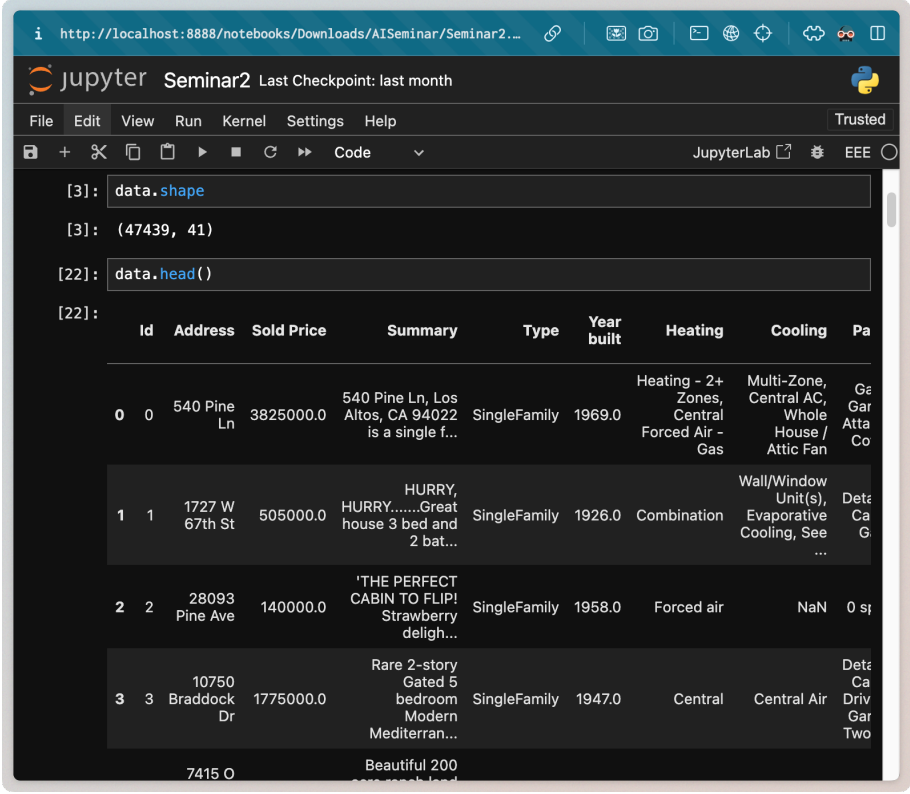


EEEE 3077 Artificial Intelligence Systems

Coursework 2024

- **Data Exploration:**

Firstly, read the data from the CSV, using the code above to explore the data. As shown, the data shape is (47439,41), which means that data has 47439 columns and 41 rows. The data head is shown, as we can see, the dataset contains information on real estate properties, including the price at which they were sold, address, summary, type of property, year built, heating, cooling, parking details, lot size, and more.



The screenshot shows a JupyterLab window titled 'Seminar2' with a 'Last Checkpoint: last month' status. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The code editor displays the following code:

```
[3]: data.shape
[3]: (47439, 41)
[22]: data.head()
```

The output of the `data.head()` command is a table showing the first five rows of the dataset. The table has columns: Id, Address, Sold Price, Summary, Type, Year built, Heating, Cooling, and Pa. The data is as follows:

	Id	Address	Sold Price	Summary	Type	Year built	Heating	Cooling	Pa
0	0	540 Pine Ln	3825000.0	540 Pine Ln, Los Altos, CA 94022 is a single f...	SingleFamily	1969.0	Heating - 2+ Zones, Central Forced Air - Gas	Multi-Zone, Central AC, Whole House / Attic Fan	Ge Gar Atta Co
1	1	1727 W 67th St	505000.0	HURRY.....Great house 3 bed and 2 bat...	SingleFamily	1926.0	Combination	Wall/Window Unit(s), Evaporative Cooling, See ...	Det: Ca G
2	2	28093 Pine Ave	140000.0	'THE PERFECT CABIN TO FLIP! Strawberry deligh...	SingleFamily	1958.0	Forced air	NaN	0 s
3	3	10750 Braddock Dr	1775000.0	Rare 2-story Gated 5 bedroom Modern Mediterran...	SingleFamily	1947.0	Central	Central Air	Det: Ca Driv Gar Two
		7415 O		Beautiful 200					

And then using the code to explore the data types, there are 2 main data types: numerical and categorical/object. For those numerical variables, except Id and Zip are int, others are in the type of float.

```
[25]: data.dtypes
```

```
[25]: Id                int64
      Address           object
      Sold Price        float64
      Summary           object
      Type              object
      Year built        float64
      Heating           object
      Parking           object
      Lot               float64
      Bedrooms          object
      Bathrooms         float64
      Full bathrooms    float64
      Total interior livable area float64
      Total spaces      float64
      Garage spaces     float64
      Region            object
      Elementary School object
      Elementary School Score float64
      Elementary School Distance float64
      High School       object
      High School Score float64
      High School Distance float64
      Flooring          object
      Heating features  object
      Appliances included object
      Parking features  object
      Tax assessed value float64
      Annual tax amount float64
      Listed On         object
```

Furthermore, find the distribution of each data types as shown below:

```
[7]: data.describe()
```

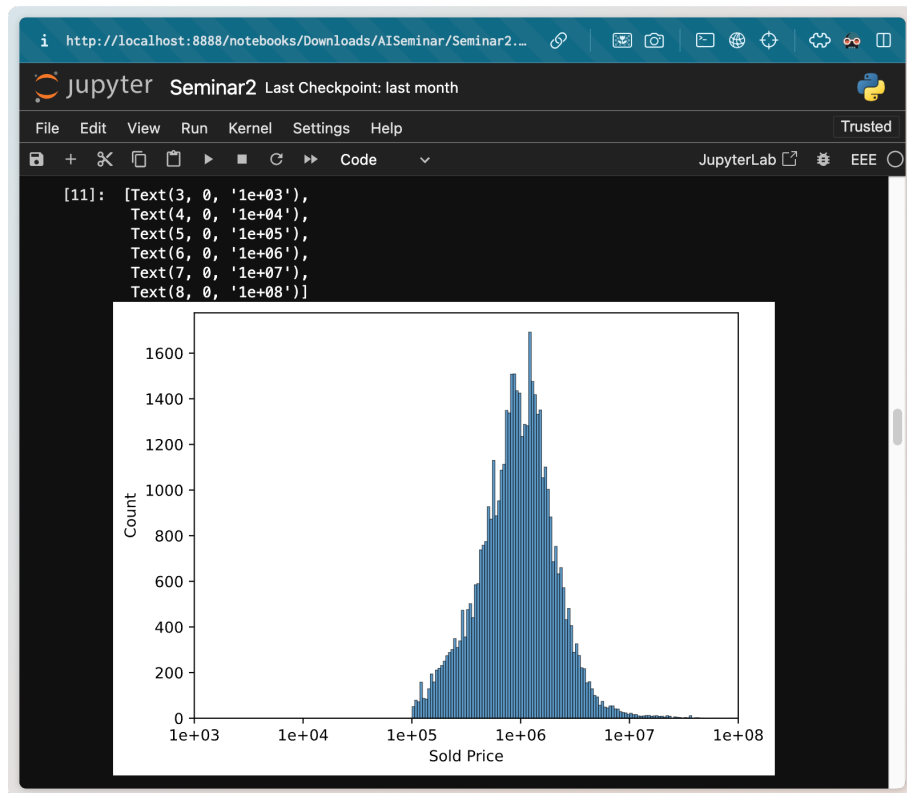
```
[7]:
```

	Id	Sold Price	Year built	Lot	Bathrooms	Full bathrooms
count	47439.000000	4.743900e+04	46394.000000	3.325800e+04	43974.000000	39574.000000
mean	23719.000000	1.296050e+06	1956.634888	2.353383e+05	2.355642	2.094961
std	13694.604047	1.694452e+06	145.802456	1.192507e+07	1.188805	0.963320
min	0.000000	1.005000e+05	0.000000	0.000000e+00	0.000000	1.000000
25%	11859.500000	5.650000e+05	1946.000000	4.991000e+03	2.000000	2.000000
50%	23719.000000	9.600000e+05	1967.000000	6.502000e+03	2.000000	2.000000
75%	35578.500000	1.525000e+06	1989.000000	1.045400e+04	3.000000	2.000000
max	47438.000000	9.000000e+07	9999.000000	1.897474e+09	24.000000	17.000000

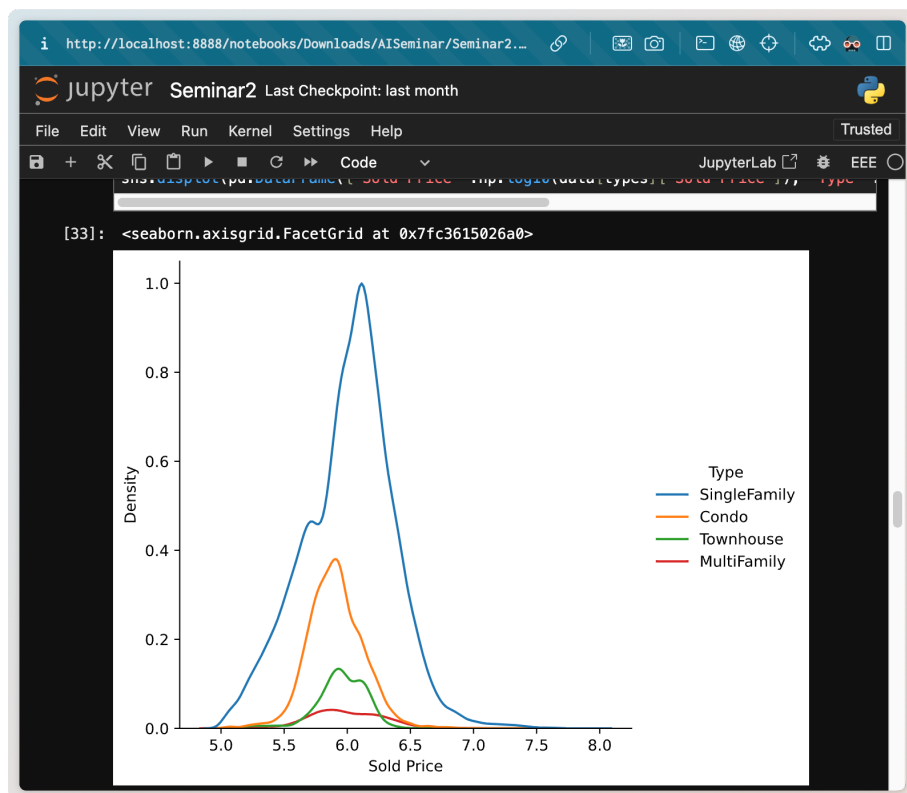
```
[11]: ax = sns.histplot(np.log10(data['Sold Price']))
      ax.set_xlim([3,8])
      ax.set_xticks(range(3,9))
      ax.set_xticklabels(['%.0e'%a for a in 10**ax.get_xticks()])
```

```
[11]: [Text(3, 0, '1e+03'),
      Text(4, 0, '1e+04'),
      Text(5, 0, '1e+05'),
      Text(6, 0, '1e+06'),
      Text(7, 0, '1e+07'),
```

