SkillSanta Project

On

“Understanding Customer’s Behaviour In A Retail Shop”

Submitted By –

Ankita Prasad

DSML Batch 1

Submitted To –

Mr. Neeraj Garg

Contents

1. Reading Data and EDA

We are given with two datasets,

(i) train.csv
(ii) test.csv

First, read the data into df_train_org and df_test dataframes using read_csv.

*df_train consists all data except 'Purchase' values, which will be used later on for training purpose.

```
data_path = '/Users/hp/Desktop/SkillSanta Project/Data'
train_file = 'train.csv'
test_file = 'test.csv'

#Using "os.path.join" because it is platform independent,
#could have also used "+" sign to concat file names with path
df_train_org = pd.read_csv(os.path.join(data_path, train_file))
df_test = pd.read_csv(os.path.join(data_path, test_file))
df_train = df_train_org.iloc[:,:-1]
```

Look at the data in the dataframes by df_train_org.head() and df_test.head(). This will show first 5 rows of the dataframes.

```
df_train_org.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Produc
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	

```
df_test.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Produc
0	1000004	P00128942	M	46-50	7	B	2	1	1	11.0	
1	1000009	P00113442	M	26-35	17	C	0	0	3	5.0	
2	1000010	P00288442	F	36-45	1	B	4+	1	5	14.0	
3	1000010	P00145342	F	36-45	1	B	4+	1	4	9.0	
4	1000011	P00053842	F	26-35	1	C	1	0	4	5.0	

By using a function to get detailed information about the dataframe features, explore and analyse the dataframe.

The function returns a dataframe consisting information, and total memory used by that dataframe.

```
#Function to get informations about the DF
def get_df_info(df, include_unique_values = False):
    col_name = list(df.columns)
    col_type = [type(df[col][0]) for col in col_name]
    col_null_count = [df[col].isnull().sum() for col in col_name]
    col_unique_count = [df[col].nunique() for col in col_name]
    col_mem_usage = [df[col].memory_usage(deep = True) for col in col_name]
    df_total_mem = sum(col_mem_usage) / 1048576
    if include_unique_values:
        col_unique_list = [df[col].unique() for col in col_name]
        df_info = pd.DataFrame({'column_name': col_name,
                                'type': col_type,
                                'null_count': col_null_count,
                                'nunique': col_unique_count,
                                'unique_values': col_unique_list})
    else:
        df_info = pd.DataFrame({'column_name': col_name,
                                'type': col_type,
                                'null_count': col_null_count,
                                'nunique': col_unique_count})
    return df_info, df_total_mem
```

For df_train_org –

```
df_train_org_info, df_train_org_total_mem = get_df_info(df_train_org, True)
print(df_train_org_total_mem)
df_train_org_info
```

```
193.79075241088867
```

	column_name	type	null_count	nunique	unique_values
0	User_ID	<class 'numpy.int64'>	0	5891	[1000001, 1000002, 1000003, 1000004, 1000005, ...
1	Product_ID	<class 'str'>	0	3631	[P00069042, P00248942, P00087842, P00085442, P...
2	Gender	<class 'str'>	0	2	[F, M]
3	Age	<class 'str'>	0	7	[0-17, 55+, 26-35, 46-50, 51-55, 36-45, 18-25]
4	Occupation	<class 'numpy.int64'>	0	21	[10, 16, 15, 7, 20, 9, 1, 12, 17, 0, 3, 4, 11,...
5	City_Category	<class 'str'>	0	3	[A, C, B]
6	Stay_In_Current_City_Years	<class 'str'>	0	5	[2, 4+, 3, 1, 0]
7	Marital_Status	<class 'numpy.int64'>	0	2	[0, 1]
8	Product_Category_1	<class 'numpy.int64'>	0	20	[3, 1, 12, 8, 5, 4, 2, 6, 14, 11, 13, 15, 7, 1...
9	Product_Category_2	<class 'numpy.float64'>	173638	17	[nan, 6.0, 14.0, 2.0, 8.0, 15.0, 16.0, 11.0, 5...
10	Product_Category_3	<class 'numpy.float64'>	383247	15	[nan, 14.0, 17.0, 5.0, 4.0, 16.0, 15.0, 8.0, 9...
11	Purchase	<class 'numpy.int64'>	0	18105	[8370, 15200, 1422, 1057, 7969, 15227, 19215, ...

```
df_train_org.shape
```

```
(550068, 12)
```

For df_test –

In [12]:
```
df_test_info
```

Out[12]:

	column_name	type	null_count	nunique	unique_values
0	User_ID	<class 'numpy.int64'>	0	5891	[1000004, 1000009, 1000010, 1000011, 1000013, ...
1	Product_ID	<class 'str'>	0	3491	[P00128942, P00113442, P00288442, P00145342, P...
2	Gender	<class 'str'>	0	2	[M, F]
3	Age	<class 'str'>	0	7	[46-50, 26-35, 36-45, 18-25, 51-55, 55+, 0-17]
4	Occupation	<class 'numpy.int64'>	0	21	[7, 17, 1, 15, 3, 0, 8, 16, 4, 12, 13, 18, 11,...
5	City_Category	<class 'str'>	0	3	[B, C, A]
6	Stay_In_Current_City_Years	<class 'str'>	0	5	[2, 0, 4+, 1, 3]
7	Marital_Status	<class 'numpy.int64'>	0	2	[1, 0]
8	Product_Category_1	<class 'numpy.int64'>	0	18	[1, 3, 5, 4, 2, 10, 15, 18, 8, 13, 6, 11, 12, ...
9	Product_Category_2	<class 'numpy.float64'>	72344	17	[11.0, 5.0, 14.0, 9.0, 3.0, 4.0, 13.0, 2.0, na...
10	Product_Category_3	<class 'numpy.float64'>	162562	15	[nan, 12.0, 15.0, 9.0, 16.0, 14.0, 4.0, 3.0, 5...

We can observe –

(i) Datatypes of features are string, integer and float.
(ii) Missing values in Product_Category_2 and Product_Category_3.
(iii) Memory usage is quite high.

---

2. Data Wrangling

First, concatenate the df_train and df_test into one dataframe, df_concat_org.

In [15]:
```
#Concatinating train and test set into one dataset
df_concat = pd.concat([df_train, df_test], ignore_index = True, join = 'outer', sort = False)
df_concat
```

Out[15]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	P
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	
5	1000003	P00193542	M	26-35	15	A	3	0	1	2.0	
6	1000004	P00184942	M	46-50	7	B	2	1	1	8.0	
7	1000004	P00346142	M	46-	7	B	2	1	1	15.0	

[16]:
```
df_concat_info, df_concat_total_mem = get_df_info(df_concat, True)
print(df_concat_total_mem)
df_concat_info
```

270.1093740463257

[16]:

	column_name	type	null_count	nunique	unique_values
0	User_ID	<class 'numpy.int64'>	0	5891	[1000001, 1000002, 1000003, 1000004, 1000005, ...
1	Product_ID	<class 'str'>	0	3677	[P00069042, P00248942, P00087842, P00085442, P...
2	Gender	<class 'str'>	0	2	[F, M]
3	Age	<class 'str'>	0	7	[0-17, 55+, 26-35, 46-50, 51-55, 36-45, 18-25]
4	Occupation	<class 'numpy.int64'>	0	21	[10, 16, 15, 7, 20, 9, 1, 12, 17, 0, 3, 4, 11,...
5	City_Category	<class 'str'>	0	3	[A, C, B]
6	Stay_In_Current_City_Years	<class 'str'>	0	5	[2, 4+, 3, 1, 0]
7	Marital_Status	<class 'numpy.int64'>	0	2	[0, 1]
8	Product_Category_1	<class 'numpy.int64'>	0	20	[3, 1, 12, 8, 5, 4, 2, 6, 14, 11, 13, 15, 7, 1...
9	Product_Category_2	<class 'numpy.float64'>	245982	17	[nan, 6.0, 14.0, 2.0, 8.0, 15.0, 16.0, 11.0, 5...
10	Product_Category_3	<class 'numpy.float64'>	545809	15	[nan, 14.0, 17.0, 5.0, 4.0, 16.0, 15.0, 8.0, 9...

