house price prediction

dataset description

Variable	Description
id	A notation for a house
date	Date house was sold
price	Price is prediction target
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms
sqft_living	Square footage of the home
sqft_lot	Square footage of the lot
floors	Total 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 code
lat	Latitude coordinate
long	Longitude coordinate
sqft_living15	Living room area in 2015 (implies-- some renovations) This might or might not have affected the lotsize area
sqft_lot15	LotSize area in 2015 (implies-- some renovations)

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

Type *Markdown* and LaTeX: α^2

Module 1: Importing Data Sets

Using the Pandas method **read_csv()** to load the data from the web address.

In [ ]:
```
file_name = './data.csv'
df = pd.read_csv(file_name)
```

We're using the method `head` to display the first 5 columns of the dataframe.

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

Out[83]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
0	0	7129300520	20141013T000000	221900.0	3.0	1.00	1180	5650	
1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	7242	
2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770	10000	
3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960	5000	
4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	8080	

5 rows × 22 columns

In [ ]:
```
df.dtypes
```

Out[84]:
```
Unnamed: 0          int64
id                  int64
date               object
price             float64
bedrooms          float64
bathrooms         float64
sqft_living         int64
sqft_lot            int64
floors            float64
waterfront          int64
view                int64
condition           int64
grade               int64
sqft_above          int64
sqft_basement       int64
yr_built            int64
yr_renovated        int64
zipcode             int64
lat               float64
long              float64
sqft_living15       int64
sqft_lot15          int64
dtype: object
```

Apart from the two features Unnamed and id, these features don't seem to be much effective in price prediction compared with other features:

1. yr_renovated
2. sqft_living15
3. sqft_lot15
4. view (possibly correlated with waterfront)

In [ ]:
```
df.drop(columns=['Unnamed: 0' , 'id' , 'yr_renovated', 'sqft_living15' , 's
```

Using the method describe to obtain a statistical summary of the dataframe.

In [ ]:
```
df.describe()
```

Out[86]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.(
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.(
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.(
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.(
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.(
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.(
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.(
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.(

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21613 non-null  object 
 1   price          21613 non-null  float64
 2   bedrooms       21600 non-null  float64
 3   bathrooms      21603 non-null  float64
 4   sqft_living    21613 non-null  int64  
 5   sqft_lot       21613 non-null  int64  
 6   floors         21613 non-null  float64
 7   waterfront     21613 non-null  int64  
 8   condition      21613 non-null  int64  
 9   grade          21613 non-null  int64  
 10  sqft_above     21613 non-null  int64  
 11  sqft_basement  21613 non-null  int64  
 12  yr_built       21613 non-null  int64  
 13  zipcode        21613 non-null  int64  
 14  lat            21613 non-null  float64
 15  long           21613 non-null  float64
dtypes: float64(6), int64(9), object(1)
memory usage: 2.6+ MB
```

Module 2: Data Wrangling

In [ ]:
```
df.isnull().any()
```

Out[88]:
```
date             False
price            False
bedrooms          True
bathrooms         True
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
zipcode          False
lat              False
long             False
dtype: bool
```

There are missing values for the columns `bedrooms` and `bathrooms`

In [ ]:
```
print(f"number of NaN values for the column bedrooms are {df['bedrooms'].is
print(f"number of NaN values for the column bathrooms are {df['bathrooms'].
```

