practical-4

May 9, 2025

[1]:
```
import pandas as pd
```

[2]:
```
import numpy as np
```

[3]:
```
import matplotlib.pyplot as plt
```

[4]:
```
x=np.array([95,85,80,70,60])
y=np.array([85,95,70,65,70])
```

[5]:
```
model=np.polyfit(x,y,1)
```

[6]:
```
model
```

[6]:
```
array([ 0.64383562, 26.78082192])
```

[7]:
```
predict=np.poly1d(model)
predict(65)
```

[7]:
```
np.float64(68.63013698630135)
```

[8]:
```
y_pred=predict(x)
y_pred
```

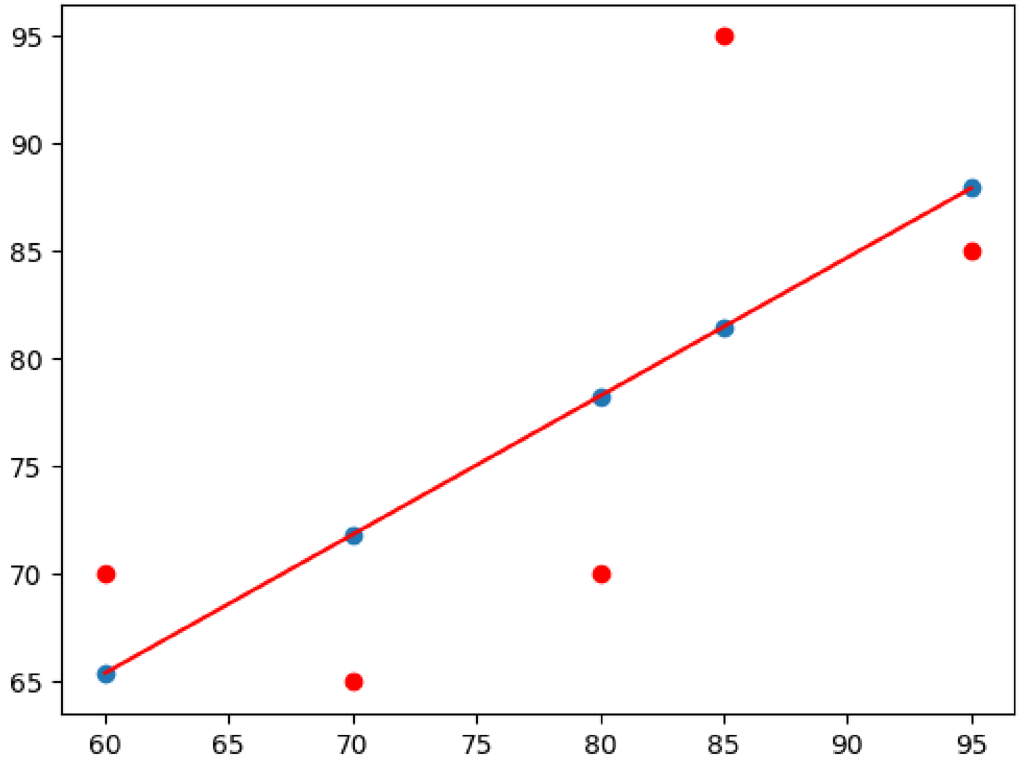
[8]:
```
array([87.94520548, 81.50684932, 78.28767123, 71.84931507, 65.4109589 ])
```

[9]:
```
from sklearn.metrics import r2_score
r2_score(y,y_pred)
```

[9]:
```
0.4803218090889323
```

[10]:
```
y_line=model[1]+model[0]*x
plt.plot(x,y_line,c='r')
plt.scatter(x,y_pred)
plt.scatter(x,y,c='r')
```

[10]:
```
<matplotlib.collections.PathCollection at 0x1963d230e10>
```



[ ]:

[ ]:

[11]:
```
import numpy as np
```

[12]:
```
import pandas as pd
```

[13]:
```
import matplotlib.pyplot as plt
```

[14]:
```
data = pd.read_csv("HousingData.csv")
```

[15]:
```
data.head()
```

[15]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222     18.7

        B  LSTAT  MEDV
```

```
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90    NaN  36.2
```

[16]:
```python
data.tail()
```

[16]:
```
        CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
501  0.06263  0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273     21.0
502  0.04527  0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273     21.0
503  0.06076  0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273     21.0
504  0.10959  0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273     21.0
505  0.04741  0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273     21.0

          B  LSTAT  MEDV
501  391.99    NaN  22.4
502  396.90   9.08  20.6
503  396.90   5.64  23.9
504  393.45   6.48  22.0
505  396.90   7.88  11.9
```

[17]:
```python
data.isnull().sum()
```

[17]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

[18]:
```python
x = data.iloc[:,0:13]
y = data.iloc[:,-1]
```

[19]:
```python
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.
 ↪2,random_state=0)
```

[20]:
```python
import sklearn
```

[21]:
```python
from sklearn.linear_model import LinearRegression
```

[22]:
```python
lm=LinearRegression()
```

[23]:
```python
model=lm.fit(xtrain, ytrain)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[23], line 1
----> 1 model=lm.fit(xtrain, ytrain)

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\base.py:
↪1389, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,␣
↪**kwargs)
   1382     estimator._validate_params()
   1384 with config_context(
   1385     skip_parameter_validation=(
   1386         prefer_skip_nested_validation or global_skip_validation
   1387     )
   1388 ):
-> 1389     return fit_method(estimator, *args, **kwargs)

File␣
↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\linear_model\_base.
↪py:601, in LinearRegression.fit(self, X, y, sample_weight)
    597 n_jobs_ = self.n_jobs
    599 accept_sparse = False if self.positive else ["csr", "csc", "coo"]
--> 601 X, y = validate_data(
    602     self,
    603     X,
    604     y,
    605     accept_sparse=accept_sparse,
    606     y_numeric=True,
    607     multi_output=True,
    608     force_writeable=True,
    609 )
    611 has_sw = sample_weight is not None
    612 if has_sw:

File␣
↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.
↪py:2961, in validate_data(_estimator, X, y, reset, validate_separately,␣
↪skip_check_array, **check_params)
   2959         y = check_array(y, input_name="y", **check_y_params)
   2960     else:
-> 2961         X, y = check_X_y(X, y, **check_params)
```

```
   2962     out = X, y
   2964 if not no_val_X and check_params.get("ensure_2d", True):

File␣
↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.
↪py:1370, in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order,␣
↪copy, force_writeable, force_all_finite, ensure_all_finite, ensure_2d,␣
↪allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric,␣
↪estimator)
   1364     raise ValueError(
   1365         f"{estimator_name} requires y to be passed, but the target y is␣
↪None"
   1366     )
   1368 ensure_all_finite = _deprecate_force_all_finite(force_all_finite,␣
↪ensure_all_finite)
-> 1370 X = check_array(
   1371     X,
   1372     accept_sparse=accept_sparse,
   1373     accept_large_sparse=accept_large_sparse,
   1374     dtype=dtype,
   1375     order=order,
   1376     copy=copy,
   1377     force_writeable=force_writeable,
   1378     ensure_all_finite=ensure_all_finite,
   1379     ensure_2d=ensure_2d,
   1380     allow_nd=allow_nd,
   1381     ensure_min_samples=ensure_min_samples,
   1382     ensure_min_features=ensure_min_features,
   1383     estimator=estimator,
   1384     input_name="X",
   1385 )
   1387 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric,␣
↪estimator=estimator)
   1389 check_consistent_length(X, y)

