PORTFOLIO PROJECT ON HOUSING DATASET

Scenario:

You are a Lead Data Scientist at a FirstService Corporation, a Canadian real estate company . The company wants to develop a predictive model to estimate house price based on various features such as the size of the property, the number of bedrooms, the furnishing status, and the city. The company has provided you with a dataset containing information about properties and their respective prices.

Your task is to preprocess the dataset, build a predictive model using Linear Regression, and evaluate its performance using appropriate regression metrics. You will also need to interpret the model's coefficients and assess its accuracy in predicting rent values.

Explaining the results

From our linear regression model, the model is performing very well with an R2 of 0.9955 showing that it is able to predict the house prices based on the features. This means that 99.55% of the variance in house prices is explained by the model. A value close to 1.0 indicates excellent model performance.

With our model performing very well, this means that the model when used will be will be able to provide the company with accurate and reliable predictions that drive informed decision making which in the long run will lead to better profitability and efficiency

Explain the cofficients :

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

#if using google collAB
from google.colab import drive

drive.mount('/content/drive')  # Re-mount
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Step 1: Load the dataset from a CSV file
file_path = '/content/drive/MyDrive/Colab Notebooks/Ontario_House_Price_Dataset.csv'  # Adjust the path

df = pd.read_csv(file_path)

# Step 2: Explore the dataset (optional)
print("Dataset Preview:")
print(df.head())
```

```
Dataset Preview:
       City  Square Footage  Bedrooms  Bathrooms  Lot Size  Year Built  \
0   Markham            3840         2          3     15901        1917
1  Brampton            2439         5          3      6445        1920
2   Vaughan            1627         1          3      2380        1991
3  Hamilton            3530         5          1     10865        1998
4   Markham            3851         5          2     14926        1985

  Furnishing Status    Area Type Property Type      Price
0       Unfurnished     Suburban     Townhouse   979202.1
1       Unfurnished     Suburban      Detached   847597.5
2       Unfurnished  City Center     Townhouse   498869.0
3         Furnished        Rural      Detached  1035639.5
4         Furnished     Suburban         Condo  1094068.6
```

```
# Step 3: Clean and preprocess the data
# 3a.# Handle missing values
# Replace missing values in numerical columns with their mean
import numpy as np
numeric_columns = df.select_dtypes(include=[np.number]).columns
df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].mean())
```

```
# Replace missing values in categorical columns with their mode
df['Area Type'].fillna(df['Area Type'].mode()[0], inplace=True) # Replace missing categorical values with the mode
df['City'].fillna(df['City'].mode()[0], inplace=True) # Replace missing categorical values with the mode
df['Furnishing Status'].fillna(df['Furnishing Status'].mode()[0], inplace=True) # Replace missing categorical values with the mode
df['Property Type'].fillna(df['Property Type'].mode()[0], inplace=True) # Replace missing categorical values with the mode
```

```
<ipython-input-17-695c8a1aae5c>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df['Area Type'].fillna(df['Area Type'].mode()[0], inplace=True) # Replace missing categorical values with the mode
<ipython-input-17-695c8a1aae5c>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df['City'].fillna(df['City'].mode()[0], inplace=True) # Replace missing categorical values with the mode
<ipython-input-17-695c8a1aae5c>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df['Furnishing Status'].fillna(df['Furnishing Status'].mode()[0], inplace=True) # Replace missing categorical values with the mode
<ipython-input-17-695c8a1aae5c>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me

  df['Property Type'].fillna(df['Property Type'].mode()[0], inplace=True) # Replace missing categorical values with the mode
```

```
# One-hot encode a column
df = pd.get_dummies(df, columns=['Area Type','Furnishing Status', 'City', 'Property Type'], drop_first=True)  #Use when the categorical vari

print(df) # to confirm if the encoding has taken place
```

```
996           True              False                           True
997          False              False                          False
998          False               True                          False
999          False              False                          False

     City_Hamilton  ...  City_London  City_Markham  City_Mississauga  \
0            False  ...        False          True             False
1            False  ...        False         False             False
2            False  ...        False         False             False
3             True  ...        False         False             False
4            False  ...        False          True             False
..             ...  ...          ...           ...               ...
995          False  ...        False         False             False
996          False  ...        False         False             False
997          False  ...        False         False             False
998          False  ...        False         False             False
999          False  ...        False         False             False

     City_Ottawa  City_Toronto  City_Vaughan  City_Windsor  \
0          False         False         False         False
1          False         False         False         False
2          False         False          True         False
3          False         False         False         False
4          False         False         False         False
..           ...           ...           ...           ...
995        False         False         False          True
996        False         False         False          True
997        False         False          True         False
998         True         False         False         False
999        False         False         False         False

     Property Type_Detached  Property Type_Semi-Detached  \
0                     False                        False
1                      True                        False
2                     False                        False
3                      True                        False
4                     False                        False
..                      ...                          ...
995                   False                        False
996                    True                        False
997                   False                        False
998                    True                        False
999                    True                        False

