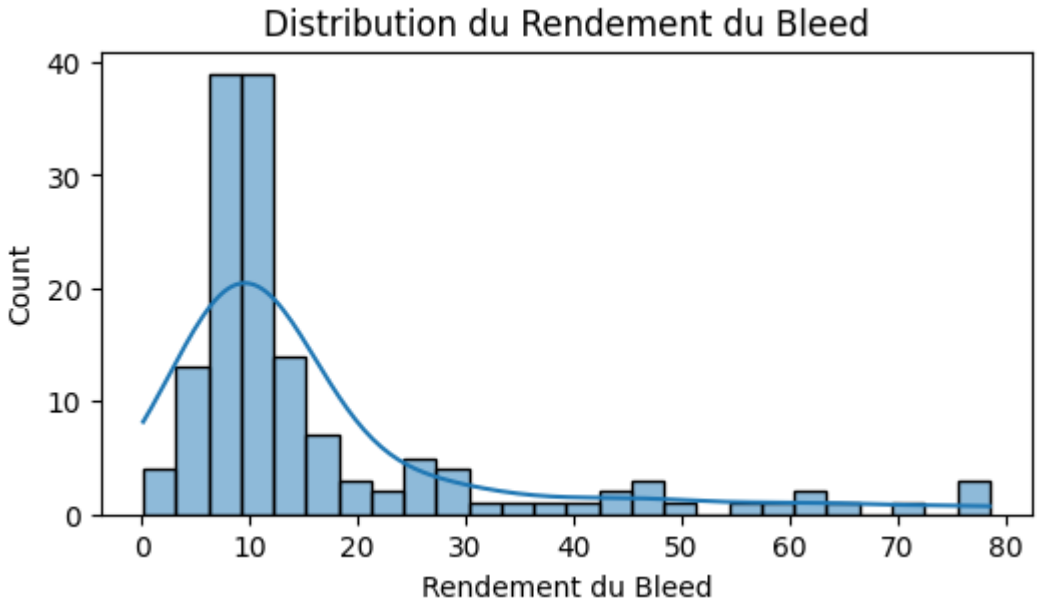
In [6]:
```
data['Rendement du Bleed'].describe()
```

Out[6]:
```
count    149.000000
mean      16.568020
std       16.362182
min        0.125682
25%        7.759413
50%       10.286098
75%       15.904781
max       78.500000
Name: Rendement du Bleed, dtype: float64
```

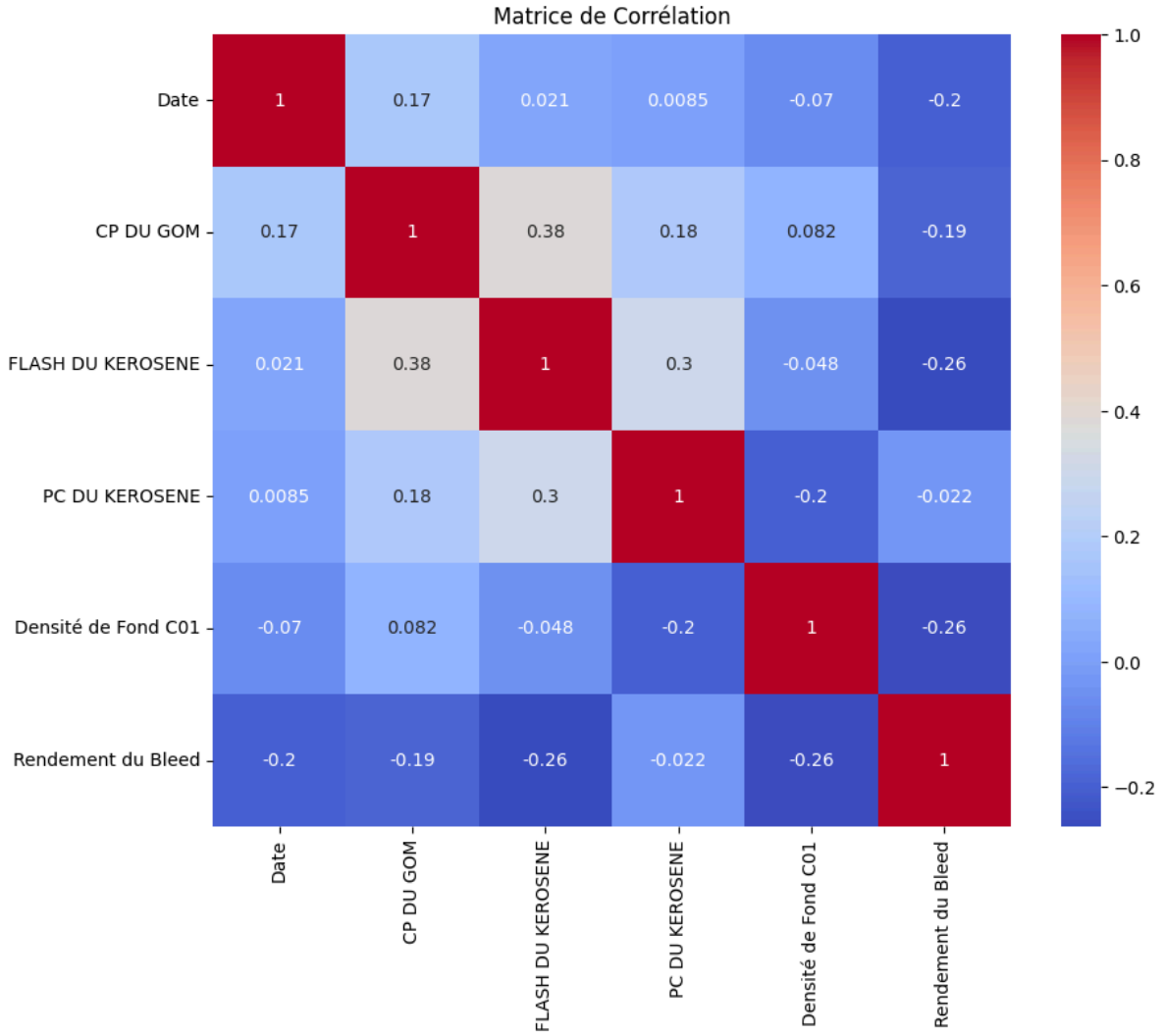
In [7]:
```
# Vérifier combien de valeurs manquantes il y a dans chaque colonne
print(data.isnull().sum())
```

```
Date                  0
CP DU GOM             0
FLASH DU KEROSENE     0
PC DU KEROSENE        0
Densité de Fond C01   0
Rendement du Bleed    0
dtype: int64
```

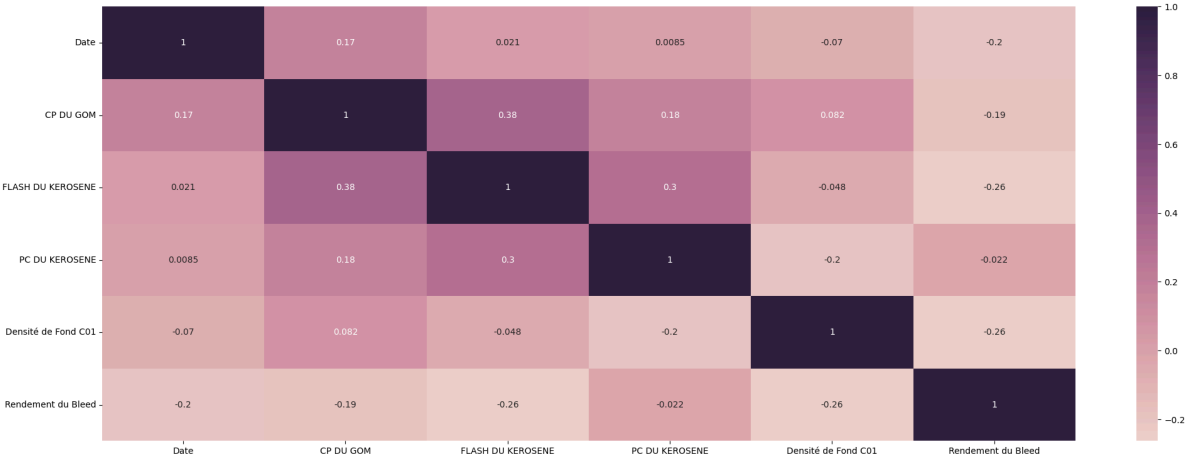
In [8]:
```
 #3. Analyse exploratoire des données
# Distribution des variables
plt.figure(figsize=(6, 3))
sns.histplot(data['Rendement du Bleed'], kde=True)
plt.title('Distribution du Rendement du Bleed')
plt.show()
```



In [9]:
```
# Matrice de corrélation
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Matrice de Corrélation')
plt.show()
```



In [18]:
```
#Correlation entre les variables quantitaitves
plt.figure(figsize=(25, 9))
ax=sns.heatmap(data.corr(), annot=True, cmap=sns.cubehelix_palette(as_cmap=True))
```



4. Modélisation

In [19]:
```
# Séparation des variables indépendantes et dépendantes
```

In [20]:
```
#Récuperer la variable des segments
y = data['Rendement du Bleed']
#Supprimer les variables inutiles
x=data.drop(columns=['Rendement du Bleed','Date'], axis=1)
```

```
# Séparer les données en jeu d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(x, y, random_state = 0,test_siz
```

4.1. Régression linéaire multiple

In [21]:
```
# 4.1. Régression linéaire multiple
lin_reg = LinearRegression()
```

In [22]:
```
X_train.isnull().sum()
```

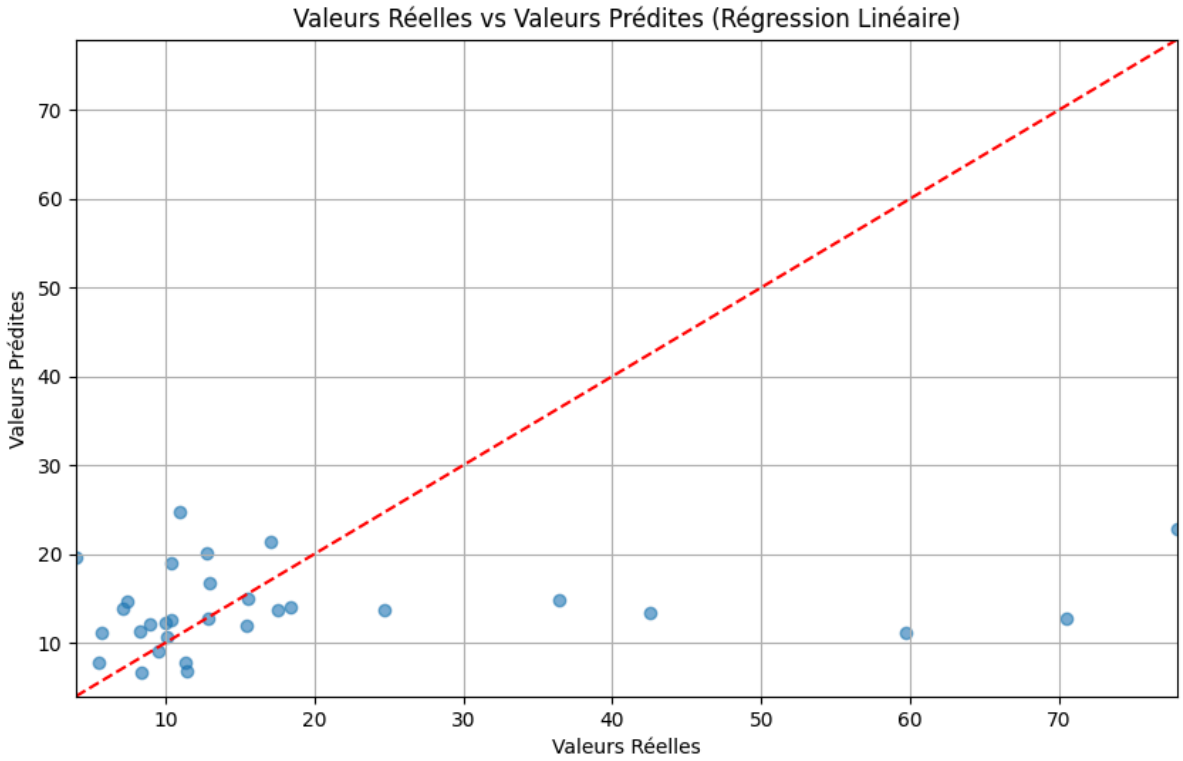
Out[22]:
```
CP DU GOM              0
FLASH DU KEROSENE      0
PC DU KEROSENE         0
Densité de Fond C01    0
dtype: int64
```

In [23]:
```
lin_reg.fit(X_train, y_train)
```

Out[23]:
```
LinearRegression()
```

In [24]:
```
y_pred_lin = lin_reg.predict(X_test)
mse_lin = mean_squared_error(y_test, y_pred_lin)
```

In [33]:
```
# Créer un graphique de dispersion des valeurs réelles vs prédites
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_lin, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', l
plt.title('Valeurs Réelles vs Valeurs Prédites (Régression Linéaire)')
plt.xlabel('Valeurs Réelles')
plt.ylabel('Valeurs Prédites')
plt.grid()
plt.xlim(y_test.min(), y_test.max())
plt.ylim(y_test.min(), y_test.max())
plt.show()

# Affichage de l'erreur quadratique moyenne
print(f"Mean Squared Error: {mse_lin}")
```




```
Mean Squared Error: 366.79242422567665
```

4.2. Régression polynomiale

In [25]:
```
# 4.2. Régression polynomiale
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y_train)
y_pred_poly = poly_reg.predict(poly.transform(X_test))
mse_poly = mean_squared_error(y_test, y_pred_poly)
```

In [36]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Supposons que X_train et y_train contiennent une seule caractéristique
# Transformer les données d'entraînement pour la régression polynomiale
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)

# Ajuster le modèle de régression polynomiale
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y_train)

# Prédire les valeurs pour les données de test
y_pred_poly = poly_reg.predict(poly.transform(X_test))
mse_poly = mean_squared_error(y_test, y_pred_poly)

# Pour tracer la courbe de régression, nous allons créer une plage de valeurs pour
X_grid = np.linspace(X_train.min(), X_train.max(), 100).reshape(-1, 1)  # Créer un
X_grid_poly = poly.transform(X_grid)  # Transformer les valeurs

# Prédictions pour la courbe
y_grid_pred = poly_reg.predict(X_grid_poly)
```

```
# Créer le graphique
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Valeurs Réelles', alpha=0.6)  # Va
plt.plot(X_grid, y_grid_pred, color='red', label='Régression Polynomiale', linewidt
plt.title('Régression Polynomiale (degré 2)')
plt.xlabel('Variable Indépendante')
plt.ylabel('Variable Dépendante')
plt.legend()
plt.grid()
plt.show()

# Affichage de l'erreur quadratique moyenne
print(f"Mean Squared Error: {mse_poly}")
```

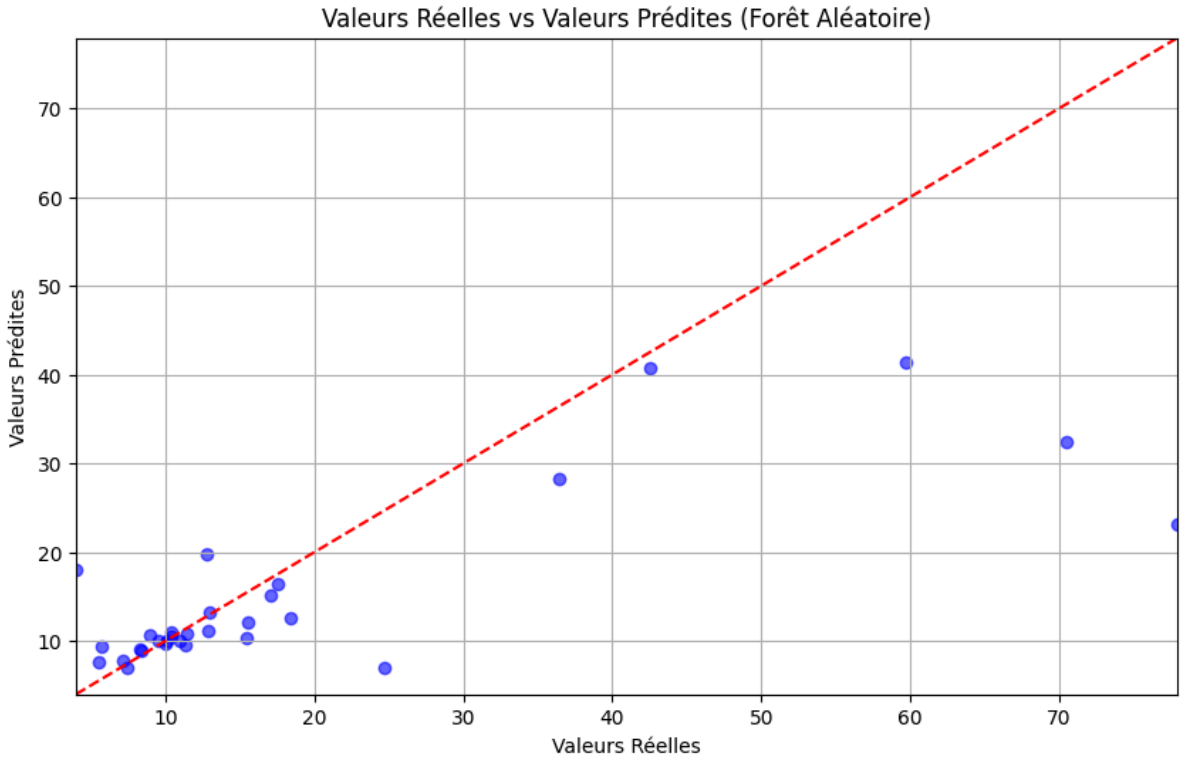
```
C:\Users\odeto\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2k
fra8p0\LocalCache\local-packages\Python312\site-packages\sklearn\base.py:493: User
Warning: X does not have valid feature names, but PolynomialFeatures was fitted wi
th feature names
  warnings.warn(
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 22
     20 # Pour tracer la courbe de régression, nous allons créer une plage de vale
urs pour X
     21 X_grid = np.linspace(X_train.min(), X_train.max(), 100).reshape(-1, 1)  #
Créer un vecteur de 100 points
---> 22 X_grid_poly = poly.transform(X_grid)  # Transformer les valeurs
     24 # Prédictions pour la courbe
     25 y_grid_pred = poly_reg.predict(X_grid_poly)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\L
ocalCache\local-packages\Python312\site-packages\sklearn\utils\_set_output.py:313,
in _wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    311 @wraps(f)
    312 def wrapped(self, X, *args, **kwargs):
--> 313     data_to_wrap = f(self, X, *args, **kwargs)
    314     if isinstance(data_to_wrap, tuple):
    315         # only wrap the first output for cross decomposition
    316         return_tuple = (
    317             _wrap_data_with_container(method, data_to_wrap[0], X, self),
    318             *data_to_wrap[1:],
    319         )

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\L
ocalCache\local-packages\Python312\site-packages\sklearn\preprocessing\_polynomia
l.py:433, in PolynomialFeatures.transform(self, X)
    403 """Transform data to polynomial features.
    404
    405 Parameters
   (...)
    429     `csr_matrix`.
    430 """
    431 check_is_fitted(self)
--> 433 X = self._validate_data(
    434     X, order="F", dtype=FLOAT_DTYPES, reset=False, accept_sparse=("csr",
"csc")
    435 )
    437 n_samples, n_features = X.shape
    438 max_int32 = np.iinfo(np.int32).max

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\L
ocalCache\local-packages\Python312\site-packages\sklearn\base.py:654, in BaseEstim
ator._validate_data(self, X, y, reset, validate_separately, cast_to_ndarray, **che
ck_params)
    651     out = X, y
    653 if not no_val_X and check_params.get("ensure_2d", True):
--> 654     self._check_n_features(X, reset=reset)
    656 return out

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\L
ocalCache\local-packages\Python312\site-packages\sklearn\base.py:443, in BaseEstim
ator._check_n_features(self, X, reset)
    440     return
    442 if n_features != self.n_features_in_:
--> 443     raise ValueError(
    444         f"X has {n_features} features, but {self.__class__.__name__} "
    445         f"is expecting {self.n_features_in_} features as input."
    446     )

ValueError: X has 1 features, but PolynomialFeatures is expecting 4 features as in
put.
```

4.3. Forêt Aléatoire

In [26]:
```
# 4.3. Forêt Aléatoire
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
```

In [37]:
```
# Créer le graphique de dispersion des valeurs réelles vs prédites
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_rf, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', l
plt.title('Valeurs Réelles vs Valeurs Prédites (Forêt Aléatoire)')
plt.xlabel('Valeurs Réelles')
plt.ylabel('Valeurs Prédites')
plt.grid()
plt.xlim(y_test.min(), y_test.max())
plt.ylim(y_test.min(), y_test.max())
plt.show()

# Affichage de l'erreur quadratique moyenne
print(f"Mean Squared Error: {mse_rf}")
```



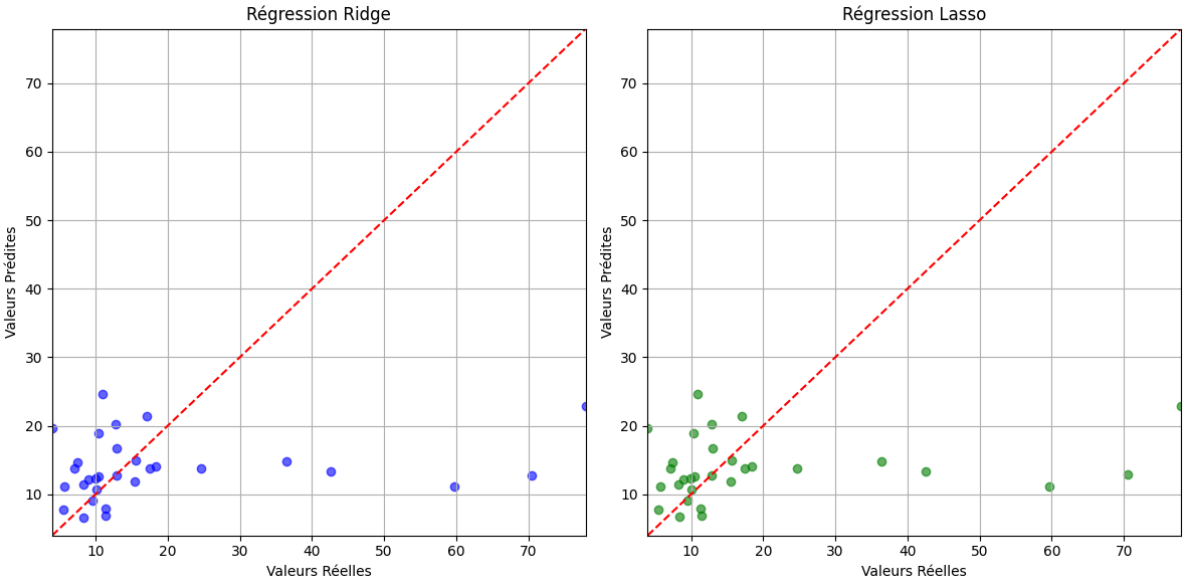

```
Mean Squared Error: 184.36951062803206
```

4.4. Régression Ridge et Lasso

In [27]:
```
# 4.4. Régression Ridge et Lasso
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, y_train)
y_pred_ridge = ridge_reg.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
```

In [28]:
```
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, y_train)
y_pred_lasso = lasso_reg.predict(X_test)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
```

In [38]:
```
# Créer le graphique de dispersion des valeurs réelles vs prédites
plt.figure(figsize=(12, 6))

# Graphique pour la régression Ridge
plt.subplot(1, 2, 1)  # 1 ligne, 2 colonnes, premier sous-graphique
plt.scatter(y_test, y_pred_ridge, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', l
plt.title('Régression Ridge')
plt.xlabel('Valeurs Réelles')
plt.ylabel('Valeurs Prédites')
plt.grid()
plt.xlim(y_test.min(), y_test.max())
plt.ylim(y_test.min(), y_test.max())

# Graphique pour la régression Lasso
plt.subplot(1, 2, 2)  # 1 ligne, 2 colonnes, deuxième sous-graphique
plt.scatter(y_test, y_pred_lasso, color='green', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', l
plt.title('Régression Lasso')
plt.xlabel('Valeurs Réelles')
plt.ylabel('Valeurs Prédites')
plt.grid()
plt.xlim(y_test.min(), y_test.max())
plt.ylim(y_test.min(), y_test.max())

plt.tight_layout()  # Ajuste les sous-graphiques pour éviter le chevauchement
plt.show()

# Affichage des erreurs quadratiques moyennes
print(f"Mean Squared Error (Ridge): {mse_ridge}")
print(f"Mean Squared Error (Lasso): {mse_lasso}")
```

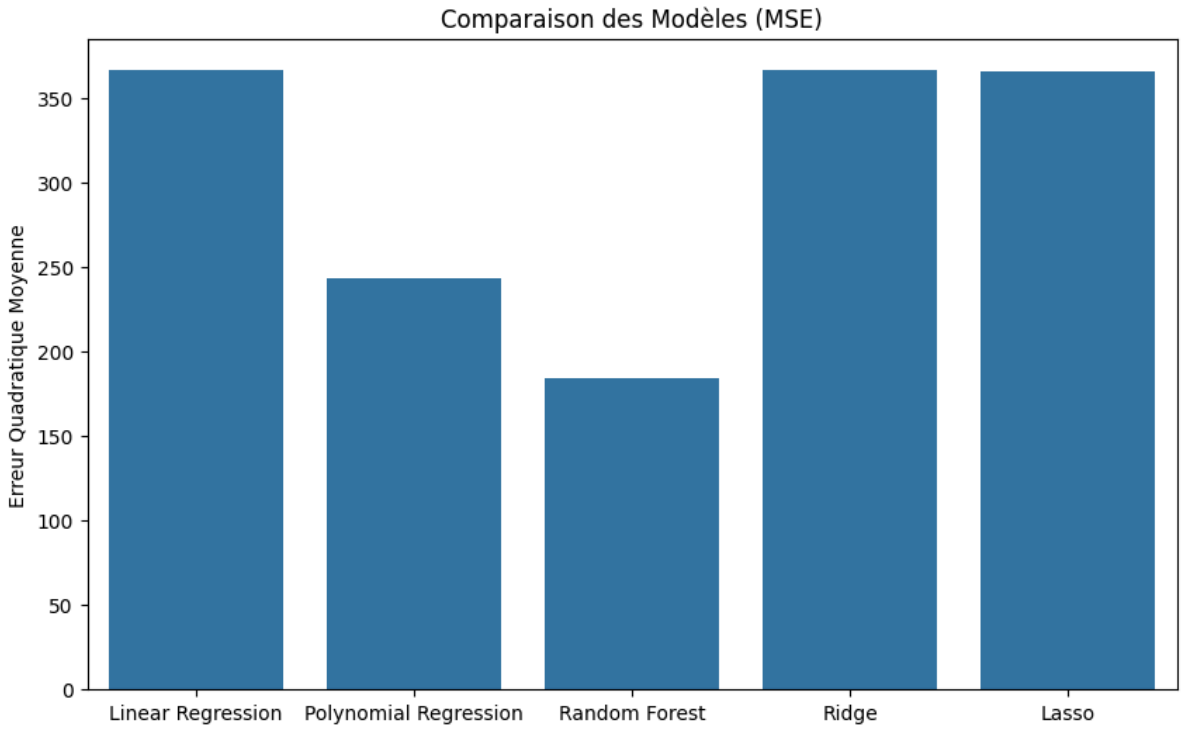



```
Mean Squared Error (Ridge): 366.78832533422485
Mean Squared Error (Lasso): 366.49857253764293
```

5. Comparaison des MSE (erreur quadratique moyenne)

In [29]:
```
# 5. Comparaison des MSE (erreur quadratique moyenne)
models = ['Linear Regression', 'Polynomial Regression', 'Random Forest', 'Ridge', '
mse_values = [mse_lin, mse_poly, mse_rf, mse_ridge, mse_lasso]
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=mse_values)
plt.title('Comparaison des Modèles (MSE)')
plt.ylabel('Erreur Quadratique Moyenne')
plt.show()
```



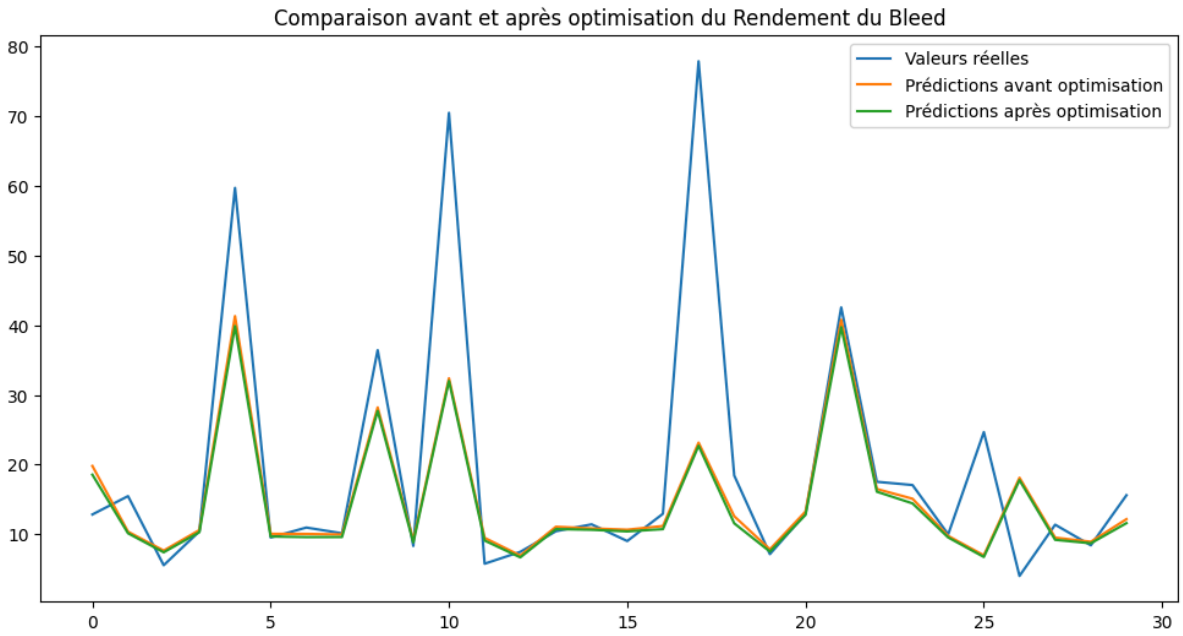
6. Optimisation du Rendement

Ajuster les variables pour réduire le rendement de 3 % (via Random Forest)

In [30]:
```
# Ajuster les variables pour réduire le rendement de 3 % (via Random Forest)
optimized_rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
optimized_rf_reg.fit(X_train, y_train * 0.97)  # Réduire la cible de 3 %
y_pred_optimized = optimized_rf_reg.predict(X_test)
```

Comparer avant et après optimisation

In [31]:
```
# Comparer avant et après optimisation
plt.figure(figsize=(12, 6))
plt.plot(y_test.values, label='Valeurs réelles')
plt.plot(y_pred_rf, label='Prédictions avant optimisation')
plt.plot(y_pred_optimized, label='Prédictions après optimisation')
plt.legend()
plt.title('Comparaison avant et après optimisation du Rendement du Bleed')
plt.show()
```



Affichage des MSE avant et après optimisation

In [32]:
```
# Affichage des MSE avant et après optimisation
mse_optimized = mean_squared_error(y_test, y_pred_optimized)
print(f"MSE avant optimisation (Random Forest): {mse_rf}")
print(f"MSE après optimisation (Random Forest): {mse_optimized}")
```

```
MSE avant optimisation (Random Forest): 184.36951062803206
MSE après optimisation (Random Forest): 189.4709702413517
```

In [ ]: