Generalised Regression

In this notebook, we will build a generalised regression model on the **electricity consumption** dataset. The dataset contains two variables - year and electricity consumption.

In [1]:
```
#importing libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn import metrics
```

In [2]:
```
#fetching data
elec_cons = pd.read_csv("total-electricity-consumption-us.csv",  sep = ',', he
ader= 0 )
elec_cons.head()
```

Out[2]:

	Year	Consumption
0	1920	57125
1	1921	53656
2	1922	61816
3	1923	72113
4	1924	76651

In [3]:
```
# number of observations: 51
elec_cons.shape
```

Out[3]: (51, 2)

In [4]:
```
# checking NA
# there are no missing values in the dataset
elec_cons.isnull().values.any()
```

Out[4]: False

In [5]:
```
size = len(elec_cons.index)
index = range(0, size, 5)

train = elec_cons[~elec_cons.index.isin(index)]
test = elec_cons[elec_cons.index.isin(index)]
```

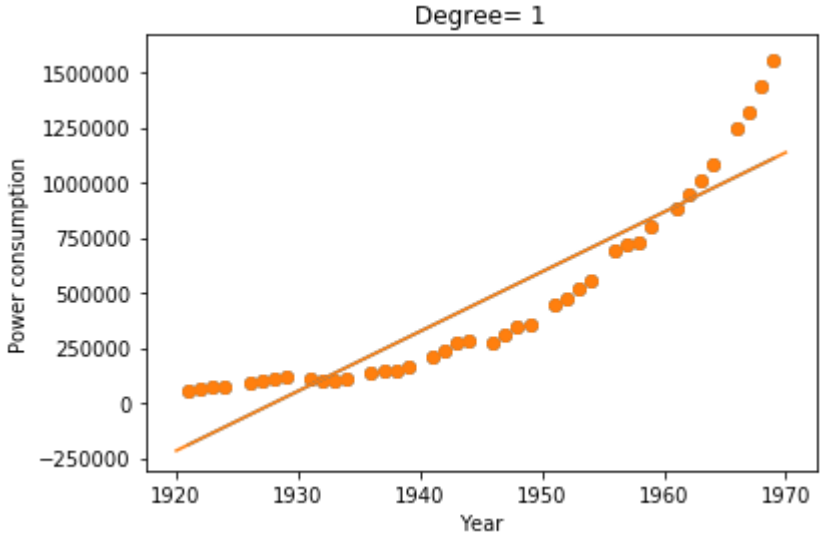
In [6]:
```
print(len(train))
print(len(test))
```

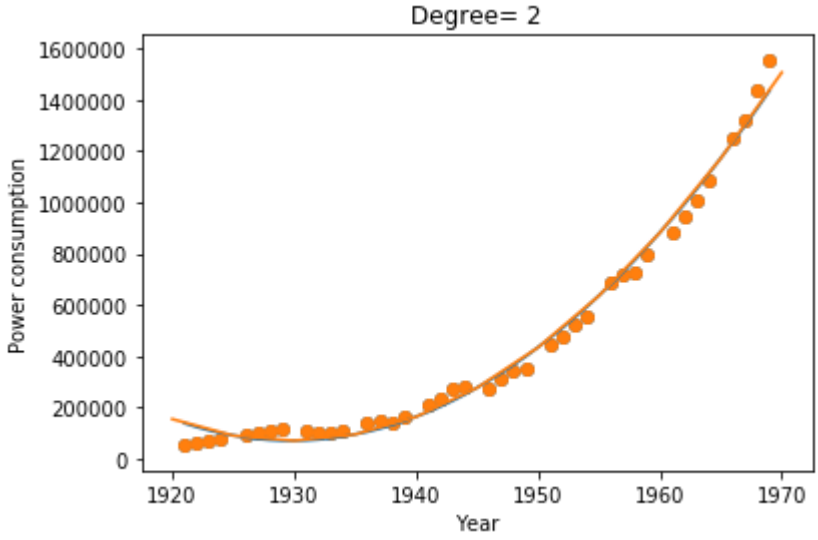
```
40
11
```

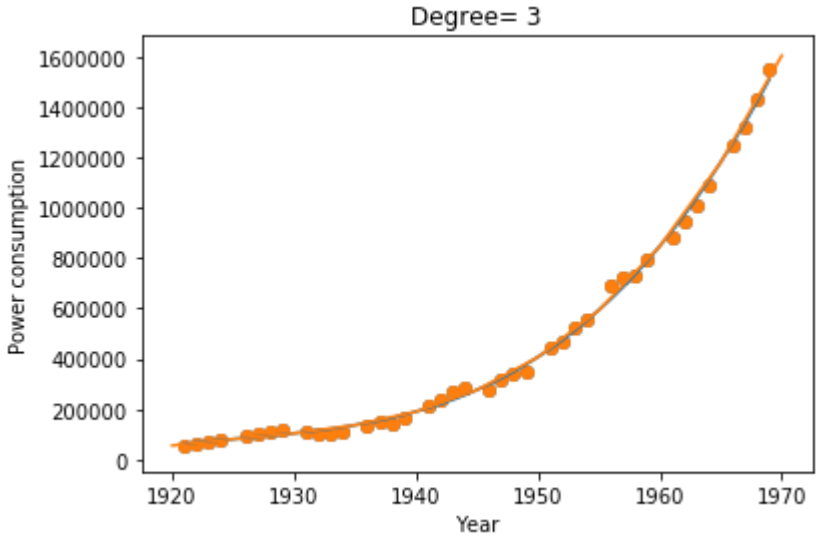
In [7]:
```
# converting X to a two dimensional array, as required by the learning algorit
hm
X_train = train.Year.reshape(-1,1) #Making X two dimensional
y_train = train.Consumption

X_test = test.Year.reshape(-1,1) #Making X two dimensional
y_test = test.Consumption
```

```
/Users/IIITBPC/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:2:
FutureWarning: reshape is deprecated and will raise in a subsequent release.
Please use .values.reshape(...) instead

/Users/IIITBPC/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:5:
FutureWarning: reshape is deprecated and will raise in a subsequent release.
Please use .values.reshape(...) instead
  """
```

In [19]:
```
# Doing a polynomial regression: Comparing linear, quadratic and cubic fits
# Pipeline helps you associate two models or objects to be built sequentially
 with each other,
# in this case, the objects are PolynomialFeatures() and LinearRegression()

r2_train = []
r2_test = []
degrees = [1, 2, 3]

for degree in degrees:
    pipeline = Pipeline([('poly_features', PolynomialFeatures(degree=degree)),
                         ('model', LinearRegression())])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    r2_test.append(metrics.r2_score(y_test, y_pred))

    # training performance
    y_pred_train = pipeline.predict(X_train)
    r2_train.append(metrics.r2_score(y_train, y_pred_train))

# plot predictions and actual values against year
    fig, ax = plt.subplots()
    ax.set_xlabel("Year")
    ax.set_ylabel("Power consumption")
    ax.set_title("Degree= " + str(degree))

    # train data in blue
    ax.scatter(X_train, y_train)
    ax.plot(X_train, y_pred_train)

    # test data
    ax.scatter(X_train, y_train)
    ax.plot(X_test, y_pred)

    plt.show()
```


Degree= 1
Power consumption
Year
1500000
1250000
1000000
750000
500000
250000
0
−250000
1920
1930
1940
1950
1960
1970


Degree= 2
Power consumption
Year
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
1920
1930
1940
1950
1960
1970


Degree= 3
Power consumption
Year
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
1920
1930
1940
1950
1960
1970

In [20]:
```
# respective test r-squared scores of predictions
print(degrees)
print(r2_train)
print(r2_test)
```

```
[1, 2, 3]
[0.84237474021761372, 0.99088967445535958, 0.9979789881969624]
[0.81651704638268097, 0.98760805026754717, 0.99848974839924587]
```