Ice cream dataset - Model to predict the daily revenue in dollars based on the outside air temperature (in celsius)

Data set:

- Independant variable X: Outside Air Temperature
- Dependant variable Y: Overall daily revenue generated in dollars

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
IceCream = pd.read_csv("IceCreamData.csv")
```

```
IceCream.sample(5)
```

	Temperature	Revenue
203	10.447126	278.309844
156	18.880356	476.794525
207	9.782381	228.901030
67	11.694538	284.772789
204	5.822332	186.476487

```
IceCream.isnull().sum()
```

```
Temperature    0
Revenue        0
dtype: int64
```

```
IceCream.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  500 non-null    float64
 1   Revenue      500 non-null    float64
dtypes: float64(2)
memory usage: 7.9 KB
```

```
IceCream.describe()
```

	Temperature	Revenue
count	500.000000	500.000000
mean	22.232225	521.570777
std	8.096388	175.404751
min	0.000000	10.000000
25%	17.122258	405.558681
50%	22.392791	529.368565
75%	27.740674	642.257922
max	45.000000	1000.000000

```
IceCream.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  500 non-null    float64
 1   Revenue      500 non-null    float64
```

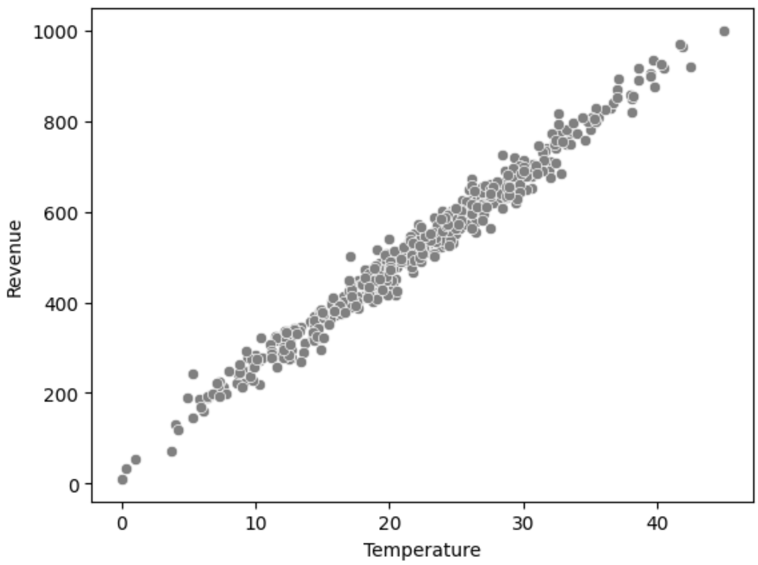
```
dtypes: float64(2)
memory usage: 7.9 KB
```

```
IceCream.corr()
```

	Temperature	Revenue
Temperature	1.000000	0.989802
Revenue	0.989802	1.000000

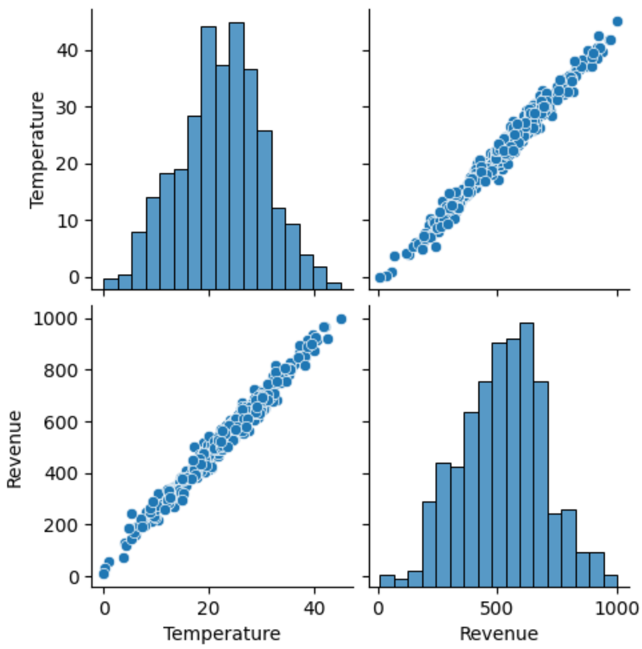
```
sns.scatterplot(x='Temperature', y='Revenue', data = IceCream, color = 'gray')
```

```
<Axes: xlabel='Temperature', ylabel='Revenue'>
```



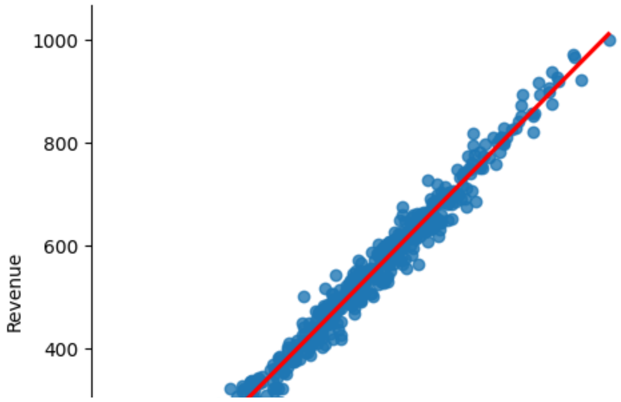

```
sns.pairplot(IceCream)
```

```
<seaborn.axisgrid.PairGrid at 0x7d6a7bf00c70>
```




```
sns.lmplot(x='Temperature', y='Revenue', data=IceCream, line_kws={'color': 'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7d6a7c020040>
```



Split data

```
y = IceCream['Revenue']
```

```
X = IceCream[['Temperature']]
```

```
X
```

	Temperature
0	24.566884
1	26.005191
2	27.790554
3	20.595335
4	11.503498
...	...
495	22.274899
496	32.893092
497	12.588157
498	22.362402
499	28.957736

500 rows × 1 columns

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
print(f"X_train - {X_train.shape},  y_train - {y_train.shape}")
print(f"X_test - {X_test.shape},  y_test - {y_test.shape}")
print(f"Train on {len(X_train)} rows")
print(f"Test on {len(X_test)} rows")
```

```
X_train - (375, 1),  y_train - (375,)
X_test - (125, 1),  y_test - (125,)
Train on 375 rows
Test on 125 rows
```

Train

```
X_train.shape
```

```
(375, 1)
```

```
model = LinearRegression()
```

```
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
print('Linear Model Coefficient (m): ', model.coef_)
print('Linear Model Coefficient (b): ', model.intercept_)
```

```
Linear Model Coefficient (m):  [21.30636481]
Linear Model Coefficient (b):  47.76383333454754
```

Test

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

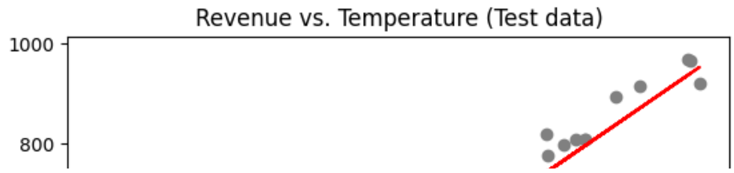
```
array([412.80142191, 477.15835979, 624.49745706, 870.80446844,
       422.50810293, 696.68623667, 388.50261608, 161.50600125,
       474.2402211 , 502.82568663, 506.71271202, 539.00793078,
       684.39743601, 585.27598418, 383.42325456, 941.02138426,
       584.50151423, 670.11667369, 581.62209325, 642.87932579,
       611.4461031 , 766.7302541 , 348.38941836, 659.28370009,
       199.19862963, 458.52224562, 571.19483502, 202.47677129,
       511.73012528, 411.90334137, 519.75496944, 415.55816168,
       497.94869169, 509.78011777, 521.80358217, 537.8845576 ,
       508.02161847, 285.90830775, 796.04659165, 663.52186118,
       409.00851504, 447.75955958, 478.71532644, 241.09434127,
       629.02413169, 685.23554275, 446.3734493 , 678.20810067,
       365.89994242, 548.41082417, 708.90742392, 556.59657378,
       504.24327728, 125.84462025, 564.2380523 , 526.04983308,
       441.39965826, 782.24044587, 353.56372999, 529.69922623,
       192.1188507 , 675.53000865, 251.39463992, 456.90931822,
       484.79226283, 476.10487084, 389.11258002, 353.75357546,
       450.18058966, 392.74152503, 315.61709192, 662.56306944,
       444.78477856, 558.60377695, 502.06650378, 305.24275785,
       953.60990772, 743.41528523, 436.24763735, 571.60016624,
       672.14768928, 234.32244949, 395.11201174, 622.11353733,
       744.83097513, 564.29643515, 545.24687373, 649.94201989,
       234.31285965, 438.66785971, 454.00357962, 420.98900173,
       434.67073049, 546.01625339, 937.65801797, 235.06708088,
       386.89037752, 267.78165242, 458.29029856, 286.13426154,
       414.22116385, 469.94174723, 592.05549433, 508.20809305,
       479.0930753 , 713.14005452, 657.84458937, 635.79961045,
       364.50645716, 795.50635023, 620.3422655 , 614.40775705,
       397.31650098, 717.05276132, 495.34760023, 469.19027536,
       693.99294469, 524.76228583, 838.8067383 , 535.05237092,
       703.20599303, 587.52692492, 183.99320643, 589.43442006,
       585.56094488])
```

```
y_test
```

```
438    412.082357
231    449.112869
313    594.651009
401    916.648613
295    412.065001
          ...
262    706.364904
478    581.262016
50     190.710941
246    583.759781
199    574.710649
Name: Revenue, Length: 125, dtype: float64
```

```
plt.scatter(X_test, y_test, color = 'gray')
plt.plot(X_test, y_predict, color = 'red')
plt.ylabel('Revenue [$]')
plt.xlabel('Temperature [degC]')
plt.title('Revenue vs. Temperature (Test data)')
```

```
Text(0.5, 1.0, 'Revenue vs. Temperature (Test data)')
```




```
print("SSE", np.sum((y_test - y_predict) ** 2))
print("SST", np.sum((y_test - np.mean(y_test)) ** 2))
print("MSE", mean_squared_error(y_test, y_predict))
print("RMSE", np.sqrt(mean_squared_error(y_test, y_predict)))
print("MAE", mean_absolute_error(y_test, y_predict))
print("R^2", r2_score(y_test, y_predict))
```

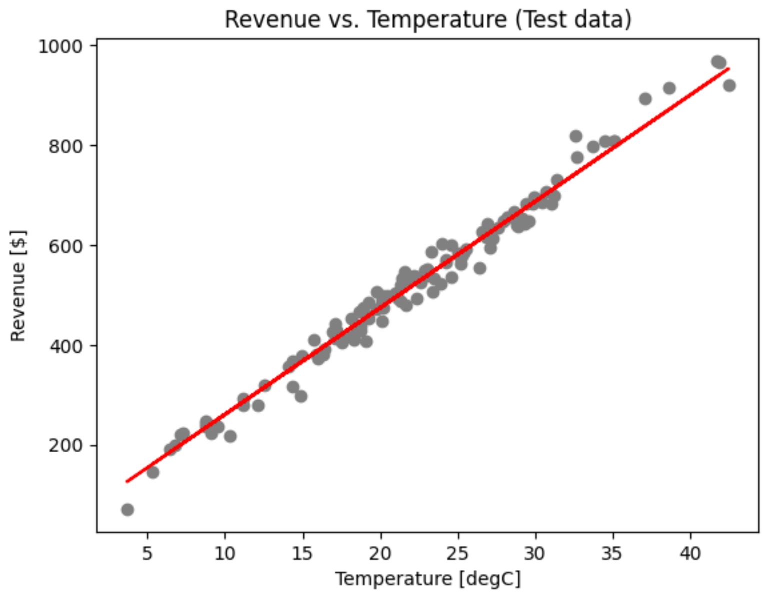
```
SSE 70396.14752312287
SST 3767647.8957343353
MSE 563.1691801849829
RMSE 23.731185814977366
MAE 18.748523562719573
R^2 0.981315624636043
```

OwnLinearRegression

```
def manualLR(data):
    X = data.drop(columns='Money earned')
    y = data['Money earned']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    data = pd.DataFrame({'feature': [1, 2, 3, 4, 5],
                     'target': [2, 3, 4, 5, 6]})

    mean_feature = data['feature'].mean()
    mean_target = data['target'].mean()

# Calculate the terms needed for the formula
    data['numerator'] = (data['feature'] - mean_feature) * (data['target'] - mean_target)
    data['denominator'] = (data['feature'] - mean_feature) ** 2

# Calculate the slope (m)
    m = data['numerator'].sum() / data['denominator'].sum()

# Calculate the intercept (b)
    b = mean_target - m * mean_feature

# Create the regression line equation
    def linear_regression(x):
        res=[]
        for I in x:
            res.append(m * i + b)

        return(res)

    # Predict a value using the regression

    predictions = linear_regression(X_test)

    print("Standard Deviaiton of y: ", np.std(y_test))
    print("Standard Deviaiton of y_pred: ", np.sqrt(np.sum(np.square(predictions-np.mean(predictions)))/(predictions.shape[0]
    print('r^2: ', r2_score(y_test, predictions))
    print("MSE: ", np.mean(np.square(y_test - predictions)))
    print("RMSE: ",np.sqrt(np.mean(np.square(y_test - predictions))))
    print("MAE: ",np.mean(np.abs(y_test - predictions)))


class OwnLinearRegression:
    def __init__(self):
        self.coef_ = None
        self.intercept_ = None

    def fit(self, X, y):
        ones = np.ones((X.shape[0], 1))
        X = np.hstack((ones, X))

        X_transpose = np.transpose(X)
        X_transpose_dot_X = X_transpose.dot(X)
        X_transpose_dot_y = X_transpose.dot(y)
        coefficients = np.linalg.solve(X_transpose_dot_X, X_transpose_dot_y)
```

```
        self.intercept_ = coefficients[0]
        self.coef_ = coefficients[1:]

    def predict(self, X):
        if self.coef_ is None or self.intercept_ is None:
            raise Exception("Fit the model first !!!")
        return X.dot(self.coef_) + self.intercept_


model_own = OwnLinearRegression()
model_own.fit(X_train, y_train)
y_predict_own = model.predict(X_test)
plt.scatter(X_test, y_test, color = 'gray')
plt.plot(X_test, y_predict_own, color = 'red')
plt.ylabel('Revenue [$]')
plt.xlabel('Temperature [degC]')
plt.title('Revenue vs. Temperature (Test data)')
```

```
Text(0.5, 1.0, 'Revenue vs. Temperature (Test data)')
```




```
print("SSE", np.sum((y_test - y_predict_own) ** 2))
print("SST", np.sum((y_test - np.mean(y_test)) ** 2))
print("MSE", mean_squared_error(y_test, y_predict_own))
print("RMSE", np.sqrt(mean_squared_error(y_test, y_predict_own)))
print("MAE", mean_absolute_error(y_test, y_predict_own))
print("R^2", r2_score(y_test, y_predict_own))
```

```
SSE 70396.14752312208
SST 3767647.8957343353
MSE 563.1691801849767
RMSE 23.731185814977234
MAE 18.74852356271954
R^2 0.9813156246360433
```