```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
# Load data
df = pd.read_csv('titanic.csv',  sep=',', decimal=',', encoding='windows-1251')

df.info()
df.head()
# Data preparation and exploration
# Remove irrelevant columns
df = df.drop(columns=['PassengerId', 'Name'])
df['Pclass'] = df['Pclass'].astype(str)
df_train, df_test = train_test_split(
    df,
    test_size=0.2,
    random_state=1
)
df_train.head()

df_train.shape, df_test.shape
df_train['Survived'].sum()
df_train['Survived'].count() - df_train['Survived'].sum()
plt.figure(figsize=(10, 10))

value_is = mpatches.Patch(color='purple', label='Value')
value_not = mpatches.Patch(color='aqua', label='No value')

plt.title('Heatmap missing values')
plt.legend(handles=[value_is, value_not], bbox_to_anchor=(1, 1), loc='upper left')
```

```
colours = ['purple', 'aqua']
sns.heatmap(
    df_train.isna(), cbar=False,
    cmap=sns.color_palette(colours),
)

plt.show()
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isna().mean() * 100).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing_data.head()
df_train = df_train.drop(columns=['Cabin', 'Ticket'])
df_test = df_test.drop(columns=['Cabin', 'Ticket'])
df_train = df_train.fillna(df_train.mean())
df_test = df_test.fillna(df_test.mean())

df_train['Embarked'] = df_train['Embarked'].fillna(df_train['Embarked'].mode()[0])
df_test['Embarked'] = df_test['Embarked'].fillna(df_train['Embarked'].mode()[0])

df_train['Age'] = df_train['Age'].fillna(df_train['Age'].mode()[0])
df_test['Age'] = df_test['Age'].fillna(df_train['Age'].mode()[0])
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isna().mean() * 100).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing_data.head()
total = df_test.isnull().sum().sort_values(ascending=False)
percent = (df_test.isna().mean() * 100).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing_data.head()
all_features = pd.concat([df_train, df_test]).reset_index(drop=True)
all_features = pd.get_dummies(all_features)
df_train = all_features.iloc[:df_train.shape[0], :]
df_test = all_features.iloc[df_train.shape[0]:, :]
# Split data into training and testing sets
X_train = df_train.drop(columns='Survived')
y_train = df_train['Survived']
```

```
X_test = df_test.drop(columns='Survived')
y_test = df_test['Survived']

X_train
# Build and evaluate models
models = [
    ('Logistic Regression', LogisticRegression()),
    ('Decision Tree', DecisionTreeClassifier(max_depth=3, random_state=1)),
    ('Random Forest', RandomForestClassifier(max_depth=5)),
    ('AdaBoost Classifier', AdaBoostClassifier(learning_rate=0.3))
]
i = 0
best = ''
for name, model in models:
    # Train model
    model.fit(X_train, y_train)

    scores = cross_val_score(model, X_train, y_train, cv=5)

    # Predict on test data
    y_pred = model.predict(X_test)

    # Evaluate model performance
    accuracy = accuracy_score(y_test, y_pred)
    print(f'{name}: \nScores - {scores} \nScores mean - {scores.mean()} \nAccuracy -  {accuracy:}\n')

    if accuracy>i:
        i = accuracy
        best = name

print(f'Best model is {best}')
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import plotly.express as px


df = pd.read_csv('Data2.csv', sep=';', encoding='cp1252')
df.info()

df.rename(columns={"Populatiion": "Population"}, inplace=True)
df['Area'] = df['Area'].str.replace(',', '.').astype(float)
df["GDP per capita"] = df["GDP per capita"].str.replace(',', '.').astype(float)
df["CO2 emission"] = df["CO2 emission"].str.replace(',', '.').astype(float)
fix_gdp = df[df['GDP per capita'] < 0]
area_gdp = df[df['Area'] < 0]
fix_gdp['GDP per capita'] *= -1
area_gdp['Area'] *= -1
df[df['GDP per capita'] < 0] = fix_gdp
df[df['Area'] < 0] = area_gdp
df = df.fillna(df.mean())


df.head()
df['Population density'] = df['Population'] / df['Area']
# Selecting the features for clustering
X = df[['GDP per capita', 'Population density']]

km_kwargs = {
    'init': 'random',
    'n_clusters': 4,
    'n_init': 10,
    'max_iter': 300,
    'random_state': 42,
}
# Using KMeans clustering algorithm to cluster the data
km = KMeans(**km_kwargs)
km.fit(X)

# Adding the predicted cluster labels to the original data
df['Cluster'] = km.labels_
```

```
# Grouping the data by region and cluster and calculating the mean for each group
region_cluster_means = df.groupby(['Region', 'Cluster']).mean()

# Sorting the data by GDP per capita and population density
sorted_data = region_cluster_means.sort_values(['GDP per capita', 'Population density'], ascending=False)

# Displaying the dominant region for GDP per capita and population density clusters
print('\n\n')
print("Dominant region for GDP per capita cluster: ", sorted_data.loc[sorted_data['GDP per
capita'].idxmax()].name[0])
print("Dominant region for population density cluster: ", sorted_data.loc[sorted_data['Population
density'].idxmax()].name[0])
print('\n\n')
fig = px.scatter(
    df, x='GDP per capita', y='Population density', color=km.labels_,
    hover_data=['Country Name', 'Region'],
    width=800, height=600
)

fig.update(layout_coloraxis_showscale=False)

fig.show()
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

labels = df.columns[2:]

for i in range(len(labels)):
    ax_i = (i // 3,  i % 3)
    axes[ax_i].set_title(labels[i])
    axes[ax_i].grid('-')
    axes[ax_i].hist(df[labels[i]])

fig.delaxes(axes[1][2])

import numpy as np
from scipy.stats import pearsonr

def linear_relationship(x, y):
    # Calculating the correlation coefficient and p-value
```

```
    corr, p_value = pearsonr(x, y)
    print(f'Correlation coefficient: {corr}')

    # Checking if the absolute value of the correlation coefficient is greater than 0.8
    if abs(corr) > 0.8:
        return True
    else:
        return False

# 1000 random integers between 0 and 50
x = np.random.randint(0, 50, 1000)

# Negative Correlation with some noise
y = -x + np.random.normal(0, 10, 1000)

linear_relationship(x, y)
```