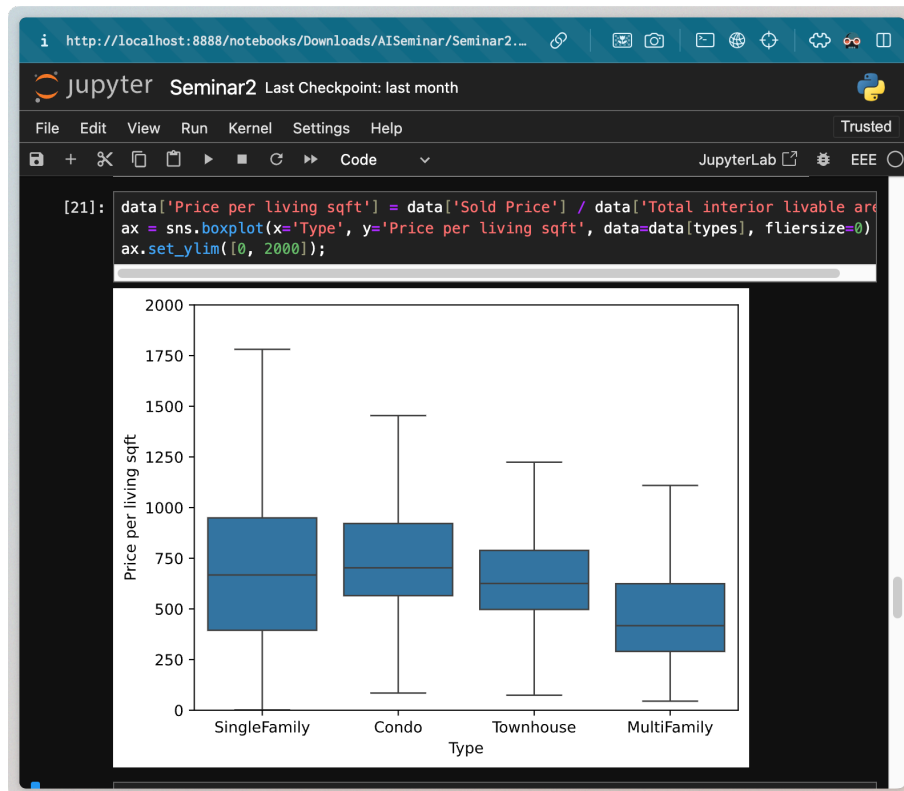
Below is the distribution for 'Sold Price' :



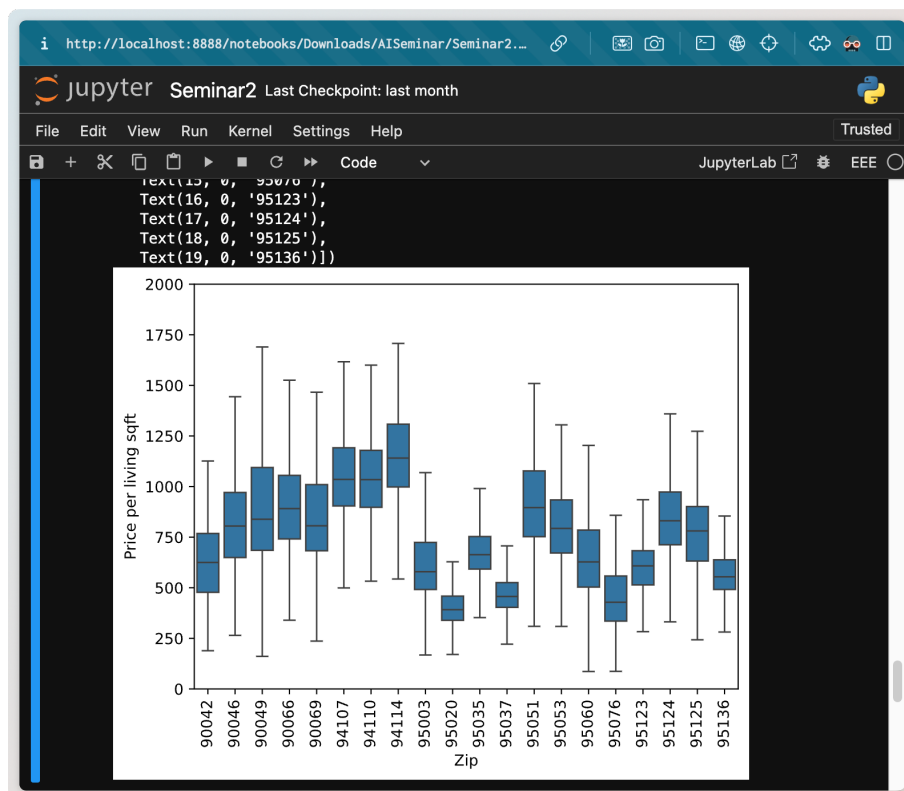
Below is the Kernel Density Estimation (KDE) for log-transformed 'Sold Price' of specific property types:



Below is the distribution of price per living square feet of different house types:

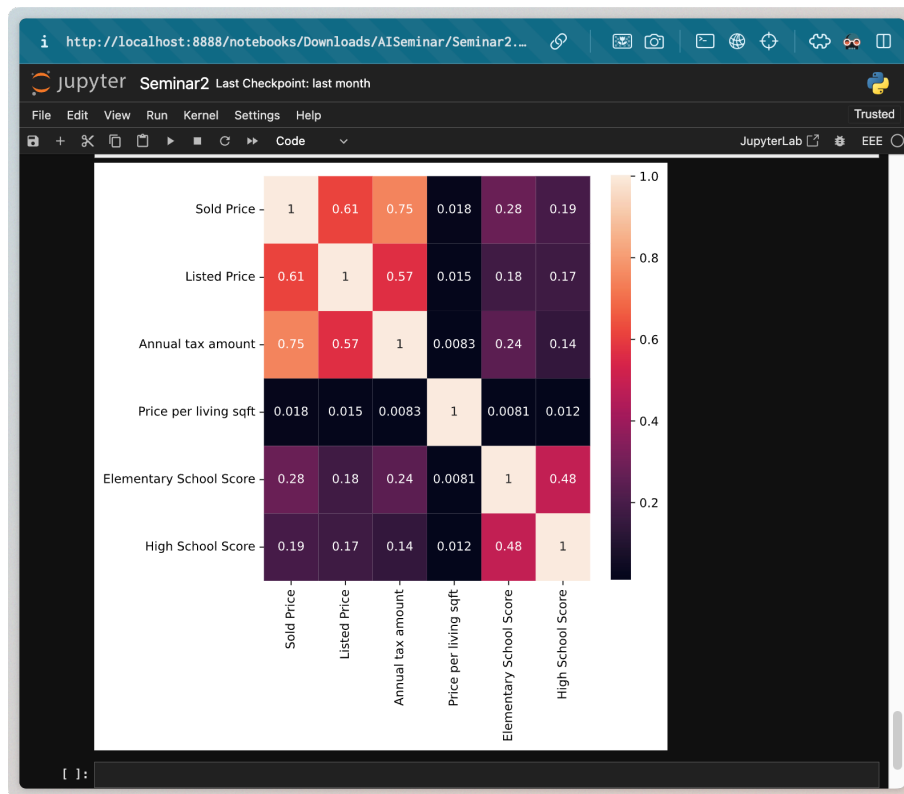


Below is the distribution of price per living square feet of different Zip:



Below is the correlation between these six factors 'Sold Price', 'Listed Price', 'Annual tax amount', 'Price

per living sqft', 'Elementary School Score', 'High School Score':



- **Data Preprocessing:**

Using the code shown below to remove features with too many NaN values.

```
[6]: data.drop(columns=data.columns[null_sum>len(data)*0.3],inplace=True)
```

Using the code below to remove the redundant features: 'Address', 'Summary', 'City', 'State'.

```
redundant_cols = ['Address', 'Summary', 'City', 'State']
for c in redundant_cols:
    del test_data[c], train_data[c]
```

Using the code below to using Log preprocess those big values: 'Lot', 'Total interior livable area', 'Tax assessed value', 'Annual tax amount', 'Listed Price'

```
large_vel_cols = ['Lot', 'Total interior livable area', 'Tax assessed va
for c in large_vel_cols:
    train_data[c] = np.log(train_data[c]+1)
    test_data[c] = np.log(test_data[c]+1)
```

Using the code below to filling 0 in missing values:

```
all_features = all_features.fillna(method='bfill', axis=0).fillna(0)
```

Using the code below to normalize the data:

```
all_features[numeric_features] = all_features[numeric_features].apply(lambda x: (x - x.mean()) / (x.std()))
```

And then, choose appropriate non-numerical features for the One-hot process. Display all the non-numeric objects and how many unique categories they have, as shown below.

```
for in_object in all_features.dtypes[all_features.dtypes=='object'].index:
    print(in_object.ljust(20), len(all_features[in_object].unique()))

Type                174
Heating             2658
Parking            9911
Bedrooms           277
Region            1258
Elementary School   3567
High School         921
Flooring           1738
Heating features    1761
Appliances included 11289
Parking features    9693
Listed On           2815
Cooling             640
Middle School       801
Cooling features    466
Laundry features    1946
Last Sold On        5844
```

I choose 'Type' and 'Bedrooms' features for one-hot because the unique categories they have are not too much and these features are more related to the sold price. The figure below shows the data shape difference between the one hot process.

```
print('before one hot', all_features.shape)
all_features = pd.get_dummies(all_features, dummy_na=True)
print('after one hot', all_features.shape)

before one hot (79065, 20)
after one hot (79065, 471)
```

Finally, using the codes below to split the data set into test part and validation part:

```

from sklearn.model_selection import train_test_split
#splitting train, validation, and test set
n_train = train_data.shape[0]
train_validation_features = torch.tensor(all_features[:n_train].values.tolist(), dtype=torch.float)
test_features = torch.tensor(all_features[n_train:].values.tolist(), dtype=torch.float)
train_validation_labels = torch.tensor(train_data['Sold Price'].values.reshape(-1, 1), dtype=torch.float)
#splitting to train and validation dataset
train_features, validation_features, train_labels, validation_labels = train_test_split(train_validation_features, train_validation_labels,

```

● Model Selection:

4 models are chosen to train a suitable regression model, and those are Linear Regression, Decision Tree Regression, Random Forest Regression that I have learned in class, and MLP (Multi-Layer Perceptron) Regression that I have self-learned which might be useful for this task.

Reasons for models' choice: Every model has specific characteristics and benefits, making them suitable for various aspects of predictive analysis.

1. Linear Regression is a simple model that assumes a linear relationship between input variables and the target variable. It is quite fast in training and provides a good baseline for performance comparison. The model is simple and interpretable, which is the most important thing in the initial stages of the analysis when an understanding of the influence of variables is as important as prediction.

2. Decision Tree Regression forms a model in the form of a tree structure. It divides the dataset into smaller subsets while at the same time an associated decision tree is incrementally developed. The key advantage of such a model is that it can capture non-linear patterns without the need to perform any transformation of features, making it quite robust in the face of complex datasets.

3. Random Forest Regression is an ensemble method that uses multiple decision trees to improve the predictive performance and control over-fitting. It is an ensemble of simple decision trees with flexibility, which gives us higher accuracy. Parameter tuning in Random Forest involves adjusting the number of trees, the depth of the trees, and the minimum number of samples required to split a node, which enhances the model's ability to generalize.

4. MLP Regression uses a neural network approach for regression tasks. It is particularly useful for the capturing of complex relationships in the data through its layers and neurons. The MLP model was set up with parameters such as the number of hidden layers and nodes, learning rate, and the number of iterations over the data (epochs). The learning rate is tuned to optimize the time of training as well as

accuracy of the model. Early stopping was implemented to prevent overfitting.

Models' implementation process: Each of these implementations was done with careful consideration of parameter tuning.

1. Linear Regression has very few critical hyperparameters, and default settings are used because there is nothing to tune. This model acts as a baseline by assuming there is a linear relationship between the independent and dependent variables.

2. Decision Tree Regression requires setting the `random_state` for reproducibility. Decision trees are useful for non-linear data, but they are prone to overfitting. Overfitting is usually overcome by pruning, which is not a parameter that needs tuning here.

3. Random Forest Regression is similar to decision tree in that it builds an ensemble of trees. Typically, this improves accuracy and robustness. Generally, parameters like `n_estimators` are set to 20 in this case to determine how many trees to build in the forest, and `min_samples_split` is set to 48, which means there has to be at least 48 samples to split an internal node. The `max_depth` is set to be unlimited, meaning the trees expand until all leaves are pure or contain less than `min_samples_split` samples.

4. MLP Regression is a type of neural network. It has parameters such as `hidden_layer_sizes`, with 256 neurons, `max_iter` set to unlimited iterations, and a learning rate initialization of 0.005. It also introduces techniques like `early_stopping`, which halts training when the validation score stops improving to prevent overfitting. Here is the code of each model and the parameters I chose, as well as the related codes:

```
# Define the models with customizable parameters
linear_model = LinearRegression()

decision_tree_model = DecisionTreeRegressor(
    max_depth=8, # Maximum depth of the tree
    min_samples_split=16, # Minimum number of samples required to split an internal node
    random_state=0
)
random_forest_model = RandomForestRegressor(
    n_estimators=20, # Number of trees in the forest
    max_depth=None, # Maximum depth of the tree
    min_samples_split=48, # Minimum number of samples required to split an internal node
    random_state=0
)
mlp_model = MLPRegressor(
    hidden_layer_sizes=(256,), # Number and size of the hidden layers
    max_iter=5000, # Maximum number of iterations
    solver='adam', # Solver for weight optimization
    learning_rate_init=0.005, # Initial learning rate
    early_stopping=True, # Whether to use early stopping to terminate training when validation score is not improving
    validation_fraction=0.1, # Proportion of training data to set aside as validation set for early stopping
    random_state=0
)
```

- **Model Validation:**

Demonstration of Model Performance: The Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error are used to evaluate each model. These metrics measure the accuracy of prediction and the ability of the models in generalizing, for the more complex datasets. By comparing these metrics across different models, one will be able to assess which model is the best in capturing the underlying patterns of the dataset without overfitting. Models like Random Forest and MLP typically tend to work better on larger and more complex datasets due to the ability of modeling non-linear relationships and interactions between features. However, they require a good amount of care and tuning to balance bias and variance adequately.

```
[28]: # Linear Regression
print("Linear Regression Model Evaluation:")

linear_results = train_evaluate(linear_model, train_features, train_labels, validation_features, validation_labels)
print(f" Train - MSE: {linear_results[0][0]}, RMSE: {linear_results[0][1]}, MAE: {linear_results[0][2]}")
print(f" Validation - MSE: {linear_results[1][0]}, RMSE: {linear_results[1][1]}, MAE: {linear_results[1][2]}\n")
print(f" Difference - MSE: {linear_results[0][0]-linear_results[1][0]}, RMSE: {linear_results[0][1]-linear_results[1][1]}, MAE: {linear_results[0][2]-linear_results[1][2]}")

Linear Regression Model Evaluation:
Train - MSE: 1582648000512.0, RMSE: 1258033.375, MAE: 493546.15625
Validation - MSE: 57870351597568.0, RMSE: 7607256.5, MAE: 791800.3125

Difference - MSE: -56287702286336.0, RMSE: -6349223.0, MAE: -298254.15625

[29]: # Decision Tree Regression
print("Decision Tree Model Evaluation:")

decision_tree_results = train_evaluate(decision_tree_model, train_features, train_labels, validation_features, validation_labels)
print(f" Train - MSE: {decision_tree_results[0][0]}, RMSE: {decision_tree_results[0][1]}, MAE: {decision_tree_results[0][2]}")
print(f" Validation - MSE: {decision_tree_results[1][0]}, RMSE: {decision_tree_results[1][1]}, MAE: {decision_tree_results[1][2]}\n")
print(f" Difference - MSE: {decision_tree_results[0][0]-decision_tree_results[1][0]}, RMSE: {decision_tree_results[0][1]-decision_tree_results[1][1]}, MAE: {decision_tree_results[0][2]-decision_tree_results[1][2]}")

Decision Tree Model Evaluation:
Train - MSE: 441587697453.7708, RMSE: 664520.6523907069, MAE: 141671.29821131565
Validation - MSE: 931156285906.1067, RMSE: 964964.3961857384, MAE: 167452.58658047416

Difference - MSE: -489568588452.3359, RMSE: -300443.74379503145, MAE: -25781.288369158516

[30]: # Random Forest Regression
print("Random Forest Model Evaluation:")

random_forest_results = train_evaluate(random_forest_model, train_features, train_labels, validation_features, validation_labels)
print(f" Train - MSE: {random_forest_results[0][0]}, RMSE: {random_forest_results[0][1]}, MAE: {random_forest_results[0][2]}")
print(f" Validation - MSE: {random_forest_results[1][0]}, RMSE: {random_forest_results[1][1]}, MAE: {random_forest_results[1][2]}\n")
print(f" Difference - MSE: {random_forest_results[0][0]-random_forest_results[1][0]}, RMSE: {random_forest_results[0][1]-random_forest_results[1][1]}, MAE: {random_forest_results[0][2]-random_forest_results[1][2]}")

Random Forest Model Evaluation:
Train - MSE: 520293391717.3658, RMSE: 721313.6569602477, MAE: 123053.45506929864
Validation - MSE: 710399086135.8394, RMSE: 842851.7581021228, MAE: 149900.95975540808

Difference - MSE: -190105694418.47357, RMSE: -121538.10114187514, MAE: -26847.504686109445
```

```
[31]: # MLP Regression
print("MLP Model Evaluation:")

mlp_results = train_evaluate(mlp_model, train_features, train_labels, validation_features, validation_labels)
print(f" Train - MSE: {mlp_results[0][0]}, RMSE: {mlp_results[0][1]}, MAE: {mlp_results[0][2]}")
print(f" Validation - MSE: {mlp_results[1][0]}, RMSE: {mlp_results[1][1]}, MAE: {mlp_results[1][2]}\n")
print(f" Difference - MSE: {mlp_results[0][0]-mlp_results[1][0]}, RMSE: {mlp_results[0][1]-mlp_results[1][1]}, MAE: {mlp_results[0][2]-mlp_results[1][2]}")

MLP Model Evaluation:
Train - MSE: 709725229335.2216, RMSE: 842451.9151472216, MAE: 173406.82142168185
Validation - MSE: 824573654627.4146, RMSE: 908060.3804964814, MAE: 174901.33554506625

Difference - MSE: -114848425292.193, RMSE: -65608.46534925979, MAE: -1494.5141233844842
```

Linear Regression Model: The Linear Regression model has a fairly high difference in error metrics between train and validation datasets:

Mean Squared Error (MSE): It is very high in the validation set compared to the train set. This could be

an indication of overfitting of the model on the training data, where the model performs well on the train data but poorly on unseen data.

Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE): Both metrics follow the trend of the MSE, with RMSE and MAE showing a significant increment in the validation set compared to the train set, which may indicate that the model's predictions for the new data are not accurate.

Decision Tree Model: The Decision Tree model fits the training data with a much better performance than the Linear Model: MSE, RMSE, and MAE on Training mean that the model has learned the training data well.

Validation Metrics: The MSE, RMSE, and MAE on the validation data are substantially greater compared to the training, which leads us to consider a little bit of overfitting though the parameters are tuned to get this best performance.

Random Forest Model: The Random Forest model leaves far better prediction accuracy on the validation set than the Decision Tree:

Training Metrics: MSE, RMSE, and MAE tell us that the model has captured complex patterns in the data without fitting noise.





Validation Metrics: Although higher than training errors, the MSE, RMSE, and MAE are smaller than those of the single Decision Tree, and thus, it shows that the model's performance has improved, and it is doing less overfitting.

MLP: MLP Regressor gives a medium performance:

Training and Validation Metrics: Both datasets have higher MSE, RMSE, and MAE compared with the Random Forest model and Decision tree model, with a slightly higher error in the validation dataset. This is indicative of the model not being able to fit the data that well, possibly due to the complexity of the model and the need for more training or hyperparameter tuning.

Summary: Among the chosen models, the Random Forest is the most robust, achieving a good balance between the model's bias and variance. The Linear Regression model does not perform so well.

The Kaggle’s submission was used to validate the models, results of the 4 models are shown below, also shows that the Random Forest Model has the best performance:

Submission and Description		Private Score ⓘ	Public Score ⓘ
 Linear.csv Complete (after deadline) · now		3.94709	3.96183
 Decision_Tree_model.csv Complete (after deadline) · 1h ago		0.18977	0.19174
 Random_forest_model.csv Complete (after deadline) · 10m ago		0.17023	0.18442
 MLP_model.csv Complete (after deadline) · now		0.24569	0.27279