1. Write a PySpark program to create a DataFrame with four columns: “name”, “age”, “city”, and
“gender” and perform the following operations:

• Insert minimum 10 values for the given columns.
• Filter rows with age greater than 30.
• Add a new column named it “tax”.
• Rename the “age” column to “years”.
• Drop Multiple Columns from the given data frame.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Initialize SparkSession
spark =
SparkSession.builder.appName("DataFrameOperations").getOrCreate()

# Create a DataFrame with minimum 10 values
data = [
    ("Alice", 25, "New York", "Female"),
    ("Bob", 35, "Los Angeles", "Male"),
    ("Charlie", 45, "Chicago", "Male"),
    ("Diana", 28, "Houston", "Female"),
    ("Eve", 32, "Phoenix", "Female"),
    ("Frank", 38, "San Diego", "Male"),
    ("Grace", 29, "San Francisco", "Female"),
    ("Hank", 41, "Seattle", "Male"),
    ("Ivy", 33, "Boston", "Female"),
    ("Jack", 27, "Austin", "Male"),
]

columns = ["name", "age", "city", "gender"]

df = spark.createDataFrame(data, columns)

# 1. Filter rows with age greater than 30
filtered_df = df.filter(col("age") > 30)

# 2. Add a new column named "tax" (example: flat tax of 100 for
demonstration)
updated_df = filtered_df.withColumn("tax", col("age") * 3)

# 3. Rename the "age" column to "years"
renamed_df = updated_df.withColumnRenamed("age", "years")

# 4. Drop multiple columns ("city" and "gender")
final_df = renamed_df.drop("city", "gender")
```

```python
# Show results for each step
print("Original DataFrame:")
df.show()

print("Filtered DataFrame (age > 30):")
filtered_df.show()

print("Updated DataFrame with 'tax' column:")
updated_df.show()

print("Renamed DataFrame (age -> years):")
renamed_df.show()

print("Final DataFrame after dropping columns:")
final_df.show()

# Stop SparkSession
spark.stop()
```

```
Original DataFrame:
+-------+---+-------------+------+
|   name|age|         city|gender|
+-------+---+-------------+------+
|  Alice| 25|     New York|Female|
|    Bob| 35|  Los Angeles|  Male|
|Charlie| 45|      Chicago|  Male|
|  Diana| 28|      Houston|Female|
|    Eve| 32|      Phoenix|Female|
|  Frank| 38|    San Diego|  Male|
|  Grace| 29|San Francisco|Female|
|   Hank| 41|      Seattle|  Male|
|    Ivy| 33|       Boston|Female|
|   Jack| 27|       Austin|  Male|
+-------+---+-------------+------+

Filtered DataFrame (age > 30):
+-------+---+-----------+------+
|   name|age|       city|gender|
+-------+---+-----------+------+
|    Bob| 35|Los Angeles|  Male|
|Charlie| 45|    Chicago|  Male|
|    Eve| 32|    Phoenix|Female|
|  Frank| 38|  San Diego|  Male|
|   Hank| 41|    Seattle|  Male|
|    Ivy| 33|     Boston|Female|
+-------+---+-----------+------+
```

```
Updated DataFrame with 'tax' column:
+-------+---+-----------+------+---+
|   name|age|       city|gender|tax|
+-------+---+-----------+------+---+
|    Bob| 35|Los Angeles|  Male|105|
|Charlie| 45|    Chicago|  Male|135|
|    Eve| 32|    Phoenix|Female| 96|
|  Frank| 38|  San Diego|  Male|114|
|   Hank| 41|    Seattle|  Male|123|
|    Ivy| 33|     Boston|Female| 99|
+-------+---+-----------+------+---+

Renamed DataFrame (age -> years):
+-------+-----+-----------+------+---+
|   name|years|       city|gender|tax|
+-------+-----+-----------+------+---+
|    Bob|   35|Los Angeles|  Male|105|
|Charlie|   45|    Chicago|  Male|135|
|    Eve|   32|    Phoenix|Female| 96|
|  Frank|   38|  San Diego|  Male|114|
|   Hank|   41|    Seattle|  Male|123|
|    Ivy|   33|     Boston|Female| 99|
+-------+-----+-----------+------+---+

