TESLA SCRAPER AND PRICE PREDICTOR

In [1]:

```python
# importing necessary libraries for web scraping, data cleaning and predicting
from bs4 import BeautifulSoup as bs
import requests
import re

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

import torch
import torch.nn as nn
from torchmetrics import Accuracy
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
```

In [2]:

```python
def get_cars(n_pages):
    # += 1 since page 1 doesnt exist
    n_pages += 1
    res = []
    for i in range(n_pages):
        link = f'https://www.finn.no/car/used/search.html?model=1.8078.2000501&page={i}&sort=PUBLISHED_DESC'
        r = requests.get(link)
        soup = bs(r.content, 'html.parser')

        res.append(soup)
    return res

# helper function for extracting numbers (i.e. price) from text
def extract_numbers(s):
    number_extract_pattern = "\\d+"
    r = re.findall(number_extract_pattern, s)
    if len(r) > 0:
        return int(''.join(r))

def flatten(arr):
    return [item for sub_arr in arr for item in sub_arr]

def accuracy(y_pred, y_test):
    n = len(y_pred)
    return np.sqrt((1/n) * np.sum(y_pred.detach().numpy() - y_test.numpy())**2)
```

In [3]:

```python
def equal_lengths(arr):
    for i in range(1, len(arr)):
        if len(arr[i-1]) != len(arr[i]):
            return False
    return True

def print_lengths(arr):
    for a in arr:
        print(len(a))
```

In [4]:

```python
# extracting features from n number of pages
def get_features(n_pages):
    print('Fetching Teslas...')
    pages = get_cars(n_pages)
    print('Done!')

    # current year to calculate the age of the car
    current_year = datetime.now().year

    # getting and cleaning key info and titles
    key_info = []
    titles = []
    for i in range(len(pages)):
        res = pages[i].find_all('div', {'class': 'ads__unit__content__keys'})
        title = pages[i].find_all('a', {'class': 'ads__unit__link'})
        if len(res) > 0:
            key_info.append(res)
            titles.append(title)

    key_info = flatten(key_info)
    titles = flatten(titles)
    titles = [titles[i].text.upper() for i in range(len(titles))]

    # features from key info
    age = []
    kms = []
    price = []

    # additional features from titles
    long_range = []
    performance = []
    ordinary = []


    for i in range(len(key_info)):

        # using beautiful soup to extract needed information
        curr = key_info[i].get_text('div').split('div')
        curr = [extract_numbers(text) for text in curr if extract_numbers(text)]
        title = titles[i]

        # if the title includes both 'long range' and 'performance'
        # it's highly likely that the ad belongs to a car dealer company
        unique_car_condition = not((('LONG RANGE' in title) or ('LR' in title)) and ('PERFORMANCE' in title))


        # include only ads where
        # - production year is given,
        # - price > 150 000kr and
        # - kms driven is provided
        if (len(curr) == 3) and (curr[2] > 150000) and unique_car_condition:
            age.append(current_year - curr[0])
            kms.append(curr[1])
            price.append(curr[2])

            # add only if title is unique
            if ('LONG RANGE' in title) or ('LR' in title):
                long_range.append(1)
                performance.append(0)
                ordinary.append(0)

            elif ('PERFORMANCE' in title):
                long_range.append(0)
                performance.append(1)
                ordinary.append(0)

            else:
                long_range.append(0)
                performance.append(0)
                ordinary.append(1)


    features = np.array([age, kms, price, long_range, performance, ordinary])

    # throwing an error if the lengths are unequal
    if not equal_lengths(features):
        print_lengths(features)
        raise Exception('Lengths are not equal')

    print(f'\nFetched {len(age)} Teslas')
    return features
```

In [5]:

```python
# max number of pages on Finn.no
max_pages = 50

# this takes a minute or two...
features = get_features(max_pages)
```

```
Fetching Teslas...
Done!

Fetched 1216 Teslas
```

In [6]:

```python
# creating a pandas dataframe with the features as columns
def dataset_of(features):
    columns = ['Age', 'KMS', 'Price', 'Long Range', 'Performance', 'Standard']
    return pd.DataFrame(features.T, columns=columns)
```

In [7]:

