US Housing Market Prediction

BIG DATA – HANDS ON PROJECT

USUC ALEXANDRU

Introduction

- The housing market plays a crucial role in the US economy, affecting everything from personal wealth to national financial stability
- Predicting housing prices is a challenge due to the vast amount of data and the complex factors involved, such as location, economy, demographics and market trends.

Objectives

The objectives for this project are:

- Analyze the large dataset with millions of records and over 3GB of real estate data
- Preprocess the data
- Generate helpful diagrams which will help us understand the dataset better
- Choose the relevant features for the training process

Dataset

- The current dataset contains information from year 2012 until 2024, having features such as:
  - period of time
  - region
  - state code
  - property type
  - median sale price
  - median list price
  - median days on the market
  - medina price per square foot
  - pending sales
  - homes sold, etc…
- Out of all 58 features that this dataset has, I chose just the relevant columns which will help the model predict the median sale price better.

region	city	state	state_code	property_type	homes_sold	homes_sold_mom
Chicago, IL	Chicago	Illinois	IL	Multi-Family (2-4 Unit)	284.0	-0.2303523035230352
Parsippany, NJ	Parsippany	New Jersey	NJ	All Residential	14.0	0.75
Oakbrook, KY	Oakbrook	Kentucky	KY	All Residential	17.0	0.1333333333333333
Dunstable, MA	Dunstable	Massachusetts	MA	All Residential	6.0	0.0
Kalamazoo, MI	Kalamazoo	Michigan	MI	All Residential	42.0	-0.4473684210526315
Tysons, VA	Tysons	Virginia	VA	Condo/Co-op	11.0	-0.3529411764705882
Myrtle Creek, OR	Myrtle Creek	Oregon	OR	Single Family Residential	5.0	1.5
Valencia West, AZ	Valencia West	Arizona	AZ	Single Family Residential	25.0	0.3157894736842106
Erda, UT	Erda	Utah	UT	All Residential	3.0	-0.25

median_sale_price	median_sale_price_mom	median_sale_price_yoy	median_list_price	median_dom	median_dom_mom
259500.0	0.0176470588235293	0.1662921348314607	285000.0	42.0	6.0
485000.0	-0.0673076923076922	-0.0351801179467439	477500.0	33.0	3.0
265000.0	-0.0363636363636363	0.4887640449438202	260000.0	5.0	2.0
522500.0	0.1578947368421053	-0.0141509433962264	529900.0	50.0	41.0
187500.0	0.2798634812286688	0.25	179900.0	22.0	-1.0
325000.0	0.0483870967741935	0.0420006412311637	439450.0	23.0	-64.0
308000.0	0.3508771929824561	0.4322250639386189	209500.0	18.0	6.0
159000.0	-0.1740259740259739	0.0192307692307691	163386.0	72.0	-55.0
630000.0	0.0637220139803464	-0.0631970260223048	699900.0	163.0	37.0

Tools

Tools used during the implementation of this project:

- Libraries
  - numpy
  - pandas
  - matplotlib
  - seaborn
  - sklearn
- Development environments
  - Pycharm
  - Jupyter notebook
  - Google colab

Implementation

The implementation process consists of multiple steps:

1. Analyze the dataset
2. Cleansing
3. Add new relevant columns
4. Normalize the data
5. Generate helpful diagrams
6. Choose the most relevant columns for the training process
7. Train the model

1. Analyze the data

2. Cleansing

- drop records with null features

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(0)
```
[1]

```python
file_path = "dataset/us_city_market_tracker.csv"

us_ds = pd.read_csv(file_path, sep='\t')
```
[2]

```python
us_ds = us_ds.dropna()
```
[4]

median_sale_price	median_sale_price_mom
35000	<null>
123145	0.21986131748390303
37478	-0.65616513761467887
142000	<null>
166945	<null>
121500	0.59474979491386382
949000	-0.66046511627906979
<null>	<null>
475000	-0.0083507306889353261
177500	-0.44357366771159878
119900	<null>

3. Add new relevant columns

