
A structured and detailed list covering a wide range of functions across various libraries commonly used in data analytics. This list will include functions from libraries like NumPy, pandas, matplotlib, seaborn, scikit-learn, and more. Each entry will include a brief description and an example where applicable.

NumPy Functions

1. **`np.array()`** - Create a NumPy array.

```
import numpy as np
arr = np.array([1, 2, 3])
```

2. **`np.zeros()`** - Create an array filled with zeros.

```
arr = np.zeros((3, 3))
```

3. **`np.ones()`** - Create an array filled with ones.

```
arr = np.ones((2, 2))
```

4. **`np.eye()`** - Create an identity matrix.

```
arr = np.eye(3)
```

5. **`np.arange()`** - Create an array with a range of values.

```
arr = np.arange(0, 10, 2)
```

6. **`np.linspace()`** - Create an array with evenly spaced values.

```
arr = np.linspace(0, 1, 5)
```

7. **`np.random.rand()`** - Generate random values between 0 and 1.

```
arr = np.random.rand(2, 3)
```

8. **`np.random.randint()`** - Generate random integers.

```
arr = np.random.randint(0, 10, (2, 3))
```

9. **`np.mean()`** - Compute the mean of an array.

```
mean = np.mean([1, 2, 3, 4])
```

10. **`np.median()`** - Compute the median of an array.

```
median = np.median([1, 2, 3, 4])
```

11. **`np.std()`** - Compute the standard deviation of an array.

```
std_dev = np.std([1, 2, 3, 4])
```

12. **`np.sum()`** - Compute the sum of array elements.

```
total = np.sum([1, 2, 3, 4])
```

13. **`np.prod()`** - Compute the product of array elements.

```
product = np.prod([1, 2, 3, 4])
```

14. **`np.min()`** - Compute the minimum of an array.

```
min_val = np.min([1, 2, 3, 4])
```

15. **`np.max()`** - Compute the maximum of an array.

```
max_val = np.max([1, 2, 3, 4])
```

16. **`np.dot()`** - Compute the dot product of two arrays.

```
result = np.dot([1, 2], [3, 4])
```

17. **`np.transpose()`** - Transpose an array.

```
arr = np.array([[1, 2], [3, 4]])
transposed = np.transpose(arr)
```

18. **`np.concatenate()`** - Concatenate two or more arrays.

```
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
concatenated = np.concatenate((arr1, arr2))
```

19. **`np.reshape()`** - Reshape an array.

```
arr = np.array([1, 2, 3, 4])
reshaped = np.reshape(arr, (2, 2))
```

20. **`np.split()`** - Split an array into multiple sub-arrays.

```
arr = np.array([1, 2, 3, 4])
split = np.split(arr, 2)
```

21. **`np.where()`** - Return indices where conditions are true.

```
indices = np.where([1, 2, 3] > 1)
```

22. **`np.unique()`** - Find unique elements in an array.

```
unique_elements = np.unique([1, 1, 2, 2])
```

23. **`np.histogram()`** - Compute the histogram of a dataset.

```
hist, bin_edges = np.histogram([1, 2, 2, 3], bins=3)
```

24. **`np.corrcoef()`** - Compute the correlation coefficient matrix.

```
corr_matrix = np.corrcoef([1, 2, 3], [4, 5, 6])
```

25. **`np.polyfit()`** - Fit a polynomial to data.

```
coeffs = np.polyfit([1, 2, 3], [4, 5, 6], 1)
```

pandas Functions

26. **`pd.DataFrame()`** - Create a DataFrame.

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
```

27. **`pd.Series()`** - Create a Series.

```
series = pd.Series([1, 2, 3])
```

28. **`pd.read_csv()`** - Read a CSV file into a DataFrame.

```
df = pd.read_csv('data.csv')
```

29. **`pd.read_excel()`** - Read an Excel file into a DataFrame.

```
df = pd.read_excel('data.xlsx')
```

30. **`pd.to_datetime()`** - Convert a series to datetime.

```
df['date'] = pd.to_datetime(df['date'])
```

31. **`pd.concat()`** - Concatenate DataFrames.

```
df1 = pd.DataFrame({'A': [1, 2]})
df2 = pd.DataFrame({'B': [3, 4]})
result = pd.concat([df1, df2], axis=1)
```

32. **`pd.merge()`** - Merge DataFrames.

```
df1 = pd.DataFrame({'key': ['A', 'B'], 'value': [1, 2]})
df2 = pd.DataFrame({'key': ['A', 'C'], 'value': [3, 4]})
result = pd.merge(df1, df2, on='key')
```

33. **`pd.groupby()`** - Group data by a column.

```
grouped = df.groupby('A')
```

34. **`pd.pivot_table()`** - Create a pivot table.

```
pivot = pd.pivot_table(df, values='B', index='A', columns='C')
```

35. **`df.head()`** - Return the first n rows of a DataFrame.

```
head = df.head(5)
```

36. **`df.tail()`** - Return the last n rows of a DataFrame.

```
tail = df.tail(5)
```

37. **`df.describe()`** - Generate descriptive statistics.

```
stats = df.describe()
```

38. **`df.info()`** - Get summary of a DataFrame.

```
info = df.info()
```

39. **`df.drop()`** - Drop rows or columns.

```
df_dropped = df.drop('A', axis=1)
```

40. **`df.fillna()`** - Fill missing values.

```
df_filled = df.fillna(0)
```

41. **`df.dropna()`** - Drop missing values.

```
df_dropped_na = df.dropna()
```

42. **`df.isnull()`** - Check for missing values.

```
nulls = df.isnull()
```

43. **`df.notnull()`** - Check for non-missing values.

```
not_nulls = df.notnull()
```

44. **`df.apply()`** - Apply a function along an axis.

```
result = df.apply(np.sum, axis=0)
```

45. **`df.map()`** - Map values using a function or dictionary.

```
df['A'] = df['A'].map(lambda x: x*2)
```

46. **`df.sort_values()`** - Sort DataFrame by values.

```
sorted_df = df.sort_values(by='A')
```

47. **`df.set_index()`** - Set the DataFrame index.

```
df_indexed = df.set_index('A')
```

48. **`df.reset_index()`** - Reset the DataFrame index.

```
df_reset = df.reset_index()
```

49. **`df.rename()`**
    - Rename columns or indices.

      ```
      df_renamed = df.rename(columns={'A': 'Alpha'})
      ```

50. **`df.query()`** - Query the DataFrame.

```
result = df.query('A > 2')
```

51. **`df.to_csv()`** - Write DataFrame to a CSV file.

```
df.to_csv('output.csv')
```

52. **`df.to_excel()`** - Write DataFrame to an Excel file.

```
df.to_excel('output.xlsx')
```

53. **`df.corr()`** - Compute pairwise correlation of columns.

```
correlation = df.corr()
```

54. **`df.cumsum()`** - Compute cumulative sum.

```
cumsum = df.cumsum()
```

55. **`df.duplicated()`** - Detect duplicate rows.

```
duplicates = df.duplicated()
```

56. **`df.drop_duplicates()`** - Drop duplicate rows.

```
df_unique = df.drop_duplicates()
```

57. **`df.groupby().agg()`** - Aggregate data after grouping.

```
agg = df.groupby('A').agg({'B': 'sum'})
```

58. **`df.rolling()`** - Create a rolling window.

```
rolling_mean = df['A'].rolling(window=3).mean()
```

59. **`df.expanding()`** - Create an expanding window.

```
expanding_sum = df['A'].expanding().sum()
```

60. **`df.ewm()`** - Apply exponential weighted functions.

```
ewm_mean = df['A'].ewm(span=3).mean()
```

matplotlib Functions

61. **`plt.plot()`** - Plot y versus x as lines and/or markers.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
```

62. **`plt.scatter()`** - Create a scatter plot.

```
plt.scatter([1, 2, 3], [4, 5, 6])
```

63. **`plt.bar()`** - Create a bar plot.

```
plt.bar(['A', 'B', 'C'], [3, 5, 7])
```

64. **`plt.hist()`** - Create a histogram.

```
plt.hist([1, 2, 2, 3, 4], bins=4)
```

65. **`plt.boxplot()`** - Create a box plot.

```
plt.boxplot([1, 2, 2, 3, 4])
```

66. **`plt.pie()`** - Create a pie chart.

```
plt.pie([3, 5, 7], labels=['A', 'B', 'C'])
```

67. **`plt.title()`** - Set the title of the plot.

```
plt.title('My Plot')
```

68. **`plt.xlabel()`** - Set the x-axis label.

```
plt.xlabel('X Axis')
```

69. **`plt.ylabel()`** - Set the y-axis label.

```
plt.ylabel('Y Axis')
```

70. **`plt.legend()`** - Show a legend on the plot.

```
plt.plot([1, 2, 3], [4, 5, 6], label='Line')
plt.legend()
```

71. **`plt.grid()`** - Show grid lines.

```
plt.grid(True)
```

72. **`plt.show()`** - Display the plot.

```
plt.show()
```

73. **`plt.savefig()`** - Save the plot to a file.

```
plt.savefig('plot.png')
```

74. **`plt.xlim()`** - Set limits for x-axis.

```
plt.xlim(0, 10)
```

75. **`plt.ylim()`** - Set limits for y-axis.

```
plt.ylim(0, 10)
```

76. **`plt.subplots()`** - Create a figure and a set of subplots.

```
fig, ax = plt.subplots(2, 2)
```

77. **`plt.subplot()`** - Add a subplot to the figure.

```
plt.subplot(2, 2, 1)
```

78. **`plt.tight_layout()`** - Adjust subplots to fit into figure area.

```
plt.tight_layout()
```

79. **`plt.axhline()`** - Add a horizontal line across the axis.

```
plt.axhline(y=0.5, color='r')
```

80. **`plt.axvline()`** - Add a vertical line across the axis.

```
plt.axvline(x=0.5, color='r')
```

81. **`plt.annotate()`** - Annotate a point in the plot.

```
plt.annotate('Annotation', xy=(1, 2), xytext=(2, 3))
```

82. **`plt.errorbar()`** - Plot with error bars.

```
plt.errorbar([1, 2, 3], [4, 5, 6], yerr=0.1)
```

83. **`plt.imshow()`** - Display an image.

```
plt.imshow([[1, 2], [3, 4]], cmap='gray')
```

84. **`plt.contour()`** - Create a contour plot.

```
plt.contour([[1, 2], [3, 4]])
```

85. **`plt.contourf()`** - Create a filled contour plot.

```
plt.contourf([[1, 2], [3, 4]])
```

seaborn Functions

86. **`sns.scatterplot()`** - Create a scatter plot.

```
import seaborn as sns
sns.scatterplot(x='x', y='y', data=df)
```

87. **`sns.lineplot()`** - Create a line plot.

```
sns.lineplot(x='x', y='y', data=df)
```

88. **`sns.barplot()`** - Create a bar plot.

```
sns.barplot(x='x', y='y', data=df)
```

89. **`sns.boxplot()`** - Create a box plot.

```
sns.boxplot(x='x', y='y', data=df)
```

90. **`sns.histplot()`** - Create a histogram.

```
sns.histplot(df['x'])
```

91. **`sns.heatmap()`** - Create a heatmap.

```
sns.heatmap(data=df.corr())
```

92. **`sns.pairplot()`** - Plot pairwise relationships in a dataset.

```
sns.pairplot(df)
```

93. **`sns.heatmap()`** - Plot a heatmap.

```
sns.heatmap(data=df.corr())
```

94. **`sns.distplot()`** - Plot a univariate distribution.

```
sns.distplot(df['x'])
```

95. **`sns.violinplot()`** - Create a violin plot.

```
sns.violinplot(x='x', y='y', data=df)
```

96. **`sns.kdeplot()`** - Create a Kernel Density Estimate plot.

```
sns.kdeplot(df['x'])
```

97. **`sns.jointplot()`** - Create a joint plot.

```
sns.jointplot(x='x', y='y', data=df)
```

98. **`sns.lmplot()`** - Create a scatter plot with a linear fit.

```
sns.lmplot(x='x', y='y', data=df)
```

99. **`sns.palplot()`** - Plot a color palette.

```
sns.palplot(sns.color_palette("husl"))
```

.00. **`sns.set_style()`** - Set the aesthetic style of the plots.

```
sns.set_style('whitegrid')
```

scikit-learn Functions

.01. **`sklearn.model_selection.train_test_split()`** - Split arrays or matrices into random train and test subsets.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

.02. **`sklearn.preprocessing.StandardScaler()`** - Standardize features by removing the mean and scaling to unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

03. **`sklearn.preprocessing.MinMaxScaler()`** - Transform features by scaling each feature to a

    given range.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

04. **`sklearn.impute.SimpleImputer()`** - Impute missing values.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

05. **`sklearn.feature_selection.SelectKBest()`** - Select features according to the k highest scores.

```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(score_func=f_classif, k=10)
X_new = selector.fit_transform(X, y)
```

06. **`sklearn.feature_extraction.text.CountVectorizer()`** - Convert a collection of text documents to a matrix of token counts.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)
```

07. **`sklearn.feature_extraction.text.TfidfVectorizer()`** - Convert a collection of text documents to a matrix of TF-IDF features.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(documents)
```

08. **`sklearn.decomposition.PCA()`** - Perform Principal Component Analysis (PCA) for dimensionality reduction.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

09. **`sklearn.cluster.KMeans()`** - K-means clustering.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

10. **`sklearn.linear_model.LinearRegression()`** - Linear regression.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

11. **`sklearn.ensemble.RandomForestClassifier()`** - Random Forest classifier.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

12. **`sklearn.ensemble.GradientBoostingClassifier()`** - Gradient Boosting classifier.

```
from sklearn.ensemble import GradientBoostingClassifier
model = GradientBoostingClassifier()
model.fit(X_train, y_train)
```

13. **`sklearn.svm.SVC()`** - Support Vector Classification.

```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train, y_train)
```

14. **`sklearn.metrics.accuracy_score()`** - Compute the accuracy of a model.

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

15. **`sklearn.metrics.confusion_matrix()`** - Compute the confusion matrix.

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
```

16. **`sklearn.metrics.classification_report()`** - Generate a classification report.

```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
```

17. **`sklearn.metrics.roc_auc_score()`** - Compute the area under the receiver operating characteristic curve (ROC AUC).

```
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_prob)
```

18. **`sklearn.metrics.roc_curve()`** - Compute Receiver Operating Characteristic (ROC) curve.

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
```

19. **`sklearn.model_selection.GridSearchCV()`** - Perform grid search to find the best parameters for a model.

```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(model, param_grid)
```

```
grid.fit(X_train, y_train)
```

20. **`sklearn.model_selection.cross_val_score()`** - Evaluate a model using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
```

Other Useful Functions

21. **`datetime.datetime.now()`** - Get the current date and time.

```
from datetime import datetime
now = datetime.now()
```

22. **`datetime.timedelta()`** - Create a time difference.

```
from datetime import timedelta
delta = timedelta(days=5)
```

23. **`itertools.chain()`** - Chain multiple iterables.

```
from itertools import chain
result = list(chain([1, 2], [3, 4]))
```

24. **`itertools.combinations()`** - Create combinations of elements.

```
from itertools import combinations
combs = list(combinations([1, 2, 3], 2))
```

25. **`itertools.permutations()`** - Create permutations of elements.

```
from itertools import permutations
perms = list(permutations([1, 2, 3], 2))
```

26. **`functools.reduce()`** - Apply a function cumulatively to the items of an iterable.

```
from functools import reduce
result = reduce(lambda x, y: x + y, [1, 2, 3])
```

27. **`operator.itemgetter()`** - Create a function to get an item from a sequence.

```
from operator import itemgetter
getter = itemgetter(1)
result = getter([1, 2, 3])
```

28. **`pandas.plotting.scatter_matrix()`** - Create a scatter matrix plot.

```
from pandas.plotting import scatter_matrix
scatter_matrix(df)
```

29. **`scipy.stats.describe()`** - Compute various descriptive statistics.

```
from scipy.stats import describe
stats = describe([1, 2, 3, 4, 5])
```

30. **`scipy.optimize.minimize()`** - Minimize a function.

```
from scipy.optimize import minimize
result = minimize(lambda x: x**2, 0)
```