In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```python
data_df = pd.read_csv('supermarket_sales.csv')
data_df
```

Out[2]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Total	
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	548.9715	1/5
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	80.2200	3/8
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	340.5255	3/3
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	489.0480	1/27
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	634.3785	2/8
...	...	...	...	...	...	...	...	...	...	
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	42.3675	1/29
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	1022.4900	3/2
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	33.4320	2/9
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	69.1110	2/22
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	649.2990	2/18

1000 rows × 16 columns

In [3]:
```python
data_df.describe()
```

Out[3]:

	Unit price	Quantity	Total	cogs	gross margin percentage	gross income	Rating
count	1000.000000	1000.000000	1000.000000	1000.00000	1.000000e+03	1000.000000	1000.00000
mean	55.672130	5.510000	322.966749	307.58738	4.761905e+00	15.379369	6.97270
std	26.494628	2.923431	245.885335	234.17651	6.131498e-14	11.708825	1.71858
min	10.080000	1.000000	10.678500	10.17000	4.761905e+00	0.508500	4.00000
25%	32.875000	3.000000	124.422375	118.49750	4.761905e+00	5.924875	5.50000
50%	55.230000	5.000000	253.848000	241.76000	4.761905e+00	12.088000	7.00000
75%	77.935000	8.000000	471.350250	448.90500	4.761905e+00	22.445250	8.50000
max	99.960000	10.000000	1042.650000	993.00000	4.761905e+00	49.650000	10.00000

In [4]:

```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   City                     1000 non-null   object
 3   Customer type            1000 non-null   object
 4   Gender                   1000 non-null   object
 5   Product line             1000 non-null   object
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64
 8   Total                    1000 non-null   float64
 9   Date                     1000 non-null   object
 10  Time                     1000 non-null   object
 11  Payment                  1000 non-null   object
 12  cogs                     1000 non-null   float64
 13  gross margin percentage  1000 non-null   float64
 14  gross income             1000 non-null   float64
 15  Rating                   1000 non-null   float64
dtypes: float64(6), int64(1), object(9)
memory usage: 125.1+ KB
```

Prepocessing

In [5]:

```
## For attribute 'Time' and 'Date'
from pandas import DatetimeIndex as dt
day_of_week = pd.to_datetime(data_df['Date']).dt.dayofweek
data_df['Date'] = day_of_week

hour = pd.to_datetime(data_df['Time']).dt.strftime('%H').astype('float')
hour_cat = pd.cut(hour, bins=[0.0, 12.0, 16.0, 18.0, np.inf], labels=[1,2,3,4])
data_df['Time'] = hour_cat

data_df
```

Out[5]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Total	Dat
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	548.9715	
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	80.2200	
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	340.5255	
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	489.0480	
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	634.3785	
...	...	...	...	...	...	...	...	...	...	.
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	42.3675	
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	1022.4900	
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	33.4320	
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	69.1110	
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	649.2990	

1000 rows × 16 columns

In [6]:
```
data_df.to_csv('data.csv', index=False)
```

In [7]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

# Atrribute 'Invoice ID' and 'gross margin percentage' are useless for training and
X_df = data_df.drop(labels=['Invoice ID', 'gross margin percentage'], axis=1)

num_attribute = ['Unit price', 'Quantity', 'Total', 'cogs', 'gross income', 'Rating'
cat_1hot_attribute = ['Branch', 'City', 'Product line', 'Date', 'Time', 'Payment']
cat_ordin_attribute = ['Customer type', 'Gender']

pre_pipeline = ColumnTransformer([('num', StandardScaler(), num_attribute),
```

```
                                  ('cat_1hot', OneHotEncoder(), cat_1hot_attribute),
                                  ('cat_ordin', OrdinalEncoder(), cat_ordin_attribute

train, test = train_test_split(X_df,
                               test_size=0.2,
                               random_state=8) ## In this project, all random states

train_prepared = pre_pipeline.fit_transform(train)
test_prepared = pre_pipeline.transform(test)

Column = ['Unit price', 'Quantity', 'Total', 'cogs', 'gross income', 'Rating',
         'Branch A', 'Branch B', 'Branch C', 'Mandalay', 'Naypyitaw', 'Yangon',
          'Electronic accessories', 'Fashion accessories', 'Food and beverages', 'He
         'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday
         'Morning', 'Afternoon', 'Evening', 'Night',
         'Cash', 'Credit card', 'Ewallet',
         'Customer type', 'Gender']

train_prepared_df = pd.DataFrame(train_prepared,
                                columns=Column)
test_prepared_df = pd.DataFrame(test_prepared,
                                columns=Column)
train_prepared_df
```

