PRACTICAL 1

AIM: Introduction to Excel

1. **Perform conditional formatting on a dataset using various criteria.**
2. **Create a pivot table to analyze and summarize data.**
3. **Use VLOOKUP function to retrieve information from a different worksheet or table.**
4. **Perform what-if analysis using Goal Seek to determine input values for desired output.**

Microsoft Excel is a powerful spreadsheet application that helps in managing, analyzing, and visualizing data effectively. It offers a range of functionalities for data manipulation, analysis, and reporting.

1. Perform Conditional Formatting on a Dataset Using Various Criteria

Conditional formatting allows users to visually highlight important data points or trends in a dataset based on specific rules.

Steps:

1. Select the dataset.
2. Go to the Home tab and click on Conditional Formatting in the Styles group.
3. Choose a rule type (e.g., Highlight Cells Rules, Top/Bottom Rules, Data Bars, Color Scales, or Icon Sets).
4. Set the criteria (e.g., values greater than 5000, specific text, or duplicate values).
5. Apply the desired formatting style.

Example: Highlight sales figures above 5000 in green and below 5,000 in red.

2. Create a Pivot Table to Analyze and Summarize Data

A Pivot Table is a tool used for summarizing large datasets dynamically.

Steps:

1. Select the dataset and go to the Insert tab.
2. Click on Pivot Table.
3. Choose the data range and destination for the Pivot Table (new or existing worksheet).
4. Drag fields into the Rows, Columns, Values, and Filters areas as needed.
5. Use filters or sort options for deeper analysis.

3. Use VLOOKUP Function to Retrieve Information from a Different Worksheet or Table

The VLOOKUP function searches for a value in a column and returns a value in the same row from another column.

Syntax: =VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])

Steps:

1. Identify the lookup value and the table where the data exists.
2. Enter the VLOOKUP formula in the desired cell.
3. Specify the column index from which to retrieve the value.
4. Use TRUE for an approximate match or FALSE for an exact match.

4. Perform What-If Analysis Using Goal Seek to Determine Input Values for Desired Output

Goal Seek is a tool for backward solving by adjusting an input value to achieve a specific output.

Steps:

1. Go to the Data tab and select What-If Analysis, then choose Goal Seek.
2. In the Goal Seek dialog box:
3. Set Set Cell: The cell containing the formula/result.
4. Set To Value: The desired outcome.
5. Set By Changing Cell: The cell with the input value to adjust.
6. Click OK to run the analysis.

PRACTICAL 2

AIM: Data Frames and Basic Data Pre-processing

- **Read data from CSV and JSON files into a data frame.**
- **Perform basic data pre-processing tasks such as handling missing values and outliers.**
- **Manipulate and transform data using functions like filtering, sorting, and**
- **grouping.**

```
import pandas as pd
#Reading a CSV file
df_csv = pd.read_csv('iris.csv')
print(df_csv.head())

# Reading a JSON file
df_json = pd.read_json('iris.json')
print(df_json.head())

print(df_csv.isnull().sum())
print(df_csv.head())
df_csv['SepalWidthCm'].fillna(value=3,inplace=True)  # Replace with a specific value
print(df_csv.head())

df_csv.dropna(inplace=True)
print(df_csv.head())

Q1 = df_csv['SepalWidthCm'].quantile(0.25)
Q3 = df_csv['SepalWidthCm'].quantile(0.75)

IQR = Q3 - Q1
df = df_csv[(df_csv['SepalWidthCm'] >= (Q1 - 1.5 * IQR)) & (df_csv['SepalWidthCm'] <= (Q3 + 1.5 * IQR))]
print("Outliers are: \n")
print(df)

filtered_df = df_csv[df_csv['SepalLengthCm'] > 5.0]
print(filtered_df)

sorted_df = df_csv.sort_values(by='PetalWidthCm', ascending=True)
print(sorted_df)

grouped_df = df_csv.groupby('Species')['PetalWidthCm'].sum()
print(grouped_df)
```

PRACTICAL 3

AIM: Feature Scaling and Dummification

- **Apply feature-scaling techniques like standardization and normalization to numerical features.**
- **Perform feature dummification to convert categorical variables into numerical representations.**

```
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import pandas as pd
# Load Titanic Dataset
titanic = sns.load_dataset("titanic")
# Drop rows with missing values for simplicity
titanic = titanic.dropna()
# Identify numerical and categorical columns
numerical_columns = ['age', 'fare']
categorical_columns = ['sex', 'embarked', 'class']
# Standardization
standard_scaler = StandardScaler()
titanic_standardized = titanic.copy()
titanic_standardized[numerical_columns] = standard_scaler.fit_transform
(titanic[numerical_columns])
# Normalization
min_max_scaler = MinMaxScaler()
titanic_normalized = titanic.copy()
titanic_normalized[numerical_columns] = min_max_scaler.fit_transform
(titanic[numerical_columns])
# Dummification
titanic_dummified = pd.get_dummies(titanic, columns=categorical_columns,
                          drop_first=True)

# Save Results
titanic_standardized.to_csv("titanic_standardized.csv", index=False)
titanic_normalized.to_csv("titanic_normalized.csv", index=False)
titanic_dummified.to_csv("titanic_dummified.csv", index=False)

# Display Results
print("Standardized Data:\n", titanic_standardized.head())
print("\nNormalized Data:\n", titanic_normalized.head())
print("\nDummified Data:\n", titanic_dummified.head())
```

PRACTICAL 4

AIM: Hypothesis Testing

- **Formulate null and alternative hypotheses for a given problem.**
- **Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chi□square test).**
- **Interpret the results and draw conclusions based on the test outcomes.**

```
import numpy as np
from scipy.stats import ttest_1samp, ttest_ind, chi2_contingency
# One-Sample t-Test
sample_data = [2.7, 2.0, 2.8, 2.6, 2.0]  # Sample resolution times
population_mean = 2  # Null hypothesis: mean = 2 days
t, p= ttest_1samp(sample_data, population_mean)
print("\n--- One-Sample t-Test ---")
print(f"T-statistic: {t:.2f}, P-value: {p:.4f}")
if p < 0.05:
    print("Reject H0")
else:
    print("Fail to reject H0")
sample1 = np.random.normal(loc=10, scale=2, size=15)  # Class A scores
sample2 = np.random.normal(loc=12, scale=2, size=15)  # Class B scores
t, p = ttest_ind(sample1, sample2)
print("\n--- Two-Sample t-Test ---")
print(f"T-statistic: {t:.2f}, P-value: {p:.4f}")
if p < 0.05:
    print("Reject H0")
else:
    print("Fail to reject H0")
observed = np.array([
    [30, 20],  # Male preferences
    [40, 10]   # Female preferences
])
chi, p, dof, expected = chi2_contingency(observed)
print("\n--- Chi-Square Test for Independence ---")
print(f"Chi-square Statistic: {chi:.2f}, P-value: {p:.4f}")

if p < 0.05:
    print("Conclusion:", "Reject H0")
else:
    print("Fail to reject H0")
```

PRACTICAL 5

AIM: ANOVA (Analysis of Variance)

- **Perform one-way ANOVA to compare means across multiple groups.**
- **Conduct post-hoc tests to identify significant differences between group means.**

```
import pandas as pd
import scipy.stats as stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd

group1 = [23, 25, 29, 34, 30]
group2 = [19, 20, 22, 24, 25]
group3 = [15, 18, 20, 21, 17]
group4 = [28, 24, 26, 30, 29]
# Combine data into a DataFrame
data = pd.DataFrame({'value': group1 + group2 + group3 + group4,
 'group': ['Group1'] * len(group1) + ['Group2'] * len(group2) +
 ['Group3'] * len(group3) + ['Group4'] * len(group4)})
# Perform one-way ANOVA
f_statistics, p_value = stats.f_oneway(group1, group2, group3, group4)
print("one-way ANOVA:")
print("F-statistics:", f_statistics)
print("p-value", p_value)
# Perform Tukey-Kramer post-hoc test
tukey_results = pairwise_tukeyhsd(data['value'], data['group'])
print("\nTukey-Kramer post-hoc test:")
print(tukey_results)
```

PRACTICAL 6

AIM: Regression and Its Types

- **Implement simple linear regression using a dataset.**
- **Explore and interpret the regression model coefficients and goodness-of-fit measures.**
- **Extend the analysis to multiple linear regression and assess the impact of additional predictors.**

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
housing = fetch_california_housing()
housing_df = pd.DataFrame(housing.data,columns=housing.feature_names)
print(housing_df)
housing_df['PRICE'] = housing.target
X =housing_df[['AveRooms']]
y = housing_df['PRICE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                        random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
mse = mean_squared_error(y_test, model.predict(X_test))
r2 = r2_score(y_test, model.predict(X_test))
print("Mean Squared Error:",mse)
print("R-squared:", r2)
print("Intercept:", model.intercept_)
print("Coefficient:",  model.coef_)
#Multiple Linear Regression
X =housing_df.drop('PRICE',axis=1)
y = housing_df['PRICE']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,
                                        random_state=42)
model = LinearRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test,y_pred)
r2 = r2_score(y_test,y_pred)
print("Mean Squared Error:",mse)
print("Rsquared:",r2)
print("Intercept:",model.intercept_)
print("Coefficient:",model.coef_)
```

PRACTICAL 7

AIM: Logistic Regression and Decision Tree

- **Build a logistic regression model to predict a binary outcome.**
- **Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).**
- **Construct a decision tree model and interpret the decision rules for classification.**

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score,recall_score,
classification_report
# Load the Iris dataset and create a binary classification problem
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
                  columns=iris['feature_names'] +['target'])
binary_df =iris_df[iris_df['target'] != 2]
X =binary_df.drop('target', axis=1)
y = binary_df['target']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                             random_state=42)
# Train a logistic regression model and evaluate its performance
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
y_pred_logistic= logistic_model.predict(X_test)
p
print("Logistic Regression Metrics")
print("Accuracy: ", accuracy_score(y_test,y_pred_logistic))
print("Precision:", precision_score(y_test, y_pred_logistic))
print("Recall: ", recall_score(y_test,y_pred_logistic))
print("\nClassification Report")
print(classification_report(y_test, y_pred_logistic))

# Train a decision tree model and evaluate its performance
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train, y_train)
y_pred_tree = decision_tree_model.predict(X_test)
print("\nDecision Tree Metrics")
print("Accuracy: ", accuracy_score(y_test, y_pred_tree))
print("Precision:", precision_score(y_test, y_pred_tree))
print("Recall: ", recall_score(y_test, y_pred_tree))
print("\nClassification Report")
print(classification_report(y_test, y_pred_tree))
```

PRACTICAL 8

AIM: K-Means Clustering

- **Apply the K-Means algorithm to group similar data points into clusters.**
- **Determine the optimal number of clusters using elbow method or silhouette analysis.**
- **Visualize the clustering results and analyze the cluster characteristics.**

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
data =
pd.read_csv("C:\\Users\\Mathe\\OneDrive\\Documents\\SLRTDC\\CS\\TY\\DS\\Practical\\wholesale.csv")
print(data.head())
categorical_features = ['Channel', 'Region']
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen',
                       'Detergents_Paper', 'Delicassen']
print(data[continuous_features].describe())
for col in categorical_features:
    dummies = pd.get_dummies(data[col], prefix=col)
    data = pd.concat([data, dummies], axis=1)
    data.drop(col, axis=1, inplace=True)
print(data.head())
mms = MinMaxScaler()
mms.fit(data)
data_transformed = mms.transform(data)
sum_of_squared_distances = []
K = range(1, 15)
for k in K:
    km = KMeans(n_clusters=k)
    km.fit(data_transformed)  # You need to fit the model to the data first
    sum_of_squared_distances.append(km.inertia_)  # Correct: inertia_ is an attribute, not a method
plt.plot(K, sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal k')
plt.show()
optimal_k = 4  # Replace this with the value you choose based on the elbow method
km = KMeans(n_clusters=optimal_k)
km.fit(data_transformed)
```