```python
from pyspark.sql import SparkSession
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler, StringIndexer,
OneHotEncoder
from pyspark.sql.functions import col

# Initialize Spark Session
spark =
SparkSession.builder.appName("AirbnbPricePrediction").getOrCreate()

# Load the Data into a Spark DataFrame
df = spark.read.csv("listings.csv", header=True, inferSchema=True)

# Explore the Dataset
df.printSchema()
df.show(5)
```

```
root
 |-- id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- host_id: string (nullable = true)
 |-- host_name: string (nullable = true)
 |-- neighbourhood_group: string (nullable = true)
 |-- neighbourhood: string (nullable = true)
 |-- latitude: string (nullable = true)
 |-- longitude: string (nullable = true)
 |-- room_type: string (nullable = true)
 |-- price: string (nullable = true)
 |-- minimum_nights: integer (nullable = true)
 |-- number_of_reviews: string (nullable = true)
 |-- last_review: string (nullable = true)
 |-- reviews_per_month: string (nullable = true)
 |-- calculated_host_listings_count: double (nullable = true)
 |-- availability_365: integer (nullable = true)

+----+--------------------+-------+---------+-------------------
+-------------+--------+---------+--------------+-----
+--------------+-----------------+-----------+-----------------
+------------------------------+----------------+
|  id|                name|host_id|host_name|neighbourhood_group|
neighbourhood|latitude| longitude|     room_type|price|
minimum_nights|number_of_reviews|last_review|reviews_per_month|
calculated_host_listings_count|availability_365|
+----+--------------------+-------+---------+-------------------
+-------------+--------+---------+--------------+-----
+--------------+-----------------+-----------+-----------------
+------------------------------+----------------+
|2318|Casa Madrona - Ur...|   2536|    Megan|       Central Area|
Madrona|47.61094|-122.29286|Entire home/apt|  475|            30|
32| 2020-02-01|             0.58|                           2.0|
```

```
238|
|6606|Fab, private seat...|  14942|    Joyce|Other neighborhoods|
Wallingford|47.65444|-122.33629|Entire home/apt|  102|             2|
153| 2021-07-12|             2.45|                           1.0|
87|
|9419|Glorious sun room...|  30559|Angielena|Other neighborhoods|
Georgetown|47.55017|-122.31937|   Private room|   75|             2|
149| 2021-06-28|             1.12|                           9.0|
275|
|9531|The Adorable Swee...|  31481|   Cassie|       West Seattle|
Fairmount Park|47.55495|-122.38663|Entire home/apt|  165|
5|                45| 2021-05-31|             0.52|
2.0|              276|
|9534|The Coolest Tange...|  31481|   Cassie|       West Seattle|
Fairmount Park|47.55627|-122.38607|Entire home/apt|  125|
5|                58| 2021-04-25|             0.58|
2.0|              311|
+----+--------------------+-------+---------+-------------------
+--------------+--------+----------+--------------+-----
+--------------+------------------+-----------+-----------------
+------------------------------+---------------+
only showing top 5 rows
```

```python
# Select relevant columns
selected_columns = ["price", "number_of_reviews", "availability_365",
"room_type"]
df_selected = df.select(selected_columns)

# Show the selected columns
df_selected.show(5)
```

```
+-----+-----------------+----------------+---------------+
|price|number_of_reviews|availability_365|      room_type|
+-----+-----------------+----------------+---------------+
|  475|               32|             238|Entire home/apt|
|  102|              153|              87|Entire home/apt|
|   75|              149|             275|   Private room|
|  165|               45|             276|Entire home/apt|
|  125|               58|             311|Entire home/apt|
+-----+-----------------+----------------+---------------+
only showing top 5 rows
```

Checking Missing Value Percentage of each column

```python
from pyspark.sql.functions import col, isnan, when, count
# Select relevant columns
selected_columns = ["price", "number_of_reviews", "availability_365",
"room_type"]
```

```
df_selected = df[selected_columns]

# Calculate the percentage of missing values for each column
missing_percentage = df_selected.select([(count(when(col(c).isNull() |
isnan(col(c)), c)) / df_selected.count()).alias(c) for c in
df_selected.columns])

missing_percentage.show()

+-------------------+-------------------+--------------------
+-------------------+
|              price|  number_of_reviews|    availability_365|
room_type|
+-------------------+-------------------+--------------------
+-------------------+
|9.75609756097561E-4|9.75609756097561E-4|0.001707317073170...|
9.75609756097561E-4|
+-------------------+-------------------+--------------------
+-------------------+
```

Since Missing Value Percentage is very low , we can remove the missing values...

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, regexp_replace, isnan
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# Clean the Data: Remove rows with missing values and convert price
column to numerical type
df_cleaned = df_selected.dropna()
df_cleaned = df_cleaned.withColumn("price",
regexp_replace(col("price"), "\\$", "").cast("float"))
# Cleaning the data is crucial for accurate model training. dropna()
removes rows with missing values. regexp_replace(col("price"), "\\$",
"").cast("float") removes the dollar sign from the price column and
converts it to a float type, enabling numerical operations.

# Filter out rows where price is null or NaN
df_cleaned = df_cleaned.filter(col("price").isNotNull() &
~isnan(col("price")))
#Any null or NaN values in the target variable (price) can cause
issues during model training. This step ensures that all values in the
price column are valid numbers.

# Convert number_of_reviews and availability_365 to integer types
df_cleaned = df_cleaned.withColumn("number_of_reviews",
```

```
col("number_of_reviews").cast("integer"))
df_cleaned = df_cleaned.withColumn("availability_365",
col("availability_365").cast("integer"))
#Ensuring that numerical columns are in the correct format (integer or
float) is necessary for numerical computations and model training.


# Encode Categorical Variables: StringIndexer and OneHotEncoder for
room_type
indexer = StringIndexer(inputCol="room_type",
outputCol="room_type_index")
encoder = OneHotEncoder(inputCol="room_type_index",
outputCol="room_type_vec")
#Machine learning algorithms require numerical input. StringIndexer
converts categorical values to numerical indices, and OneHotEncoder
converts these indices to one-hot encoded vectors. This process
transforms categorical data into a format suitable for machine
learning.
pipeline = Pipeline(stages=[indexer, encoder])
df_encoded = pipeline.fit(df_cleaned).transform(df_cleaned)

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=["number_of_reviews",
"availability_365", "room_type_vec"], outputCol="features")
df_final = assembler.transform(df_encoded).select("features", "price")
#"VectorAssembler" combines multiple columns into a single vector
column called features. This step is essential as most machine
learning algorithms in Spark expect the input data to be in this
format.

# Split the data into training and test sets
train_data, test_data = df_final.randomSplit([0.8, 0.2], seed=1234)
```

Linear Regression Model

```
# Fit a Machine Learning Model to Predict Price
lr = LinearRegression(featuresCol="features", labelCol="price")
lr_model = lr.fit(train_data)

# Evaluate the Model using Test Data
predictions = lr_model.transform(test_data)
# Initialize RegressionEvaluator
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
#"RegressionEvaluator" is used to evaluate the performance of
regression models. Here, we use RMSE (Root Mean Squared Error) as the
metric to measure the accuracy of our models.

print(f"Root Mean Squared Error (RMSE) on test data: {rmse}")
```

```
Root Mean Squared Error (RMSE) on test data: 114.20545932687672
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, regexp_replace, isnan
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression,
DecisionTreeRegressor, RandomForestRegressor, GBTRegressor
from pyspark.ml.evaluation import RegressionEvaluator
```

Decision Tree Regressor

```python
# Train and evaluate Decision Tree Regressor
dt = DecisionTreeRegressor(featuresCol="features", labelCol="price")
dt_model = dt.fit(train_data)
dt_predictions = dt_model.transform(test_data)
dt_rmse = evaluator.evaluate(dt_predictions)
print(f"Decision Tree Regressor - Root Mean Squared Error (RMSE) on
test data: {dt_rmse}")
```

```
Decision Tree Regressor - Root Mean Squared Error (RMSE) on test data:
113.76494204439346
```

Random Forest Regressor

```python
# Train and evaluate Random Forest Regressor
rf = RandomForestRegressor(featuresCol="features", labelCol="price")
rf_model = rf.fit(train_data)
rf_predictions = rf_model.transform(test_data)
rf_rmse = evaluator.evaluate(rf_predictions)
print(f"Random Forest Regressor - Root Mean Squared Error (RMSE) on
test data: {rf_rmse}")
```

```
Random Forest Regressor - Root Mean Squared Error (RMSE) on test data:
112.29230554900435
```

Gradient-Boosted Tree Regressor

```python
# Train and evaluate Gradient-Boosted Tree Regressor
gbt = GBTRegressor(featuresCol="features", labelCol="price")
gbt_model = gbt.fit(train_data)
gbt_predictions = gbt_model.transform(test_data)
gbt_rmse = evaluator.evaluate(gbt_predictions)
print(f"Gradient-Boosted Tree Regressor - Root Mean Squared Error
(RMSE) on test data: {gbt_rmse}")
```

```
Gradient-Boosted Tree Regressor - Root Mean Squared Error (RMSE) on
test data: 113.482566169064
```

```python
from sklearn.metrics import mean_squared_error
import numpy as np

# Extract features and labels for Scikit-Learn
train_features = np.array(train_data.select("features").collect())
train_labels = np.array(train_data.select("price").collect())
test_features = np.array(test_data.select("features").collect())
test_labels = np.array(test_data.select("price").collect())

train_features = np.array([np.array(x[0]) for x in train_features])
train_labels = np.array([x[0] for x in train_labels])
test_features = np.array([np.array(x[0]) for x in test_features])
test_labels = np.array([x[0] for x in test_labels])
```

Support Vector Regressor

```python
from sklearn.svm import LinearSVR
# Train and evaluate Support Vector Regressor
svr = LinearSVR(max_iter=100)
svr.fit(train_features, train_labels)
svr_predictions = svr.predict(test_features)
svr_rmse = mean_squared_error(test_labels, svr_predictions,
squared=False)
print(f"Support Vector Regressor - Root Mean Squared Error (RMSE) on
test data: {svr_rmse}")
```

```
Support Vector Regressor - Root Mean Squared Error (RMSE) on test
data: 125.52824819303551
```

```
c:\Users\Yehan Perera\anaconda3\envs\myenv\lib\site-packages\sklearn\
svm\_base.py:1235: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  warnings.warn(
c:\Users\Yehan Perera\anaconda3\envs\myenv\lib\site-packages\sklearn\
metrics\_regression.py:492: FutureWarning: 'squared' is deprecated in
version 1.4 and will be removed in 1.6. To calculate the root mean
squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

```python
from pyspark.ml.regression import (
    LinearRegression, DecisionTreeRegressor, RandomForestRegressor,
    GBTRegressor, GeneralizedLinearRegression, IsotonicRegression
)
```

Generalized Linear Regression

```python
# Train and evaluate Generalized Linear Regression
glr = GeneralizedLinearRegression(featuresCol="features",
labelCol="price", family="gaussian", link="identity")
```

```
glr_model = glr.fit(train_data)
glr_predictions = glr_model.transform(test_data)
glr_rmse = evaluator.evaluate(glr_predictions)
print(f"Generalized Linear Regression - Root Mean Squared Error (RMSE)
on test data: {glr_rmse}")
```

```
Generalized Linear Regression - Root Mean Squared Error (RMSE) on test
data: 114.20545932687672
```

Isotonic Regression

```
# Train and evaluate Isotonic Regression
iso = IsotonicRegression(featuresCol="features", labelCol="price")
iso_model = iso.fit(train_data)
iso_predictions = iso_model.transform(test_data)
iso_rmse = evaluator.evaluate(iso_predictions)
print(f"Isotonic Regression - Root Mean Squared Error (RMSE) on test
data: {iso_rmse}")
```

```
Isotonic Regression - Root Mean Squared Error (RMSE) on test data:
122.92554791201124
```

```
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
```

XGBoost Regressor

```
# Train and evaluate XGBoost Regressor
xgb = XGBRegressor(n_estimators=100, max_depth=3, learning_rate=0.1,
objective='reg:squarederror')
xgb.fit(train_features, train_labels)
xgb_predictions = xgb.predict(test_features)
xgb_rmse = mean_squared_error(test_labels, xgb_predictions,
squared=False)
print(f"XGBoost Regressor - Root Mean Squared Error (RMSE) on test
data: {xgb_rmse}")
```

```
XGBoost Regressor - Root Mean Squared Error (RMSE) on test data:
111.41781368154977
```

```
c:\Users\Yehan Perera\anaconda3\envs\myenv\lib\site-packages\sklearn\
metrics\_regression.py:492: FutureWarning: 'squared' is deprecated in
version 1.4 and will be removed in 1.6. To calculate the root mean
squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

CatBoost Regressor

```python
# Train and evaluate CatBoost Regressor
cat = CatBoostRegressor(iterations=100, depth=3, learning_rate=0.1,
loss_function='RMSE', verbose=False)
cat.fit(train_features, train_labels)
cat_predictions = cat.predict(test_features)
cat_rmse = mean_squared_error(test_labels, cat_predictions,
squared=False)
print(f"CatBoost Regressor - Root Mean Squared Error (RMSE) on test
data: {cat_rmse}")
```

```
CatBoost Regressor - Root Mean Squared Error (RMSE) on test data:
111.06339996789264
```

```
c:\Users\Yehan Perera\anaconda3\envs\myenv\lib\site-packages\sklearn\
metrics\_regression.py:492: FutureWarning: 'squared' is deprecated in
version 1.4 and will be removed in 1.6. To calculate the root mean
squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Smallest RMSE on test data was obtained by CatBoost Regressor.(111.0634)

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