Out[7]:

	Unit price	Quantity	Total	cogs	gross income	Rating	Branch A	Branch B	Branch C	M
0	0.360863	-0.504378	-0.189186	-0.189186	-0.189186	0.430554	1.0	0.0	0.0	
1	-0.798463	-1.178004	-1.005079	-1.005079	-1.005079	-0.154338	0.0	0.0	1.0	
2	-0.924559	0.842875	-0.258357	-0.258357	-0.258357	0.430554	0.0	1.0	0.0	
3	-0.228586	-1.514817	-1.083733	-1.083733	-1.083733	-1.499590	0.0	0.0	1.0	
4	1.183684	-1.178004	-0.553150	-0.553150	-0.553150	0.723000	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	
795	-1.502716	-1.178004	-1.165649	-1.165649	-1.165649	-1.558079	0.0	1.0	0.0	
796	1.233369	0.169249	0.959175	0.959175	0.959175	1.073935	0.0	1.0	0.0	
797	1.497606	0.842875	1.950657	1.950657	1.950657	1.249403	0.0	0.0	1.0	
798	-0.248159	-0.841191	-0.673255	-0.673255	-0.673255	0.489043	0.0	1.0	0.0	
799	-1.229823	0.169249	-0.725648	-0.725648	-0.725648	0.372064	0.0	1.0	0.0	

800 rows × 34 columns

In [8]:

```
train_prepared_df.to_csv('train_prepared.csv', index=False)
test_prepared_df.to_csv('test_prepared.csv', index=False)
```

Problem 2: Train a multiple linear regression with and without Lasso regularization to predict gross income.

In [9]:

```
# Problem 2: Train a multiple linear regression with and without Lasso regularizatio
from sklearn.linear_model import LinearRegression, Lasso
```

```python
from sklearn.model_selection import train_test_split

X_train = train_prepared_df[['Unit price', 'Quantity',
                        'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturd
                         'Morning', 'Afternoon', 'Evening', 'Night',
                         'Electronic accessories', 'Fashion accessories', 'Food and bev
t_train = train_prepared_df['gross income'].copy()

lin_reg = LinearRegression()
lasso_reg = Lasso()
```

In [10]:
```python
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import joblib

## Linear regression without Lasso regularization
lin_reg.fit(X_train, t_train)
joblib.dump(lin_reg, 'Model/lin_reg_problem2.pkl')
```

Out[10]:
```
['Model/lin_reg_problem2.pkl']
```

In [11]:
```python
## Linear regression with Lasso regularization
lasso_reg.get_params()
```

Out[11]:
```
{'alpha': 1.0,
 'copy_X': True,
 'fit_intercept': True,
 'max_iter': 1000,
 'normalize': 'deprecated',
 'positive': False,
 'precompute': False,
 'random_state': None,
 'selection': 'cyclic',
 'tol': 0.0001,
 'warm_start': False}
```

In [12]:
```python
from scipy.stats import expon

lamb = np.linspace(0.001,0.2,1000)
lamb_distribution = expon(scale = 1/10)
param_grid = {'alpha': lamb}
param_distribution = {'alpha': lamb_distribution}

gridcv_lasso = GridSearchCV(lasso_reg,
                            param_grid=param_grid,
                            cv=10,
                            scoring='neg_mean_squared_error')

randomcv_lasso = RandomizedSearchCV(lasso_reg,
                                    param_distributions=param_distribution,
                                    cv=10,
                                    random_state=8,
                                    scoring='neg_mean_squared_error')
```

