
multiverse

multiverse

Data Science Unit 3

Linear Regression

Before we start...

- Make sure you are comfortable
- Have water and maybe a strong coffee handy
- If you need a break...take it!
- If you need a stretch - please go ahead!
- Please mute yourselves if you are not talking
- Have your video on at all times

...and let's get started!

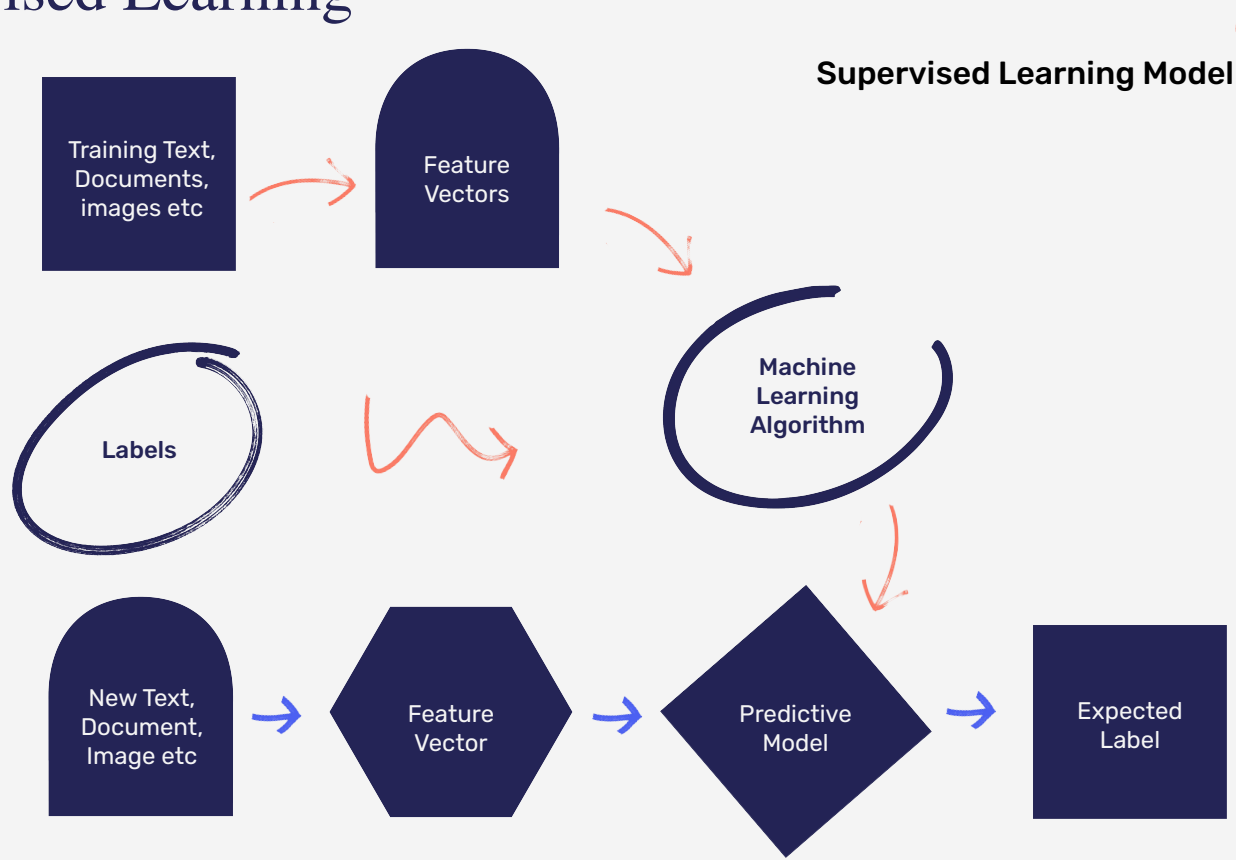
In this session we will...

- **Define** data modeling and simple linear regression
- **Build** a linear regression model using a data set that meets the linearity assumption using the scikit-learn library
- **Understand** and identify multicollinearity in a multiple regression

multiverse

Supervised Learning

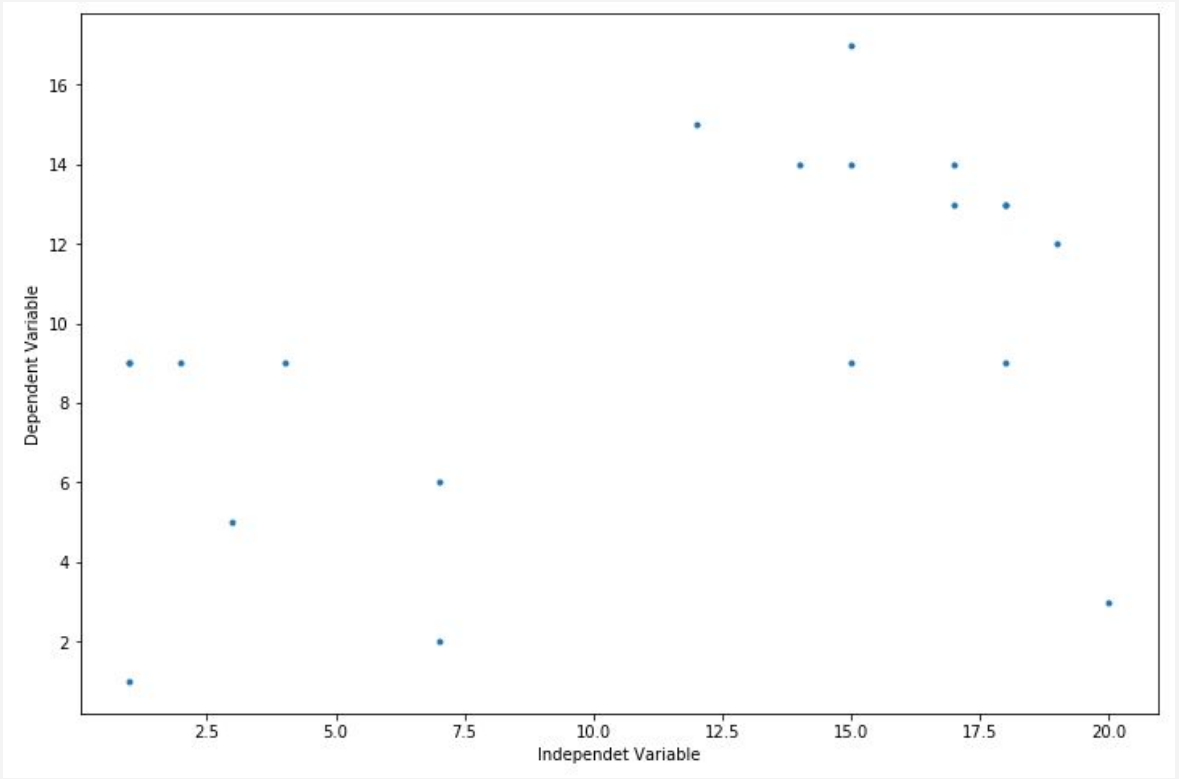
Supervised Learning



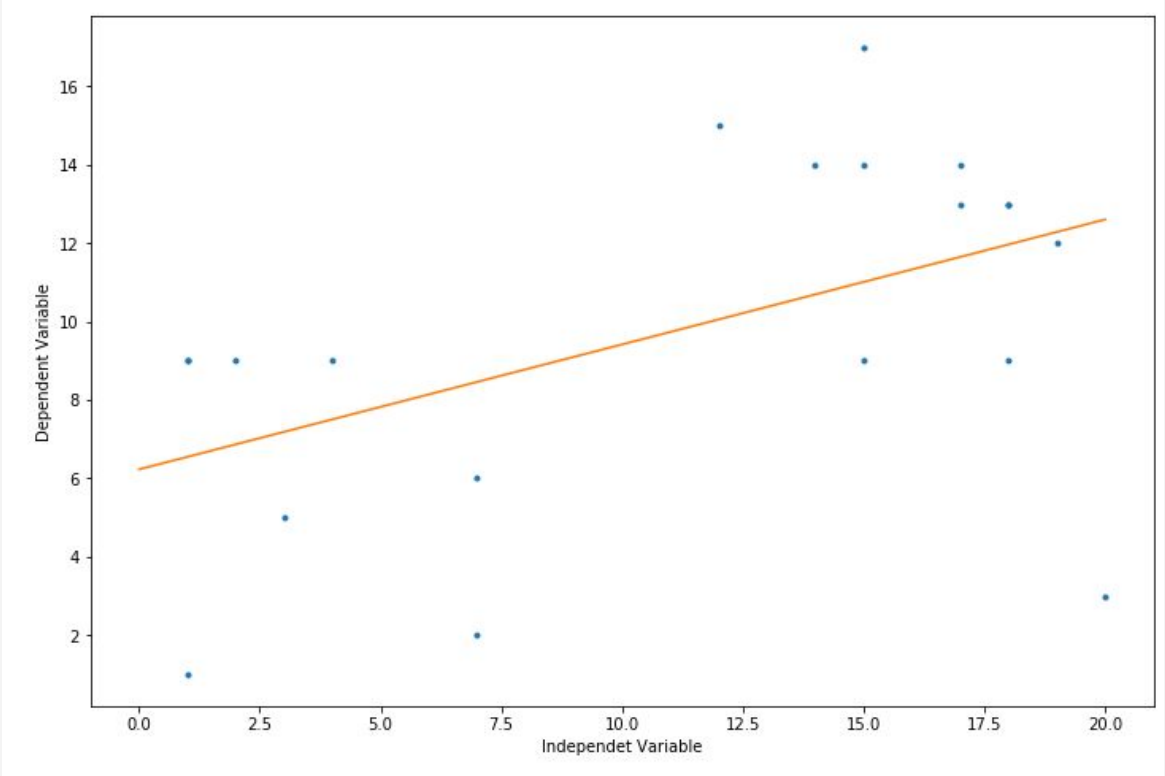
multiverse

Linear Regression

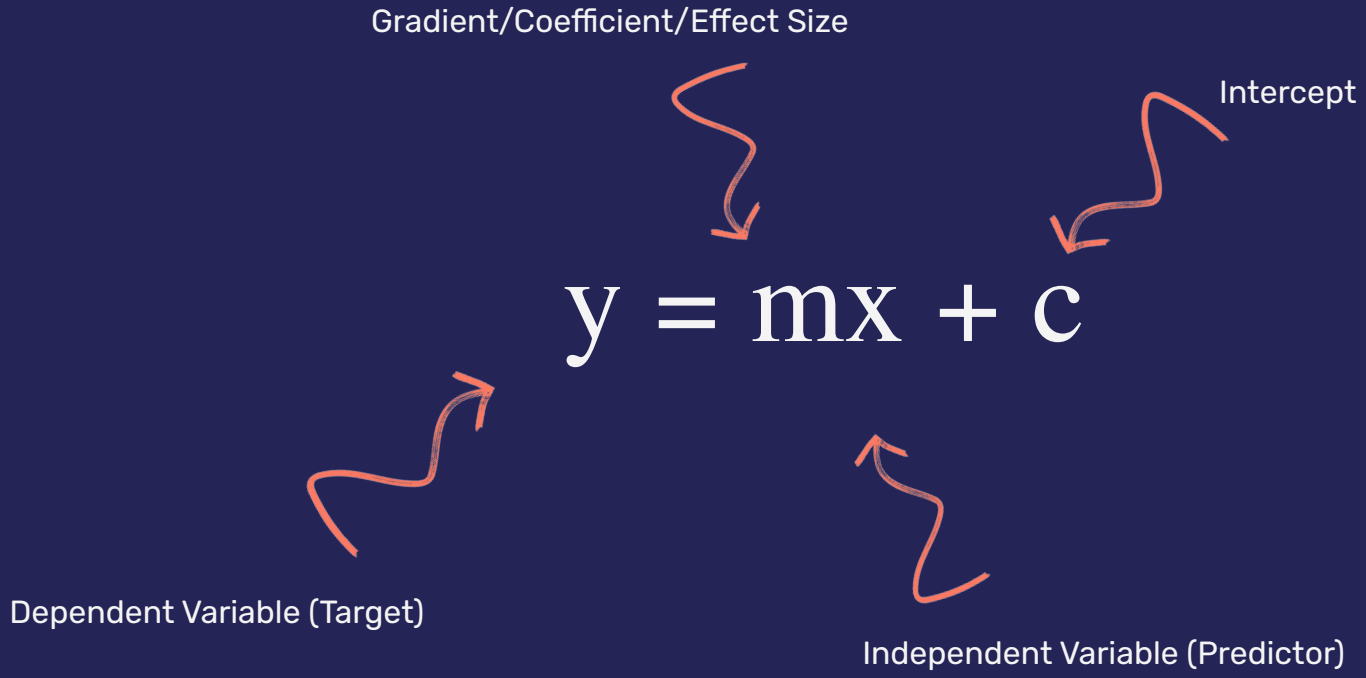
Linear Regression



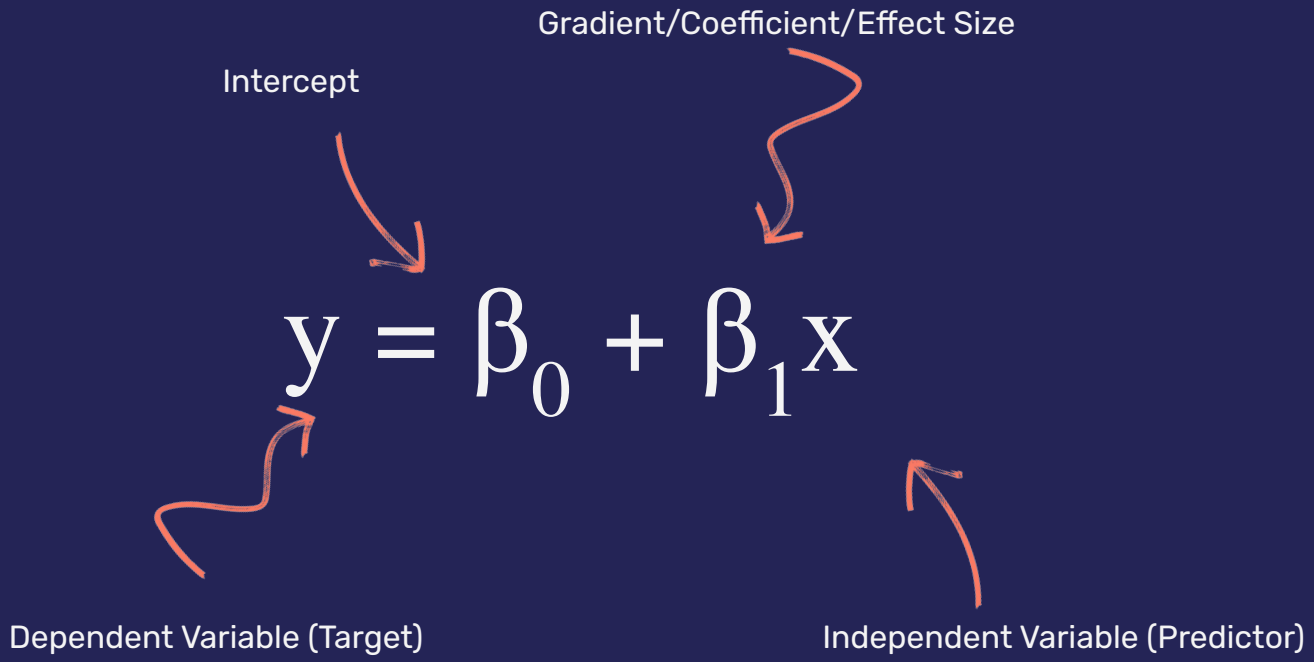
Linear Regression



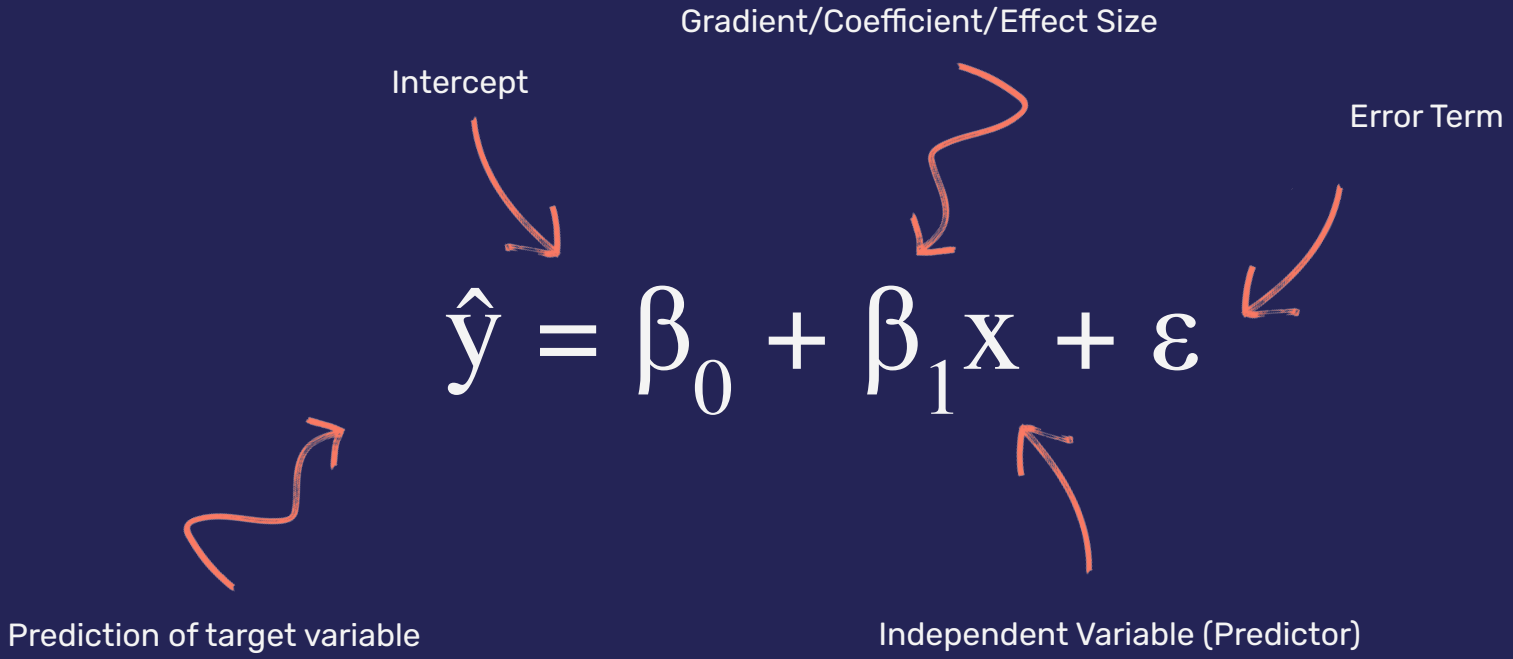
Linear Regression



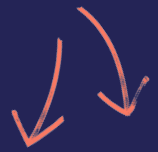
Linear Regression



Linear Regression

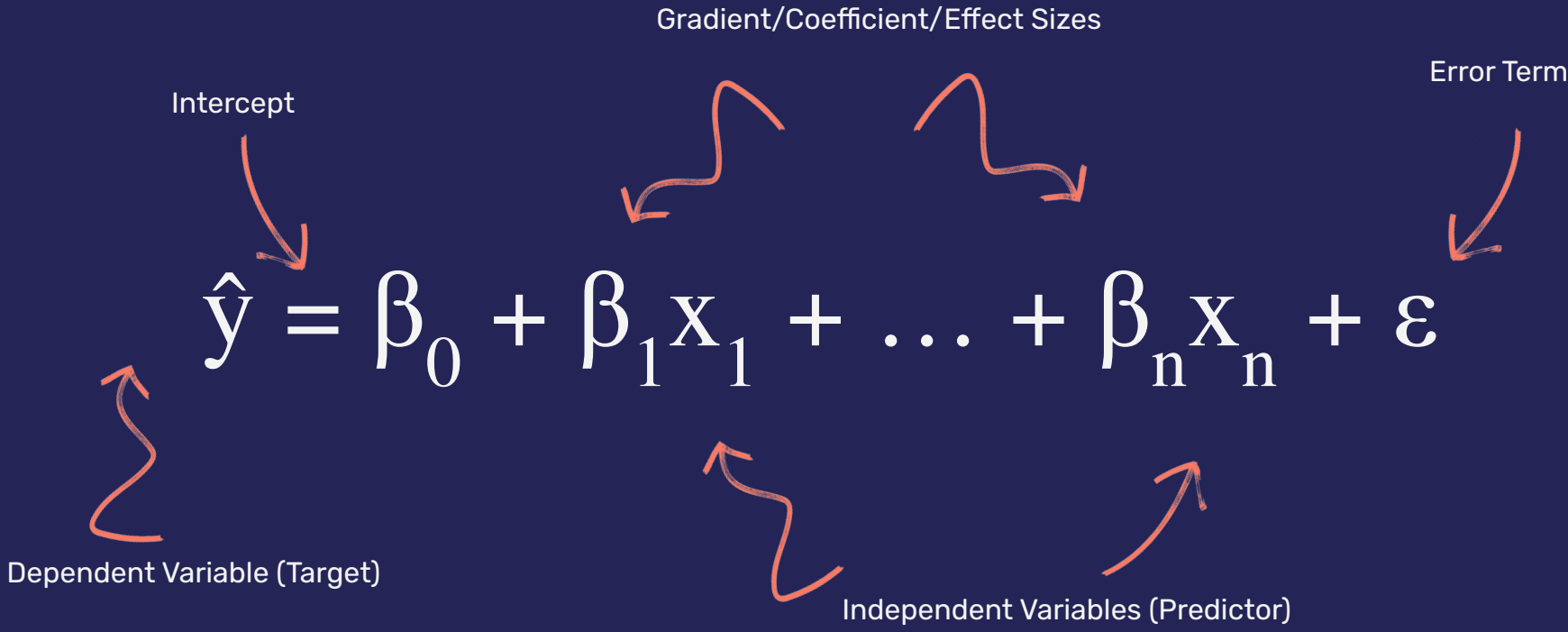


Linear Regression

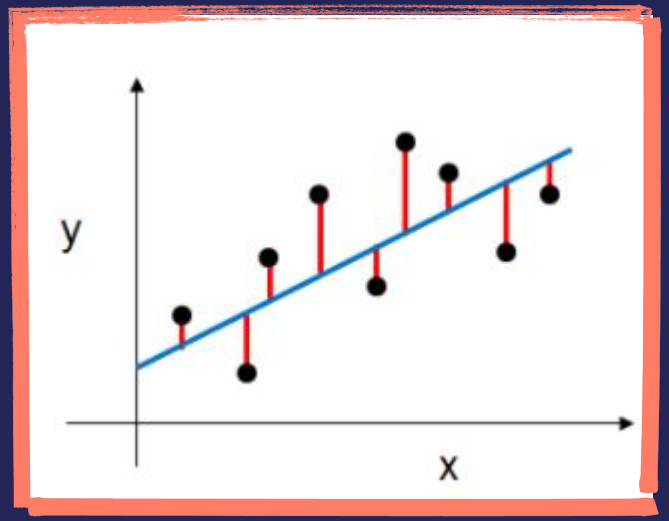


$$test\_score = 20 + 1.2 * time\_spent\_revising$$

Linear Regression



Linear Regression



multiverse

Linear Regression in Python

Linear Regression

Benefits and Drawbacks of scikit-learn
Benefits

- ➔ Consistent interface to machine learning models
- ➔ Provides many turning parameters but with sensible defaults
- ➔ Exceptional documentation
- ➔ Rich set of functionality for companion tasks
- ➔ Active community for development and support

Potential drawbacks:

- ➔ Harder (than R) to get started with machine learning
- ➔ Less emphasis (than R) on model interpretability
  - Scikit-learn tends not to run detailed
  - statistical tests e.g ANOVA
  - For more detail on model fit, try the statsmodels library

Requirements for Working With Data in scikit-learn

- ➔ Features and response should be separate.
- ➔ Features and response should be entirely numeric.
- ➔ Features and response should be Numpy arrays (or easily converted NumPy arrays)
- ➔ Features and response should have specific shapes (outlined below)

Step one:

Import the model you want to use:

```
from sklearn.linear_model import LinearRegression
```

Step two:

Instantiate the estimator (model)

```
lr = LinearRegression()
type(lr)
```

```
sklearn.linear model. base.LinearRegression
```

```
lr
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Step three:

‘Fit’ the model with data (aka training the model)

```
lr.fit(X,y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Step four:

Predict the response of a new variable:

```
X_new=[[0],[10]]
```

```
lr.predict(X_new)
```

```
array([6.23141892, 9.41765203])
```

Step five:

Evaluate your model:

```
print(lr.coef_)
```

```
[0.31862331]
```

```
print(lr.intercept_)
```

```
6.23141891891892
```

```
lr.score(X,y)
```

```
0.2445587873924279
```

multiverse

Let's Practice

multiverse

Multiple Features

Multiple Features

```
X=df[['temp']]
```

```
X=df[['temp','season','weather','humidity']]
```

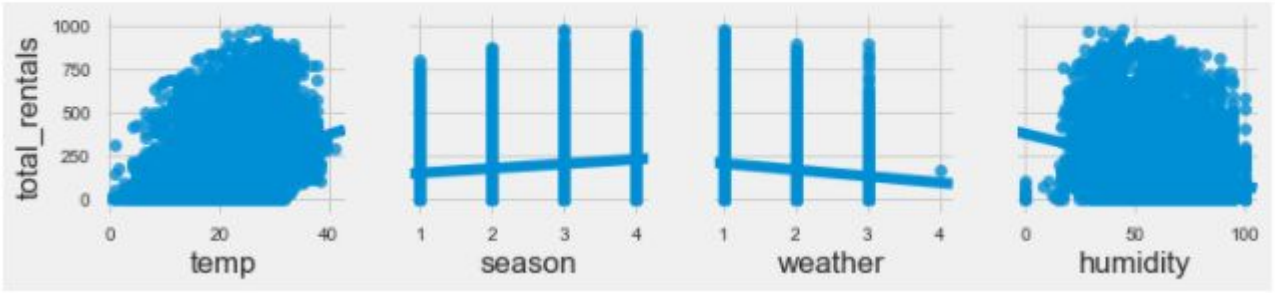
```
lr=LinearRegression().fit(X,y)
```

```
print(lr.coef_)
```

```
[-0.41812686 -0.14661346 -0.43457062  0.1314158 ]
```

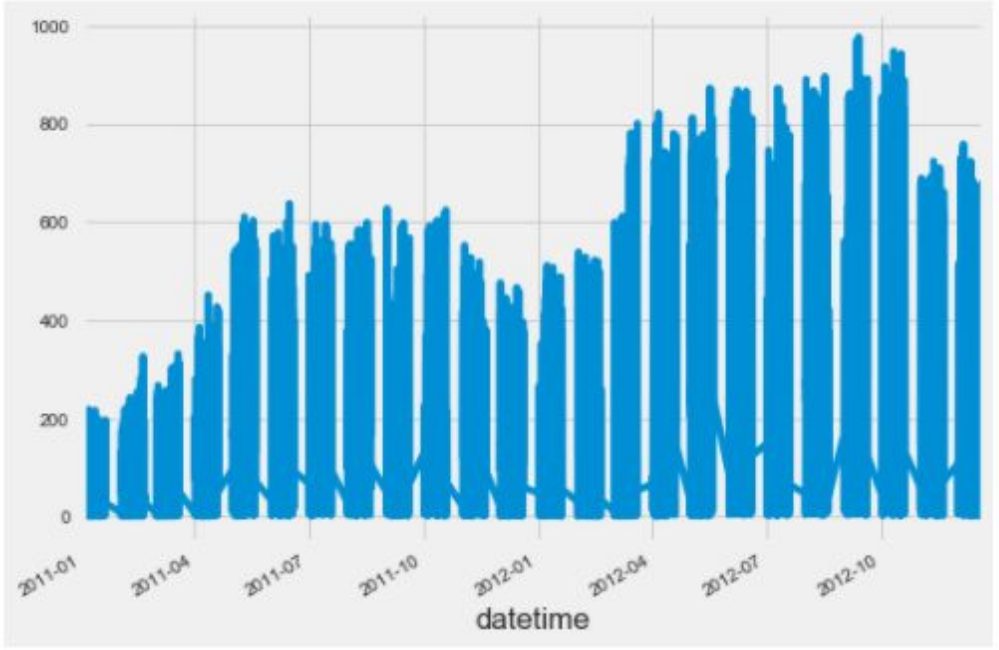
Feature Selection

```
# multiple scatterplots in Seaborn
sns.pairplot(bikes, x_vars=feature_cols, y_vars='total_rentals', kind='reg');
```



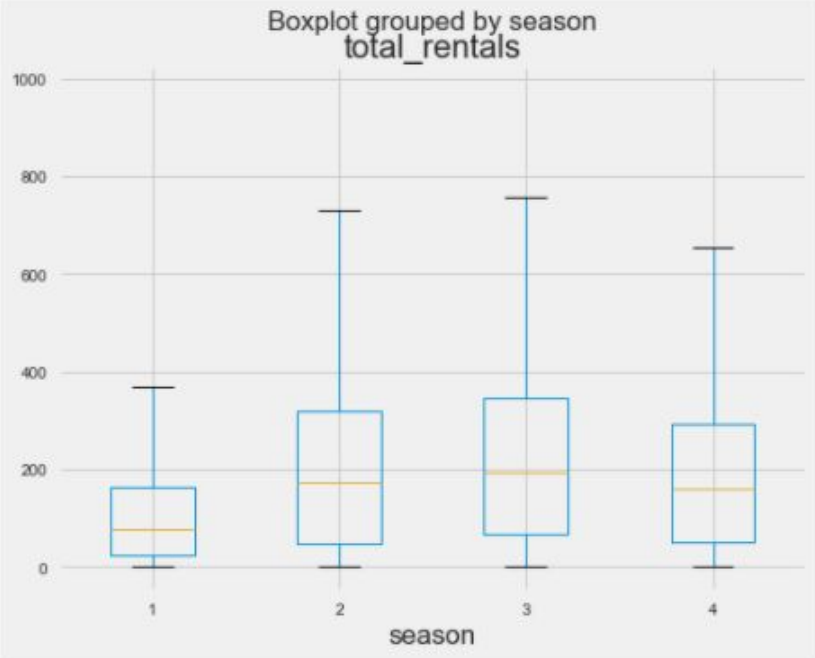
Feature Selection

```
# Line plot of rentals
bikes.total_rentals.plot();
```



Feature Selection

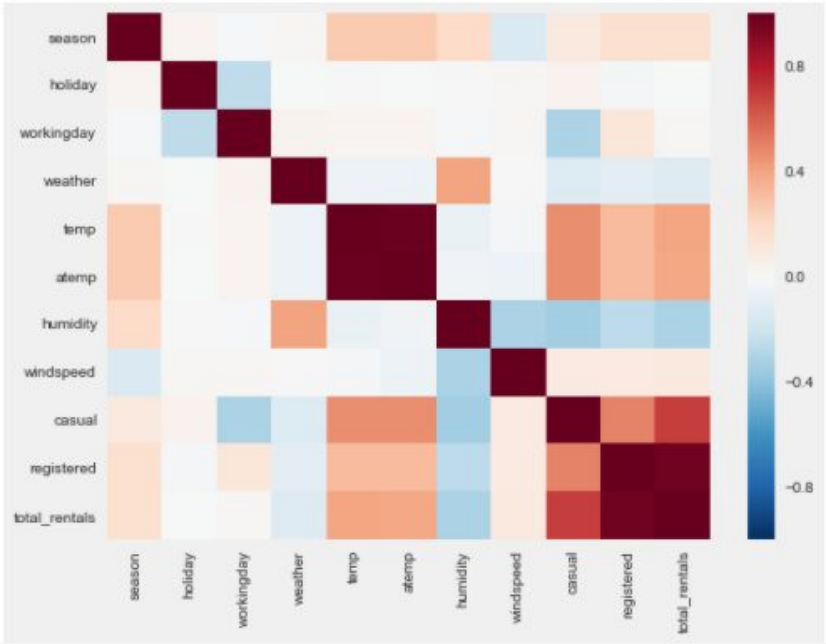
```
# Box plot of rentals, grouped by season
bikes.boxplot(column='total_rentals', by='season');
```



Feature Selection

```
# Visualize correlation matrix in Seaborn using a heat map.
sns.heatmap(bikes.corr())
```

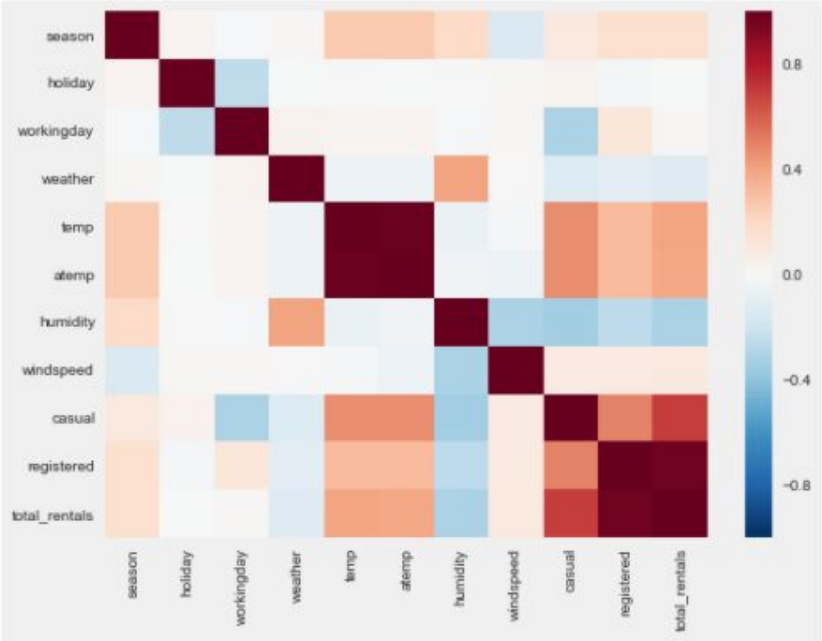
```
<matplotlib.axes._subplots.AxesSubplot at 0x119f8b4a8>
```



Multicollinearity

```
# Visualize correlation matrix in Seaborn using a heat map.
sns.heatmap(bikes.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x119f8b4a8>
```



multiverse

Let's Practice

multiverse

Model Selection

R-Squared

$$R^2 = 1 - \frac{RSS}{TSS}$$

```
lr.score(X,y)
```

```
0.26140210092577043
```

MAE

Mean Absolute Error

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

```
from sklearn.metrics import mean_absolute_error
```

```
print(mean_absolute_error(predicted,true))
```

```
5.751521924637449
```

MSE

Mean Squared Error

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
from sklearn.metrics import mean_squared_error
```

```
print(mean_squared_error(predicted,true))
```

```
43.81916685732635
```

RMSE

Root Mean Squared Error

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
from sklearn.metrics import mean_squared_error
import numpy as np
```

```
print(np.sqrt(mean_squared_error(predicted,true)))
```

```
6.619604735732063
```

multiverse

Let's Practice

multiverse

Feature Engineering

Dummies

Index	Colour
0	Red
1	Blue
2	Blue
3	Green

Dummies

Index	Colour_Red	Colour_Blue	Colour_Green
0	1	0	0
1	0	1	0
2	0	1	0
3	0	0	1

Dummies

Index	Colour_Blue	Colour_Green
0	0	0
1	1	0
2	1	0
3	0	1

Dummies

Index	Colour_Blue	Colour_Green
0	0	0
1		0
2	1	0
3	0	1

Dummies

```
df
```

	colour
0	Red
1	Red
2	Blue
3	Red
4	Blue
5	Green
6	Blue
7	Blue
8	Red
9	Red

Dummies

```
pd.get_dummies(df)
```

	colour_Blue	colour_Green	colour_Red
0	0	0	1
1	0	0	1
2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0
6	1	0	0
7	1	0	0
8	0	0	1
9	0	0	1

Dummies

```
pd.get_dummies(df,drop_first=True)
```

	colour_Green	colour_Red
0	0	1
1	0	1
2	0	0
3	0	1
4	0	0
5	1	0
6	0	0
7	0	0
8	0	1
9	0	1

Dummies

```
pd.get_dummies(df,drop_first=True,columns=['colour'])
```

	colour_Green	colour_Red
0	0	1
1	0	1
2	0	0
3	0	1
4	0	0
5	1	0
6	0	0
7	0	0
8	0	1
9	0	1

Datetime

Function	Effect
pd.to_datetime()	Converts data into a datetime
pd.DatetimeIndex()	Extracts information from datetime data

pd.to_datetime()

```
df
```

	Date
0	3/4/1990
1	8/11/1990
2	1/5/1992
3	20/8/1962
4	18/2/1965
5	30/12/1990

```
df.dtypes
```

```
Date     object
dtype: object
```

```
df.Date.plot()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-61-44c56742b039> in <module>
----> 1 df.Date.plot()

~\anaconda3\lib\site-packages\pandas\plotting\_core.py in __call__(self, *args, **kwargs)
    845                     data.columns = label_name
    846
--> 847         return plot_backend.plot(data, kind=kind, **kwargs)
    848
    849     __call__.__doc__ = __doc__

~\anaconda3\lib\site-packages\pandas\plotting\_matplotlib\__init__.py in plot(data, kind, **kwargs)
     59             kwargs["ax"] = getattr(ax, "left_ax", ax)
     60     plot_obj = PLOT_CLASSES[kind](data, **kwargs)
---> 61     plot_obj.generate()
     62     plot_obj.draw()
     63     return plot_obj.result

~\anaconda3\lib\site-packages\pandas\plotting\_matplotlib\core.py in generate(self)
    259     def generate(self):
    260         self._args_adjust()
--> 261         self._compute_plot_data()
    262         self._setup_subplots()
    263         self._make_plot()

~\anaconda3\lib\site-packages\pandas\plotting\_matplotlib\core.py in _compute_plot_data(self)
    408         # no non-numeric frames or series allowed
    409         if is_empty:
--> 410             raise TypeError("no numeric data to plot")
    411
    412         # GH25587: cast ExtensionArray of pandas (IntegerArray, etc.) to

TypeError: no numeric data to plot
```

pd.to_datetime()

```
df['datetime']=pd.to_datetime(df.Date)
```

```
df
```

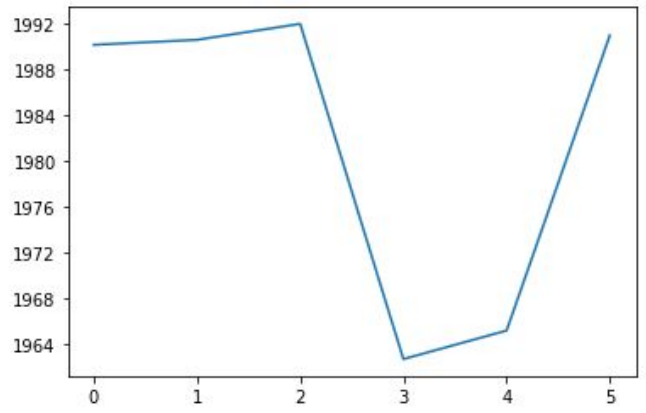
	Date	datetime
0	3/4/1990	1990-03-04
1	8/11/1990	1990-08-11
2	1/5/1992	1992-01-05
3	20/8/1962	1962-08-20
4	18/2/1965	1965-02-18
5	30/12/1990	1990-12-30

```
df.dtypes
```

```
Date                 object
datetime     datetime64[ns]
dtype: object
```

```
df.datetime.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x186a9982348>
```



pd.DatetimeIndex()

```
bike=pd.read_csv('bikeshare.csv')
```

```
bike.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

pd.DatetimeIndex()

```
bike_date['hour']=pd.DatetimeIndex(bike.datetime).hour
bike_date.head()
```

	datetime	year	month	quarter	day	dayofweek	hour
0	2011-01-01 00:00:00	2011	1	1	1	5	0
1	2011-01-01 01:00:00	2011	1	1	1	5	1
2	2011-01-01 02:00:00	2011	1	1	1	5	2
3	2011-01-01 03:00:00	2011	1	1	1	5	3
4	2011-01-01 04:00:00	2011	1	1	1	5	4

pd.DatetimeIndex()

```
bike_date['month']=pd.DatetimeIndex(bike.datetime).month

bike_date_dummy=pd.get_dummies(bike_date,columns=['month'],drop_first=True)
bike_date_dummy.head()
```

	datetime	month_2	month_3	month_4	month_5	month_6	month_7	month_8	month_9	month_10	month_11	month_12
0	2011-01-01 00:00:00	0	0	0	0	0	0	0	0	0	0	0
1	2011-01-01 01:00:00	0	0	0	0	0	0	0	0	0	0	0
2	2011-01-01 02:00:00	0	0	0	0	0	0	0	0	0	0	0
3	2011-01-01 03:00:00	0	0	0	0	0	0	0	0	0	0	0
4	2011-01-01 04:00:00	0	0	0	0	0	0	0	0	0	0	0

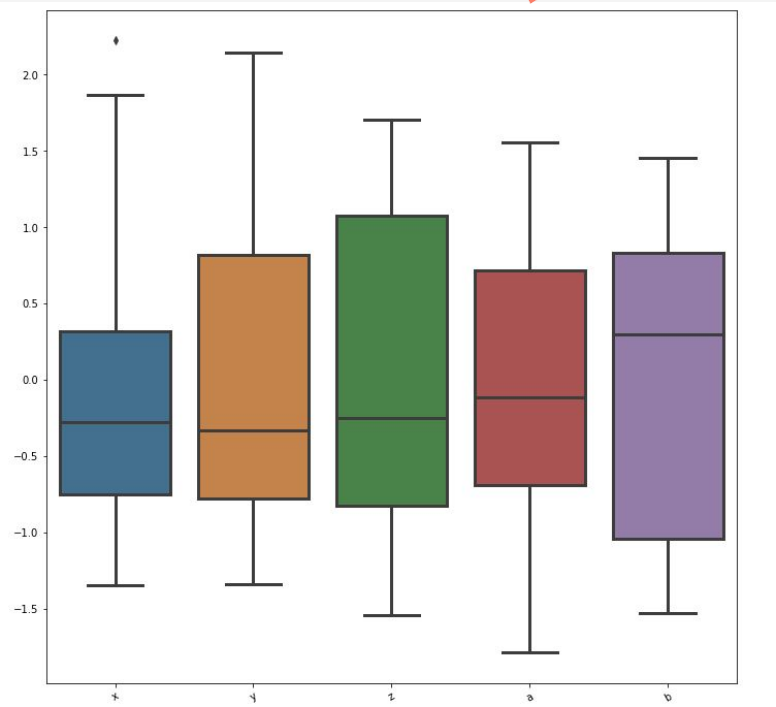
multiverse

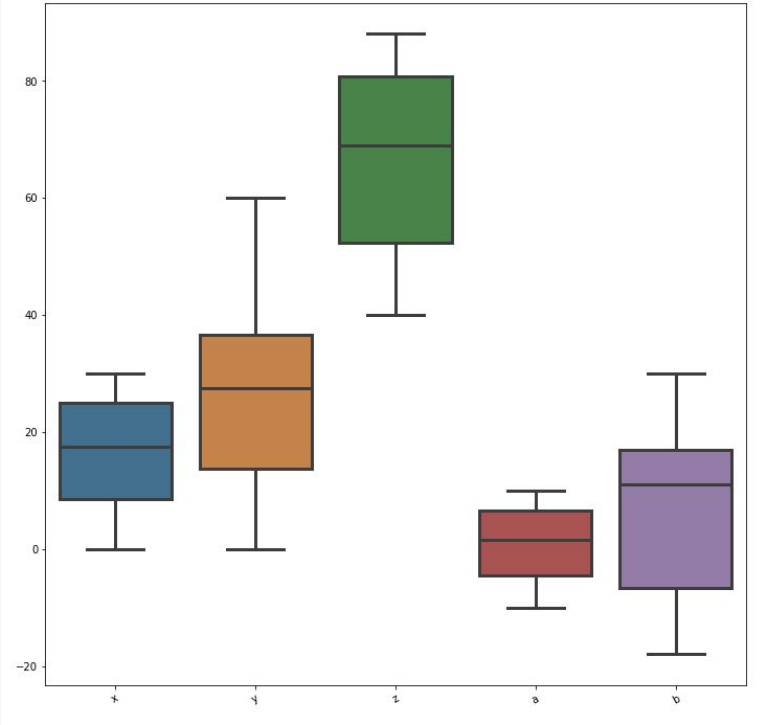
Let's Practice

multiverse

Optimisation

Standardising





Standardising

```
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
```

```
standardised_values=scaler.fit_transform(df)
```

```
lr=LinearRegression().fit(X,y)
lr.score(X,y)
```

```
0.09968133204960594
```

```
lr2=LinearRegression().fit(standardised_values,y)
lr2.score(standardised_values,y)
```

```
0.09968133204960594
```

Regularisation

Regularisation

```
from sklearn.linear_model import Lasso

lasso=Lasso().fit(X,y)
lasso.score(X,y)
```

```
0.13276162918943524
```

```
from sklearn.linear_model import Ridge

ridge=Ridge().fit(X,y)
ridge.score(X,y)
```

```
0.1335271926002526
```

```
lr=LinearRegression().fit(X,y)
lr.score(X,y)
```

```
0.13352731465559609
```

multiverse

Let's Practice

multiverse

Pipelines

Pipelines

Pipelines

```
from sklearn.pipeline import Pipeline
```

```
# Build Pipeline instance, defining in a list of tuples each step. Note you need to name each step, and define it
pipe=Pipeline([('scaler',StandardScaler()),('model',LinearRegression())])
```

```
# Fit our data and evaluate its performance
pipe.fit(X,y)
print(pipe.score(X,y))
```

```
0.15559367802794855
```

multiverse

Summary

Summary

Comparing Linear Regression With Other Models

Advantages of linear regression

- ➔ Simple to explain
- ➔ Highly interpretable
- ➔ Model training and prediction are fast
- ➔ No turning is required (excluding regularization)
- ➔ Features don’t need scaling.
- ➔ Can perform well with a small number of observations.
- ➔ Well understood.

Disadvantages of linear regression

- ➔ Presumes is a linear relationships between the features and the response.
- ➔ Performance is (generally) not competitive with the best supervised learning methods due to high bias.
- ➔ Can’t automatically learn feature interactions.

multiverse

Practice

multiverse

Thank you