In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from plotly.offline import init_notebook_mode
init_notebook_mode(connected = True)

import os
for dirname, _, filenames in os.walk('C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML')
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML\ML.ipynb
C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML\Test.csv
C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML\Train.csv
C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML\.ipynb_checkpoints\ML-checkpoint.ipynb
C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML\input\Test.csv
C:/Users/Dell/Desktop/EDU/SEM5/SE/J Comp/ML\input\Train.csv
```

In [2]:
```
data = pd.read_csv('Train.csv')
data.head()
```

Out[2]:

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Scor
0	462809	Male	No	22	No	Healthcare	1.0	Lo
1	462643	Female	Yes	38	Yes	Engineer	NaN	Averag
2	466315	Female	Yes	67	Yes	Engineer	1.0	Lo
3	461735	Male	Yes	67	Yes	Lawyer	0.0	Hig
4	462669	Female	Yes	40	Yes	Entertainment	NaN	Hig

Variable	Definition
ID Unique	ID
Gender	Gender of the customer
Ever_Married	Marital status of the customer
Age	Age of the customer
Graduated	Is the customer a graduate?
Profession	Profession of the customer
Work_Experience	Work Experience in years
Spending_Score	Spending score of the customer
Family_Size	Number of family members for the customer (including the customer)
Var_1	Anonymised Category for the customer
Segmentation	(target) Customer Segment of the customer

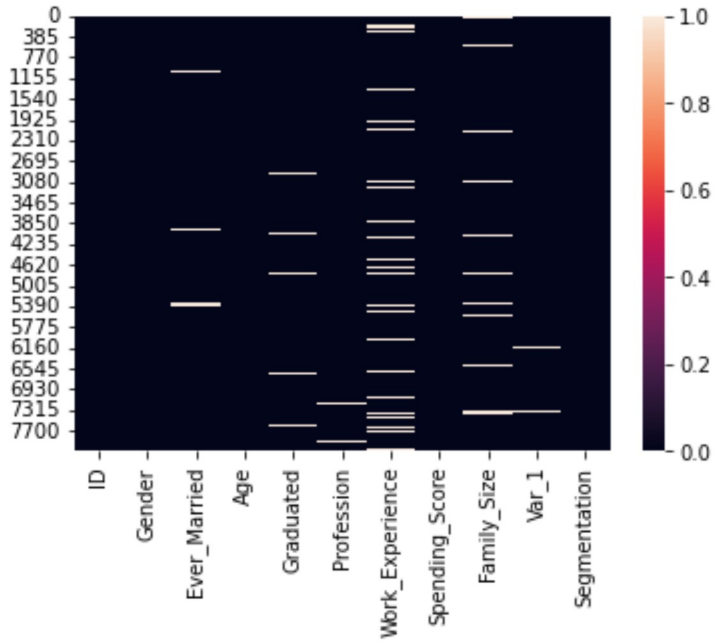
In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8068 entries, 0 to 8067
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               8068 non-null   int64  
 1   Gender           8068 non-null   object 
 2   Ever_Married     7928 non-null   object 
 3   Age              8068 non-null   int64  
 4   Graduated        7990 non-null   object 
 5   Profession       7944 non-null   object 
 6   Work_Experience  7239 non-null   float64
 7   Spending_Score   8068 non-null   object 
 8   Family_Size      7733 non-null   float64
 9   Var_1            7992 non-null   object 
 10  Segmentation     8068 non-null   object 
dtypes: float64(2), int64(2), object(7)
memory usage: 693.5+ KB
```

Visualizations

In [4]:
```
import seaborn as sns
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode,iplot
```

