In [ ]:
```python
import os
import numpy as np
import pandas as pd
import statsmodels.api as sm

from rich import inspect
from icecream import ic
from src.print import print
```

1.) Import Data from FRED

In [ ]:
```python
data = pd.read_csv(
    os.path.join('data', 'TaylorRuleData.csv'),
    index_col = 0
)
```

In [ ]:
```python
# Checking the last rows of the data.
data.tail()
```

Out[ ]:

	FedFunds	Unemployment	HousingStarts	Inflation
2023-08-01	5.33	3.8	1305.0	306.269
2023-09-01	5.33	3.8	1356.0	307.481
2023-10-01	5.33	3.8	1359.0	307.619
2023-11-01	5.33	3.7	1560.0	307.917
2023-12-01	5.33	3.7	NaN	NaN

In [ ]:
```python
# Converting the index to datetime.
data.index = pd.to_datetime(data.index)
```

In [ ]:
```python
# Checking amount of nan values and dropping them.
_ = ic(data.isna().sum())
```

```
ic| data.isna().sum(): FedFunds          90
                       Unemployment      12
                       HousingStarts    145
                       Inflation          1
                       dtype: int64
```

In [ ]:
```python
data = data.dropna()
```

2.) Do Not Randomize, split your data into Train, Test Holdout

In [ ]:
```python
# Names of the features and target.
x_names = ['Unemployment', 'HousingStarts', 'Inflation']
y_names = ['FedFunds']
```

In [ ]:
```python
# Defining the percentage of the data to be used for training,
# testing and holdout.

train_size = 0.6
test_size = 0.2
hold_size = 0.2

_ = ic(train_size + test_size + hold_size)
```

```
ic| train_size + test_size + hold_size: 1.0
```

In [ ]:
```python
split_1 = np.ceil(data.shape[0] * train_size).astype(int)
split_2 = split_1 + np.ceil(data.shape[0] * test_size).astype(int)
```

```
data_in = data.iloc[:split_1]
data_out = data.iloc[split_1:split_2]
data_hold = data.iloc[split_2:]
```

In [ ]:
```
X_in = data_in[x_names]
y_in = data_in[y_names]
X_out = data_out[x_names]
y_out = data_out[y_names]
X_hold = data_hold[x_names]
y_hold = data_hold[y_names]
```

In [ ]:
```
# Add Constants
X_in = sm.add_constant(X_in)
X_out = sm.add_constant(X_out)
X_hold = sm.add_constant(X_hold)
```

3.) Build a model that regresses FF~Unemp, HousingStarts, Inflation

In [ ]:
```
model1 = sm.OLS(y_in, X_in).fit()
print(model1.summary())
```

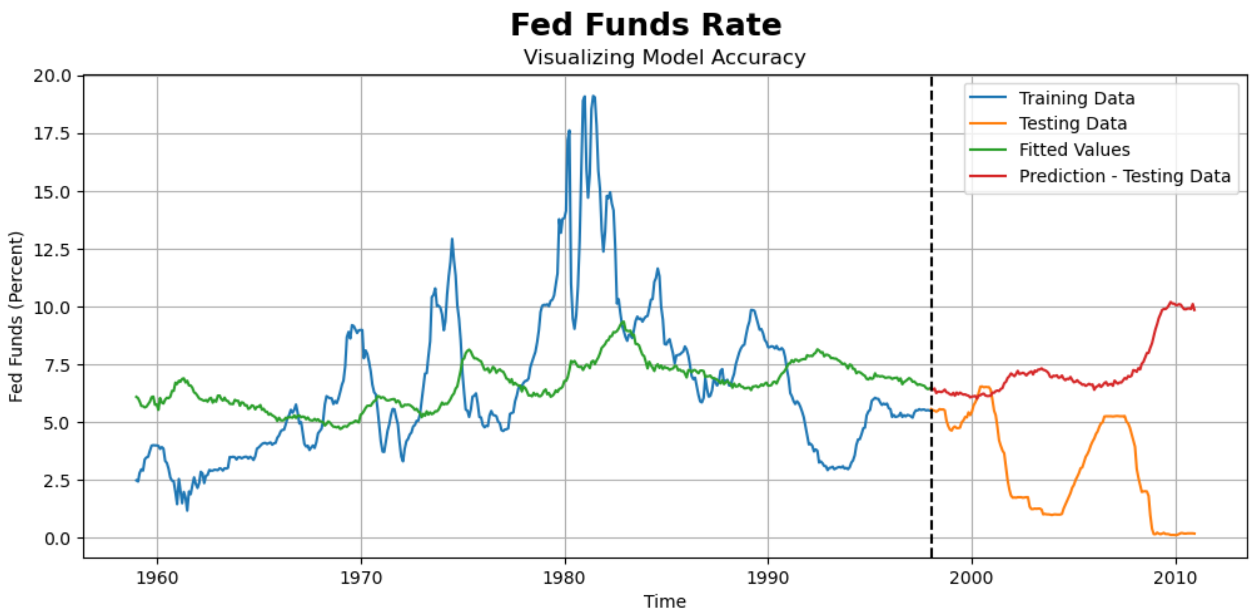
```
                            OLS Regression Results
==============================================================================
Dep. Variable:               FedFunds   R-squared:                       0.088
Model:                            OLS   Adj. R-squared:                  0.082
Method:                 Least Squares   F-statistic:                     14.87
Date:                Thu, 11 Jan 2024   Prob (F-statistic):           2.93e-09
Time:                        07:51:37   Log-Likelihood:                -1204.1
No. Observations:                 468   AIC:                             2416.
Df Residuals:                     464   BIC:                             2433.
Df Model:                           3
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const             3.4728      0.984      3.530      0.000       1.540       5.406
Unemployment      0.5331      0.106      5.050      0.000       0.326       0.741
HousingStarts    -0.0005      0.000     -1.054      0.292      -0.001       0.000
Inflation         0.0076      0.004      2.155      0.032       0.001       0.015
==============================================================================
Omnibus:                       78.107   Durbin-Watson:                   0.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              123.676
Skew:                           1.040   Prob(JB):                     1.39e-27
Kurtosis:                       4.420   Cond. No.                     1.03e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.03e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

4.) Recreate the graph for your model

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

fig, ax = plt.subplots(figsize = (12,5))

# Observed Data
ax.plot(y_in, label = 'Training Data')
ax.plot(y_out, label = 'Testing Data')

# Predicted Data
ax.plot(model1.predict(X_in), label = 'Fitted Values')
ax.plot(model1.predict(X_out), label = 'Prediction - Testing Data')
```

```python
# Testing Data Separation Line
ax.axvline(x = y_out.index[0], color = 'black', linestyle = '--')

# Legend, axes, title
plt.grid()
ax.legend()
ax.set_ylabel('Fed Funds (Percent)')
ax.set_xlabel('Time')
ax.set_title('Visualizing Model Accuracy')
fig.suptitle('Fed Funds Rate', fontsize = 18, weight = 'bold')
```

Out[ ]: Text(0.5, 0.98, 'Fed Funds Rate')



"All Models are wrong but some are useful" - 1976 George Box

5.) What are the in/out of sample MSEs

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

In [ ]:
```python
in_mse_1 = mean_squared_error(y_in, model1.fittedvalues)
out_mse_1 = mean_squared_error(y_out, model1.predict(X_out))
```

In [ ]:
```python
print('Insample MSE : ', round(in_mse_1, 2))
print('Outsample MSE : ', round(out_mse_1, 2))
```

```
Insample MSE :  10.05
Outsample MSE :  27.19
```

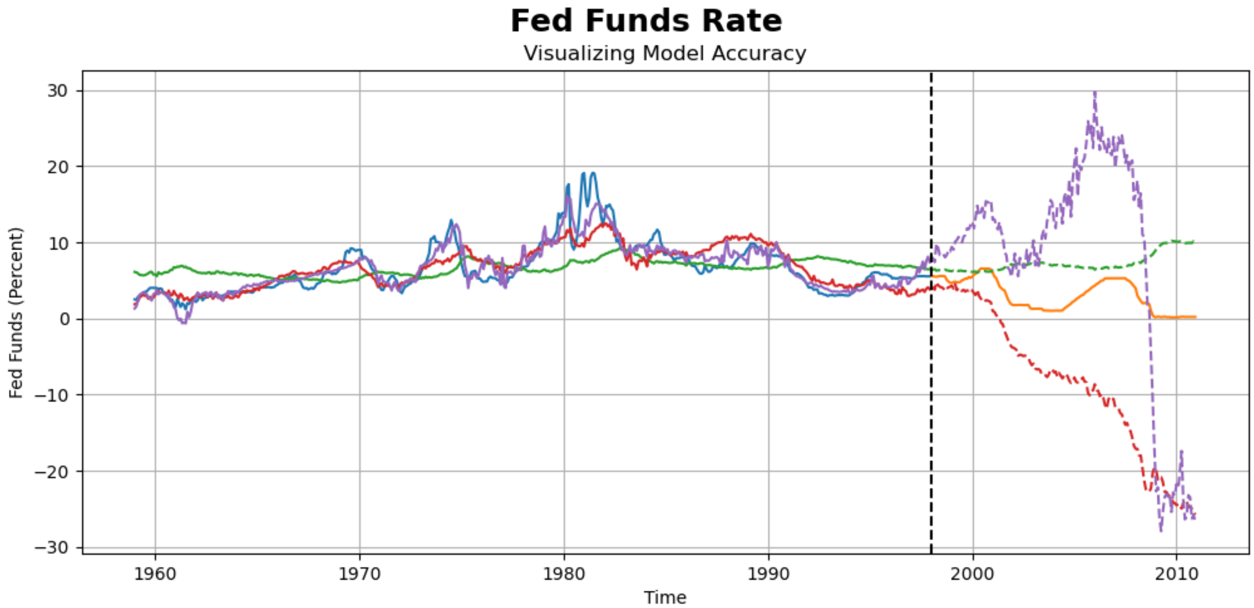
6.) Using a for loop. Repeat 3,4,5 for polynomial degrees 1,2,3

In [ ]:
```python
from sklearn.preprocessing import PolynomialFeatures

max_degrees = 3

results = {}
for degree in range(1,max_degrees+1):

    poly = PolynomialFeatures(degree = degree)
```

```python
    X_in_poly = pd.DataFrame(
        poly.fit_transform(X_in)
    ).set_index(X_in.index)

    X_out_poly = pd.DataFrame(
        poly.fit_transform(X_out)
    ).set_index(X_out.index)

    temp_model = sm.OLS(y_in, X_in_poly).fit()

    results[degree] = {
        'model' : temp_model,
        'y_in_pred' : temp_model.fittedvalues,
        'y_out_pred' : temp_model.predict(X_out_poly),
        'in_mse' : mean_squared_error(y_in, temp_model.fittedvalues),
        'out_mse' : mean_squared_error(y_out, temp_model.predict(X_out_poly))
    }

fig, ax = plt.subplots(figsize = (12,5))

# Observed Data
ax.plot(y_in, label = 'Training Data')
ax.plot(y_out, label = 'Testing Data')

# Predicted Data
for degree in results.keys():
    ax.plot(
        results[degree]['y_in_pred'],
        label = f'Fitted Values - Degree {degree}'
    )
    # Extra color for the testing data
    color = ax.lines[-1].get_color()

    ax.plot(
        results[degree]['y_out_pred'],
        label = f'Prediction - Testing Data - Degree {degree}',
        color = color,
        linestyle = '--'
    )

# Testing Data Separation Line
ax.axvline(x = y_out.index[0], color = 'black', linestyle = '--')

# Legend, axes, title
plt.grid()
ax.set_ylabel('Fed Funds (Percent)')
ax.set_xlabel('Time')
ax.set_title('Visualizing Model Accuracy')
fig.suptitle('Fed Funds Rate', fontsize = 18, weight = 'bold')
```

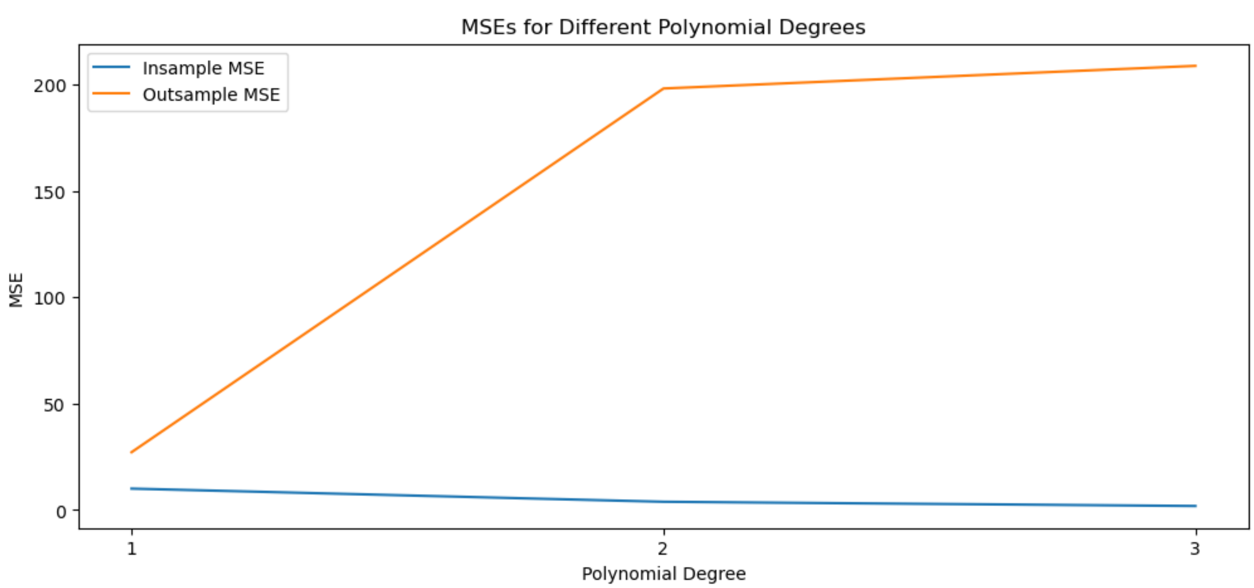
Out[ ]:

```
Text(0.5, 0.98, 'Fed Funds Rate')
```



In [ ]:
```python
# Plot in and out of sample MSEs
fig, ax = plt.subplots(figsize = (12,5))

ax.plot(
    [results[degree]['in_mse'] for degree in results.keys()],
    label = 'Insample MSE'
)
ax.plot(
    [results[degree]['out_mse'] for degree in results.keys()],
    label = 'Outsample MSE'
)

ax.set_xticks(range(0,max_degrees))
ax.set_xticklabels(range(1,max_degrees+1))
ax.set_xlabel('Polynomial Degree')
ax.set_ylabel('MSE')
ax.set_title('MSEs for Different Polynomial Degrees')
ax.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x72e46a0c8ed0>



7.) State your observations :

As the degree of the polynomial increases, the in-sample MSE decreases, while the out-of-sample MSE increases, indicating a clear sign of overfitting. In the plot of observed versus predicted values, it is clear that even though a higher-degree polynomial fits the training data better, higher-degree polynomials typically have highly volatile predictions.