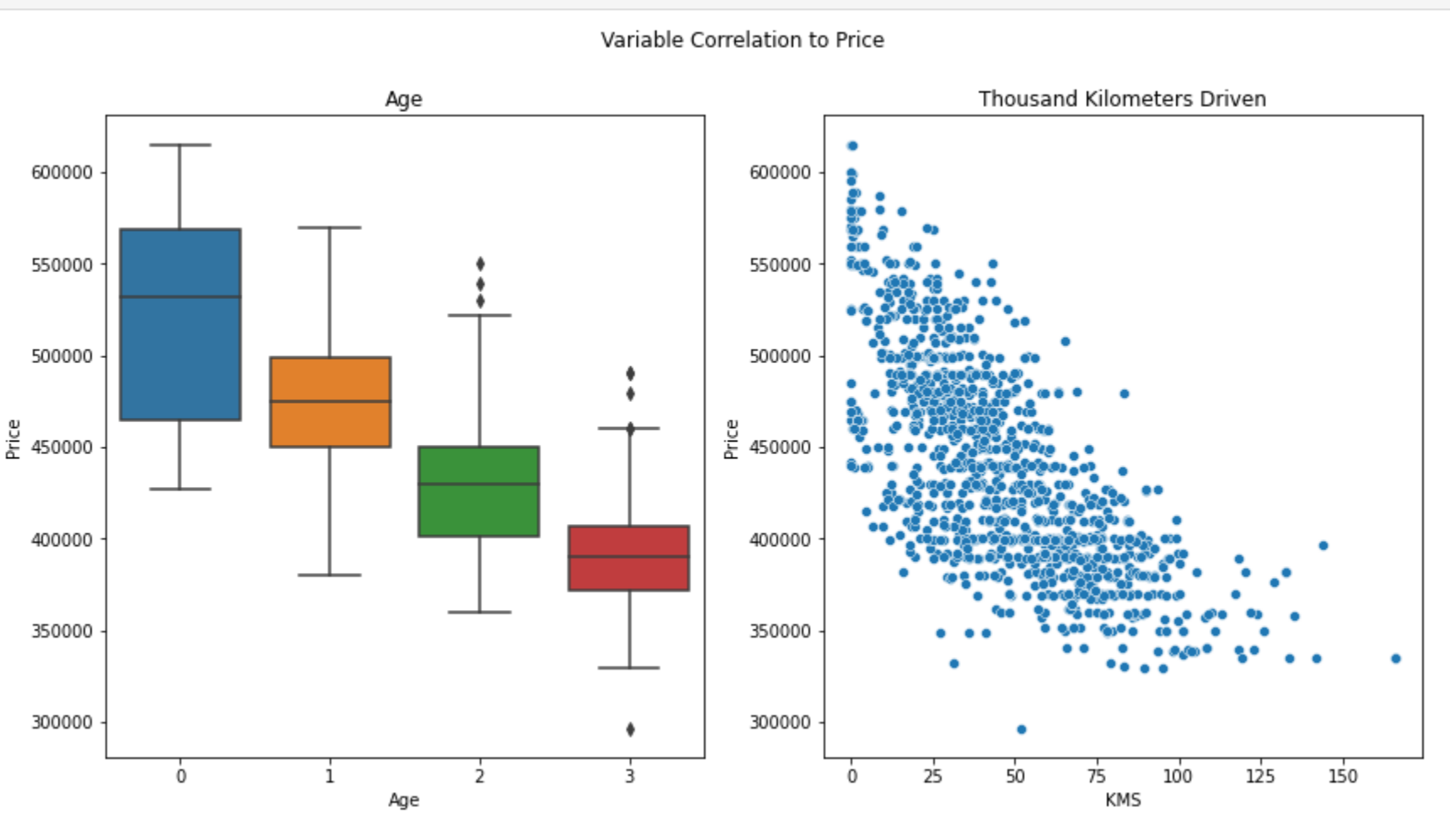
```python
df = dataset_of(features)
df.name = 'MODEL 3 DATASET'

# displaying age, kms and price only
df.describe().T[:3]
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Age	1216.0	1.849507	1.057001	0.0	1.0	2.0	3.0	3.0