In [5]:
```
sns.heatmap(data.isnull());
```

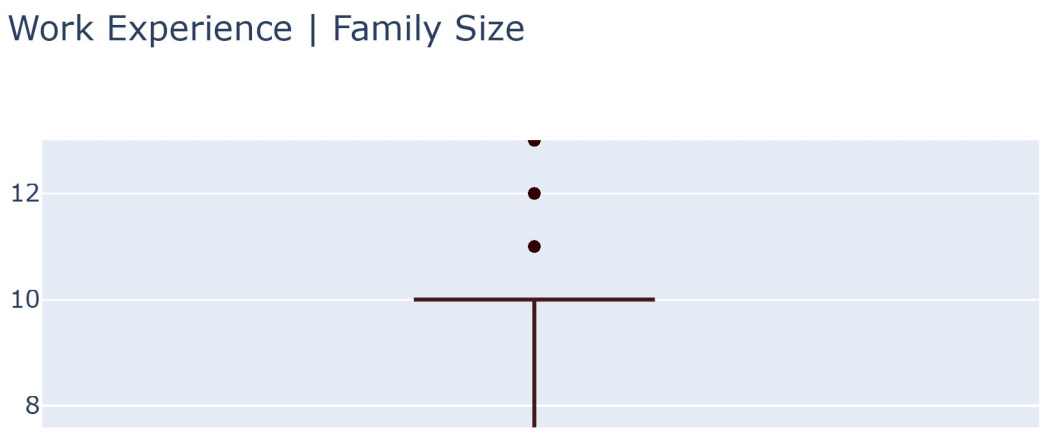


In [6]:
```
temp = data.describe()
temp.style.background_gradient(cmap='Oranges')
```

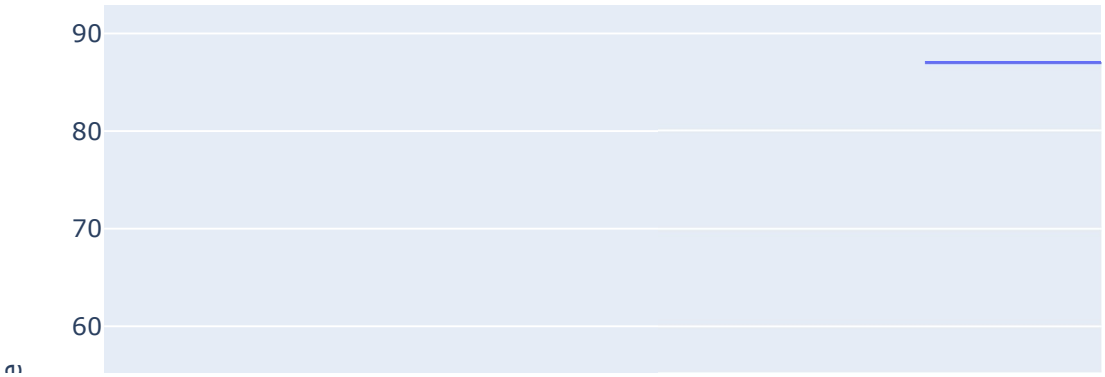
Out[6]:

	ID	Age	Work_Experience	Family_Size
count	8068.000000	8068.000000	7239.000000	7733.000000
mean	463479.214551	43.466906	2.641663	2.850123
std	2595.381232	16.711696	3.406763	1.531413
min	458982.000000	18.000000	0.000000	1.000000
25%	461240.750000	30.000000	0.000000	2.000000
50%	463472.500000	40.000000	1.000000	3.000000
75%	465744.250000	53.000000	4.000000	4.000000
max	467974.000000	89.000000	14.000000	9.000000

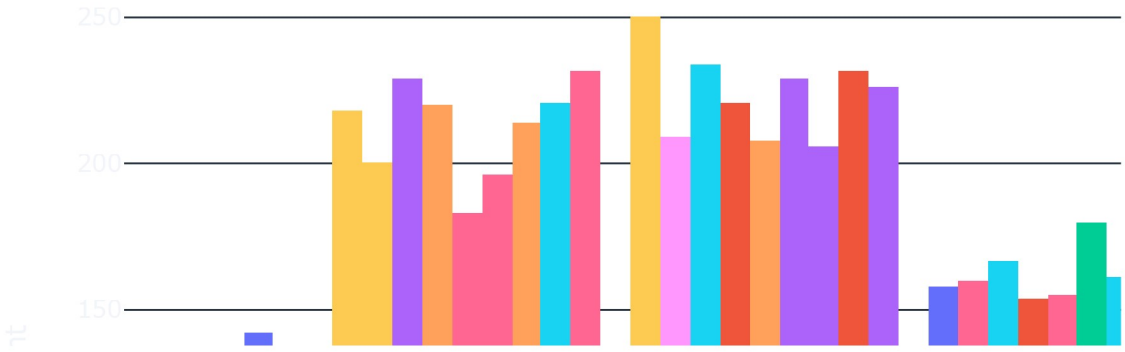
In [7]:
```
g1 = [go.Box(y=data.Work_Experience,name="Work_Experience",marker=dict(color="rgba(
g2 = [go.Box(y=data.Family_Size,name="Family_Size",marker=dict(color="rgba(0,102,10
layout2 = go.Layout(title="Work Experience | Family Size",yaxis=dict(range=[0,13]))
fig2 = go.Figure(data=g1+g2,layout=layout2)
iplot(fig2)
```



