Data Cleaning

In [ ]:
```python
# Preparing notebook and loading data
import pandas as pd
df = pd.read_csv('data.csv')

# Identifying missing or null values
missing_values = df.isnull().sum()

# Imputing missing values
df['column'].fillna(value, inplace=True)

# Checking for data types
data_types = df.dtypes

# Assigning correct data types
df['column'] = pd.to_datetime(df['column'])

# Dealing with duplicated data
df.drop_duplicates(inplace=True)
```

Sorting and Filtering

In [ ]:
```python
# Sorting dataset
df.sort_values(by='column', ascending=False, inplace=True)

# Boolean indexing
filtered_data = df[df['column'] > 5]

# Query Method
filtered_data = df.query('column > 5')

# isin Method
filtered_data = df[df['column'].isin([1, 2, 3])]

# Combining Conditions
filtered_data = df[(df['column1'] > 5) & (df['column2'] < 10)]

# Using Loc and iloc
subset = df.loc[:, ['column1', 'column2']]
```

Data Joining

In [ ]:
```python
# Data joining
merged_data = pd.merge(df1, df2, on='key_column')

# Data concatenation
concatenated_data = pd.concat([df1, df2], axis=0)
```

EDA methods

In [ ]:
```python
# Value counts method
value_counts = df['column'].value_counts()

# Describe data
description = df.describe()

# Group by analysis
grouped_data = df.groupby('column').mean()

# Pivot table
pivot_table = pd.pivot_table(df, values='values', index=['index_column'], columns=['column'])

# Crosstab analysis
cross_tab = pd.crosstab(df['column1'], df['column2'])

# Correlation analysis
correlation_matrix = df.corr()
```

Data Visualisations

In [ ]:
```python
import matplotlib.pyplot as plt

# Bar charts
plt.bar(x_values, y_values)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Bar Chart')
plt.show()

# Pie charts
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Pie Chart')
plt.show()

# Line charts
plt.plot(x_values, y_values)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Line Chart')
plt.show()

# Histogram
plt.hist(data, bins=10)
plt.xlabel('X-axis label')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()

# Scatterplot
plt.scatter(x_values, y_values)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Scatter Plot')
plt.show()

# Heatmap
import seaborn as sns
sns.heatmap(data, cmap='viridis')
plt.title('Heatmap')
plt.show()

# Boxplot
plt.boxplot(data)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Box Plot')
plt.show()
```

Data Transformations

In [ ]:
```python
# Checking the distribution
sns.distplot(data)

# Normality test
from scipy.stats import shapiro
stat, p = shapiro(data)
if p > 0.05:
    print('Data looks normally distributed')
else:
    print('Data does not look normally distributed')

# Square root transformation
sqrt_transformed_data = np.sqrt(data)

# Logarithmic transformation
log_transformed_data = np.log(data)

# Boxcox transformation
from scipy.stats import boxcox
boxcox_transformed_data, _ = boxcox(data)

# Yeo-Johnson transformation
from scipy.stats import yeojohnson
yeo_johnson_transformed_data, _ = yeojohnson(data)
```

Statistical Tests

In [ ]:
```python
# One sample t-test
from scipy.stats import ttest_1samp
stat, p = ttest_1samp(data, popmean)

# Independent sample t-test
from scipy.stats import ttest_ind
stat, p = ttest_ind(data1, data2)

# One-way ANOVA
from scipy.stats import f_oneway
stat, p = f_oneway(data1, data2, data3)

# Chi-square test for independence
from scipy.stats import chi2_contingency
stat, p, dof, expected = chi2_contingency(observed)

# Pearson correlation
correlation_matrix = df.corr()

# Linear regression analysis
import statsmodels.api as sm
X = sm.add_constant(X)
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

Feature Engineering

In [ ]:
```python
# Dealing with date
df['date_column'] = pd.to_datetime(df['date_column'])
df['year'] = df['date_column'].dt.year
df['month'] = df['date_column'].dt.month
df['day'] = df['date_column'].dt.day

# Feature encoding
encoded_data = pd.get_dummies(df['categorical_column'])

# Feature binning
bins = [0, 25, 50, 75, 100]
labels = ['Low', 'Medium', 'High', 'Very High']
df['binned_feature'] = pd.cut(df['feature'], bins=bins, labels=labels)

# Feature mapping
mapping_dict = {'low': 0, 'medium': 1, 'high': 2}
df['mapped_feature'] = df['feature'].map(mapping_dict)

# Creating dummies
dummy_variables = pd.get_dummies(df['categorical_feature'])
```

Data Preprocessing

In [ ]:
```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

# Selecting features
X = df[['feature1', 'feature2']]

# Standard scaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# MinMax scaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Principal component analysis
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Regression ML

In [ ]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, accuracy_score

# Linear regression ML model
linear_regression_model = LinearRegression()
linear_regression_model.fit(X_train, y_train)
linear_regression_predictions = linear_regression_model.predict(X_test)
linear_regression_mse = mean_squared_error(y_test, linear_regression_predictions)

# Decision Tree regressor ML model
decision_tree_model = DecisionTreeRegressor()
decision_tree_model.fit(X_train, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test)
decision_tree_mse = mean_squared_error(y_test, decision_tree_predictions)

# Random Forest regressor ML model
random_forest_model = RandomForestRegressor()
random_forest_model.fit(X_train, y_train)
random_forest_predictions = random_forest_model.predict(X_test)
random_forest_mse = mean_squared_error(y_test, random_forest_predictions)

print("Linear Regression MSE:", linear_regression_mse)
print("Decision Tree MSE:", decision_tree_mse)
print("Random Forest MSE:", random_forest_mse)
```

Classification ML

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Logistic regression ML model
logistic_regression_model = LogisticRegression()
logistic_regression_model.fit(X_train, y_train)
logistic_regression_predictions = logistic_regression_model.predict(X_test)
logistic_regression_accuracy = accuracy_score(y_test, logistic_regression_predictions)

# Decision Tree classification ML model
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test)
decision_tree_accuracy = accuracy_score(y_test, decision_tree_predictions)

# Random Forest classification ML model
random_forest_model = RandomForestClassifier()
random_forest_model.fit(X_train, y_train)
random_forest_predictions = random_forest_model.predict(X_test)
random_forest_accuracy = accuracy_score(y_test, random_forest_predictions)

print("Logistic Regression Accuracy:", logistic_regression_accuracy)
print("Decision Tree Accuracy:", decision_tree_accuracy)
print("Random Forest Accuracy:", random_forest_accuracy)
```

KMeans Clustering

In [ ]:
```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Calculate WCSS for different values of k
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the elbow method graph
plt.figure(figsize=(10,6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()

# Choose the optimal number of clusters based on the elbow method
# From the graph, select the point where the decrease in WCSS breaks down (elbow point)
# In this example, let's assume the optimal number of clusters is 3

# Build KMeans clustering model with the optimal number of clusters
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42)
kmeans.fit(X)

# Get cluster labels for each data point
cluster_labels = kmeans.labels_
```