In [13]:
```python
gridcv_lasso.fit(X_train, t_train)
gridcv_lasso.best_params_
```

Out[13]:
```
{'alpha': 0.004983983983983846}
```

In [14]:
```python
randomcv_lasso.fit(X_train, t_train)
randomcv_lasso.best_params_
```

Out[14]:
```
{'alpha': 0.0011464268599014138}
```

In [15]:
```
## Choose the result from GridsearchCV
joblib.dump(gridcv_lasso.best_estimator_, 'Model/lasso_reg_grid_problem2.pkl')
joblib.dump(randomcv_lasso.best_estimator_, 'Model/lasso_reg_random_problem2.pkl')
```

Out[15]:
```
['Model/lasso_reg_random_problem2.pkl']
```

Problem 3: Train a multiple linear regression with and without Lasso regularization to predict Unit price.

In [16]:
```
# Problem 3: Train a multiple linear regression with and without Lasso regularizatio
X_train = train_prepared_df[['gross income', 'Quantity',
                     'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturd
                      'Morning', 'Afternoon', 'Evening', 'Night',
                      'Electronic accessories', 'Fashion accessories', 'Food and bev
t_train = train_prepared_df['Unit price'].copy()

lin_reg = LinearRegression()
lasso_reg = Lasso()
```

In [17]:
```
## Linear regression without Lasso regularization
lin_reg.fit(X_train, t_train)
joblib.dump(lin_reg, 'Model/lin_reg_problem3.pkl')
```

Out[17]:
```
['Model/lin_reg_problem3.pkl']
```

In [18]:
```
## Linear regression without Lasso regularization
param_grid = {'alpha': lamb}
param_distribution = {'alpha': lamb_distribution}

gridcv_lasso = GridSearchCV(lasso_reg,
                         param_grid=param_grid,
                         cv=10,
                         scoring='neg_mean_squared_error')

randomcv_lasso = RandomizedSearchCV(lasso_reg,
                                 param_distributions=param_distribution,
                                 cv=10,
                                 random_state=8,
                                 scoring='neg_mean_squared_error')
```

In [19]:
```
gridcv_lasso.fit(X_train, t_train)
gridcv_lasso.best_params_
```

Out[19]:
```
{'alpha': 0.006179179179179179}
```

In [20]:
```
randomcv_lasso.fit(X_train, t_train)
randomcv_lasso.best_params_
```

Out[20]:
```
{'alpha': 0.0011464268599014138}
```

In [21]:
```
## Choose the result from RandomsearchCV because of more accurate alpha
joblib.dump(gridcv_lasso.best_estimator_, 'Model/lasso_reg_grid_problem3.pkl')
joblib.dump(randomcv_lasso.best_estimator_, 'Model/lasso_reg_random_problem3.pkl')
```

Out[21]:
```
['Model/lasso_reg_random_problem3.pkl']
```

Problem 4

In [22]:
```
# Problem 4:
# Train a logistic regression to classify gender and study the relationship between
# Namely, explain the relationship between gender, product line, payment and gross i
# To study this relationship, consider all the interaction attribution of degree 2.

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

X_train = train_prepared_df[['Electronic accessories', 'Fashion accessories', 'Food
                              'Cash', 'Credit card', 'Ewallet',
                              'gross income']].to_numpy()
X_train = X_train[np.where(train_prepared_df['Branch C']==1)]
t_train = train_prepared_df['Gender'].to_numpy()
t_train = t_train[np.where(train_prepared_df['Branch C']==1)]

X_train.shape, t_train.shape
```

Out[22]:
```
((270, 10), (270,))
```

In [23]:
```
pipe = Pipeline([('poly_feature', PolynomialFeatures(degree=2, interaction_only=Tru
                 ('log_reg', LogisticRegression(fit_intercept=True))])
```

