Scikit-learn

(Code: Subhajit Das)

What is Scikit-Learn:

1. **Machine Learning in Python**: Scikit-learn is a library in Python that provides simple and efficient tools for predictive data analysis.
2. **Open Source**: It is open source and commercially usable under the BSD license.
3. **Built on NumPy, SciPy, and matplotlib**: Scikit-learn is built on top of these three Python libraries, which makes it a robust tool for machine learning.
4. **Various Algorithms**: It features various classification, regression, and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
5. **Model Selection**: It provides tools for comparing, validating, and choosing parameters and models.
6. **Preprocessing**: Scikit-learn also provides tools for feature extraction and normalization.
7. **Unified Interface**: All the algorithms in scikit-learn share a uniform and limited API consisting of complementary interfaces.
8. **Installation**: The latest version of Scikit-learn is 1.3.1 and it requires Python 3.8 or newer[2]. It can be installed using pip: `pip install -U scikit-learn` .

Where we can use Scikit-Learn:

1. **Supervised Learning Algorithms**: These include Linear Regression, Support Vector Machines (SVM), Decision Trees, Random Forests, Gradient Boosting, K-Nearest Neighbors, and many more.
2. **Unsupervised Learning Algorithms**: These include clustering algorithms like K-Means, Hierarchical Clustering, DBSCAN, etc. It also includes dimensionality reduction techniques like Principal Component Analysis (PCA), Non-negative Matrix Factorization (NMF), and Independent Component Analysis (ICA).
3. **Model Selection and Evaluation**: Scikit-learn provides tools for model selection (like GridSearchCV and RandomizedSearchCV) and evaluation metrics for regression, classification, and clustering tasks.
4. **Preprocessing**: Scikit-learn also provides utilities for preprocessing data, feature extraction, and feature selection.

Here's a brief description of the scikit packages:

1. **Scikit-learn**: This is a library in Python that provides simple and efficient tools for predictive data analysis. It is built on top of NumPy, SciPy, and matplotlib and features various classification, regression, and clustering algorithms.
2. **Scikit-metrics**: Scikit-learn provides various metrics for evaluating the quality of a model's predictions, such as precision_score, recall_score, f1_score, etc. However, there's also a package named 'scikit-metrics' listed on PyPI, but it doesn't provide a project description.
3. **Scikit-meta**: The term 'meta' in the context of scikit-learn often refers to meta-estimators or methods that combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability/robustness over a single estimator. There's also a concept of 'Super Learner' which is an application of stacked generalization using out-of-fold predictions during k-fold cross-validation.

4. **Scikit-processing**: This term doesn't seem to correspond to a specific package. However, scikit-learn does provide several common utility functions and transformer classes for preprocessing data. There's also 'scikit-image' for image processing, and 'scikit-video' for video processing.

Install Scikit-Learn

In [1]:
```
# pip install scikit-learn
```

Import Scikit-Learn

In [2]:
```
import sklearn as sk
sk.__version__
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Linking the dataset from Scikit-learn

In [3]:
```
from sklearn.datasets import load_diabetes
load_diabetes()
```

Out[3]:
```
{'data': array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
          0.01990749, -0.01764613],
        [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
         -0.06833155, -0.09220405],
        [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
          0.00286131, -0.02593034],
        ...,
        [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
         -0.04688253,  0.01549073],
        [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
          0.04452873, -0.02593034],
        [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
         -0.00422151,  0.00306441]]),
 'target': array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 31
0., 101.,
         69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
         68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
         87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
        259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
        128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
        150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
        200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
         42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
         83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
        104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
        173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
        107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
         60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
        197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
         59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
        237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
        143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
        142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
         77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
         78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
        154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
         71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
        150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
        145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
         94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
         60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
         31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
        114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
        191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
        244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
        263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
         77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
         58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
        140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
        219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
         43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
        140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
         84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
         94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
        220.,  57.]),
 'frame': None,
 'DESCR': '.. _diabetes_dataset:\n\nDiabetes dataset\n----------------\n\n
Ten baseline variables, age, sex, body mass index, average blood\npressur
e, and six blood serum measurements were obtained for each of n =\n442 dia
betes patients, as well as the response of interest, a\nquantitative measu
re of disease progression one year after baseline.\n\n**Data Set Character
istics:**\n\n  :Number of Instances: 442\n\n  :Number of Attributes: First
```

```
10 columns are numeric predictive values\n\n  :Target: Column 11 is a quan
titative measure of disease progression one year after baseline\n\n  :Attr
ibute Information:\n      - age     age in years\n      - sex\n      - bmi
body mass index\n      - bp      average blood pressure\n      - s1      t
c, total serum cholesterol\n      - s2      ldl, low-density lipoproteins
\n      - s3      hdl, high-density lipoproteins\n      - s4      tch, tot
al cholesterol / HDL\n      - s5      ltg, possibly log of serum triglycer
ides level\n      - s6      glu, blood sugar level\n\nNote: Each of these
10 feature variables have been mean centered and scaled by the standard de
viation times the square root of `n_samples` (i.e. the sum of squares of e
ach column totals 1).\n\nSource URL:\nhttps://www4.stat.ncsu.edu/~boos/va
r.select/diabetes.html\n\nFor more information see:\nBradley Efron, Trevor
Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regressio
n," Annals of Statistics (with discussion), 407-499.\n(https://web.stanfor
d.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)\n',
 'feature_names': ['age',
  'sex',
  'bmi',
  'bp',
  's1',
  's2',
  's3',
  's4',
  's5',
  's6'],
 'data_filename': 'diabetes_data_raw.csv.gz',
 'target_filename': 'diabetes_target.csv.gz',
 'data_module': 'sklearn.datasets.data'}
```

In [4]:
```
print(load_diabetes()['DESCR'])
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one
year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scale
d by the standard deviation times the square root of `n_samples` (i.e. the
sum of squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html (https://www4.st
at.ncsu.edu/~boos/var.select/diabetes.html)

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004)
"Least Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

In [5]:
```python
# If return_X_y is True, then ( data , target ) will be pandas DataFrames or
load_diabetes(return_X_y = True)
```

Out[5]:
```
(array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
          0.01990749, -0.01764613],
        [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
         -0.06833155, -0.09220405],
        [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
          0.00286131, -0.02593034],
        ...,
        [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
         -0.04688253,  0.01549073],
        [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
          0.04452873, -0.02593034],
        [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
         -0.00422151,  0.00306441]]),
 array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
         69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
         68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
         87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
        259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
        128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
        150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
        200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
         42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
         83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
        104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
        173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
        107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
         60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
        197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
         59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
        237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
        143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
        142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
         77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
         78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
        154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
         71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
        150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
        145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
         94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
         60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
         31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
        114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
        191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
        244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
        263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
         77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
         58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
        140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
        219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
         43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
        140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
         84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
         94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
        220.,  57.]))
```

In [6]:
```python
x, y = load_diabetes(return_X_y = True)
```

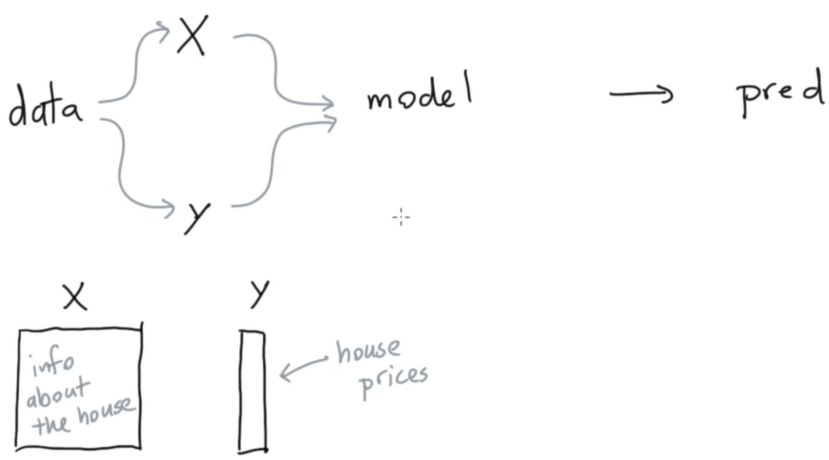
Convert this dataset into dataframe

In [7]:
```python
# Load the dataset
diabetes = load_diabetes()

# Convert it into a DataFrame
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)

# Drop the column. For example, let's drop 'age'
df = df.drop('age', axis=1)

df # Age column not printed
```

Out[7]:

	sex	bmi	bp	s1	s2	s3	s4	s5	
0	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.
1	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.
2	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.
3	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.
4	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.
...	...	...	...	...	...	...	...	...	
437	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.
438	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018114	0.
439	0.050680	-0.015906	0.017293	-0.037344	-0.013840	-0.024993	-0.011080	-0.046883	0.
440	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.044529	-0.
441	-0.044642	-0.073030	-0.081413	0.083740	0.027809	0.173816	-0.039493	-0.004222	0.

442 rows × 9 columns



K-Nearest Neighbour

In [8]:
```
from sklearn.neighbors import KNeighborsRegressor  # KNeighborsClassificatic
```

In [9]:
```
model = KNeighborsRegressor()
```

In [10]:
```
model.fit(x, y) # KNeighbor will learn from data as much as possible.
```

Out[10]:
```
KNeighborsRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [11]:
```
model.predict(x)
```

Out[11]:
```
array([181.4,  80.8, 150.8, 203.4, 119.4, 108. ,  83.6, 120.6, 127.8,
       187.8, 121.4, 129.6,  98.8, 166.8, 106.6, 144. , 174.4, 177.8,
       132.8, 142. ,  80. ,  73.4, 113.4, 273.8, 151.4, 126.4, 132. ,
       137.8, 107.2, 192.2, 154.6,  65. , 287.8,  78.6,  78.6, 109.6,
       175.2, 172.6, 235. ,  76.4, 158.6, 113.4,  97.6,  74. , 257. ,
        92.8, 163. , 150.2, 110.8, 145.4, 121.2, 152.4, 130.6,  94. ,
       157.2,  88.8, 137.2, 101. , 132.2, 156.6, 115. , 124. ,  65.8,
       132.4, 128.4, 154.4, 119.6,  88.8,  96.2, 136.6,  81.6, 234.6,
       174.4, 109.8, 133.2,  72. , 171.8, 106.6, 179.4, 121.6, 134.4,
       107.8,  69.2, 141.2,  80. , 108.6,  92.8, 124.6,  68.6, 110.6,
        95. , 136.4, 153.4,  82.8,  83.8, 126.8, 170.6, 165.8,  88.6,
       126.2, 158.8,  86.2, 151.8, 155.4, 139.4,  87. , 115. , 143.2,
       181. , 183. ,  71.2,  99.4, 134.8, 202. , 277.4, 165.4, 255.4,
       207. , 126.6, 132.2, 130. , 202. , 232.4, 158.2, 136.4, 166.4,
       111.6,  92.8, 112.2, 204.4, 255.4, 105.4, 106. ,  77. , 155.2,
       250.8,  85.6, 261.2, 265.2, 250.2, 163.6, 272.4, 207.4,  99.6,
       192.6, 220.4, 142.8, 235.4,  89. , 143. , 223.6, 121.6, 221.2,
       131.4, 180.4, 233. , 122.6, 122.4, 100.2, 230.6,  85.2, 227.6,
       129.6, 248.2, 135.6,  88.2,  73.4, 211.4, 261.6, 189.2,  71.2,
        64.2, 278.8,  93. , 119.2, 103. , 221.8, 199.4, 106.8, 185.6,
       112.2,  75.8, 193.8, 174.8, 237.2, 125.8, 151.4,  98. , 165. ,
       114. , 193.2, 123.4,  94.6, 158.4, 130.2, 192.6,  74.2, 169. ,
       166.4, 184.6,  96.8,  71.4, 166.6, 211.8, 201.8, 192.4, 173.2,
       232. , 234.8, 135.2,  97. , 148.8, 134.6,  92.8,  89.6, 258.8,
       230.8, 241.2, 169.4, 142.2,  82.8, 136.4, 134. , 146.8,  62.8,
       181. ,  84.6, 110.2, 146.8,  65. , 193.8, 126. , 174.6,  98.2,
       246. , 144.6, 133.8,  81.8, 143.6, 139. , 244.6, 136.6,  65.4,
        87.6,  89.6, 128.8, 121.4, 102. , 171.4, 179. , 271.6, 239.6,
       142.6, 168. , 278. , 117. , 220.4,  96. , 122.4, 101.8,  79.4,
       124.4, 274.8,  84.8, 105. ,  79.6,  62.2, 156.8, 265.2, 116.4,
       179. , 156. , 148.8, 203.6, 168.6, 116.6, 170.6,  82.8, 108.2,
        96.2, 220. ,  93. , 156.2,  87. , 126.4, 215. ,  69.4, 139. ,
        83.2, 150.8, 265.2, 198. ,  84. , 173.2,  74. ,  97.8,  83.2,
        78.8, 145.6, 105. , 212. , 133.2, 195.2, 223.6, 131.2, 186.4,
       136.6, 156.8,  85. , 141. , 226.6, 167.8, 101.2, 209.8, 142.4,
        76.2, 153.6, 152.8, 137.4, 201. , 146.6, 245.6, 264.4, 247.6,
       156.8, 201.8, 110.8, 207.2, 111.4,  77.8, 170.6, 128.4, 266.4,
       173. ,  81.4, 106.2, 237. , 127.8, 131.8, 135.6, 169. , 167.4,
       167.6, 180.2, 185. , 139.2, 173.2,  97.6, 140.2,  97.8, 273.4,
        68. ,  74.6, 130.4, 198.2,  97.8,  63.4, 139.8,  91.6, 204.6,
       192.8, 111.2, 265.2,  94.2, 171.8, 211. , 192.8, 228.2, 235.8,
       167.4,  88.2, 197.2,  69.6, 181.4, 161.4, 221.2, 193.8, 188.6,
       167.6, 101.2, 181.8,  94. , 228.4, 110. , 157.4, 156.2, 120.2,
       124.4, 131.4,  67.4, 260.4,  88.2, 108.2, 108. , 242.2, 172.6,
        61.2, 189. , 163. , 217. , 169. ,  75. , 132.4, 208.4, 184.2,
       238.6, 110.8, 164.2, 202.2, 183.2, 169. , 138.8, 239. , 120.6,
       144. , 166.4, 241.8, 103.8, 141.6,  83.8, 124.4, 191.8, 207.8,
       179.6, 119.6, 108. , 189.6, 130.8, 295.2,  76.4, 154.4, 132.8,
       213.6,  86.6, 100.8, 102. ,  68.2, 156. , 134. , 100.8, 176. ,
        80.4])
```

In [12]:
```
pred_knn = model.predict(x) # To predict the model
```

Linear Regression

In [13]:
```python
from sklearn.linear_model import LinearRegression
```

In [14]:
```python
model = LinearRegression()
```

In [15]:
```python
model.fit(x, y)
```

Out[15]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:
```python
model.predict(x)
```

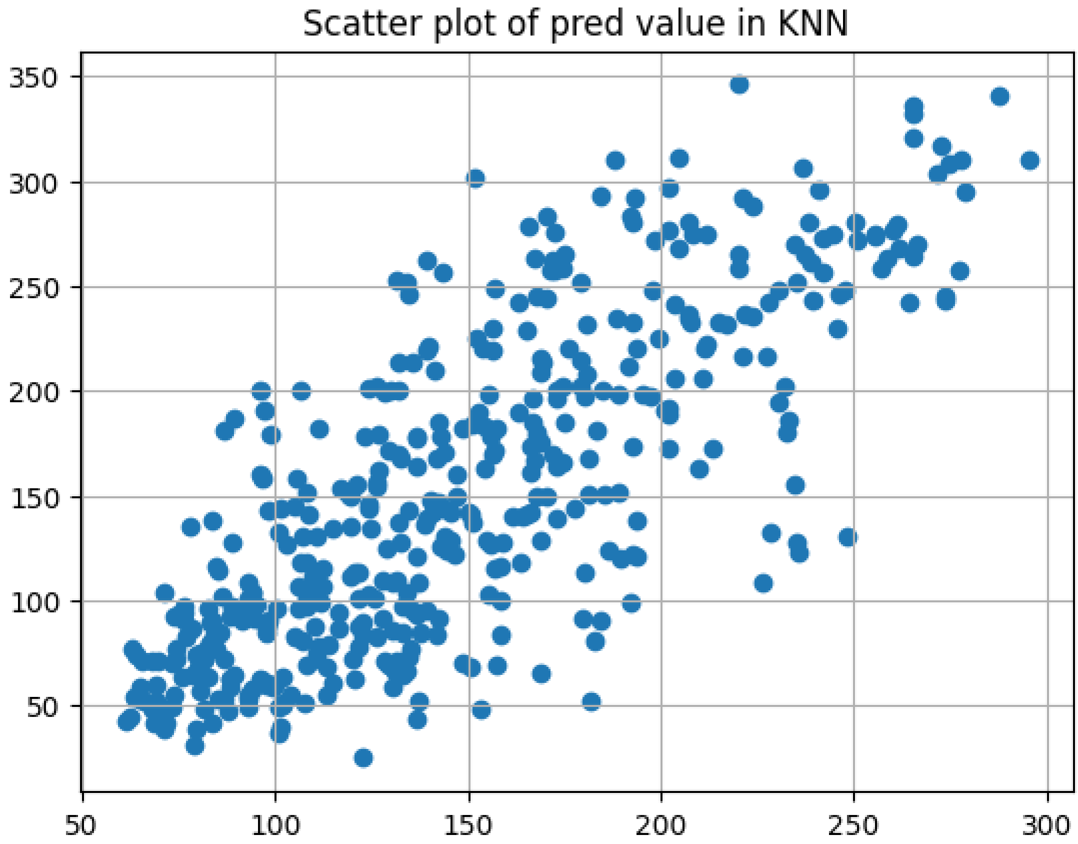
Out[16]:
```
array([206.11667725,  68.07103297, 176.88279035, 166.91445843,
       128.46225834, 106.35191443,  73.89134662, 118.85423042,
       158.80889721, 213.58462442,  97.07481511,  95.10108423,
       115.06915952, 164.67656842, 103.07814257, 177.17487964,
       211.7570922 , 182.84134823, 148.00326937, 124.01754066,
       120.33362197,  85.80068961, 113.1134589 , 252.45225837,
       165.48779206, 147.71997564,  97.12871541, 179.09358468,
       129.05345958, 184.7811403 , 158.71516713,  69.47575778,
       261.50385365, 112.82234716,  78.37318279,  87.66360785,
       207.92114668, 157.87641942, 240.84708073, 136.93257456,
       153.48044608,  74.15426666, 145.62742227,  77.82978811,
       221.07832768, 125.21957584, 142.6029986 , 109.49562511,
        73.14181818, 189.87117754, 157.9350104 , 169.55699526,
       134.1851441 , 157.72539008, 139.11104979,  72.73116856,
       207.82676612,  80.11171342, 104.08335958, 134.57871054,
       114.23552012, 180.67628279,  61.12935368,  98.72404613,
       113.79577026, 189.95771575, 148.98351571, 124.34152283,
       114.8395504 , 121.99957578,  73.91017087, 236.71054289,
       142.31126791, 124.51672384, 150.84073896, 127.75230658,
```

In [17]:
```python
pred_linear = model.predict(x)
```

K-Nearest Neighbour vs Lineear Regression

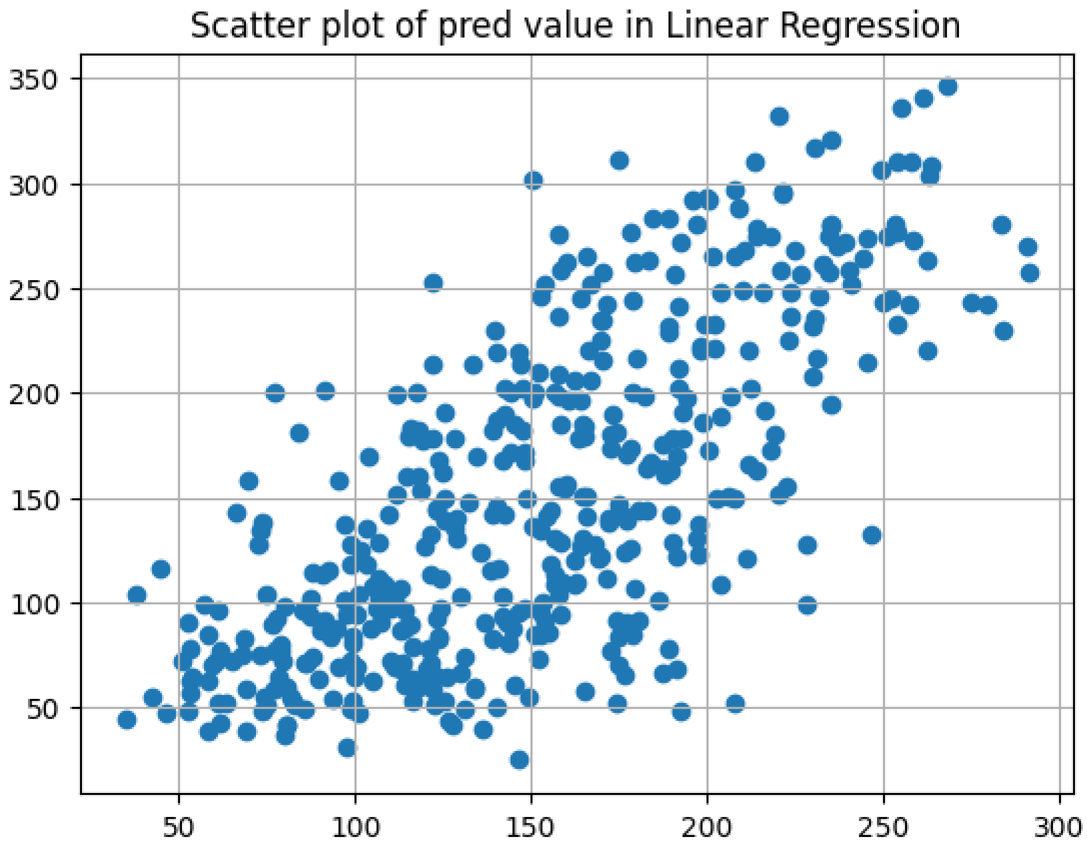
In [18]:
```
plt.scatter(pred_knn, y)
plt.grid()
plt.title('Scatter plot of pred value in KNN')
```

Out[18]: Text(0.5, 1.0, 'Scatter plot of pred value in KNN')



In [19]:
```
plt.scatter(pred_linear, y)
plt.grid()
plt.title('Scatter plot of pred value in Linear Regression')
```

Out[19]: Text(0.5, 1.0, 'Scatter plot of pred value in Linear Regression')



Standard Scaling and Pipeline in K-Nearest Neighbour

In [20]:
```
from sklearn.preprocessing import StandardScaler
''' The StandardScaler is a popular data pre-processing technique that is us
 scaling each feature to unit variance. This makes it easier for machine lea
```

Out[20]: ' The StandardScaler is a popular data pre-processing technique that is us
ed to standardize the distribution of features in a dataset. This is done
by removing the mean and\n scaling each feature to unit variance. This mak
es it easier for machine learning algorithms to learn and generalize bette
r.'

In [21]:
```
from sklearn.pipeline import Pipeline
''' The pipeline class has fit, predict, and score methods just like any oth
```

Out[21]: ' The pipeline class has fit, predict, and score methods just like any oth
er estimator. To implement pipeline, you first separate features and label
s from the data-set.'

In [22]:
```
mod_pipe = KNeighborsRegressor().fit(x, y)

pipe = Pipeline([
    ('Scale', StandardScaler()),
    ('model', KNeighborsRegressor())
])
```

In [23]:
```
pipe.fit(x,y)
```

Out[23]:
```
Pipeline(steps=[('Scale', StandardScaler()), ('model', KNeighborsRegressor
())])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

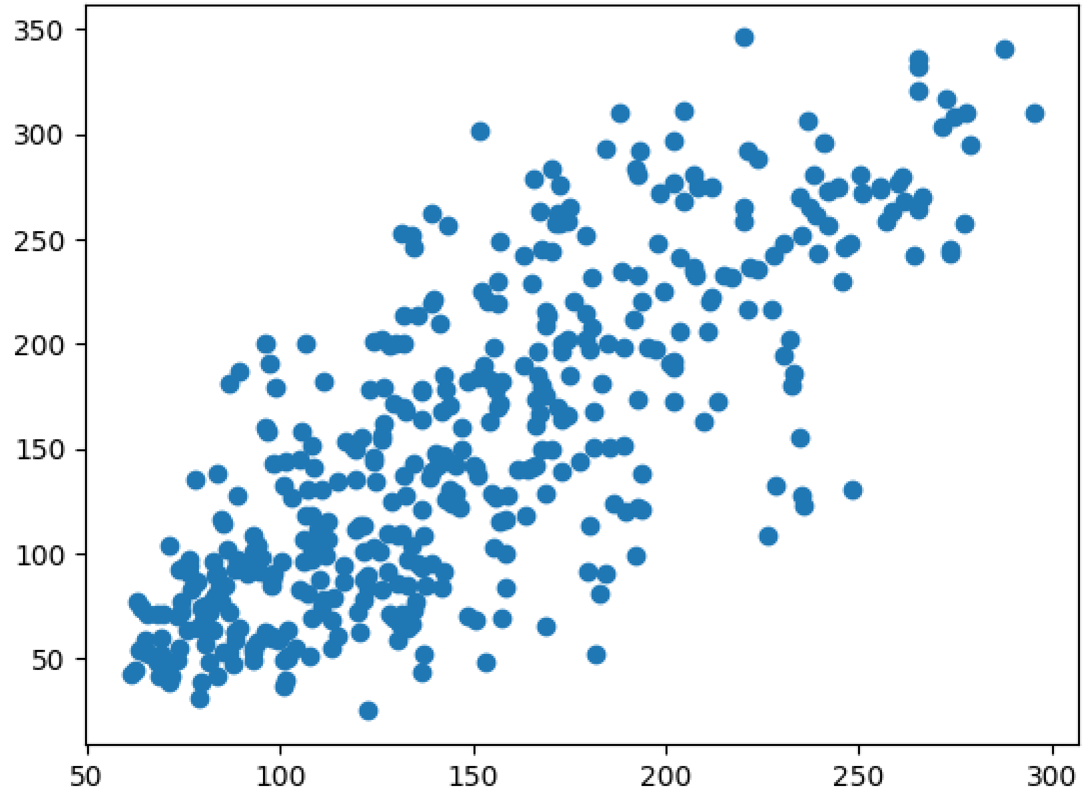
In [24]:
```
pipe.predict(x)
```

Out[24]:
```
array([181.4,  80.8, 150.8, 203.4, 119.4, 108. ,  83.6, 120.6, 127.8,
       187.8, 121.4, 129.6,  98.8, 166.8, 106.6, 144. , 174.4, 177.8,
       132.8, 142. ,  80. ,  73.4, 113.4, 273.8, 151.4, 126.4, 132. ,
       137.8, 107.2, 192.2, 154.6,  65. , 287.8,  78.6,  78.6, 109.6,
       175.2, 172.6, 235. ,  76.4, 158.6, 113.4,  97.6,  74. , 257. ,
        92.8, 163. , 150.2, 110.8, 145.4, 121.2, 152.4, 130.6,  94. ,
       157.2,  88.8, 137.2, 101. , 132.2, 156.6, 115. , 124. ,  65.8,
       132.4, 128.4, 154.4, 119.6,  88.8,  96.2, 136.6,  81.6, 234.6,
       174.4, 109.8, 133.2,  72. , 171.8, 106.6, 179.4, 121.6, 134.4,
       107.8,  69.2, 141.2,  80. , 108.6,  92.8, 124.6,  68.6, 110.6,
        95. , 136.4, 153.4,  82.8,  83.8, 126.8, 170.6, 165.8,  88.6,
       126.2, 158.8,  86.2, 151.8, 155.4, 139.4,  87. , 115. , 143.2,
       181. , 183. ,  71.2,  99.4, 134.8, 202. , 277.4, 165.4, 255.4,
       207. , 126.6, 132.2, 130. , 202. , 232.4, 158.2, 136.4, 166.4,
       111.6,  92.8, 112.2, 204.4, 255.4, 105.4, 106. ,  77. , 155.2,
       250.8,  85.6, 261.2, 265.2, 250.2, 163.6, 272.4, 207.4,  99.6,
       192.6, 220.4, 142.8, 235.4,  89. , 143. , 223.6, 121.6, 221.2,
       131.4, 180.4, 233. , 122.6, 122.4, 100.2, 230.6,  85.2, 227.6,
       129.6, 248.2, 135.6,  88.2,  73.4, 211.4, 261.6, 189.2,  71.2,
        64.2, 278.8,  93. , 119.2, 103. , 221.8, 199.4, 106.8, 185.6,
       112.2,  75.8, 193.8, 174.8, 237.2, 125.8, 151.4,  98. , 165. ,
       114. , 193.2, 123.4,  94.6, 158.4, 130.2, 192.6,  74.2, 169. ,
       166.4, 184.6,  96.8,  71.4, 166.6, 211.8, 201.8, 192.4, 173.2,
       232. , 234.8, 135.2,  97. , 148.8, 134.6,  92.8,  89.6, 258.8,
       230.8, 241.2, 169.4, 142.2,  82.8, 136.4, 134. , 146.8,  62.8,
       181. ,  84.6, 110.2, 146.8,  65. , 193.8, 126. , 174.6,  98.2,
       246. , 144.6, 133.8,  81.8, 143.6, 139. , 244.6, 136.6,  65.4,
        87.6,  89.6, 128.8, 121.4, 102. , 171.4, 179. , 271.6, 239.6,
       142.6, 168. , 278. , 117. , 220.4,  96. , 122.4, 101.8,  79.4,
       124.4, 274.8,  84.8, 105. ,  79.6,  62.2, 156.8, 265.2, 116.4,
       179. , 156. , 148.8, 203.6, 168.6, 116.6, 170.6,  82.8, 108.2,
        96.2, 220. ,  93. , 156.2,  87. , 126.4, 215. ,  69.4, 139. ,
        83.2, 150.8, 265.2, 198. ,  84. , 173.2,  74. ,  97.8,  83.2,
        78.8, 145.6, 105. , 212. , 133.2, 195.2, 223.6, 131.2, 186.4,
       136.6, 156.8,  85. , 141. , 226.6, 167.8, 101.2, 209.8, 142.4,
        76.2, 153.6, 152.8, 137.4, 201. , 146.6, 245.6, 264.4, 247.6,
       156.8, 201.8, 110.8, 207.2, 111.4,  77.8, 170.6, 128.4, 266.4,
       173. ,  81.4, 106.2, 237. , 127.8, 131.8, 135.6, 169. , 167.4,
       167.6, 180.2, 185. , 139.2, 173.2,  97.6, 140.2,  97.8, 273.4,
        68. ,  74.6, 130.4, 198.2,  97.8,  63.4, 139.8,  91.6, 204.6,
       192.8, 111.2, 265.2,  94.2, 171.8, 211. , 192.8, 228.2, 235.8,
       167.4,  88.2, 197.2,  69.6, 181.4, 161.4, 221.2, 193.8, 188.6,
       167.6, 101.2, 181.8,  94. , 228.4, 110. , 157.4, 156.2, 120.2,
       124.4, 131.4,  67.4, 260.4,  88.2, 108.2, 108. , 242.2, 172.6,
        61.2, 189. , 163. , 217. , 169. ,  75. , 132.4, 208.4, 184.2,
       238.6, 110.8, 164.2, 202.2, 183.2, 169. , 138.8, 239. , 120.6,
       144. , 166.4, 241.8, 103.8, 141.6,  83.8, 124.4, 191.8, 207.8,
       179.6, 119.6, 108. , 189.6, 130.8, 295.2,  76.4, 154.4, 132.8,
       213.6,  86.6, 100.8, 102. ,  68.2, 156. , 134. , 100.8, 176. ,
        80.4])
```

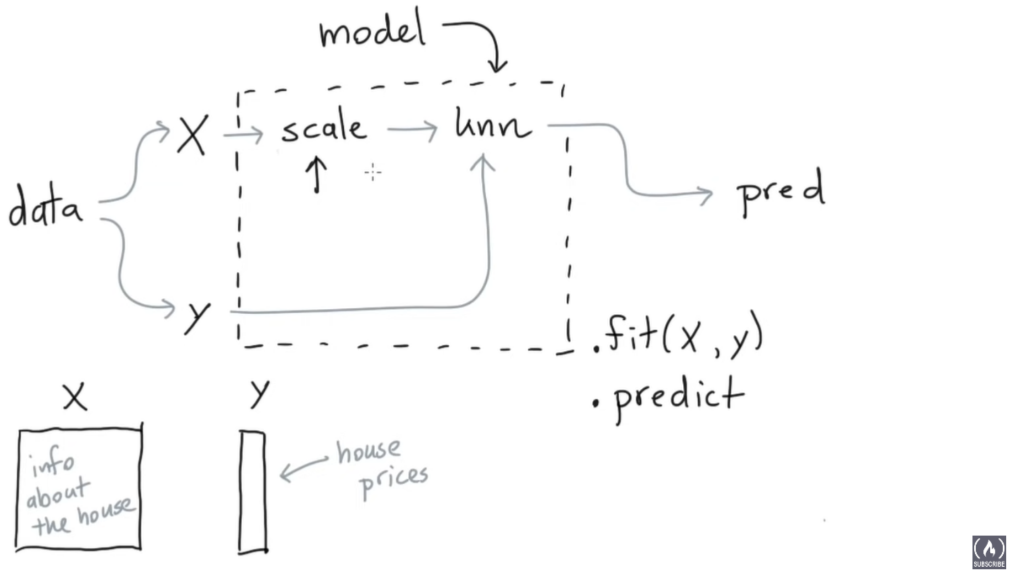
In [25]:
```
pred = pipe.predict(x)
```

In [26]:
```
plt.scatter(pred, y)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x7cf5d84150f0>
```



K-Nearest Neighbour using n_neighbours in parameter



In [27]:
```
mod_pipe = KNeighborsRegressor().fit(x, y)

pipe = Pipeline([
    ('Scale', StandardScaler()),
    ('model', KNeighborsRegressor(n_neighbors = 1)) # The parameter 'n_neigh
])
```

In [28]:
```
pipe.fit(x,y)
```

Out[28]:
```
Pipeline(steps=[('Scale', StandardScaler()),
                ('model', KNeighborsRegressor(n_neighbors=1))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

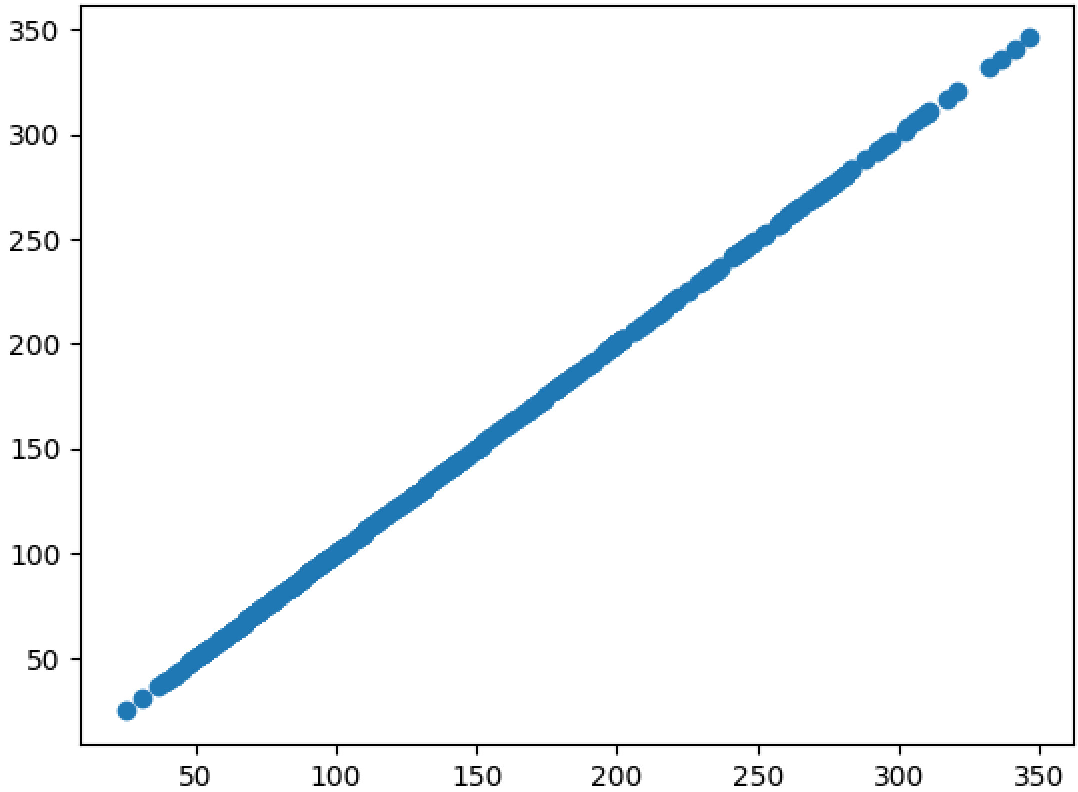
In [29]:
```
pipe.predict(x)
```

Out[29]:
```
array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
        68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
        87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
       259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
       128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
       150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
       200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
        42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
        83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
       104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
       173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
       107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
        60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
       197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
        59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
       237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
       143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
       142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
        77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
        78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
       154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
        71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
       150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
       145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
        94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
        60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
        31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
       114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
       191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
       244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
       263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
        77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
        58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
       140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
       219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
        43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
       140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
        84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
        94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
       220.,  57.])
```

In [30]:
```
pred = pipe.predict(x)
```

```
In [31]: plt.scatter(pred, y)
```

```
Out[31]: <matplotlib.collections.PathCollection at 0x7cf5d848b550>
```

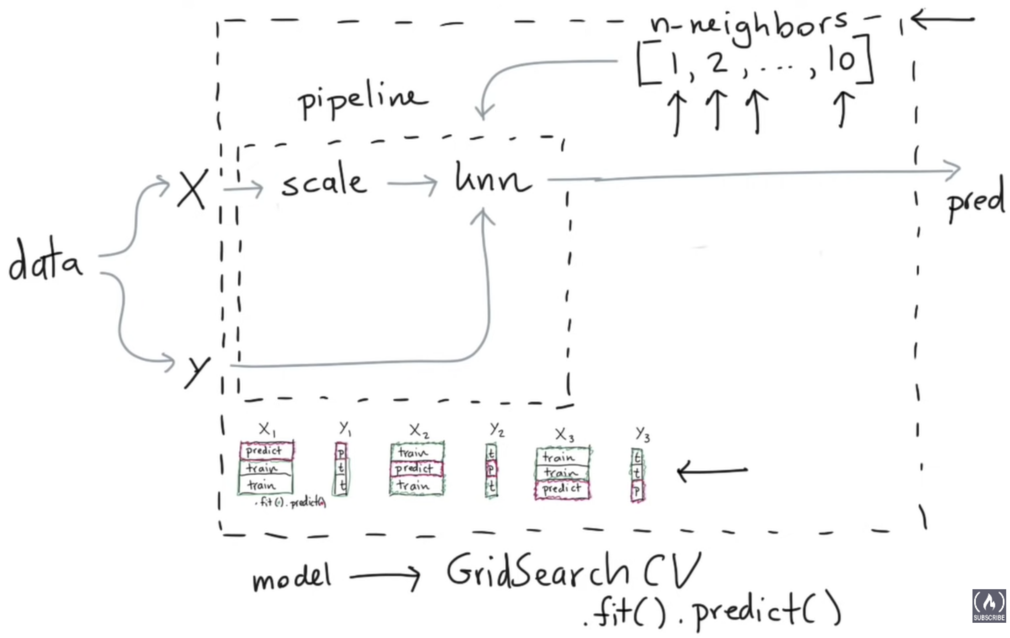


GridSearchCV

GridSearchCV is a function that comes in Scikit-learn's model_selection package to find the best values for hyperparameters of a model. GridSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used. CV means cross validation

Parameters used in GridSearchCV:

1. **estimator**: The machine learning model or algorithm you want to tune.
2. **param_grid**: A dictionary or list of dictionaries with parameters names (strings) as keys and lists of parameter settings to try as values.
3. **scoring**: A string, callable, list/tuple, or dict, default=None. Strategy to evaluate the performance of the cross-validated model on the test set.
4. **n_jobs**: Number of jobs to run in parallel. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors.
5. **iid**: If True, return the average score across folds, weighted by the number of samples in each test set. This parameter is deprecated and will be removed in version 0.24.
6. **refit**: Refit an estimator using the best found parameters on the whole dataset.
7. **cv**: Determines the cross-validation splitting strategy.
8. **verbose**: Controls the verbosity: the higher, the more messages.
9. **pre_dispatch**: Controls the number of jobs that get dispatched during parallel execution.
10. **error_score**: Value to assign to the score if an error occurs in estimator fitting.



Rank_test_score: In GridSearchCV, the rank_test_score attribute is a vector that indicates the rank of each parameter combination based on the mean_test_score.

The parameter combination that results in the lowest mean_test_score will have a rank_test_score of N and the parameter combination with the highest mean_test_score will have a rank_test_score of 1.

In [32]:
```
# Importing
from sklearn.model_selection import GridSearchCV
```

The `GridSearchCV` function from `sklearn.model_selection` is a powerful tool for hyperparameter tuning in machine learning. Here's what you can do with it:

1. **Automate Hyperparameter Tuning**: GridSearchCV exhaustively searches through a specified grid of hyperparameters and evaluates the model performance for each combination. This allows you to find the optimal hyperparameters for your model without the need for manual tuning.
2. **Evaluate Model Performance**: GridSearchCV trains and evaluates a machine learning model using different combinations of hyperparameters. The best set of hyperparameters is then selected based on a specified performance metric.
3. **Optimize Classifier**: GridSearchCV can be used to optimize your classifier and iterate through different parameters to find the best model. It can provide you with the best parameters from the set you enter.

In summary, `GridSearchCV` helps in building successful machine learning models by automating the process of hyperparameter tuning and model evaluation.

In [33]:
```
x, y = load_diabetes(return_X_y = True)

pipe = Pipeline([
    ('Scale', StandardScaler()),
    ('model', KNeighborsRegressor(n_neighbors = 1))
])

pipe.get_params()
```

Out[33]:
```
{'memory': None,
 'steps': [('Scale', StandardScaler()),
  ('model', KNeighborsRegressor(n_neighbors=1))],
 'verbose': False,
 'Scale': StandardScaler(),
 'model': KNeighborsRegressor(n_neighbors=1),
 'Scale__copy': True,
 'Scale__with_mean': True,
 'Scale__with_std': True,
 'model__algorithm': 'auto',
 'model__leaf_size': 30,
 'model__metric': 'minkowski',
 'model__metric_params': None,
 'model__n_jobs': None,
 'model__n_neighbors': 1,
 'model__p': 2,
 'model__weights': 'uniform'}
```

In [34]:
```
mod_grid = GridSearchCV(estimator = pipe,
             param_grid = {'model__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9,
             cv = 3)

mod_grid
```

Out[34]:
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('Scale', StandardScaler()),
                                       ('model',
                                        KNeighborsRegressor(n_neighbors=
1))]),
             param_grid={'model__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9,
10]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [35]:
```
mod_grid.fit(x, y)
pd.DataFrame(mod_grid.cv_results_)
```

Out[35]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model__n_neighbors
0	0.003060	0.000881	0.002736	0.000011	1
1	0.003227	0.001501	0.003233	0.000426	2
2	0.002160	0.000064	0.002956	0.000093	3
3	0.002109	0.000068	0.003079	0.000275	4
4	0.002059	0.000061	0.002881	0.000037	5
5	0.001997	0.000027	0.002930	0.000042	6
6	0.002026	0.000060	0.002965	0.000009	7
7	0.002041	0.000062	0.003287	0.000373	8
8	0.002424	0.000421	0.003186	0.000127	9
9	0.002012	0.000049	0.003033	0.000052	10

Scaling and Pre-processing

In [36]:
```
df = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/5._CarPriceData_
```

In [37]:
```
df.head(4)
```

Out[37]:

	ID	Company	Model	Type	Fuel	Transmission	Engine	Mileage	Kms_driven	Buye
0	1	Maruti	Alto	Hatchback	Petrol	Manual	796	19.7	45000	
1	2	Maruti	Wagon R	Hatchback	Petrol	Manual	998	20.5	40005	
2	3	Maruti	Wagon R	Hatchback	Petrol	Manual	998	20.5	40005	
3	4	Maruti	Ertiga	MUV	Petrol	Automatic	1462	18.5	28000	

In [38]:
```python
x = df[['Mileage ', 'Kms_driven']].values
x
```

Out[38]:
```
array([[1.9700e+01, 4.5000e+04],
       [2.0500e+01, 4.0005e+04],
       [2.0500e+01, 4.0005e+04],
       [1.8500e+01, 2.8000e+04],
       [1.8500e+01, 4.0000e+04],
       [2.3500e+01, 3.6000e+04],
       [2.0890e+01, 4.1000e+04],
       [2.0890e+01, 4.1000e+04],
       [2.0040e+01, 2.5000e+04],
       [2.0040e+01, 1.5000e+04],
       [1.9560e+01, 2.4530e+04],
       [2.5110e+01, 6.0000e+04],
       [2.5110e+01, 4.0000e+04],
       [2.1500e+01, 6.0000e+04],
       [2.3840e+01, 3.0000e+04],
       [2.3840e+01, 5.0000e+04],
       [1.7000e+01, 3.2000e+04],
       [1.7000e+01, 2.0000e+04],
       [2.0300e+01, 4.8660e+04],
```

In [39]:
```python
y = df['Price (Lakhs)'].values
y
```

Out[39]:
```
array([  1.2 ,   3.  ,   4.  ,   5.1 ,   4.  ,   1.58,   2.5 ,   3.  ,
         6.1 ,   5.  ,   2.3 ,   2.8 ,   3.4 ,   5.02,   3.12,   3.8 ,
         9.35,   6.7 ,   2.87,  10.  ,   4.7 ,   0.8 ,   8.24,   4.7 ,
         8.1 ,   5.29,   6.54,   3.8 ,   7.5 ,   3.32,   4.1 ,   4.5 ,
         6.45,   4.9 ,   5.8 ,   5.  ,   5.2 ,   6.31,   0.9 ,   1.9 ,
         5.67,   3.66,   8.2 ,   2.33,   1.67,   4.  ,  12.93,   5.93,
         1.56,   1.89,   2.12,   3.72,   2.29, 190.  , 120.  , 100.  ,
       106.  , 200.  , 350.  , 300.  , 400.  , 200.  , 262.  ,  38.  ,
        65.  , 133.  ,  90.  ,  40.33,   8.09,   5.  ,   4.72,   4.  ,
         7.83,  10.77,   4.72,   7.8 ,   5.15,   6.16,  10.  ,   6.  ,
         5.51,   8.  ,  67.  ,  50.81,  35.  ,  30.  ,  20.04,  50.  ,
        78.  ,  80.  ,  34.98,  30.  ,  40.  ,  50.  ,  88.  ,  80.  ,
        80.  ,  80.  ,   3.17,   7.3 ,  18.74,   2.58,   3.4 ,   4.17,
         8.17, 111.  ,  50.  ,  36.46,  75.  ,  60.  , 100.  ,  76.  ,
       100.  , 100.  ,  19.96,  70.  ,  27.  ,  87.  ,  50.  ,  70.  ,
        23.96,  81.  ,  20.  ,  50.6 ,  32.  ,  65.  ,  70.  ,  40.  ,
        21.32,  49.01,  31.44,  23.  ,  50.56, 162.  ,  67.  , 181.  ,
       301.  , 167.  , 122.  , 111.  , 200.  , 100.  , 115.  ,  89.  ,
       605.  , 189.  , 370.  , 495.  , 280.  , 292.  ])
```

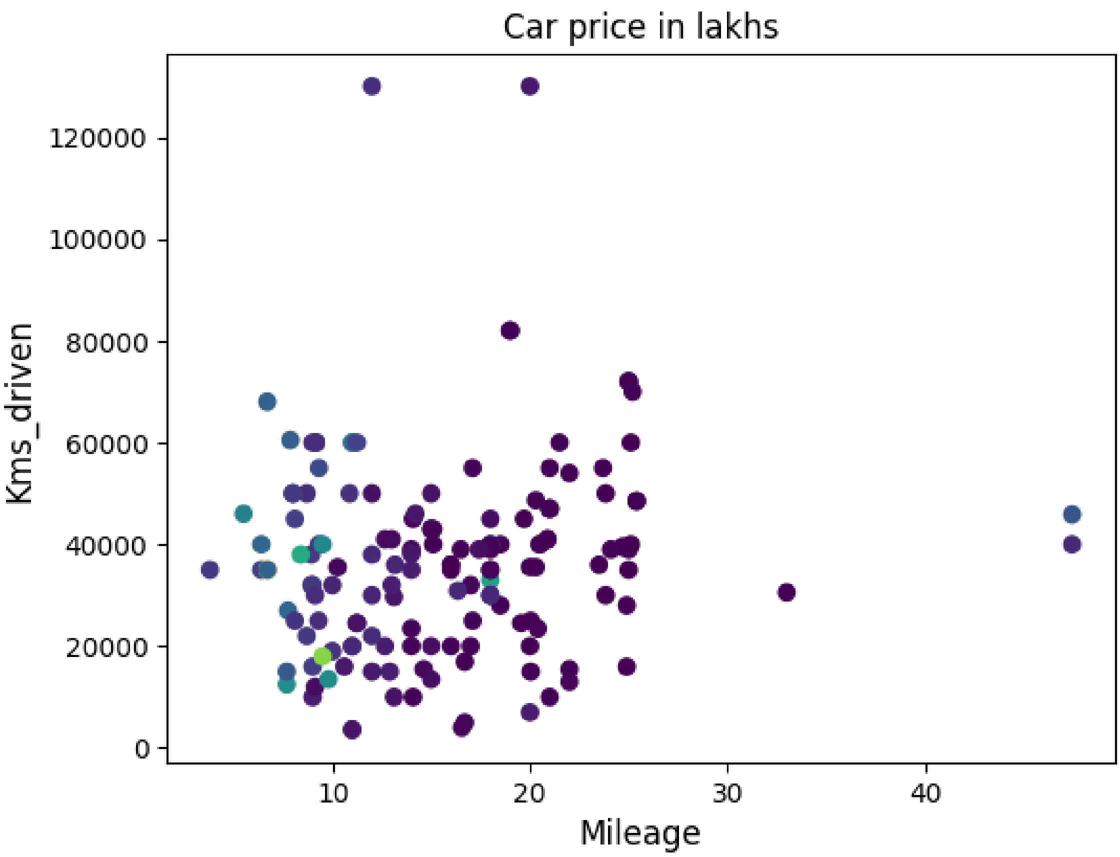
x[:, 0] selects all rows (:) from the first column (0) of the array x.

x[:, 1] selects all rows (:) from the second column (1) of the array x.

So, if x is a two-dimensional array where each row represents a data point and each column represents a feature, x[:, 0] would represent all data points' first feature and x[:, 1] would represent all data points' second feature. These are then used as the x and y coordinates for the scatter plot.

In [40]:
```
plt.title("Car price in lakhs", fontsize = 12)
plt.xlabel("Mileage", fontsize = 12)
plt.ylabel("Kms_driven", fontsize = 12)

plt.scatter(x[:, 0], x[:, 1], c=y) # c defines color based on values
```

Out[40]:
```
<matplotlib.collections.PathCollection at 0x7cf5d8333370>
```

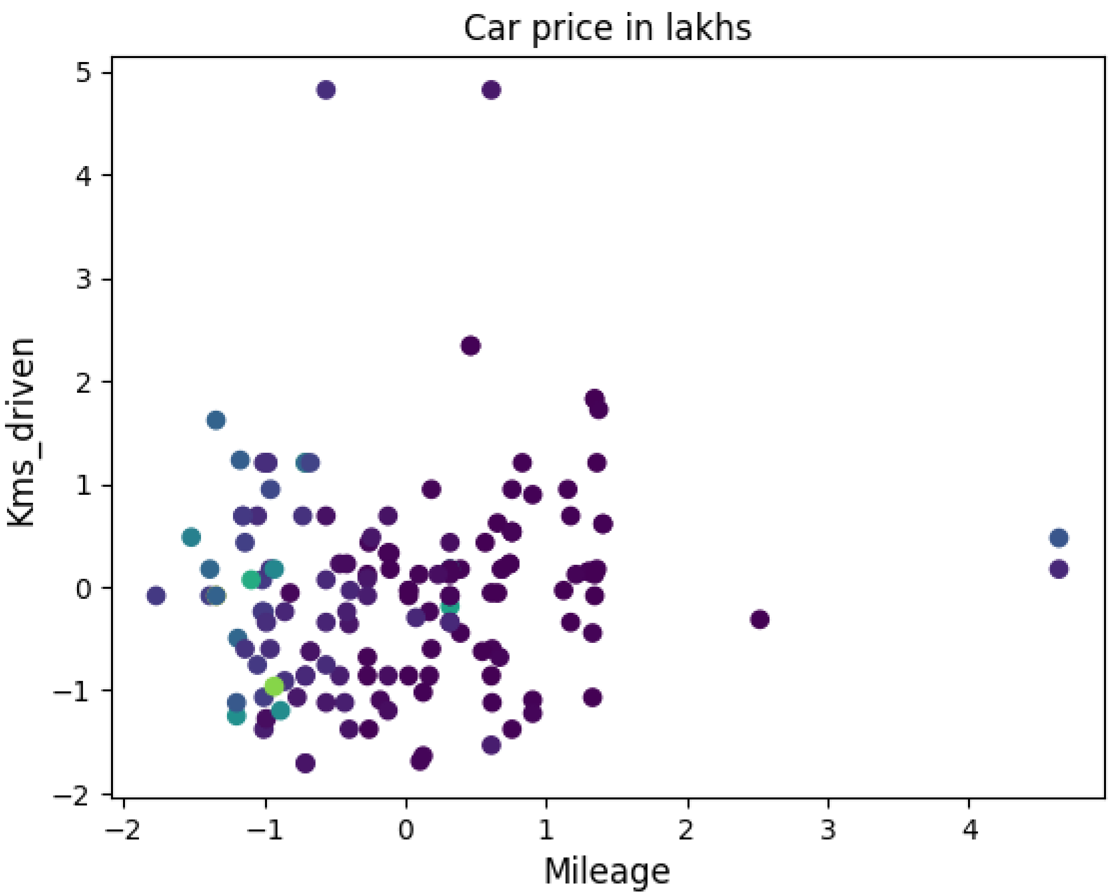


Standard Scalar

In [41]:
```
from sklearn.preprocessing import StandardScaler
```

In [42]:
```
# Through this we are scaling the values of x and y axis (using StandardScal
x_new = StandardScaler().fit_transform(x)
plt.scatter(x_new[:, 0], x_new[:, 1], c=y)

plt.title("Car price in lakhs", fontsize = 12)
plt.xlabel("Mileage", fontsize = 12)
plt.ylabel("Kms_driven", fontsize = 12)
```

Out[42]: `Text(0, 0.5, 'Kms_driven')`

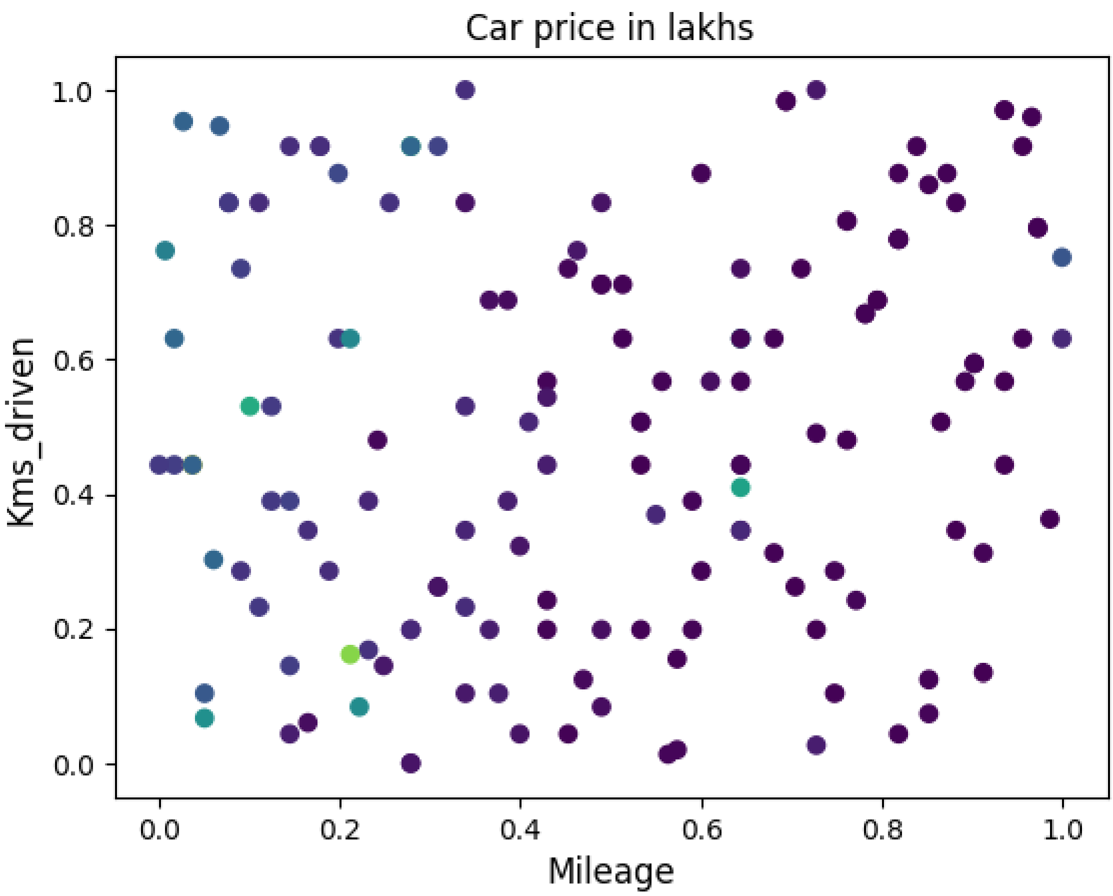


Quantile Transformer

In [43]:
```
from sklearn.preprocessing import QuantileTransformer
```

In [44]:
```
# Through this we are scaling the values of x and y axis (using QuantilleTra
x_new = QuantileTransformer(n_quantiles = 150).fit_transform(x) # n_quantile
plt.scatter(x_new[:, 0], x_new[:, 1], c=y)

plt.title("Car price in lakhs", fontsize = 12)
plt.xlabel("Mileage", fontsize = 12)
plt.ylabel("Kms_driven", fontsize = 12)
```

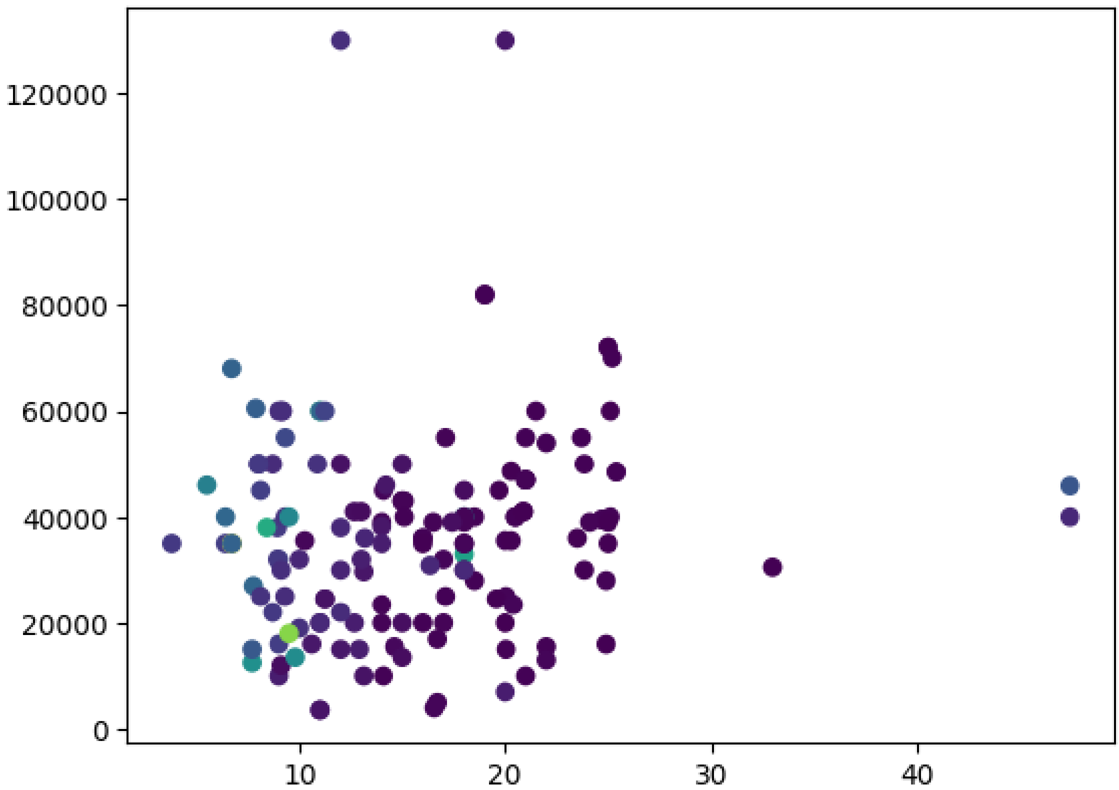
Out[44]:
```
Text(0, 0.5, 'Kms_driven')
```



Implementing the chart in Logistic Regression with Quartile Transformer

In [45]:
```
df = pd.read_csv("/content/drive/MyDrive/ML and DL DataSets/5._CarPriceData_
x = df[['Mileage ', 'Kms_driven']].values
y = df['Price (Lakhs)'].values

plt.scatter(x[:, 0], x[:, 1], c=y)
```

Out[45]:
```
<matplotlib.collections.PathCollection at 0x7cf5d80d87f0>
```



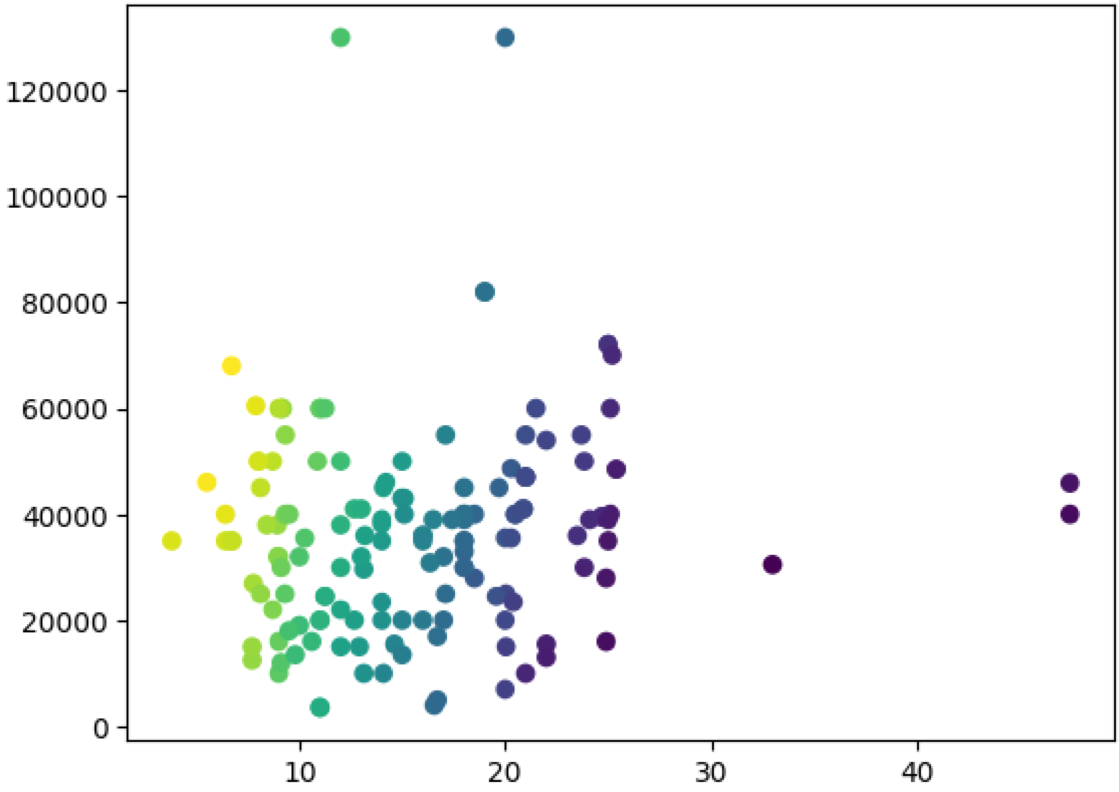
In [46]:
```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
```

In [47]:
```
pipe = Pipeline([
    ('Scale', QuantileTransformer(n_quantiles = 150)),
    ('model', LinearRegression()) # If we want to do logistic regression, we
                                  # ValueError: Unknown label type: 'continu
])

pipe.get_params()
```

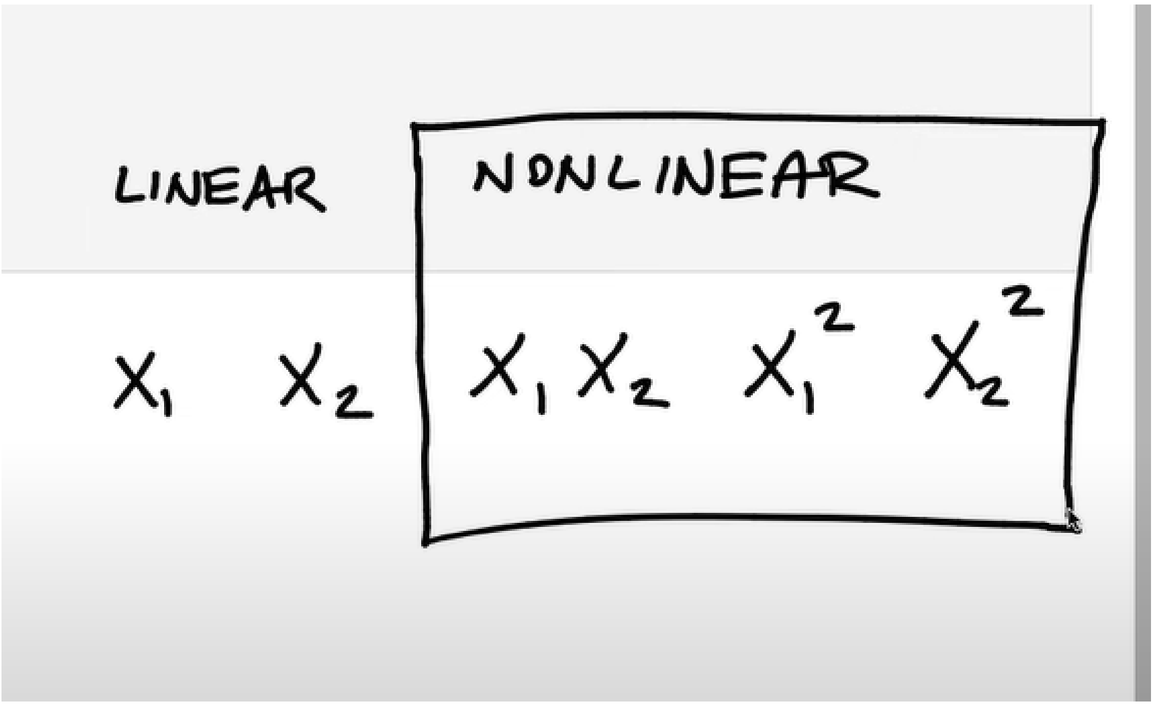
Out[47]:
```
{'memory': None,
 'steps': [('Scale', QuantileTransformer(n_quantiles=150)),
  ('model', LinearRegression())],
 'verbose': False,
 'Scale': QuantileTransformer(n_quantiles=150),
 'model': LinearRegression(),
 'Scale__copy': True,
 'Scale__ignore_implicit_zeros': False,
 'Scale__n_quantiles': 150,
 'Scale__output_distribution': 'uniform',
 'Scale__random_state': None,
 'Scale__subsample': 10000,
 'model__copy_X': True,
 'model__fit_intercept': True,
 'model__n_jobs': None,
 'model__positive': False}
```

In [48]:
```
pred_linear = pipe.fit(x, y).predict(x) # For linear and logistic model we h
plt.scatter(x[:, 0], x[:, 1], c = pred_linear)
```

Out[48]: <matplotlib.collections.PathCollection at 0x7cf5d814d240>

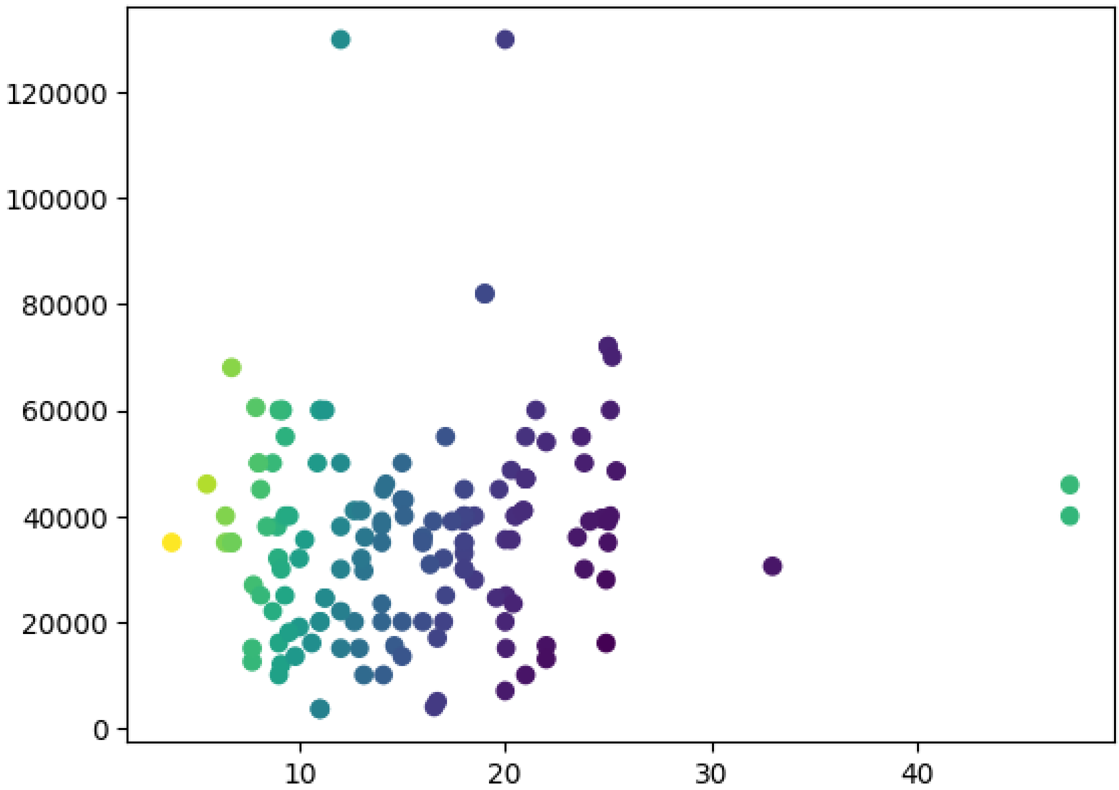


Using non linear axis in pipeline



In [49]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

pipe = Pipeline([
    ('Scale', PolynomialFeatures()),
    ('model', LinearRegression())
])

pred_linear = pipe.fit(x, y).predict(x)
plt.scatter(x[:, 0], x[:, 1], c = pred_linear)
```

Out[49]:
```
<matplotlib.collections.PathCollection at 0x7cf5d7fbdf30>
```



Preprocessing on string data

In [50]:
```
status = np.array(['lower', 'lower', 'rich', 'middle', 'rich']).reshape(-1,
status
```

Out[50]:
```
array([['lower'],
       ['lower'],
       ['rich'],
       ['middle'],
       ['rich']], dtype='<U6')
```

In [51]:
```
# OneHotEncoder is a scikit-learn function that encodes categorical features
from sklearn.preprocessing import OneHotEncoder
```

In [52]:
```python
enc = OneHotEncoder(sparse_output = False, handle_unknown = 'ignore')
enc.fit_transform(status) # This will print string as vector format
```

Out[52]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Metrics

Scikit-learn provides a number of metrics for evaluating the performance of machine learning models. These metrics can be used to assess the accuracy, precision, recall, and F1-score of a model.

Scikit-learn provides a variety of metrics for evaluating the quality of a model's predictions. Here are the metrics available in Scikit-learn:

➤ **Classification Metrics**:

Some of these are restricted to the binary classification case:

1. precision_recall_curve(y_true, probas_pred, *): Compute precision-recall pairs for different probability thresholds.
2. roc_curve(y_true, y_score, *[, pos_label, ...]): Compute Receiver operating characteristic (ROC).
3. class_likelihood_ratios(y_true, y_pred, *[, ...]): Compute binary classification positive and negative likelihood ratios.
4. det_curve(y_true, y_score[, pos_label, ...]): Compute error rates for different probability thresholds.

Others also work in the multiclass case:

1. balanced_accuracy_score(y_true, y_pred, *[, ...]): Compute the balanced accuracy.
2. cohen_kappa_score(y1, y2, *[, labels, ...]): Compute Cohen's kappa: a statistic that measures inter-annotator agreement.
3. confusion_matrix(y_true, y_pred, *[, ...]): Compute confusion matrix to evaluate the accuracy of a classification.
4. hinge_loss(y_true, pred_decision, *[, ...]): Average hinge loss (non-regularized).
5. matthews_corrcoef(y_true, y_pred, *[, ...]): Compute the Matthews correlation coefficient (MCC).
6. roc_auc_score(y_true, y_score, *[, average, ...]): Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
7. top_k_accuracy_score(y_true, y_score, *[, ...]): Top-k Accuracy classification score.

Some also work in the multilabel case:

1. accuracy_score(y_true, y_pred, *[, ...]): Accuracy classification score.
2. classification_report(y_true, y_pred, *[, ...]): Build a text report showing the main classification metrics.
3. f1_score(y_true, y_pred, *[, labels, ...]): Compute the F1 score, also known as balanced F-score or F-measure.
4. fbeta_score(y_true, y_pred, *, beta[, ...]): Compute the F-beta score.
5. hamming_loss(y_true, y_pred, *[, sample_weight]): Compute the average Hamming loss.
6. jaccard_score(y_true, y_pred, *[, labels, ...]): Jaccard similarity coefficient score.

7. log_loss(y_true, y_pred, *[, eps, ...]): Log loss, aka logistic loss or cross-entropy loss.
8. multilabel_confusion_matrix(y_true, y_pred, *): Compute a confusion matrix for each class or sample.
9. precision_recall_fscore_support(y_true, ...): Compute precision, recall, F-measure and support for each class.
10. precision_score(y_true, y_pred, *[, labels, ...]): Compute the precision.
11. recall_score(y_true, y_pred, *[, labels, ...]): Compute the recall.
12. roc_auc_score(y_true, y_score, *[, average, ...]): Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
13. zero_one_loss(y_true, y_pred, *[, ...]): Zero-one classification loss.

And some work with binary and multilabel (but not multiclass) problems:

1. average_precision_score(y_true, y_score, *): Compute average precision (AP) from prediction scores.

➤ **Multilabel Ranking Metrics**:

1. **Coverage Error**: It measures how far we need to go through the ranked scores to cover all true labels. The best value is equal to the average number of labels in `y_true` per sample.
2. **Label Ranking Average Precision (LRAP)**: It measures the label rankings of each sample. The best value is 1.
3. **Ranking Loss**: It is defined as the number of incorrectly ordered labels with respect to the number of correctly ordered labels. The best value is zero.
4. **Normalized Discounted Cumulative Gain (NDCG)**: It is a measure of ranking quality independent of the particular query.

➤ **Regression Metrics**:

1. **R^2 Score (Coefficient of Determination)**: It measures the proportion of the variance in the dependent variable that can be predicted from the independent variable[25]. It ranges from 0 to 1, where 1 indicates a perfect fit.
2. **Mean Absolute Error (MAE)**: It is the average of the absolute differences between the predicted and actual values. It quantifies the average magnitude of errors in a set of predictions, without considering their direction.
3. **Mean Squared Error (MSE)**: It is the average of the squared differences between the predicted and actual values. Squaring the differences gives more weight to larger differences.
4. **Mean Squared Logarithmic Error (MSLE)**: It is a variation of MSE that only cares about the percentual difference. It utilizes a logarithm to offset large outliers in a dataset and treats them as if they were on the same scale.
5. **Mean Absolute Percentage Error (MAPE)**: It is a measure of prediction accuracy of a forecasting method in statistics, usually expressed as a ratio. It represents the average of the absolute percentage errors of each entry in a dataset.
6. **Median Absolute Error**: It is the median difference between the observations (true values) and model output (predictions).
7. **Max Error**: This metric represents the maximum residual error value, i.e., it gives us an idea about the worst case error between the predicted value and the true value.
8. **Explained Variance Score**: This score measures how much variance in your data is explained by your model[^30^]. A higher score indicates that your model explains more variance, which means it's more accurate.

➤ **Clustering Metrics**

These metrics help in assessing prediction error for specific purposes and measure the

In [53]:
```
fraud_data = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/5._Credi
fraud_data.head(4)
```

Out[53]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436

4 rows × 31 columns

In [54]:
```
# Taking data of x (removing Time, Amount and Class column from data)
x = fraud_data.drop(columns = ['Time', 'Amount', 'Class']).values
x
```

Out[54]:
```
array([[-1.35980713e+00, -7.27811733e-02,  2.53634674e+00, ...,
        -1.89114844e-01,  1.33558377e-01, -2.10530535e-02],
       [ 1.19185711e+00,  2.66150712e-01,  1.66480113e-01, ...,
         1.25894532e-01, -8.98309914e-03,  1.47241692e-02],
       [-1.35835406e+00, -1.34016307e+00,  1.77320934e+00, ...,
        -1.39096572e-01, -5.53527940e-02, -5.97518406e-02],
       ...,
       [ 1.91956501e+00, -3.01253846e-01, -3.24963981e+00, ...,
        -8.73705959e-02,  4.45477214e-03, -2.65608286e-02],
       [-2.40440050e-01,  5.30482513e-01,  7.02510230e-01, ...,
         5.46668462e-01,  1.08820735e-01,  1.04532821e-01],
       [-5.33412522e-01, -1.89733337e-01,  7.03337367e-01, ...,
        -8.18267121e-01, -2.41530880e-03,  1.36489143e-02]])
```

In [55]:
```
# Taking data of y as prediction data(class)
y = fraud_data['Class'].values
y
```

Out[55]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [56]:
```
# Printing the shape of x and y. And total fraud cases
print("Shapes of X",x.shape,"Shapes of Y",y.shape,"Total fraud cases",y.sum(
```

```
Shapes of X (284807, 28) Shapes of Y (284807,) Total fraud cases 492
```

Predicting through Logistic Regression

In [57]:
```
from sklearn.linear_model import LogisticRegression
```

In [58]:
```
mod_logistic = LogisticRegression(max_iter = 1000) # The datset is very huge
mod_logistic.fit(x, y).predict(x).sum() # LogisticRegression predicting fewe
```

Out[58]:
```
348
```

Let's make some changes to detect fraud cases

In [59]:
```
mod_logistic = LogisticRegression(max_iter = 1000,
                                  class_weight = {0: 1, 1 : 2}) # Throgh thi
mod_logistic.fit(x, y).predict(x).sum()
```

Out[59]:
```
437
```

Model Selection through GridSearchCV

In [60]:
```
grid_logistic = GridSearchCV(
    estimator = LogisticRegression(max_iter = 1000),
    param_grid = {'class_weight': [{0: 1, 1 : v} for v in range (1, 4)]}, #
    cv = 6, # This will split the data in 6 parts
    n_jobs = -1
)
grid_logistic
```

Out[60]:
```
GridSearchCV(cv=6, estimator=LogisticRegression(max_iter=1000), n_jobs=-1,
             param_grid={'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2},
                                          {0: 1, 1: 3}]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
grid_logistic.fit(x, y).predict(x).sum()
```

Out[61]:
```
462
```

In [62]:
```
grid_logistic.cv_results_
```

Out[62]:
```
{'mean_fit_time': array([4.98879405, 5.90031143, 5.27042262]),
 'std_fit_time': array([1.70463855, 1.14834491, 1.20955792]),
 'mean_score_time': array([0.01617948, 0.02426585, 0.03202923]),
 'std_score_time': array([0.0081367 , 0.0109966 , 0.01903262]),
 'param_class_weight': masked_array(data=[{0: 1, 1: 1}, {0: 1, 1: 2}, {0:
1, 1: 3}],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'class_weight': {0: 1, 1: 1}},
  {'class_weight': {0: 1, 1: 2}},
  {'class_weight': {0: 1, 1: 3}}],
 'split0_test_score': array([0.99890453, 0.99890453, 0.99892559]),
 'split1_test_score': array([0.99932586, 0.999368  , 0.99953653]),
 'split2_test_score': array([0.99890453, 0.99922053, 0.99934693]),
 'split3_test_score': array([0.999368  , 0.99913626, 0.99905199]),
 'split4_test_score': array([0.99907306, 0.99928373, 0.99932586]),
 'split5_test_score': array([0.99919944, 0.99917838, 0.99917838]),
 'mean_test_score': array([0.99912924, 0.9991819 , 0.99922755]),
 'std_test_score': array([0.00018473, 0.00014464, 0.00020158]),
 'rank_test_score': array([3, 2, 1], dtype=int32)}
```

In [63]:
```
pd.DataFrame(grid_logistic.cv_results_) #highest mean_test_score will set ro
```

Out[63]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_class_weight	p
0	4.988794	1.704639	0.016179	0.008137	{0: 1, 1: 1}	{'class_\ {0:
1	5.900311	1.148345	0.024266	0.010997	{0: 1, 1: 2}	{'class_\ {0:
2	5.270423	1.209558	0.032029	0.019033	{0: 1, 1: 3}	{'class_\ {0:

Using metrices

In [64]:
```
''' By default scikit-learn uses 'accuracy_score' to get the mean_test_score
```

Out[64]:
```
" By default scikit-learn uses 'accuracy_score' to get the mean_test_score
"
```

In [65]:
```
from sklearn.metrics import precision_score, recall_score, make_scorer
```

In [66]:
```
''' Suppose in extreme case, all datapoint cases are fraud then recall_score
```

Out[66]:
```
' Suppose in extreme case, all datapoint cases are fraud then recall_score
will high and precision_score will low, and vice versa '
```

In [67]:
```
# precision_score will tell: given that I predict fraud, how accurate am I
precision_score (y, grid_logistic.predict(x))
```

Out[67]:
```
0.816017316017316
```

In [68]:
```
# recall_score will tell: did I get all the fraud cases
recall_score (y, grid_logistic.predict(x))
```

Out[68]:
```
0.766260162601626
```

Adding precision_score and recall_score in GridSearchCV

In [69]:
```
grid_log_metr = GridSearchCV(
    estimator = LogisticRegression(max_iter = 1000),
    param_grid = {'class_weight': [{0: 1, 1 : v} for v in range (1, 4)]},
    scoring = {'precision': make_scorer(precision_score), 'recall_score': ma
    refit = 'precision',
    return_train_score = True, # To cross check with test score
    cv = 6, # The higher cv will help to predict more accurately. It also to
    n_jobs = -1
)
grid_log_metr
```

Out[69]:
```
GridSearchCV(cv=6, estimator=LogisticRegression(max_iter=1000), n_jobs=-1,
             param_grid={'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2},
                                          {0: 1, 1: 3}]},
             refit='precision', return_train_score=True,
             scoring={'precision': make_scorer(precision_score),
                      'recall_score': make_scorer(recall_score)})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [70]:
```
grid_log_metr.fit(x, y)
num = pd.DataFrame(grid_log_metr.cv_results_)
num
```

Out[70]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_class_weight	p
0	3.687644	0.979976	0.052186	0.003489	{0: 1, 1: 1}	{'class_ {0:
1	3.520839	0.597726	0.076740	0.032730	{0: 1, 1: 2}	{'class_ {0:
2	3.370100	0.559737	0.069149	0.025899	{0: 1, 1: 3}	{'class_ {0:

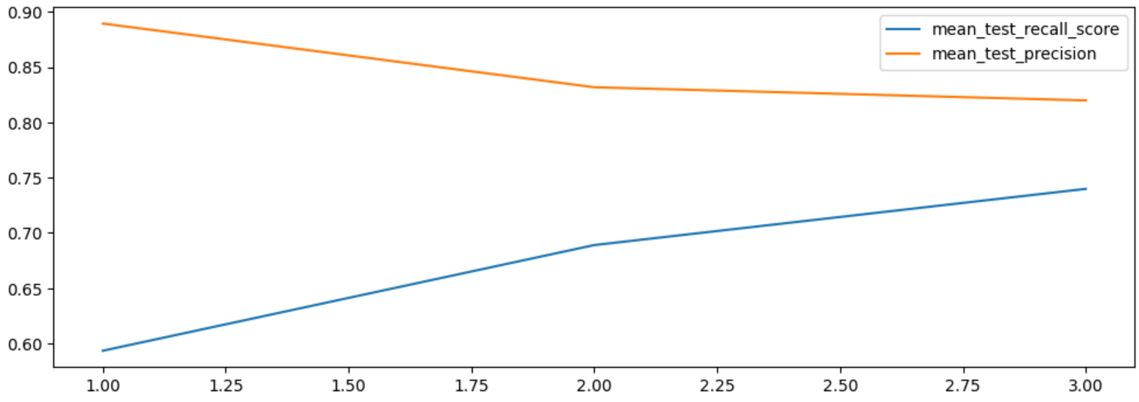
3 rows × 40 columns

In [71]:
```
num.columns
```

Out[71]:
```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
e',
       'param_class_weight', 'params', 'split0_test_precision',
       'split1_test_precision', 'split2_test_precision',
       'split3_test_precision', 'split4_test_precision',
       'split5_test_precision', 'mean_test_precision', 'std_test_precisio
n',
       'rank_test_precision', 'split0_train_precision',
       'split1_train_precision', 'split2_train_precision',
       'split3_train_precision', 'split4_train_precision',
       'split5_train_precision', 'mean_train_precision', 'std_train_precis
ion',
       'split0_test_recall_score', 'split1_test_recall_score',
       'split2_test_recall_score', 'split3_test_recall_score',
       'split4_test_recall_score', 'split5_test_recall_score',
       'mean_test_recall_score', 'std_test_recall_score',
       'rank_test_recall_score', 'split0_train_recall_score',
       'split1_train_recall_score', 'split2_train_recall_score',
       'split3_train_recall_score', 'split4_train_recall_score',
       'split5_train_recall_score', 'mean_train_recall_score',
       'std_train_recall_score'],
      dtype='object')
```

In [72]:
```
plt.figure(figsize=(12, 4))
df_result = pd.DataFrame (grid_log_metr.cv_results_)

# In x axis it's class weight and in y axis it's scores
for score in ['mean_test_recall_score', 'mean_test_precision']: # This loop
  plt.plot([_[1] for _ in df_result['param_class_weight']], # In x-values, i
           df_result[score], # In y-values, it’s using the values in df_resu
           label=score)
plt.legend()
```

Out[72]: <matplotlib.legend.Legend at 0x7cf5d83872e0>



Detect Outlier

In [73]:
```
# The collections.Counter is a dict subclass in Python for counting hashable
from collections import Counter

# IsolationForest is a machine learning algorithm for anomaly detection.
from sklearn.ensemble import IsolationForest

mod_outlier = IsolationForest().fit(x)
Counter(mod_outlier.predict(x)) # Here, 1 represents not outlier and -1 repr
```

Out[73]:
```
Counter({1: 274920, -1: 9887})
```

In [74]:
```
# Transleting the values of outlier
Counter(np.where(mod_outlier.predict(x) == -1, 1, 0)) # Here, 0 represents r
```

Out[74]:
```
Counter({0: 274920, 1: 9887})
```

Meta Estimators

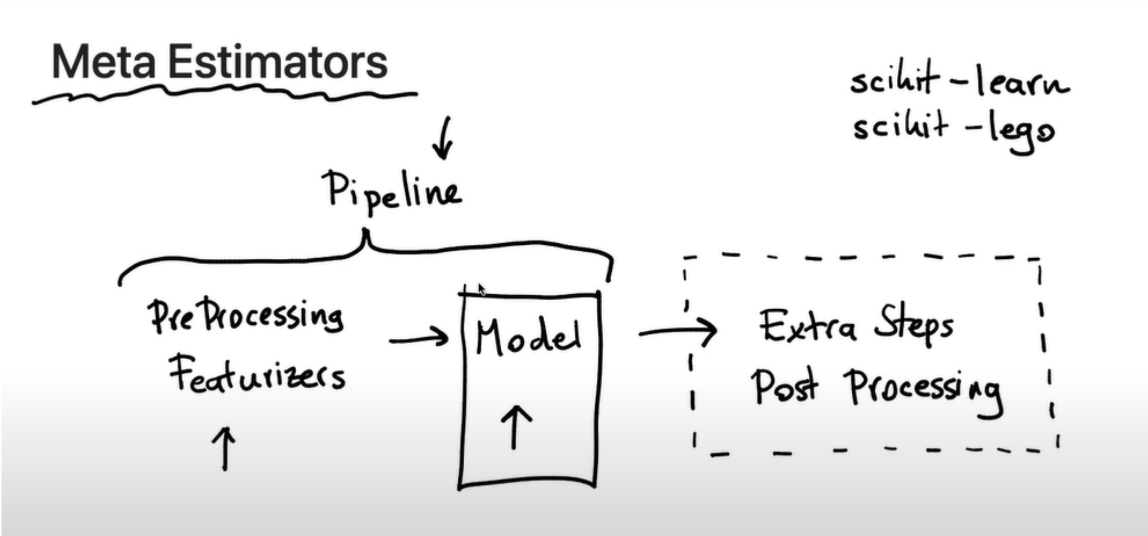
In Scikit-Learn, a meta-estimator is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor. All estimators implement the `fit` method.

A meta-estimator in Scikit-Learn is an estimator that takes other estimators as input and combines their outputs in some way to make a final prediction. They are used to improve the performance of the base estimators by leveraging their strengths and mitigating their weaknesses.

Here are some examples of meta-estimators in Scikit-Learn:

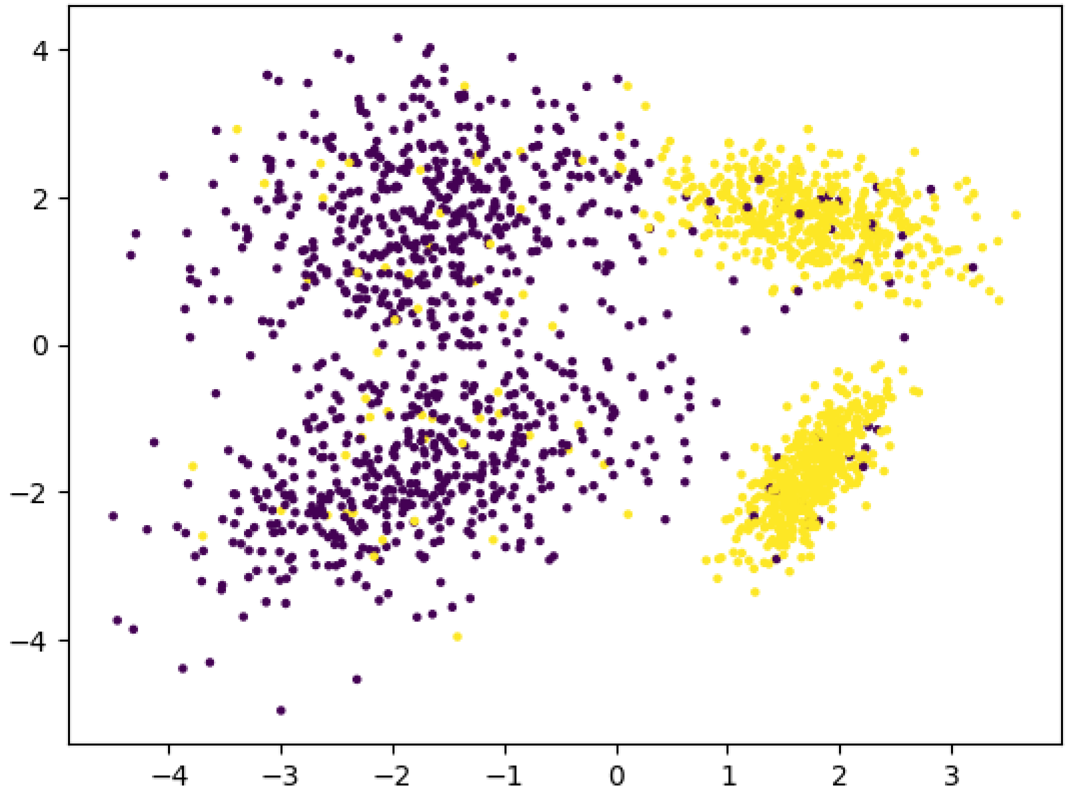
1. **Random Forest**: A random forest is a meta-estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
2. **AdaBoost**: An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
3. **Bagging Classifier**: A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

These meta-estimators provide powerful methods for improving model performance by combining multiple models together.



In [75]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.datasets import make_classification
```

In [76]:
```
X, y = make_classification(n_samples=2000, n_features=2,
                           n_redundant=0, random_state=21,
                           class_sep=1.75, flip_y=0.1)
plt.scatter(X[:, 0], X[:, 1], c=y, s=5);
```



Wine Quality Analysis

Importing Libraries

In [77]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import svm
from sklearn.neural_network import MLPClassifier
#from sklearn. Linear_model import SGDC Lassifier
from sklearn.metrics import confusion_matrix, classification_report, accurac
from sklearn.preprocessing import StandardScaler, LabelEncoder #  These are
from sklearn.model_selection import train_test_split
```

Explanation of classes in Library imported here

1. **StandardScaler:** is a class that standardizes features by removing the mean and scaling to unit variance. This is often a good preprocessing step to do if you're working with algorithms that are sensitive to the scale of the features, like Support Vector Machines (SVM) or k-nearest neighbors (KNN).
2. **LabelEncoder:** is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1. It can be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.
3. **Train_test_split:** The purpose of splitting your data is to be able to evaluate your model's performance on unseen data. During the training phase, your model learns patterns from the training data. You then test the model on the testing data to see how well it generalizes to new, unseen data.

In [78]:
```
wine_data = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/5._WineQu
wine_data
```

Out[78]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 12 columns

In [79]:
```
wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [80]:
```
# View and count the category based on the quality of wine
wine_data['quality'].value_counts()
```

Out[80]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

Preprocessing Data

Getting 3 categorical value (For example)

In [81]:
```
wine_data_3 = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/5._Wine
```

In [82]:
```
# If we want to get 3 categorical value
bin_3 = (2, 5, 6.5, 8)
names_3 = ('bad', 'better', 'good')

wine_data_3['quality'] = pd.cut(wine_data_3['quality'], bins = bin_3, labels
wine_data_3['quality'].unique()
```

Out[82]:
```
['bad', 'better', 'good']
Categories (3, object): ['bad' < 'better' < 'good']
```

Getting 2 categorical value

In [83]:
```
bin = (2, 6.5, 8) # ‘quality’ that are greater than 2 and up to 6.5 will be
names = ('bad', 'good') # contains the labels for the two categories

#  Here, ‘quality’ column where the original numerical values have been repl
wine_data['quality'] = pd.cut(wine_data['quality'], bins = bin, labels = nam
wine_data['quality'].unique() # Returning the categorical value
```

Out[83]:
```
['bad', 'good']
Categories (2, object): ['bad' < 'good']
```

In [84]:
```
# Printing the 'bad' wine categorical value
print(wine_data.loc[wine_data['quality'] == 'bad'])
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlori
des  \
0               7.4             0.700         0.00             1.9      0.
076
1               7.8             0.880         0.00             2.6      0.
098
2               7.8             0.760         0.04             2.3      0.
092
3              11.2             0.280         0.56             1.9      0.
075
4               7.4             0.700         0.00             1.9      0.
076
...             ...               ...          ...             ...
...
1594            6.2             0.600         0.08             2.0      0.
090
1595            5.9             0.550         0.10             2.2      0.
062
1596            6.3             0.510         0.13             2.3      0.
076
1597            5.9             0.645         0.12             2.0      0.
075
1598            6.0             0.310         0.47             3.6      0.
067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates
\
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol quality
0         9.4     bad
1         9.8     bad
2         9.8     bad
3         9.8     bad
4         9.4     bad
...       ...     ...
1594     10.5     bad
1595     11.2     bad
1596     11.0     bad
1597     10.2     bad
1598     11.0     bad

[1382 rows x 12 columns]
```

Assuming X is your feature matrix and y are your labels

In [85]:
```
wine_encoder = LabelEncoder()
wine_data['quality'] = wine_encoder.fit_transform(wine_data['quality'])
wine_data['quality']
```

Out[85]:
```
0       0
1       0
2       0
3       0
4       0
       ..
1594    0
1595    0
1596    0
1597    0
1598    0
Name: quality, Length: 1599, dtype: int64
```

In [86]:
```
# Now, you can see values are set in 0 and 1 format
wine_data.head(8) # If there is 3 categorical values are avail, then it will
```

Out[86]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	1

In [87]:
```
wine_data['quality'].value_counts()
```

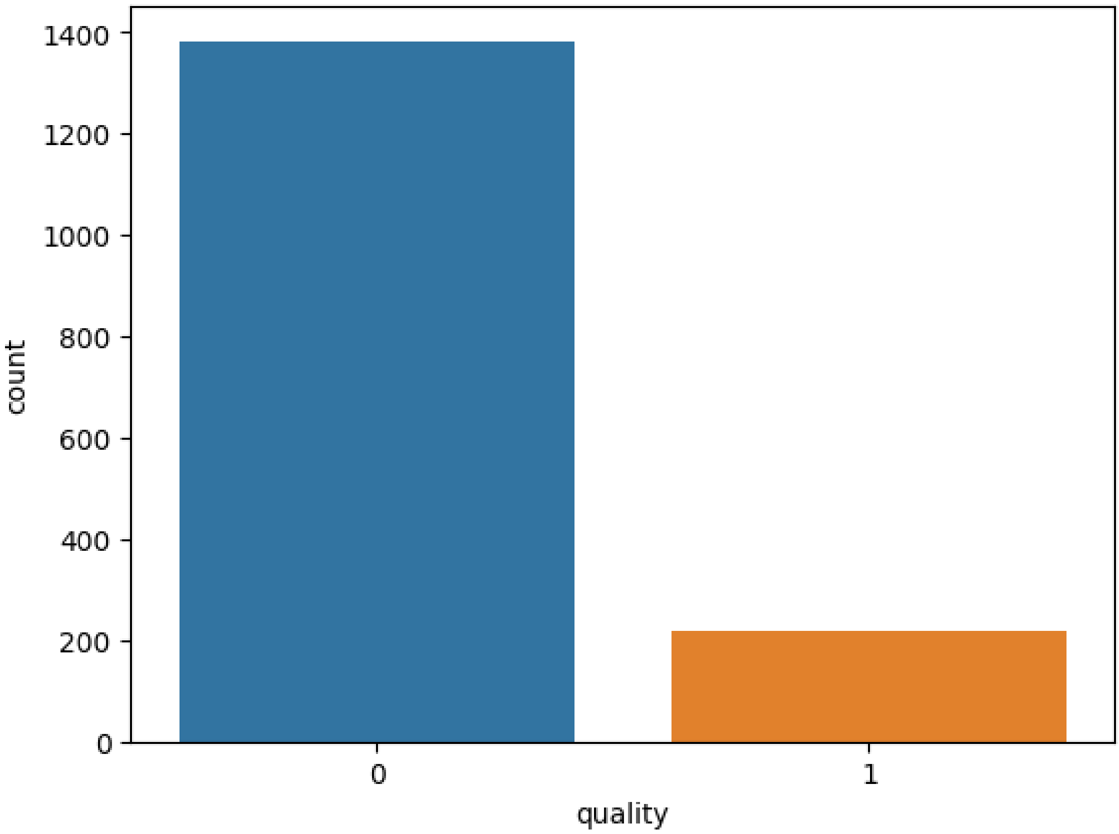
Out[87]:
```
0    1382
1     217
Name: quality, dtype: int64
```

In [88]:
```
wine_data['quality'].dtype
```

Out[88]:
```
dtype('int64')
```

```
In [89]: sns.countplot(x = wine_data['quality'])
```

```
Out[89]: <Axes: xlabel='quality', ylabel='count'>
```



Separating the Dataset

```
In [90]: x = wine_data.drop('quality', axis = 1) # This means you're dropping a colum
         y = wine_data['quality']
```

Spliting the Dataset

```
In [91]: x_train, x_test, y_train, y_test = train_test_split(x, y, # Features and Lab
                                                             test_size=0.2, # 20% of th
                                                             random_state=42) # Ensures
```

Applying Standard Scaling

```
In [92]: wine_scaler = StandardScaler()
         x_train_scaler = wine_scaler.fit_transform(x_train)
         x_test_scaler = wine_scaler.transform(x_test)
```

In [93]:
```
x_train_scaler[:4] # Viewing data of first 4 rows
```

Out[93]:
```
array([[ 0.21833164,  0.88971201,  0.19209222,  0.30972563, -0.04964208,
         0.69100692,  1.04293362,  1.84669643,  1.09349989,  0.45822284,
         1.12317723],
       [-1.29016623, -1.78878251,  0.65275338, -0.80507963, -0.45521361,
         2.38847304,  3.59387025, -3.00449133, -0.40043872, -0.40119696,
         1.40827174],
       [ 1.49475291, -0.78434707,  1.01104539, -0.52637831,  0.59927236,
        -0.95796016, -0.99174203,  0.76865471, -0.07566946,  0.51551749,
        -0.58738978],
       [ 0.27635078,  0.86181102, -0.06383064, -0.66572897, -0.00908493,
         0.01202048, -0.71842739,  0.08948842,  0.05423824, -1.08873281,
        -0.96751578]])
```

In [94]:
```
x_test_scaler[:4] # Viewing data of first 4 rows
```

Out[94]:
```
array([[-3.61859850e-01,  1.64286407e-01, -9.85152962e-01,
        -3.86510130e-02,  5.18158057e-01, -1.81975648e-01,
        -1.99566462e-02,  1.75731759e-01, -4.65392578e-01,
        -1.34389336e-04, -7.77452782e-01],
       [-3.03840702e-01, -1.70525408e-01, -5.24491803e-01,
        -6.65728970e-01, -1.30756387e-01,  4.97010797e-01,
         1.68066777e+00, -4.17191190e-01,  5.08915214e-01,
        -1.03143815e+00, -8.72484283e-01],
       [ 1.37871461e+00,  7.78108067e-01, -2.68568937e-01,
         1.00699644e-01,  3.76208022e-01,  1.09018543e-01,
        -3.84376165e-01,  1.95450060e+00, -2.05577167e-01,
         1.83329452e+00, -4.92358280e-01],
       [ 1.02293339e-01, -3.93733284e-01,  1.92092221e-01,
        -2.12839335e-01, -2.11870693e-01,  1.56398950e+00,
         3.44462872e-01,  6.60850535e-01,  1.19192097e-01,
        -6.87670232e-01, -5.87389780e-01]])
```

Using different model for predictions

Random Forest Classifier

In [95]:
```
wine_rfc = RandomForestClassifier(n_estimators= 200)
wine_rfc.fit (x_train, y_train)

pred_rfc = wine_rfc.predict(x_test)
```

In [96]:
```
# Checking how our model is working
print(classification_report(y_test, pred_rfc))
print(confusion_matrix(y_test, pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       273
           1       0.74      0.53      0.62        47

    accuracy                           0.90       320
   macro avg       0.83      0.75      0.78       320
weighted avg       0.90      0.90      0.90       320

[[264   9]
 [ 22  25]]
```

In [97]:
```
# Finding the accuracy of the model
rfc_acc = accuracy_score(y_test, pred_rfc)
rfc_acc
```

Out[97]:
```
0.903125
```

SVM Classifier

In [98]:
```
wine_svm = svm.SVC()
wine_svm.fit (x_train, y_train)

pred_svm = wine_svm.predict(x_test)
```

In [99]:
```
# Checking how our model is working
print(classification_report(y_test, pred_rfc))
print(confusion_matrix(y_test, pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       273
           1       0.74      0.53      0.62        47

    accuracy                           0.90       320
   macro avg       0.83      0.75      0.78       320
weighted avg       0.90      0.90      0.90       320

[[273   0]
 [ 46   1]]
```

In [100]:
```
# Finding the accuracy of the model
svm_acc = accuracy_score(y_test, pred_svm)
svm_acc
```

Out[100]:
```
0.85625
```

Neural Network

In [101]:
```
wine_mlpc = MLPClassifier(hidden_layer_sizes = (11, 11, 11), #  MLP model hc
                          max_iter = 500) # The solver will go through a max
''' If the solver does not converge within 500 iterations, it will stop and
but it could indicate that the model could benefit from more iterations, or
wine_mlpc.fit (x_train, y_train)

pred_mlpc = wine_mlpc.predict(x_test)
```

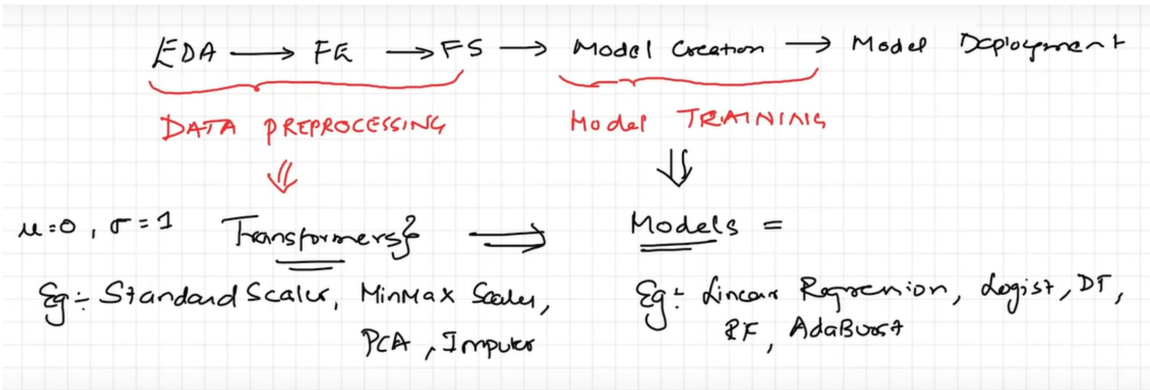
In [102]:
```
# Checking how our model is working
print(classification_report(y_test, pred_mlpc))
print(confusion_matrix(y_test, pred_mlpc))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       273
           1       0.52      0.28      0.36        47

    accuracy                           0.86       320
   macro avg       0.70      0.62      0.64       320
weighted avg       0.83      0.86      0.84       320

[[261  12]
 [ 34  13]]
```

In [103]:
```
# Finding the accuracy of the model
mlpc_acc = accuracy_score(y_test, pred_mlpc)
mlpc_acc
```

Out[103]:
```
0.85625
```

Difference between fit(), transform(), fit_transform(), predict() methods

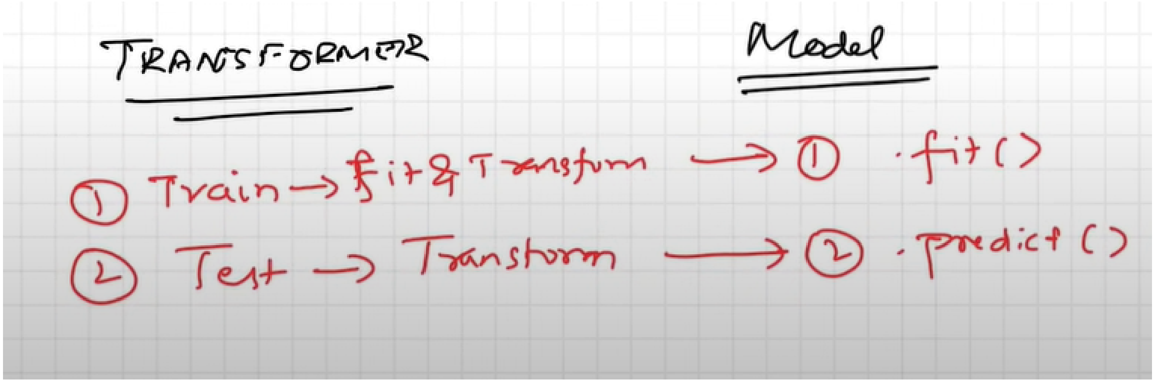


Transformer: A Transformer is used for transforming datasets. It provides a library of transformers that can clean, reduce, expand, or generate feature representations. A transformer is used for preprocessing and transforming the input data. It changes the format of the training dataset, which is then used for model training.

Transformers implement a `fit()` method to learn some characteristics of the data, and a `transform()` method to apply this transformation to any dataset. Examples of transformers include StandardScaler, PCA, Imputer, MinMaxScaler, etc

Model: A model (or estimator), is used to make predictions. It uses learning algorithms like linear regression, logistic regression, KNN, etc, to predict a new value (or values) by using the input data.

Models implement a `fit()` method to learn from the data and a `predict()` method to make predictions on unseen data1.



1. **fit():** This method is used to compute the necessary parameters from the training data. For example, in the case of a transformer like StandardScaler, it calculates parameters like mean and standard deviation. For a model (estimator), it calculates the weights on the training data.
2. **transform():** This method applies the transformation on a dataset using the parameters computed during the fit process. For example, in StandardScaler, it uses the calculated mean and standard deviation to standardize the data.
3. **fit_transform():** This method is a combination of fit() and transform(). It fits the data, then transforms it. This is often more efficient than calling fit() and transform() separately. It's mainly used on the training data.
4. **predict():** This method is used in models (estimators) to generate predictions. It uses the model parameters learned during the fit() process to predict the target variable for unseen data.

In [104]:
```
# .Fit Scenarios
'''1. At time of scaling: (scaler.fit_transform(xtrain) and scaler.transform
2. At models training: (model.fit(xtrain)) there we  use fit to fetch the pa
```

Out[104]:
```
'1. At time of scaling: (scaler.fit_transform(xtrain) and scaler.transform
(xtest) that is part of Data preprocessing step\n2. At models training: (m
odel.fit(xtrain)) there we  use fit to fetch the parameters like slope and
y intercept'
```

Implementing this concepts with Standardization

Standardization: We try to bring all the variables or features to a similar scale, standardization means centering the variable at zero.

z = (x - x_mean)/std

Importing Dataset

In [105]:
```
import pandas as pd
df_titan = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/5._Titanic
                       usecols = ['Pclass', 'Age', 'Fare', 'Survived'])

df_titan.head(4)
```

Out[105]:

	Survived	Pclass	Age	Fare
0	0	3	22.0	7.2500
1	1	1	38.0	71.2833
2	1	3	26.0	7.9250
3	1	1	35.0	53.1000

In [106]:
```
df_titan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Age       714 non-null    float64
 3   Fare      891 non-null    float64
dtypes: float64(2), int64(2)
memory usage: 28.0 KB
```

Data Manipulation

In [107]:
```
df_titan.isnull().sum()
```

Out[107]:
```
Survived      0
Pclass        0
Age         177
Fare          0
dtype: int64
```

In [108]:
```
df_titan['Age'].fillna(df_titan.Age.median(), inplace = True)
df_titan.isnull().sum()
```

Out[108]:
```
Survived    0
Pclass      0
Age         0
Fare        0
dtype: int64
```

Separating the dataset

In [109]:
```
x = df_titan.drop('Survived', axis = 1)
x
```

Out[109]:

	Pclass	Age	Fare
0	3	22.0	7.2500
1	1	38.0	71.2833
2	3	26.0	7.9250
3	1	35.0	53.1000
4	3	35.0	8.0500
...	...	...	...
886	2	27.0	13.0000
887	1	19.0	30.0000
888	3	28.0	23.4500
889	1	26.0	30.0000
890	3	32.0	7.7500

891 rows × 3 columns

In [110]:
```
y = df_titan['Survived']
y
```

Out[110]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Spliting the Dataset

In [111]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, # Features and Lab
                                                     test_size=0.2, # 20% of th
                                                     random_state=42) # Ensures
```

In [112]:
```
# Viewing the dataframe
x_train.head(4)
```

Out[112]:

	Pclass	Age	Fare
331	1	45.5	28.5000
733	2	23.0	13.0000
382	3	32.0	7.9250
704	3	26.0	7.8542

In [113]:
```
x_test.head(4)
```

Out[113]:

	Pclass	Age	Fare
709	3	28.0	15.2458
439	2	31.0	10.5000
840	3	20.0	7.9250
720	2	6.0	33.0000

In [114]:
```
# Viewing the series
y_train.head(4)
```

Out[114]:
```
331    0
733    0
382    0
704    0
Name: Survived, dtype: int64
```

In [115]:
```
y_test.head(4)
```

Out[115]:
```
709    1
439    0
840    0
720    1
Name: Survived, dtype: int64
```

Implementing Standard Scalar for Scaling

In [116]:
```
from sklearn.preprocessing import StandardScaler
```

In [117]:
```
# x_train is for training data
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_train_scaled
```

Out[117]:
```
array([[-1.61413602,  1.25364106, -0.07868358],
       [-0.40055118, -0.47728355, -0.37714494],
       [ 0.81303367,  0.21508629, -0.47486697],
       ...,
       [ 0.81303367,  0.90745614, -0.35580399],
       [-1.61413602, -1.1696534 ,  1.68320121],
       [-1.61413602, -0.63114352,  0.86074761]])
```

In [118]:
```
# Now you can transform your test data
x_test_scaled = scaler.transform(x_test)
x_test_scaled
```

Out[118]:
```
array([[ 8.13033667e-01, -9.26336398e-02, -3.33900778e-01],
       [-4.00551178e-01,  1.38156309e-01, -4.25283869e-01],
       [ 8.13033667e-01, -7.08073503e-01, -4.74866965e-01],
       [-4.00551178e-01, -1.78509326e+00,  7.96648968e-03],
       [ 8.13033667e-01, -1.16965340e+00, -4.11002011e-01],
       [-1.61413602e+00, -2.46493606e-01,  8.90834443e-01],
       [ 8.13033667e-01, -9.26336398e-02, -4.78236690e-01],
       [ 8.13033667e-01, -1.01579343e+00, -2.80867083e-01],
       [ 8.13033667e-01, -1.01579343e+00, -4.78236690e-01],
       [-1.61413602e+00, -7.85003486e-01, -1.21367407e-01],
       [-1.61413602e+00,  5.99736206e-01,  3.95003477e-01],
       [ 8.13033667e-01,  1.13824609e+00, -4.72460019e-01],
       [ 8.13033667e-01, -9.26336398e-02, -1.37091507e-01],
       [ 8.13033667e-01,  6.12263260e-02, -4.88345865e-01],
       [-4.00551178e-01,  5.22806223e-01, -3.77144940e-01],
       [-1.61413602e+00, -1.01579343e+00,  1.31202147e-01],
       [-1.61413602e+00,  9.84386120e-01,  3.84493786e-01],
       [ 8.13033667e-01, -9.26336398e-02, -4.76711649e-01],
       [-4.00551178e-01, -1.69563623e-01, -3.77144940e-01],
```

Model Building (Through Logistic Regression)

In [119]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [120]:
```
# Fit the model
fit_titan = model.fit(x_train_scaled, y_train)
fit_titan
```

Out[120]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [121]:
```
# Predict the model
pred_titan = model.predict(x_test_scaled)
pred_titan
```

Out[121]:
```
array([0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0])
```

In [122]:
```
# Checking how our model is working
print(classification_report(y_test, pred_titan))
print(confusion_matrix(y_test, pred_titan))
```

```
              precision    recall  f1-score   support

           0       0.73      0.89      0.80       105
           1       0.76      0.53      0.62        74

    accuracy                           0.74       179
   macro avg       0.75      0.71      0.71       179
weighted avg       0.74      0.74      0.73       179

[[93 12]
 [35 39]]
```

Predict the values in Binary Classification

In [123]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Define the data

In [124]:
```
data = {
    'Blood glucose': [67, 103, 114, 72, 116, 65],
    'Diabetic': [0, 1, 1, 0, 1, 0]
}

# Create the dataframe
df = pd.DataFrame(data)

# Print the dataframe
print(df)
```

```
   Blood glucose  Diabetic
0             67         0
1            103         1
2            114         1
3             72         0
4            116         1
5             65         0
```

Separating the dataset

In [125]:
```
x = df[['Blood glucose']]
x
```

Out[125]:

	Blood glucose
0	67
1	103
2	114
3	72
4	116
5	65

In [126]:
```
y = df['Diabetic']
y
```

Out[126]:
```
0    0
1    1
2    1
3    0
4    1
5    0
Name: Diabetic, dtype: int64
```

Spliting the dataset

In [127]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, rar
```

Train the model

In [128]:
```
# Create a Logistic Regression object
log_reg = LogisticRegression()

# Train the model
log_reg.fit(x_train, y_train)

# Now you can use log_reg.predict(X_test) to make predictions
```

Out[128]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating a Binary Classification Model

In [129]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

Define the data

In [130]:
```
new_data = {
    'Blood glucose': [66, 107, 112, 71, 87, 89],
    'Diabetic': [0, 1, 1, 0, 1, 1]
}

# Create a DataFrame from the new data
new_df = pd.DataFrame(new_data)
```

Separating the dataset

In [131]:
```
# Separate features and target
X_new = new_df[['Blood glucose']]
y_new = new_df['Diabetic']
```

Using trained model to make predictions

In [132]:
```
y_pred = log_reg.predict(X_new)

# Print the classification report
print(classification_report(y_new, y_pred))

# Print the confusion matrix
print(confusion_matrix(y_new, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         2
           1       1.00      0.50      0.67         4

    accuracy                           0.67         6
   macro avg       0.75      0.75      0.67         6
weighted avg       0.83      0.67      0.67         6

[[2 0]
 [2 2]]
```

Printing the predicted value

In [133]:
```
# Add the predictions to the new_df DataFrame
new_df['Predicted (ŷ)'] = y_pred

# Print the DataFrame
print(new_df)
```

```
   Blood glucose  Diabetic  Predicted (ŷ)
0             66         0              0
1            107         1              1
2            112         1              1
3             71         0              0
4             87         1              0
5             89         1              0
```

Printing the predicted value with user input

In [134]:
```
diabetic_var = int(input("Enter the Blood Sugar level to see, you have diabe

predict_diabetic = log_reg.predict([[diabetic_var]])

print(predict_diabetic) #Reason of warning: supply a non-default value for t
```

```
Enter the Blood Sugar level to see, you have diabetic or not: 120
[1]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
X does not have valid feature names, but LogisticRegression was fitted wit
h feature names
  warnings.warn(
```

Predict the values in Regression

In [135]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Define the data

In [136]:
```
data = {
    'Temperature': [51, 52, 67, 65, 70, 69, 72, 75, 73, 81, 78, 83],
    'Ice cream sales': [1, 0, 14, 14, 23, 20, 23, 26, 22, 30, 26, 36]
}

# Create the dataframe
df = pd.DataFrame(data)

# Print the dataframe
print(df)
```

```
    Temperature  Ice cream sales
0            51                1
1            52                0
2            67               14
3            65               14
4            70               23
5            69               20
6            72               23
7            75               26
8            73               22
9            81               30
10           78               26
11           83               36
```

Separating the dataset

In [137]:
```
X = df[['Temperature']]
y = df['Ice cream sales']
```

Spliting the dataset

In [138]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

Train the model

In [139]:
```
# Create a Linear Regression object
lin_reg = LinearRegression()

# Train the model
lin_reg.fit(X_train, y_train)

# Now you can use lin_reg.predict(X_test) to make predictions
```

Out[139]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating a Regression Model

In [140]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

Define the data

In [141]:
```
new_data = {
    'Temperature': [52, 67, 70, 73, 78, 83],
    'Ice cream sales': [0, 14, 23, 22, 26, 36]
}

# Create a DataFrame from the new data
new_df = pd.DataFrame(new_data)
```

Separating the dataset

In [142]:
```
# Separate features and target
X_new = new_df[['Temperature']]
y_new = new_df['Ice cream sales']
```

Using trained model to make predictions

In [143]:
```
y_pred = lin_reg.predict(X_new)

# Calculate and print the MSE, RMSE, and R² score
mse = mean_squared_error(y_new, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_new, y_pred)

print('Mean Squared Error (MSE):', mse)
print('Root Mean Squared Error (RMSE):', rmse)
print('R-squared (R²) score:', r2)
```

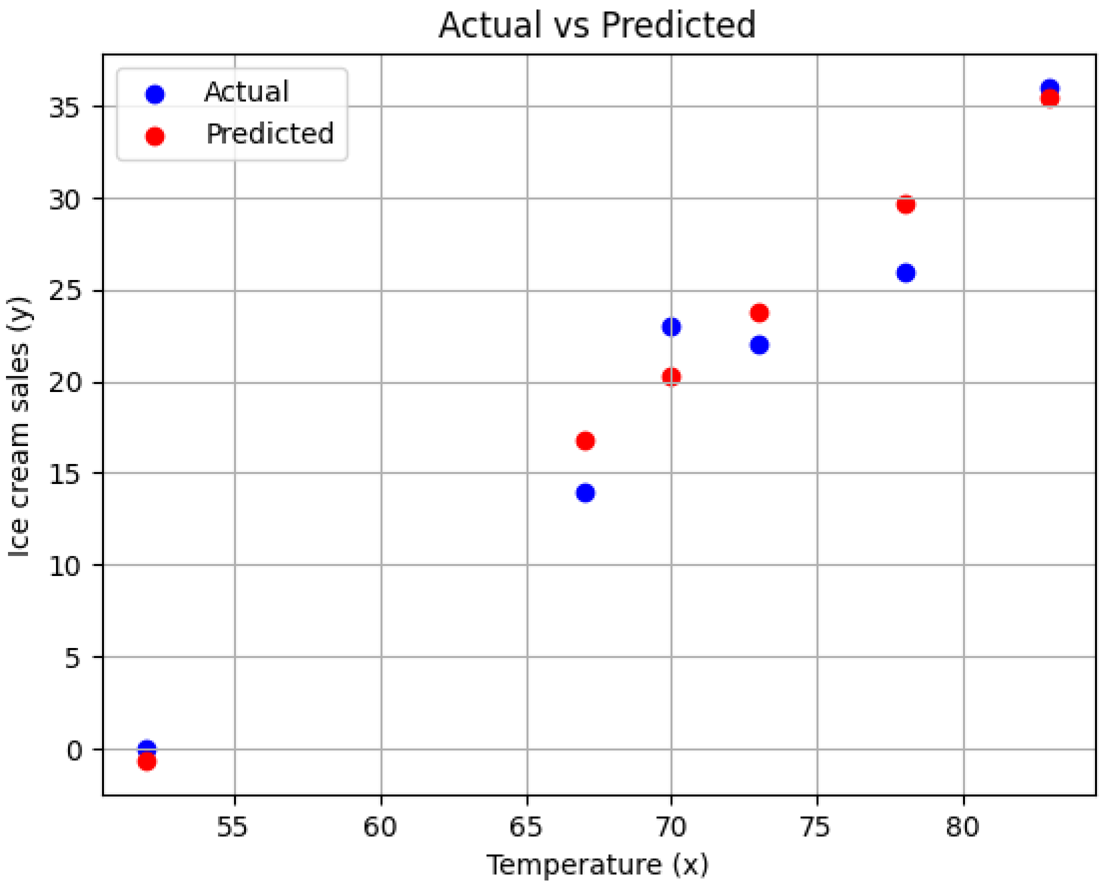
```
Mean Squared Error (MSE): 5.363223460633513
Root Mean Squared Error (RMSE): 2.315863437388637
R-squared (R²) score: 0.9565633195539243
```

Printing the predicted value

In [144]:
```
# Add the predictions to the new_df DataFrame
new_df['Predicted (ŷ)'] = y_pred

# Print the DataFrame
print(new_df)
```

```
   Temperature  Ice cream sales  Predicted (ŷ)
0           52                0      -0.667980
1           67               14      16.801497
2           70               23      20.295392
3           73               22      23.789287
4           78               26      29.612446
5           83               36      35.435605
```

In [145]:
```
# Plot the actual data
plt.scatter(X_new, y_new, color='blue', label='Actual')

# Plot the predicted data
plt.scatter(X_new, y_pred, color='red', label='Predicted')

# Set the labels and title
plt.xlabel('Temperature (x)')
plt.ylabel('Ice cream sales (y)')
plt.title('Actual vs Predicted')
plt.legend()
plt.grid()

# Show the plot
plt.show()
```



Printing the predicted value with user input

In [146]:
```
temp_var = int(input("Enter the Temperature to predict the ice-cream sales:

predict_sales = lin_reg.predict([[temp_var]])

print(predict_sales) #Reason of warning: supply a non-default value for the
```

```
Enter the Temperature to predict the ice-cream sales: 80
[31.94170933]

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with
feature names
  warnings.warn(
```