Typical problem: we are given $n$ observations of variables $X_1, \ldots, X_p$ and $Y$.  
Goal: Use $X_1, \ldots, X_p$ to try to predict $Y$.  
Example: Cars data compiled using Kelley Blue Book ($n = 805, p = 11$).  
Find a linear model $Y = \beta_1 X_1 + \cdots + \beta_p X_p$.  
In the example, we want:  
price = $\beta_1 \cdot$ mileage + $\beta_2 \cdot$ cylinder + 

Linear regression: classical setting

$p = \text{nb. of variables}, n = \text{nb. of observations}$.  
**Classical setting:**  
- $n \gg p$ ($n$ much larger than $p$). With enough observations, we hope to be able to build a good model.  
- Note: even if the “true” relationship between the variables is not linear, we can include transformations of variables.  
  E.g.  
  $$X_{p+1} = X_1^2, X_{p+2} = X_2^2, \ldots$$  
- Note: adding transformed variables can increase $p$ significantly.  
- A complex model requires a lot of observations.  

**Modern setting:**  
- In modern problems, it is often the case that $n \ll p$.  
- Requires supplementary assumptions (e.g. sparsity).  
- Can still build good models with very few observations.

Classical setting  

**Idea:**  
$$Y \in \mathbb{R}^{n \times 1} \quad X \in \mathbb{R}^{n \times p}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where $x_1, \ldots, x_p \in \mathbb{R}^{n \times 1}$ are the observations of $X_1, \ldots, X_p$.  

- We want $Y = \beta_1 X_1 + \cdots + \beta_p X_p$.  
- Equivalent to solving  

$$Y = X \beta \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$
We need to solve \( Y = X\beta \).

- Obviously, in general, the system has no solution.
- A popular approach is to solve the system in the least squares sense:
  \[
  \hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \| Y - X\beta \|^2.
  \]

How do we compute the solution?

**Calculus approach:**

\[
\frac{\partial}{\partial \beta_i} \| Y - X\beta \|^2 = \frac{\partial}{\partial \beta_i} \left( \sum_{k=1}^{n} (y_k - X_k\beta_1 - X_k\beta_2 - \cdots - X_k\beta_p)^2 \right)
\]
\[
= 2 \sum_{k=1}^{n} (y_k - X_k\beta_1 - X_k\beta_2 - \cdots - X_k\beta_p) \times (-X_k)
\]
\[
= 0.
\]

Therefore,
\[
\sum_{k=1}^{n} X_k(X_k\beta_1 + X_k\beta_2 + \cdots + X_k\beta_p) = \sum_{k=1}^{n} X_ky_k
\]

- Now
\[
\sum_{k=1}^{n} X_k(X_k\beta_1 + X_k\beta_2 + \cdots + X_k\beta_p) = \sum_{k=1}^{n} X_ky_k \quad i = 1, \ldots, p,
\]
is equivalent to:
\[
X^T X\beta = X^T y
\]

(Note: this system always has a solution.)

With a little more work, we can find an explicit solution:

\[
Y - X\hat{\beta} = Y - \text{proj}_{\text{col}(X)}(Y) = \text{proj}_{\text{null}(X^T)}(Y).
\]

**Linear algebra approach:**

Want to solve \( Y = X\beta \).

**Linear algebra approach:** Recall: If \( V \subset \mathbb{R}^n \) is a subspace and \( w \not\in V \), then the best approximation of \( w \) be a vector in \( V \) is
\[
\text{proj}_V(w).
\]

"Best" in the sense that:
\[
\| w - \text{proj}_V(w) \| \leq \| w - v \| \quad \forall v \in V.
\]

Here:
\[
X\beta \in \text{col}(X) = \text{span}(x_1, \ldots, x_p).
\]

If \( Y \not\in \text{col}(X) \), then the best approximation of \( Y \) by a vector in \( \text{col}(X) \) is
\[
\text{proj}_{\text{col}(X)}(Y).
\]

That implies:
\[
X^T(Y - X\hat{\beta}) = 0.
\]

Equivalently,
\[
X^T X\hat{\beta} = X^T Y
\]

(Note: this system always has a solution.)
The least squares theorem

**Theorem (Least squares theorem)**

Let $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. Then

1. $Ax = b$ always has a least squares solution $\hat{x}$.
2. A vector $\hat{x}$ is a least squares solution iff it satisfies the normal equations
   
   $$A^T A \hat{x} = A^T b.$$

3. $\hat{x}$ is unique $\iff$ the columns of $A$ are linearly independent $\iff$ $A^T A$ is invertible. In that case, the unique least squares solution is given by
   
   $$\hat{x} = (A^T A)^{-1} A^T b.$$

Building a simple linear model with Python (cont.)

The scikit-learn package provides a lot of very powerful functions/objects to analyse datasets.

Typical syntax:

1. Create object representing the model.
2. Call the fit method of the model with the data as arguments.
3. Use the predict method to make predictions.

```python
from sklearn.linear_model import LinearRegression
lin_model = LinearRegression(fit_intercept=True)
lin_model.fit(x, y)

print lin_model.coef_
print lin_model.intercept_
```

We obtain $\text{price} \approx -0.17 \cdot \text{mileage} + 24764.5$.

Measuring the fit of a linear model

How good is our linear model?

- We examine the residual sum of squares:

  $$\text{RSS}(\hat{\beta}) = \|y - X\hat{\beta}\|^2 = \sum_{k=1}^n (y_i - \hat{y}_i)^2.$$

  $$((y - \text{lin_model.predict(x)})**2).\text{sum()}$$

  We find: 76855792485.91. Quite a large error... The average absolute error:

  $$\text{abs}(y - \text{lin_model.predict(x))).mean()$$

  is 7596.28. Not so good...

- We examine the distribution of the residuals:

  ```python
  import matplotlib.pyplot as plt
  plt.hist(y - lin_model.predict(x))
  plt.show()
  ```

Building a simple linear model with Python

The file JSE_Car_Lab.csv:

Loading the data with the headers using Pandas:

```python
import pandas as pd
data = pd.read_csv('./data/JSE_Car_Lab.csv', delimiter=',')
```

We extract the numerical columns:

```python
y = np.array(data['Price'])
x = np.array(data['Mileage'])
x = x.reshape(len(x), 1)
```

11/15 10/15
Measuring the fit of a linear model (cont.)

Histogram of the residuals:

- Non-symmetric.
- Heavy tail.

The heavy tail suggests there may be outliers.
- It also suggests transforming the response variable using a transformation such as $\log$, $\sqrt{\cdot}$, or $1/x$.

Plotting the residuals as a function of the fitted values, we immediately observe some patterns.

Outliers? Separate categories of cars?

Improving the model

- Add more variables to the model.
- Select the best variables to include.
- Use transformations.
- Separate cars into categories (e.g. exclude expansive cars).
- etc.

For example, let us use all the variables, and exclude Cadillacs from the dataset.

- Much more symmetric.
- Closer to a Gaussian distribution.

Average absolute error drops to 4241.21.