In [24]:
```
pipe.fit(X_train, t_train)
```

Out[24]:
```
▸ Pipeline
  ▸ PolynomialFeatures
  ▸ LogisticRegression
```

In [25]:
```
joblib.dump(pipe, 'Model/pipeline_problem4.pkl')
```

Out[25]:
```
['Model/pipeline_problem4.pkl']
```

Problem 5

In [26]:
```
# Train a logistic regression to classify customer type and study the relationship b
# Namely, explain the relationship between customer type, gender, day and timeslot f
# To study this relationship, consider all the interaction attribution of degree 2.

X_train = train_prepared_df[['Gender', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
X_train = X_train[np.where(train_prepared_df['Branch C']==1)]
t_train = train_prepared_df['Customer type'].to_numpy()
t_train = t_train[np.where(train_prepared_df['Branch C']==1)]

X_train.shape, t_train.shape
```

Out[26]:
```
((270, 12), (270,))
```

In [27]:
```
pipe = Pipeline([('poly_feature', PolynomialFeatures(degree=2, interaction_only=Tru
                 ('log_reg', LogisticRegression(fit_intercept=True))])
```

In [28]:
```
pipe.fit(X_train, t_train)
```

Out[28]:

```
▸ Pipeline
  ▸ PolynomialFeatures
  ▸ LogisticRegression
```

In [29]:

```
joblib.dump(pipe, 'Model/pipeline_problem5.pkl')
```

Out[29]:

```
['Model/pipeline_problem5.pkl']
```

Problem 6: Train a classifier to predict the day of purchase

In [30]:

```
train_prepared_df.index
```

Out[30]:

```
RangeIndex(start=0, stop=800, step=1)
```

In [31]:

```
# Problem 6: Train a classifier to predict the day of purchase
## In problem 6, we don't want to apply one hot encoder on attribute 'Date', so we k
## and do the prepocessing again.
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

num_attribute = ['Unit price', 'Quantity', 'Total', 'cogs', 'gross income', 'Rating'
cat_1hot_attribute = ['Branch', 'City', 'Product line', 'Time', 'Payment']
cat_ordin_attribute = ['Customer type', 'Gender']

pre_pipeline_problem6 = ColumnTransformer([('num', StandardScaler(), num_attribute),
                                  ('cat_1hot', OneHotEncoder(), cat_1hot_attribute),
                                  ('cat_ordin', OrdinalEncoder(), cat_ordin_attribute

X = X_df.drop('Date', axis=1)
t = X_df['Date'].copy()
X_train, X_test, t_train, t_test = train_test_split(X, t,
                                                    test_size=0.2,
                                                    random_state=8,
                                                    stratify=t)

X_train_prepared = pre_pipeline_problem6.fit_transform(X_train)
X_test_prepared = pre_pipeline_problem6.transform(X_test)

Column = ['Unit price', 'Quantity', 'Total', 'cogs', 'gross income', 'Rating',
          'Branch A', 'Branch B', 'Branch C', 'Mandalay', 'Naypyitaw', 'Yangon',
           'Electronic accessories', 'Fashion accessories', 'Food and beverages', 'He
          'Morning', 'Afternoon', 'Evening', 'Night',
          'Cash', 'Credit card', 'Ewallet',
          'Customer type', 'Gender']

X_train_df = pd.DataFrame(X_train_prepared, columns=Column)
X_test_df = pd.DataFrame(X_test_prepared, columns=Column)
```

In [32]:

```
X_train_df.to_csv('Problem 6 data/X_train.csv', index=False)
X_test_df.to_csv('Problem 6 data/X_test.csv', index=False)
t_train.to_csv('Problem 6 data/t_train.csv', index=False)
t_test.to_csv('Problem 6 data/t_test.csv', index=False)
```

In [33]:

```
## First model: LogisticRegression Classifier with Ridge regularization
```

```
log_reg = LogisticRegression(multi_class='multinomial', penalty='l2')
log_reg.get_params()
```

Out[33]:

```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'multinomial',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [34]:

```
C = np.linspace(0.1, 100, 1000)
param_grid = {'C': C}
gridcv_log_reg = GridSearchCV(log_reg,
                              param_grid=param_grid,
                              cv=10,
                              scoring='accuracy')
```

In [35]:

```
gridcv_log_reg.fit(X_train_prepared, t_train)
```

Out[35]:

```
▸ GridSearchCV
▸ estimator: LogisticRegression
▸ LogisticRegression
```

In [36]:

```
gridcv_log_reg.best_params_
```

Out[36]:

```
{'C': 0.7000000000000001}
```

In [37]:

```
joblib.dump(gridcv_log_reg.best_estimator_, 'Model/log_reg_problem6.pkl')
```

Out[37]:

```
['Model/log_reg_problem6.pkl']
```

In [38]:

```
## Second model: Decisiontree Classifier
tree = DecisionTreeClassifier()
tree.get_params()
```

Out[38]:

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

In [39]:

```
max_depth = list(range(1,31))
param_grid = {'criterion': ['gini', "entropy", "log_loss"],
```

```
            'max_depth': max_depth,
            'min_samples_split': list(range(1,10)),
            'min_samples_leaf': list(range(2,10))}
gridcv_tree = GridSearchCV(tree,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=10)
```

In [40]:
```
gridcv_tree.fit(X_train_prepared, t_train)
```

```
/apps/python/3.10/lib/python3.10/site-packages/sklearn/model_selection/_validation.p
y:378: FitFailedWarning:
7200 fits failed out of a total of 64800.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
7200 fits failed with the following error:
Traceback (most recent call last):
  File "/apps/python/3.10/lib/python3.10/site-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/apps/python/3.10/lib/python3.10/site-packages/sklearn/tree/_classes.py", li
ne 969, in fit
    super().fit(
  File "/apps/python/3.10/lib/python3.10/site-packages/sklearn/tree/_classes.py", li
ne 265, in fit
    check_scalar(
  File "/apps/python/3.10/lib/python3.10/site-packages/sklearn/utils/validation.py",
line 1480, in check_scalar
    raise ValueError(
ValueError: min_samples_split == 1, must be >= 2.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/apps/python/3.10/lib/python3.10/site-packages/sklearn/model_selection/_search.py:95
3: UserWarning: One or more of the test scores are non-finite: [    nan 0.1625   0.16
25  ... 0.1475  0.14625 0.1475 ]
  warnings.warn(
```

Out[40]:
```
▸          GridSearchCV
▸ estimator: DecisionTreeClassifier
      ▸ DecisionTreeClassifier
```

In [41]:
```
gridcv_tree.best_params_
```

Out[41]:
```
{'criterion': 'entropy',
 'max_depth': 11,
 'min_samples_leaf': 5,
 'min_samples_split': 3}
```

In [42]:
```
joblib.dump(gridcv_tree.best_estimator_, 'Model/tree_problem6.pkl')
```

Out[42]:
```
['Model/tree_problem6.pkl']
```

In [43]:
```
## Third model: Randomforest Classifier
random_forest = RandomForestClassifier()
random_forest.get_params()
```

Out[43]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [44]:
```
num_of_tree = np.arange(50,501,50)
param_grid = {'criterion': ["gini", "entropy", "log_loss"],
              'n_estimators': num_of_tree}
gridcv_random_forest = GridSearchCV(random_forest,
                                    param_grid=param_grid,
                                    cv=10,
                                    scoring='accuracy')
```

In [45]:
```
gridcv_random_forest.fit(X_train_prepared, t_train)
```

Out[45]:
```
▸            GridSearchCV
▸ estimator: RandomForestClassifier
       ▸ RandomForestClassifier
```

In [46]:
```
gridcv_random_forest.best_params_
```

Out[46]:
```
{'criterion': 'log_loss', 'n_estimators': 200}
```

In [47]:
```
joblib.dump(gridcv_random_forest.best_estimator_, 'Model/random_forest_problem6.pkl'
```

Out[47]:
```
['Model/random_forest_problem6.pkl']
```

In [ ]: