Importing libraries and Pokemon dataset

```
%matplotlib
```

```
Using matplotlib backend: Qt5Agg
```

```
from numpy import arange
import numpy
from matplotlib import pyplot as plt
from scipy.stats import norm
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn import model_selection
#from sklearn.metrics import accuracy_score
```

```
plt.rcParams['figure.figsize'] = [16, 7]
```

```
df = pd.read_csv(r"C:/Users/Aadya/Downloads/Pokemon.csv")
```

Data Exploration

```
df.head()
```

	Type_1	Type_2	Hit points	Attack	Defence	Special attack	Special defence	Speed	Generation	Final evolution	Catch rate	Legendary	Mega evolution	Alolan form	Galarian form	Experience_type
0	Grass	Poison	45	49	49	65	65	45	1	0	45	0	0	0	0	Medium Slow
1	Grass	Poison	60	62	63	80	80	60	1	0	45	0	0	0	0	Medium Slow
2	Grass	Poison	80	82	83	100	100	80	1	1	45	0	0	0	0	Medium Slow
3	Grass	Poison	80	100	123	122	120	80	6	1	45	0	1	0	0	Medium Slow
4	Fire	NaN	39	52	43	60	50	65	1	0	45	0	0	0	0	Medium Slow

```
#Information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1032 entries, 0 to 1031
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Type_1           1032 non-null   object
 1   Type_2           548 non-null    object
 2   Hit points       1032 non-null   int64
 3   Attack           1032 non-null   int64
 4   Defence          1032 non-null   int64
 5   Special attack   1032 non-null   int64
 6   Special defence  1032 non-null   int64
 7   Speed            1032 non-null   int64
 8   Generation       1032 non-null   int64
 9   Final evolution  1032 non-null   int64
 10  Catch rate       1032 non-null   int64
 11  Legendary        1032 non-null   int64
```

```python
#Data types of the dataset columns
df.dtypes
```

```
Type_1              object
Type_2              object
Hit points           int64
Attack               int64
Defence              int64
Special attack       int64
Special defence      int64
Speed                int64
Generation           int64
Final evolution      int64
Catch rate           int64
Legendary            int64
Mega evolution       int64
Alolan form          int64
Galarian form        int64
Experience_type     object
dtype: object
```

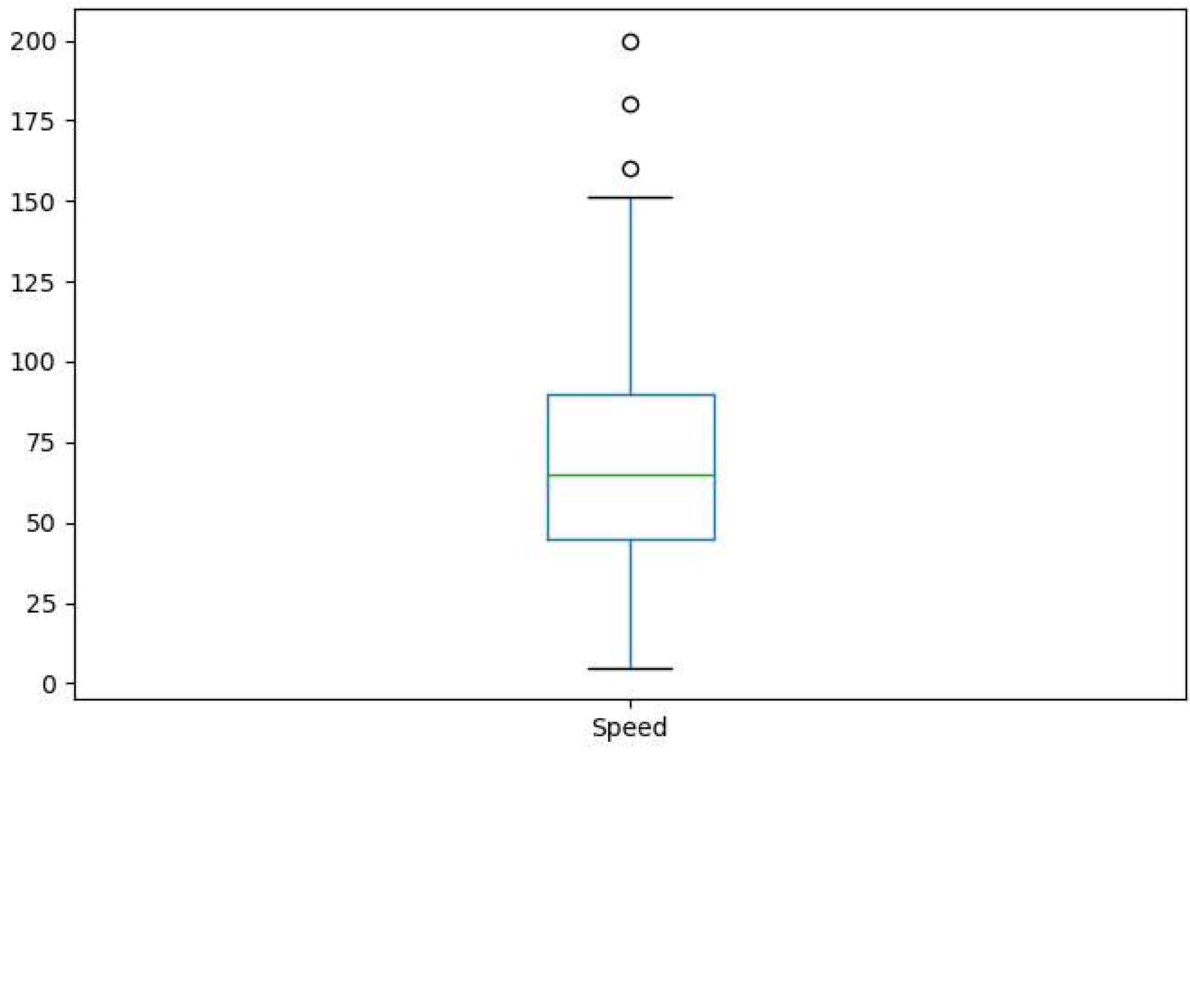
```python
#Memory used by each column in the dataset
df.memory_usage()
```

```
Index                128
Type_1              8256
Type_2              8256
Hit points          8256
Attack              8256
Defence             8256
Special attack      8256
Special defence     8256
Speed               8256
Generation          8256
Final evolution     8256
Catch rate          8256
Legendary           8256
Mega evolution      8256
Alolan form         8256
Galarian form       8256
Experience_type     8256
dtype: int64
```

```
#Total memory used by the dataset
df.memory_usage().sum()
```

```
132224
```

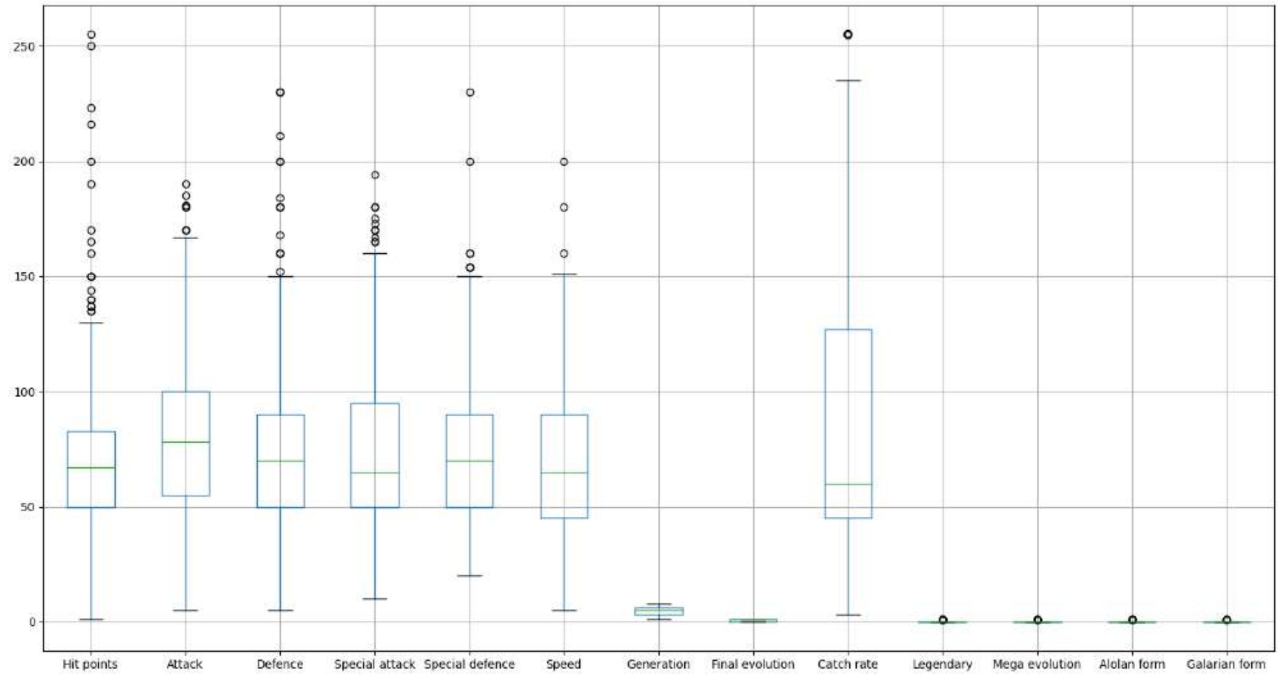
```
#Boxplot
df['Speed'].plot.box(figsize=(8, 5)); # Boxplot of a column
```



200
175
150
125
100
75
50
25
0
Speed

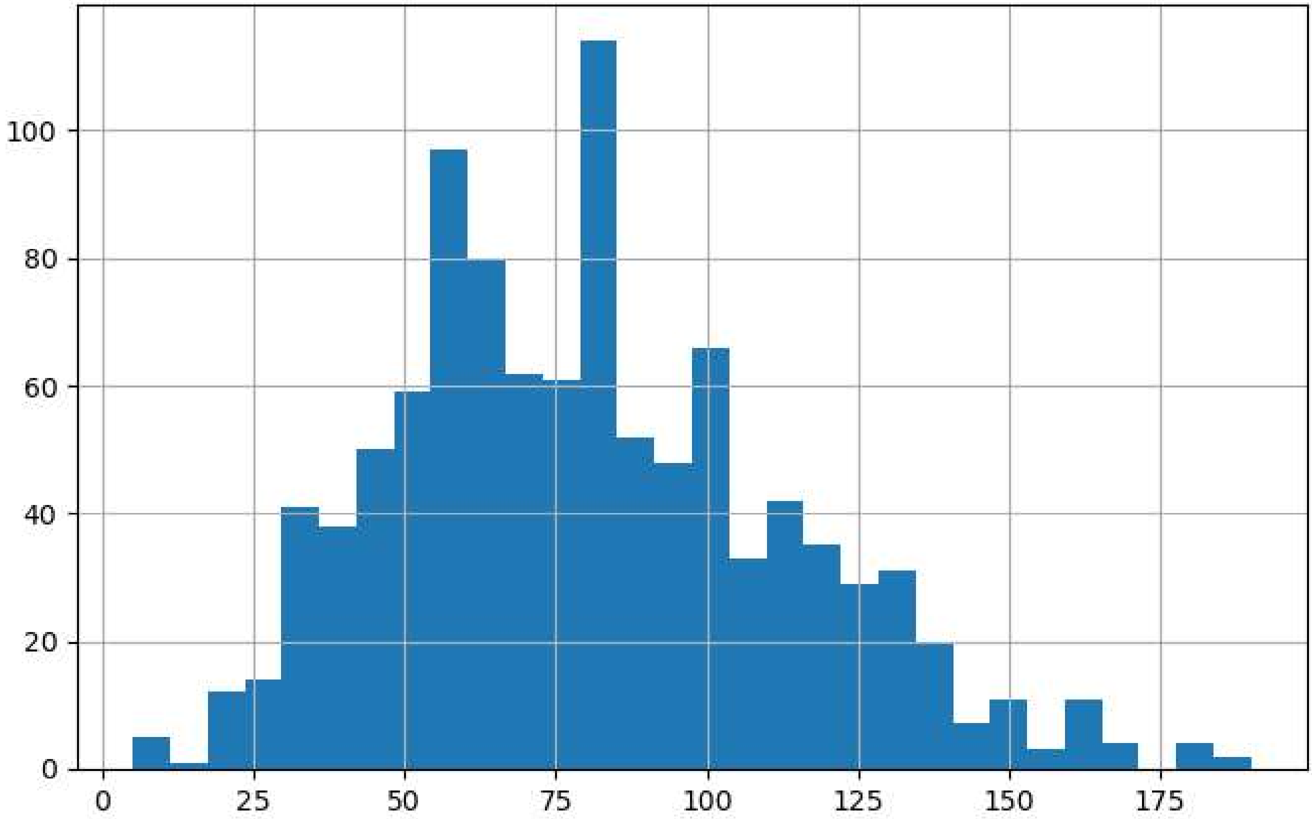
```
#Boxplot of all the columns with numerical data
df.boxplot(figsize=(20,20))
```

```
<AxesSubplot:>
```

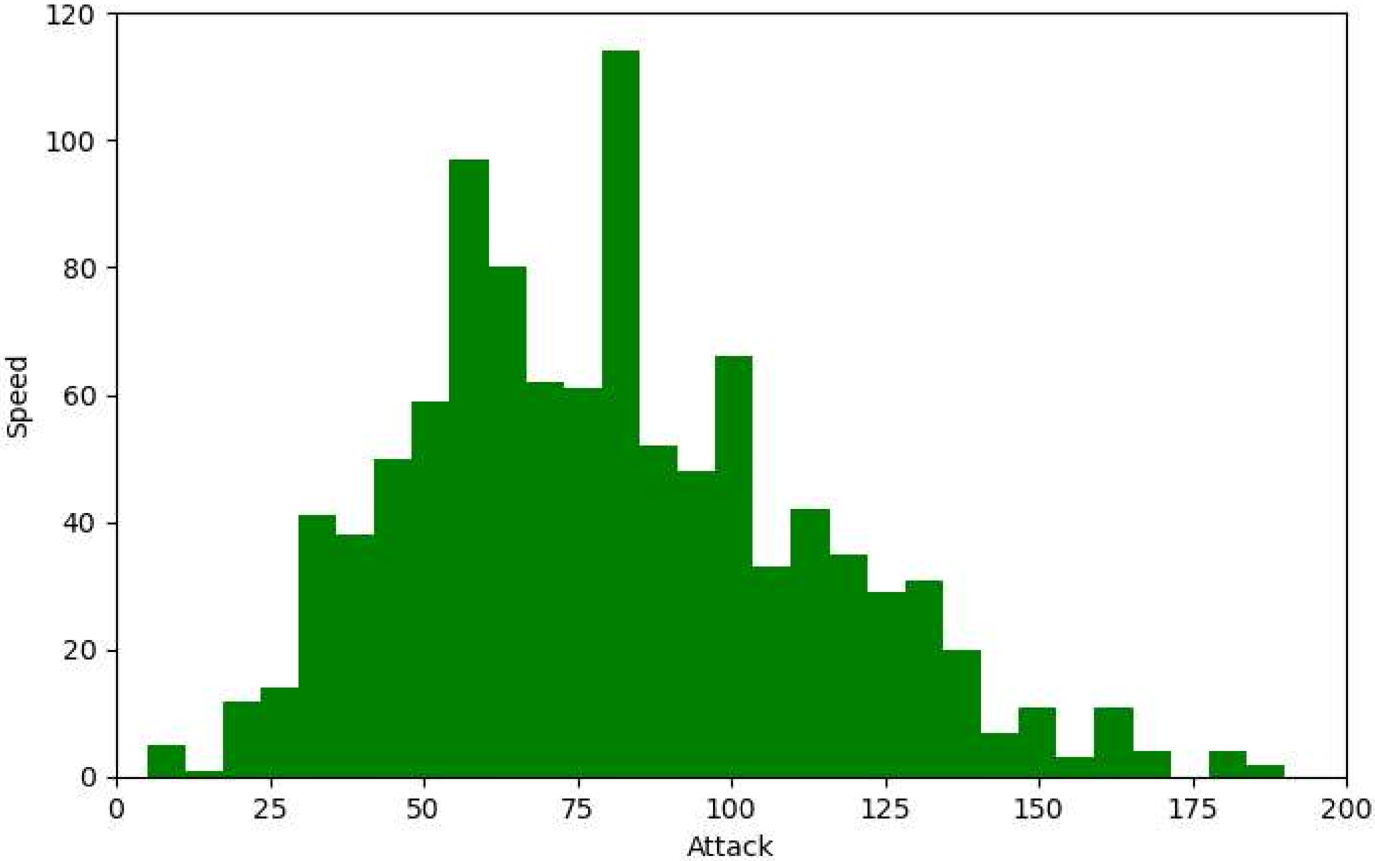


250
200
150
100
50
0
Hit points
Attack
Defence
Special attack
Special defence
Speed
Generation
Final evolution
Catch rate
Legendary
Mega evolution
Alolan form
Galarian form

```
#Histogram
df['Attack'].hist(bins=30, figsize=(8, 5)); # we can specify the number of bins
```

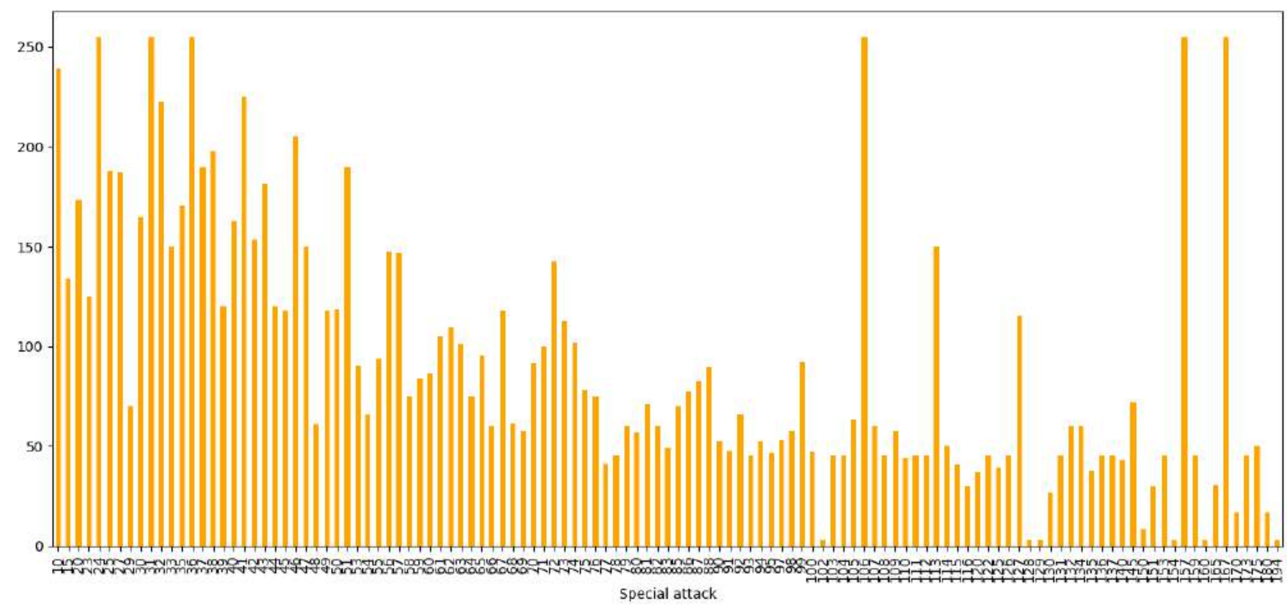

0
20
40
60
80
100
0
25
50
75
100
125
150
175

```
ax = df['Attack'].hist(bins=30, grid=False, color='green', figsize=(8, 5)) # grid turned off and color changed
ax.set_xlabel('Attack')
ax.set_ylabel('Speed')
ax.set_xlim(0,200) #limiting display range to 0-200 for the x-axis
ax.set_ylim(0,120); #limiting display range to 0-120 for the y-axis
```



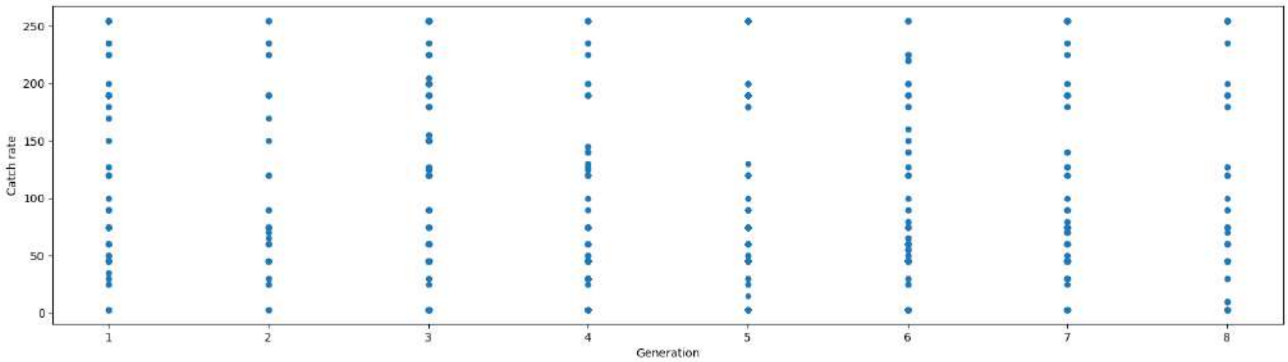
120
100
80
60
40
20
0
Speed
0
25
50
75
100
125
150
175
200
Attack

```python
#Barplot
df_special_attack = df.groupby('Special attack')['Catch rate'].mean()
df_special_attack[:].plot.bar(color='orange');
```


250
200
150
100
50
0
Special attack

```
#Scatterplot
df.plot.scatter('Generation','Catch rate',figsize=(20,5))
```

```
<AxesSubplot:xlabel='Generation', ylabel='Catch rate'>
```



250
200
150
100
50
0
Catch rate
1
2
3
4
5
6
7
8
Generation

Data Cleaning

```
#Check if there are missing values in the dataset
df.isnull().sum().sum()
```

```
484
```

```
#Remove missing values
df=df.dropna()
```

```
#Recheck if null values have been removed
df.isnull().sum().sum()
```

```
0
```

```
#Check if there are duplicate rows in the dataset
df.duplicated().sum()
```

```
0
```

```
#Count unique value in a column
df.Experience_type.value_counts()
```

```
Medium Fast    225
Slow           161
Medium Slow    122
Fast            21
Erratic         15
Fluctuating      4
Name: Experience_type, dtype: int64
```

Label encoding columns having non-numeric values

```
df['Type_1'].replace({'Bug':0,'Water':1,'Grass':2,'Normal':3,'Rock':4,'Psychic':5,'Dark':6,'Fire':7,'Dragon':8,'Ghost':9,
                      'Electric':10,'Steel':11,'Ground':12,'Poison':13,'Ice':14,'Fighting':15,'Flying':16,'Fairy':17},
                      inplace=True)
```

```
df['Type_2'].replace({'Bug':0,'Water':1,'Grass':2,'Normal':3,'Rock':4,'Psychic':5,'Dark':6,'Fire':7,'Dragon':8,'Ghost':9,
                      'Electric':10,'Steel':11,'Ground':12,'Poison':13,'Ice':14,'Fighting':15,'Flying':16,'Fairy':17},
                      inplace=True)
```

```
df['Experience_type'].replace({'Erratic':0,'Slow':1,'Medium Slow':2,'Fluctuating':3,'Medium Fast':4,'Fast':5},
                      inplace=True)
```

```
#Data after Label encoding
df.head()
```

	Type_1	Type_2	Hit points	Attack	Defence	Special attack	Special defence	Speed	Generation	Final evolution	Catch rate	Legendary	Mega evolution	Alolan form	Galarian form	Experience_type
0	2	13	45	49	49	65	65	45	1	0	45	0	0	0	0	2
1	2	13	60	62	63	80	80	60	1	0	45	0	0	0	0	2
2	2	13	80	82	83	100	100	80	1	1	45	0	0	0	0	2
3	2	13	80	100	123	122	120	80	6	1	45	0	1	0	0	2
6	7	16	78	84	78	109	85	100	1	1	45	0	0	0	0	2

```
#No. of rows in the dataset after cleaning
print(len(df.axes[0]))
```

548

```
#Data types of dataset columns after label encoding
df.dtypes
```

```
Type_1             int64
Type_2             int64
Hit points         int64
Attack             int64
Defence            int64
Special attack     int64
Special defence    int64
Speed              int64
Generation         int64
Final evolution    int64
Catch rate         int64
Legendary          int64
Mega evolution     int64
Alolan form        int64
Galarian form      int64
Experience_type    int64
dtype: object
```

```
#Statistics for all the dataset columns
df.describe()
```

	Type_1	Type_2	Hit points	Attack	Defence	Special attack	Special defence	Speed	Generation	Final evolution	Catch rate	Legendary	evol
count	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.000000	548.00
mean	5.868613	10.554745	71.996350	84.773723	80.498175	77.855839	76.175182	71.056569	4.638686	0.645985	82.899635	0.144161	0.07
std	4.607946	5.160559	24.756437	33.297670	32.023329	33.690320	27.466755	29.706955	2.206215	0.478651	70.521866	0.351573	0.26
min	0.000000	0.000000	1.000000	10.000000	15.000000	10.000000	20.000000	5.000000	1.000000	0.000000	3.000000	0.000000	0.00
25%	2.000000	6.000000	55.000000	60.000000	56.500000	53.000000	55.000000	50.000000	3.000000	0.000000	45.000000	0.000000	0.00
50%	5.000000	12.000000	70.000000	82.000000	77.500000	71.000000	75.000000	70.000000	5.000000	1.000000	45.000000	0.000000	0.00
75%	10.000000	16.000000	90.000000	105.000000	100.000000	100.000000	95.000000	93.500000	6.000000	1.000000	120.000000	0.000000	0.00
max	17.000000	17.000000	223.000000	190.000000	230.000000	180.000000	230.000000	160.000000	8.000000	1.000000	255.000000	1.000000	1.00

```python
#Variance
df.var()
```

```
Type_1                21.233163
Type_2                26.631367
Hit points           612.881157
Attack              1108.734811
Defence             1025.493598
Special attack      1135.037681
Special defence      754.422637
Speed                882.503193
Generation             4.867385
Final evolution        0.229106
Catch rate          4973.333601
Legendary              0.123604
Mega evolution         0.067788
Alolan form            0.026672
Galarian form          0.023202
Experience_type        2.047652
dtype: float64
```

```python
#Skewness
df.skew()
```

```
Type_1             0.474183
Type_2            -0.426361
Hit points         1.029989
Attack             0.474682
Defence            0.919484
Special attack     0.585686
Special defence    0.624107
Speed              0.246224
Generation        -0.187304
Final evolution   -0.612222
Catch rate         1.220664
Legendary          2.031683
Mega evolution     3.292117
Alolan form        5.809140
Galarian form      6.276438
Experience_type   -0.015229
dtype: float64
```

```
#Kurtosis
df.kurtosis()
```

```
Type_1              -0.934714
Type_2              -1.115919
Hit points           4.176249
Attack              -0.032176
Defence              1.914779
Special attack      -0.211816
Special defence      1.302196
Speed               -0.467791
Generation          -1.117694
Final evolution     -1.631151
Catch rate           0.461084
Legendary            2.135516
Mega evolution       8.870397
Alolan form         31.862378
Galarian form       37.530633
Experience_type     -1.554348
dtype: float64
```

Data Selection

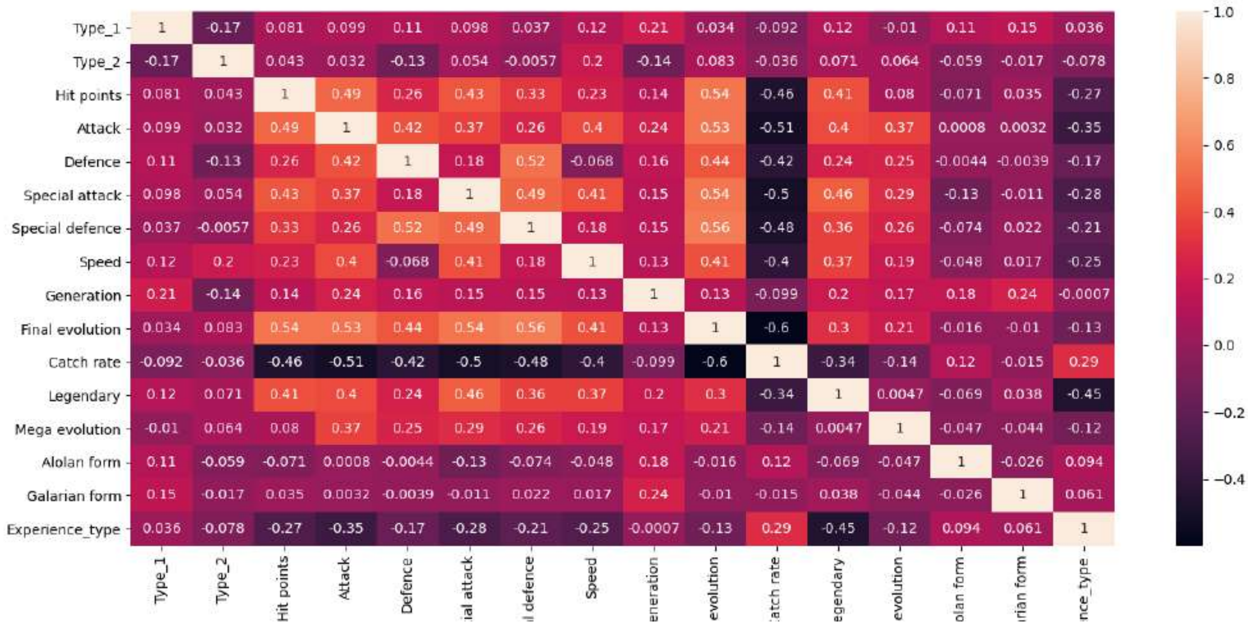
```
#Column-wise correlation in the dataset
df.corr()
```

	Type_1	Type_2	Hit points	Attack	Defence	Special attack	Special defence	Speed	Generation	Final evolution	Catch rate	Legendary	Mega evolutio
Type_1	1.000000	-0.168600	0.081118	0.098628	0.107498	0.098349	0.037073	0.123816	0.213453	0.033578	-0.092084	0.117789	-0.01027
Type_2	-0.168600	1.000000	0.043317	0.032468	-0.131305	0.054140	-0.005730	0.197165	-0.138599	0.082612	-0.035728	0.070710	0.06369
Hit points	0.081118	0.043317	1.000000	0.492741	0.260386	0.434099	0.333410	0.229494	0.135502	0.537552	-0.457811	0.413425	0.08002
Attack	0.098628	0.032468	0.492741	1.000000	0.424376	0.370145	0.256642	0.396012	0.237116	0.532584	-0.507608	0.398669	0.37325
Defence	0.107498	-0.131305	0.260386	0.424376	1.000000	0.177936	0.521697	-0.067716	0.157472	0.435170	-0.416009	0.237665	0.25304
Special attack	0.098349	0.054140	0.434099	0.370145	0.177936	1.000000	0.486423	0.412444	0.149012	0.542354	-0.502729	0.457229	0.29006
Special defence	0.037073	-0.005730	0.333410	0.256642	0.521697	0.486423	1.000000	0.178671	0.147727	0.562754	-0.478592	0.357461	0.25563
Speed	0.123816	0.197165	0.229494	0.396012	-0.067716	0.412444	0.178671	1.000000	0.128122	0.414631	-0.398958	0.368728	0.19493
Generation	0.213453	-0.138599	0.135502	0.237116	0.157472	0.149012	0.147727	0.128122	1.000000	0.134869	-0.098887	0.196908	0.17330
Final evolution	0.033578	0.082612	0.537552	0.532584	0.435170	0.542354	0.562754	0.414631	0.134869	1.000000	-0.599998	0.303827	0.20772

```
#Import seaborn library
import seaborn as sns
```

```
sns.heatmap(df.corr('pearson'),annot=True)
```

```
<AxesSubplot:>
```



Type_1
Type_2
Hit points
Attack
Defence
Special attack
Special defence
Speed
Generation
Final evolution
Catch rate
Legendary
Mega evolution
Alolan form
Galarian form
Experience_type
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Decision Tree

Target column: Experience_type

Columns: Generation, Alolan form, Galarian form are weakly correlated to Experience_type

Columns: Special attack is very similar to Hit points

Thus, columns Generation, Alolan form, Galarian form and Special attack will be dropped.

```
df=df.drop(['Generation','Alolan form','Galarian form','Special attack'],axis=1)
```

```
df.replace('', numpy.nan, inplace=True)
```

```
df.dropna(inplace=True)
```

```
df.head()
```

	Type_1	Type_2	Hit points	Attack	Defence	Special defence	Speed	Final evolution	Catch rate	Legendary	Mega evolution	Experience_type
0	2	13	45	49	49	65	45	0	45	0	0	2
1	2	13	60	62	63	80	60	0	45	0	0	2
2	2	13	80	82	83	100	80	1	45	0	0	2
3	2	13	80	100	123	120	80	1	45	0	1	2
6	7	16	78	84	78	85	100	1	45	0	0	2

Data Splitting and Model Building (Decision Tree)

```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
feature_cols = ['Type_1','Type_2','Hit points','Attack','Defence','Special defence','Speed','Final evolution',
                'Catch rate','Legendary','Mega evolution']
X = df[feature_cols] # Features
y = df.Experience_type # Target variable
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1) # 80% training and 20% test
```

```
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)
```

Without hyperparameter tuning

Hyperparameters:

criterion - Choose attribute selection measure (gini / entropy)

splitter - Splitting Strategy (best / random splitting)

max_depth - Maximum depth of the tree (None / integer)

```
#Decision tree classifier
clf = DecisionTreeClassifier(criterion="gini", splitter="random", max_depth=None)

#Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
#Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5727272727272728
```

With hyperparameter tuning

Hyperparameters:

criterion - Choose attribute selection measure (gini / entropy)

splitter - Splitting Strategy (best / random splitting)

max_depth - Maximum depth of the tree (None / integer)

```
#Decision tree classifier
clf = DecisionTreeClassifier(criterion="entropy", splitter="best", max_depth=8)

#Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
#Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6545454545454545
```

Random Forest

Target column: Experience_type

Columns: Generation, Alolan form, Galarian form are weakly correlated to Experience_type

Thus, columns Generation, Alolan form, Galarian form and Hit points will be dropped.

```
df=df.drop(['Generation','Alolan form','Galarian form'],axis=1)
```

```
df.replace('', numpy.nan, inplace=True)
```

```
df.dropna(inplace=True)
```

```
df.head()
```

	Type_1	Type_2	Hit points	Attack	Defence	Special attack	Special defence	Speed	Final evolution	Catch rate	Legendary	Mega evolution	Experience_type
0	2	13	45	49	49	65	65	45	0	45	0	0	2
1	2	13	60	62	63	80	80	60	0	45	0	0	2
2	2	13	80	82	83	100	100	80	1	45	0	0	2
3	2	13	80	100	123	122	120	80	1	45	0	1	2
6	7	16	78	84	78	109	85	100	1	45	0	0	2

Data Splitting and Model Building (Random Forest)

```
from sklearn.model_selection import train_test_split
```

```
X=df[['Type_1','Type_2','Hit points','Attack','Defence','Special attack','Special defence','Speed','Final evolution',
      'Catch rate', 'Legendary', 'Mega evolution']]  # Features
y=df['Experience_type']  # Labels
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # 80% training and 20% test
```

Without hyperparameter tuning

Hyperparameters:

max_depth - depth of a tree

min_samples_split - minimum no. of observations in a node to split

max_leaf_nodes - maximum no. of children a node can split into

min_samples_leaf - minimum no. of samples in a leaf node after parent node split

n_estimators - no. of trees in the forest

max_samples (bootstrap sample) - fraction of the original dataset given to each tree for training

max_features - maximum no. of features given to each tree in the forest

```
#Random Forest classifier
from sklearn.ensemble import RandomForestClassifier

#Train Random forest model
clf=RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

```
# Model Accuracy
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6454545454545455
```

With hyperparameter tuning

Hyperparameters:

max_depth - depth of a tree

min_samples_split - minimum no. of observations in a node to split

max_leaf_nodes - maximum no. of children a node can split into

min_samples_leaf - minimum no. of samples in a leaf node after parent node split

n_estimators - no. of trees in the forest

max_samples (bootstrap sample) - fraction of the original dataset given to each tree for training

max_features - maximum no. of features given to each tree in the forest

```
#Random Forest classifier
from sklearn.ensemble import RandomForestClassifier

#Train Random forest model
clf=RandomForestClassifier(max_depth=5,min_samples_split=6,max_leaf_nodes=100,min_samples_leaf=10,n_estimators=70,
                           max_samples=0.2,max_features=6)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

```
# Model Accuracy
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7
```

Following are the results when different hyperparameters are adjusted for Decision tree and Random forest models:

Decision tree:

Without hyperparameter tuning: 0.572

With hyperparameter tuning: 0.654

Random Forest:

Without hyperparameter tuning: 0.645

With hyperparameter tuning: 0.7