Final DataFrame after dropping columns:
+-------+-----+---+
|   name|years|tax|
+-------+-----+---+
|    Bob|   35|105|
|Charlie|   45|135|
|    Eve|   32| 96|
|  Frank|   38|114|
|   Hank|   41|123|
|    Ivy|   33| 99|
+-------+-----+---+
```

2. Write a PySpark program to create a DataFrame containing information about various products,
including ProductID, ProductName, Category, Price, StockQuantity, & Rating and perform the following operations:

• Insert minimum 10 values for the given columns.
• Sort the DataFrame first by Price in descending order and then by Category in ascending order.
• Find the total sales amount for each product by category.
• Find the total sales amount and the total quantity sold for each product.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum as _sum

# Initialize SparkSession
spark =
SparkSession.builder.appName("ProductDataFrameOperations").getOrCreate(
)

# Create a DataFrame with Product details
data = [
    (101, "Laptop", "Electronics", 700, 10, 4.5),
    (102, "Smartphone", "Electronics", 500, 20, 4.3),
    (103, "Tablet", "Electronics", 300, 15, 4.1),
    (201, "Chair", "Furniture", 50, 50, 4.0),
    (202, "Table", "Furniture", 150, 30, 4.2),
    (203, "Couch", "Furniture", 500, 10, 4.6),
    (301, "Shampoo", "Personal Care", 10, 100, 4.3),
    (302, "Soap", "Personal Care", 5, 200, 4.4),
    (303, "Toothpaste", "Personal Care", 2, 300, 4.1),
    (401, "T-shirt", "Apparel", 20, 50, 4.2),
]

columns = ["ProductID", "ProductName", "Category", "Price",
"StockQuantity", "Rating"]

df = spark.createDataFrame(data, columns)

# 1. Sort the DataFrame by Price (descending) and Category (ascending)
sorted_df = df.orderBy(col("Price").desc(), col("Category").asc())

# 2. Calculate the total sales amount for each product by category
sales_df = df.withColumn("TotalSales", col("Price") *
col("StockQuantity"))
category_sales_df = sales_df.groupBy("Category").agg(
    _sum("TotalSales").alias("TotalSalesByCategory")
)
```

```
# 3. Find the total sales amount and the total quantity sold for each
product
product_sales_df = df.withColumn("TotalSales", col("Price") *
col("StockQuantity")).groupBy(
    "ProductID", "ProductName"
).agg(
    _sum("TotalSales").alias("TotalSalesAmount"),
    _sum("StockQuantity").alias("TotalQuantitySold")
)

# Show results for each step
print("Original DataFrame:")
df.show()

print("Sorted DataFrame (by Price desc, then Category asc):")
sorted_df.show()

print("Total Sales Amount by Category:")
category_sales_df.show()

print("Total Sales Amount and Quantity Sold by Product:")
product_sales_df.show()

# Stop SparkSession
spark.stop()
```

```
Total Sales Amount by Category:
+-------------+--------------------+
|     Category|TotalSalesByCategory|
+-------------+--------------------+
|  Electronics|               21500|
|    Furniture|               12000|
|      Apparel|                1000|
|Personal Care|                2600|
+-------------+--------------------+

Total Sales Amount and Quantity Sold by Product:
+---------+-----------+----------------+-----------------+
|ProductID|ProductName|TotalSalesAmount|TotalQuantitySold|
+---------+-----------+----------------+-----------------+
|      102| Smartphone|           10000|               20|
|      103|     Tablet|            4500|               15|
|      201|      Chair|            2500|               50|
|      101|     Laptop|            7000|               10|
|      202|      Table|            4500|               30|
|      203|      Couch|            5000|               10|
|      401|    T-shirt|            1000|               50|
|      301|    Shampoo|            1000|              100|
|      303| Toothpaste|             600|              300|
|      302|       Soap|            1000|              200|
+---------+-----------+----------------+-----------------+
```

```
Original DataFrame:
+---------+-----------+------------+-----+-------------+------+
|ProductID|ProductName|    Category|Price|StockQuantity|Rating|
+---------+-----------+------------+-----+-------------+------+
|      101|     Laptop| Electronics|  700|           10|   4.5|
|      102| Smartphone| Electronics|  500|           20|   4.3|
|      103|     Tablet| Electronics|  300|           15|   4.1|
|      201|      Chair|   Furniture|   50|           50|   4.0|
|      202|      Table|   Furniture|  150|           30|   4.2|
|      203|      Couch|   Furniture|  500|           10|   4.6|
|      301|    Shampoo|Personal Care|   10|          100|   4.3|
|      302|       Soap|Personal Care|    5|          200|   4.4|
|      303| Toothpaste|Personal Care|    2|          300|   4.1|
|      401|    T-shirt|     Apparel|   20|           50|   4.2|
+---------+-----------+------------+-----+-------------+------+

Sorted DataFrame (by Price desc, then Category asc):
+---------+-----------+------------+-----+-------------+------+
|ProductID|ProductName|    Category|Price|StockQuantity|Rating|
+---------+-----------+------------+-----+-------------+------+
|      101|     Laptop| Electronics|  700|           10|   4.5|
|      102| Smartphone| Electronics|  500|           20|   4.3|
|      203|      Couch|   Furniture|  500|           10|   4.6|
|      103|     Tablet| Electronics|  300|           15|   4.1|
|      202|      Table|   Furniture|  150|           30|   4.2|
|      201|      Chair|   Furniture|   50|           50|   4.0|
|      401|    T-shirt|     Apparel|   20|           50|   4.2|
|      301|    Shampoo|Personal Care|   10|          100|   4.3|
|      302|       Soap|Personal Care|    5|          200|   4.4|
|      303| Toothpaste|Personal Care|    2|          300|   4.1|
+---------+-----------+------------+-----+-------------+------+
```