In [8]:
```
grafico = px.box(data, y='Age')
grafico.show()
```


90
80
70
60

In [9]:
```
fig2 = px.histogram(data,x='Age',color='Age',template='plotly_dark')
fig2.show()
```



In [10]:
```
fig2 = px.histogram(data,x='Gender',color='Gender',template='plotly_dark')
fig2.show()
```

EDA

In [11]:
```
plot_data = data.groupby('Segmentation')['Segmentation'].agg(['count']).reset_index

fig = px.pie(plot_data, values = plot_data['count'], names = plot_data['Segmentatio

fig.update_traces(textposition = 'inside', textinfo = 'percent + label', hole = 0.5
                  marker = dict(colors = ['#2A3132','#336B87'], line = dict(color =

fig.update_layout(title_text = 'Customer<br>Segmentation', title_x = 0.5, title_y =
                  title_font_family = 'Calibri', title_font_color = 'black', showle

fig.show()
```

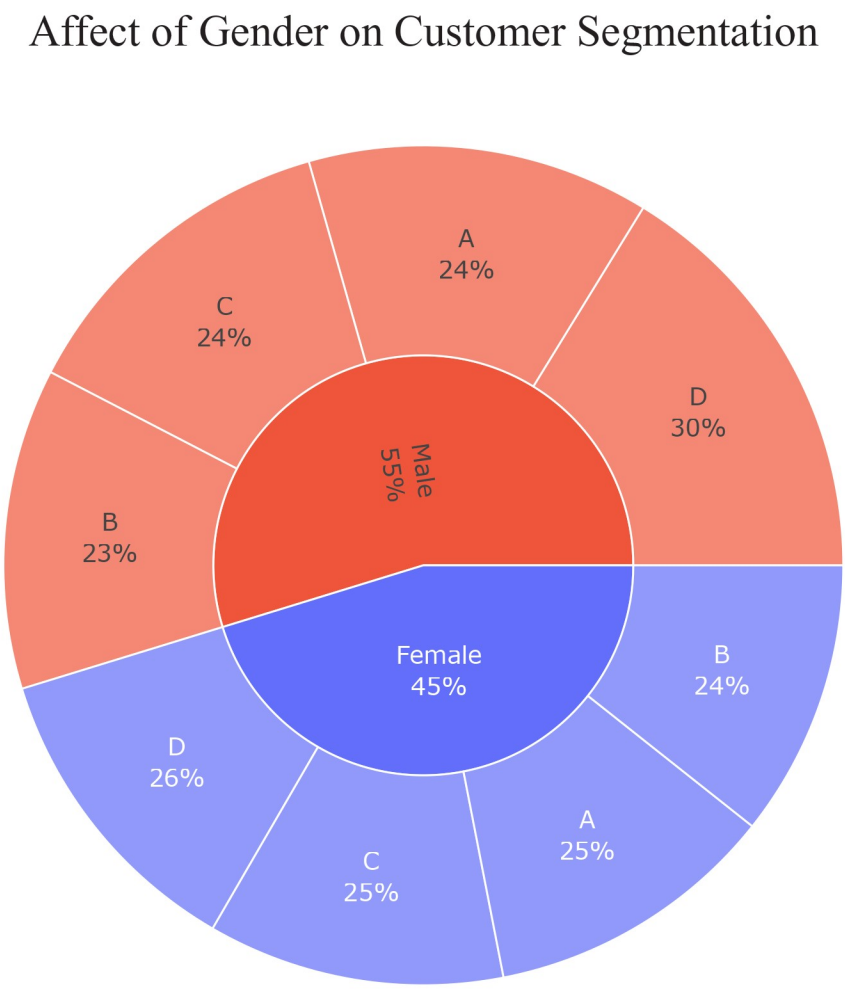

A
24.4%

In [12]:
```
def plot_category(feature, figsize=None):
    A_count = data[data['Segmentation']=='A'].groupby([feature]).size()
    B_count = data[data['Segmentation']=='B'].groupby([feature]).size()
    C_count = data[data['Segmentation']=='C'].groupby([feature]).size()
    D_count = data[data['Segmentation']=='D'].groupby([feature]).size()
    labels = A_count.index

    x = np.arange(len(labels)) # the label locations
    width = 0.7  # the width of the bars

    if figsize:
        fig, ax = plt.subplots(figsize=figsize)
    else:
        fig, ax = plt.subplots()
    rects1 = ax.bar(x-width/3, round(A_count*100/data.groupby([feature]).size(), 2)
                    width/5, label='A')
    rects2 = ax.bar(x-width/8, round(B_count*100/data.groupby([feature]).size(), 2)
                    width/5, label='B')
