DATA MINING AND MACHINE LEARNING

1) **Regression (Concrete Compressive Strength)**

Dataset Link: https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength

1.1)Business Understanding

Concrete quality plays a vital role in construction and on-site mixing often leads to inconsistent strength. This project aims to use a regression machine learning model to predict concrete's compressive strength, helping structural engineers simulate building strength by adjusting material proportions. An accurate model will make it easier to predict costs and benefits, optimizing the execution of customer orders.

Attribute information

Cement	Quantity of cement used in the concrete mixture
Blast Furnace Slag	Quantity of blast furnace slag incorporated into the concrete mix
Fly Ash	Quantity of fly ash included in the concrete mix
Water	Volume of water present in the concrete mixture
Superplasticizer	Amount of superplasticizer added to the concrete mix
Coarse Aggregate	Amount of coarse aggregate (such as gravel or crushed stone) in the mix
Fine Aggregate	Amount of fine aggregate (sand) in the concrete mixture
Age	Duration (in days) of concrete curing when compressive strength is assessed
Concrete Compressive Strength (Target)	Desired compressive strength of the concrete (dependent variable)

1.2)Data Understanding & Preparation:

First, we import the necessary dependencies for performing linear regression analysis. Next, we load and explore the dataset, then proceed to build and evaluate the linear regression model. Afterward, we conduct polynomial regression analysis and compare the accuracy of both models.

Read the Data

[21]:
```
df = pd.read_csv('data/Concrete_Data.csv')
df.head()    #check if data is loaded successfully
```

[21]:

	Cement	Blast_Furnace_Slag	Fly_Ash	Water	Super_Plastic	Coarse_Aggregate	Fine_Aggregate	Age	Concrete_compressive_strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

Explore your data

[22]:
```
df.describe()
```

[22]:

	Cement	Blast_Furnace_Slag	Fly_Ash	Water	Super_Plastic	Coarse_Aggregate	Fine_Aggregate	Age	Concrete_compressive_strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

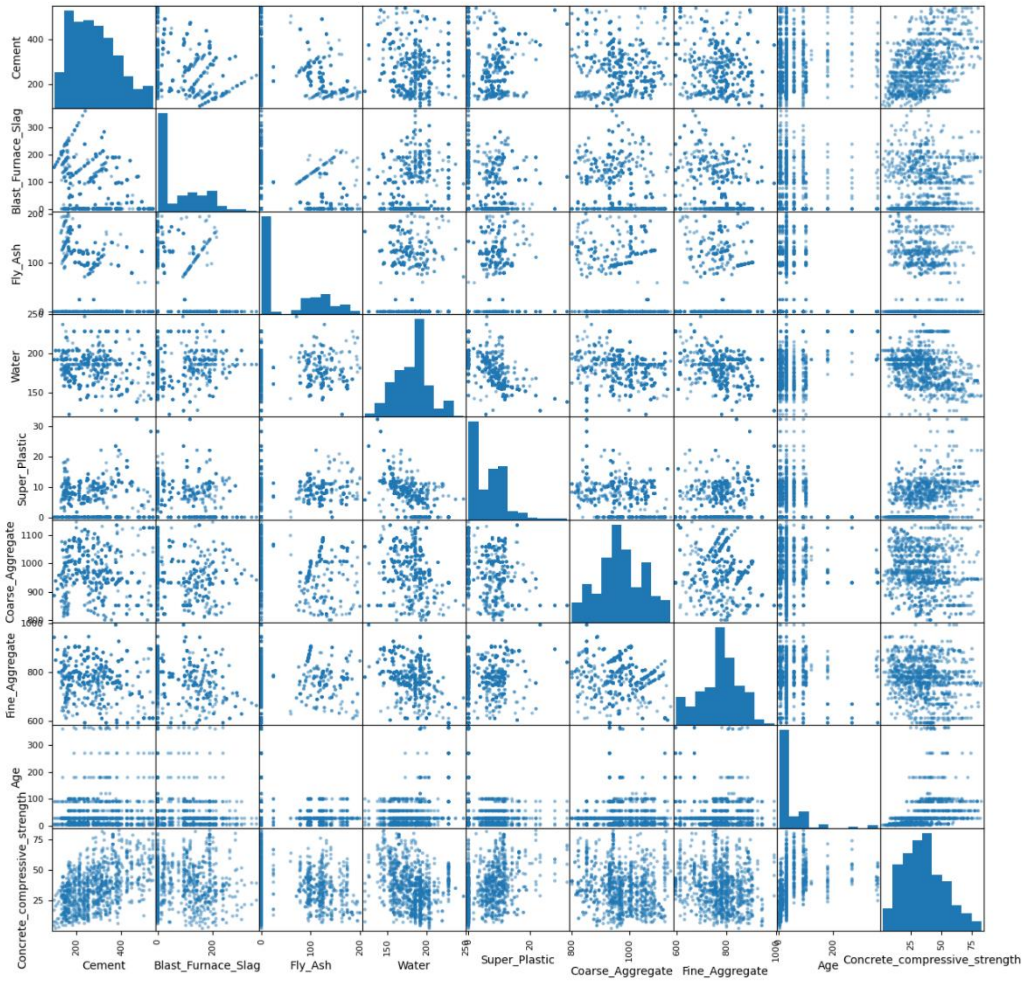
Correlation Matrix: The correlation matrix reveals the relationships between variables, indicating that some features have stronger correlations with the target variable, while others show weaker or negligible correlations. This suggests that not all features are equally important in building an accurate model.

[11]:
```
# It will show relationship between each feature
df.corr()
```

[11]:

	Cement	Blast_Furnace_Slag	Fly_Ash	Water	Super_Plastic	Coarse_Aggregate	Fine_Aggregate	Age	Concrete_compressive_strength
Cement	1.000000	-0.275216	-0.397467	-0.081587	0.092386	-0.109349	-0.222718	0.081946	0.497832
Blast_Furnace_Slag	-0.275216	1.000000	-0.323580	0.107252	0.043270	-0.283999	-0.281603	-0.044246	0.134829
Fly_Ash	-0.397467	-0.323580	1.000000	-0.256984	0.377503	-0.009961	0.079108	-0.154371	-0.105755
Water	-0.081587	0.107252	-0.256984	1.000000	-0.657533	-0.182294	-0.450661	0.277618	-0.289633
Super_Plastic	0.092386	0.043270	0.377503	-0.657533	1.000000	-0.265999	0.222691	-0.192700	0.366079
Coarse_Aggregate	-0.109349	-0.283999	-0.009961	-0.182294	-0.265999	1.000000	-0.178481	-0.003016	-0.164935
Fine_Aggregate	-0.222718	-0.281603	0.079108	-0.450661	0.222691	-0.178481	1.000000	-0.156095	-0.167241
Age	0.081946	-0.044246	-0.154371	0.277618	-0.192700	-0.003016	-0.156095	1.000000	0.328873
te_compressive_strength	0.497832	0.134829	-0.105755	-0.289633	0.366079	-0.164935	-0.167241	0.328873	1.000000

Scatter_matrix exploration: Scatter matrix of Concrete Compressive Strength dataset.



Data Splitting: Splitting the dataset, assigning the target features (Concrete_compressive_strength) to y and other features to X.

```
[38]: X = df.drop(['Concrete_compressive_strength'], axis=1)
      y = df.Concrete_compressive_strength
      X.head()
```

[38]:

	Cement	Blast_Furnace_Slag	Fly_Ash	Water	Super_Plastic	Coarse_Aggregate	Fine_Aggregate	Age
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360

1.3) Modeling:

Split the dataset into training and test data for building the linear regression model and polynomial regression model.

*** Linear regression model building**.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
intercept = model.intercept_
coefficient = model.coef_
print(X.columns)
print('intercept:', intercept, 'coefficient:', coefficient)
```

```
Index(['Cement', 'Blast_Furnace_Slag', 'Fly_Ash', 'Water', 'Super_Plastic',
       'Coarse_Aggregate', 'Fine_Aggregate', 'Age'],
      dtype='object')
intercept: -13.356302642850139 coefficient: [ 0.12198785  0.10524275  0.08729552 -0.15478128  0.33176191  0.01258243
  0.01436308  0.11555199]
```

*** Polynomial regression model building.**

```
#'Cement',  'Blast_Furnace_Slag','Fly_Ash',  'Fine_Aggregate',
X['Cement_Square'] = np.square(df.Cement)
X['Blast_Furnace_Slag_Square'] = np.square(df.Blast_Furnace_Slag)
X['Fly_Ash_Square'] = np.square(df.Fly_Ash)
X['Fine_Aggregate_Square'] = np.square(df.Fine_Aggregate)

X['Superplastic_Square'] = np.square(df.Super_Plastic)
X['Coarse_Aggregate_Square'] = np.square(df.Coarse_Aggregate)
X['Age_Square'] = np.square(df.Age)
X['Water_Square'] = np.square(df.Water)
```

```
model = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

1.4) Evaluation: Based on the code snippets, we are comparing the accuracy of both linear regression model and polynomial regression model, we can see that the accuracy for linear regression model is 54% and the accuracy for polynomial regression model is 77%.

*** Linear Regression Model Accuracy**

```
r_squared = model.score(X_test, y_test)
print('R-squared of test data=', r_squared)
```

```
R-squared of test data= 0.5414805238935221
```

88% of the features explained/predicted the amount of water, so our model is performing well

```
rmse = mean_squared_error(y_test, yhat, squared = False)
print('RMSE:', rmse)
```

```
RMSE: 10.962721175664734
```

*** Polynomial regression Model Accuracy**

```python
yhat = model.predict(X_test)

print('Polynomial R squared:', model.score(X,y))
print('Polynomial RMSE', mean_squared_error(y_test, yhat, squared=False))
```

```
Polynomial R squared: 0.7721252824632377
Polynomial RMSE 8.393929460353913
```

The polynomial dataset has boosted and improved our model performance from 54% to 77%, and dropped error, hereby improving model performance on unseen data

2) DECISION TREE (Rice dataset)

Dataset Link: https://archive.ics.uci.edu/dataset/545/rice+cammeo+and+osmancik

2.1) Business Understanding

The project focuses on creating a predictive model to classify rice grain species based on morphological features using machine learning. This classification helps improve the efficiency and accuracy of rice sorting processes in the agricultural industry. By automating the identification of different two Turkish rice varieties Osmancik and Cammeo, businesses can streamline quality control, reduce human error, and enhance the overall productivity of rice processing, ultimately leading to cost savings and higher quality products for consumers.

Attribute information

Area	Total pixels inside the rice grain.
Perimeter	Distance around the rice grain.
Major_Axis_Length	Longest line through the rice grain.
Minor_Axis_Length	Shortest line through the rice grain
Eccentricity	How round or oval the rice grain is.
Convex_Area	Smallest enclosing shape around the rice grain.
Extent	Ratio of rice grain area to bounding box area.
Class (Target)	Type of rice grain: Cammeo or Osmancik.

2.2) Data Understanding & Preparation

First, we import the necessary dependencies for performing linear regression analysis. Next, we load and explore the dataset, then proceed to build and evaluate the linear regression model. Afterward, we conduct polynomial regression analysis and compare the accuracy of both models.

```
df = pd.read_csv('data/Rice_Data.csv')
df.head()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	Class
0	15231	525.578979	229.749878	85.093788	0.928882	15617	0.572896	Cammeo
1	14656	494.311005	206.020065	91.730972	0.895405	15072	0.615436	Cammeo
2	14634	501.122009	214.106781	87.768288	0.912118	14954	0.693259	Cammeo
3	13176	458.342987	193.337387	87.448395	0.891861	13368	0.640669	Cammeo
4	14688	507.166992	211.743378	89.312454	0.906691	15262	0.646024	Cammeo

The Rice dataset contains 3800 instances and 8 features.

```
df.describe()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent
count	3810.000000	3810.000000	3810.000000	3810.000000	3810.000000	3810.000000	3810.000000
mean	12667.727559	454.239180	188.776222	86.313750	0.886871	12952.496850	0.661934
std	1732.367706	35.597081	17.448679	5.729817	0.020818	1776.972042	0.077239
min	7551.000000	359.100006	145.264465	59.532406	0.777233	7723.000000	0.497413
25%	11370.500000	426.144752	174.353855	82.731695	0.872402	11626.250000	0.598862
50%	12421.500000	448.852493	185.810059	86.434647	0.889050	12706.500000	0.645361
75%	13950.000000	483.683746	203.550438	90.143677	0.902588	14284.000000	0.726562
max	18913.000000	548.445984	239.010498	107.542450	0.948007	19099.000000	0.861050

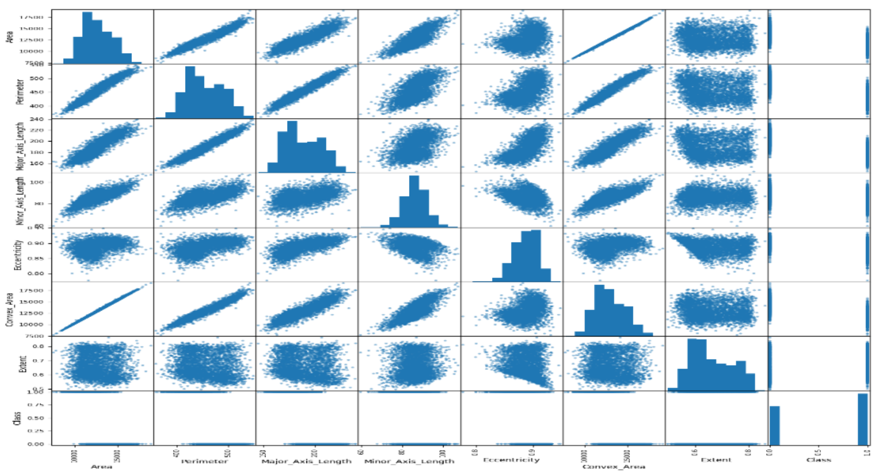
Correlation Matrix: The correlation matrix reveals the relationships between variables, indicating that some features have stronger correlations with the target variable, while others show weaker or negligible correlations. This suggests that not all features are equally important in building an accurate model.

Correlation matrix for the features

```
X.corr()
```

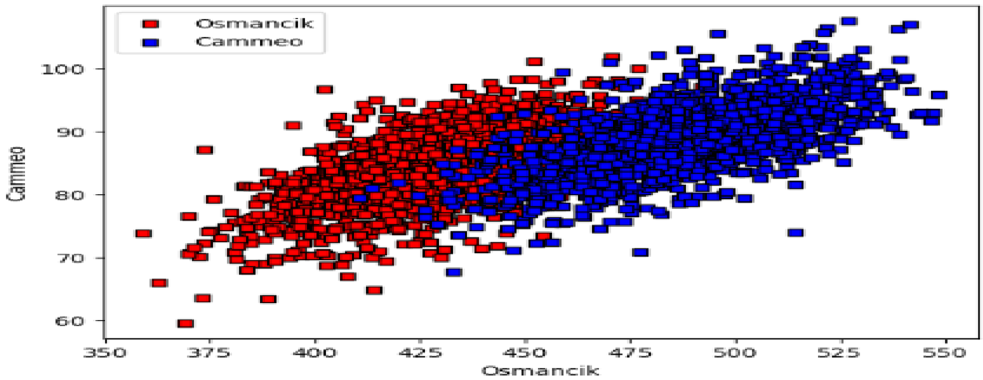
	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent
Area	1.000000	0.966453	0.903015	0.787840	0.352095	0.998939	-0.061184
Perimeter	0.966453	1.000000	0.971884	0.629828	0.544601	0.969937	-0.130923
Major_Axis_Length	0.903015	0.971884	1.000000	0.452092	0.710897	0.903381	-0.139562
Minor_Axis_Length	0.787840	0.629828	0.452092	1.000000	-0.291683	0.787318	0.063366
Eccentricity	0.352095	0.544601	0.710897	-0.291683	1.000000	0.352716	-0.198580
Convex_Area	0.998939	0.969937	0.903381	0.787318	0.352716	1.000000	-0.065826
Extent	-0.061184	-0.130923	-0.139562	0.063366	-0.198580	-0.065826	1.000000

Scatter_matrix exploration: Scatter matrix of Rice dataset with all features



Scatter matrix of Area and Minor_Axis_Length features

```python
d = df.to_numpy()

plt.scatter(
    d[d[:,7] == "Osmancik", 1], d[d[:,7] == "Osmancik", 3],
    c='red', marker='s', edgecolor='black',
    label='Osmancik'
)


plt.scatter(
    d[d[:,7] == "Cammeo", 1], d[d[:,7] == "Cammeo", 3],
    c='blue', marker='s', edgecolor='black',
    label='Cammeo'
)

plt.xlabel('Osmancik')
plt.ylabel('Cammeo')
plt.legend(loc="upper left")

plt.show()
```



Data Splitting: Splitting the dataset and assigning the target feature (Class) to y and other features to X.

Split the features (X) and target (y) variables

```python
X = df.drop('Class', axis = 1)
y = df['Class']
X.head()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent
0	15231	525.578979	229.749878	85.093788	0.928882	15617	0.572896
1	14656	494.311005	206.020065	91.730972	0.895405	15072	0.615436
2	14634	501.122009	214.106781	87.768288	0.912118	14954	0.693259
3	13176	458.342987	193.337387	87.448395	0.891861	13368	0.640669
4	14688	507.166992	211.743378	89.312454	0.906691	15262	0.646024

2.3) Modeling:

The target variable (Class) is assigned to y, while the remaining feature variables are assigned to X. The data is then split into training and test sets, and cross validation is performed to determine the optimal maximum depth for both models and assess their accuracy.

Split data into training and testing sets (70% training, 30% testing)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(2667, 7)
(2667,)
(1143, 7)
(1143,)
```

Perform cross-validation to determine the best tree depth

```
for d in range(2,10) :
    model = DecisionTreeClassifier(max_depth=d,random_state = 42)
    scores = cross_val_score(model, X_train, y_train, cv=5)
    print("d: ", d, " val accuracy: ", scores.mean())
```

```
d:  2  val accuracy:  0.9257625903830344
d:  3  val accuracy:  0.9253873558614585
d:  4  val accuracy:  0.9171328990731566
d:  5  val accuracy:  0.9141324282732889
d:  6  val accuracy:  0.9126300848142449
d:  7  val accuracy:  0.903639915396561
d:  8  val accuracy:  0.9017602293568313
d:  9  val accuracy:  0.8980099922001813
```

Calculated the training and test accuracy

```
model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)

print("Tree Depth:", model.get_depth())
print("Training Accuracy: ", model.score(X_train, y_train))
print("Test Accuracy: ", model.score(X_test, y_test))
```

```
Tree Depth: 4
Training Accuracy:  0.9347581552305961
Test Accuracy:  0.9221347331583553
```

The model without cross-validation (fixed depth of 4) shows good performance with a high training accuracy (93.48%) and a strong test accuracy (92.21%).

2.4) Evaluation

Generate confusion matrix to evaluate model performance

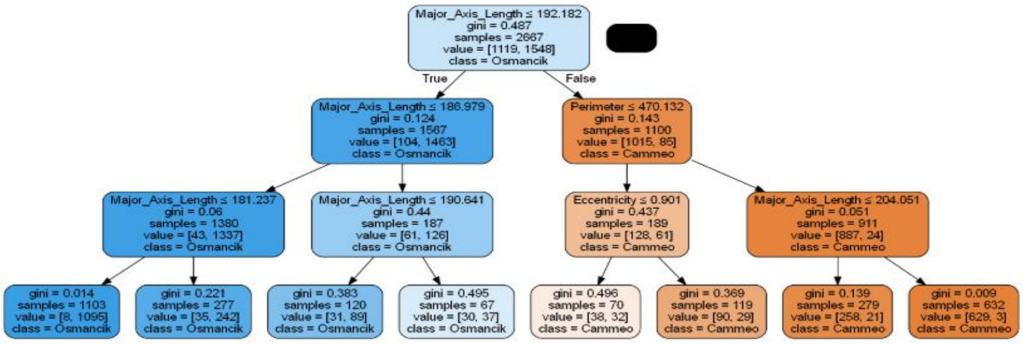
```
cm = confusion_matrix(y_test, y_hat)

print("CM", cm)
print()

tn, fp, fn, tp = cm.ravel()
print("TN", tn, "FP", fp, "FN", fn, "TP", tp)
```

```
CM [[460  51]
 [ 32 600]]

TN 460 FP 51 FN 32 TP 600
```

Decision Tree: After using cross validation, the max_depth value is 2 has the maximum accuracy, but for clear tree image, I have chosen max_depth=3 for better understanding.

```
[78]: feature_names = list(X.columns)
      target_names = list(le.classes_)

      dot_data = StringIO()
      export_graphviz(model, out_file=dot_data, filled=True, rounded=True, special_characters=True, feature_names=feature_names, class_names=target_names)
      graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
      graph.write_png('plots/Rice.png')
      Image(graph.create_png())
```

[78]:



Note: Using cross validation we can see that for depth=2 has the maximum accuracy,but for clear image I chose max_depth=3.

3. K Nearest Neighbor (Rice (Cammeo and Osmancik))

3.1) Business Understanding: Please refer **2.1** Decision Tree data set **for business understanding to avoid replication**.

3.2) Data Understanding & Preparation: First, we import the required dependencies to implement kNN. Then, we load and examine the dataset. Using cross-validation, we identify the best parameter value and apply the kNN classifier to construct and assess the model. Next, we scale the data with a Min-Max scaler, rebuild the kNN model, and evaluate its performance.

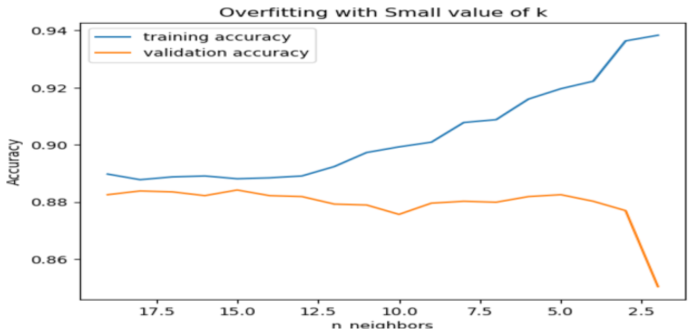
Find training and validation accuracy for range(2,20)

```
training_accuracy = []
validation_accuracy = []

for k in range(2,20) :
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train, y_train)
    training_accuracy.append(clf.score(X_train, y_train))
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    print("k: ", k , "   validation accuracy", scores.mean())
    validation_accuracy.append(scores.mean())
```

```
k:  2    validation accuracy 0.8503948962287007
k:  3    validation accuracy 0.8769711163153786
k:  4    validation accuracy 0.8802503432124688
k:  5    validation accuracy 0.8825465019246816
k:  6    validation accuracy 0.8818891491022638
k:  7    validation accuracy 0.8799203208700099
k:  8    validation accuracy 0.8802498048399687
k:  9    validation accuracy 0.8795929903900509
k:  10    validation accuracy 0.8756558722980431
k:  11    validation accuracy 0.8789377910576329
k:  12    validation accuracy 0.8792656599100919
k:  13    validation accuracy 0.8818886107297639
k:  14    validation accuracy 0.882215941209723
k:  15    validation accuracy 0.8841847694419769
k:  16    validation accuracy 0.882215402837223
```

Plot training and Validation Accuracy

```
plt.plot(range(2,20), training_accuracy , label="training accuracy")
plt.plot(range(2,20), validation_accuracy, label="validation accuracy")
plt.xlabel("n_neighbors")
plt.ylabel("Accuracy")
plt.legend()
plt.title('Overfitting with Small value of k')
ax = plt.gca()
ax.invert_xaxis()
plt.savefig('plots/kNNoverfitting.png')
```



3.3) Modeling:

The target variable is assigned to y, and the remaining features are assigned to X. The dataset is then split into training and test sets. Cross validation is performed using (cross_val_score) to determine the optimal model depth and assess its accuracy.

Model building without Scaling

Model Building before Scaling

```
clf = KNeighborsClassifier(n_neighbors=15)
clf.fit(X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=15)
```

Accuracy Value

Test Accuracy and Training Accuracy ¶

```
print('Test Accuracy: ', clf.score(X_test, y_test))
print('Train Accuracy: ', clf.score(X_train, y_train))
```

```
Test Accuracy:  0.8792650918635171
Train Accuracy:  0.8881233595800525
```

Model Building with scaled data

Min-MaxScalar

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = KNeighborsClassifier()
clf.fit(X_train_scaled,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

Accuracy Value

```
print('Test Accuracy: ', clf.score(X_test_scaled, y_test))
print('Train Accuracy: ', clf.score(X_train_scaled, y_train))
scores = cross_val_score(clf, X_train_scaled, y_train, cv=5)
print('Scores: ',scores)
print('Validation accuracy with MinMaxScaler', scores.mean())
```

```
Test Accuracy:  0.926509186351706
Train Accuracy:  0.9363517060367454
Scores:  [0.90655738 0.90327869 0.92622951 0.90640394 0.9228243 ]
Validation accuracy with MinMaxScaler 0.9130587633583677
```

3.4) Evaluation

Confusion Matrix for UnScaled data

Evaluation for un-scaled data

```
yhat = clf.predict(X_test)
cm = confusion_matrix(y_test, yhat)
tn, fp, fn, tp = cm.ravel()
print("Confusion matrix: \n", cm)
```

```
Confusion matrix:
 [[273  53]
 [ 41 395]]
```

Confusion Matrix for Scaled data

Evaluation for scaled data

```
yhat = clf.predict(X_test)
cm = confusion_matrix(y_test, yhat)

tn, fp, fn, tp = cm.ravel()
print("Confusion matrix: \n", cm)
```

```
Confusion matrix:
 [[264  62]
 [ 30 406]]
```

After comparing the both model accuracy, the scaled data is higher.

Model building without scaling

- Test Accuracy: 0.8792650918635171
- Train Accuracy: 0.8881233595800525

Model building without scaling(MinMaxScaler)

- Test Accuracy: 0.9238845144356955
- Train Accuracy: 0.9343832020997376

By comparing both decision tree and kNN classification algorithms, decision tree accuracy is 93% and kNN is 93%. So KNN has the highest accuracy compared to decision tree.