```
us_ds['sale_to_list_ratio'] = us_ds['median_sale_price'] / us_ds['median_list_price']
us_ds['year'] = us_ds['period_begin'].dt.year
us_ds['month'] = us_ds['period_begin'].dt.month
us_ds['sales_to_new_listings_ratio'] = us_ds['homes_sold'] / us_ds['new_listings']
us_ds['price_increased_mom'] = (us_ds['median_sale_price_mom'] > 0).astype(int)
us_ds['price_increased_yoy'] = (us_ds['median_sale_price_yoy'] > 0).astype(int)
us_ds['inventory_turnover'] = us_ds['homes_sold'] / us_ds['inventory']
us_ds['sale_to_list_ppsf_ratio'] = us_ds['median_ppsf'] / us_ds['median_list_ppsf']
us_ds['supply_demand_balance'] = us_ds['new_listings'] - us_ds['pending_sales']
us_ds['fast_selling'] = (us_ds['median_dom'] <= 30).astype(int)
[7]
```

price_increased_yoy	inventory_turnover	sale_to_list_ppsf_ratio	supply_demand_balance	fast_selling
1	0.21580547112462006	0.517396987426864	55.0	0
0	0.7	0.9756112590691002	3.0	0
1	1.7	0.9250675723341937	3.0	1
0	0.4	1.1586413839521625	0.0	0
1	0.4158415841584158	0.9964415983172793	-8.0	1
1	0.12359550561797752	1.0343775541330438	16.0	1
1	0.45454545454545453	0.9206754735792623	1.0	1
1	0.26881720430107525	0.9594333074000767	-8.0	0
0	0.17647058823529413	0.9560499441946902	2.0	0
1	0.4308300395256917	0.9447901798458463	-3.0	1
1	3.8	0.9025138623310607	-2.0	1
1	0.5818181818181818	0.934585179102058	10.0	0
0	0.3	0.8344535203403132	0.0	1

4. Identify the outliers and replace them with moving average (MA)

```python
exclude_columns = ['period_duration', 'region_type_id', 'table_id', 'property_type_id',
'parent_metro_region_metro_code', 'year', 'month', 'price_increased_mom',
'price_increased_yoy', 'fast_selling']

# us_ds_ca = us_ds[us_ds['state_code'] == 'CA']
numeric_columns = us_ds.select_dtypes(include=['number']).columns

if exclude_columns:
    columns_for_outliers = [col for col in numeric_columns if col not in exclude_columns]
else:
    columns_for_outliers = numeric_columns

z_scores = us_ds[columns_for_outliers].apply(zscore)

# define a threshold for outliers
threshold = 5

# identify outliers
outliers = (z_scores.abs() > threshold)
```

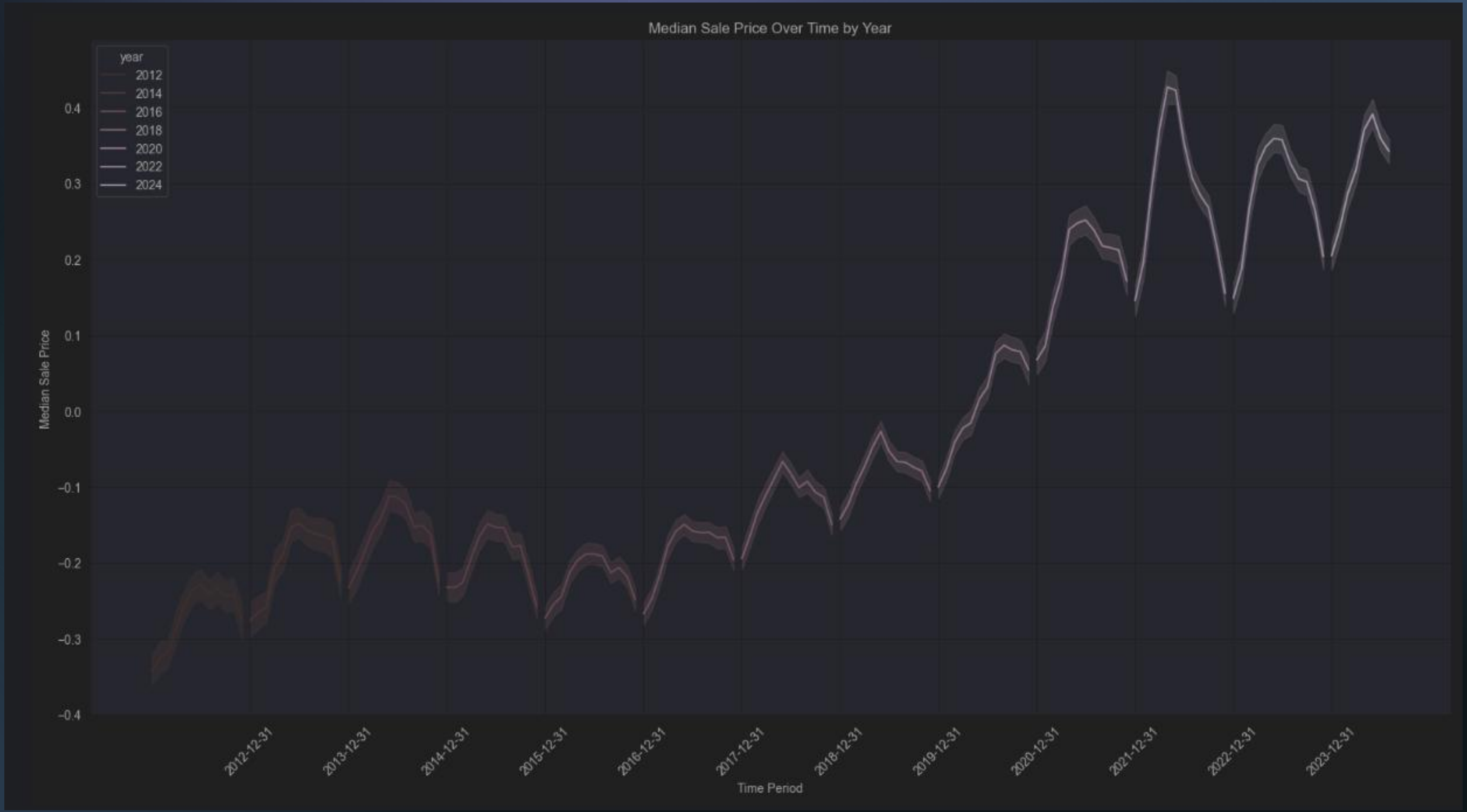
```python
for col in columns_for_outliers:
    # Calculate EMA for the column
    # ema = us_ds_ca_copy[col].ewm(span=10, adjust=False).mean()

    # calculate MA - moving average
    ma = us_ds_copy[col].rolling(window=2, center=True).mean()

    # Replace outliers with EMA
    # us_ds_ca_copy.loc[outliers[col], col] = ema[outliers[col]]
    us_ds_copy.loc[outliers[col], col] = ma[outliers[col]]
```

5. Normalize the data

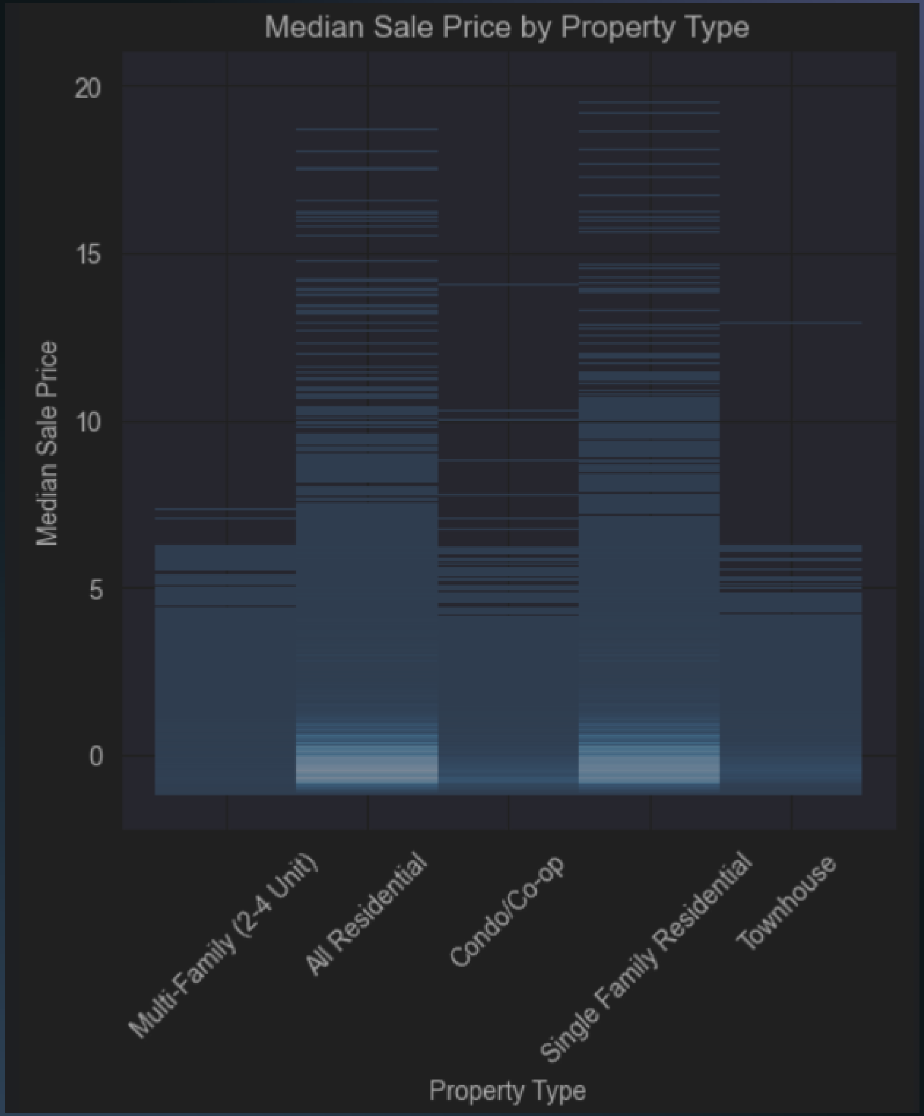
```python
def preprocess_entire_dataset(us_ds, mean, std):
    exclude_columns = ['period_duration', 'region_type_id', 'table_id', 'property_type_id',
    'parent_metro_region_metro_code', 'year', 'month', 'price_increased_mom',
    'price_increased_yoy', 'fast_selling']

    numeric_columns = us_ds.select_dtypes(include=['number']).columns

    if exclude_columns:
        columns_to_normalize = [col for col in numeric_columns if col not in exclude_columns]
    else:
        columns_to_normalize = numeric_columns

    # normalize
    print("Start normalizing...")
    print("In preprocess entire...")
    us_ds_normalized = us_ds.copy()
    us_ds_normalized[columns_to_normalize] = normalize(us_ds[columns_to_normalize], mean, std)
```

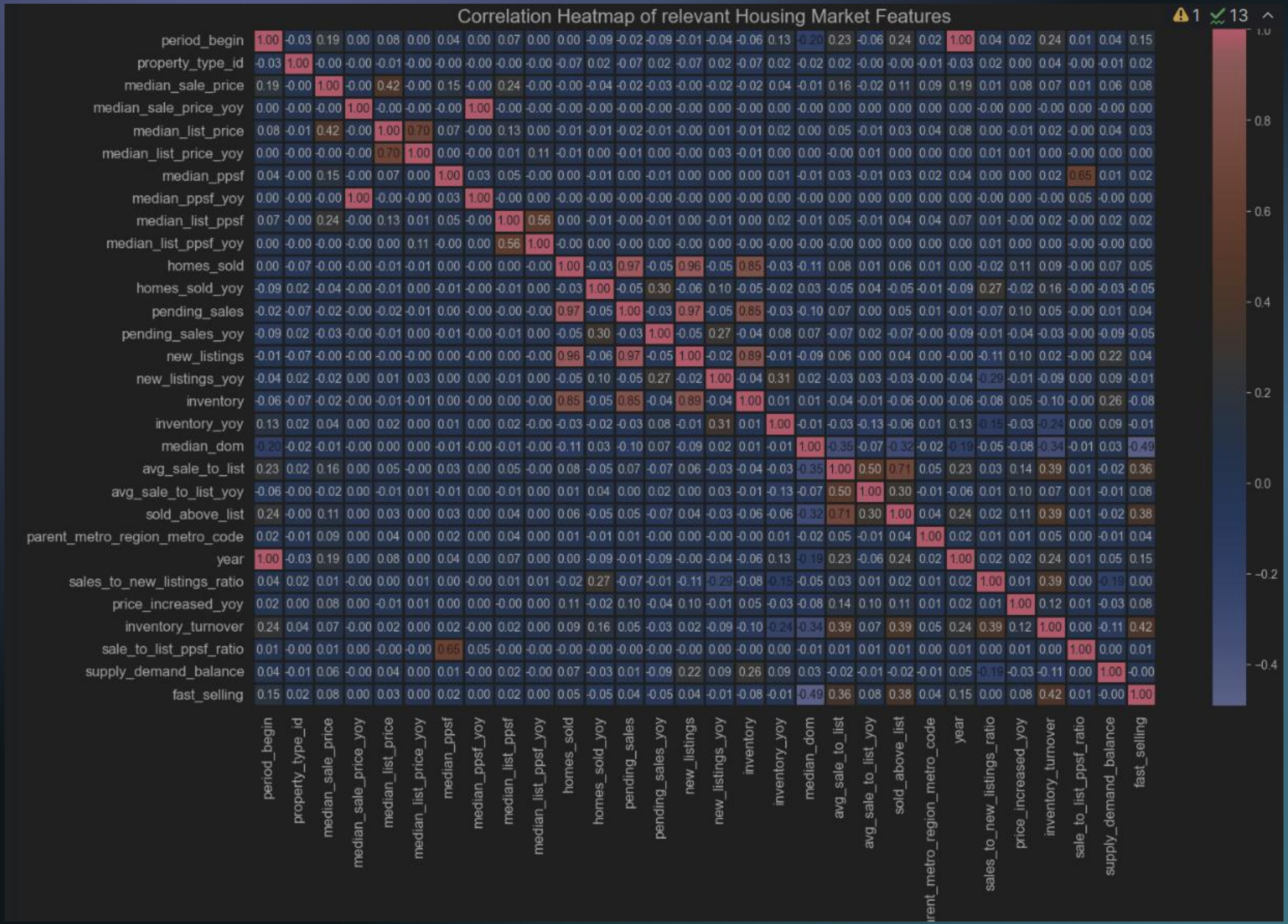
```python
def normalize(us_ds, mean=None, std=None):
    print("In normalize function...")
    us_ds_normalized = us_ds.copy()

