Linear regression

Tips and tricks for the Assignment

Outline

- Test / Train split
- Linear Regression
- One Hot Encoding
- Scaling features
- Adjusted R^2

Train/test split – Assignment task

> 3.2.2 Split your dataset into a training set (80%) and a test set (20%). Use sklearn.model_selection.train_test_split() and set the **random_state to 42.**

Train/test split – tips and tricks

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

(source: sklearn documentation)

Linear Regression – Assignment task

> 3.2.3 Train a linear regression model on the training data. What is the R^2 score for the test data?

Linear Regression – tips and tricks

In [5]:

```
# do the right imports
from sklearn.linear_model import LinearRegression

# create the model
model = LinearRegression()

# Fit the model
model.fit(X,y)
```

Out[5]:

```
LinearRegression()
```

and we do the predictions:

In [6]:

```
predictions = model.predict(X)
print(predictions)
```

Linear Regression – tips and tricks

In [8]:
```
from sklearn.metrics import mean_absolute_error, r2_score

predicted_values = model.predict(X)
mae = mean_absolute_error(y, predicted_values)
r2 = r2_score(y, predicted_values)

print("MAE %.2f" % mae)
print("R^2 %.2f" % r2)
```

Linear Regression with log of dep variable

3.2 Drop the feature "Year", since we have hot-encoded it. Regress log of life expectancy on Alcohol, gdp, population, percentage expenditure, schooling, Developing, and all the dummy variables for the years.

3.2.1. Select the dependent (y) and the independent variables (X).

```
# use the inverse of the logarithm (exponential function) to obtain the value of the prediction
y_pred_years = np.exp(y_pred)[0]
```

One Hot encoding – Assignment task

3 Before regression: One Hot Encoding

Before moving on to the regression, we need to transform the categorical variables as dummy variables for the regression. In order to do so, we use a One Hot Encoder. Pay attention: we need to consider "year" as a categorical variable as well, as it is not a continuous one.

]:
```
from sklearn.preprocessing import OneHotEncoder
```

One Hot encoding – Tips and tricks

In [63]:

```
#onehot
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
transformed = ohe.fit_transform(X[['FullBath']])
print(transformed.toarray())
print(ohe.categories_)

X[ohe.categories_[0]] = transformed.toarray()


```

```
[[0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 ...
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]]
[array([0, 1, 2, 3])]
```

In [64]:

```
X=X.drop(columns=["FullBath", 0])



```

Scaling features – Assignment task

> 3.2.5. Apply a standard scaler to the following columns: Alcohol, gdp, population, percentage expenditure, schooling
>
> Hint: use the scaler on the already split data. Fit-transform the scaler on X_train and apply transform on X_test.

Scaling features – tips and tricks

Setting remainder='passthrough' will mean that **all columns not specified in the list of "transformers" will be passed through without transformation, instead of being dropped**

In [ ]:
```
from sklearn. preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
num_cols=['number_orders', 'number_items', 'number_segments']
scaler=StandardScaler()
preprocessor = ColumnTransformer([('standardization', scaler, num_cols)], remainder='passthrough')
```

In [ ]:
```
encodedX_train = preprocessor.fit_transform(X_train)
encodedX_train = pd.DataFrame(encodedX_train, columns=X_train.columns)
encodedX_train.head(1)
```

Out[345]:

	number_orders	number_items	number_segments	year	month	day
0	1.862725	1.856302	1.288372	2017.0	11.0	5.0

In [ ]:
```
encodedX_test = preprocessor.transform(X_test)
encodedX_test = pd.DataFrame(encodedX_test, columns=X_test.columns)
encodedX_test.head(1)
```

Out[346]:

	number_orders	number_items	number_segments	year	month	day
0	0.386065	0.856442	1.288372	2015.0	10.0	3.0

Pay attention: we train the scaler on the train data, we center mean and standard deviation of train data using its own mean and standard deviation. When transforming the test data, we still use the scaler trained on train data. Also, we do not scale the output variable. (fit transform on X_train, transform on X_test)

Scaling features

> 3.3.3 With the new model, predict, as before, what would be the life expectancy of a (very small) country with an Alcohol consumption of 5 liters, GDP per capita of 800 dollars, a population of 300 individuals, 62 as percentage expenditure, 8 as schooling in year 2000. It is not a developing country.

Here, remember to scale the features with the preprocessor you trained in the previous questions!

Adjusted R^2 – Assignment task

> 3.2.9. Calculate the adjusted R-squared and identify the optimum regression coefficients using linear regression with standardisation.
>
> Hint: calculate the adjusted R-squared for the full model with linear regression and standardisation (as above). The try dropping either one of the columns: `GDP`, `Population` and all the year features and recalculate adjusted R-squared for every new model. Identify which combination of features gives the highest adjusted R-squared.
>
> Check out this documentation file on the adjusted R-squared.

Adjusted R^2 – tips and tricks

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Where

R^2 Sample R-Squared

N Total Sample Size

p Number of independent variable

Adjusted R^2 – tips and tricks

In [ ]:
```
def adjusted_r2(encodedX_train, y_train, X_test, y_test):
    n= ---
    k= ---
    reg_model = --
    reg_model.fit(--, --)
    r2= --
    adjusted_r2 = --
    print(f"{adjusted_r2:.4f}")
```

Adjusted R^2 – tips and tricks

Train the model when you drop `GDP` and calculate the adjusted R-squared.

Adjusted R^2 – tips and tricks

In [ ]:

```
#Create a list with the names of all the features we're interested in

#Select those features from the dataframe

#Do the test train split

#Fit transform the scaler on train data (X_train)

#Transform the test data (X_test) with the scaler

#Use the function for adjusted R^2
```

Adjusted R^2

3.3.5 Calculate the adjusted R-squared and identify the optimum regression coefficients using linear regression with standardisation.

Hint: calculate the adjusted R-squared for the full model with linear regression and standardisation (as above). The try dropping either one of the columns: `GDP`, `Population` and all the year features and recalculate adjusted R-squared for every new model. Identify which combination of features gives the highest adjusted R-squared.

Check out this documentation file on the adjusted R-squared.

Remember to use 42 as the **random_state**

Here you need to use the data of 3.3.2