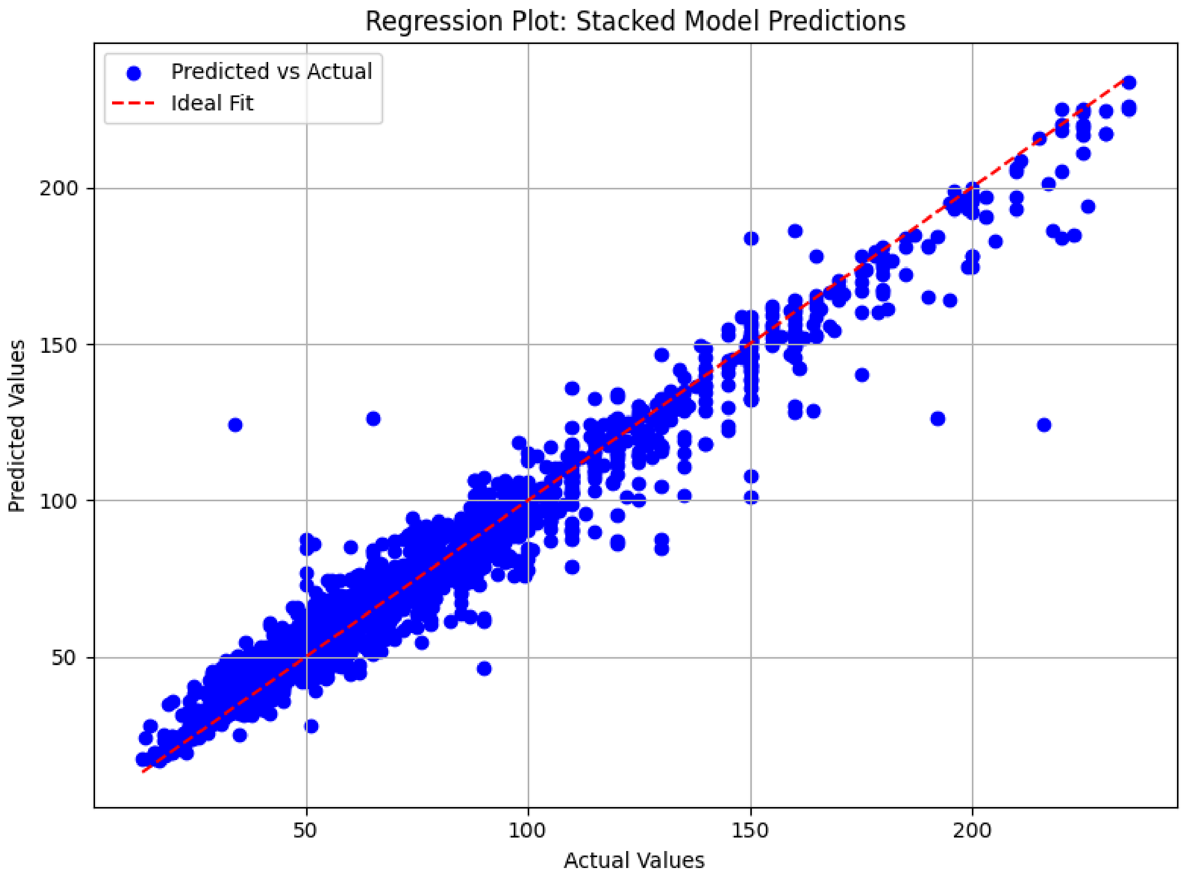
In [48]:
```
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split, cross_val_score
import numpy as np

# Splitting data into dependent and independent features
# 'X' contains all features except 'price', which is our target variable 'y'
X = data.drop('price', axis=1)
y = data['price']

# Splitting the dataset into training and testing sets
# Using 80% of the data for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=0)

# One-Hot Encoding for categorical features
# This step transforms 'area_type' and 'location' columns into one-hot encoded
format
ohe = OneHotEncoder()
column_trans = make_column_transformer(
    (OneHotEncoder(), ['area_type', 'location']),
    remainder='passthrough'
)

# Define a dictionary of models to evaluate
# Including Linear Regression, Decision Tree, Random Forest, and XGBoost
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(random_state=0),
    'XGBoost': XGBRegressor(random_state=0)
}

# Iterate through each model, train it, and evaluate its performance
for name, model in models.items():
    # Create a pipeline that applies column transformations and then fits the
model
    pipe = make_pipeline(column_trans, model)

    # Fit the model on the training data
    pipe.fit(X_train, y_train)

    # Predict on the test data
    y_pred = pipe.predict(X_test)

    # Calculate the R-squared score to evaluate model performance
    score = r2_score(y_test, y_pred)

    # Print the model name and its R-squared score
    print(f"{name} R-squared: {score}")
```

```
# Now we will train our meta-model using predictions from the base models
# Obtain predictions from the base models for the test set
y_pred_lr = models['Linear Regression'].predict(column_trans.transform(X_tes
t))
y_pred_dt = models['Decision Tree'].predict(column_trans.transform(X_test))
y_pred_rf = models['Random Forest'].predict(column_trans.transform(X_test))

# Stack the predictions as new features for the meta-model
meta_features = np.column_stack((y_pred_lr, y_pred_dt, y_pred_rf))

# Train the meta-model using the stacked features
meta_model = XGBRegressor(random_state=0)
meta_model.fit(meta_features, y_test)

# Predict using the meta-model
y_pred_stacked = meta_model.predict(meta_features)

# Calculating and printing the R-squared score for the stacked model
r2_stacked = r2_score(y_test, y_pred_stacked)
print("Stacked Model R-squared:", r2_stacked)
```

```
Linear Regression R-squared: 0.6310029465726306
Decision Tree R-squared: 0.4506507028709156
Random Forest R-squared: 0.60284969197938
XGBoost R-squared: 0.6548128433746665
Stacked Model R-squared: 0.9541540795387651
```

In [49]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_stacked, color='blue', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--
', color='red', label='Ideal Fit')
plt.title('Regression Plot: Stacked Model Predictions')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [50]:
```
# Creating a pipeline that first applies column transformations and then fits the meta-model
# 'column_trans' will handle the One-Hot Encoding for categorical features
# 'meta_model' is the model that will be trained using the transformed data
pipe = make_pipeline(column_trans, meta_model)

# Fitting the pipeline on the training data
# This step first transforms the training data using 'column_trans' and then fits 'meta_model' to the transformed data
pipe.fit(X_train, y_train)
```

Out[50]:
```
Pipeline
(https://scikit-learn.org/1.4/modules/generated/sklearn
  columntransformer: ColumnTransformer
  (https://scikit-learn.org/1.4/modules/generated/sklearn.co
    onehotencoder          remainder
    OneHotEncoder          passthrough
    (https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.OneHotEncoder
  XGBRegressor
```

In [51]:
```
data.head(1)
```

Out[51]:

	area_type	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	Electronic City Phase II	2	1056.0	2.0	1.0	39.07

In [52]:
```
predicted_price=pipe.predict(pd.DataFrame([['Super built-up Area','Electronic City Phase II',2,105.6,2.0,1.0]],
                                          columns=['area_type','location','size','total_sqft','bath','balcony']))
```

In [53]:
```
predicted_price
```

Out[53]:
```
array([24.6143], dtype=float32)
```

In [281]:
```
import pickle
```

In [282]:
```
import pickle

# Saving the trained meta-model to a file
# 'model.pkl' is the file where the model will be saved
# 'wb' mode is used to write in binary format
with open('model.pkl', 'wb') as file:
    # 'pickle.dump' serializes the meta-model and saves it to the specified file
    pickle.dump(meta_model, file)
```