File␣
↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.
↪py:1107, in check_array(array, accept_sparse, accept_large_sparse, dtype,␣
↪order, copy, force_writeable, force_all_finite, ensure_all_finite,␣
↪ensure_non_negative, ensure_2d, allow_nd, ensure_min_samples,␣
↪ensure_min_features, estimator, input_name)
   1101     raise ValueError(
   1102         "Found array with dim %d. %s expected <= 2."
   1103         % (array.ndim, estimator_name)
   1104     )
   1106 if ensure_all_finite:
-> 1107     _assert_all_finite(
   1108         array,
   1109         input_name=input_name,
   1110         estimator_name=estimator_name,
```

```
   1111         allow_nan=ensure_all_finite == "allow-nan",
   1112     )
   1114 if copy:
   1115     if _is_numpy_namespace(xp):
   1116         # only make a copy if `array` and `array_orig` may share memory

File␣
  ↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.
  ↪py:120, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name,␣
  ↪input_name)
    117 if first_pass_isfinite:
    118     return
--> 120 _assert_all_finite_element_wise(
    121     X,
    122     xp=xp,
    123     allow_nan=allow_nan,
    124     msg_dtype=msg_dtype,
    125     estimator_name=estimator_name,
    126     input_name=input_name,
    127 )

File␣
  ↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.
  ↪py:169, in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype,␣
  ↪estimator_name, input_name)
    152 if estimator_name and input_name == "X" and has_nan_error:
    153     # Improve the error message on how to handle missing values in
    154     # scikit-learn.
    155     msg_err += (
    156         f"\n{estimator_name} does not accept missing values"
    157         " encoded as NaN natively. For supervised learning, you might␣
  ↪want"
   (…)
    167         "#estimators-that-handle-nan-values"
    168     )
--> 169 raise ValueError(msg_err)

ValueError: Input X contains NaN.
LinearRegression does not accept missing values encoded as NaN natively. For␣
  ↪supervised learning, you might want to consider sklearn.ensemble.
  ↪HistGradientBoostingClassifier and Regressor which accept missing values␣
  ↪encoded as NaNs natively. Alternatively, it is possible to preprocess the␣
  ↪data, for instance by using an imputer transformer in a pipeline or drop␣
  ↪samples with missing values. See https://scikit-learn.org/stable/modules/
  ↪impute.html You can find a list of all estimators that handle NaN values at␣
  ↪the following page: https://scikit-learn.org/stable/modules/impute.
  ↪html#estimators-that-handle-nan-values
```

[24]:
```python
import pandas as pd
```

```
# Combine xtrain and ytrain to drop rows with NaN in either
train_df = pd.concat([pd.DataFrame(xtrain), pd.Series(ytrain)], axis=1)
train_df.dropna(inplace=True)

# Split them back
xtrain_clean = train_df.iloc[:, :-1].values
ytrain_clean = train_df.iloc[:, -1].values

model = lm.fit(xtrain_clean, ytrain_clean)
```

[25]:
```
from sklearn.impute import SimpleImputer
import numpy as np

imputer = SimpleImputer(strategy='mean')  # or 'median', 'most_frequent', etc.
xtrain_imputed = imputer.fit_transform(xtrain)

model = lm.fit(xtrain_imputed, ytrain)
```

[26]:
```
ytrain_pred=lm.predict(xtrain)
ytest_pred=lm.predict(xtest)
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python313\Lib\site-
packages\sklearn\utils\validation.py:2732: UserWarning: X has feature names, but
LinearRegression was fitted without feature names
  warnings.warn(
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[26], line 1
----> 1 ytrain_pred=lm.predict(xtrain)
      2 ytest_pred=lm.predict(xtest)

File␣
 ↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\linear_model\_base.
 ↪py:297, in LinearModel.predict(self, X)
    283 def predict(self, X):
    284     """
    285     Predict using the linear model.
    286
   (…)
    295         Returns predicted values.
    296     """
--> 297     return self._decision_function(X)

File␣
 ↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\linear_model\_base.
 ↪py:276, in LinearModel._decision_function(self, X)
    273 def _decision_function(self, X):
```

```
    274     check_is_fitted(self)
--> 276     X = validate_data(self, X, accept_sparse=["csr", "csc", "coo"], reset=False)
    277     coef_ = self.coef_
    278     if coef_.ndim == 1:

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2944, in validate_data(_estimator, X, y, reset, validate_separately, skip_check_array, **check_params)
   2942         out = X, y
   2943 elif not no_val_X and no_val_y:
-> 2944     out = check_array(X, input_name="X", **check_params)
   2945 elif no_val_X and not no_val_y:
   2946     out = _check_y(y, **check_params)

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:1107, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_writeable, force_all_finite, ensure_all_finite, ensure_non_negative, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
   1101     raise ValueError(
   1102         "Found array with dim %d. %s expected <= 2."
   1103         % (array.ndim, estimator_name)
   1104     )
   1106 if ensure_all_finite:
-> 1107     _assert_all_finite(
   1108         array,
   1109         input_name=input_name,
   1110         estimator_name=estimator_name,
   1111         allow_nan=ensure_all_finite == "allow-nan",
   1112     )
   1114 if copy:
   1115     if _is_numpy_namespace(xp):
   1116         # only make a copy if `array` and `array_orig` may share memory

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:120, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    117 if first_pass_isfinite:
    118     return
--> 120 _assert_all_finite_element_wise(
    121     X,
    122     xp=xp,
    123     allow_nan=allow_nan,
    124     msg_dtype=msg_dtype,
    125     estimator_name=estimator_name,
    126     input_name=input_name,
```

```
    127 )

File␣
↪~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.
↪py:169, in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype,␣
↪estimator_name, input_name)
    152 if estimator_name and input_name == "X" and has_nan_error:
    153     # Improve the error message on how to handle missing values in
    154     # scikit-learn.
    155     msg_err += (
    156         f"\n{estimator_name} does not accept missing values"
    157         " encoded as NaN natively. For supervised learning, you might␣
↪want"
   (…)
    167         "#estimators-that-handle-nan-values"
    168     )
--> 169 raise ValueError(msg_err)

ValueError: Input X contains NaN.
LinearRegression does not accept missing values encoded as NaN natively. For␣
↪supervised learning, you might want to consider sklearn.ensemble.
↪HistGradientBoostingClassifier and Regressor which accept missing values␣
↪encoded as NaNs natively. Alternatively, it is possible to preprocess the␣
↪data, for instance by using an imputer transformer in a pipeline or drop␣
↪samples with missing values. See https://scikit-learn.org/stable/modules/
↪impute.html You can find a list of all estimators that handle NaN values at␣
↪the following page: https://scikit-learn.org/stable/modules/impute.
↪html#estimators-that-handle-nan-values
```

[27]:
```
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression

# Assuming xtrain and xtest are pandas DataFrames
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on xtrain and transform both xtrain and xtest
xtrain_imputed = imputer.fit_transform(xtrain)
xtest_imputed = imputer.transform(xtest)

# Now fit the model and predict
lm = LinearRegression()
model = lm.fit(xtrain_imputed, ytrain)

ytrain_pred = lm.predict(xtrain_imputed)
ytest_pred = lm.predict(xtest_imputed)
```

[28]:
```
# Convert back to DataFrame if needed
xtrain_imputed_df = pd.DataFrame(xtrain_imputed, columns=xtrain.columns)
xtest_imputed_df = pd.DataFrame(xtest_imputed, columns=xtest.columns)
```

```
model = lm.fit(xtrain_imputed_df, ytrain)
ytrain_pred = lm.predict(xtrain_imputed_df)
ytest_pred = lm.predict(xtest_imputed_df)
```

[29]:
```
df=pd.DataFrame(ytrain_pred,ytrain)
```

[30]:
```
df=pd.DataFrame(ytest_pred,ytest)
```

[31]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

[32]:
```
mse=mean_squared_error(ytest,ytest_pred)
```

[33]:
```
print(mse)
```

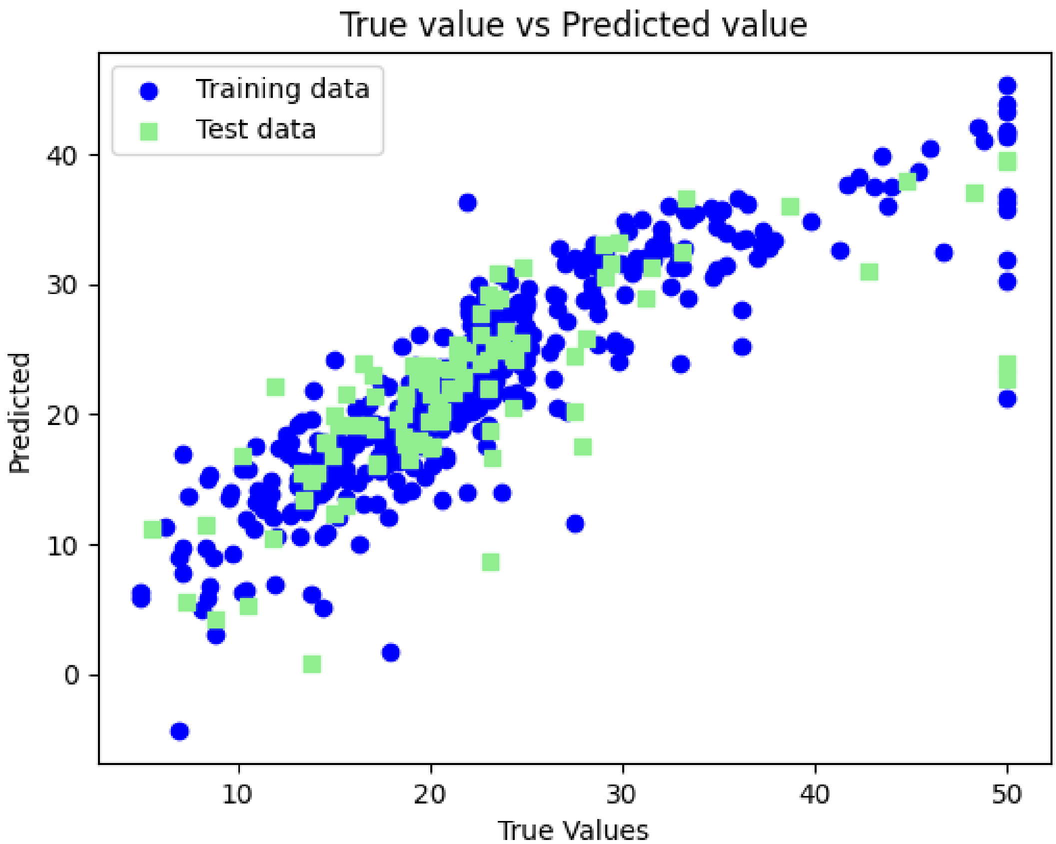
```
34.99330686034018
```

[34]:
```
mse=mean_squared_error(ytrain_pred,ytrain)
```

[35]:
```
print(mse)
```

```
20.023850985554915
```

[36]:
```
plt.scatter(ytrain,ytrain_pred,c='blue',marker='o',label='Training data')
plt.scatter(ytest,ytest_pred,c='lightgreen',marker='s',label='Test data')
plt.xlabel('True Values')
plt.ylabel('Predicted')
plt.title("True value vs Predicted value")
plt.legend(loc='upper left')
plt.plot()
plt.show()
```


True value vs Predicted value
Training data
Test data
Predicted
True Values
40
30
20
10
0
10
20
30
40
50

```
[ ]:
```