[17]:
```
df_concat.shape
```

[17]: (783667, 11)

2.1 Handling Missing Values -

For missing values in Product_Category_2 and Product_category_3, we will impute the mode value at missing place.

```
In [18]: df_concat['Product_Category_2'].mode()
Out[18]: 0    8.0
         dtype: float64

In [19]: df_concat['Product_Category_3'].mode()
Out[19]: 0    16.0
         dtype: float64
```

Create a copy (df_concat_copy) of the df_concat so that changes won't affect the original dataframe.

```
#Creating a copy of df so that changes won't affect in the original one
df_concat_copy = df_concat.copy(deep = True)
```

Impute mode values to fill missing values.

```
df_concat_copy.Product_Category_2.fillna(value = 8, inplace = True)
df_concat_copy.Product_Category_3.fillna(value = 16, inplace = True)
```

```
[22]: df_concat_copy_info, df_concat_copy_total_mem = get_df_info(df_concat_copy, True)
      print(df_concat_copy_total_mem)
      df_concat_copy_info

270.1093740463257
```

t[22]:

	column_name	type	null_count	nunique	unique_values
0	User_ID	<class 'numpy.int64'>	0	5891	[1000001, 1000002, 1000003, 1000004, 1000005, ...
1	Product_ID	<class 'str'>	0	3677	[P00069042, P00248942, P00087842, P00085442, P...
2	Gender	<class 'str'>	0	2	[F, M]
3	Age	<class 'str'>	0	7	[0-17, 55+, 26-35, 46-50, 51-55, 36-45, 18-25]
4	Occupation	<class 'numpy.int64'>	0	21	[10, 16, 15, 7, 20, 9, 1, 12, 17, 0, 3, 4, 11,...
5	City_Category	<class 'str'>	0	3	[A, C, B]
6	Stay_In_Current_City_Years	<class 'str'>	0	5	[2, 4+, 3, 1, 0]
7	Marital_Status	<class 'numpy.int64'>	0	2	[0, 1]
8	Product_Category_1	<class 'numpy.int64'>	0	20	[3, 1, 12, 8, 5, 4, 2, 6, 14, 11, 13, 15, 7, 1...
9	Product_Category_2	<class 'numpy.float64'>	0	17	[8.0, 6.0, 14.0, 2.0, 15.0, 16.0, 11.0, 5.0, 3...
10	Product_Category_3	<class 'numpy.float64'>	0	15	[16.0, 14.0, 17.0, 5.0, 4.0, 15.0, 8.0, 9.0, 1...

2.2 Conversion to Categorical Datatypes -

Now, we need to convert datatypes of all the features to numeric categorical values.

(i) User_ID has 5891 unique values.
(ii) Product_ID has 3677 unique values.
(iii) Gender has 2 values, M and F.
(iv) Age has 7 unique intervals.
(v) Occupation has 21 unique categories.
(vi) City_Category has 3 values, A, B and C.
(vii) Stay_In_Current_City_Years has 5 unique values.
(viii) Marital_Status has 2 unique values, 0 and 1.
(ix) Product_Category_1 has 20 unique values.
(x) Product_Category_2 has 17 unique values.
(xi) Product_Category_3 has 15 unique values.

Using Label Encoding, convert all of them to categorical datatype.

[35]:
```
from sklearn.preprocessing import LabelEncoder
for col in df_concat_copy_info['column_name']:
    col_encoder = LabelEncoder()
    df_concat_copy[col] = col_encoder.fit_transform(df_concat_copy[col])
```

[36]:
```
df_concat_copy_info, df_concat_copy_total_mem = get_df_info(df_concat_copy, True)
print(df_concat_copy_total_mem)
df_concat_copy_info
```

```
50.821529388427734
```

:[36]:

	column_name	type	null_count	nunique	unique_values
0	User_ID	<class 'numpy.int64'>	0	5891	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
1	Product_ID	<class 'numpy.int32'>	0	3677	[684, 2406, 868, 844, 2769, 1857, 1771, 3364, ...
2	Gender	<class 'numpy.int32'>	0	2	[0, 1]
3	Age	<class 'numpy.int32'>	0	7	[0, 6, 2, 4, 5, 3, 1]
4	Occupation	<class 'numpy.int64'>	0	21	[10, 16, 15, 7, 20, 9, 1, 12, 17, 0, 3, 4, 11,...
5	City_Category	<class 'numpy.int32'>	0	3	[0, 2, 1]
6	Stay_In_Current_City_Years	<class 'numpy.int32'>	0	5	[2, 4, 3, 1, 0]
7	Marital_Status	<class 'numpy.int64'>	0	2	[0, 1]
8	Product_Category_1	<class 'numpy.int64'>	0	20	[2, 0, 11, 7, 4, 3, 1, 5, 13, 10, 12, 14, 6, 1...
9	Product_Category_2	<class 'numpy.int64'>	0	17	[6, 4, 12, 0, 13, 14, 9, 3, 1, 2, 10, 7, 8, 15...
10	Product_Category_3	<class 'numpy.int64'>	0	15	[12, 10, 13, 2, 1, 11, 4, 5, 9, 3, 8, 0, 14, 7...

After encoding, the total memory usage has decreased significantly from 270 to 50 bytes.

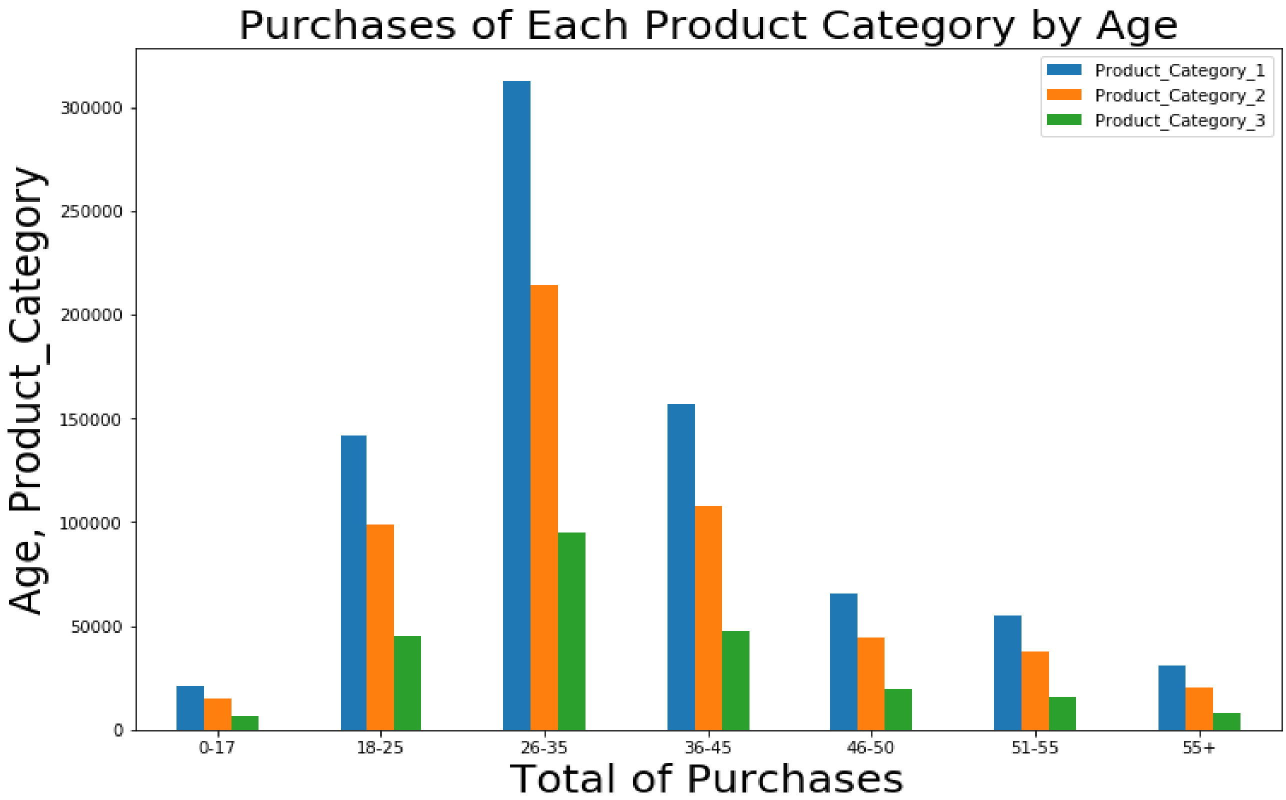
```
df_concat_copy.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Produc
0	0	684	0	0	10	0	2	0	2	6	
1	0	2406	0	0	10	0	2	0	0	4	
2	0	868	0	0	10	0	2	0	11	6	
3	0	844	0	0	10	0	2	0	11	12	
4	1	2769	1	6	16	2	4	0	7	6	

----

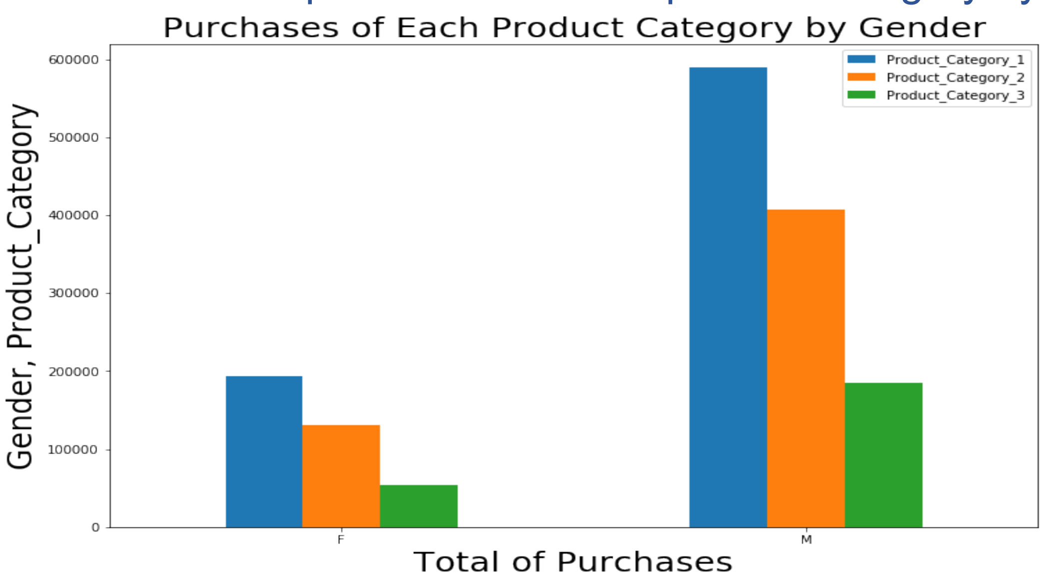
3. Visualisation

3.1 - Number of purchases of each product category by Age -



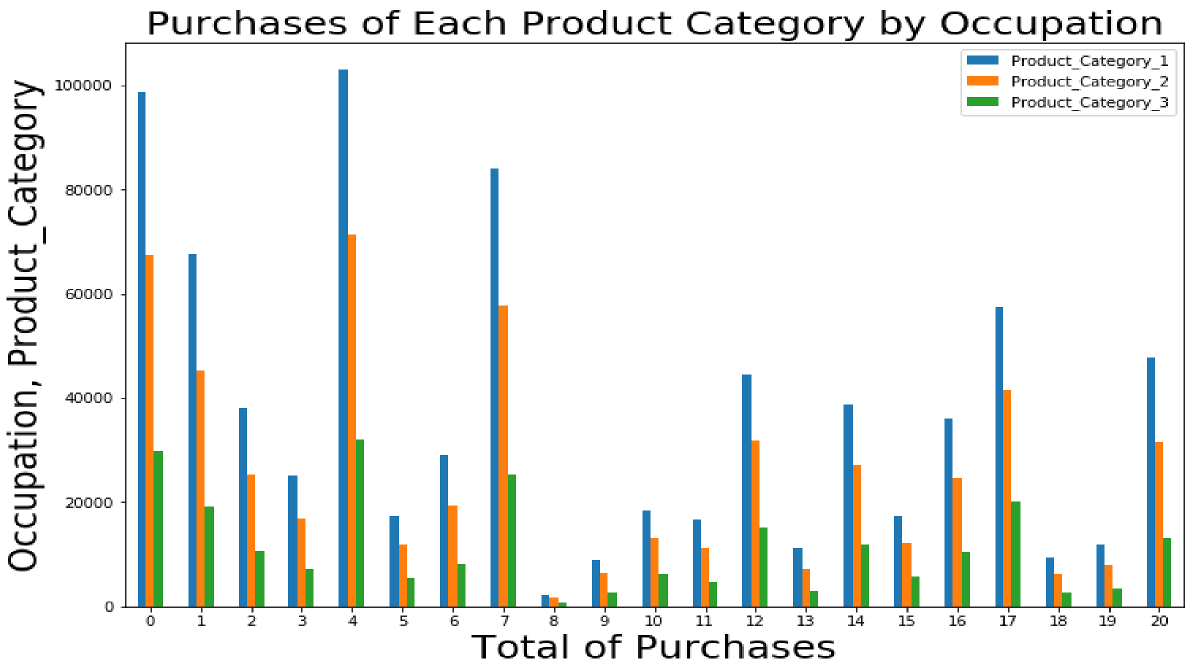
The graph shows that age group of 26 – 35 buys most of the products, followed by group of age 36 – 45 and 18 – 25.

3.2 - Number of purchases of each product category by Gender



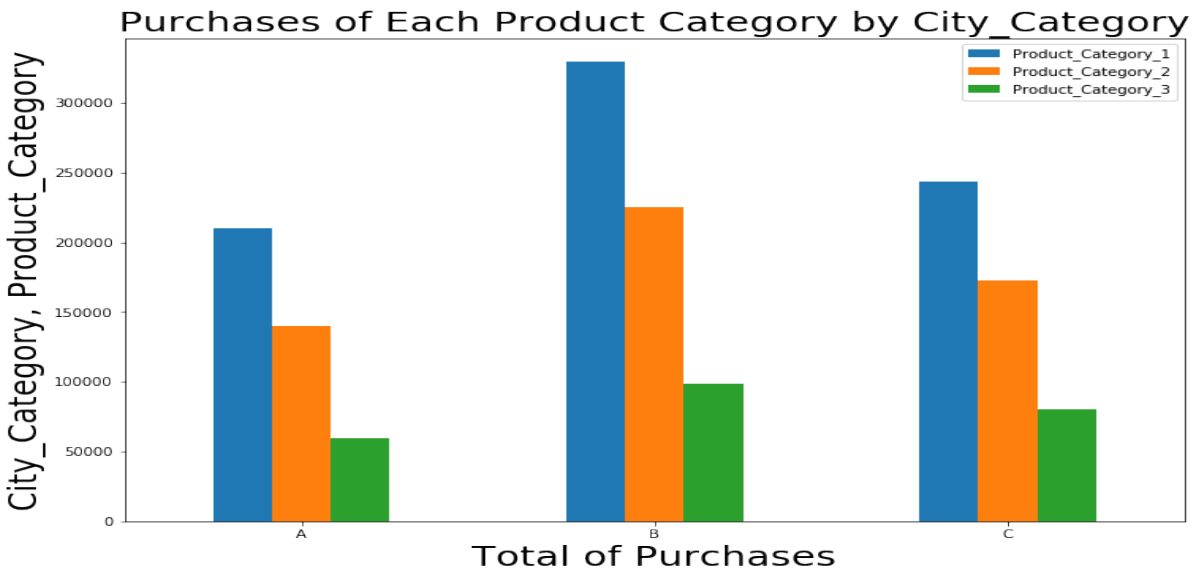
From the graph, it is clear that male purchases way more than females. Also, males buy Product_Category_1 more than other products.

3.3 - Number of purchases of each product category by Occupation



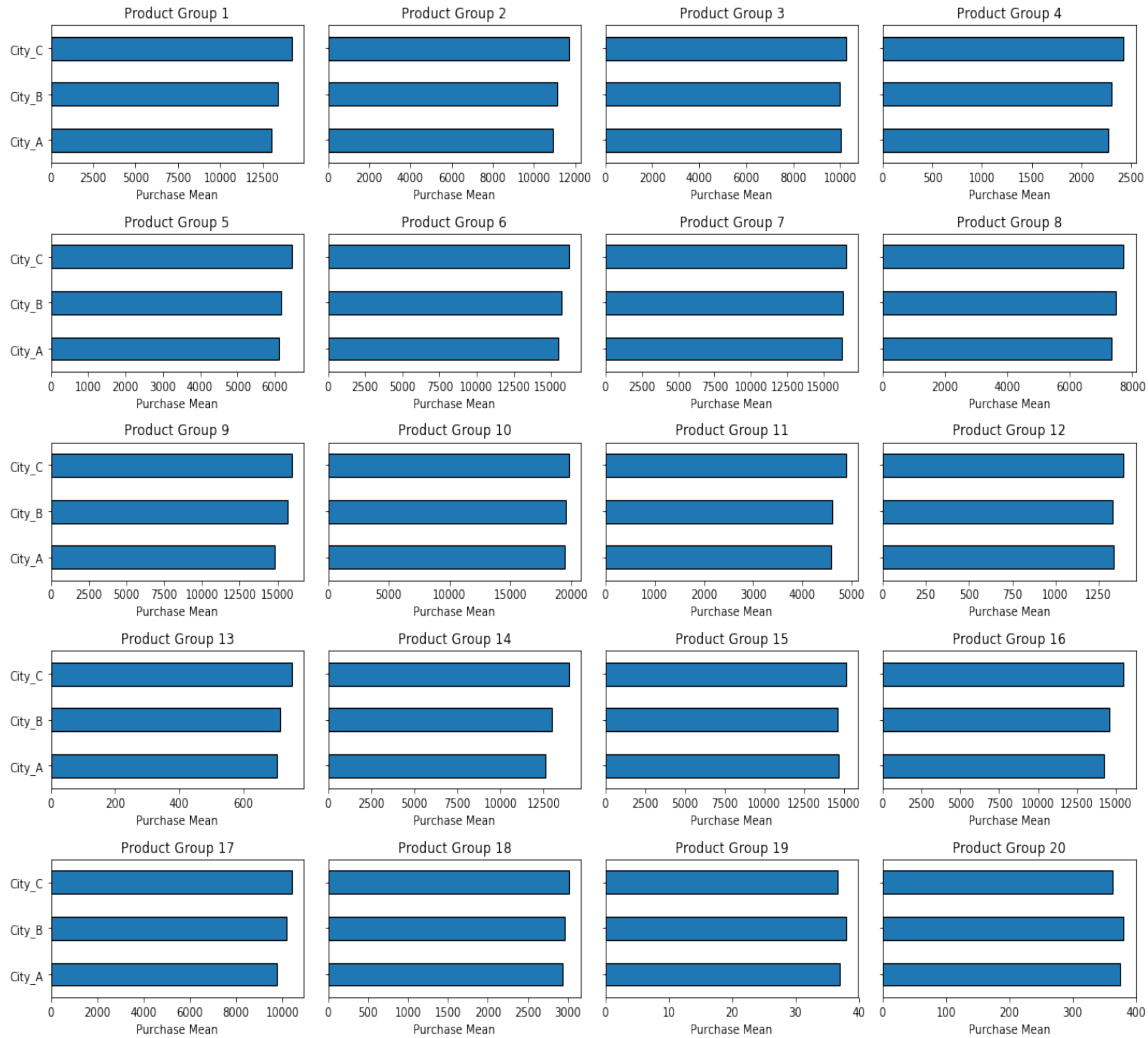
From this, we have occupation 4, 1 and 7 as top 3 buyers.

3.4 - Number of purchases of each product category by City_Category



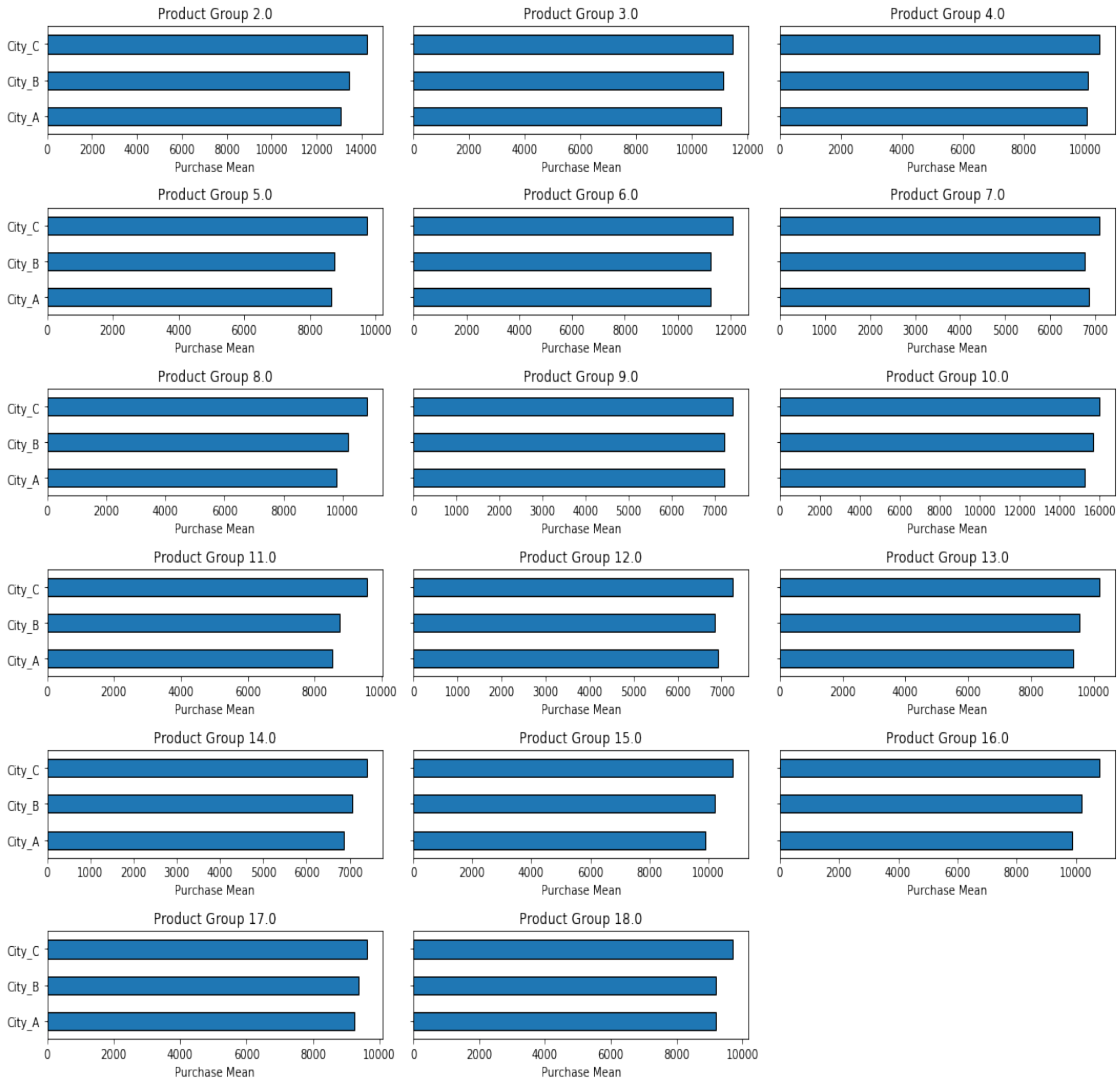
Here, most of the purchase is from city B.

3.4.1 - Purchase Mean of Product_Category_1 by City



From this graph, sale of product group 19 is among all. Also, product group 14 has the highest sale in city C.

3.4.2 - Purchase Mean of Product_Category_2 by City

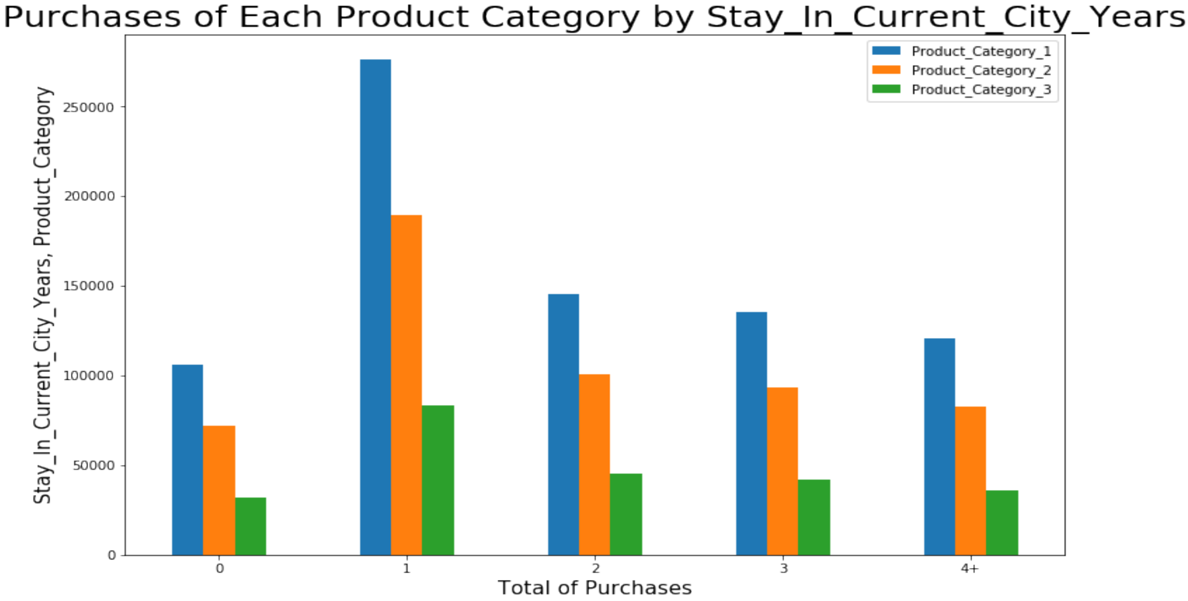


Here, product group 10 has the highest sale, again in city C. Rest are same.

3.4.3 - Purchase Mean of Product_Category_3 by City

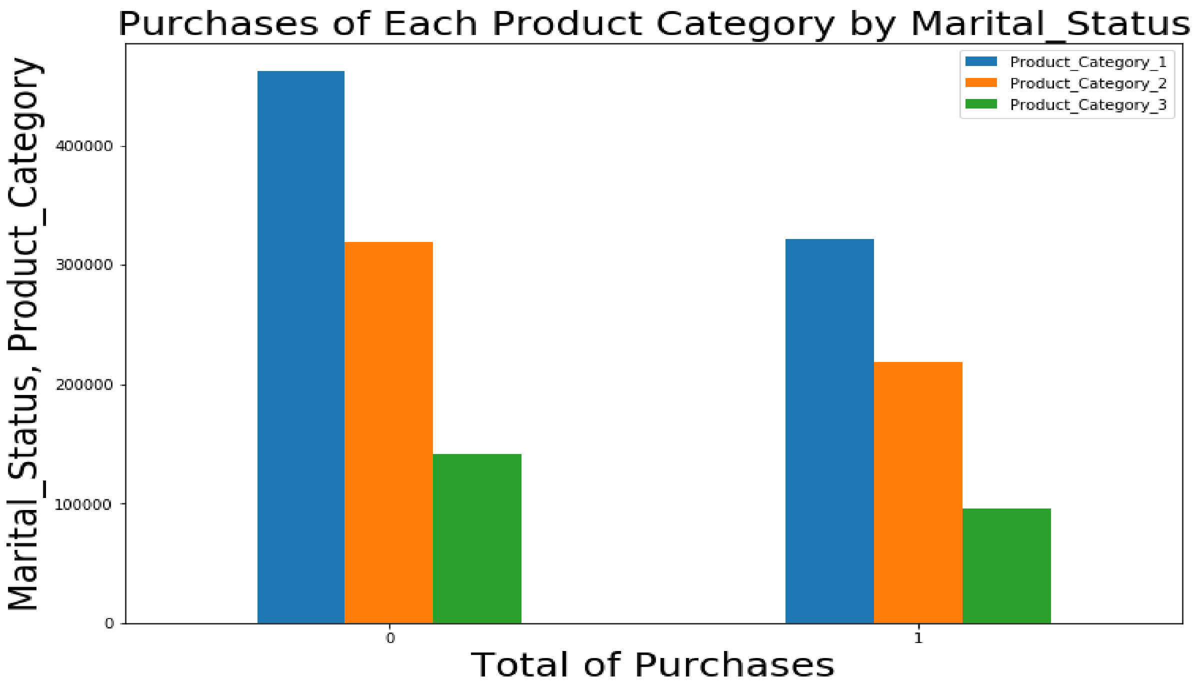
Here, product group 3 has the highest purchase in city A.

3.5 - Number of purchases of each product category by Stay_In_Current_City_Years



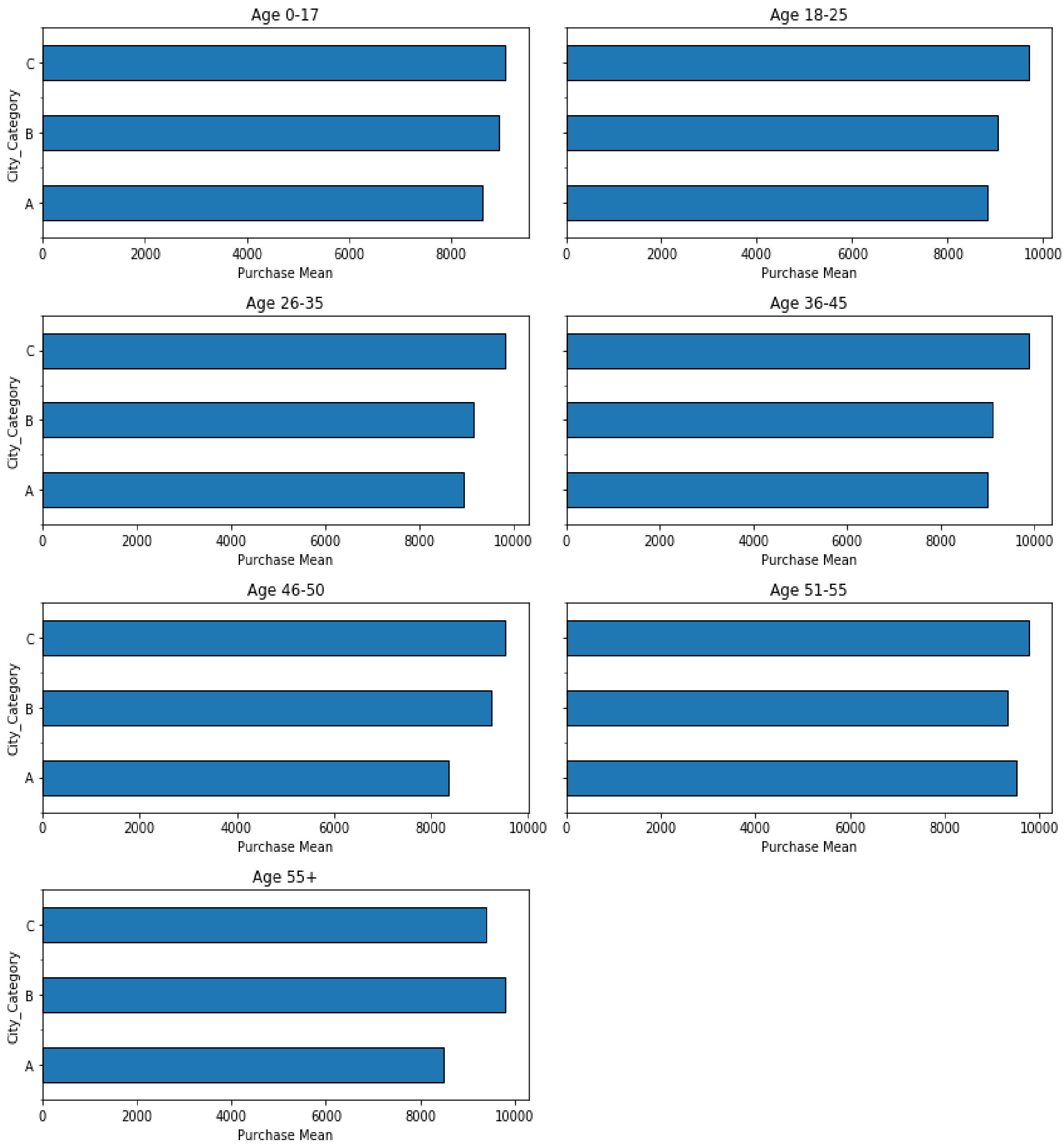
We can say, people who are staying for 1 year in the current city, buy more than other. Also, product category 1 has the highest sales.

3.6 - Number of purchases of each product category by Marital_Status



Clearly, those who are not married purchase more than those of married people. Here. Product category 1 has the highest sales.

3.7 - Age and City_Category vs Purchase

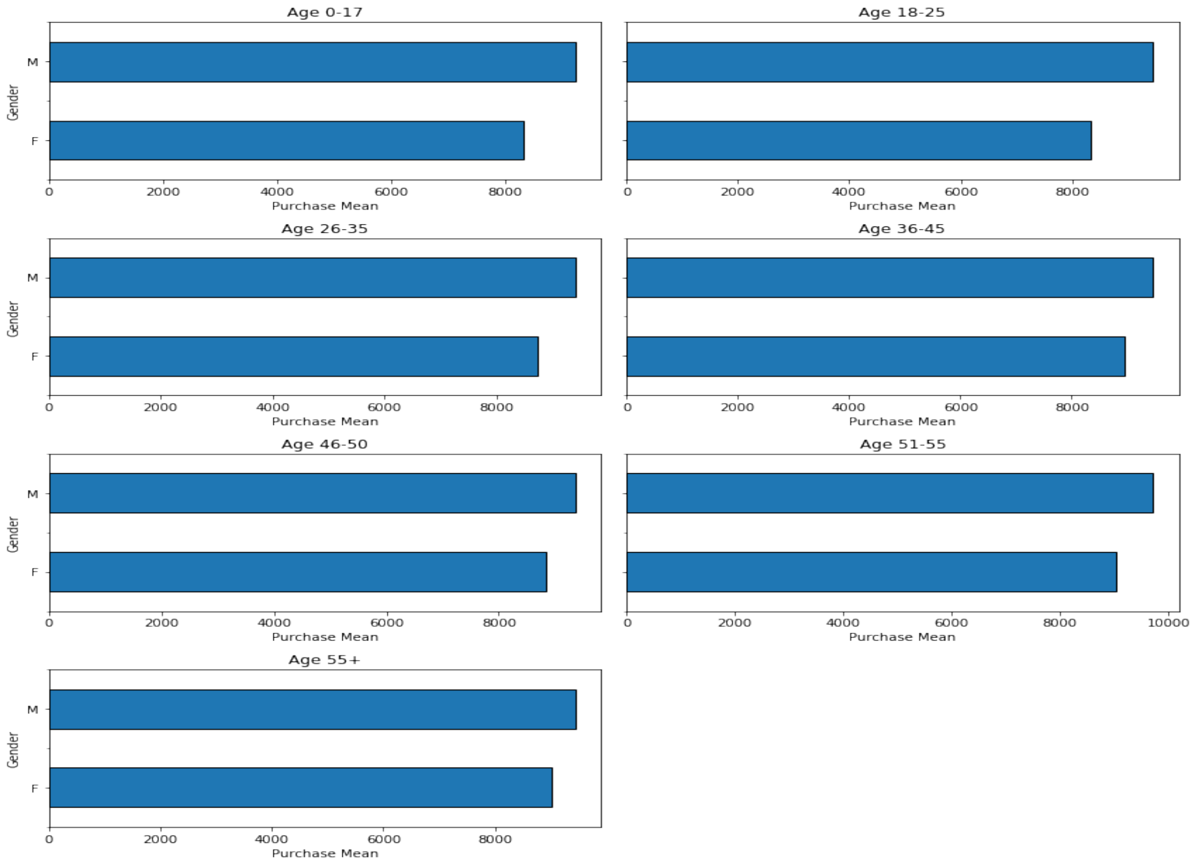


For city A, only people of age group 51 – 55 has the highest purchase score.

For city B, people above 55 age buys more products.

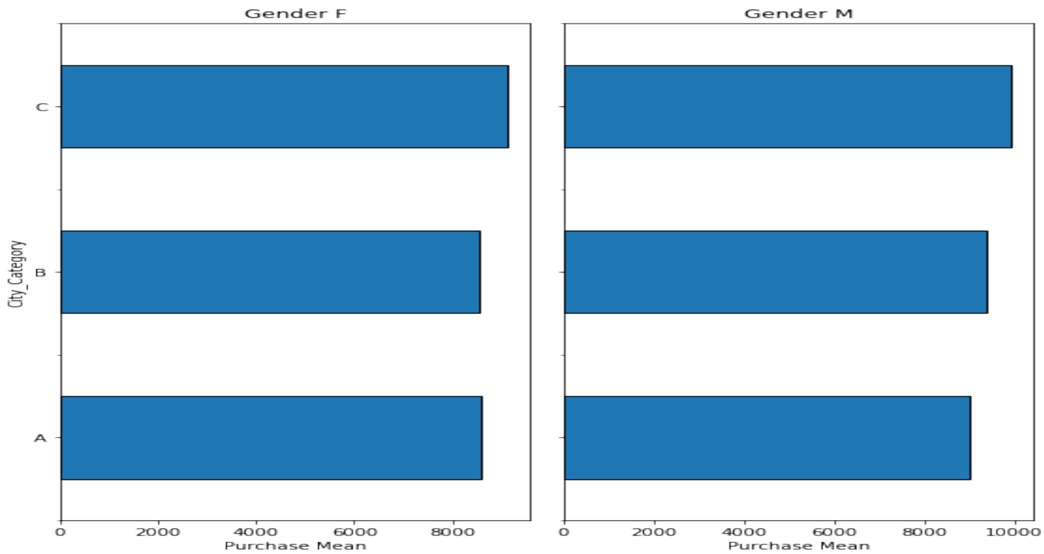
For city C, all the rest age groups buy products more than other city.

3.8 - Age and Gender vs Purchase



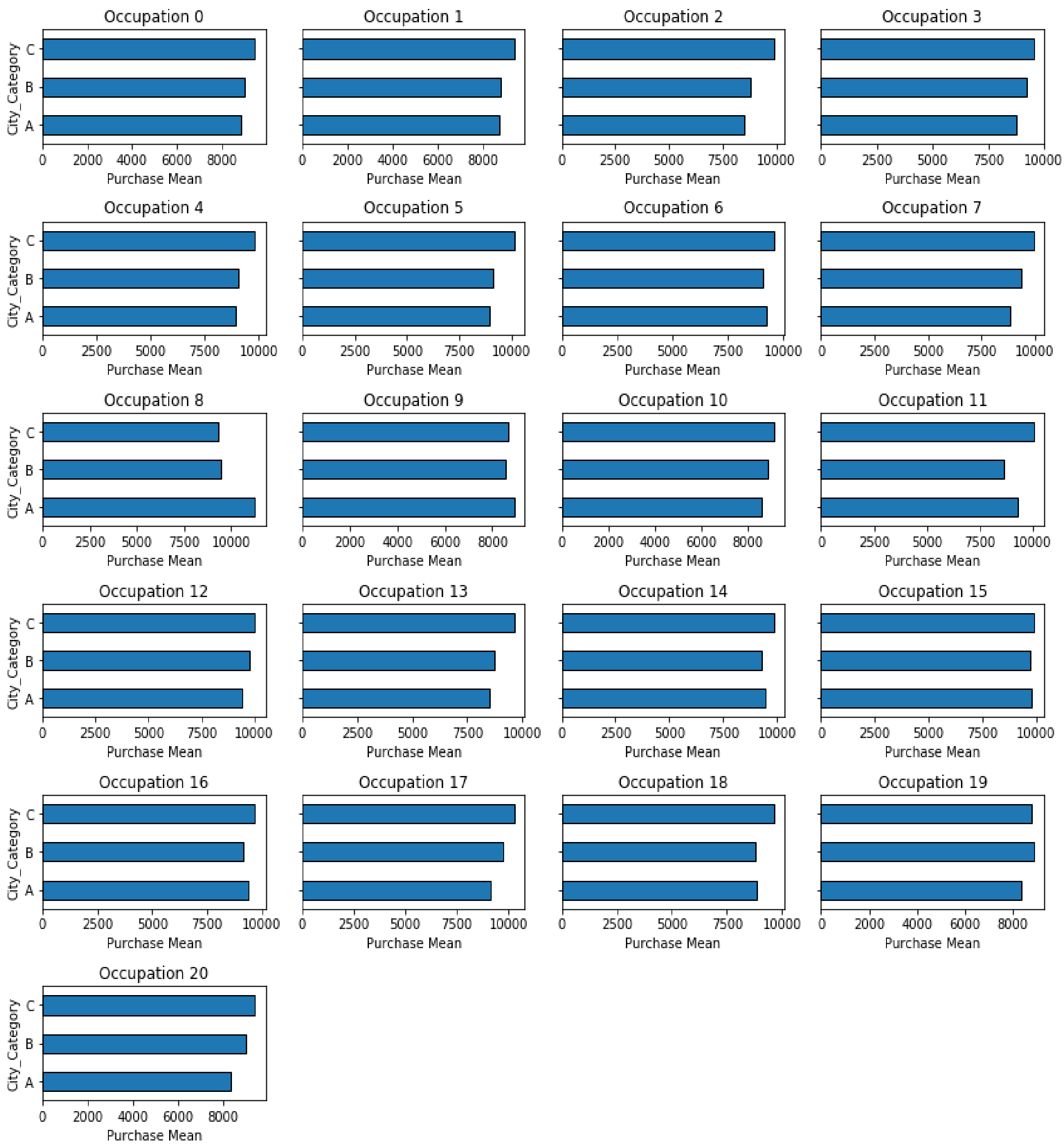
Clearly, male in all age groups buy more than that of females.

3.9 - City_Category and Gender vs Purchase



Males are more than females in all the cities.

3.10 - City_Category and Occupation vs Purchase

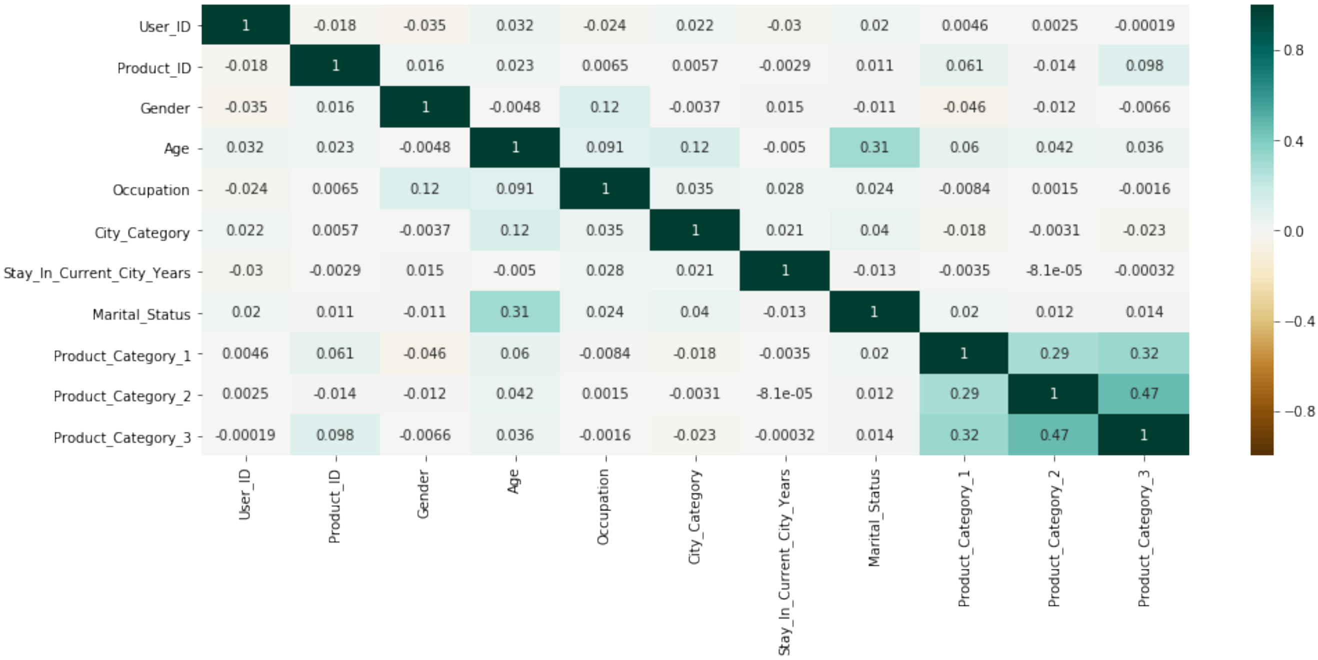


For city A, occupation 8 and 9 are at top

For city B, occupation 19 exceeds other significantly.

For city C, almost all of the rest occupations are on top in terms of purchasing products.

3.11 Heatmap



There is a slight positive correlation between product categories, other than that, there is no strong correlation between features.

----

4. Model Selection

First, we will make a util function to store certain information about the models in a tabular format using list. This will make the task easier to choose and compare the algorithms.

```
n [57]: def show_model_eval_table(model_attrib):
            df_model_eval = pd.DataFrame({
                'Names' : model_attrib['Names'],
                'Feature_Counts' : model_attrib['Feature_Counts'],
                'Feature_Names' : model_attrib['Feature_Names'],
                'R2' : model_attrib['R2'],
                'RMSE' : model_attrib['RMSE']
            })
            return df_model_eval.round(2)
```

```
n [58]: model_attrib = {'Names' : [],
                'Feature_Counts' : [],
                'Feature_Names' : [],
                'R2' : [],
                'RMSE' : []
        }
```

The given problem is a regression problem. Hence candidate algorithms to choose from are –

(i) Random Forest Regressor
(ii) Linear Regression
(iii) Decision Tree Regressor

We will use GridSearchCV for tuning the hyper-parameters and cross-validation.

4.1 Random Forest Regressor

```
n [61]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```
n [62]: rfr = RFR(random_state = 0)
        rfr.get_params().keys()
```

```
ut[62]: dict_keys(['bootstrap', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_spli
        t', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state',
        'verbose', 'warm_start'])
```

```
n [63]: param_grid = {
            'n_estimators': [20, 30],
            'max_features': ['auto', 'sqrt'],
            'min_samples_leaf' : [70, 80],
            'max_depth' : [ 7, 8]
        }
```

```
n [64]: CV_rfr = GridSearchCV(estimator = rfr, param_grid = param_grid, cv = 4)
        CV_rfr.fit(X_train, y_train)
```

```
ut[64]: GridSearchCV(cv=4, error_score='raise-deprecating',
               estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                   max_features='auto', max_leaf_nodes=None,
                   min_impurity_decrease=0.0, min_impurity_split=None,
                   min_samples_leaf=1, min_samples_split=2,
                   min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
                   oob_score=False, random_state=0, verbose=0, warm_start=False),
               fit_params=None, iid='warn', n_jobs=None,
               param_grid={'n_estimators': [20, 30], 'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [70, 80], 'max_depth': [7,
        8]},
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring=None, verbose=0)
```

Best parameters for RFR model are –

```
 [67]: CV_rfr.best_params_
t[67]: {'max_depth': 8,
        'max_features': 'auto',
        'min_samples_leaf': 70,
        'n_estimators': 20}
```

Using these parameters, we get two models as –

In [68]:
```
RFR_model_1 = cross_validate(RFR(random_state = 0, min_samples_split = 8, min_samples_leaf = 80, n_estimators = 30),
                             X, y, cv = 5, n_jobs = 5, verbose = 10, scoring = cv_score)
model_attrib['Names'].append('RFR_model_1')
model_attrib['Feature_Counts'].append(X.shape[1])
model_attrib['Feature_Names'].append(list(X.columns))
model_attrib['R2'].append(RFR_model_1['test_r2'].mean())
model_attrib['RMSE'].append((abs(RFR_model_1['test_neg_mean_squared_error']) ** 0.5).mean())

RFR_model_2 = cross_validate(RFR(random_state = 0, min_samples_split = 8, max_depth = 8, max_features = 'auto', min_samples_leaf
                             X, y, cv = 5, n_jobs = 5, verbose = 10, scoring = cv_score)
model_attrib['Names'].append('RFR_model_2')
model_attrib['Feature_Counts'].append(X.shape[1])
model_attrib['Feature_Names'].append(list(X.columns))
model_attrib['R2'].append(RFR_model_2['test_r2'].mean())
model_attrib['RMSE'].append((abs(RFR_model_2['test_neg_mean_squared_error']) ** 0.5).mean())
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:  2.4min remaining:  3.6min
[Parallel(n_jobs=5)]: Done   3 out of   5 | elapsed:  2.4min remaining:  1.6min
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:  2.5min remaining:    0.0s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:  2.5min finished
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:   55.4s remaining:  1.4min
[Parallel(n_jobs=5)]: Done   3 out of   5 | elapsed:   55.4s remaining:   36.9s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:   55.5s remaining:    0.0s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:   55.5s finished
```

In [69]:
```
show_model_eval_table(model_attrib)
```

Out[69]:

	Names	Feature_Counts	Feature_Names	R2	RMSE
0	RFR_model_1	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.71	2703.97
1	RFR_model_2	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.67	2883.50

4.2 Linear Regression

```
In [70]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

In [71]: lr = LR()
         lr.get_params().keys()

Out[71]: dict_keys(['copy_X', 'fit_intercept', 'n_jobs', 'normalize'])

In [72]: params_lr = {
             'copy_X': [True, False],
             'fit_intercept': [True, False],
             'normalize': [True, False]
         }

In [73]: CV_lr = GridSearchCV(estimator = lr, param_grid = params_lr, cv = 3)
         CV_lr.fit(X_train, y_train)

Out[73]: GridSearchCV(cv=3, error_score='raise-deprecating',
                estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'copy_X': [True, False], 'fit_intercept': [True, False], 'normalize': [True, False]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)
```

Best parameters for LR model are –

```
In [74]: CV_lr.best_params_

Out[74]: {'copy_X': True, 'fit_intercept': True, 'normalize': False}
```

Using these parameters, we get two models as –

75]:
```
LR_model_1 = cross_validate(LR(), X, y, cv = 5, n_jobs = 5, verbose = 10, scoring = cv_score)
model_attrib['Names'].append('LR_model_1')
model_attrib['Feature_Counts'].append(X.shape[1])
model_attrib['Feature_Names'].append(list(X.columns))
model_attrib['R2'].append(LR_model_1['test_r2'].mean())
model_attrib['RMSE'].append((abs(LR_model_1['test_neg_mean_squared_error']) ** 0.5).mean())

LR_model_2 = cross_validate(LR(copy_X = True, fit_intercept = True, normalize = False), X, y, cv = 5, n_jobs = 5, verbose = 10, 
model_attrib['Names'].append('LR_model_2')
model_attrib['Feature_Counts'].append(X.shape[1])
model_attrib['Feature_Names'].append(list(X.columns))
model_attrib['R2'].append(LR_model_2['test_r2'].mean())
model_attrib['RMSE'].append((abs(LR_model_2['test_neg_mean_squared_error']) ** 0.5).mean())
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:    4.7s remaining:    7.0s
[Parallel(n_jobs=5)]: Done   3 out of   5 | elapsed:    4.7s remaining:    3.1s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    4.7s remaining:    0.0s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    4.7s finished
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:    1.7s remaining:    2.6s
[Parallel(n_jobs=5)]: Done   3 out of   5 | elapsed:    1.7s remaining:    1.1s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    1.7s remaining:    0.0s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    1.7s finished
```

76]:
```
show_model_eval_table(model_attrib)
```

76]:

	Names	Feature_Counts	Feature_Names	R2	RMSE
0	RFR_model_1	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.71	2703.97
1	RFR_model_2	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.67	2883.50
2	LR_model_1	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.13	4667.42
3	LR_model_2	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.13	4667.42

4.3 Decision Tree Regressor

```
In [77]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

In [78]: dtr = DTR()
         dtr.get_params().keys()

Out[78]: dict_keys(['criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_split', 'min_samp
         les_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'presort', 'random_state', 'splitter'])

In [79]: param_grid = {
             'max_features': ['auto', 'sqrt'],
             'min_samples_leaf' : [70, 80],
             'max_depth' : [ 7, 8]
         }

In [80]: CV_dtr = GridSearchCV(estimator = dtr, param_grid = param_grid, cv = 4)
         CV_dtr.fit(X_train, y_train)

Out[80]: GridSearchCV(cv=4, error_score='raise-deprecating',
                estimator=DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                    max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    presort=False, random_state=None, splitter='best'),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [70, 80], 'max_depth': [7, 8]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)
```

Best parameters for DTR model are –

```
In [81]: CV_dtr.best_params_

Out[81]: {'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 80}
```

Using these parameters, we get two models as –

```
n [82]: DTR_model_1 = cross_validate(DTR(random_state = 0), X, y, cv = 5, n_jobs = 5, verbose = 10, scoring = cv_score)
        model_attrib['Names'].append('DTR_model_1')
        model_attrib['Feature_Counts'].append(X.shape[1])
        model_attrib['Feature_Names'].append(list(X.columns))
        model_attrib['R2'].append(DTR_model_1['test_r2'].mean())
        model_attrib['RMSE'].append((abs(DTR_model_1['test_neg_mean_squared_error']) ** 0.5).mean())

        DTR_model_2 = cross_validate(DTR(random_state = 0, max_depth = 8, max_features = 'auto', min_samples_leaf = 80),
                                     X, y, cv = 5, n_jobs = 5, verbose = 10, scoring = cv_score)
        model_attrib['Names'].append('DTR_model_2')
        model_attrib['Feature_Counts'].append(X.shape[1])
        model_attrib['Feature_Names'].append(list(X.columns))
        model_attrib['R2'].append(DTR_model_2['test_r2'].mean())
        model_attrib['RMSE'].append((abs(DTR_model_2['test_neg_mean_squared_error']) ** 0.5).mean())

        [Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
        [Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:   13.6s remaining:   20.4s
        [Parallel(n_jobs=5)]: Done   3 out of   5 | elapsed:   13.7s remaining:    9.1s
        [Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:   14.1s remaining:    0.0s
        [Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:   14.1s finished
        [Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
        [Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:    4.4s remaining:    6.6s
        [Parallel(n_jobs=5)]: Done   3 out of   5 | elapsed:    4.4s remaining:    2.9s
        [Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    4.4s remaining:    0.0s
        [Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    4.4s finished

n [83]: show_model_eval_table(model_attrib)

ut[83]:
```

	Names	Feature_Counts	Feature_Names	R2	RMSE
0	RFR_model_1	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.71	2703.97
1	RFR_model_2	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.67	2883.50
2	LR_model_1	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.13	4667.42
3	LR_model_2	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.13	4667.42
4	DTR_model_1	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.44	3767.69
5	DTR_model_2	11	[User_ID, Product_ID, Gender, Age, Occupation,...	0.67	2895.43

From the table, we can clearly conclude that RFR_model_1 is best with R2 score of 0.71 and RMSE score of 2703.97.

----

5. Training and Evaluating the Model

Train the RFR_model_1 and fit X and y in it.

Predict the values using test dataset (df_test_copy), in which all the features are of int categorical data type.

```
[89]: RFR_model_1 = RFR(random_state = 0, min_samples_split = 8, min_samples_leaf = 80, n_estimators = 30)
      RFR_model_1.fit(X,y)
```

```
t[89]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                  max_features='auto', max_leaf_nodes=None,
                  min_impurity_decrease=0.0, min_impurity_split=None,
                  min_samples_leaf=80, min_samples_split=8,
                  min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=None,
                  oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
[91]: RFR_model_1_y_hat = RFR_model_1.predict(df_test_copy)
```

```
[92]: RFR_model_1_y_hat
```

```
t[92]: array([15849.23990638, 12006.73596883,  5448.90648063, ...,
              10283.15845541, 20418.96611741,  2309.51555634])
```

Convert predicted value into dataframe and concatenate it with User_ID and Purchase_ID, and store it in Result_Problem2.csv file.

```
[93]: df_RFR_model_1_y_hat =  pd.DataFrame(RFR_model_1_y_hat, columns = ['Purchase'])
```

```
[94]: df_RFR_model_1_y_hat.head()
```

[94]:

	Purchase
0	15849.239906
1	12006.735969
2	5448.906481
3	2674.055083
4	2682.947088

```
[95]: result = pd.concat([df_test.loc[:,['User_ID', 'Product_ID']],
                          df_RFR_model_1_y_hat], axis = 1)
```

```
[96]: result.head()
```

[96]:

	User_ID	Product_ID	Purchase
0	1000004	P00128942	15849.239906
1	1000009	P00113442	12006.735969
2	1000010	P00288442	5448.906481
3	1000010	P00145342	2674.055083
4	1000011	P00053842	2682.947088

```
[97]: result.to_csv('Result_Problem2.csv', index = False)
```