




# PROJECT TWO

Navneet Kaur

# Project Overview

The Data This project uses the King County House Sales dataset, which can be found in `kc_house_data.csv` in the data folder in this repo. The description of the column names can be found in `column_names.md` in the same folder.




# Column Names and descriptions for Kings County Data Set

**\*\*id\*\*** - unique identified for a house\*

- **\*\*dateDate\*\*** - house was sold\*
- **\*\*pricePrice\*\*** - is prediction target\*
- **\*\*bedroomsNumber\*\*** - of Bedrooms/House\*
- **\*\*bathroomsNumber\*\*** - of bathrooms/bedrooms\*
- **\*\*sqft\_livingsquare\*\*** - footage of the home
- **\*\*sqft\_lotsquare\*\*** - footage of the lot\*
- **\*\*floorsTotal\*\*** - floors (levels) in house\*
- **\*\*waterfront\*\*** - House which has a view to a waterfront\*
- **\*\*view\*\*** - Has been viewed\*
- **\*\*condition\*\*** - How good the condition is ( Overall )\*
- **\*\*grade\*\*** - overall grade given to the housing unit, based on King County grading system\*
- **\*\*sqft\_above\*\*** - square footage of house apart from basement\*
- **\*\*sqft\_basement\*\*** - square footage of the basement
- **\*\*yr\_built\*\*** - Built Year\*
- **\*\*yr\_renovated\*\*** - Year when house was renovated\*
- **\*\*zipcode\*\*** - zip\*
- **\*\*lat\*\*** - Latitude coordinate\*
- **\*\*long\*\*** - Longitude coordinate\*
- **\*\*sqft\_living15\*\*** - The square footage of interior housing living space for the nearest 15 neighbors\*
- **\*\*sqft\_lot15\*\*** - The square footage of the land lots of the nearest 15 neighbors

# Business Problem



A business problem for real estate agency is the need to provide advice to homeowners about how home renovations might increase the estimated value of their homes, and by what amount.