3. Using PySpark, analyze airline flight data (e.g., departure and arrival times, delays, carrier information) and perform the following operations:

- Load a CSV file containing airline flight data
- Filter flights that were more than 15 minutes delayed.
- Analyze whether there is any correlation between the flight length and the likelihood of a delay.

```python
import random
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, when, rand

# Initialize Spark session
spark = SparkSession.builder \
    .appName("/content/Airlines.csv") \
    .getOrCreate()

# Load the data
file_path = "/content/Airlines.csv"  # Replace with your file path
data = spark.read.csv(file_path, header=True, inferSchema=True)

# Add a random delay time column
# Assign random values (1-30) to rows where Delay = 1, and 0 for Delay
= 0
data_with_delay_time = data.withColumn(
    "DelayTime",
    when(col("Delay") == 1, (rand() * 30).cast("int") + 1).otherwise(0)
)

# Display data with the new DelayTime column
print("Data with added DelayTime column:")
data_with_delay_time.show()

# Filter flights delayed by more than 15 minutes (using DelayTime)
delayed_flights = data_with_delay_time.filter(col("DelayTime") > 15)

print("Flights delayed by more than 15 minutes:")
delayed_flights.show()

# Stop Spark session
spark.stop()
```

Flights delayed by more than 15 minutes:

id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	DelayTime
2	US	1558	PHX	CLT	3	15	222	1	28
3	AA	2400	LAX	DFW	3	20	165	1	22
4	AA	2466	SFO	DFW	3	20	195	1	22
6	CO	1094	LAX	IAH	3	30	181	1	17
9	DL	2606	SFO	MSP	3	35	216	1	29
11	CO	223	ANC	SEA	3	49	201	1	21
12	DL	1646	PHX	ATL	3	50	212	1	28
24	HA	17	LAS	HNL	3	100	380	1	25
25	US	122	ANC	PHX	3	113	327	1	26
29	HA	206	HNL	OGG	3	300	36	1	25
39	OH	6338	GSO	ATL	3	315	93	1	16
56	9E	3886	DSM	ATL	3	330	135	1	30
94	AA	674	ORD	MIA	3	335	185	1	26
96	CO	463	ORD	IAH	3	335	164	1	23
103	CO	214	DFW	IAH	3	340	59	1	16
110	XE	2616	LRD	IAH	3	340	66	1	27
112	XE	3015	CRP	IAH	3	340	59	1	30
115	9E	3862	SAT	MEM	3	345	119	1	22
116	9E	3949	PVD	DTW	3	345	135	1	24
127	DL	1112	PIT	ATL	3	345	131	1	20

only showing top 20 rows

Data with added DelayTime column:

id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	DelayTime
1	CO	269	SFO	IAH	3	15	205	1	12
2	US	1558	PHX	CLT	3	15	222	1	28
3	AA	2400	LAX	DFW	3	20	165	1	22
4	AA	2466	SFO	DFW	3	20	195	1	22
5	AS	108	ANC	SEA	3	30	202	0	0
6	CO	1094	LAX	IAH	3	30	181	1	17
7	DL	1768	LAX	MSP	3	30	220	0	0
8	DL	2722	PHX	DTW	3	30	228	0	0
9	DL	2606	SFO	MSP	3	35	216	1	29
10	AA	2538	LAS	ORD	3	40	200	1	3
11	CO	223	ANC	SEA	3	49	201	1	21
12	DL	1646	PHX	ATL	3	50	212	1	28
13	DL	2055	SLC	ATL	3	50	210	0	0
14	AA	2408	LAX	DFW	3	55	170	0	0
15	AS	132	ANC	PDX	3	55	215	0	0
16	US	498	DEN	CLT	3	55	179	0	0
17	B6	98	DEN	JFK	3	59	213	0	0
18	CO	1496	LAS	IAH	3	60	162	0	0
19	DL	1450	LAS	MSP	3	60	181	0	0
20	CO	507	ONT	IAH	3	75	167	0	0

4. Consider airline flight data, given in the previous question. Perform the following operation
using PySpark

• Group the data by airline carrier and compute the average delay for each one.

```
import random
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, when, rand

# Initialize Spark session
spark = SparkSession.builder \
    .appName("/content/Airlines.csv") \
    .getOrCreate()

# Load the data
file_path = "/content/Airlines.csv"  # Replace with your file path
data = spark.read.csv(file_path, header=True, inferSchema=True)

# Add a random delay time column
# Assign random values (1-30) to rows where Delay = 1, and 0 for Delay = 0
data_with_delay_time = data.withColumn(
    "DelayTime",
    when(col("Delay") == 1, (rand() * 30).cast("int") + 1).otherwise(0)
)


# Analyze correlation between flight length and likelihood of delay
correlation = data_with_delay_time.corr("Length", "DelayTime")  # Correlation between Length and DelayTime
print(f"Correlation between flight length and delay time: {correlation}")

# Group by airline and calculate average delay time
average_delay_by_airline = data_with_delay_time.groupBy("Airline") \
    .agg(avg("DelayTime").alias("Average_DelayTime"))

# Display average delay time by airline
print("Average delay time by airline:")
average_delay_by_airline.show()

# Stop Spark session
spark.stop()
```

```
Correlation between flight length and delay time: 0.03235526239581078
Average delay time by airline:
+-------+------------------+
|Airline| Average_DelayTime|
+-------+------------------+
|     UA|5.0414207610702775|
|     AA| 6.012813211845103|
|     EV| 6.236464996605082|
|     B6| 7.221676236749117|
|     DL| 6.981358713488677|
|     OO| 6.993990528117164|
|     F9| 6.932156133828996|
|     YV|3.8070673952641165|
|     US| 5.214144927536232|
|     MQ|5.3928698265264305|
|     OH| 4.315360253365004|
|     HA|  5.09752599498028|
|     XE| 5.905930733149136|
|     AS| 5.234417226048295|
|     CO|  8.76697603939767|
|     FL| 4.679214481202285|
|     WN|10.821184522354592|
|     9E| 6.218070192400657|
+-------+------------------+
```

5. Given a Movie dataset containing user ratings for movies, using PySpark SQL perform the following operations

- Load a CSV file containing movie data
- Create temporary views for movies and ratings.
- Write queries to find the top 10 highest-rated movies with at least 10 ratings.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, avg, desc, explode, split

# Initialize Spark session
spark = SparkSession.builder.appName("MovieLensAnalysis").getOrCreate()

# File paths
movies_file = "/content/movies.csv"   # Replace with the path to your
movies.csv
ratings_file = "/content/ratings.csv"  # Replace with the path to your
ratings.csv

# Load datasets
movies_df = spark.read.csv(movies_file, header=True, inferSchema=True)
ratings_df = spark.read.csv(ratings_file, header=True,
inferSchema=True)
```

```python
# Create temporary views for SQL operations
movies_df.createOrReplaceTempView("movies")
ratings_df.createOrReplaceTempView("ratings")

# SQL Query for Top 10 Highest-Rated Movies with At Least 10 Ratings
query_top_rated = """
SELECT m.title, r.avg_rating, r.count_ratings
FROM (
    SELECT movieId, AVG(rating) AS avg_rating, COUNT(rating) AS
count_ratings
    FROM ratings
    GROUP BY movieId
    HAVING count_ratings >= 10
) r
JOIN movies m ON m.movieId = r.movieId
ORDER BY r.avg_rating DESC
LIMIT 10
"""

top_rated_movies = spark.sql(query_top_rated)
top_rated_movies.show()
```

```
+--------------------+------------------+-------------+
|               title|        avg_rating|count_ratings|
+--------------------+------------------+-------------+
|Secrets & Lies (1...|4.590909090909091|           11|
|Guess Who's Comin...|4.545454545454546|           11|
|Paths of Glory (1...|4.541666666666667|           12|
|Streetcar Named D...|             4.475|           20|
|Celebration, The ...|4.458333333333333|           12|
|          Ran (1985)|4.433333333333334|           15|
|Shawshank Redempt...|4.429022082018927|          317|
|His Girl Friday (...|4.392857142857143|           14|
|All Quiet on the ...|              4.35|           10|
| Hustler, The (1961)|4.333333333333333|           18|
+--------------------+------------------+-------------+
```

6. Consider the movie dataset provided in the previous question. Perform the given operation
using PySpark
• Find the most active users (users who have rated the most movies).
• Short the movies name in alphabetic order
• Calculate the average rating per genre

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, avg, desc, explode, split

# Initialize Spark session
spark = SparkSession.builder.appName("MovieLensAnalysis").getOrCreate()

# File paths
movies_file = "/content/movies.csv"   # Replace with the path to your
movies.csv
ratings_file = "/content/ratings.csv"  # Replace with the path to your
ratings.csv

# Load datasets
movies_df = spark.read.csv(movies_file, header=True, inferSchema=True)
ratings_df = spark.read.csv(ratings_file, header=True,
inferSchema=True)

# Create temporary views for SQL operations
movies_df.createOrReplaceTempView("movies")
ratings_df.createOrReplaceTempView("ratings")

# SQL Query for Most Active Users
query_active_users = """
SELECT userId, COUNT(movieId) AS movie_count
FROM ratings
GROUP BY userId
ORDER BY movie_count DESC
LIMIT 10
"""

active_users = spark.sql(query_active_users)
active_users.show()

# SQL Query to Sort Movies Alphabetically
query_sorted_movies = """
SELECT title
FROM movies
ORDER BY title ASC
"""

sorted_movies = spark.sql(query_sorted_movies)
```

```
sorted_movies.show()
# Split genres and explode
movies_with_genres = movies_df.withColumn("genre",
explode(split(col("genres"), "\\|")))
movies_with_genres.createOrReplaceTempView("movies_with_genres")

# SQL Query to Calculate Average Rating Per Genre
query_avg_rating_genre = """
SELECT g.genre, AVG(r.rating) AS avg_rating
FROM movies_with_genres g
JOIN ratings r ON g.movieId = r.movieId
GROUP BY g.genre
ORDER BY avg_rating DESC
"""

avg_rating_per_genre = spark.sql(query_avg_rating_genre)
avg_rating_per_genre.show()
```

userId	movie_count
414	2698
599	2478
474	2108
448	1864
274	1346
610	1302
68	1260
380	1218
606	1115
288	1055

genre	avg_rating
Film-Noir	3.920114942528736
War	3.8082938876312
Documentary	3.797785069729286
Crime	3.658293867274144
Drama	3.6561844113718758
Mystery	3.632460255407871
Animation	3.6299370349170004
IMAX	3.618335343787696
Western	3.583937823834197
Musical	3.5636781053649105
Adventure	3.5086089151939075
Romance	3.5065107040388437
Thriller	3.4937055799183425
Fantasy	3.4910005070136894
(no genres listed)	3.4893617021276597
Sci-Fi	3.455721162210752
Action	3.447984331646809
Children	3.412956125108601
Comedy	3.3847207640898267
Horror	3.258195034974626

title
"11'09""01 - Sept...
'71 (2014)
'Hellboy': The Se...
'Round Midnight (...
'Salem's Lot (2004)
'Til There Was Yo...
'Tis the Season f...
'burbs, The (1989)
'night Mother (1986)
(500) Days of Sum...
*batteries not in...
...All the Marble...
...And Justice fo...
00 Schneider - Ja...
1-900 (06) (1994)
10 (1979)
10 Cent Pistol (2...
10 Cloverfield La...
10 Items or Less ...
10 Things I Hate ...

only showing top 20 rows

7. Create a DataFrame containing real estate data with the following columns: HouseID, Location,
Size, Bedrooms, Bathrooms, Price etc. Use the given dataset to build a linear regression model
using PySpark's MLlib to predict the Price of a house based on the other features.

• Preprocess the data by handling missing values, encoding categorical variables (Location), and
normalizing numerical features (Size, Bedrooms, Bathrooms, etc).
• Split the data into training and testing sets.
• Train a linear regression model on the training data.
• Evaluate the model's performance on the test data using the root mean square error (RMSE)
• Display the feature importances and interpret the results.

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer,
MinMaxScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col

# Initialize Spark session
spark = SparkSession.builder.appName("House Price
Prediction").getOrCreate()

# Load dataset
file_path = "House Price India.csv"  # Replace with your file path
data = spark.read.csv(file_path, header=True, inferSchema=True)

# Display the first few rows to inspect the data
data.show(5)

# Check the column names
print("Columns in dataset:", data.columns)

# Handle missing values by dropping rows with null values
data = data.dropna()

# Encode categorical variable 'waterfront present' (binary: 0 or 1)
indexer = StringIndexer(inputCol="waterfront present",
outputCol="WaterfrontIndex")
data = indexer.fit(data).transform(data)

# Select features for the model
feature_columns = ["living area", "number of bedrooms", "number of
bathrooms", "WaterfrontIndex",
                   "number of floors", "number of views", "condition of
the house",
```

```
                    "grade of the house", "Area of the house(excluding
basement)", "Area of the basement"]
assembler = VectorAssembler(inputCols=feature_columns,
outputCol="features")
data = assembler.transform(data)

# Normalize numerical features
scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(data)
data = scaler_model.transform(data)

# Prepare final data for modeling
final_data = data.select(col("scaledFeatures").alias("features"),
col("Price").alias("label"))

# Split data into training and testing sets
train_data, test_data = final_data.randomSplit([0.8, 0.2], seed=42)

# Train a linear regression model
lr = LinearRegression(featuresCol="features", labelCol="label")
lr_model = lr.fit(train_data)

# Evaluate the model's performance
predictions = lr_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol="label",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE): {rmse}")

# Display model coefficients and intercept
print(f"Coefficients: {lr_model.coefficients}")
print(f"Intercept: {lr_model.intercept}")

# Interpret feature importances
feature_importances = list(zip(feature_columns, lr_model.coefficients))
print("Feature Importances:")
for feature, importance in feature_importances:
    print(f"{feature}: {importance}")
```

```
only showing top 5 rows

Columns in dataset: ['id', 'Date', 'number of bedrooms', 'number of bathrooms', 'living area', 'lot area', 'number of fl
Root Mean Squared Error (RMSE): 238449.36786611713
Coefficients: [1342301.3579574071,-1246667.7204660834,-100604.58998449595,531095.7654531715,15122.947711419945,209031.17
Intercept: -216276.6769940065
Feature Importances:
living area: 1342301.3579574071
number of bedrooms: -1246667.7204660834
number of bathrooms: -100604.58998449595
WaterfrontIndex: 531095.7654531715
number of floors: 15122.947711419945
number of views: 209031.17152502888
condition of the house: 232548.1048728994
grade of the house: 893691.852276711
Area of the house(excluding basement): 813883.6196434143
Area of the basement: 605437.5254883701
```