```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("companies.csv")

# Display basic info
print("Dataset shape:", df.shape)
print("\nFirst few rows:")
display(df.head())
print("\nData types and missing values:")
display(df.info())
```

Show hidden output

```
# Check for missing values
missing_values = df.isnull().sum()
print("Missing values per column:")
display(missing_values[missing_values > 0])

# Drop columns with too many missing values (more than 50%)
threshold = len(df) * 0.5
df_cleaned = df.dropna(thresh=threshold, axis=1)

# For numerical columns with missing values, fill with median
num_cols = df_cleaned.select_dtypes(include=['int64', 'float64']).columns
for col in num_cols:
    if df_cleaned[col].isnull().sum() > 0:
        df_cleaned[col].fillna(df_cleaned[col].median(), inplace=True)

# For categorical columns with missing values, fill with mode
cat_cols = df_cleaned.select_dtypes(include=['object']).columns
for col in cat_cols:
    if df_cleaned[col].isnull().sum() > 0:
        df_cleaned[col].fillna(df_cleaned[col].mode()[0], inplace=True)

# Verify no missing values remain
print("\nMissing values after cleaning:")
display(df_cleaned.isnull().sum().sum())
```

Show hidden output

```
# Check for duplicate rows
print("Number of duplicate rows:", df_cleaned.duplicated().sum())

# Remove duplicates
df_cleaned = df_cleaned.drop_duplicates()
print("Shape after removing duplicates:", df_cleaned.shape)
```

Show hidden output

```
# Convert date columns to datetime
date_cols = ['founded_at', 'closed_at', 'first_investment_at', 'last_investment_at',
             'first_funding_at', 'last_funding_at', 'first_milestone_at',
             'last_milestone_at', 'created_at', 'updated_at']

for col in date_cols:
    if col in df_cleaned.columns:
        df_cleaned[col] = pd.to_datetime(df_cleaned[col], errors='coerce')

# Check numeric columns stored as strings
for col in df_cleaned.columns:
    if df_cleaned[col].dtype == 'object':
        try:
            df_cleaned[col] = pd.to_numeric(df_cleaned[col])
            print(f"Converted {col} to numeric")
        except:
            pass

print("\nData types after conversion:")
display(df_cleaned.dtypes)
```

Show hidden output

What can I help you build?

```
# Clean categorical columns by stripping whitespace and standardizing
cat_cols = df_cleaned.select_dtypes(include=['object']).columns
for col in cat_cols:
    df_cleaned[col] = df_cleaned[col].str.strip().str.lower()

# Example: Clean status column
if 'status' in df_cleaned.columns:
    df_cleaned['status'] = df_cleaned['status'].str.strip().str.lower()
    print("Status values after cleaning:", df_cleaned['status'].unique())
```

Show hidden output

TASK 2 : Data Manipulation

```
# Calculate company age
if 'founded_at' in df_cleaned.columns:
    current_year = datetime.now().year
    df_cleaned['founded_year'] = pd.to_datetime(df_cleaned['founded_at']).dt.year
    df_cleaned['company_age'] = current_year - df_cleaned['founded_year']
    # Handle missing/outlier ages
    df_cleaned['company_age'] = df_cleaned['company_age'].apply(lambda x: x if x > 0 and x < 100 else np.nan)
    df_cleaned['company_age'].fillna(df_cleaned['company_age'].median(), inplace=True)

# Extract month and day from dates if needed
for col in ['founded_at', 'closed_at']:
    if col in df_cleaned.columns:
        df_cleaned[f'{col}_month'] = pd.to_datetime(df_cleaned[col]).dt.month
        df_cleaned[f'{col}_day'] = pd.to_datetime(df_cleaned[col]).dt.day

# Create a binary indicator for whether the company has a homepage
if 'homepage_url' in df_cleaned.columns:
    df_cleaned['has_homepage'] = df_cleaned['homepage_url'].notnull().astype(int)
else:
    df_cleaned['has_homepage'] = 0  # Assume no homepage if column doesn't exist

# Create a binary indicator for whether the company has a Twitter account
if 'twitter_username' in df_cleaned.columns:
    df_cleaned['has_twitter'] = df_cleaned['twitter_username'].notnull().astype(int)
else:
    df_cleaned['has_twitter'] = 0


from sklearn.preprocessing import MinMaxScaler

# Select numerical columns to scale
num_cols_to_scale = ['funding_total_usd', 'investment_rounds', 'milestones', 'company_age']
num_cols_to_scale = [col for col in num_cols_to_scale if col in df_cleaned.columns]

if num_cols_to_scale:
    scaler = MinMaxScaler()
    df_cleaned[num_cols_to_scale] = scaler.fit_transform(df_cleaned[num_cols_to_scale])
    print("Scaled numerical columns:", num_cols_to_scale)


from sklearn.preprocessing import LabelEncoder

# Label encode binary categorical variables
binary_cat_cols = ['has_homepage', 'has_twitter']
for col in binary_cat_cols:
    if col in df_cleaned.columns:
        df_cleaned[col] = LabelEncoder().fit_transform(df_cleaned[col])

# One-hot encode other categorical variables
cat_cols_to_encode = ['category_code', 'country_code', 'state_code', 'region']
cat_cols_to_encode = [col for col in cat_cols_to_encode if col in df_cleaned.columns]

if cat_cols_to_encode:
    df_cleaned = pd.get_dummies(df_cleaned, columns=cat_cols_to_encode, drop_first=True)
    print("One-hot encoded columns:", cat_cols_to_encode)
```

Show hidden output

Task 3 :Data Labeling

```
# Create target variable based on status
if 'status' in df_cleaned.columns:
    # Define acquisition/closed as 0, operating/ipo as 1
    df_cleaned['active_status'] = df_cleaned['status'].apply(
        lambda x: 0 if x in ['acquired', 'closed'] else 1 if x in ['operating', 'ipo'] else np.nan
    )
```

```
    # Drop rows where we couldn't determine status
    df_cleaned = df_cleaned.dropna(subset=['active_status'])

    # Convert to integer
    df_cleaned['active_status'] = df_cleaned['active_status'].astype(int)

    print("\nTarget variable distribution:")
    display(df_cleaned['active_status'].value_counts())
```

Show hidden output

Dataset Preparation

```
# Drop unnecessary columns
cols_to_drop = ['id', 'Unnamed: 0.1', 'entity_type', 'entity_id', 'parent_id',
                'name', 'normalized_name', 'permalink', 'domain', 'homepage_url',
                'twitter_username', 'logo_url', 'short_description', 'description',
                'overview', 'tag_list', 'created_by', 'created_at', 'updated_at']

cols_to_drop = [col for col in cols_to_drop if col in df_cleaned.columns]
df_final = df_cleaned.drop(columns=cols_to_drop)

# Save the cleaned dataset
df_final.to_csv('cleaned_startup_data.csv', index=False)
print("\nFinal dataset shape:", df_final.shape)
display(df_final.head())
```

Show hidden output

Summary Report

```
# Generate a summary report
report = {
    "original_shape": df.shape,
    "cleaned_shape": df_final.shape,
    "columns_removed": list(set(df.columns) - set(df_final.columns)),
    "missing_values_initial": df.isnull().sum().sum(),
    "missing_values_final": df_final.isnull().sum().sum(),
    "duplicates_removed": df.duplicated().sum(),
    "target_distribution": dict(df_final['active_status'].value_counts()),
    "numerical_features_scaled": num_cols_to_scale,
    "categorical_features_encoded": cat_cols_to_encode
}

print("\nData Preprocessing Summary Report:")
for key, value in report.items():
    print(f"\n{key.replace('_', ' ').title()}:")
    display(value)
```

Show hidden output

```
# Save the cleaned dataset with active_status column
df_final.to_csv('cleaned_startup_data.csv', index=False)
print("Cleaned dataset saved as 'cleaned_startup_data.csv'")
```

Show hidden output

```
# Verify the active_status column was added correctly
if 'active_status' in df_final.columns:
    print("\nActive Status Distribution:")
    print(df_final['active_status'].value_counts())

    # Check which columns are available to display
    available_cols = []
    for col in ['name', 'status', 'active_status']:
        if col in df_final.columns:
            available_cols.append(col)

    if len(available_cols) > 1:  # Need at least status and active_status
        print("\nSample of data with active_status:")
        display(df_final[available_cols].head())
    else:
        print("\nCould not display sample - missing required columns")
else:
    print("Error: active_status column was not created successfully")
```

Show hidden output

```
Start coding or generate with AI.
```