Week 4

Model Development

1. Simple/Multiple Lin Reg
2. Model Eval Using Visualization
3. Polynomial Regression ; Pipelines
4. R^2 + MSE for In-Sample Eval
5. Prediction + Decision Making

---

1. Linear regression refers to 1 I.V. to make a prediction.

- Simple lin reg (SLR) : $\hat{y} = \underset{\text{int}}{b_0} + \underset{\text{slope}}{b_1} x$ (Data pts are usually stored into df or numpy arrays)
  - predictor (IV) → x
  - target (DV) → y
  - In Python:

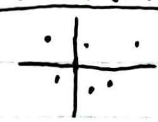
```
1 from sklearn.linear_model import LinearRegression()
2 lm = LinearRegression()   uses lm as a construct for lin reg function
3 x = df['var1']
4 y = df['var2']
5 lm.fit(x,y)
```

- Multiple lin reg (MLR): $\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + \ldots$
  - Predictors (IVs) → $x_1, x_2 \ldots$
  - Target (DV) → y
  - In Python:

```
1, 2  "   "
3 z = df[['var1', 'var2', ...]]
4 y = df['vary']
5 lm.fit(z,y)   ← yhat = lm.predict(x) : prediction
```

---

2. Residual Plots



x-value = x-value
y-value = residual

Residual = $\hat{y} - y$

If the points on a residual plot are spread out around the x-axis, lin reg is appropriate. If there is a slope or curve of residual points (or var changes w/ x), then lin reg isn't accurate.

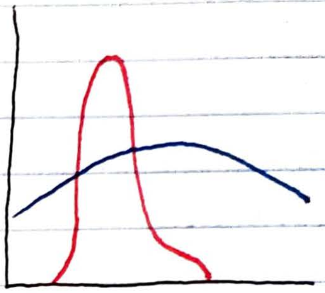
In Python:

```
import seaborn as sns
sns.residplot(df['var1'], df['var2'])
```

Distribution Plots count $\hat{y}$ vs y

In Python:

```
ax1 = sns.distplot(df['vary'], hist=False,
color='r', label='Actual Value')
sns.distplot(Yhat, hist=False, color='b',
label='Fitted Values', ax=ax1)
```



3. Polynomial Regression
   - Best for desc curvilinear rel's:
   - Quadratic: $\hat{y} = b_0 + b_1 x_1 + b_2 (x_2)^2 \ldots$
   - In Python:

```
f = np.polyfit(x, y, deg)
p = np.poly1d(f)
print(p)
```

Multidimensional Poly Reg

```
from sklearn.preprocessing import PolynomialFeatures
pr = PolynomialFeatures(degree=#)
x_polly = pr.fit_transform(x['var1','var2'], include_bias=False)
```

Pre-processing: Normalize

```
from sklearn.preprocessing import StandardScaler
SCALE = StandardScaler()
SCALE.fit(x['var1', 'var2'])
x_scale = SCALE.transform(x['var1', 'var2'])
```

Pipelines are a way to simplify a code.

Normalization → Polynomial Transform → Lin Reg (prediction)

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear-model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
Input = [('scale', StandardScaler()), ('polynomial',
PolynomialFeatures(degree=#)), ('mode', 'LinearRegression())]
pipe = Pipeline(Input)
Pipe.train(x['var1', 'var2',...], y)
yhat = Pipe.predict(x['var1', 'var2'])
```

4. In-Sample Eval
   - A way to determine how good the model fits on dataset
   - Mean squared error (MSE)
     - Square the residual for each value
     - MSE = $\frac{\Sigma(\text{residual}^2)}{\#\text{ samples}}$
     - In Python:

MSE for MLR models will be smaller than MSE for SLR models because ↑ vars → ↑ accuracy

```
from sklearn.metrics import mean_squared_error
mean_squared_error(df['var1'], Y-predict-simple-fit)
```

   - R^2: coeff of determination
     - Measures how close the data is to reg line (or % of var on target var (y) exp. by model)
     - $R^2 = (1 - \frac{\text{MSE of reg line}}{\text{MSE of avg of data}})$
     - See last line of 4.1

5. Prediction + Decision Making
   1. Do results make sense?
   2. Always visualize
   3. Numerical measures for eval
   4. Comparing models