     Property Type_Townhouse
0                       True
1                      False
2                       True
3                      False
4                      False
..                       ...
995                    False
996                    False
997                    False
998                    False
999                    False

[1000 rows x 21 columns]
```

```
# Step 4: Define features (X) and target (y)
X = df.drop(columns=['Price']) # Independent variables #this means part apart from Rent, all the other columns should be included in the X v
y = df['Price']  # Dependent variable (target) #the Y is what we are trying to predict
```

```
# Step 5: Split the data into training (70%) and testing sets (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Step 6: Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

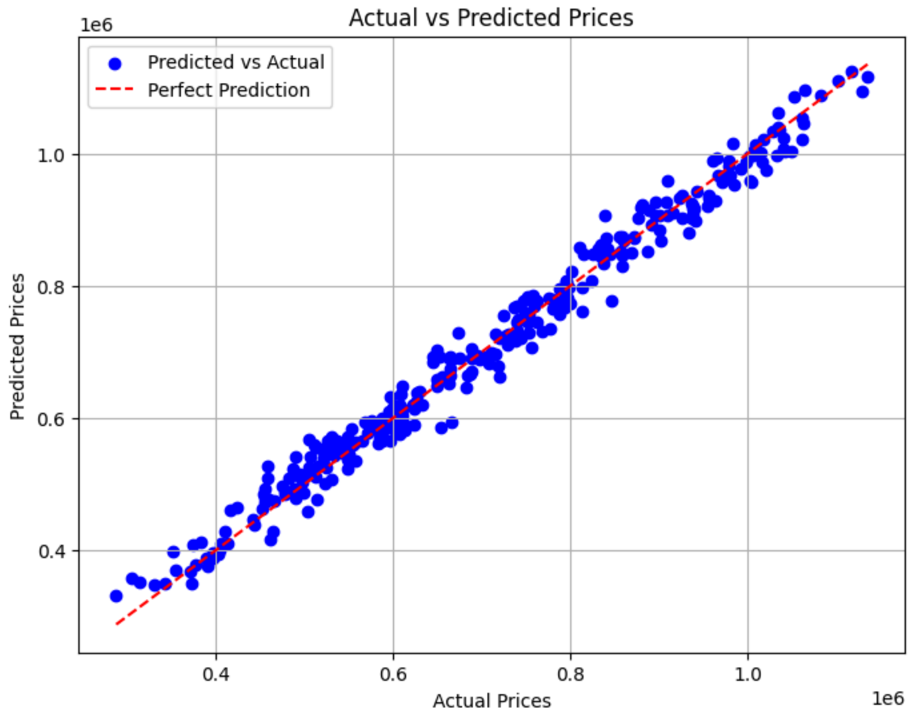
```
▾ LinearRegression
LinearRegression()
```

```
# Step 7: Make predictions on the test data
y_pred = model.predict(X_test)

# Step 8: Evaluate the model
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
r2 = r2_score(y_test, y_pred)  # R-squared score  #higher r2 score means model is good: anything above the threshold the company wants it to
```

```
# Step 9: Display the predictions and evaluation metrics
test_results = pd.DataFrame({
    "Actual Price": y_test.values,
    "Predicted Price": y_pred,
    "Error (Actual - Predicted)": y_test.values - y_pred
})

print("=== Predictions on Test Data ===")
print(test_results)
print("\n=== Model Evaluation Metrics ===")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2): {r2}")
print("\n=== Model Coefficients ===")
print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {dict(zip(X.columns, model.coef_))}")
```

```
=== Predictions on Test Data ===
     Actual Price  Predicted Price  Error (Actual - Predicted)
0        610713.4     6.321510e+05               -21437.616851
1        858515.5     8.373855e+05                21129.954228
2        998653.1     9.999038e+05                -1250.669019
3        397757.1     3.893258e+05                 8431.330475
4        389888.0     3.883618e+05                 1526.214952
..            ...              ...                         ...
295      812802.6     8.044422e+05                 8360.410993
296     1064053.2     1.084367e+06               -20313.705477
297      872522.2     8.711346e+05                 1387.551893
298      653975.0     6.477534e+05                 6221.550825
299      623823.7     6.350964e+05               -11272.734019

[300 rows x 3 columns]

=== Model Evaluation Metrics ===
Mean Squared Error (MSE): 178472830.83136845
Mean Absolute Error (MAE): 11593.82989567256
R-squared (R2): 0.9955423438282373

=== Model Coefficients ===
Intercept: 9466.81081223744
Coefficients: {'Square Footage': 199.683374309138, 'Bedrooms': 50641.541445666735, 'Bathrooms': 29483.017280283137, 'Lot Size': 0.120796
```

```
# Step 10: Plot actual prices vs. predicted prices
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color="blue", label="Predicted vs Actual")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color="red", linestyle="--", label="Perfect Prediction")

# Add labels, title, and legend
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.legend()
plt.grid(True)
plt.show()
```



Actual vs Predicted Prices
1e6
Predicted vs Actual
Perfect Prediction
Predicted Prices
1.0
0.8
0.6
0.4
0.4
0.6
0.8
1.0
Actual Prices
1e6