Task-3A:

We cannot take the data we have on a machine learning problem and apply fit () directly. First of all, we need to build a dataset. The task of transforming this raw data into a dataset is called Feature engineering. For example, a raw dataset contains details of each customer such as location, age, interests, average time spent on the product, customer's subscription, credit risk ratio. These details are actually feautures of the dataset.The purpose of building a dataset is to understand useful properties from these raw data, and also to create new properties from existing properties that have an impact on the results, or to manipulate these properties to be model-ready. This whole process is actually called Feature Engineering.

Various standard methods used in the application of feature engineering can be listed as follows:

- Encoding (One-hot encoding, label encoding)
- Binning
- Normalization
- Standardization
- Dealing with missing values
- Data Imputation techniques

Example for Label Encoding: Label Encoding allows categorical data to be digitized by assigning a unique integer value to each category. Like 0 for ‘comedy’, 1 for ‘horror’, 2 for ‘romantic ’ randomly. But assigning so may lead to giving unnecessary ordinality to the categories.

In [ ]:
```
from sklearn.preprocessing import ColumnTransformer
labelencoder = ColumnTransformer()
x[:, 0] = labelencoder.fit_transform(x[:, 0])
```

Example of Binning: Binning is n opposite situation, occurring less frequently in practice. It is used when we have a numeric property but need to convert it to a categorical property. The main motivation of binning is to make the model more robust and prevent overfitting, however, it has a cost to the performance.

In [ ]:
```
#Numerical Binning Example
Value      Bin
0-30   ->  Low
31-70  ->  Mid
71-100 ->  High

#Categorical Binning Example
Value      Bin
Germany->  Europe
Italy  ->  Europe
India  ->  Asia
Japan  ->  Asia
```

Normalization is a scaling technique such that when it is applied the features will be rescaled so that the data will fall in the range of [0,1].

For example, In the raw data, feature alcohol lies in [11,15] and, feature malic lies in [0,6]. In the normalized data, feature alcohol lies in [0,1] and, feature malic lies in [0,1].

We can see feature engineering as a process that generally increases the success of the model. We can say that the correct features obtained as a result of feature engineering can also enable simpler models to work more successfully.

Task-3B:

Naive Bayes Classification is one of the easily understandable and applicable supervised learning algorithm in cases where the target variable consists of categorical classes. This algorithm takes its name from the 18th century English mathematician Thomas BAYES and his theorem on conditional probabilities. The adjective naïve is based on the assumption that the variables affecting the classification are independent from each other.

Bayes Theorem

P (A | B) : The probability of having A when B is known to be true.

P (B | A) : The probability of having B when A is known to be true.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

P (A) : Probability of having A

P (B) : Probability of having B

As an example, we can use the weather (x) and the corresponding categorical play states (y) in the table below.

Weather	Play or not
sunny	No
cloudy	Yes
rainy	Yes
sunny	Yes
sunny	Yes
cloudy	Yes
rainy	No
rainy	No
sunny	Yes
rainy	Yes
sunny	No
cloudy	Yes
cloudy	Yes
rainy	No

frequency table		
Weather	No	Yes
cloudy	0	4
rainy	3	2
sunny	2	3
total	5	9

■ Do I play games when it's raining?

P (Yes | rainy) = P (rainy | Yes) * P (Yes) / P (rainy)

P(rainy | Yes) = 2/9 =0.22

P (Yes) = 9/14 = 0.64

P (rainy) = 5/14 = 0.36

P (Yes | rainy) = 0.39

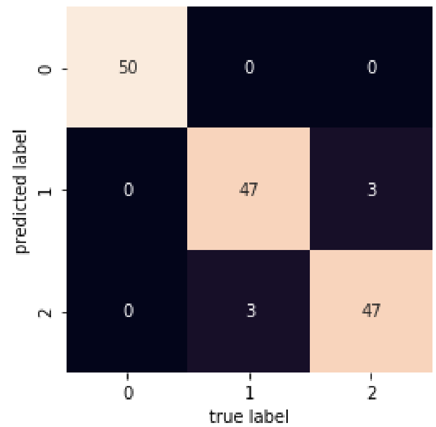
Here we used a single column feature (Weather). If we had more than one column, we would do the same for each column. In other words, every feature (X1, X2,… Xn) in Naive Bayes would be evaluated independently from each other.

$$P(y|x_1, \ldots, x_n) = \frac{P(x_1|y)P(x_2|y)\ldots P(x_n|y)P(y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

Naïve Bayes with Scikit-Learn

```
In [19]: from sklearn import datasets
         from sklearn import metrics
         from sklearn.naive_bayes import GaussianNB
         import seaborn as sns
         import matplotlib.pyplot as plt

         data_iris = datasets.load_iris()

         model = GaussianNB()
         model.fit(data_iris.data,data_iris.target)

Out[19]: GaussianNB(priors=None, var_smoothing=1e-09)

In [14]: expected = data_iris.target
         predicted = model.predict(data_iris.data)

In [23]: print(metrics.classification_report(expected,predicted))

                      precision    recall  f1-score   support

                 0       1.00      1.00      1.00        50
                 1       0.94      0.94      0.94        50
                 2       0.94      0.94      0.94        50

          accuracy                           0.96       150
         macro avg       0.96      0.96      0.96       150
      weighted avg       0.96      0.96      0.96       150
```

```
In [21]: from sklearn.metrics import confusion_matrix
         mat = confusion_matrix(expected, predicted)
         sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
         plt.xlabel('true label')
         plt.ylabel('predicted label');
```



The accuracy score value of the Gaussian naive Bayes model, that is, the success of predicting untrained data, was calculated as 96 percent. When we look at the confusion matrix, the model established correctly predicted 50 samples with a real value of 0. Again, the model correctly predicted 47 samples with a real value of 1, namely the label of 1. For an example, 3 samples was actually labeled as 1 but model predicted as 2. The 47 samples labeled as 2 are labeled as 2 in the model.

The pros and cons of the Naive Bayes algorithm:

Pros:

- Simple and easy to apply
- Can be trained and do well on small datasets
- They have very few (if any) tunable parameters
- Often very easy to interpret
- Can be used with continuous and discrete data
- Can be used in real time systems due to its speed

Cons:

- It has a ‘Zero conditional probability Problem’, for features having zero frequency the total probability also becomes zero. There are several sample correction techniques to fix this problem such as “Laplacian Correction.”
- Relationships between variables cannot be modeled because operations are performed by assuming the properties independently from each other.
- Another disadvantage is the very strong assumption of independence class features that it makes. It is near to impossible to find such data sets in real life. In real life, every feature is dependent on each other at some point.

Task-3C:

Multinomial Naive Bayes is a classification method that Whether a document/topic belongs to a particular category. The features/predictors used by the classifier are the frequency of the words present in the document. The Multinomial Naive Bayes classifier is a specific example of a Naive Bayes classifier that uses a multinomial distribution for each of the features.

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification).

Text classification aims to assign documents into one or many categories and one kind of the most useful text classification is Sentiment analysis. Sentiment analysis is objective to determine the writer's point of view about a particular topic, product, or service.

It is mostly used in natural language processing (NLP) problems, sentiment analysis etc. Naive Bayes predict the tag of a text. They calculate the probability of each tag for a given text and then output the tag with the highest one.

Example:

In [26]:
```
import pandas as pd
data = pd.read_json('News_Category_Dataset_v2.json', lines=True)
data.head()
```

Out[26]:

	category	headline	authors	link	short_description	date
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	Melissa Jeltsen	https://www.huffingtonpost.com/entry/texas-ama...	She left her husband. He killed their children...	2018-05-26
1	ENTERTAINMENT	Will Smith Joins Diplo And Nicky Jam For The 2...	Andy McDonald	https://www.huffingtonpost.com/entry/will-smit...	Of course it has a song.	2018-05-26
2	ENTERTAINMENT	Hugh Grant Marries For The First Time At Age 57	Ron Dicker	https://www.huffingtonpost.com/entry/hugh-gran...	The actor and his longtime girlfriend Anna Ebe...	2018-05-26
3	ENTERTAINMENT	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	Ron Dicker	https://www.huffingtonpost.com/entry/jim-carre...	The actor gives Dems an ass-kicking for not fi...	2018-05-26
4	ENTERTAINMENT	Julianna Margulies Uses Donald Trump Poop Bags...	Ron Dicker	https://www.huffingtonpost.com/entry/julianna-...	The "Dietland" actress said using the bags is ...	2018-05-26

In [33]:
```
mapper = {}

for i,cat in enumerate(data["category"].unique()):
        mapper[cat] = i

data["category_target"] = data["category"].map(mapper)
data[["category", "headline", "category_target"]].head()
```

Out[33]:

	category	headline	category_target
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	0
1	ENTERTAINMENT	Will Smith Joins Diplo And Nicky Jam For The 2...	1
2	ENTERTAINMENT	Hugh Grant Marries For The First Time At Age 57	1
3	ENTERTAINMENT	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	1
4	ENTERTAINMENT	Julianna Margulies Uses Donald Trump Poop Bags...	1

In [34]:
```
#we need to calculate the count of each word.
from sklearn.feature_extraction.text import CountVectorizer

text=["My name is Paul my life is Jane! And we live our life together" , "My name is Guido my life is Victoria! And we live our l
toy = CountVectorizer(stop_words = 'english')
toy.fit_transform(text)
print (toy.vocabulary_)

matrix = toy.transform(text)
print (matrix)
features = toy.get_feature_names()
df_res = pd.DataFrame(matrix.toarray(), columns=features)
df_res
```

```
{'paul': 4, 'life': 2, 'jane': 1, 'live': 3, 'guido': 0, 'victoria': 5}
  (0, 1)	1
  (0, 2)	2
  (0, 3)	1
  (0, 4)	1
  (1, 0)	1
  (1, 2)	2
  (1, 3)	1
  (1, 5)	1
```

Out[34]:

	guido	jane	life	live	paul	victoria
0	0	1	2	1	1	0
1	1	0	2	1	0	1

In [35]:
```
vect = CountVectorizer(stop_words = 'english')
X_train_matrix = vect.fit_transform(data["headline"])
```

In [36]:
```
print (X_train_matrix.shape)
```

```
(200853, 55356)
```

In [45]:
```
X_train_matrix
```

Out[45]:
```
<200853x55356 sparse matrix of type '<class 'numpy.int64'>'
	with 1269212 stored elements in Compressed Sparse Row format>
```

In [37]:
```
column = vect.vocabulary_["hollywood"]
print (column)
vect.get_feature_names()[column]
```

```
23211
```

Out[37]:
```
'hollywood'
```

Build the classifier in Scikit Learn

In [46]:
```
y = data["category_target"]  #target feature
```

In [47]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train_matrix, y, test_size=0.3)
```

In [49]:
```
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
model.fit(X_train, y_train)
print (model.score(X_train, y_train))
print (model.score(X_test, y_test))
```

```
0.6093942260503424
0.5225703664365374
```

Build the classifier in Scikit Learn

In [46]:
```
y = data["category_target"]  #target feature
```

In [47]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train_matrix, y, test_size=0.3)
```

In [53]:
```
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
model.fit(X_train, y_train)
print (model.score(X_train, y_train))
print (model.score(X_test, y_test))
predicted_result=model.predict(X_test)
```

```
0.6093942260503424
0.5225703664365374
```

Probability of each class

In [51]:
```
pi = {}
All = data["category_target"].value_counts().sum()
```

In [52]:
```
for i, cat in enumerate (data["category_target"].value_counts(sort = False)):
    pi[i] = cat / All

print("Probability of each class:")
print("\n".join("{}: {}".format(k, v) for k, v in pi.items()))
```

```
Probability of each class:
0: 0.01695269674836821
1: 0.07994901744061578
2: 0.010838772634712949
3: 0.017221550088870965
4: 0.16299980582814297
5: 0.013293304058191811
6: 0.022543850477712554
7: 0.017375891821381807
8: 0.025765111798180758
9: 0.031435925776562956
10: 0.0243162910188048
11: 0.029558931158608536
12: 0.04922505513982863
13: 0.014015225065097359
14: 0.010365789906050693
15: 0.012725724783797106
16: 0.010843751400277815
17: 0.005621026322733542
18: 0.00499868062712531
19: 0.005695707806206529
20: 0.019691017809044427
21: 0.006666567091355369
22: 0.01122213758320762
23: 0.013054323311078251
24: 0.010435492623958816
25: 0.03332785669121198
26: 0.018242197029668464
27: 0.006960314259682454
```

There are 40 different category classes.

Task-3D:

```
In [59]: data = pd.read_excel('salary_data.xlsx')
         data.head()
```

Out[59]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

```
In [75]: import pandas as pd

         X = data.iloc[:, 1:-1].values
         y = data.iloc[:, -1].values
```

```
In [76]: X
```

```
Out[76]: array([[ 1],
                [ 2],
                [ 3],
                [ 4],
                [ 5],
                [ 6],
                [ 7],
                [ 8],
                [ 9],
                [10]], dtype=int64)
```

```
In [78]: y
```

```
Out[78]: array([  45000,   50000,   60000,   80000,  110000,  150000,  200000,
                 300000,  500000, 1000000], dtype=int64)
```

```
In [66]: y
```

```
Out[66]: array([  45000,   50000,   60000,   80000,  110000,  150000,  200000,
                 300000,  500000, 1000000], dtype=int64)
```

The Level is taken as the independent variable and is assigned to X. The dependent variable that is to be predicted is the last column (-1) which is Salary and it is assigned to y.

4^{th} -Polynomial Regression Model

```
In [97]: from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression
         poly_reg = PolynomialFeatures(degree = 4)  #4th degree
         X_poly = poly_reg.fit_transform(X)
         lin_reg = LinearRegression()
         lin_reg.fit(X_poly, y)
```

```
Out[97]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

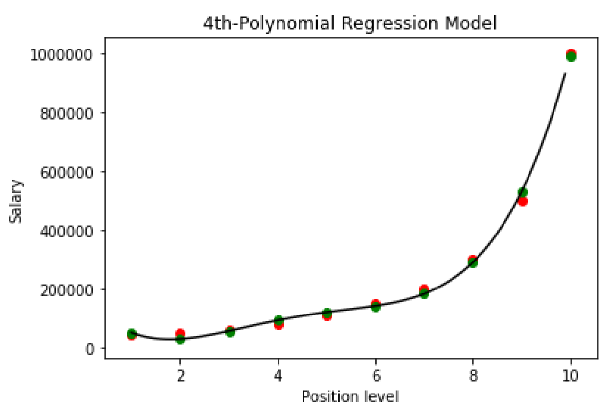
```
In [98]: y_pred = lin_reg.predict(X_poly)
```

In [99]:
```
df = pd.DataFrame({'True Values':y, 'Predicted Values':y_pred})
df
```

Out[99]:

	True Values	Predicted Values
0	45000	53356.643357
1	50000	31759.906760
2	60000	58642.191142
3	80000	94632.867133
4	110000	121724.941725
5	150000	143275.058275
6	200000	184003.496503
7	300000	289994.172494
8	500000	528694.638695
9	1000000	988916.083916

In [100]:
```
import numpy as np
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter(X, y_pred, color = 'green')
plt.plot(X_grid, lin_reg.predict(poly_reg.fit_transform(X_grid)), color = 'black')
plt.title('4th-Polynomial Regression Model')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

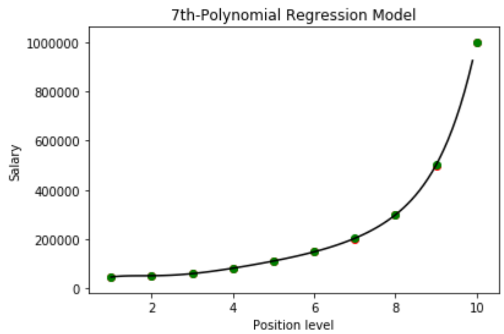


In this graph, the Real values are plotted in “Red” color and the Predicted values are plotted in “Green” color. The Polynomial Regression line that is generated is drawn in “Black” color.

7th -Polynomial Regression Model

In [103]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 7)  #7th degree
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
y_pred = lin_reg.predict(X_poly)
```

In [104]:
```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter(X, y_pred, color = 'green')
plt.plot(X_grid, lin_reg.predict(poly_reg.fit_transform(X_grid)), color = 'black')
plt.title('7th-Polynomial Regression Model')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



10th –Polynomial Regression Model

In [105]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 10)  #7th degree
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
y_pred = lin_reg.predict(X_poly)
```

In [107]:
```
df = pd.DataFrame({'True Values':y, 'Predicted Values':y_pred})
df
```

Out[107]:

	True Values	Predicted Values
0	45000	4.500000e+04
1	50000	5.000000e+04
2	60000	6.000000e+04
3	80000	8.000000e+04
4	110000	1.100000e+05
5	150000	1.500000e+05
6	200000	2.000000e+05
7	300000	3.000000e+05
8	500000	5.000000e+05
9	1000000	1.000000e+06

In [106]:

```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter(X, y_pred, color = 'green')
plt.plot(X_grid, lin_reg.predict(poly_reg.fit_transform(X_grid)), color = 'black')
plt.title('10th-Polynomial Regression Model')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```