KMS	1216.0	46220.551809	27849.547718	5.0	25781.5	42000.0	65292.5	166000.0
Price	1216.0	438487.352796	57240.448397	296670.0	394975.0	429800.0	479000.0	615000.0

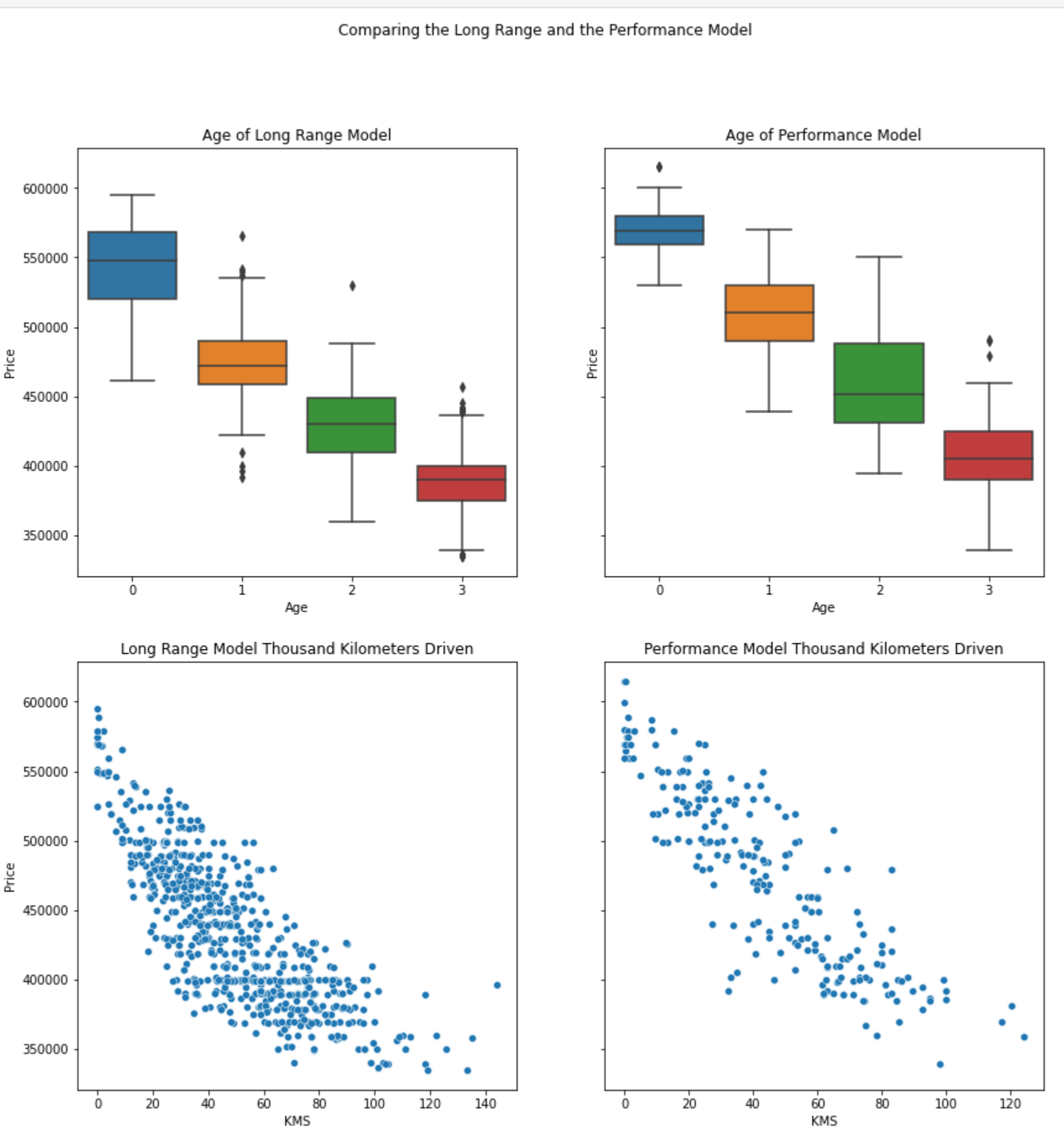
In [8]:

```python
fig, ax = plt.subplots(1, 2, figsize=(14, 7), sharey=False)
fig.suptitle('Variable Correlation to Price')

box_fig = sns.boxplot(data=df, x='Age', y='Price', ax=ax[0])
box_fig.set_title('Age')

scatter_fig = sns.scatterplot(data=df, x=df['KMS'] / 1000, y='Price', ax=ax[1])
scatter_fig.set_title('Thousand Kilometers Driven')
plt.show()
```



In [14]:

```python
# splitting the original dataset into three:
# - standard 2WD
# - long range 4WD
# - performance 4WD

df_standard = df[df['Standard'] == 1]
df_standard = df_standard.drop(['Performance', 'Long Range'], axis=1)
df_standard.name = 'STANDARD DATASET'

df_long_range = df[df['Long Range'] == 1]
df_long_range = df_long_range.drop(['Performance', 'Standard'], axis=1)
df_long_range.name = 'LONG RANGE DATASET'

df_performance = df[df['Performance'] == 1]
df_performance = df_performance.drop(['Long Range', 'Standard'], axis=1)
df_performance.name = 'PERFORMANCE DATASET'

# plotting the comparison
fig, ax = plt.subplots(2, 2, figsize=(14, 14), sharey=True)
fig.suptitle('Comparing the Long Range and the Performance Model')

lr_age = sns.boxplot(data=df_long_range, x='Age', y='Price', ax=ax[0][0])
lr_age.set_title('Age of Long Range Model')

p_age = sns.boxplot(data=df_performance, x='Age', y='Price', ax=ax[0][1])
p_age.set_title('Age of Performance Model')

lr_age = sns.scatterplot(data=df_long_range, x=df['KMS'] / 1000, y='Price', ax=ax[1][0])
lr_age.set_title('Long Range Model Thousand Kilometers Driven')

p_age = sns.scatterplot(data=df_performance, x=df['KMS'] / 1000, y='Price', ax=ax[1][1])
p_age.set_title('Performance Model Thousand Kilometers Driven')

plt.show()
```



In [15]:

```python
print('\t' * 4, 'Displaying Long Range vs. Performance\n\n')
print(f'LONG RANGE\n\tSample Size:    {len(df_long_range)}\n\tMean:\t\t{df_long_range["Price"].mean():.3f}\n\tMax/Min:\t{(df_long_range["Price"].max(), df_long_range["F
print(f'PERFORMANCE\n\tSample Size:    {len(df_performance)}\n\tMean:\t\t{df_performance["Price"].mean():.3f}\n\tMax/Min:\t{(df_performance["Price"].max(), df_performan
```

```
                                Displaying Long Range vs. Performance


LONG RANGE
        Sample Size:    712
        Mean:           434677.694
        Max/Min:        (595000, 334900)

PERFORMANCE
        Sample Size:    224
        Mean:           473728.094
        Max/Min:        (615000, 339000)
```

In [16]:

```python
def linear_regression_on(datasets, show_graph=True):
    for dataset in datasets:
        X = dataset.drop(['Price'], axis=1)
        y = dataset['Price']

        # splitting data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

        # scaling
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        # defining linear regressor and fitting
        linear_regressor = LinearRegression().fit(X_train, y_train)
        prediction = linear_regressor.predict(X_test)
        result_dict = {int(prediction[i]): list(y_test)[i] for i in range(len(prediction))}

        print(f'{dataset.name}')
        print(f'  Score: {round(linear_regressor.score(X_test, y_test), 4)}')
        print(f'  4 first predictions: {list(result_dict.items())[:4]}\n')

        a, b = linear_regressor.coef_, linear_regressor.intercept_
        if show_graph:
            fig, ax = plt.subplots(1, 2, figsize=(14, 6), sharey=False)
            fig.suptitle(f'Multivariate Regression on {dataset.name}')
            ax[0].plot(X_test, y_test, '.', color='#99c3a6')
            ax[0].set_title('Original Test Data')

            ax[1].plot(X_test, y_test, '.', color='#99c3a6')
            ax[1].plot(X_test, prediction, 'o', color='orange')
            ax[1].set_title('Prediction on Test Data')
            plt.show()
```

In [17]:

```python
def neural_net_on(datasets, show_graph=True, show_training=False):
    for dataset in datasets:
        X = np.array(dataset[['KMS', 'Age']])
        y = np.array(dataset['Price'])
        y = y.reshape(y.shape[0], 1)

        n_samples, n_features = X.shape

        # splitting data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

        # scaling
        scaler = MinMaxScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        X_train = torch.from_numpy(X_train.astype(np.float32))
        X_test = torch.from_numpy(X_test.astype(np.float32))
        y_train = torch.from_numpy(y_train.astype(np.float32))
        y_test = torch.from_numpy(y_test.astype(np.float32))


        # defining model, loss funciton and optimizer
        model = nn.Linear(n_features, 1)
        loss = nn.MSELoss()
        optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

        # epic training montage
        for epoch in range(300):
            y_pred = model(X_train)
            l = loss(y_pred, y_train)

            l.backward()

            optimizer.step()
            optimizer.zero_grad()

            if show_training and epoch % 100 == 0:
                print(len(y_pred))
                print(f'\tepoch: {epoch}, loss: {l.item()}')


        prediction = model(X_test).detach().numpy()

        result_dict = {int(prediction[i][0]): int(y_test[i].numpy()[0]) for i in range(len(prediction))}

        print(f'\n{dataset.name}')
        print(f'  First 4 predictions: {list(result_dict.items())[:4]}\n')

        if show_graph:
            fig, ax = plt.subplots(1, 2, figsize=(14, 6), sharey=True)
            fig.suptitle(f'Neural Network on {dataset.name}')
            ax[0].plot(X_test, y_test, '.', color='#99c3a6')
            ax[0].set_title('Original Test Data')

            ax[1].plot(X_test, y_test, '.', color='#99c3a6')
            ax[1].plot(X_test, prediction, 'o', color='orange')
            ax[1].set_title('Prediction on Test Data')
            plt.show()
```

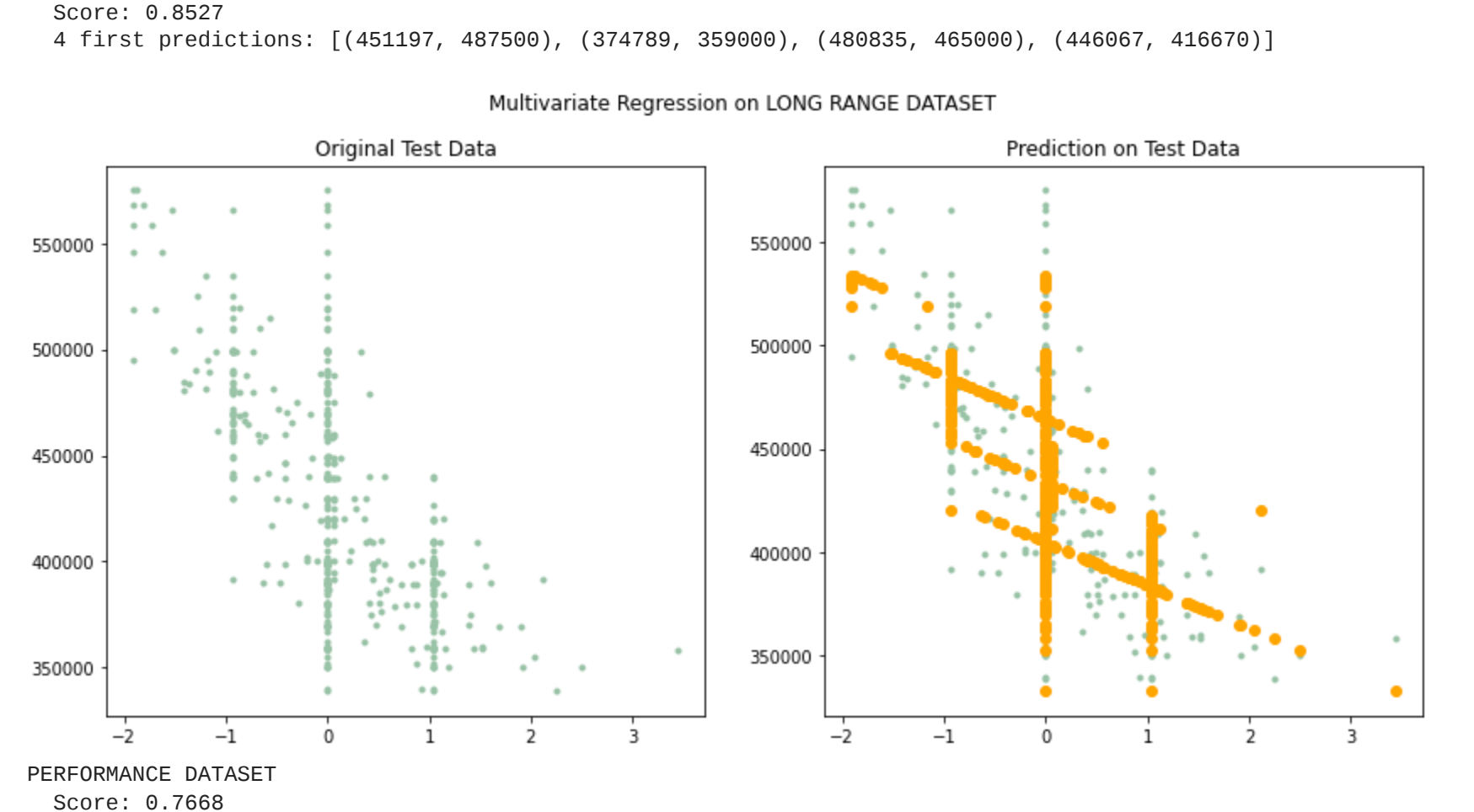
In [18]:

```python
datasets_list = [df_long_range, df_performance]
```

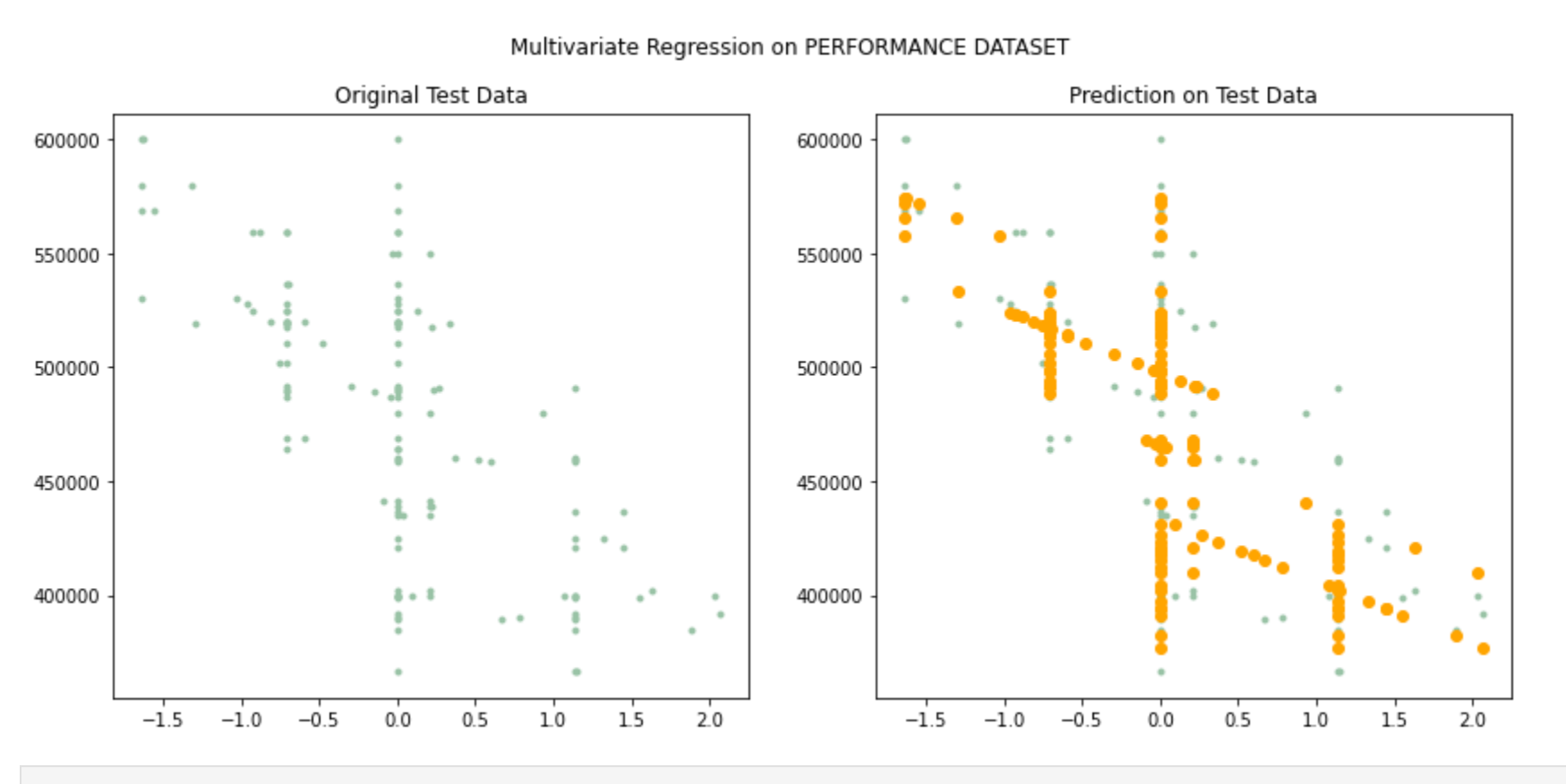
In [19]:

```python
linear_regression_on(datasets_list)
```

```
LONG RANGE DATASET
  Score: 0.8527
  4 first predictions: [(451197, 487500), (374789, 359000), (480835, 465000), (446067, 416670)]
```



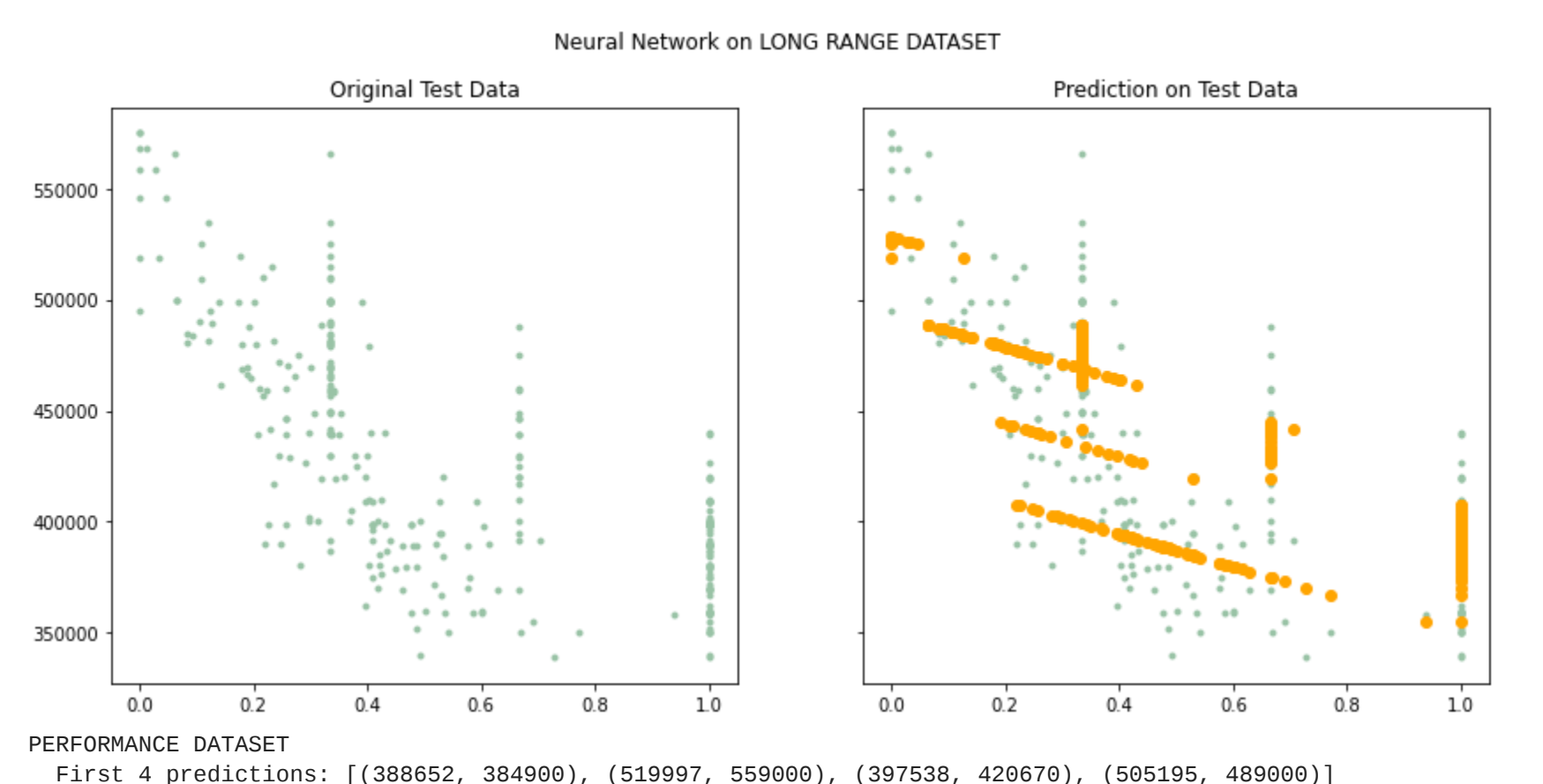

```
PERFORMANCE DATASET
  Score: 0.7668
  4 first predictions: [(382061, 384900), (522023, 559000), (394219, 420670), (501771, 489000)]
```



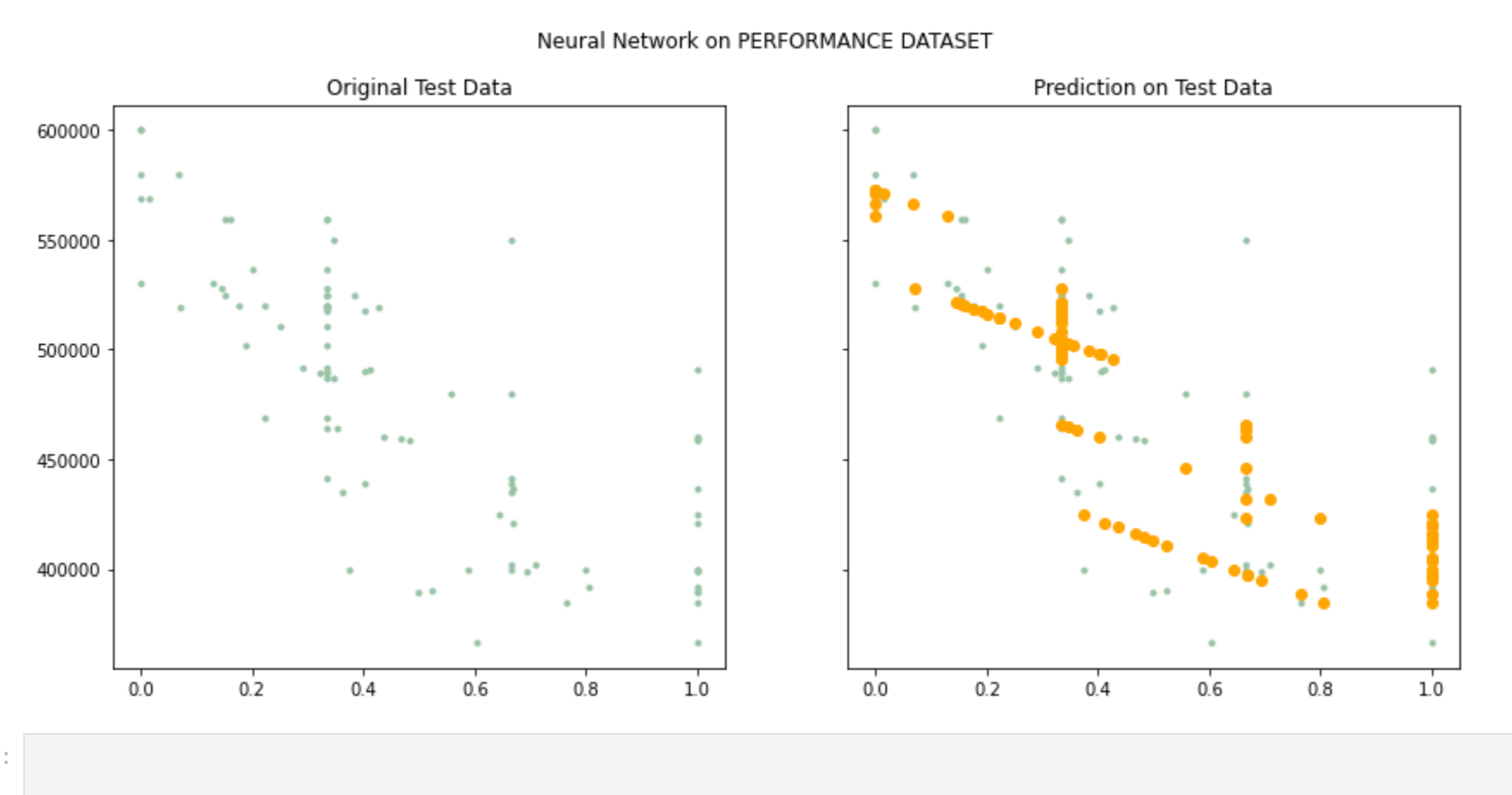
In [20]:

```python
neural_net_on(datasets_list, show_training=False)
```

```
LONG RANGE DATASET
  First 4 predictions: [(444507, 487500), (380671, 359000), (479179, 465000), (441308, 416670)]
```




```
PERFORMANCE DATASET
  First 4 predictions: [(388652, 384900), (519997, 559000), (397538, 420670), (505195, 489000)]
```



In [ ]: