**Introduction**

In this project, We have used various python libraries and jupyter notebook to work on predicting the house price by the following features:

- Year of sale of the house
- The age of the house at the time of sale
- Distance from city center
- Number of stores in the locality
- The latitude
- The longitude

https://www.kaggle.com/datasets/yasserh/housing-prices-dataset?resource=download

```
pip install utils
```

```
pip install numpy pandas scikit-learn
```

```
pip install pandas
```

```
pip install tensorflow
```

```
#Importing Libraries
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns

from utils import *
from sklearn.metrics import precision_score, recall_score, accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, LambdaCallback

%matplotlib inline
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

print('Libraries are imported.')
```

```
Libraries are imported.
```

**Importing Data:**

```
#Importing datasets
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data.csv to data.csv
```

```
#Fetching the data set
df = pd.read_csv('data.csv')
print(df)
```

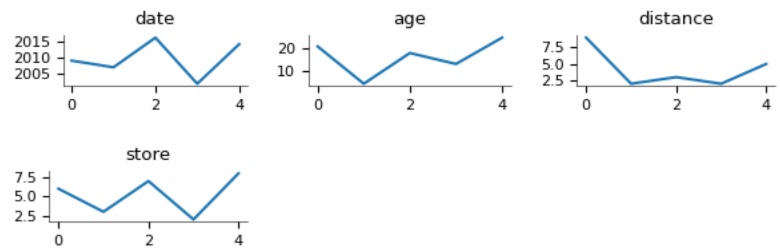
```
      date  age  distance  store  latitude  longtitude  price
0     2009   21         9      6        84         121  14264
1     2007    4         2      3        86         121  12032
2     2016   18         3      7        90         120  13560
3     2002   13         2      2        80         128  12029
4     2014   25         5      8        81         122  14157
...    ...  ...       ...    ...       ...         ...    ...
4995  2007   17         6      3        90         125  13539
4996  2016    7        10      0        85         129  14757
4997  2017    6        10      5        90         125  14102
4998  2010   37         3      5        81         128  14313
4999  2018    9         1      9        90         127  12770
```

```
[5000 rows x 7 columns]
```

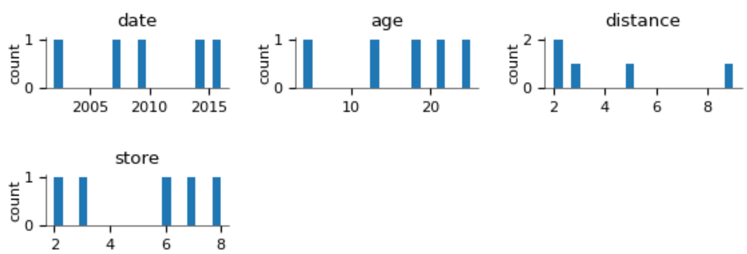
```
df.head()
```

| | date | age | distance | store | latitude | longtitude | price |
|---|---|---|---|---|---|---|---|
| **0** | 2009 | 21 | 9 | 6 | 84 | 121 | 14264 |
| **1** | 2007 | 4 | 2 | 3 | 86 | 121 | 12032 |
| **2** | 2016 | 18 | 3 | 7 | 90 | 120 | 13560 |
| **3** | 2002 | 13 | 2 | 2 | 80 | 128 | 12029 |
| **4** | 2014 | 25 | 5 | 8 | 81 | 122 | 14157 |

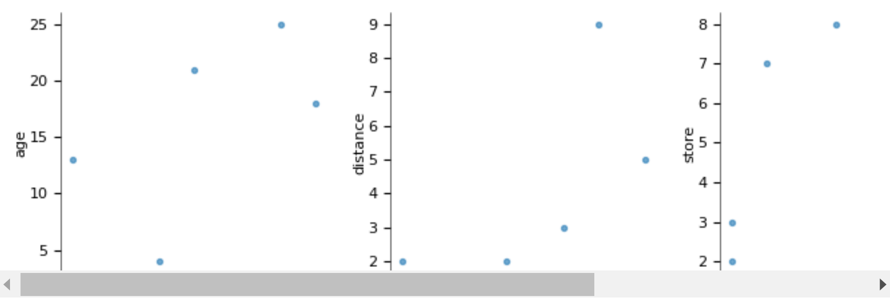
**Values**




**Distributions**

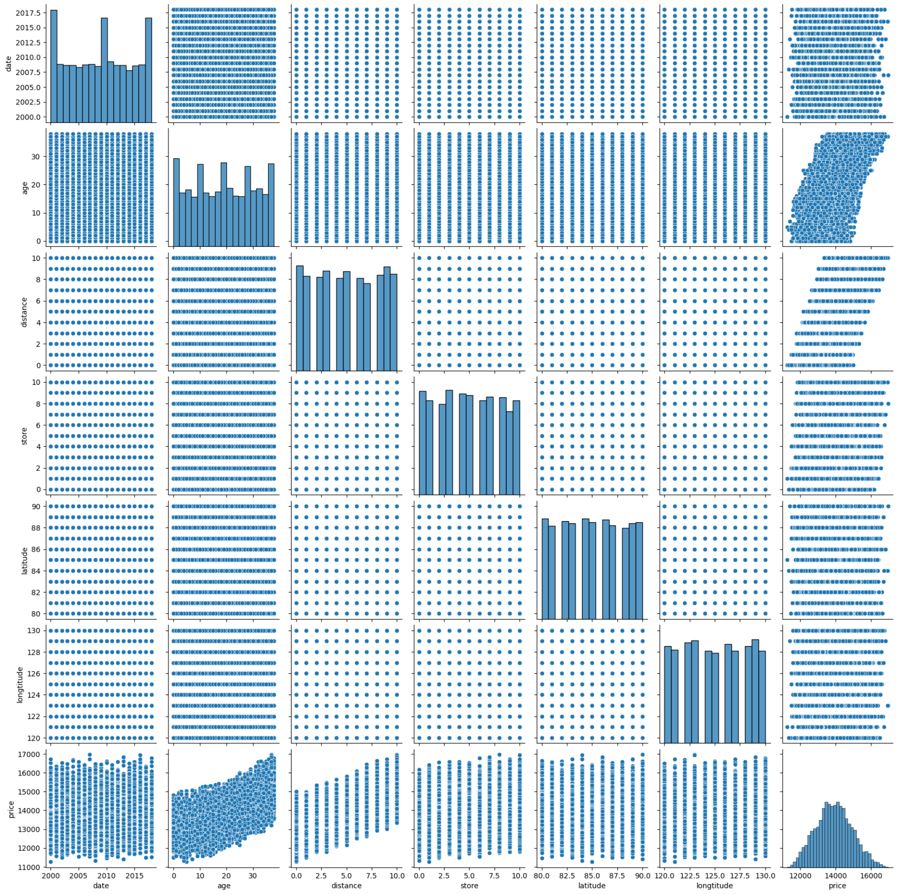



**2-d distributions**



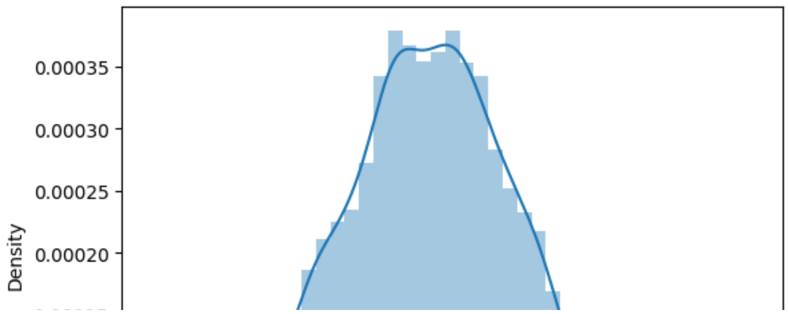

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7b3142fac5e0>
```



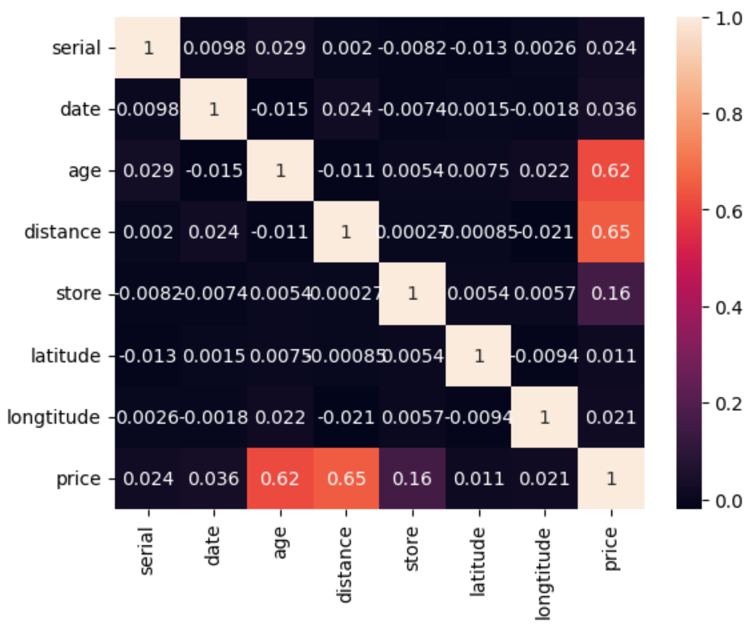
```
sns.distplot(df['price'])
```

```
<ipython-input-22-86c1ddc3c66a>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['price'])
<Axes: xlabel='price', ylabel='Density'>
```




```
sns.heatmap(df.corr(),annot = True) #Just to see the correlation between the features and the label
```

```
<Axes: >
```




### Checking For Missing Data:

```
#Checking For Missing Data
df.isna().sum()
```

```
serial        0
date          0
age           0
distance      0
store         0
latitude      0
longtitude    0
price         0
dtype: int64
```

### Data Normalization

Normalizing the data to bring all the different features to a similar range to make it easier for optimization algorithms to find minimas.

```
#Normalizing the data to bring all the different features to a similar range
#to make it easier for optimization algorithms to find minimas.
df = df.iloc[:,1:]
```

```
df_norm = (df - df.mean()) / df.std()
df_norm.head()
```

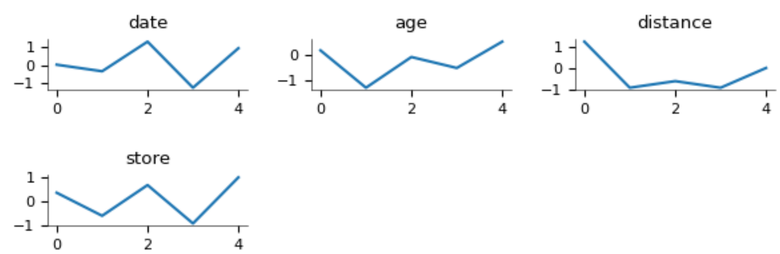
| | date | age | distance | store | latitude | longtitude | price |
|---|---|---|---|---|---|---|---|
| **0** | 0.015978 | 0.181384 | 1.257002 | 0.345224 | -0.307212 | -1.260799 | 0.350088 |
| **1** | -0.350485 | -1.319118 | -0.930610 | -0.609312 | 0.325301 | -1.260799 | -1.836486 |
| **2** | 1.298598 | -0.083410 | -0.618094 | 0.663402 | 1.590328 | -1.576456 | -0.339584 |
| **3** | -1.266643 | -0.524735 | -0.930610 | -0.927491 | -1.572238 | 0.948803 | -1.839425 |
| **4** | 0.932135 | 0.534444 | 0.006938 | 0.981581 | -1.255981 | -0.945141 | 0.245266 |

```
#Convert Label Value Back To Original:
y_mean = df['price'].mean()
y_std = df['price'].std()

def convert_label_value(pred):
    return int(pred * y_std + y_mean)
```

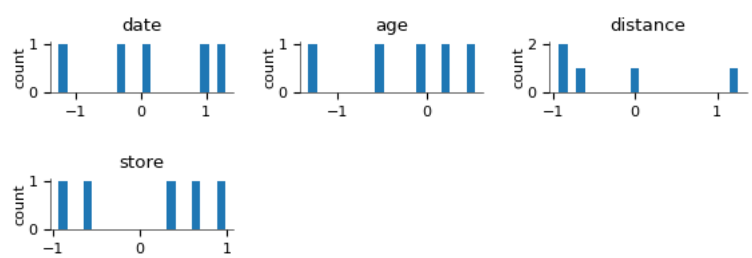
```
#Creating Training and Test Sets:

X = df_norm.iloc[:, :6] #Storing the features in 'X'
X.head()
```

| | date | age | distance | store | latitude | longtitude |
|---|---|---|---|---|---|---|
| **0** | 0.015978 | 0.181384 | 1.257002 | 0.345224 | -0.307212 | -1.260799 |
| **1** | -0.350485 | -1.319118 | -0.930610 | -0.609312 | 0.325301 | -1.260799 |
| **2** | 1.298598 | -0.083410 | -0.618094 | 0.663402 | 1.590328 | -1.576456 |
| **3** | -1.266643 | -0.524735 | -0.930610 | -0.927491 | -1.572238 | 0.948803 |
| **4** | 0.932135 | 0.534444 | 0.006938 | 0.981581 | -1.255981 | -0.945141 |

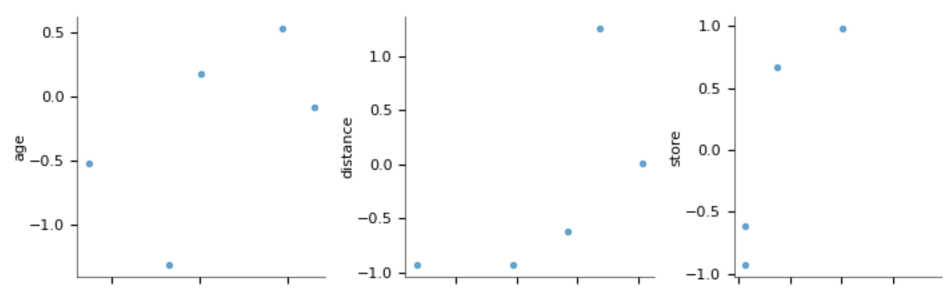
**Values**




**Distributions**




**2-d distributions**



```
Y = df_norm.iloc[:, -1] #Storing the labels in 'Y'
Y.head()
```

```
0    0.350088
1   -1.836486
2   -0.339584
3   -1.839425
4    0.245266
Name: price, dtype: float64
```

```
X_arr = X.values
Y_arr = Y.values

print('X_arr shape: ', X_arr.shape) #'shape' gives the dimension of the entity
print('Y_arr shape: ', Y_arr.shape)
```

```
X_arr shape:  (5000, 6)
Y_arr shape:  (5000,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_arr, Y_arr, test_size = 0.05, shuffle = True, random_state=0)
print('X_train shape: ', X_train.shape)
print('y_train shape: ', y_train.shape)
print('X_test shape: ', X_test.shape)
print('y_test shape: ', y_test.shape)
```

```
X_train shape:  (4750, 6)
y_train shape:  (4750,)
X_test shape:  (250, 6)
y_test shape:  (250,)
```

**Definig Model and Training data sets:**

```
#Creating the Neural Network Model

def get_model():

    model = Sequential([
        Dense(10, input_shape = (6,), activation = 'relu'), #10 neurons, Input Layer
        Dense(20, activation = 'relu'),                     #20 neurons, Hidden Layer
        Dense(5, activation = 'relu'),                      #5  neurons, Hidden Layer
        Dense(1)                                            #Output Layer
    ])                                                      #'relu' activation

    model.compile(
        loss='mse',                                         #Trained using Mean square error loss (Cost function)
        optimizer='adam'                                    #Optimizer used is 'adam' (One of the Fastest optimizers)
    )

    return model

model = get_model()
model.summary()
```

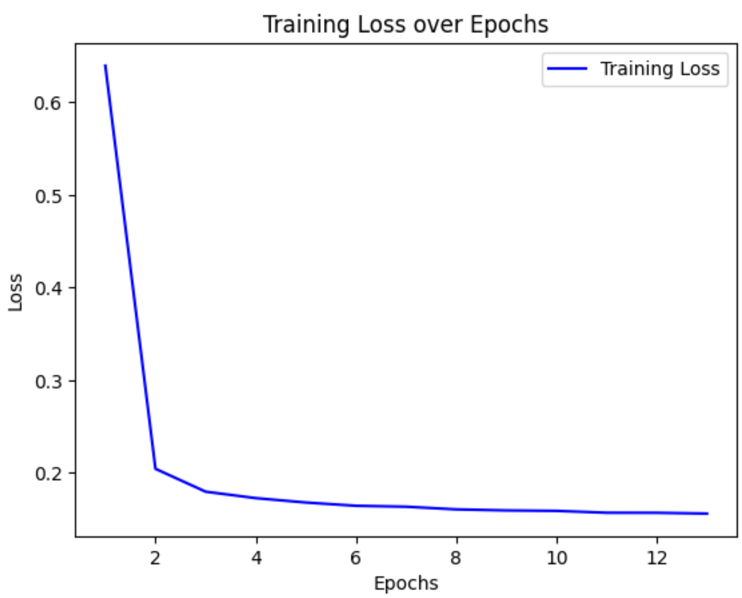
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 10)                70

 dense_9 (Dense)             (None, 20)                220

 dense_10 (Dense)            (None, 5)                 105

 dense_11 (Dense)            (None, 1)                 6

=================================================================
Total params: 401
Trainable params: 401
Non-trainable params: 0
_________________________________________________________________
```

```
#Trainig the dataset into the model
early_stopping = EarlyStopping(monitor='val_loss', patience = 5) #Defining early stopping parameter (optional, to save time)

model = get_model()
```

```
preds_on_untrained = model.predict(X_test) #Make predictions on the test set before training the parameters

#Finally training the model-->
history = model.fit(
    X_train, y_train,
    validation_data = (X_test, y_test),
    epochs = 100,
    callbacks = [early_stopping]
)
```

```
8/8 [==============================] - 0s 2ms/step
Epoch 1/100
149/149 [==============================] - 2s 5ms/step - loss: 0.5439 - val_loss: 0.2443
Epoch 2/100
149/149 [==============================] - 0s 3ms/step - loss: 0.2028 - val_loss: 0.1632
Epoch 3/100
149/149 [==============================] - 0s 3ms/step - loss: 0.1755 - val_loss: 0.1520
Epoch 4/100
149/149 [==============================] - 0s 3ms/step - loss: 0.1695 - val_loss: 0.1476
Epoch 5/100
149/149 [==============================] - 0s 3ms/step - loss: 0.1653 - val_loss: 0.1514
Epoch 6/100
149/149 [==============================] - 0s 3ms/step - loss: 0.1627 - val_loss: 0.1491
Epoch 7/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1610 - val_loss: 0.1538
Epoch 8/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1603 - val_loss: 0.1457
Epoch 9/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1584 - val_loss: 0.1499
Epoch 10/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1572 - val_loss: 0.1502
Epoch 11/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1566 - val_loss: 0.1484
Epoch 12/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1560 - val_loss: 0.1553
Epoch 13/100
149/149 [==============================] - 0s 2ms/step - loss: 0.1560 - val_loss: 0.1493
```

```
# Finding The model accuracy:
def plot_loss(history):
    loss = history.history['loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'b-', label='Training Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training Loss over Epochs')
    plt.legend()
    plt.show()


plot_loss(history)
```




```
from tensorflow.keras.models import Model
```

**Predicting the Price:**

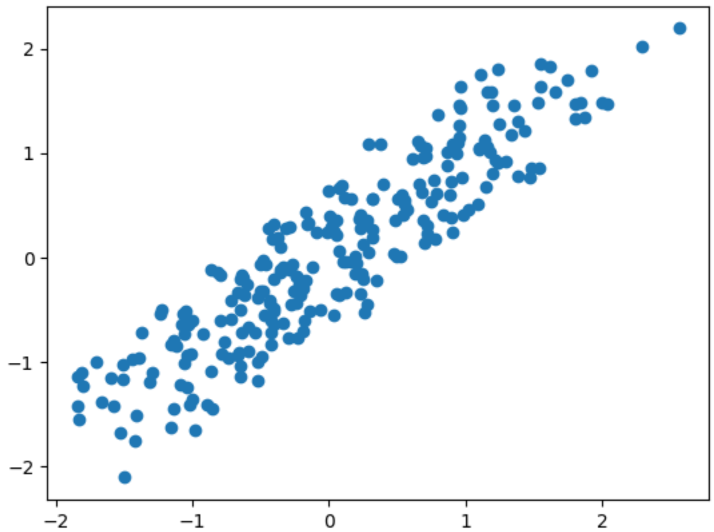
```
predictions = model.predict(X_test)
```

```
8/8 [==============================] - 0s 3ms/step
```

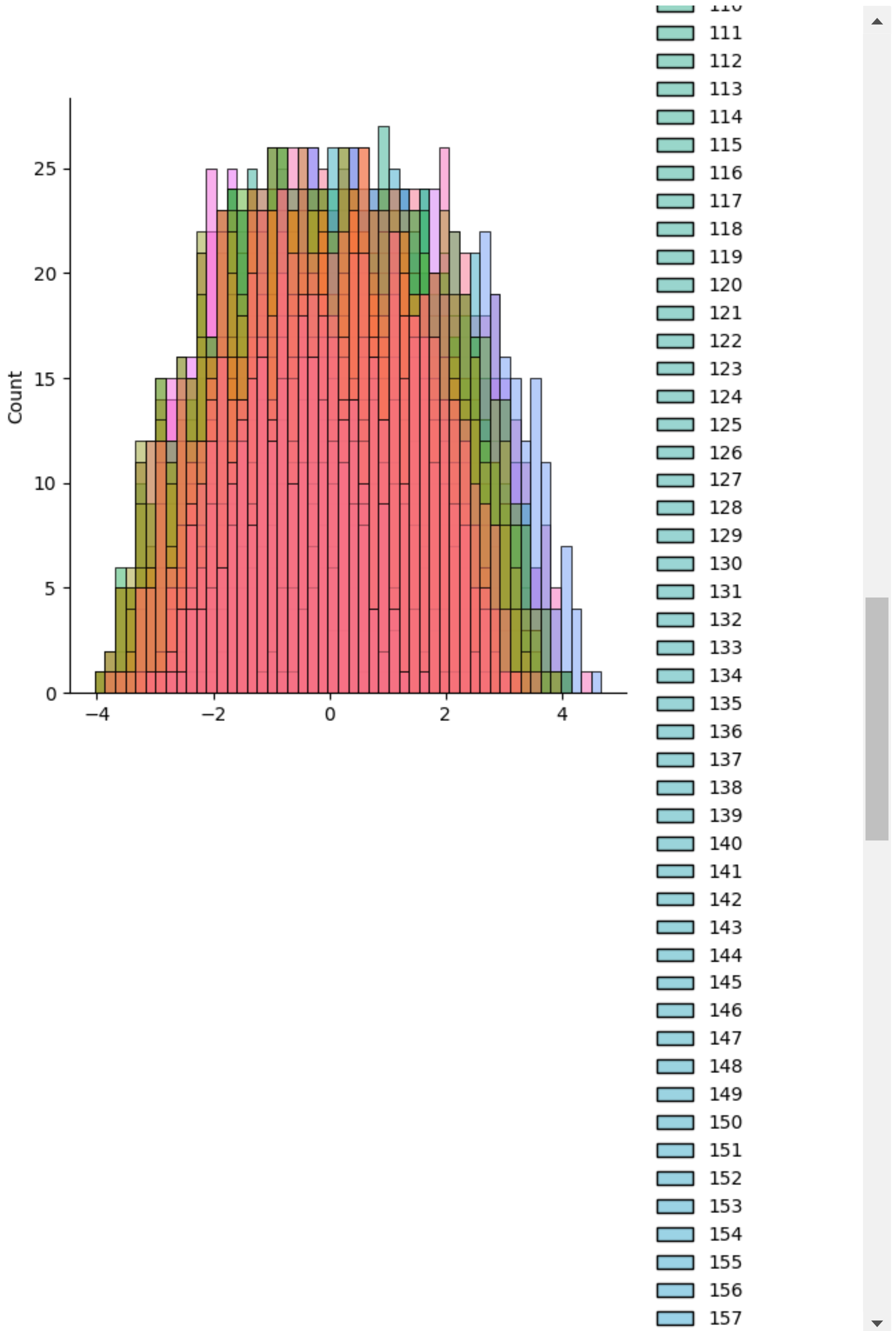
```
sns.displot(predictions)
```

```
plt.scatter(y_test, predictions)
```

```
<matplotlib.collections.PathCollection at 0x7b313b490130>
```




```
sns.displot((y_test-predictions), bins=50);
```


Count
0
5
10
15
20
25
−4
−2
0
2
4
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157