Chapter 2 - Regression

Creating features

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
sales_df = pd.read_csv('./datasets/advertising_and_sales_clean.csv')
```

sales_df

In [3]:
```
# Create X from the radio column's values
X = sales_df["radio"].values.reshape(-1, 1)

# Create y from the sales column's values
y = sales_df["sales"].values
```

In [4]:
```
X.shape, y.shape
```

Out[4]:
```
((4546, 1), (4546,))
```

Linear regression model

Simple linear regression: $y = ax + b$ (y: target, x: single feature, a, b: parameters of the model - slope, intercept)

Need to choose a line that minimises the error function/loss function/cost function (which is defined for any given line)

Residual: difference between line and an individual data point

Ordinary Least Squares (OLS): minimise $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

In [5]:
```
from sklearn.linear_model import LinearRegression
```

In [6]:
```
reg = LinearRegression()
reg.fit(X, y)
```

Out[6]:
```
▾ LinearRegression
LinearRegression()
```

In [7]:
```
predictions = reg.predict(X)
predictions[:5]
```

Out[7]:
```
array([ 95491.17119147, 117829.51038393, 173423.38071499, 291603.11444202,
        111137.28167129])
```

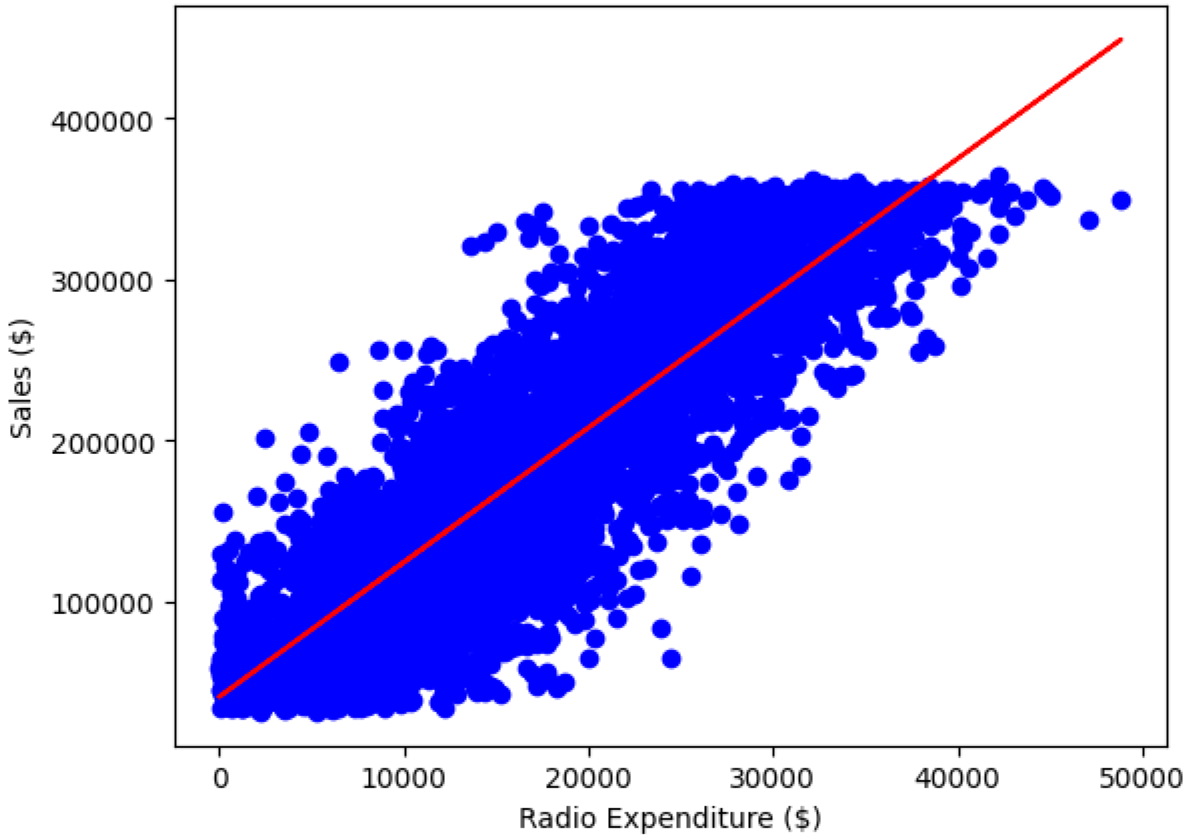
First 5 predictions range from 95,000 to over 290,000.

Visualise linear regression model

In [8]:
```
import matplotlib.pyplot as plt
```

In [9]:
```
# Create scatter plot
plt.scatter(X, y, color="blue")

# Create line plot
plt.plot(X, predictions, color="red")
plt.xlabel("Radio Expenditure ($)")
plt.ylabel("Sales ($)")
plt.show()
```



Near perfect correlation between radio advertising expenditure and sales.

Multiple linear regression

e.g. $y = a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n + b$

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
# Create X and y arrays
X = sales_df.drop(["sales", "influencer"], axis=1).values
y = sales_df["sales"].values
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
```

In [13]:
```
reg = LinearRegression()
reg.fit(X_train, y_train)
```

Out[13]:
```
▾ LinearRegression
LinearRegression()
```

In [14]:
```
# Make predictions
y_pred = reg.predict(X_test)
print("Predictions: {}, Actual Values: {}".format(y_pred[:2], y_test[:2]))
```

```
Predictions: [53176.66154234 70996.19873235], Actual Values: [55261.28 67574.9 ]
```

First 2 predictions appear to be within around 5% of the actual values from the test set.

Regression performance

R^2: quantifies the variance in target values explained by the features (0-1)

$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ (measured in target units, squared)

$RMSE = \sqrt{MSE}$ (measure $RMSE$ in same units as target variable)

In [15]:
```
from sklearn.metrics import mean_squared_error
```

In [16]:
```
r_squared = reg.score(X_test, y_test) # computes R-squared
```

In [17]:
```
rmse = mean_squared_error(y_test, y_pred, squared=False) # rmse (note squared=False)
```

In [18]:
```
# Print the metrics
print("R^2: {}".format(r_squared))
print("RMSE: {}".format(rmse))
```

```
R^2: 0.9990152104759368
RMSE: 2944.4331996001
```

Features explain 99.9% of the variance in sales values, hence the company's advertising strategy seems to be working well.

Cross-validation for R-squared

Use cross-validation: model performance dependent on way we split the data (not representative of model's ability to generalise to unseen data)

More folds = more computationally expensive

In [19]:
```
from sklearn.model_selection import cross_val_score, KFold
```

In [20]:
```
kf = KFold(n_splits=6, shuffle=True, random_state=5)
```

In [21]:
```
reg = LinearRegression()
```

In [22]:
```
cv_scores = cross_val_score(reg, X[:, [1, 2]], y, cv=kf) # computes 6-fold cross-validat
```

In [23]:
```
cv_scores
```

Out[23]:
```
array([0.74451678, 0.77241887, 0.76842114, 0.7410406 , 0.75170022,
       0.74406484])
```

For the chosen features, R-squared for each fold ranged between 0.74 and 0.77. Cross-validation helped us see how performance varies depending on how the data is split.

In [24]:
```
print("Mean:", np.mean(cv_scores))
print("Standard deviation:", np.std(cv_scores))
print("95% confidence interval: ", np.quantile(cv_scores, [0.025, 0.975]))
```

```
Mean: 0.7536937414361207
Standard deviation: 0.012305389070474737
95% confidence interval:  [0.74141863 0.77191916]
```

We get an average score of 0.75 with a low standard deviation.

Regularised regression: Ridge

Linear regression minimises a loss function (chooses coefficient a for each feature variable plus b)

Large coefficients can lead to overfitting

Regularisation penalises large +ve or -ve coefficients

Ridge regression performs regularisation by computing the squared values of the model parameters multiplied by alpha and adding them to the loss function

Ridge regression loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} a_i^2$ (need to choose α hyperparameter)

$\alpha = 0$ = OLS (can lead to overfitting)

Very high α: can lead to underfitting

In [25]:
```
from sklearn.linear_model import Ridge
```

In [26]:
```
alphas = [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
ridge_scores = []
```

In [27]:
```
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    score = ridge.score(X_test, y_test) # obtain R-squared
    ridge_scores.append(score)
```

In [28]:
```
ridge_scores
```

Out[28]:
```
[0.9990152104759369,
 0.9990152104759373,
 0.9990152104759419,
 0.999015210475987,
 0.9990152104764387,
 0.9990152104809561]
```

The scores don't seem to change much as alpha increases, which is indicative of how well the features explain the variance in the target. Even by penalising large coefficients, underfitting does not occur.

Lasso regression for feature importance

Lasso regression loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} |a_i|$

Can use same sklearn code as ridge regression for lasso regression

Lasso can select important features of a dataset

Lasso shrinks coefficients of less important features to 0

Features not shrunk to 0 are selected by lasso

```
In [29]: from sklearn.linear_model import Lasso
```

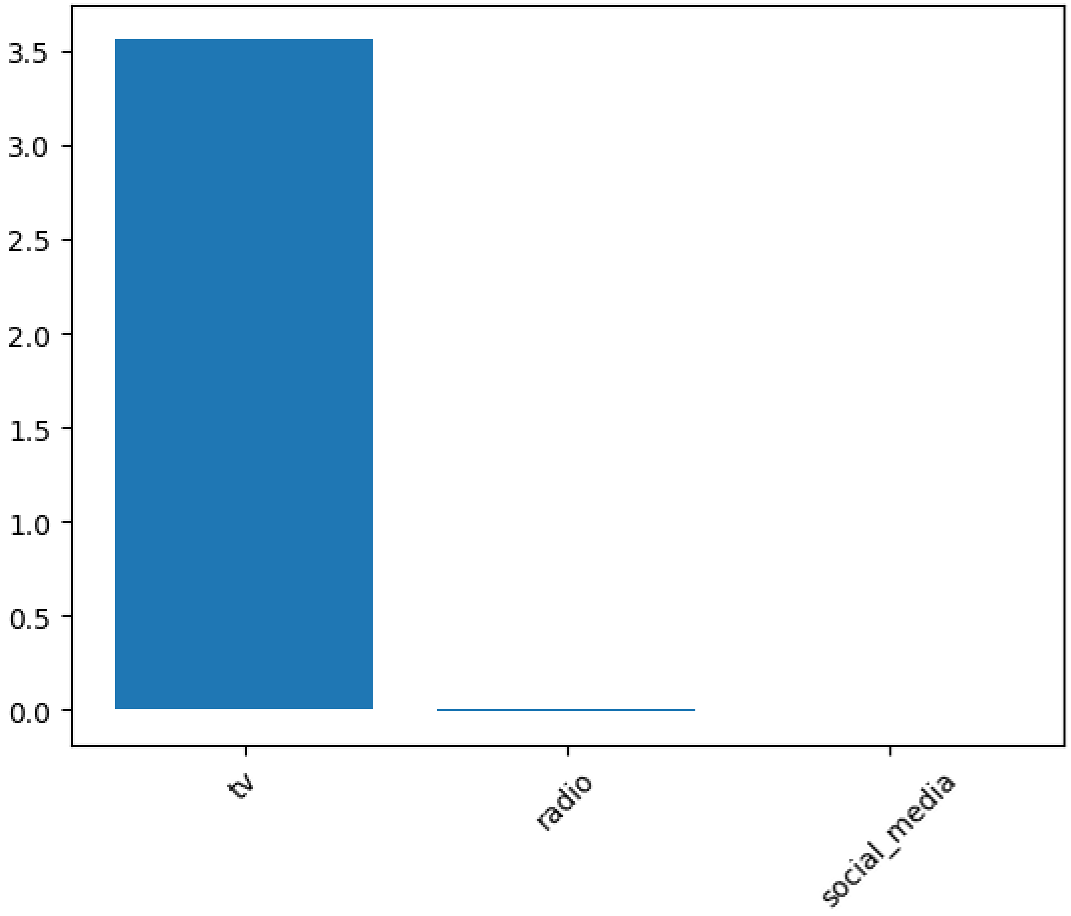
```
In [30]: lasso = Lasso(alpha=0.3)
         lasso.fit(X, y)
         lasso_coef = lasso.coef_
```

```
In [31]: lasso_coef
```

```
Out[31]: array([ 3.56256962, -0.00397035,  0.00496385])
```

```
In [32]: sales_columns = ['tv', 'radio', 'social_media']
```

```
In [33]: plt.bar(sales_columns, lasso_coef)
         plt.xticks(rotation=45)
         plt.show()
```



Expenditure on TV advertising is the most important feature in the dataset to predict sales values.

```
In [ ]:
```