```
# 1. Series ===================================================================
    # 1.1 create Series from a list
    s1 = pd.Series([14, -8, 0, 3, 9])

    # 1.2 create series with index, which can be numbers or strings
    s2 = pd.Series([14, -8, 0, 3, 9], index=['d','a','b','c', 'x'])

    # 1.3 Apply operations with method
    def power(x):
        return x**2
    s2.apply(power)

# 2. Data Frames ==============================================================
    # 2.1 Create From a List/Dict
    data = [
        ['Chiang Mai', 2016, 1630428],
        ['Phrae', 2018, 421653]
    ]
    data = {
        'province': ['Chiang Mai', 'Phrae'],
        'year': [2016, 2018],
        'population': [1630428, 421653]
    }
    df = pd.DataFrame(
        data=data,
        columns=['province','year','population']
    )

    # 2.2 Create df from data
    df = pd.read_csv('name.csv')
    df = pd.read_excel('name.xlsx', sheet_name='sheet1')
    df = pd.read_sas('name.sas7bdat')

    # 2.3 Write Files
    df.to_csv('name.csv', index=False)
    df.to_excel('name.xlsx', sheet_name='sheet1', index=False)

# 3. Index ====================================================================
    df.set_index('province')
    df = df.set_index(['province','year'])
    df.reset_index()

#4. Inspecting Data ===========================================================
    # 4.1 Rows
    df.head(3)
    df.tail(3)
    df.iloc[0] # Index Position
    df.loc['Chiang Mai'] # Index name

    # 4.2 Columns
    df.columns

        # 4.2.1 Dropping
        df.drop(columns=['province'])

        # 4.2.2 Renaming
        mapper = {
            'province':'Province',
            'year':'Year',
            'population':'Population'
        }
        df.rename(columns=mapper)

        # 4.2.3 Creating/Replacing columns
        df['population (K)'] = df['population'] / 1000

    # 4.3 Inspecting df Information
    df.shape # shows row + col
    df.info() # show everything
    df.dtypes # show type of col

# 5. Filtering and Sorting ====================================================
    # 5.1 Filtering
    condition_list = df['province'] == 'Chiang Mai'
    # and
    condition_list = (df['province'] == 'Chiang Mai') & (df['population'] > 1000000)
    # or
    condition_list = (df['province'] == 'Chiang Mai') | (df['population'] > 1000000)

    # 5.2 Sorting
    df.sort_values(by='population', ascending=False)
    df.sort_values(by=['year','population'], ascending=[True,False])

# 6. Statistics ===============================================================
    # 6.1 Numeric
    df.mean() # all numeric col
    df['population'].mean() # target specific col
    df[column_list].min()
    df[column_list].max()
    df[column_list].median()
    df[column_list].count()
    df[column_list].std()
    df[column_list].corr()
    df[column_list].quantile(0.75) # Q1 = 0.25, Q2 = 0.5, Q3 = 0.75

    # 6.2 Object
    df.unique() # tells unique value
    df.nunique() # tells unique count
    df.value_counts() # tells count by value

    # 6.3 Describe
    # Shows the numeric infos
    df.describe(include='all')

# 7. Grouping =================================================================
    # 7.1 Group By
    # select avg(population) from df group by province
    df.groupby('province')['population'].mean()

    # Convert to DataFrame
    df.groupby('province')['population'].mean().reset_index(name='avg_pop')

    # Multiple Group By
    mapper = {
    'population' : ['min','max','mean']
    }
    df.groupby('province').agg(mapper)

    # 7.2 Group By Window
    df['population'].rolling(2).sum()
    df.groupby('province')['population'].rolling(2, min_periods=1).sum()
        .reset_index(name='avg_pop_last2year')
```

```
# 8. Pivot =====================================================================
    # 8.1 Pivot
    # Basically reshape the df
    pivot = df.pivot_table(index=['province'], columns=['year'],
        values=['population'])

    # 8.2 Melt
    # Bascially unformats pivot
    melt = pd.melt(pivot, id_vars=['province'],
        value_vars=['2016','2017','2018'])

# 9. Data Cleansing ============================================================
    # 9.1 Find Null
    df.isnull()
    df.isnull().sum()

    # 9.2 Drop na
    df.isna()
    df.dropna()
    df.dropna(subset = ['province'])

    # 9.3 Fill na
    df.fillna('Missing')

# 10. Append and Join ==========================================================
    # 10.1 Append
    df = df1.append(df2)
    df = pd.concat([df1, df2])
    df2 = pd.concat([df1, df2], axis=1)

    # 10.2 Join
    df.merge(avg_df, on = 'province', how = 'inner')

# 11. SQL ======================================================================
    import pandasql
    sql_df = pandasql
        .sqldf("select province, avg(population) from df group by province;")

# 12. Imputing Values ==========================================================
    # 12.1 Numeric values
    filtered_loans.mean(numeric_only =True)
    filtered_loans.fillna(filtered_loans.mean(numeric_only =True),
        inplace=True)

    #12.2 Categorical values
    # For 'mode', there can be many outputs, so we pic the first one.
    filtered_loans_v1.fillna(filtered_loans_v1.mode().iloc[0], inplace=True)

    # 12.3 Simple Imputer Mean
    from sklearn.impute import SimpleImputer
    num_imp=SimpleImputer(missing_values=np.NaN, strategy='mean')
    filtered_loans[['revol_util','pub_rec_bankruptcies']]=
        pd.DataFrame(num_imp.fit_transform(filtered_loans_num))

    # 12.4 Simple Imputer Mode
    cat_imp=SimpleImputer(missing_values=np.NaN, strategy='most_frequent')
    filtered_loans['emp_length']=
        pd.DataFrame(cat_imp.fit_transform(filtered_loans_cat))
```

```
# 13. Split Train Test ========================================================
    from sklearn.model_selection import train_test_split
    y = loans.pop('loan_status')
    X = loans
    X_train,X_test,y_train,y_test =
        train_test_split(X,y,stratify=y,test_size=0.25, random_state=42)

# 14. Remove Outlier with Log =================================================
    # Remove any zeros (otherwise we get (-inf))
    # Alternative: we can +1 before taking log!!!
    df.loc[df.Fare == 0, 'Fare'] = np.nan
    df.dropna(inplace=True)

    # Log Transform
    df['Log_' + i] = np.log(df[i])

    # Find Quartiles
    q75, q25 = np.percentile(df.Log_Fare.dropna(), [75 ,25])
    iqr = q75 - q25
    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)

    # create variable 'Outlier'
    df['Outlier'] = 0
    df.loc[df[i] < min, 'Outlier'] = 1
    df.loc[df[i] > max, 'Outlier'] = 1

# 15. Decision Trees ==========================================================
    # 15.1 Decision Tree
    from sklearn.tree import DecisionTreeClassifier
    dtree = DecisionTreeClassifier(min_samples_leaf=10, criterion='entropy')
    dtree.fit(X_train,y_train)

    # 15.2 Classification Report
    from sklearn.metrics import classification_report,confusion_matrix
    predictions = dtree.predict(X_test)
    print(classification_report(y_test,predictions,digits=4))

    # 15.3 Confusion Matrix
    print(confusion_matrix(y_test,predictions,labels=['absent','present']))

#16. Linear Regression ========================================================
    # 16.1 Linear Regression
    from sklearn.linear_model import LinearRegression
    lm = LinearRegression()
    lm.fit(X_train,y_train)
    lm
    lm.intercept_
    lm.n_features_in_
    lm.feature_names_in_
    predictions = lm.predict(X_test)

    # 16.2 Metrics
    from sklearn import metrics
    print('MAE:', metrics.mean_absolute_error(y_test, predictions))
    print('MSE:', metrics.mean_squared_error(y_test, predictions))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```