Loading libraries for the task.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler
import joblib
pd.set_option('display.max_columns', 81)
```

Reading train data and dropping unnecessary column Id.

In [2]:
```
train = pd.read_csv('train.csv') # read train data
train.drop(['Id'], axis=1, inplace= True) # drop Id column
train.head() # display head
```

Out[2]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norr
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norr
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norr
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norr
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norr

Reading test data and dropping unnecessary column Id.

In [3]:
```
test = pd.read_csv('test.csv') # read test data
test.drop(['Id'], axis=1, inplace= True) # drop Id column
test.head() # display head
```

Out[3]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition
0	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Feedr	Norr
1	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norr
2	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norr
3	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norr
4	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Inside	Gtl	StoneBr	Norm	Norr

Data Preprocessing: Dropping columns that have more than 15% of data missing.

In [4]:
```
missing_percentage = (train.isnull().sum() / len(train)) * 100 # find the percentage of missing values in data

columns_with_high_missing = missing_percentage[missing_percentage > 15].index.tolist() # find columns having more than 15% missi

train.drop(columns_with_high_missing,axis=1, inplace= True) # drop these columns from train data
test.drop(columns_with_high_missing,axis=1, inplace= True) # drop these columns from test data
```

Data Preprocessing:

- Imputing numerical values with the median
- Imputing categorical values with the mode

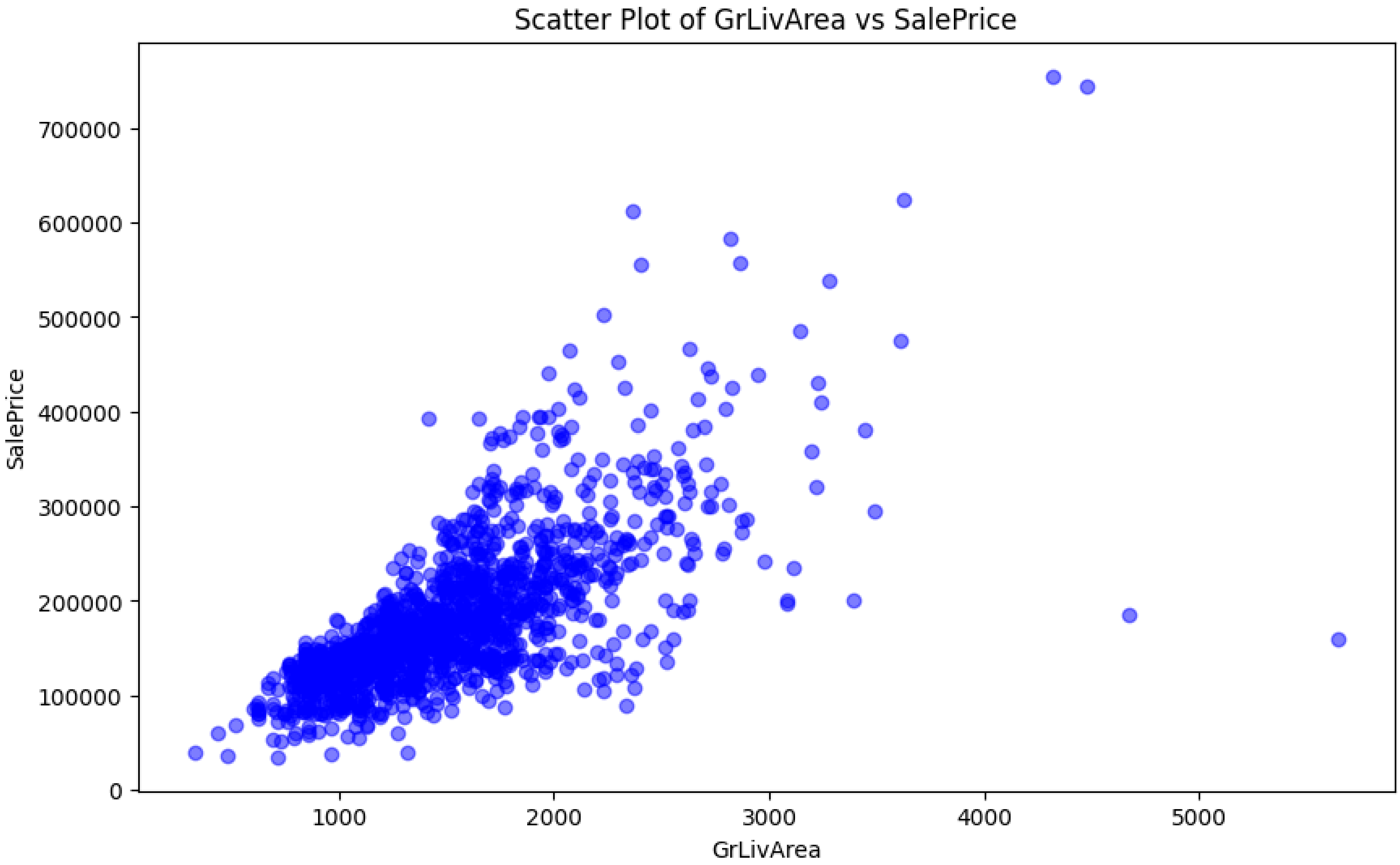
In [5]:

```
numeric_columns = train.select_dtypes(include='number').columns.drop('SalePrice') # select numerical features from data
train[numeric_columns] = train[numeric_columns].fillna(train[numeric_columns].median()) # impute train data with median
test[numeric_columns] = test[numeric_columns].fillna(test[numeric_columns].median()) # impute test data with median


categorical_columns = train.select_dtypes(include='object').columns # select categorical features from data
train[categorical_columns] = train[categorical_columns].fillna(train[categorical_columns].mode().iloc[0]) # impute train with mo
test[categorical_columns] = test[categorical_columns].fillna(test[categorical_columns].mode().iloc[0]) # impute test with mode
```

Exploratory Data Analysis: Plot of GrLivArea with SalePrice.

In [6]:
```
plt.figure(figsize=(10, 6))
plt.scatter(train['GrLivArea'], train['SalePrice'], alpha=0.5, color='b') # scatter plot of GrLivArea with SalePrice
plt.title('Scatter Plot of GrLivArea vs SalePrice')
plt.xlabel('GrLivArea')
plt.ylabel('SalePrice')
plt.show()
```



Two data points show an anomaly where the area is >4000 yet prices are low. We locate them and drop them from the data as they are outliers.

In [7]:
```
train[train['GrLivArea']>4000] ## find out the outlier points
```

Out[7]:

	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	Hou
523	60	RL	40094	Pave	IR1	Bnk	AllPub	Inside	Gtl	Edwards	PosN	PosN	1Fam	
691	60	RL	21535	Pave	IR1	Lvl	AllPub	Corner	Gtl	NoRidge	Norm	Norm	1Fam	
1182	60	RL	15623	Pave	IR1	Lvl	AllPub	Corner	Gtl	NoRidge	Norm	Norm	1Fam	
1298	60	RL	63887	Pave	IR3	Bnk	AllPub	Corner	Gtl	Edwards	Feedr	Norm	1Fam	

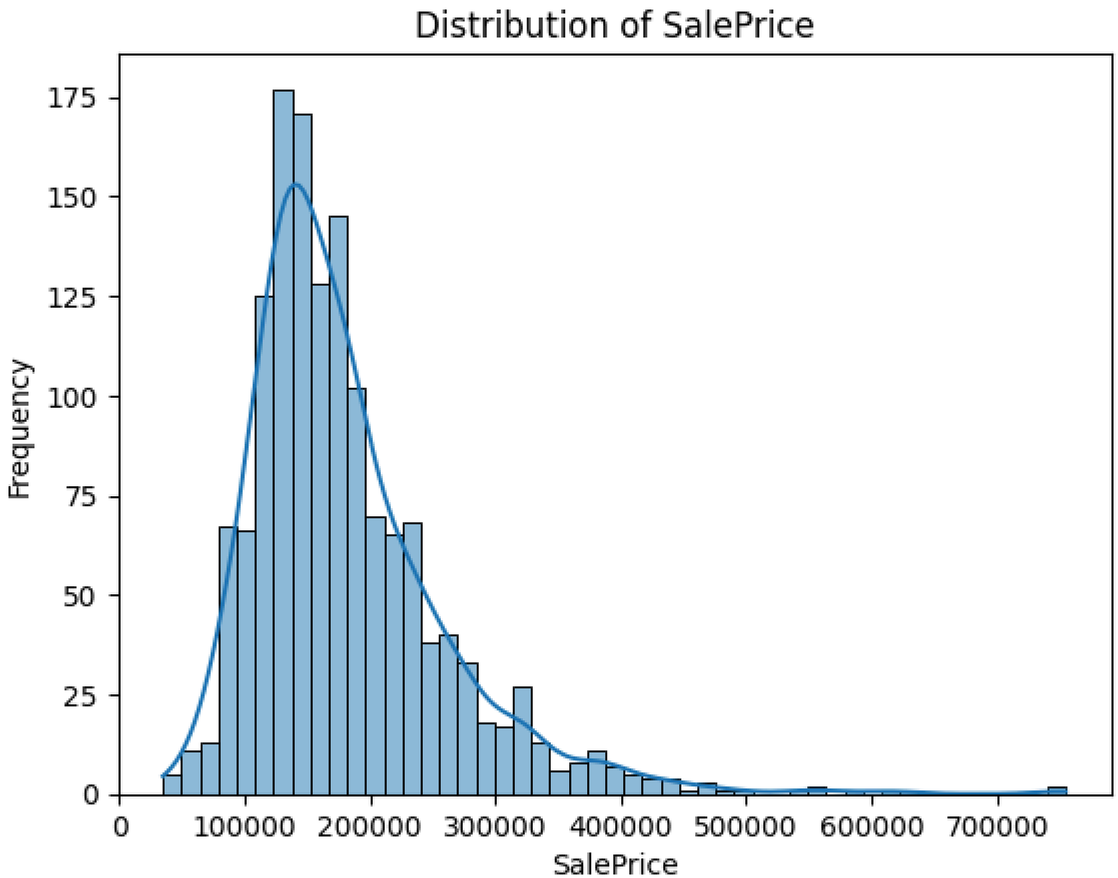
Remove outlier points

In [8]:
```
train = train.drop(index=[523, 1298]) # drop outlier points
```

Exploratory Data Analysis: Distribution of Sales Price.

In [9]:
```
sns.histplot(train['SalePrice'], kde=True)
plt.title('Distribution of SalePrice')
plt.xlabel('SalePrice')
plt.ylabel('Frequency')
plt.show()
```

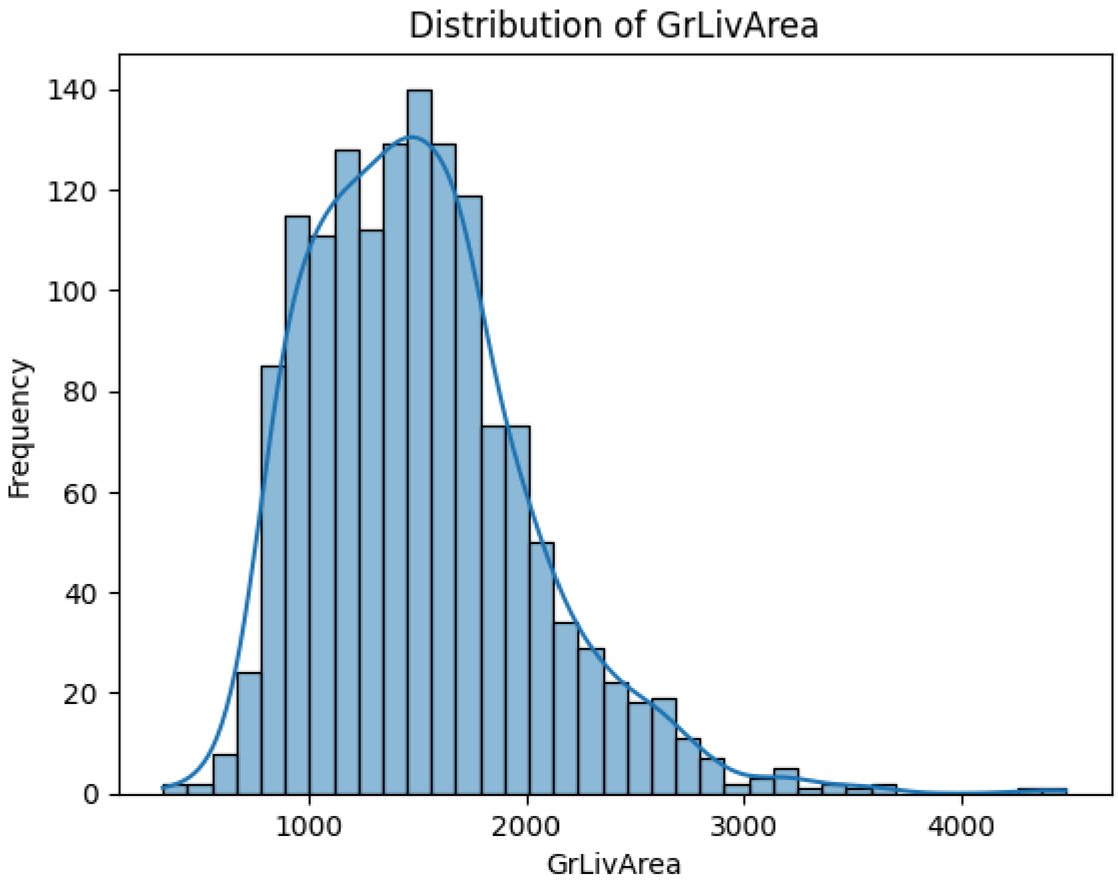
```
c:\Users\bashs\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na
option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Exploratory Data Analysis: Distribution of GrLivArea.

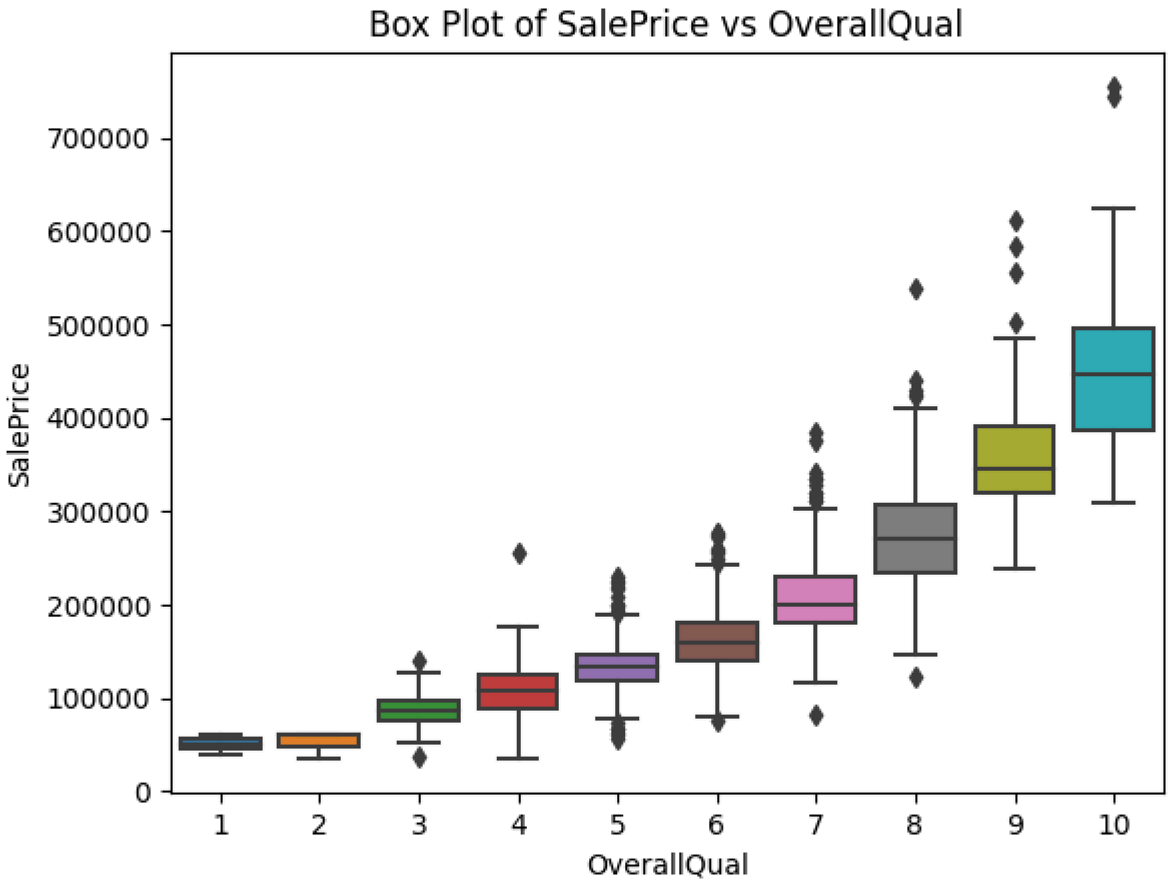
In [10]:
```
sns.histplot(train['GrLivArea'], kde=True)
plt.title('Distribution of GrLivArea')
plt.xlabel('GrLivArea')
plt.ylabel('Frequency')
plt.show()
```

```
c:\Users\bashs\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



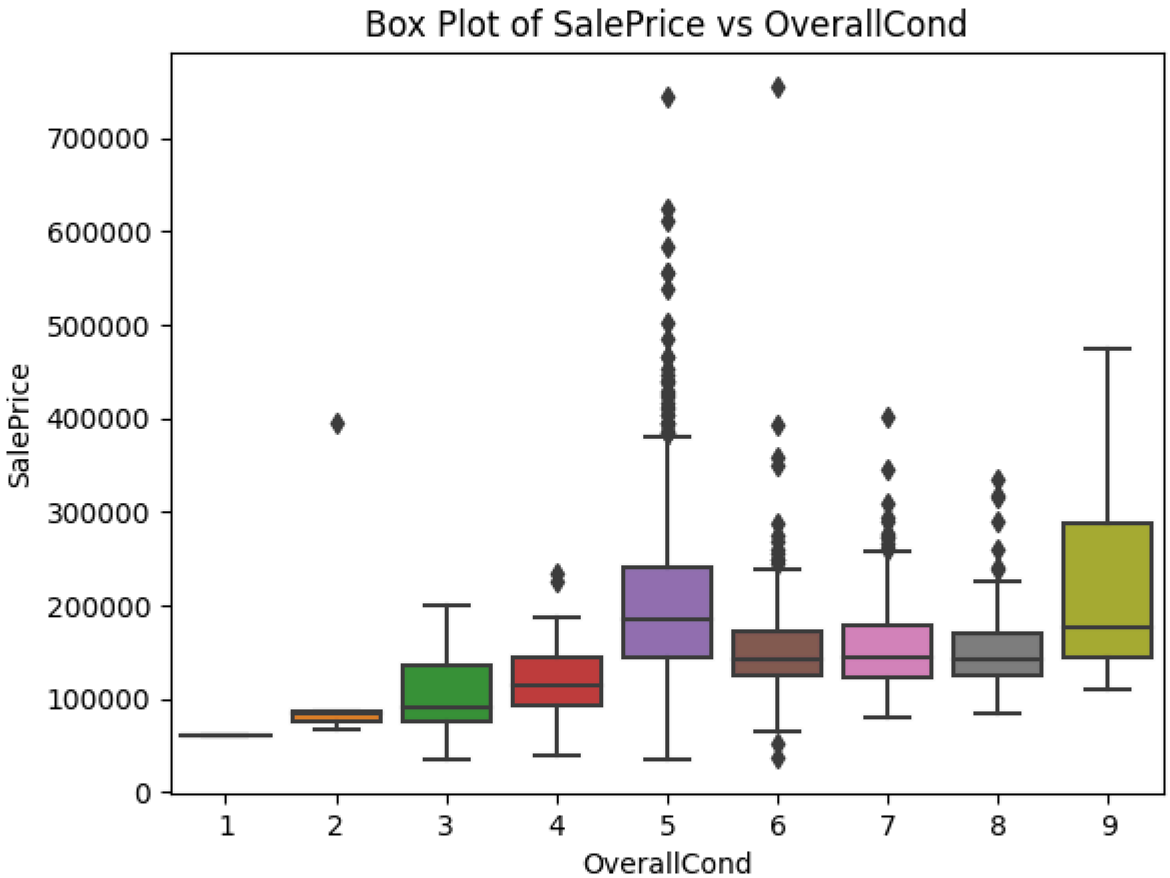
Exploratory Data Analysis: Box Plot of sales price with overall quality.

In [11]:
```
sns.boxplot(x='OverallQual', y='SalePrice', data=train)
plt.title('Box Plot of SalePrice vs OverallQual')
plt.show()
```



Exploratory Data Analysis: Box plot of sales price with overall condition.

In [12]:
```
sns.boxplot(x='OverallCond', y='SalePrice', data=train)
plt.title('Box Plot of SalePrice vs OverallCond')
plt.show()
```

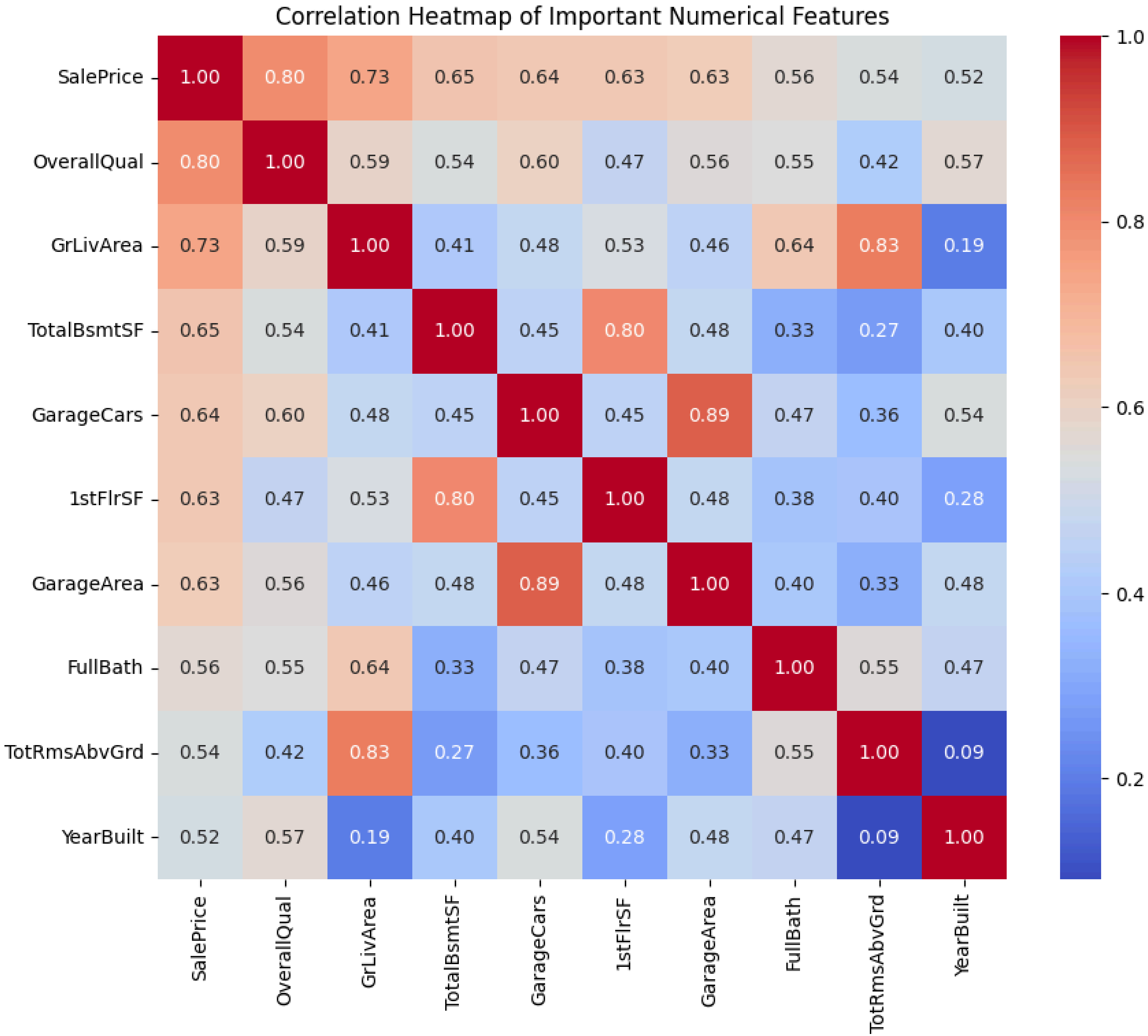


Exploratory Data Analysis: Heatmap of most correlated features.

In [13]:
```
numerical_features = train.select_dtypes(include=['int', 'float']) # select numerical features
important_numerical_features = numerical_features.corr()['SalePrice'].abs().nlargest(10).index # select 10 most correlated featu
correlation_matrix = train[important_numerical_features].corr()

## Plotting heatmap

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Important Numerical Features')
plt.show()
```


Correlation Heatmap of Important Numerical Features
SalePrice 1.00 0.80 0.73 0.65 0.64 0.63 0.63 0.56 0.54 0.52
OverallQual 0.80 1.00 0.59 0.54 0.60 0.47 0.56 0.55 0.42 0.57
GrLivArea 0.73 0.59 1.00 0.41 0.48 0.53 0.46 0.64 0.83 0.19
TotalBsmtSF 0.65 0.54 0.41 1.00 0.45 0.80 0.48 0.33 0.27 0.40
GarageCars 0.64 0.60 0.48 0.45 1.00 0.45 0.89 0.47 0.36 0.54
1stFlrSF 0.63 0.47 0.53 0.80 0.45 1.00 0.48 0.38 0.40 0.28
GarageArea 0.63 0.56 0.46 0.48 0.89 0.48 1.00 0.40 0.33 0.48
FullBath 0.56 0.55 0.64 0.33 0.47 0.38 0.40 1.00 0.55 0.47
TotRmsAbvGrd 0.54 0.42 0.83 0.27 0.36 0.40 0.33 0.55 1.00 0.09
YearBuilt 0.52 0.57 0.19 0.40 0.54 0.28 0.48 0.47 0.09 1.00
SalePrice
OverallQual
GrLivArea
TotalBsmtSF
GarageCars
1stFlrSF
GarageArea
FullBath
TotRmsAbvGrd
YearBuilt
1.0
0.8
0.6
0.4
0.2

Data Preprocessing: Label encoding categorical features

In [14]:
```
categorical_features = train.select_dtypes(include=['object']).columns # select categorical features

label_encoder = LabelEncoder()

# Fitting LabelEncoder on training data and transform both training and testing data
for feature in categorical_features:
    # Fit on training data
    label_encoder.fit(train[feature])

    # Transform training data
    train[feature] = label_encoder.transform(train[feature])

    # Transform testing data
    test[feature] = label_encoder.transform(test[feature])
```

Data Preprocessing: Scaling numerical features.

In [15]:
```
numerical_features = train.select_dtypes(include=['int', 'float']).columns # select numerical features


scaler = StandardScaler()

# Fitting StandardScaler on training data and transform both training and testing data
for feature in numerical_features:

    if feature not in ['SalePrice']:
        scaler.fit(train[[feature]])

    # Transform training data
        train[feature] = scaler.transform(train[[feature]])

    # Transform testing data
        test[feature] = scaler.transform(test[[feature]])
```

Defining the model with the parameter grids.

In [16]:
```
models = {
    "Linear Regression": (LinearRegression(), {}),
    "Random Forest": (RandomForestRegressor(), {'n_estimators': [100, 200, 300]}),
    "Gradient Boosting": (GradientBoostingRegressor(), {'n_estimators': [100, 200, 300], 'learning_rate': [0.1, 0.05, 0.01]})
}
```

Doing training and validation split and declaring variables for storing results.

In [17]:
```
test_result = test.copy()

# Splitting training set into training and validation sets with 80-20% ratio.
X_train, X_val, y_train, y_val = train_test_split(train.drop(columns=['SalePrice']), train['SalePrice'], test_size=0.2, random_s

## For storing results
results_dict = {'Model': [], 'RMSE': [], 'MAE': []}

test_errors_rmse = {}
test_errors_mae = {}
```

Performing the ML model training in the following steps:

- Looping each model to be trained
- Performing hyperparameter tuning using grid search with 5 fold CV
- Storing the best model and predicting on validation set.
- Storing the prediction for each model on the test data
- Evaluating the RMSE and MAE for each model

In [18]:

```python
# Looping each model top erform grid search and selecting best model
for model_name, (model, param_grid) in models.items():

    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)

    # Selecting best model
    best_model = grid_search.best_estimator_

    # Training best model on the entire training set
    best_model.fit(X_train, y_train)

    # Predicting on test set
    y_test_pred = best_model.predict(X_val)


    test_predictions = best_model.predict(test)
    test_result['SalesPrice_' + model_name] = test_predictions

    # Test set RMSE and MAE
    test_rmse = mean_squared_error(y_val, y_test_pred, squared=False)
    test_mae = mean_absolute_error(y_val, y_test_pred)

    # Storing test errors
    test_errors_rmse[model_name] = test_rmse
    test_errors_mae[model_name] = test_mae

    # Storing best model
    joblib.dump(best_model, 'best_model_{}.pkl'.format(model_name))

    results_dict['Model'].append(model_name)
    results_dict['RMSE'].append(test_rmse)
    results_dict['MAE'].append(test_mae)


test.to_csv('test_with_predictions.csv', index=False)
```

Displaying results data.

In [19]:
```
results_df = pd.DataFrame(results_dict)
results_df
```

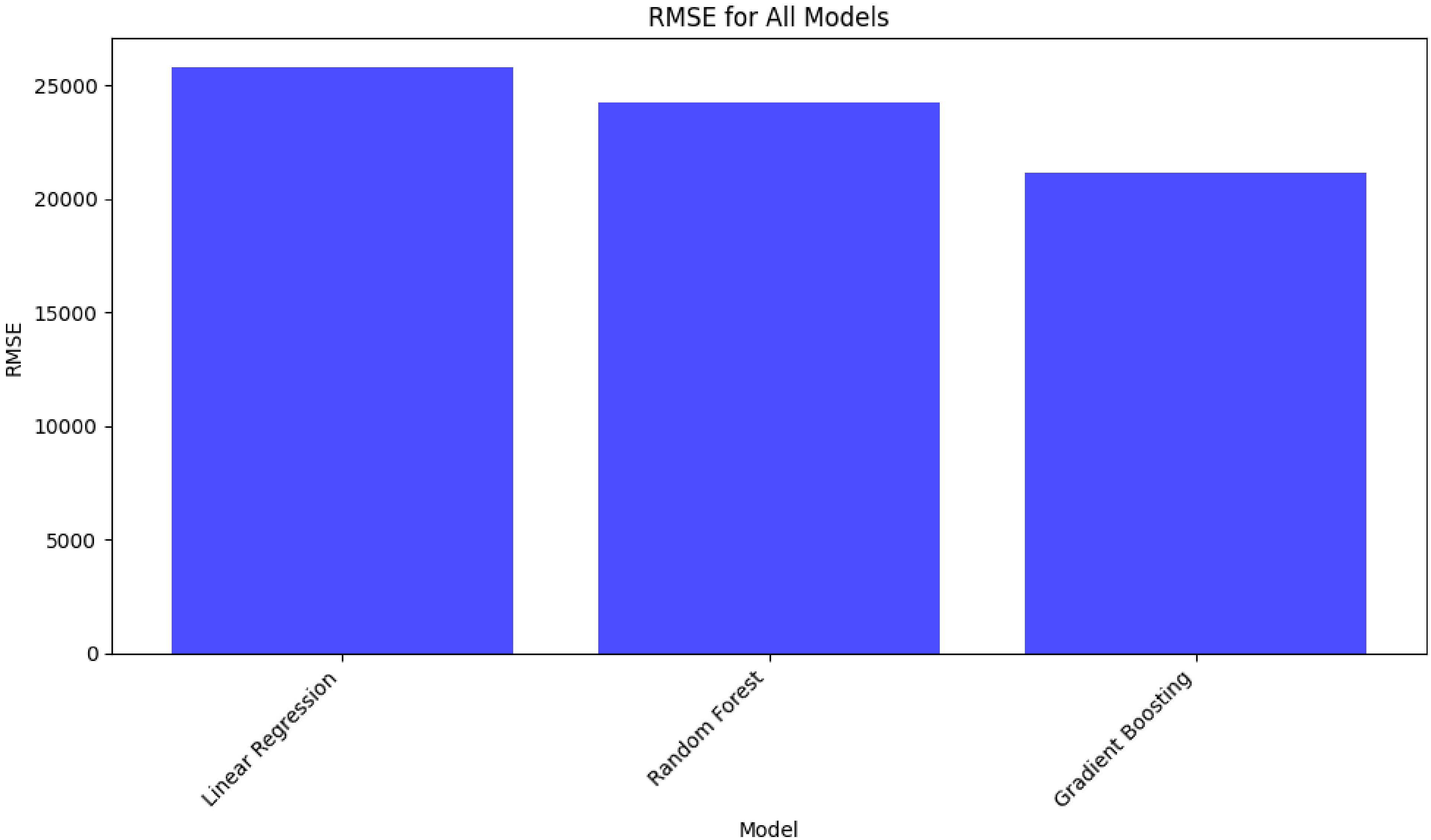
Out[19]:

	Model	RMSE	MAE
0	Linear Regression	25754.875952	18840.058246
1	Random Forest	24207.888585	16722.584760
2	Gradient Boosting	21155.814175	14925.770087

Plotting histogram of RMSE for all three models

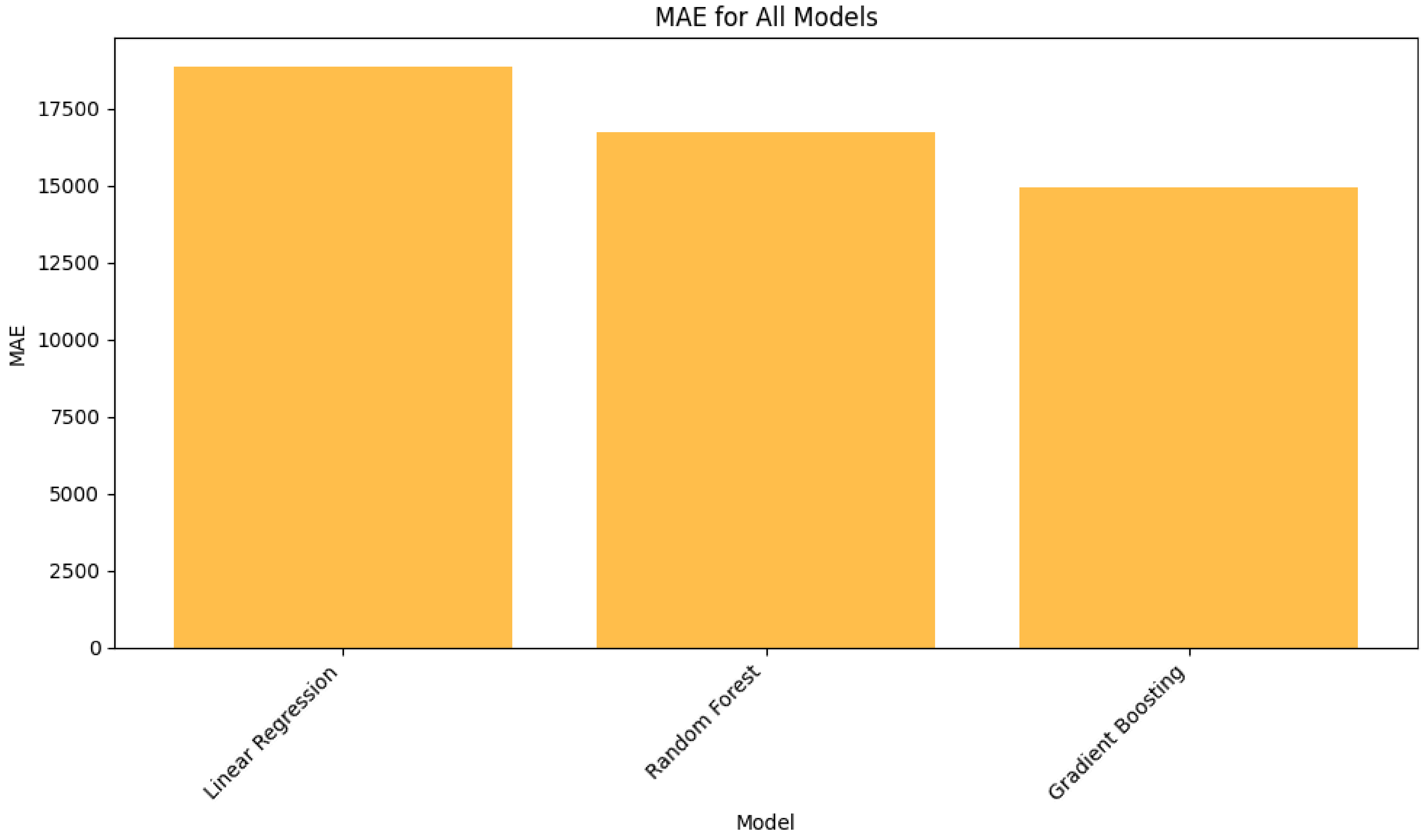
In [20]:

```
plt.figure(figsize=(10, 6))
plt.bar(results_df['Model'], results_df['RMSE'], color='blue', alpha=0.7)
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('RMSE for All Models')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```


RMSE for All Models
RMSE
0
5000
10000
15000
20000
25000
Linear Regression
Random Forest
Gradient Boosting
Model

Plotting histogram of MAE for all three models

In [21]:
```
plt.figure(figsize=(10, 6))
plt.bar(results_df['Model'], results_df['MAE'], color='orange', alpha=0.7)
plt.xlabel('Model')
plt.ylabel('MAE')
plt.title('MAE for All Models')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```


MAE for All Models
MAE
17500
15000
12500
10000
7500
5000
2500
0
Linear Regression
Random Forest
Gradient Boosting
Model