    # Normalize only the selected numeric columns
    us_ds_normalized = (us_ds_normalized - mean) / std

    # us_ds_normalized = (us_ds - mean) / std
    return us_ds_normalized
```

6. Generate diagrams




Trend of Homes Sold and Inventory Over Time (Monthly)
Homes Sold
Inventory
Count
0.2
0.1
0.0
-0.1
2012
2014
2016
2018
2020
2022
2024
Date


Median Sale Price by Property Type
Median Sale Price
0
5
10
15
20
Multi-Family (2-4 Unit)
All Residential
Condo/Co-op
Single Family Residential
Townhouse
Property Type


Correlation Heatmap of relevant Housing Market Features
period_begin
property_type_id
median_sale_price
median_sale_price_yoy
median_list_price
median_list_price_yoy
median_ppsf
median_ppsf_yoy
median_list_ppsf
median_list_ppsf_yoy
homes_sold
homes_sold_yoy
pending_sales
pending_sales_yoy
new_listings
new_listings_yoy
inventory
inventory_yoy
median_dom
avg_sale_to_list
avg_sale_to_list_yoy
sold_above_list
parent_metro_region_metro_code
year
sales_to_new_listings_ratio
price_increased_yoy
inventory_turnover
sale_to_list_ppsf_ratio
supply_demand_balance
fast_selling
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4

7. Encoding string columns

```python
import category_encoders as ce

# Apply Binary Encoding
binary_encoder = ce.BinaryEncoder(cols=columns_for_encoding)
us_ds_binary = binary_encoder.fit_transform(us_ds)
```

state_code_0	state_code_1	state_code_2	state_code_3	state_code_4	state_code_5
0	0	0	0	0	1

property_type_0	property_type_1	property_type_2
0	0	1

8. Select the most relevant columns

```python
columns_for_model = [
    'state_code_0', 'state_code_1', 'state_code_2', 'state_code_3', 'state_code_4', 'state_code_5', 'property_type_0',
     'property_type_1', 'property_type_2', 'median_sale_price', 'median_dom', 'price_drops', 'inventory_turnover',
     'price_increased_mom', 'median_sale_price_mom', 'median_list_price', 'median_ppsf', 'median_list_ppsf',
     'avg_sale_to_list', 'sold_above_list', 'sale_to_list_ratio', 'sin_year', 'cos_year', 'sin_month', 'cos_month'
]
us_ds_for_model = us_ds[columns_for_model]
```

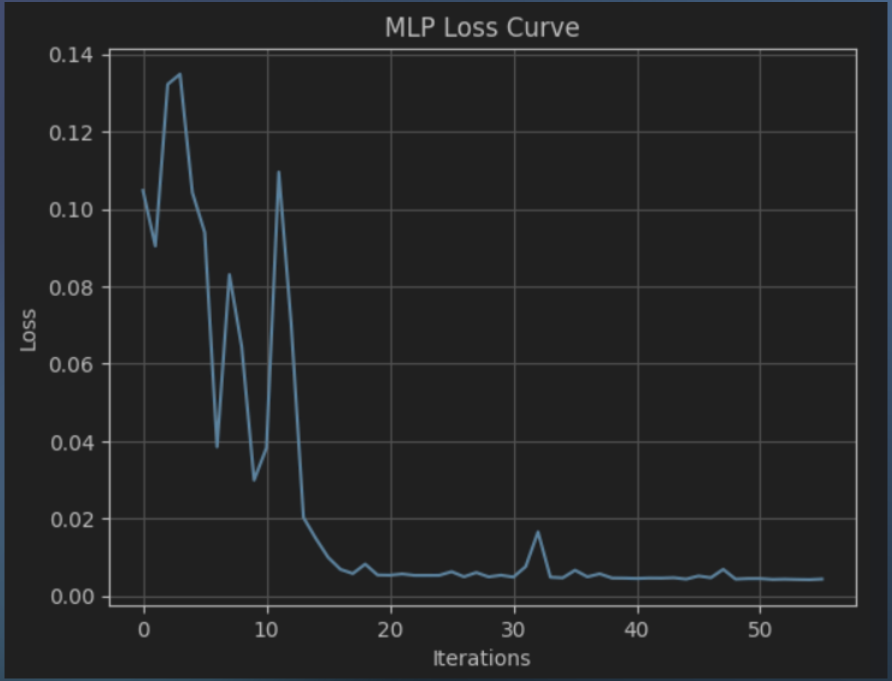
9. Model Training

```python
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
```
[10]

```python
y = us_ds['median_sale_price']

X = us_ds.drop(columns=['median_sale_price'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```
[11]

```python
mlp_model = MLPRegressor(hidden_layer_sizes=(64, 32), max_iter=200, random_state=42)
mlp_model.fit(X_train, y_train)
mlp_predictions = mlp_model.predict(X_test)
mlp_mse = mean_squared_error(y_test, mlp_predictions)
print(f"MLP Mean Squared Error: {mlp_mse}")
```
[13]

```
MLP Mean Squared Error: 0.012069494427731918
```

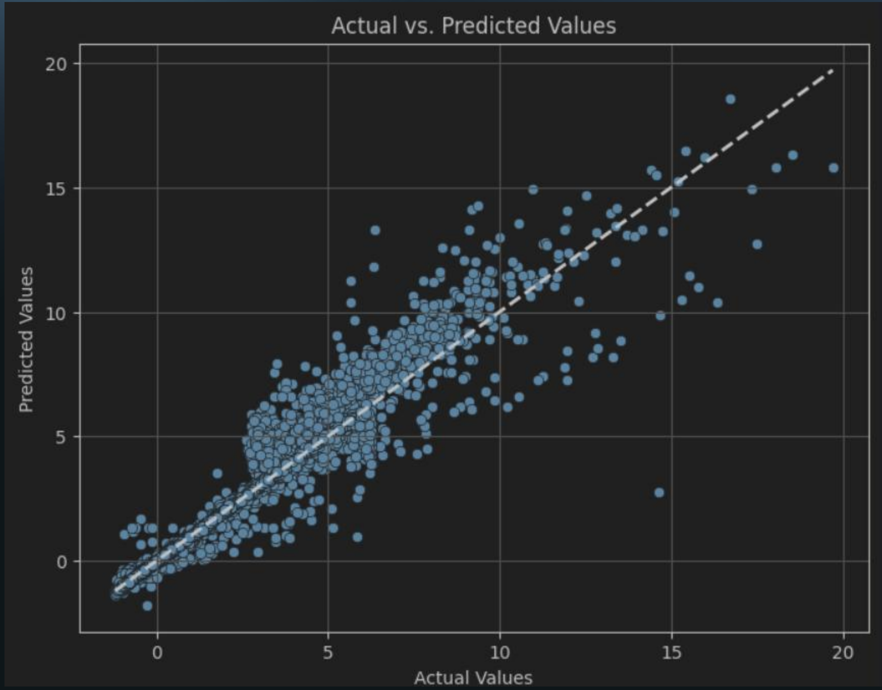



```python
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_test, mlp_predictions)
print(f"MAE: {mae}")
```
[41]

```
MAE: 0.022050769570323953
```

```python
from sklearn.metrics import r2_score

r2 = r2_score(y_test, mlp_predictions)
print(f"R²: {r2}")
```
[42]

```
R²: 0.9875581700356271
```



Bibliography

- https://contrib.scikit-learn.org/category_encoders/
- https://developer.nvidia.com/blog/three-approaches-to-encoding-time-information-as-features-for-ml-models/
- https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning
- https://towardsdatascience.com/multilayer-perceptron-explained-a-visual-guide-with-mini-2d-dataset-0ae8100c5d1c#:~:text=A%20Multilayer%20Perceptron%20%28MLP%29%20is%20a%20type%20of,connects%20to%20all%20nodes%20in%20the%20next%20layer.
- https://pythongeeks.org/data-preprocessing-in-machine-learning/

Thank you!