```
number of NaN values for the column bedrooms are 13
number of NaN values for the column bathrooms are 10
```

Replacing the missing values of the column `'bedrooms'` with the mean of the column `'bedrooms'` using the method `replace()`. Don't forget to set the `inplace` parameter to `True`

In [ ]:
```
df["bedrooms"].fillna(df["bedrooms"].mean(), inplace=True)
df["bathrooms"].fillna(df["bathrooms"].mean(), inplace=True)
df.isnull().sum()
```

Out[90]:
```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
zipcode          0
lat              0
long             0
dtype: int64
```

Converting date values to numbers.

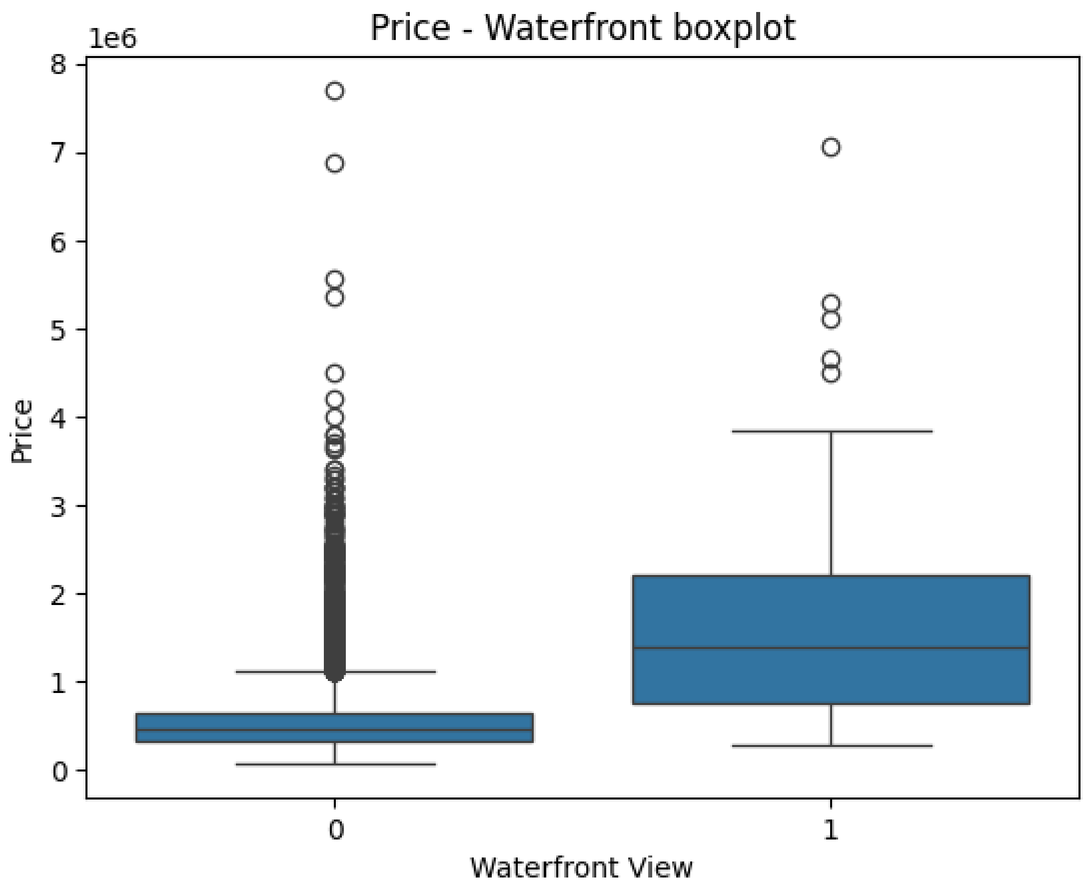
In [ ]:
```
df['date'] = df['date'].astype(str).str[:4].astype(int)
df['date']
```

Out[95]:
```
0        2014
1        2014
2        2015
3        2014
4        2015
         ...
21608    2014
21609    2015
21610    2014
21611    2015
21612    2014
Name: date, Length: 21613, dtype: int64
```

Module 3: Exploratory Data Analysis

Use the function `boxplot` in the seaborn library to determine whether houses with a waterfront view or without a waterfront view have more price outliers.

In [ ]:
```
sns.boxplot(x="waterfront", y="price", data=df[["waterfront" , "price"]])
plt.xlabel("Waterfront View")
plt.ylabel("Price")
plt.title("Price - Waterfront boxplot")
plt.show()
```



1. Properties Without Waterfront View (0):

Median price: Approximately $450000.

Interquartile range (IQR): From around 320000 to $640000.

Outliers: Some individual points above the upper whisker.

2. Properties With Waterfront View (1):

Higher median price: Near $1.4 million.

IQR: From about 800000 to over $2 million.

Outliers: Also present above the upper whisker.

Using the Pandas method `corr()` to find the feature other than price that is most correlated with price.

In [ ]:
```
correlation = df.select_dtypes(include=['number']).corr()['price'].abs()
correlation
```

Out[97]:
```
date             0.003576
price            1.000000
bedrooms         0.308797
bathrooms        0.525738
sqft_living      0.702035
sqft_lot         0.089661
floors           0.256794
waterfront       0.266369
condition        0.036362
grade            0.667434
sqft_above       0.605567
sqft_basement    0.323816
yr_built         0.054012
zipcode          0.053203
lat              0.307003
long             0.021626
Name: price, dtype: float64
```

In [ ]:
```
del correlation["price"]
print(f"The feature most correlated with price is {correlation.idxmax()} wi
```

```
The feature most correlated with price is sqft_living with a correlation
of 0.7020350546118005
```

Module 4: Model Development

It is necessary to scale the target variable (y) since we're using a model that's sensitive to the scale of the target variable.

In [172]:
```
# split and standard the data in this place
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


X , y = df.drop(columns = ["price"]) , df[["price"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra


sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
y_train_scaled = sc.fit_transform(y_train)
y_test_scaled = sc.transform(y_test)
```

In [176]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import keras
from keras import backend as K

model = Sequential()
model.add(Dense(15, activation='relu', input_dim=15))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(120, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Module 5: Model Train

In [178]:
```
def r2(y_true, y_pred):
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res/(SS_tot + K.epsilon()))


model.compile(optimizer="Adam" , loss='mean_squared_error' , metrics=[r2])


model.fit(X_train_scaled, y_train_scaled, epochs=200, batch_size=64)


history = model.fit(X_train_scaled, y_train_scaled, epochs=200, batch_size=
```

```
Epoch 15/200
271/271 [==============================] - 1s 3ms/step - loss: 0.6458 - r
2: 0.3588
Epoch 16/200
271/271 [==============================] - 1s 3ms/step - loss: 0.6455 - r
2: 0.3615
Epoch 17/200
271/271 [==============================] - 1s 4ms/step - loss: 0.6448 - r
2: 0.3621
Epoch 18/200
271/271 [==============================] - 1s 4ms/step - loss: 0.6441 - r
2: 0.3668
Epoch 19/200
271/271 [==============================] - 1s 3ms/step - loss: 0.6434 - r
2: 0.3642
Epoch 20/200
271/271 [==============================] - 1s 2ms/step - loss: 0.6428 - r
2: 0.3545
Epoch 21/200
271/271 [==============================] - 1s 2ms/step - loss: 0.6427 - r
```

Module 6: Evaluate your model with test data

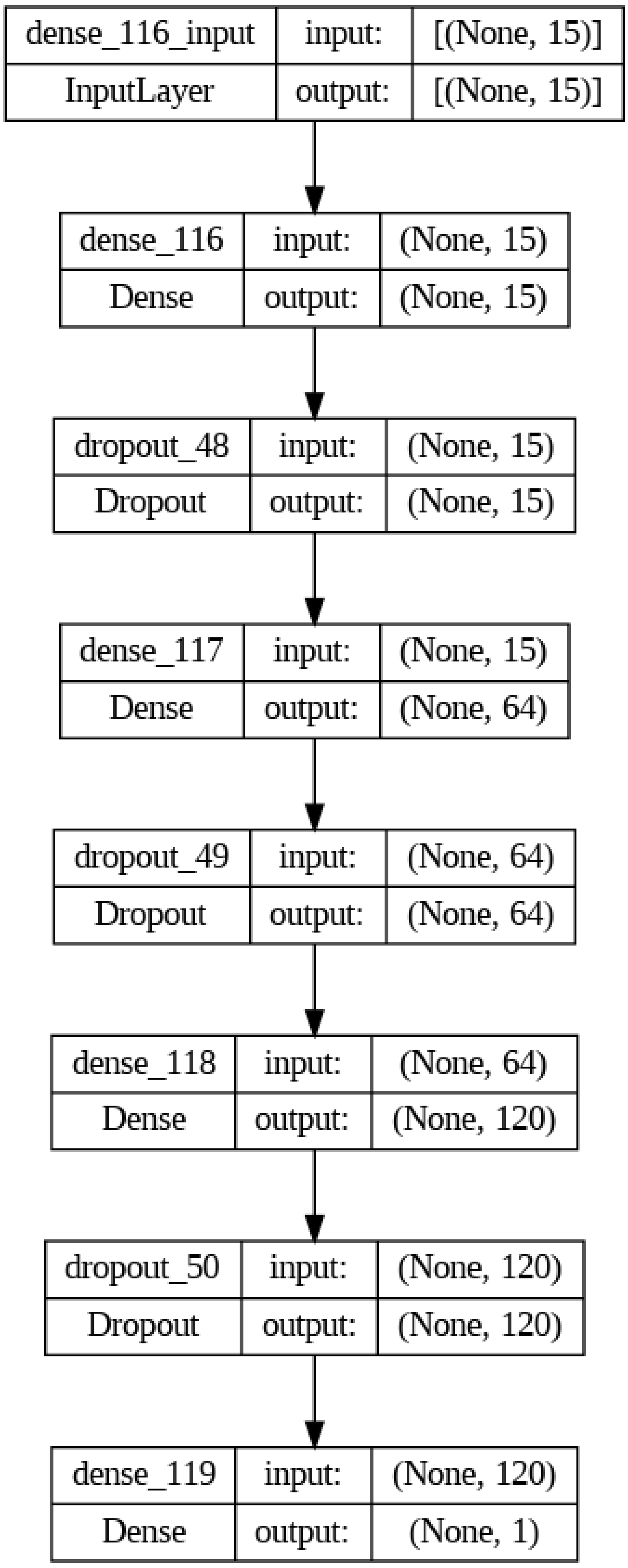
In [180]:
```
mse , r2 = model.evaluate(X_test_scaled, y_test_scaled)

print(f"R-squared (Coefficient of determination) is {r2}")
print(f'MSE on testing set is {mse}')
```

```
136/136 [==============================] - 0s 1ms/step - loss: 0.7520 - r
2: 0.3463
R-squared (Coefficient of determination) is 0.3462739884853363
MSE on testing set is 0.7520144581794739
```

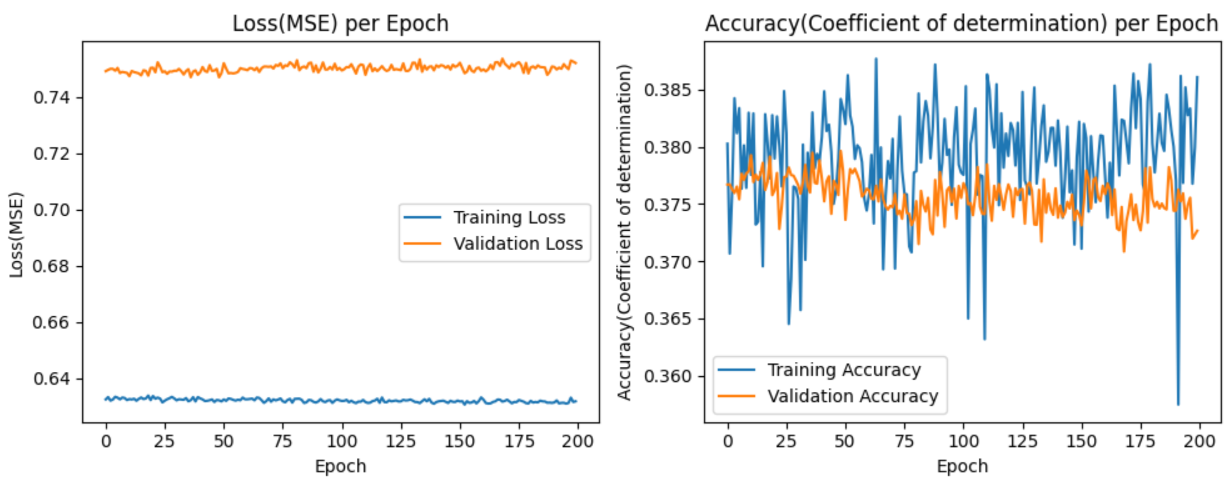
Module 7: visualize your model

In [181]:
```
keras.utils.plot_model(model, to_file='model.png', show_shapes=True, show_l
```

Out[181]:



In [182]:
```
# MSE plot
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss(MSE)')
plt.title('Loss(MSE) per Epoch')
plt.legend()

# R2 plot
plt.subplot(1, 2, 2)
plt.plot(history.history['r2'], label='Training Accuracy')
plt.plot(history.history['val_r2'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy(Coefficient of determination)')
plt.title('Accuracy(Coefficient of determination) per Epoch')
plt.legend()

plt.tight_layout()
plt.show()
```



Module 8

In [183]:
```
X , y = df[["waterfront" , "floors" , "sqft_living"]] , df[["price"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra


sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
y_train_scaled = sc.fit_transform(y_train)
y_test_scaled = sc.transform(y_test)
```

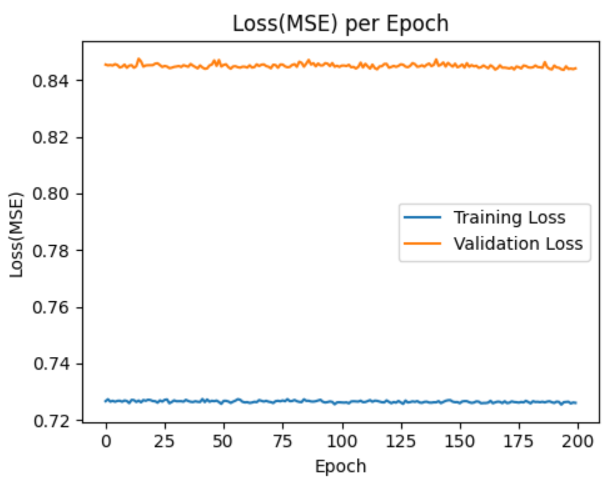
In [186]:
```
def r2(y_true, y_pred):
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res/(SS_tot + K.epsilon()))

model = Sequential()
model.add(Dense(15, activation='relu', input_dim=3))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(120, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer="Adam" , loss='mean_squared_error' , metrics=[r2])


model.fit(X_train_scaled, y_train_scaled, epochs=200, batch_size=64)


history = model.fit(X_train_scaled, y_train_scaled, epochs=200, batch_size=
```

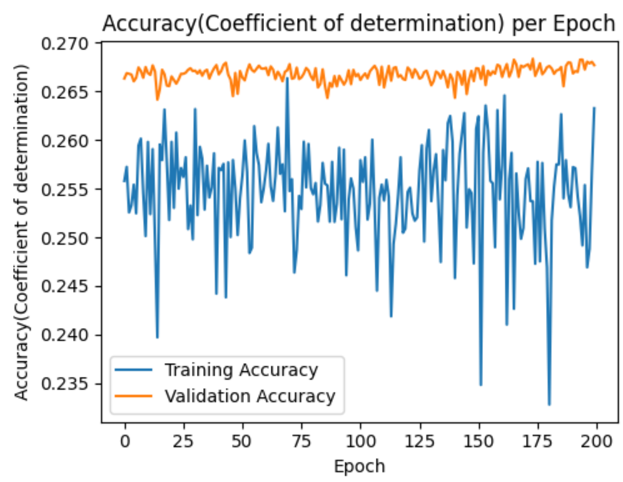
```
Epoch 1/200
271/271 [==============================] - 2s 2ms/step - loss: 0.8130 - r
2: 0.1536
Epoch 2/200
271/271 [==============================] - 1s 2ms/step - loss: 0.7434 - r
2: 0.2336
Epoch 3/200
271/271 [==============================] - 1s 2ms/step - loss: 0.7392 - r
2: 0.2394
Epoch 4/200
271/271 [==============================] - 1s 2ms/step - loss: 0.7350 - r
2: 0.2492
Epoch 5/200
271/271 [==============================] - 1s 2ms/step - loss: 0.7366 - r
2: 0.2414
Epoch 6/200
271/271 [==============================] - 1s 2ms/step - loss: 0.7338 - r
2: 0.2424
Epoch 7/200
```

In [187]:
```
mse , r2 = model.evaluate(X_test_scaled, y_test_scaled)

print(f"R-squared (Coefficient of determination) is {r2}")
print(f'MSE on testing set is {mse}')
```

```
136/136 [==============================] - 0s 2ms/step - loss: 0.8441 - r
2: 0.2209
R-squared (Coefficient of determination) is 0.22091415524482727
MSE on testing set is 0.8441365361213684
```

In [188]:
```
# MSE plot
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss(MSE)')
plt.title('Loss(MSE) per Epoch')
plt.legend()

# R2 plot
plt.subplot(1, 2, 2)
plt.plot(history.history['r2'], label='Training Accuracy')
plt.plot(history.history['val_r2'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy(Coefficient of determination)')
plt.title('Accuracy(Coefficient of determination) per Epoch')
plt.legend()

plt.tight_layout()
plt.show()
```





In [ ]: