```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from statsmodels.regression.quantile_regression import QuantReg

data = pd.read_csv('/mnt/data/50_Startups (1).csv')


X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state ↵
=42)

results = {}

1. Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)

results['Linear Regression'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_linear)),
    'R2 Score': r2_score(y_test, y_pred_linear)
}

2. Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)

results['Ridge Regression'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_ridge)),
    'R2 Score': r2_score(y_test, y_pred_ridge)
}


3. Lasso Regression
lasso_model = Lasso(alpha=0.01)
lasso_model.fit(X_train, y_train)
y_pred_lasso = lasso_model.predict(X_test)

results['Lasso Regression'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_lasso)),
    'R2 Score': r2_score(y_test, y_pred_lasso)
}


4. Polynomial Regression (Degree 2)
poly_features = PolynomialFeatures(degree=2)
```

```
X_poly_train = poly_features.fit_transform(X_train)
X_poly_test = poly_features.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_poly_train, y_train)
y_pred_poly = poly_model.predict(X_poly_test)

results['Polynomial Regression (Degree 2)'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_poly)),
    'R2 Score': r2_score(y_test, y_pred_poly)
}

5. Support Vector Regression (SVR)
scaler = StandardScaler()
X_scaled_train = scaler.fit_transform(X_train)
X_scaled_test = scaler.transform(X_test)

svr_model = SVR(kernel='linear')
svr_model.fit(X_scaled_train, y_train)
y_pred_svr = svr_model.predict(X_scaled_test)

results['Support Vector Regression (SVR)'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_svr)),
    'R2 Score': r2_score(y_test, y_pred_svr)
}

6. Decision Tree Regression
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)

results['Decision Tree Regression'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_tree)),
    'R2 Score': r2_score(y_test, y_pred_tree)
}

7. Random Forest Regression
rf_model = RandomForestRegressor(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

results['Random Forest Regression'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_rf)),
    'R2 Score': r2_score(y_test, y_pred_rf)
}

8. Gradient Boosting Regression
gboost_model = GradientBoostingRegressor(random_state=42)
gboost_model.fit(X_train, y_train)
y_pred_gboost = gboost_model.predict(X_test)

results['Gradient Boosting Regression'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_gboost)),
    'R2 Score': r2_score(y_test, y_pred_gboost)
}

9. Quantile Regression (50th percentile)
```

```
quant_model = QuantReg(y_train, X_train)
quant_fit = quant_model.fit(q=0.5)
y_pred_quant = quant_fit.predict(X_test)

results['Quantile Regression (50th percentile)'] = {
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_quant)),
    'R2 Score': r2_score(y_test, y_pred_quant)
}
#finding best model
best_model = None
best_rmse = float('inf')
best_r2 = float('-inf')

for model_name, metrics in results.items():
    rmse = metrics['RMSE']
    r2 = metrics['R2 Score']

    if rmse < best_rmse:
        best_rmse = rmse
        best_model = model_name

    if r2 > best_r2:
        best_r2 = r2
        best_model_r2 = model_name

print(f"Best Model based on RMSE: {best_model} with RMSE: {best_rmse}")
print(f"Best Model based on R² Score: {best_model_r2} with R² Score: {best_r2}")

#for data interpretation

import matplotlib.pyplot as plt
import numpy as np
model_names = list(results.keys())
rmses = [metrics['RMSE'] for metrics in results.values()]
r2_scores = [metrics['R2 Score'] for metrics in results.values()]
fig, ax1 = plt.subplots(figsize=(12, 6))
color = 'tab:red'
ax1.set_xlabel('Models')
ax1.set_ylabel('RMSE', color=color)
ax1.bar(model_names, rmses, color=color, alpha=0.6, label='RMSE')
ax1.tick_params(axis='y', labelcolor=color)
ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('R² Score', color=color)
ax2.plot(model_names, r2_scores, color=color, marker='o', label='R² Score')
ax2.tick_params(axis='y', labelcolor=color)
plt.title('Model Performance: RMSE and R² Score Comparison')
fig.tight_layout()
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
plt.xticks(rotation=45)
plt.show()
```