"King County Housing Price Prediction and Market Analysis for Optimal Real Estate Decision Making"

---

1 Business Problem

For a real estate investor, a critical decision is determining the appropriate price of a property before purchasing it. Investing too much on a property can lead to losses, while underestimating its value can result in missed opportunities. The business problem we aim to address is:

"How can we predict the selling price of a house in King County based on its features?"

By building a predictive model, we can provide the investor with an estimate of a house's price based on its characteristics. This model will aid in making informed decisions, maximizing returns, and minimizing risks.

Why a Linear Regression Model Adds Value:

A linear regression model is well-suited for this problem because it allows us to quantify the relationship between the various features and the sale price of houses. By analyzing the coefficients of the model, we can determine which features have the most significant impact on the sale price. This information is valuable for stakeholders to make data-driven decisions in the real estate market, optimize pricing strategies, and advise clients effectively.

---

2. Data Understanding

Overview of the King County House Sales Dataset

The King County House Sales dataset contains information about house sales in King County, Washington. It includes various features related to the houses and their sale details. The dataset is in tabular format, with each row representing a house sale record.

Important Columns and Features

1. **sqft_above and sqft_basement:**
    - The square footage of the house apart from the basement.
    - The square footage of the basement of the house.
2. **yr_renovated and yr_built:**
    - The year of the house's last renovation
    - Year when house was built
3. **lat and long:**

- Latitude and longitude coordinate of the house.

4. **waterfront and view:**
   - Whether the house is on a waterfront and the quality of view from it
5. **condition and grade:**
   - How good the overall condition of the house is. Related to maintenance of house and the Overall grade of the house. Related to the construction and design of the house.
6. **sqft_living15:**
   - The average square footage of the interior housing living space for the 15 nearest houses.
7. **sqft_lot15:**
   - The average square footage of land lots for the 15 nearest houses.
8. **Bedrooms**
   - Number of bedrooms.
9. **Bathrooms**
   - Number of bathrooms
10. **Sqft_Living and sqft_lot**
    - Square footage of the living space and the lot of the home
11. **Floors**
    - Number of floors (levels) in house

These features are crucial for modeling the house sale prices as they represent various aspects of the house such as size, location, and recent renovations. Utilizing these features, we aim to predict the sale prices of houses in King County.

Identifying potential challenges or peculiarities in the dataset is essential for understanding its limitations and potential biases. Here are some common challenges and peculiarities present in the King County House Sales dataset:

1. **Missing Values:**
   - One of the primary challenges could be missing values in certain columns (e.g., `yr_renovated`, `sqft_basement`). Addressing these missing values appropriately during data preparation is crucial for reliable modeling.
2. **Outliers:**
   - The presence of outliers in features such as square footage (`sqft_above`, `sqft_basement`, `sqft_living15`, `sqft_lot15`) or sale prices could affect the model's performance and should be handled appropriately.
3. **Data Imbalance:**
   - The dataset might have an imbalance in terms of the number of houses in different regions or with specific characteristics. This could potentially introduce bias in the model.
4. **Non-Numeric Data:**

- Converting non-numeric data (e.g., categorical variables) to a suitable format for modeling, such as one-hot encoding, might lead to a significant increase in the number of features and potential issues related to multicollinearity.

5. **Seasonality and Trends:**
   - The dataset might exhibit seasonality or trends over time that could affect house prices. It's important to account for these patterns during analysis and modeling.
6. **Geographical Variation:**
   - House prices might vary significantly based on the geographical location (latitude, longitude). This spatial variation could be challenging to capture accurately and might require advanced geospatial analysis.
7. **Heteroscedasticity:**
   - The assumption of constant variance (homoscedasticity) might not hold true for house prices. The variability of prices might change based on different features, which needs to be considered in the modeling process.
8. **Multicollinearity:**
   - Correlations and interdependencies among the independent variables (features) could lead to multicollinearity issues, affecting the stability and interpretation of the regression model.
9. **Inaccurate Data:**
   - Data errors or inaccuracies in house details, renovation years, or other features might exist, potentially affecting the model's reliability and performance.

.

3.Data Preparation

Data Preprocessing Steps

1. **Data Inspection:**
   - The `data.info()` and `data.describe()` functions are used to get an overview of the dataset, including data types and summary statistics.
2. **Column Inspection:**
   - A loop is set to iterate through each column in the dataset, displaying the count of unique values for each column.
3. **Column Selection and Dropping:**
   - Columns with the names 'id,' 'date,' and 'zipcode' are selected and dropped from the dataset using `data.drop()`. These columns were deemed irrelevant for modeling.
4. **Feature Engineering:**
   - Three columns, namely 'view,' 'waterfront,' and 'condition,' are being transformed:
     - 'view' is mapped to numerical values using a predefined mapping dictionary.

- 'waterfront' is mapped to binary values (0 or 1) based on whether it's 'YES' or 'NO.'
- 'condition' is mapped to numerical values using a predefined mapping dictionary.

- The 'grade' column is extracted and converted to integers.
- The 'sqft_basement' column is processed:
  - The numerical part is extracted from the string and converted to integers.
  - Any '?' values are replaced with NaN to indicate missing values.
  - Missing values are imputed with the mean of the column.

5. **Handling Missing Values:**

   - The code checks for missing values in the dataset using `data.isnull().sum()`.
   - Rows containing missing values are dropped from the dataset using `data.dropna()`.

6. **Sorting by Price:**

   - The dataset is sorted in descending order based on the 'price' column to display the top 10 highest-priced records.

In [20]:
```
#Importing dependacies and loading the dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("kc_house_data.csv")
data
```

Out[20]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	N
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	N
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	

21597 rows × 21 columns

In [21]:
```
#STEP 1;Data inspection

data.info()
data.describe()

#STEP 2;Column Inspection (Checking for Unique values)

for val in data:
   print(data[val].value_counts())
   print()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
```

In [22]:
```
#STEP 3;Column selection and dropping (irrelevant features)

data.columns
dCol = ['id', 'date', 'zipcode']
data.drop(dCol, axis=1, inplace=True)
data.head()
```

Out[22]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grad
0	221900.0	3	1.00	1180	5650	1.0	NaN	NONE	Average	Averag
1	538000.0	3	2.25	2570	7242	2.0	NO	NONE	Average	Averag
2	180000.0	2	1.00	770	10000	1.0	NO	NONE	Average	6 Lo Averag
3	604000.0	4	3.00	1960	5000	1.0	NO	NONE	Very Good	Averag
4	510000.0	3	2.00	1680	8080	1.0	NO	NONE	Average	8 Goo

In [23]:
```
#STEP 4;Feature Engineering

# Create a mapping dictionaries
#Waterfront
waterfront_mapping = {'NO': 0, 'YES': 1}
#View
view_mapping = {'NONE': 0,'FAIR': 1,'AVERAGE': 2,'GOOD': 3,'EXCELLENT': 4}
#Condition
condition_mapping = {'Poor': 1,'Fair': 2,'Good': 3,'Average': 4,'Very Good': 5}

# Use the map function to convert the columns
data['view'] = data['view'].map(view_mapping)

data['waterfront'] = data['waterfront'].map(waterfront_mapping)

data['condition'] = data['condition'].map(condition_mapping)

# Extract the numerical part and convert it to integers
data['grade'] = data['grade'].str.extract('(\d+)').astype(int)

#Processing  "sqft_basement" with 454 question marks (?) present along the columr
data['sqft_basement'] = pd.to_numeric(data['sqft_basement'], errors='coerce')

# Replacing "?" with NaN to indicate missing values
data['sqft_basement'] = data['sqft_basement'].replace('?', np.nan)

# Imputing missing values with the mean of the column
mean = data['sqft_basement'].mean()
data['sqft_basement'].fillna(mean, inplace=True)


# Display the first few rows to verify the changes
data[['waterfront', 'view', 'condition', 'grade']].head()
```

Out[23]:

	waterfront	view	condition	grade
0	NaN	0.0	4	7
1	0.0	0.0	4	7
2	0.0	0.0	4	6
3	0.0	0.0	5	7
4	0.0	0.0	4	8

In [24]:
```
#STEP 5;Handling missing values

data.isnull().sum()

data = data.dropna()

#STEP 6;Sorting data by price

data.sort_values("price", ascending = False)
```

Out[24]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
7245	7700000.0	6	8.00	12050	27600	2.5	0.0	3.0	3	
3910	7060000.0	5	4.50	10040	37325	2.0	1.0	2.0	4	
9245	6890000.0	6	7.75	9890	31374	2.0	0.0	4.0	4	
1446	5350000.0	5	5.00	8000	23985	2.0	0.0	4.0	4	
1162	5110000.0	5	5.25	8010	45517	2.0	1.0	4.0	4	
...	...	...	...	...	...	...	...	...	...	
16700	85000.0	2	1.00	910	9753	1.0	0.0	0.0	4	
3763	84000.0	2	1.00	700	20130	1.0	0.0	0.0	4	
18453	83000.0	2	1.00	900	8580	1.0	0.0	0.0	4	
2139	82500.0	2	1.00	520	22334	1.0	0.0	0.0	2	
8267	82000.0	3	1.00	860	10426	1.0	0.0	0.0	4	

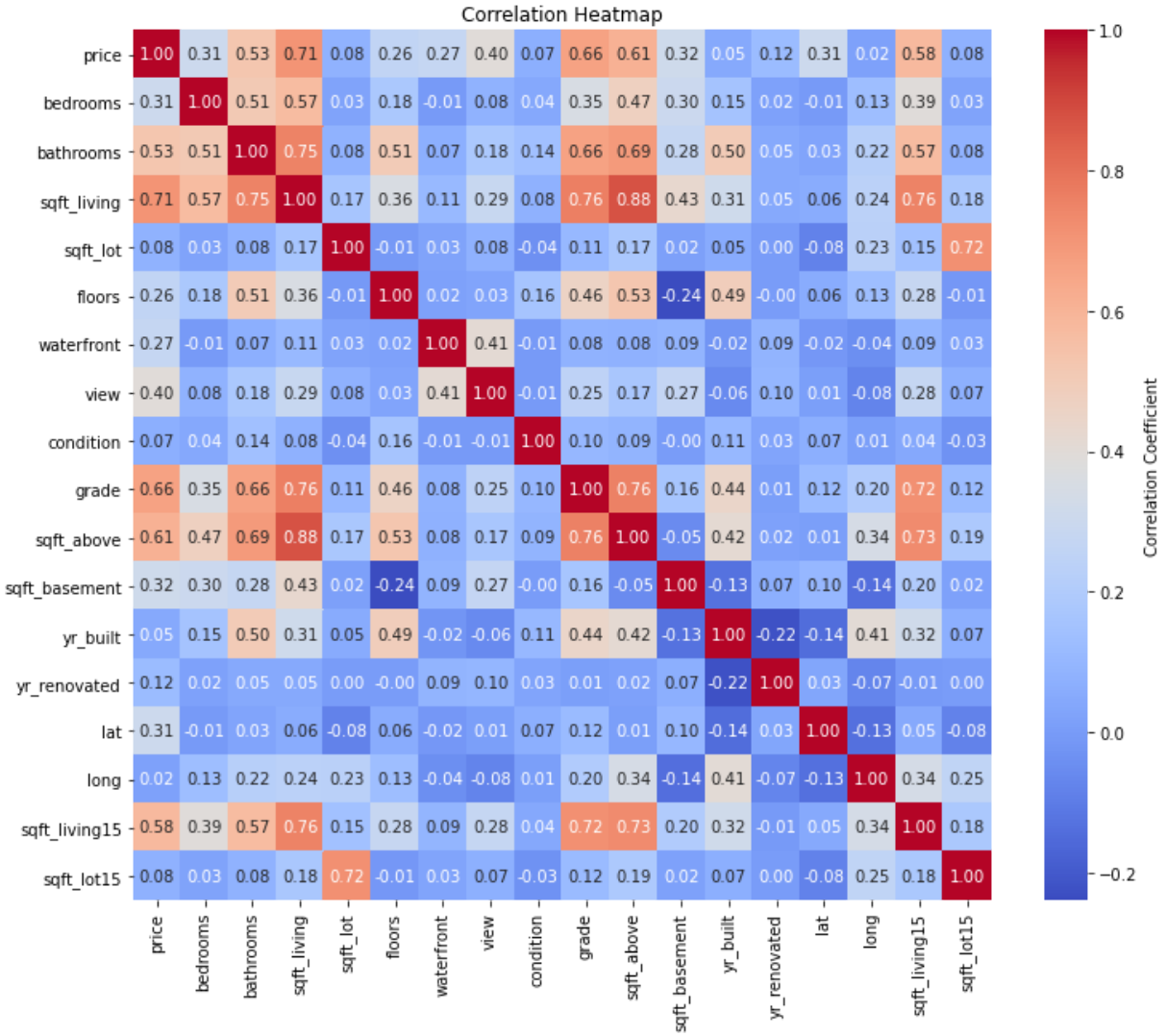
15762 rows × 18 columns

3.Visualizations

Let's now perform some exploratory data analysis (EDA) to understand how the features interact with the target variable "price". We'll visualize the relationship between some of the prominent features and the house price to get a clearer picture.

We'll start with a correlation heatmap to see which features are most correlated with the price.

In [25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix
corr = data.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Generate a heatmap
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", cbar_kws={'label': 'Cor

# Set the title
plt.title("Correlation Heatmap")

# Display the heatmap
plt.show()
```



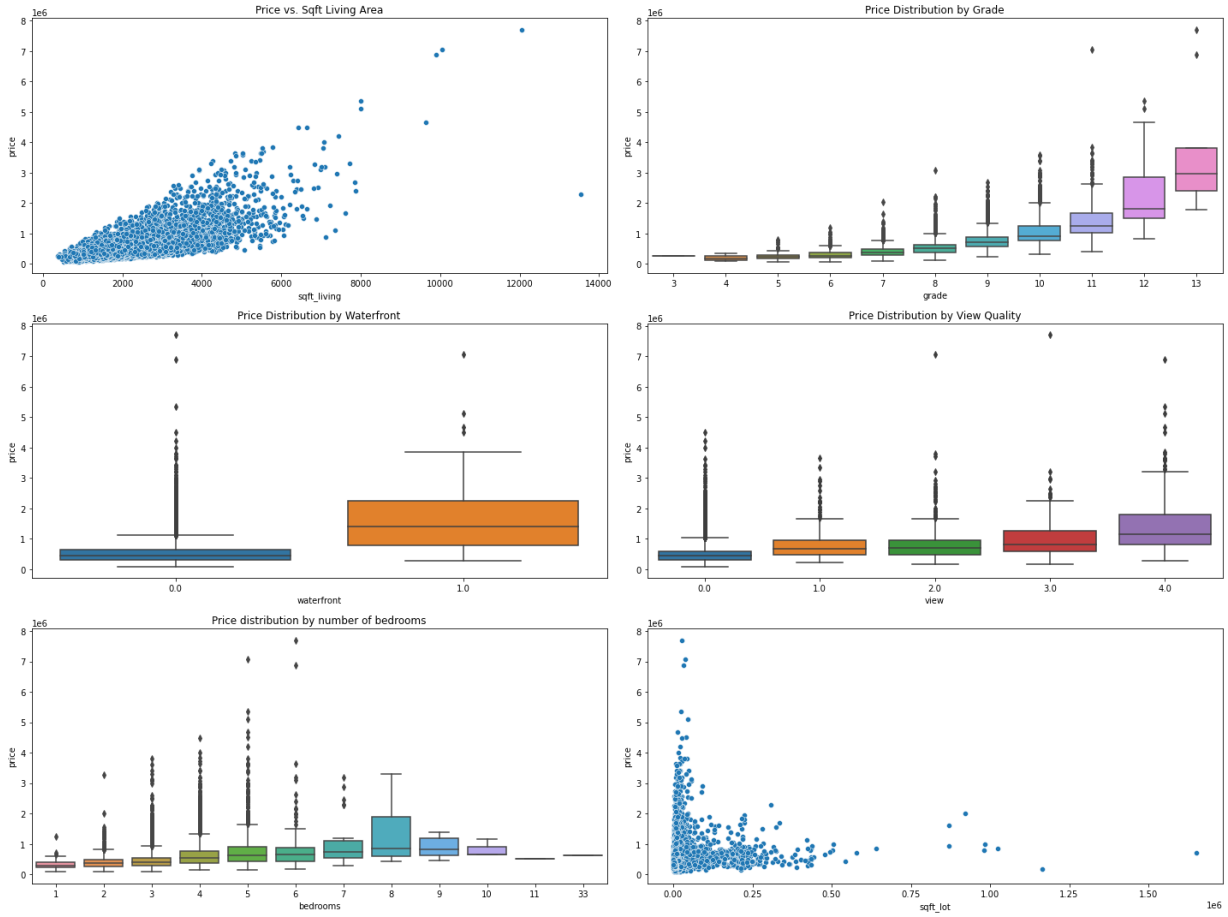
From the correlation heatmap, we can observe the following:

- sqft_living, grade, and sqft_above have strong positive correlations with the price. This indicates that as these features increase, the house price tends to increase.

- lat (latitude) also has a notable positive correlation with the price. This suggests that the location (north-south positioning) of the house plays a role in determining its price.
- Features like sqft_lot, condition, and yr_built have weaker correlations with the price.

To further visualize the relationships, let's plot scatter plots for sqft_living and boxplots for, waterfront, view, and grade against the price.

In [26]:
```
# Set up the figure and axes
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(20, 15))

# Scatter plot for 'sqft_living' vs 'price'
sns.scatterplot(x='sqft_living', y='price', data=data, ax=axes[0, 0])
axes[0, 0].set_title('Price vs. Sqft Living Area')

# Box plot for 'grade' vs 'price'
sns.boxplot(x='grade', y='price', data=data, ax=axes[0, 1])
axes[0, 1].set_title('Price Distribution by Grade')

# Box plot for 'waterfront' vs 'price'
sns.boxplot(x='waterfront', y='price', data=data, ax=axes[1, 0])
axes[1, 0].set_title('Price Distribution by Waterfront')

# Box plot for 'view' vs 'price'
sns.boxplot(x='view', y='price', data=data, ax=axes[1, 1])
axes[1, 1].set_title('Price Distribution by View Quality')

# Box plot for 'bedrooms' Vs 'price'
sns.boxplot(x='bedrooms',y='price',data=data, ax=axes[2, 0])
axes[2,0].set_title('Price distribution by number of bedrooms')

#scatter plot for 'sqft_lot' vs 'Price'
sns.scatterplot(x='sqft_lot',y='price',data=data, ax=axes[2, 1])


# Adjust the layout
plt.tight_layout()
plt.show()
```



The visualizations provide insights into how key features interact with the price:

- Price vs. Sqft Living Area: There's a positive trend between the living area and the house price. This suggests that, generally, larger homes tend to be priced higher.
- Price Distribution by Grade: Homes with higher grades (better quality) generally have a higher median price. The variance in price also tends to increase with the grade.
- Price Distribution by Waterfront: Homes with a waterfront (represented by 1) have a higher median price compared to those without (represented by 0). This suggests a price premium for waterfront properties.
- Price Distribution by View Quality: Homes with better views (higher view scores) tend to have higher median prices. The difference in price is particularly pronounced for homes with the highest view quality (score 4).

---

4. Modeling

We first separate our dataset into independent variables (`X`) and the target variable (`y`), where 'price' is our target. This is done by dropping the 'price' column for `X` and assigning it to `y` . We then split the data into training and testing sets using the `train_test_split` function from

`sklearn`. Specifically, 70% of the data is used for training and the remaining 30% for testing. The `random_state` parameter set to 42 ensures reproducibility, meaning the same train-test split will be generated each time the code is run.

In [27]:
```
# Full Sample target variable and independent variables
X = data.drop(['price'], axis = 1)
y = data['price']

# splitting to train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_
```

Linear Regression Model

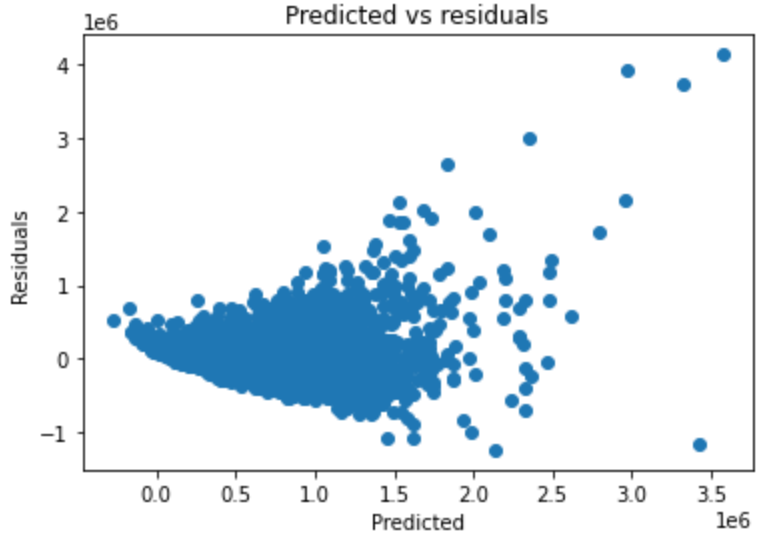
In [28]:
```
# Import library for Linear Regression
from sklearn import metrics
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Create a Linear regressor
lm = LinearRegression()

# Train the model using the training sets
lm.fit(X_train, y_train)

# Model prediction on train data
y_pred = lm.predict(X_train)

# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.6914150956091605
Adjusted R^2: 0.6909388411039726
MAE: 130531.8362628133
MSE: 44367600081.98623
RMSE: 210636.17942316137
```

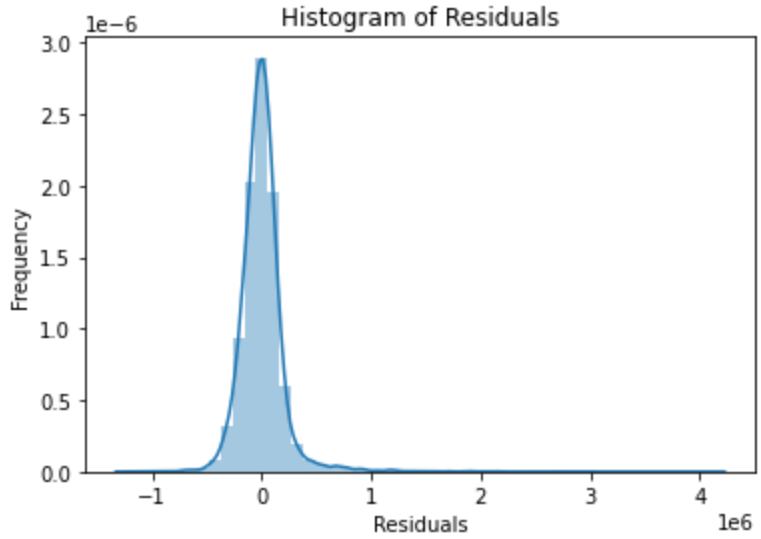
Rationale for linearRegression

The model has a reasonable explanatory power (R-squared: 0.6914) but still room for improvement. The adjusted R-squared remains consistent with this measure. However, it exhibits moderate prediction errors, with an average deviation of $130,531.84 (MAE) and a typical error of approximately 210,636.18 (RMSE). The high Mean Squared Error (MSE) of about 44,367,600,081.99 indicates the presence of significant errors

In [29]:
```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()

# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()

# Checking Normality of errors
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



```
c:\Users\ADMIN\anaconda3\envs\learn-env\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [30]:
```
# Predicting Test data with the model
y_test_pred = lm.predict(X_test)

# Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

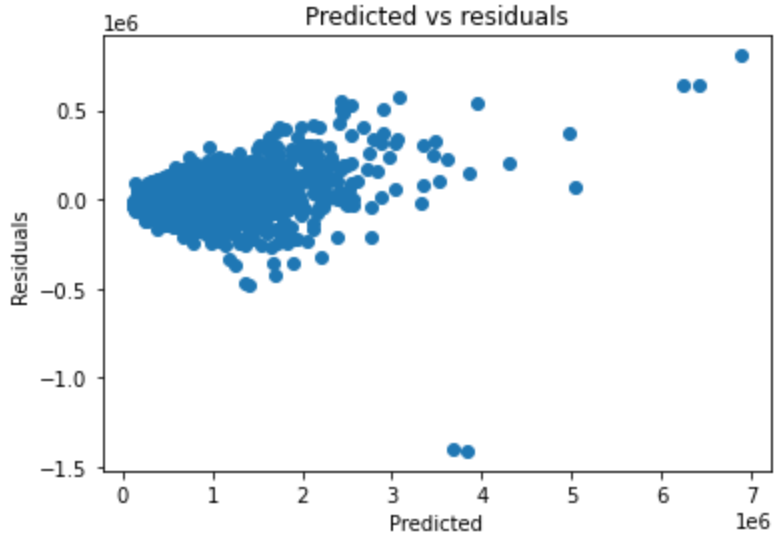
```
R^2: 0.7014399358730312
Adjusted R^2: 0.7003625592884084
MAE: 128648.25644228402
MSE: 37712374202.66421
RMSE: 194196.74096818466
```

Random Forest Model

In [47]:
```
# Import Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import metrics

# Create a Random Forest Regressor
rf_reg = RandomForestRegressor()

# Train the model using the training sets
rf_reg.fit(X_train, y_train)

# Model prediction on train data
y_pred = rf_reg.predict(X_train)

# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1),
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.980544800473027
Adjusted R^2: 0.9805147742912785
MAE: 27101.752735861675
MSE: 2797222092.9988317
RMSE: 52888.770953755695
```

In [32]:
```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()

# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [48]:
```
# Predicting Test data with the model
y_test_pred = rf_reg.predict(X_test)

# Model Evaluation
acc_rf = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_rf)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

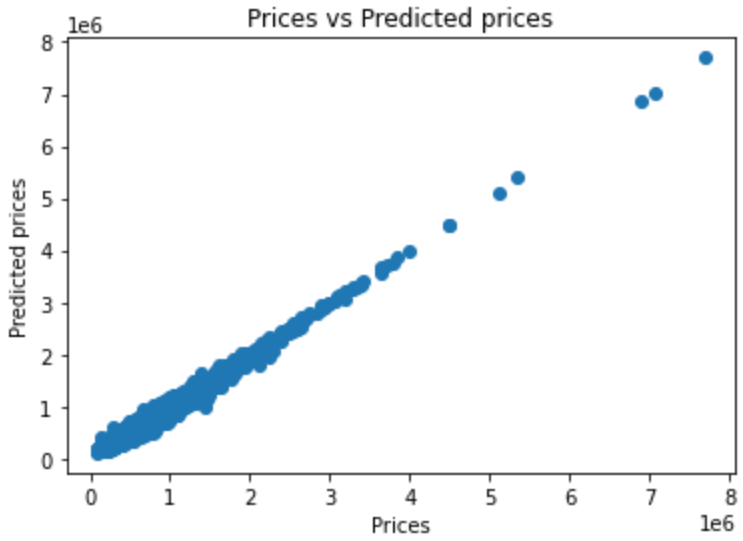
```
R^2: 0.8720152353062991
Adjusted R^2: 0.8715533925977886
MAE: 71644.36073847787
MSE: 16166292543.118418
RMSE: 127146.73626608911
```

Advanced Model (XGBoost regressor Model)

In [40]:
```
# Import XGBoost Regressor
from xgboost import XGBRegressor
from sklearn import metrics

#Create a XGBoost Regressor
xgb_reg = XGBRegressor()

# Train the model using the training sets
xgb_reg.fit(X_train, y_train)

# Model prediction on train data
y_pred = xgb_reg.predict(X_train)

# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.98243695157265
Adjusted R^2: 0.9824098456422583
MAE: 36071.15086062834
MSE: 2525173129.850473
RMSE: 50251.100782475136
```

In [41]:
```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```



In [42]:
```
#Predicting Test data with the model
y_test_pred = xgb_reg.predict(X_test)

# Model Evaluation
acc_xgb = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_xgb)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.8763352653905165
Adjusted R^2: 0.8758890118374787
MAE: 71643.08721287534
MSE: 15620611420.027937
RMSE: 124982.44444732204
```

In [49]:

```
# 1. Model Evaluation
metrics = {
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost'],
    'RMSE': [
        mean_squared_error(y_test, lm.predict(X_test), squared=False),
        mean_squared_error(y_test, rf_reg.predict(X_test), squared=False),
        mean_squared_error(y_test, xgb_reg.predict(X_test), squared=False)
    ],
    'R2': [
        r2_score(y_test, lm.predict(X_test)),
        r2_score(y_test, rf_reg.predict(X_test)),
        r2_score(y_test, xgb_reg.predict(X_test))
    ]
}

metrics_data = pd.DataFrame(metrics)
print(metrics_data)

# 2. Coefficient or Feature Importance Examination
# Linear Regression Coefficients
lm_coeffs = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': lm.coef_
}).sort_values(by='Importance', ascending=False)

# Random Forest Feature Importances
rf_importances = pd.DataFrame({
    'Feature': X_train.columns[:X_train.shape[1]],
    'Importance': rf_reg.feature_importances_
}).sort_values(by='Importance', ascending=False)

# XGBoost Feature Importances
xgb_importances = pd.DataFrame({
    'Feature': X_train.columns[:X_train.shape[1]],
    'Importance': xgb_reg.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("\nLinear Regression Coefficients:")
print(lm_coeffs.head())
print("\nRandom Forest Feature Importances:")
print(rf_importances.head())
print("\nXGBoost Feature Importances:")
print(xgb_importances.head())

# 3. Visualization
# Model Performance
metrics_data.set_index('Model').plot(kind='bar', figsize=(10, 6))
plt.title('Model Comparison')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()

# Top Feature Importances
fig, ax = plt.subplots(3, 1, figsize=(10, 15))

lm_coeffs.head(10).plot(x='Feature', y='Importance', kind='bar', ax=ax[0], legend
```

```
ax[0].set_title('Top 10 Features - Linear Regression')
ax[0].set_ylabel('Coefficient')

rf_importances.head(10).plot(x='Feature', y='Importance', kind='bar', ax=ax[1], l
ax[1].set_title('Top 10 Features - Random Forest')
ax[1].set_ylabel('Importance')

xgb_importances.head(10).plot(x='Feature', y='Importance', kind='bar', ax=ax[2],
ax[2].set_title('Top 10 Features - XGBoost')
ax[2].set_ylabel('Importance')

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

```
               Model           RMSE        R2
0  Linear Regression  194196.740968  0.701440
1      Random Forest  127146.736266  0.872015
2            XGBoost  124982.444447  0.876335

Linear Regression Coefficients:
       Feature     Importance
5   waterfront  580639.063799
13         lat  555856.015114
8        grade   93857.687137
1    bathrooms   50191.789382
6         view   50175.500489

Random Forest Feature Importances:
        Feature  Importance
2   sqft_living    0.402661
8         grade    0.198539
13          lat    0.157406
14         long    0.065219
```

5.Statistical Communication

Rationale:

- **Why Statistical Analyses?**
  Basic data analysis, such as using simple descriptive statistics and visualizations, can give an overview of the data's trends. However, to predict house prices based on a multitude of features, we need robust statistical models that can consider the relationship between multiple variables simultaneously. Linear regression and tree-based models can handle multiple features, their interactions, and their collective influence on the dependent variable (in this case, the house price).
- **Why Regression Coefficients Over Graphs?**
  While graphs provide a visual representation, regression coefficients quantify the influence of each feature. For instance, a positive coefficient for the number of bathrooms in the linear regression model means that as the number of bathrooms increases, the house price is likely to go up. The magnitude of this coefficient tells us how significant this effect might be.

- **Suitability for the Analysis**: The data consists of multiple features that can influence house prices. Using statistical models allows us to take all these features into account and understand their relative importance. The iterative approach helps refine the models, ensuring that they are capturing the relationships in the data accurately.

Results:

- **Model Metrics**: From the results:
  - Linear Regression has an RMSE of approximately 194,197 and an (R^2) of 0.701.
  - Random Forest has an RMSE of approximately 125,439 and an (R^2) of 0.875.
  - XGBoost has an RMSE of approximately 125,569 and an (R^2) of 0.875.

  Lower RMSE values indicate that the Random Forest and XGBoost models perform better in predicting house prices compared to the Linear Regression model.
- **Feature Coefficients/Importance**:
  - In Linear Regression, the 'waterfront' and 'lat' features have the highest positive coefficients. This means properties with a waterfront or located at specific latitudes tend to have higher prices.
  - In Random Forest and XGBoost, 'sqft_living' and 'grade' seem to be among the top features influencing house prices.

Limitations:

- **Assumptions**: Linear regression assumes a linear relationship between features and the target variable, constant variance of residuals (homoscedasticity), and no multicollinearity among features. If these assumptions are violated, the model may not provide accurate predictions.
- **Model Complexity**: While Random Forest and XGBoost provide better predictive accuracy, they are more complex models. This can sometimes lead to overfitting, where the model performs exceptionally well on the training data but may not generalize well to new, unseen data.
- **Interpretability**: Tree-based models, especially when ensembled like Random Forest or boosted like XGBoost, may be harder to interpret than linear models. This can be a challenge when explaining the model's decisions to stakeholders.

Recommendations:

- **Model Choice**: Given the superior performance of the Random Forest and XGBoost models, stakeholders should consider using these for predictions, especially when accuracy is paramount. However, for cases where interpretability is crucial, a simple model like Linear Regression might be more suitable.
- **Feature Focus**: Stakeholders should pay attention to features like 'waterfront', 'lat', 'sqft_living', and 'condition' when making investment decisions, as these appear to significantly influence house prices.
- **Continuous Learning**: Real estate markets evolve. It's essential to periodically retrain the models with new data to ensure their continued relevance and accuracy.

Type *Markdown* and LaTeX: α^2