# Previewing the data

```
[3]: import pandas as pd  
df=pd.read_csv("kc_house_data.csv")
```

```
[4]: df.head()
```

```
[4]:
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	1955	0.0
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1951	1991.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1933	NaN
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1965	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1987	0.0

5 rows × 21 columns

Home

Untitled

localhost:8888/notebooks/Untitled.ipynb

Import favorites | Gmail | YouTube

jupyter

Untitled

Last Checkpoint: 3 minutes ago

Python

File | Edit | View | Run | Kernel | Settings | Help

Trusted

Open in... Python 3 (ipykernel)

[5]: df.info()

<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 float64  
9 view 21534 non-null float64  
10 condition 21597 non-null int64  
11 grade 21597 non-null int64  
12 sqft\_above 21597 non-null int64  
13 sqft\_basement 21597 non-null object  
14 yr\_built 21597 non-null int64  
15 yr\_renovated 17755 non-null float64  
16 zipcode 21597 non-null int64  
17 lat 21597 non-null float64  
18 long 21597 non-null float64  
19 sqft\_living15 21597 non-null int64  
20 sqft\_lot15 21597 non-null int64  
dtypes: float64(8), int64(11), object(2)  
memory usage: 3.5+ MB

# Histograms with kde overlay to check distribution

[14]: import seaborn as sns  
import matplotlib.pyplot as plt

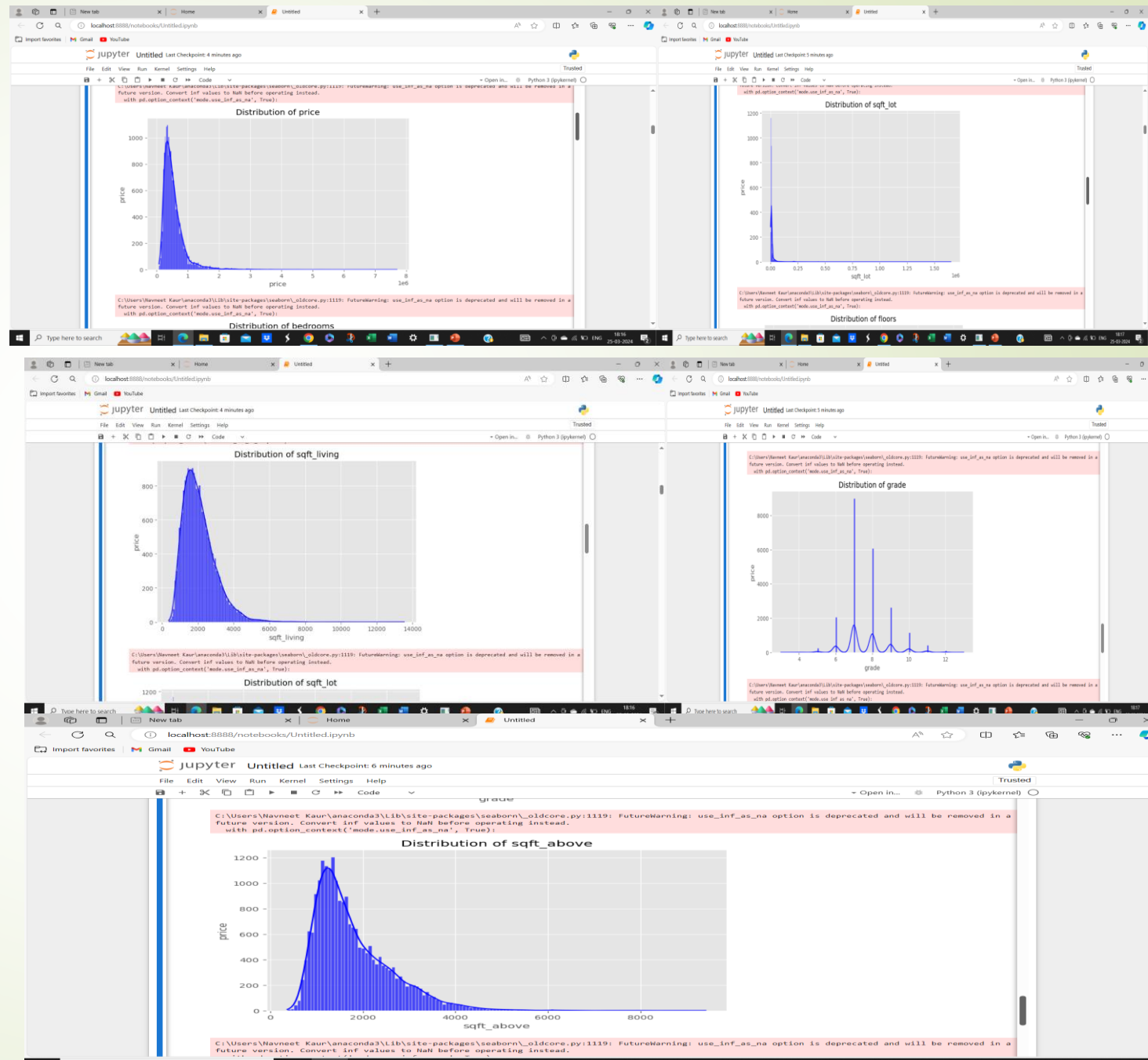
Type here to search

18:15

25-03-2024

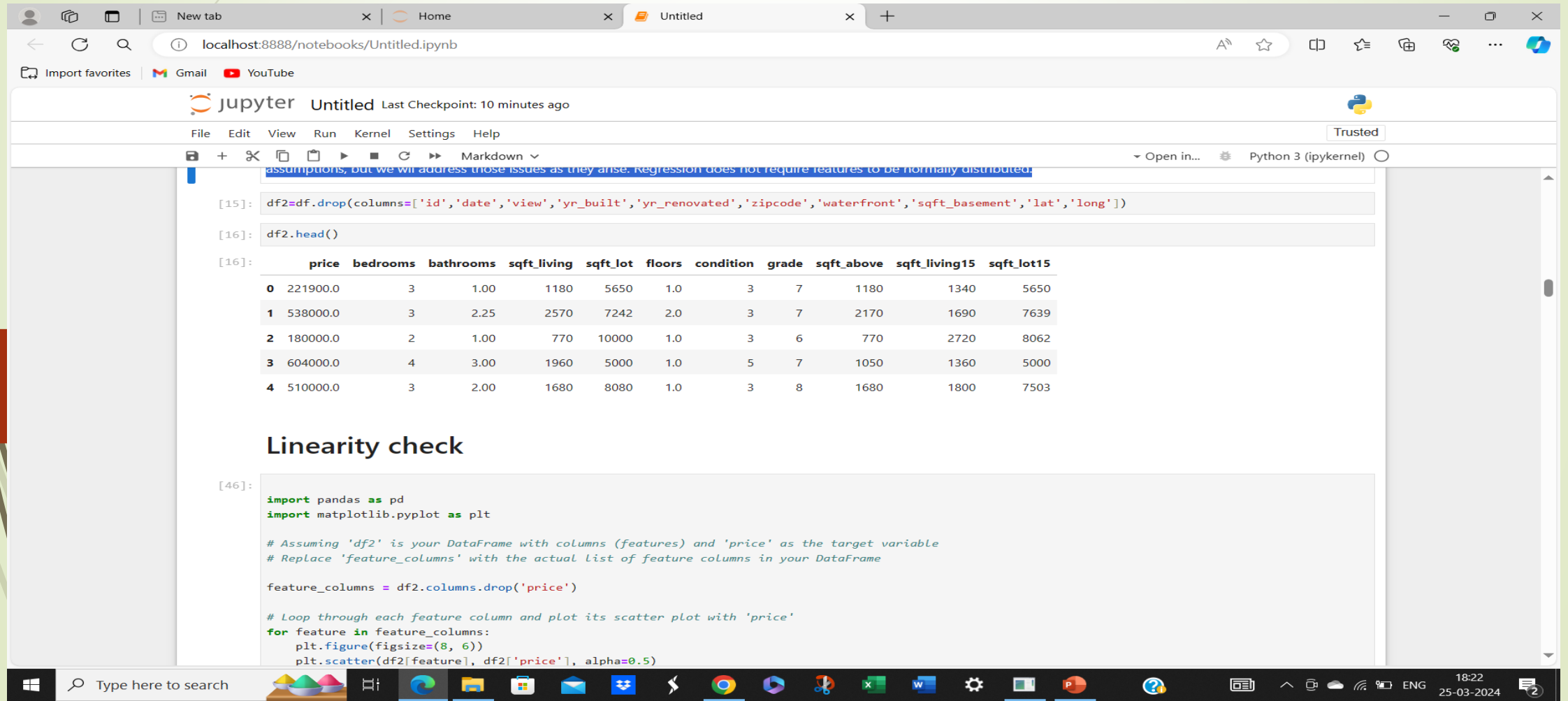
# Distribution of Predictors

Many of the variables do not follow a normal distribution, and scales are dramatically different for some variables. This may create issues with satisfying all regression assumptions, but we will address those issues as they arise. Regression does not require features to be normally distributed.





# Data Cleaning



The screenshot shows a Jupyter Notebook titled 'Untitled' running on a local host. The notebook contains two code cells. The first cell drops several columns from a DataFrame and displays the first five rows of the resulting DataFrame. The second cell imports pandas and matplotlib, and sets up a loop to create scatter plots for each feature against the price variable.

Code cell [15]:

```
df2=df.drop(columns=['id','date','view','yr_built','yr_renovated','zipcode','waterfront','sqft_basement','lat','long'])
```

Code cell [16]:

```
df2.head()
```

Output of [16]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_living15	sqft_lot15
0	221900.0	3	1.00	1180	5650	1.0	3	7	1180	1340	5650
1	538000.0	3	2.25	2570	7242	2.0	3	7	2170	1690	7639
2	180000.0	2	1.00	770	10000	1.0	3	6	770	2720	8062
3	604000.0	4	3.00	1960	5000	1.0	5	7	1050	1360	5000
4	510000.0	3	2.00	1680	8080	1.0	3	8	1680	1800	7503

Code cell [46]:

```
import pandas as pd
import matplotlib.pyplot as plt

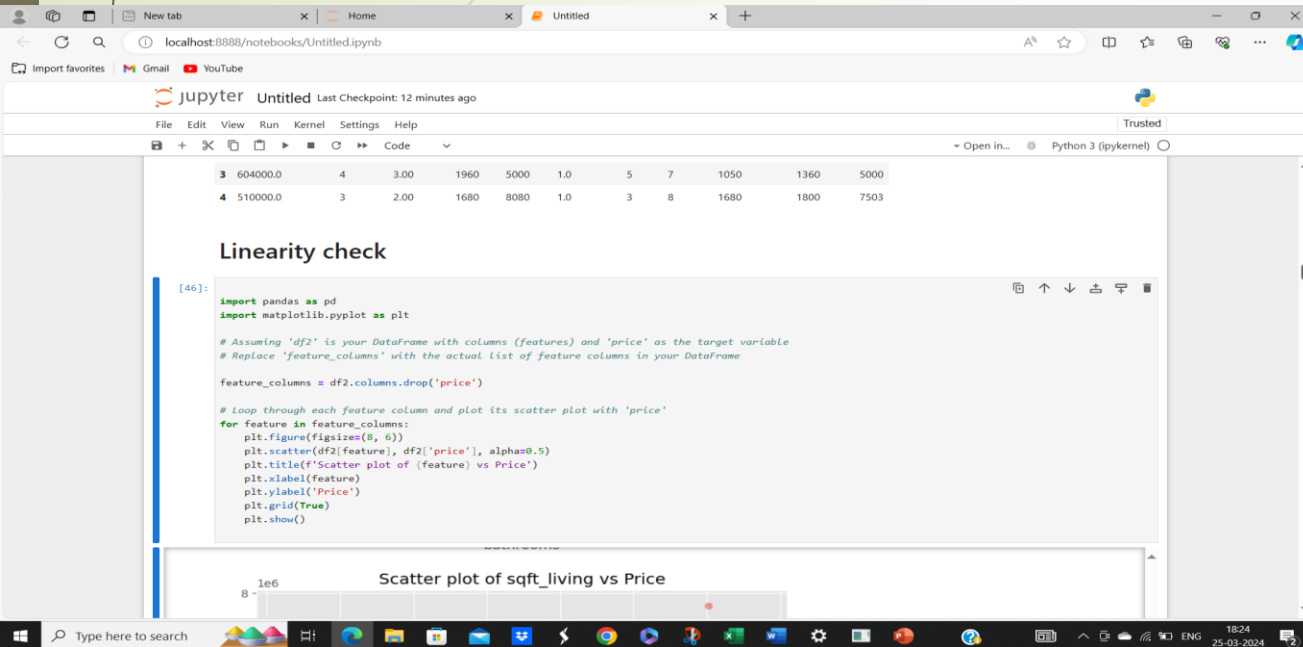
# Assuming 'df2' is your DataFrame with columns (features) and 'price' as the target variable
# Replace 'feature_columns' with the actual list of feature columns in your DataFrame

feature_columns = df2.columns.drop('price')

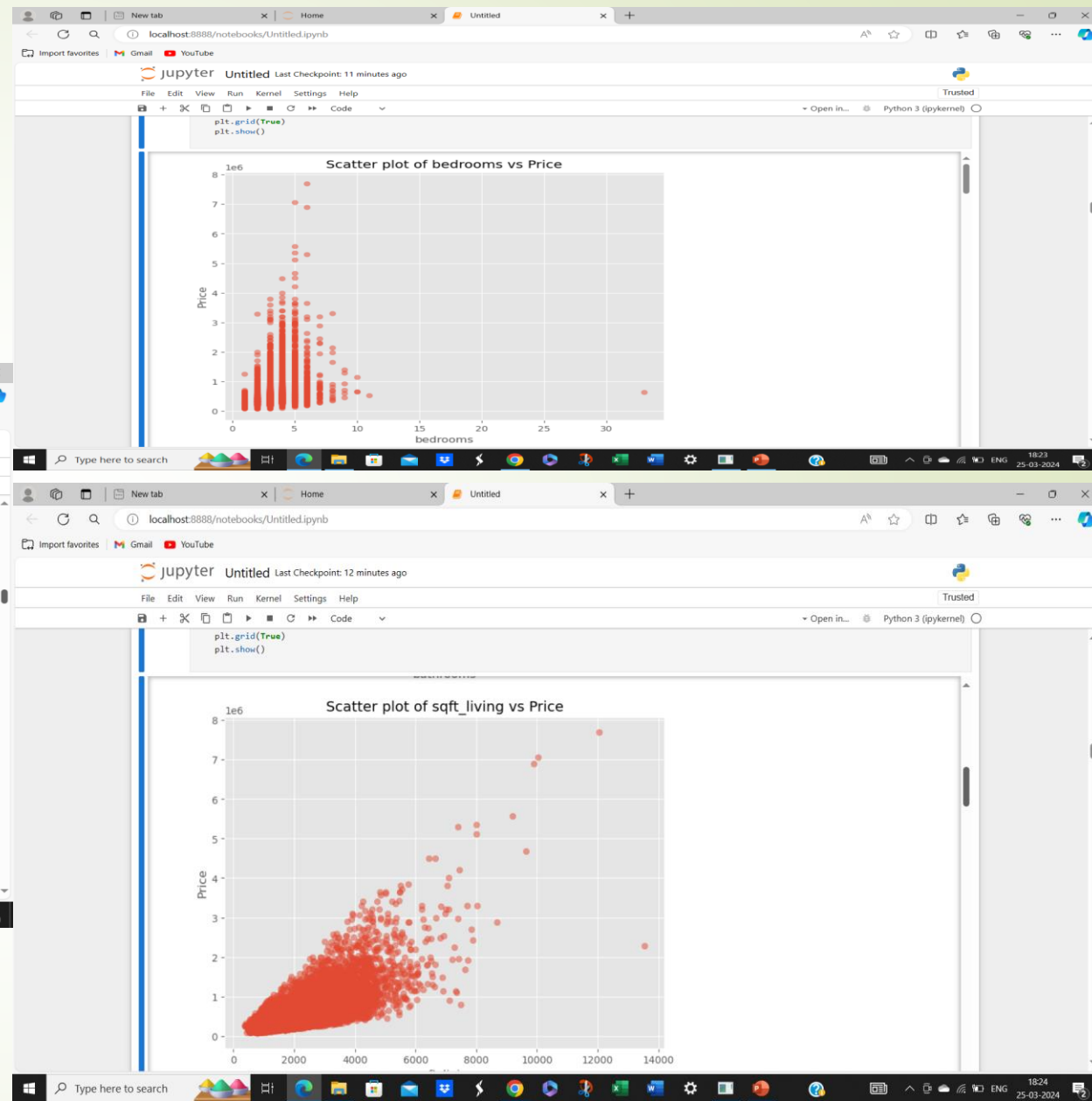
# Loop through each feature column and plot its scatter plot with 'price'
for feature in feature_columns:
    plt.figure(figsize=(8, 6))
    plt.scatter(df2[feature], df2['price'], alpha=0.5)
```



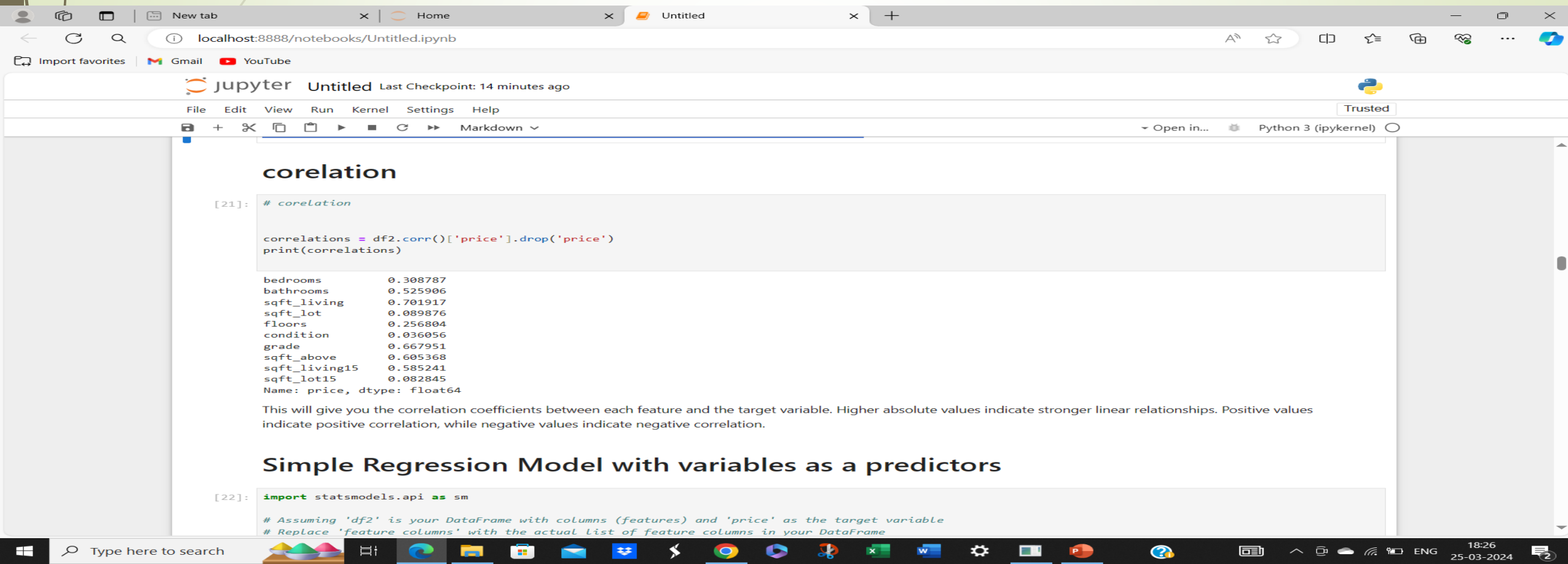
# Linearity check



The 'view' and 'floors' do not have a clear linear relationship with house price. Sqft\_living has a stronger linear relationship than Sqft\_above, so it will be used in the multiregression model. Due to its multicollinearity with sqft\_above, sqft\_above will be excluded.



# Co-relation



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs include 'New tab', 'Home', and 'Untitled'. The address bar shows 'localhost:8888/notebooks/Untitled.ipynb'. The Jupyter Notebook interface has a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Settings', and 'Help'. The toolbar includes icons for file operations and a 'Markdown' dropdown. The notebook content is titled 'corelation' and shows a code cell with the following code:

```
[21]: # corelation

correlations = df2.corr()['price'].drop('price')
print(correlations)
```

The output of the code is a list of correlation coefficients for various features:

bedrooms	0.308787
bathrooms	0.525906
sqft_living	0.701917
sqft_lot	0.089876
floors	0.256804
condition	0.036056
grade	0.667951
sqft_above	0.605368
sqft_living15	0.585241
sqft_lot15	0.082845

Name: price, dtype: float64

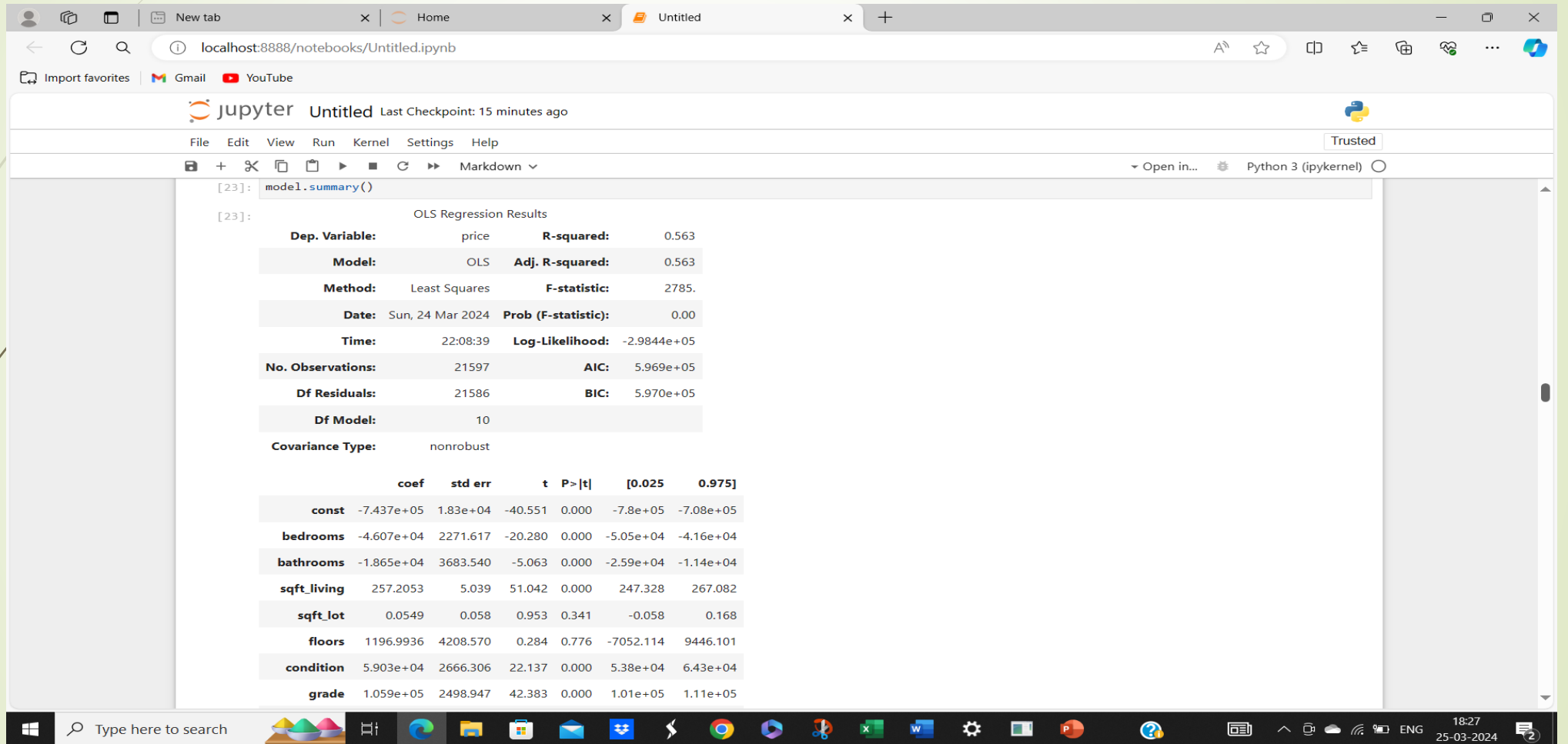
This will give you the correlation coefficients between each feature and the target variable. Higher absolute values indicate stronger linear relationships. Positive values indicate positive correlation, while negative values indicate negative correlation.

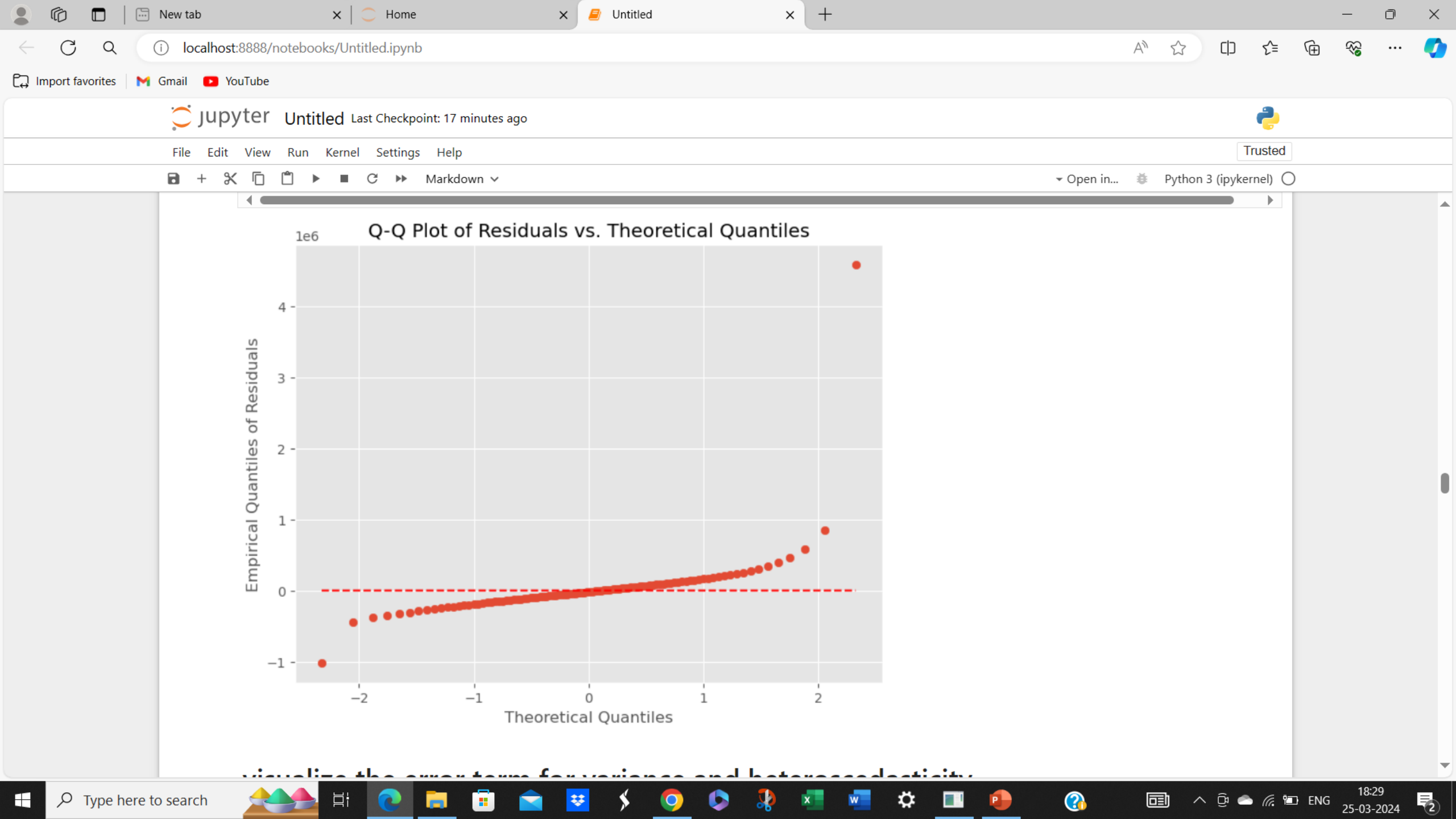
## Simple Regression Model with variables as a predictors

```
[22]: import statsmodels.api as sm

# Assuming 'df2' is your DataFrame with columns (features) and 'price' as the target variable
# Replace 'feature columns' with the actual list of feature columns in your DataFrame
```

# Regression model summary





```
# Print the summary of the regression results
print(model.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.520
Model:	OLS	Adj. R-squared:	0.520
Method:	Least Squares	F-statistic:	2339.
Date:	Sun, 24 Mar 2024	Prob (F-statistic):	0.00
Time:	22:42:02	Log-Likelihood:	-2.9946e+05
No. Observations:	21597	AIC:	5.989e+05
Df Residuals:	21586	BIC:	5.990e+05
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	-8.211e+05	6.84e+04	-12.012	0.000	-9.55e+05	-6.87e+05
sqft_living	6.909e+04	1e+04	6.887	0.000	4.94e+04	8.87e+04
sqft_lot	0.3794	0.052	7.365	0.000	0.278	0.480
floors	-1.025e+05	4628.289	-22.146	0.000	-1.12e+05	-9.34e+04
sqft_above	104.6051	4.710	22.209	0.000	95.373	113.837
sqft_living15	76.6256	4.311	17.774	0.000	68.176	85.076
sqft_lot15	-6.599e+04	3146.787	-20.971	0.000	-7.22e+04	-5.98e+04
condition	7.56e+04	2797.859	27.019	0.000	7.01e+04	8.11e+04
grade	1.261e+05	2618.622	48.166	0.000	1.21e+05	1.31e+05
bedrooms	-1.988e+04	2476.453	-8.027	0.000	-2.47e+04	-1.5e+04
bathrooms	3.903e+04	3846.776	10.145	0.000	3.15e+04	4.66e+04

Omnibus:	18928.147	Durbin-Watson:	1.983
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1703934.960
Skew:	3.798	Prob(JB):	0.00
Kurtosis:	45.846	Cond. No.	1.76e+06

Notes:

```
[38]: import pandas as pd

# Assuming 'df2' is your DataFrame with columns (features) and 'price' as the target variable

# Identify categorical and numerical columns
categorical_columns = ['condition', 'grade', 'bedrooms', 'bathrooms'] # List categorical column names
numerical_columns = ['sqft_living', 'sqft_lot15',] # List numerical column names

# Create dummy variables for categorical columns
df_dummies = pd.get_dummies(df2[categorical_columns], drop_first=True) # drop_first=True to avoid multicollinearity
df_with_dummies = pd.concat([df2.drop(categorical_columns, axis=1), df_dummies], axis=1)

# Perform log transformation on numerical columns
df_with_dummies[numerical_columns] = df_with_dummies[numerical_columns].apply(lambda x: np.log1p(x))

# Now, you can use df_with_dummies for further analysis with transformed features
```

```
[39]: import statsmodels.api as sm

# Assuming 'df_with_dummies' is your DataFrame with transformed features and 'price' as the target variable
# Replace 'feature_columns' with the actual list of transformed feature columns in your DataFrame

# Define the feature columns (including dummy variables and log-transformed numerical columns)
feature_columns = df_with_dummies.columns.drop('price')

# Add a constant term to the features for the intercept
X = sm.add_constant(df_with_dummies[feature_columns])

# Define the target variable
y = df_with_dummies['price']

# Fit the multiple linear regression model
```

## # model evaluation

```
[40]: # model evaluation

from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Predictions
y_pred = model.predict(X)

# R-squared
r_squared = model.rsquared
adjusted_r_squared = model.rsquared_adj

# Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)

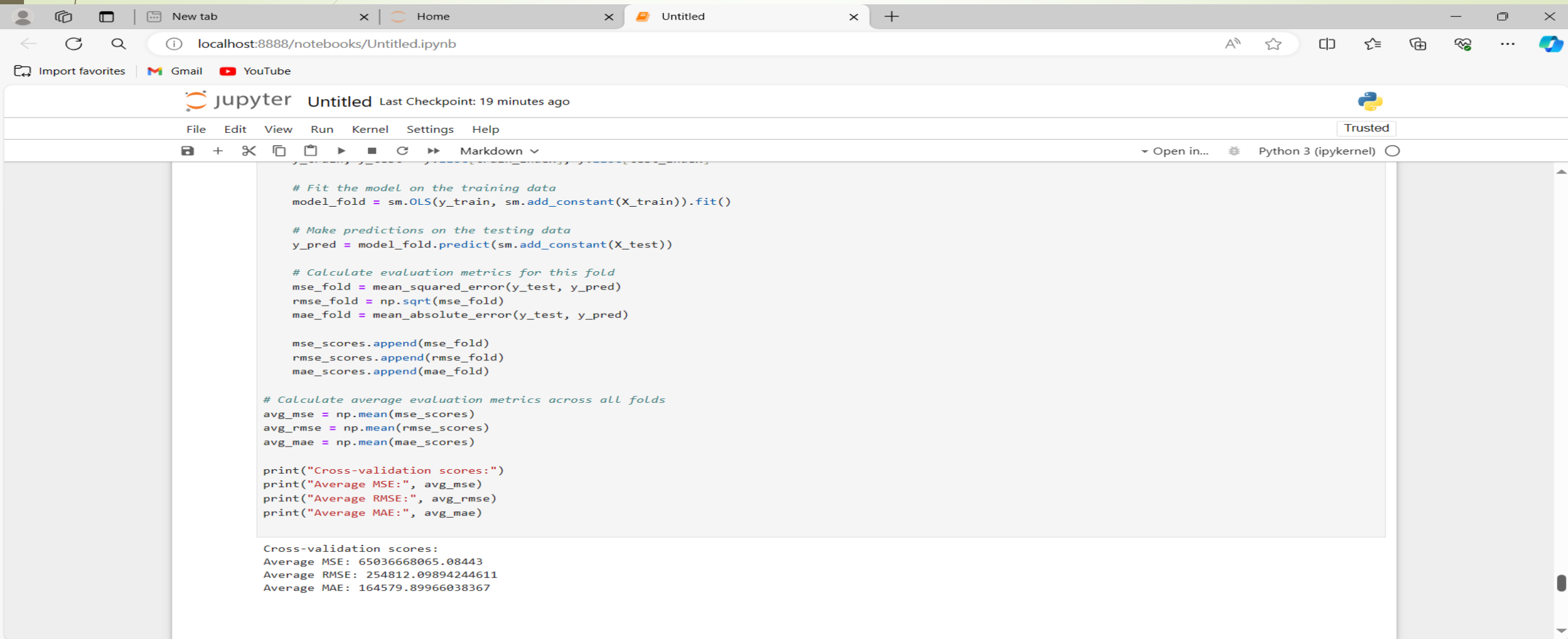
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y, y_pred)

print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")

R-squared: 0.520060479842454
Adjusted R-squared: 0.5198381415119816
Mean Squared Error (MSE): 64769326717.9641
Root Mean Squared Error (RMSE): 254498.18607990924
Mean Absolute Error (MAE): 164376.50418287885
```



# Model validation



The screenshot shows a Jupyter Notebook titled "Untitled" running on a local server at localhost:8888. The notebook contains Python code for fitting a linear model, making predictions, and calculating evaluation metrics (MSE, RMSE, MAE) across multiple folds. The output shows the cross-validation scores.

```
# Fit the model on the training data
model_fold = sm.OLS(y_train, sm.add_constant(X_train)).fit()

# Make predictions on the testing data
y_pred = model_fold.predict(sm.add_constant(X_test))

# Calculate evaluation metrics for this fold
mse_fold = mean_squared_error(y_test, y_pred)
rmse_fold = np.sqrt(mse_fold)
mae_fold = mean_absolute_error(y_test, y_pred)

mse_scores.append(mse_fold)
rmse_scores.append(rmse_fold)
mae_scores.append(mae_fold)

# Calculate average evaluation metrics across all folds
avg_mse = np.mean(mse_scores)
avg_rmse = np.mean(rmse_scores)
avg_mae = np.mean(mae_scores)

print("Cross-validation scores:")
print("Average MSE:", avg_mse)
print("Average RMSE:", avg_rmse)
print("Average MAE:", avg_mae)
```

Cross-validation scores:  
Average MSE: 65036668065.08443  
Average RMSE: 254812.09894244611  
Average MAE: 164579.89966038367

# Limitations of train test split

The screenshot displays a Jupyter Notebook interface with the following components:

- Browser:** The address bar shows `localhost:8888/notebooks/Untitled.ipynb`. The page title is "Untitled".
- Jupyter Notebook Header:** The header includes the Jupyter logo, the name "jupyter", and the notebook title "Untitled". It also shows "Last Checkpoint: 20 minutes ago".
- Menu Bar:** The menu bar includes "File", "Edit", "View", "Run", "Kernel", "Settings", and "Help".
- Toolbar:** The toolbar includes icons for "New", "Open", "Save", "Run", "Stop", "Restart", "Clear", "Undo", "Redo", and "Markdown".
- Code Cell:** The code cell contains the following Python code:

```
train_predictions = model.predict(sm.add_constant(X_train))
test_predictions = model.predict(sm.add_constant(X_test))

# Calculate evaluation metrics for the training and testing sets
train_mse = mean_squared_error(y_train, train_predictions)
test_mse = mean_squared_error(y_test, test_predictions)

train_rmse = np.sqrt(train_mse)
test_rmse = np.sqrt(test_mse)

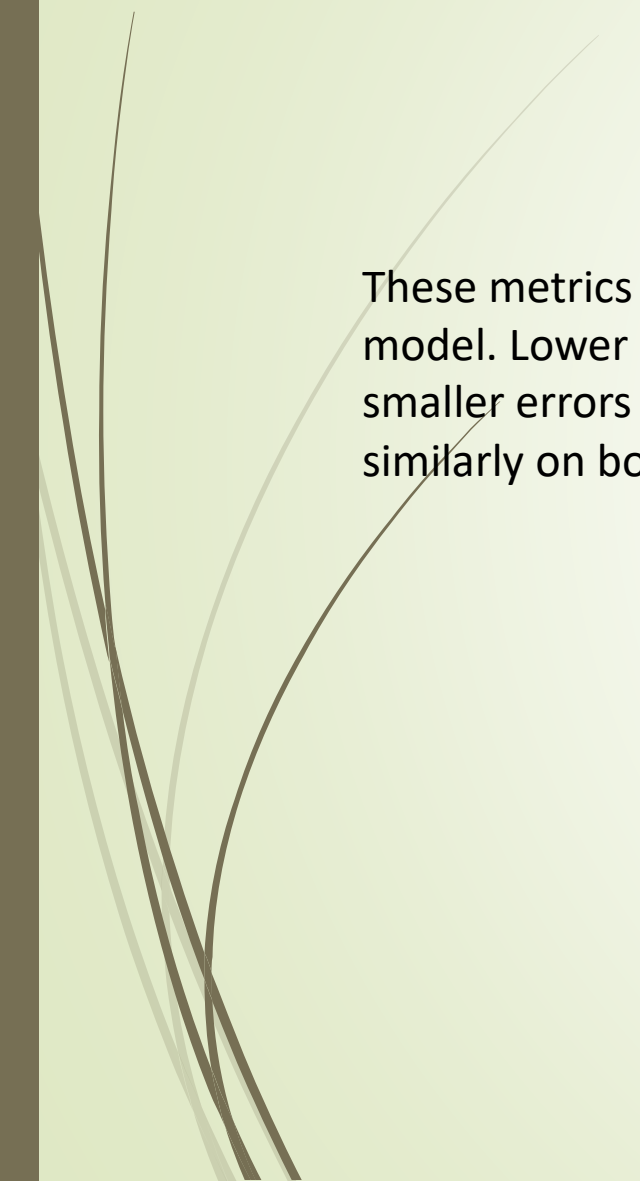

train_mae = mean_absolute_error(y_train, train_predictions)
test_mae = mean_absolute_error(y_test, test_predictions)

print("Performance on training data:")
print(f"Mean Squared Error (MSE): {train_mse}")
print(f"Root Mean Squared Error (RMSE): {train_rmse}")
print(f"Mean Absolute Error (MAE): {train_mae}")

print("\nPerformance on testing data:")
print(f"Mean Squared Error (MSE): {test_mse}")
print(f"Root Mean Squared Error (RMSE): {test_rmse}")
print(f"Mean Absolute Error (MAE): {test_mae}")
```
- Output:** The output of the code cell shows the performance metrics for both training and testing data:

```
Performance on training data:
Mean Squared Error (MSE): 64822231366.861595
Root Mean Squared Error (RMSE): 254602.10401106585
Mean Absolute Error (MAE): 165303.64189324272

Performance on testing data:
Mean Squared Error (MSE): 64682259118.974655
Root Mean Squared Error (RMSE): 254327.07114850092
Mean Absolute Error (MAE): 164294.18303530742
```
- Conclusion:** The output indicates that the model's performance is very similar on both training and testing data, which is a sign of overfitting.



These metrics provide insights into the overall performance of the multiple linear regression model. Lower values of MSE, RMSE, and MAE indicate better model performance, as they indicate smaller errors between the predicted and actual values. In this case, the model seems to perform similarly on both the training and testing datasets, which is a good indication of generalization.



# Thank you