    rects3 = ax.bar(x+width/8, round(C_count*100/data.groupby([feature]).size(), 2)
                    width/5, label='C')
    rects4 = ax.bar(x+width/3, round(D_count*100/data.groupby([feature]).size(), 2)
                    width/5, label='D')

    ax.set_ylabel('Count')
    ax.set_title('Based on %s'%feature)
    ax.set_xticks(x)
    ax.set_xticklabels(labels, rotation=80)
    ax.legend(loc=0, bbox_to_anchor=(1, 1));

    ax.bar_label(rects1, padding=1)
    ax.bar_label(rects2, padding=1)
    ax.bar_label(rects3, padding=1)
    ax.bar_label(rects4, padding=1)

    fig.tight_layout()
    plt.show()

def plot_numerical(feature, figsize=None):
    fig = plt.figure(figsize=(10,6))

    sns.kdeplot(data[data['Segmentation']=='A'][feature])
    sns.kdeplot(data[data['Segmentation']=='B'][feature])
    sns.kdeplot(data[data['Segmentation']=='C'][feature])
    sns.kdeplot(data[data['Segmentation']=='D'][feature])

    fig.legend(labels=['Segmentation A', 'Segmentation B', 'Segmentation C', 'Segme
    plt.title('Based on %s'%feature)
    plt.show()

def plot_pie(feature):
    plot_data = data.groupby([feature, 'Segmentation'])[feature].agg({'count'}).res

    fig = px.sunburst(plot_data, path = [feature, 'Segmentation'], values = 'count'
                      title = 'Affect of %s on Customer Segmentation'%feature, widt

    fig.update_layout(plot_bgcolor = 'white', title_font_family = 'Calibri Black',
                      title_font_size = 22, title_x = 0.5)

    fig.update_traces(textinfo = 'label + percent parent')
```

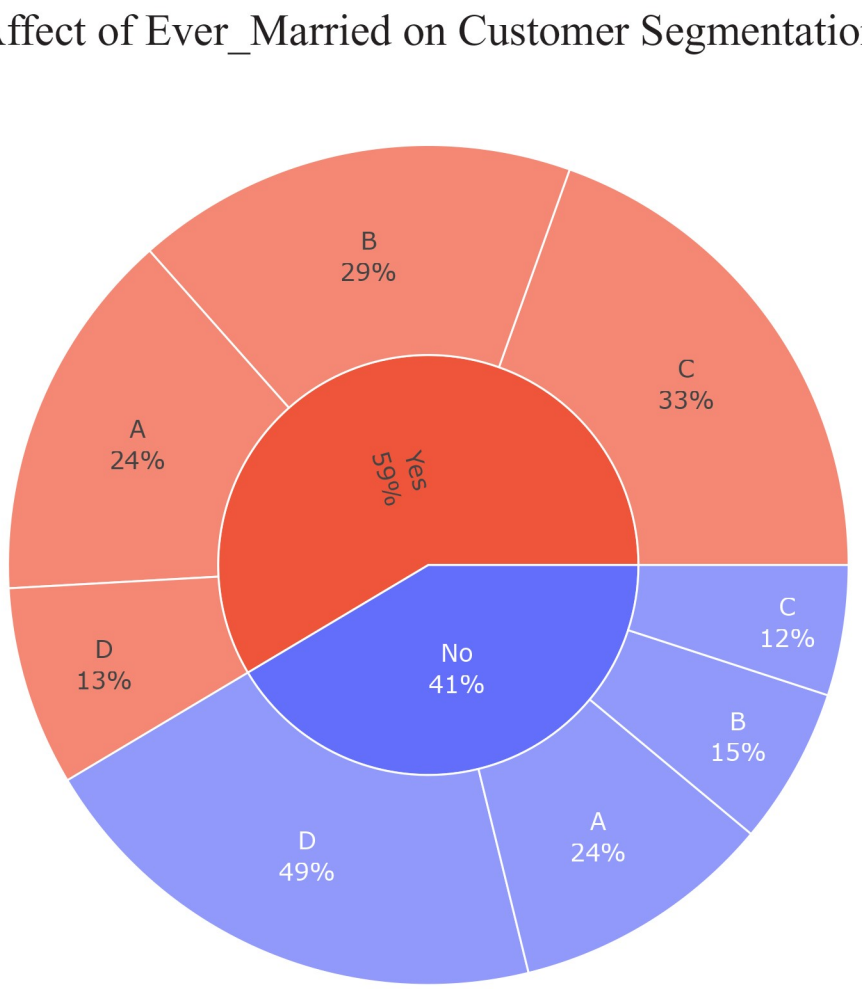
```
    fig.update_traces(textinfo = 'label + percent parent')
    fig.show()
```

In [13]:
```
for feature in ['Gender', 'Ever_Married', 'Graduated', 'Spending_Score']:
    plot_pie(feature)
```

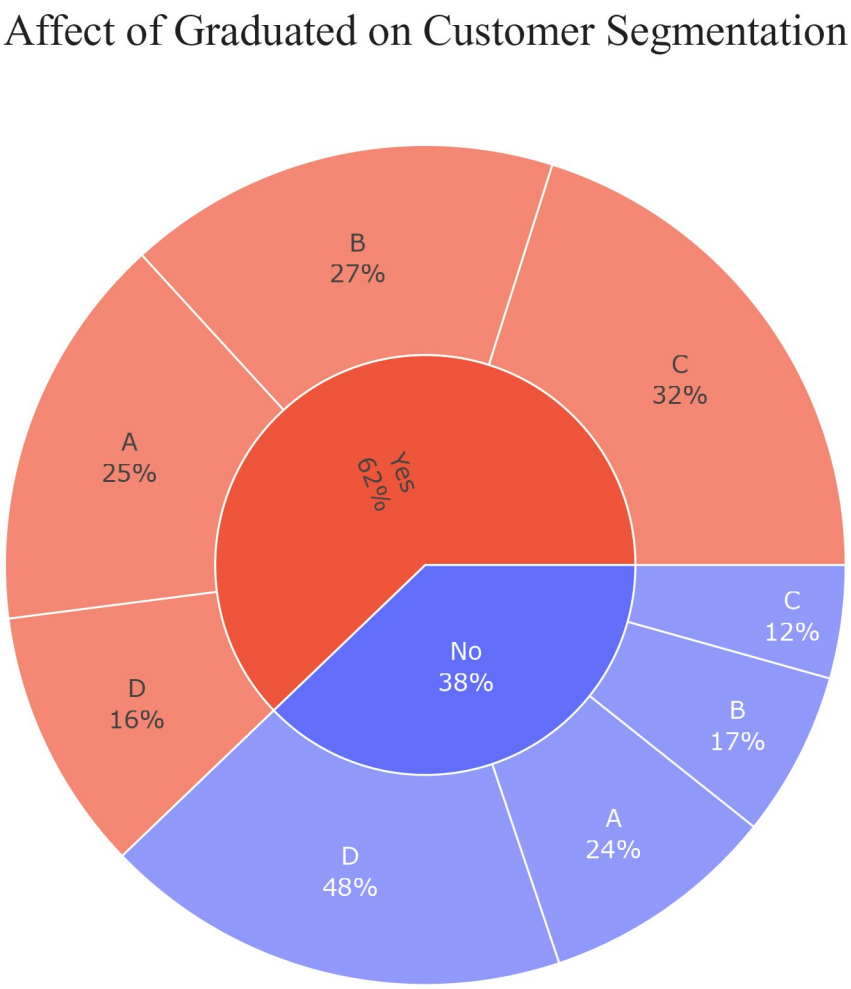
```
C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.

C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
```



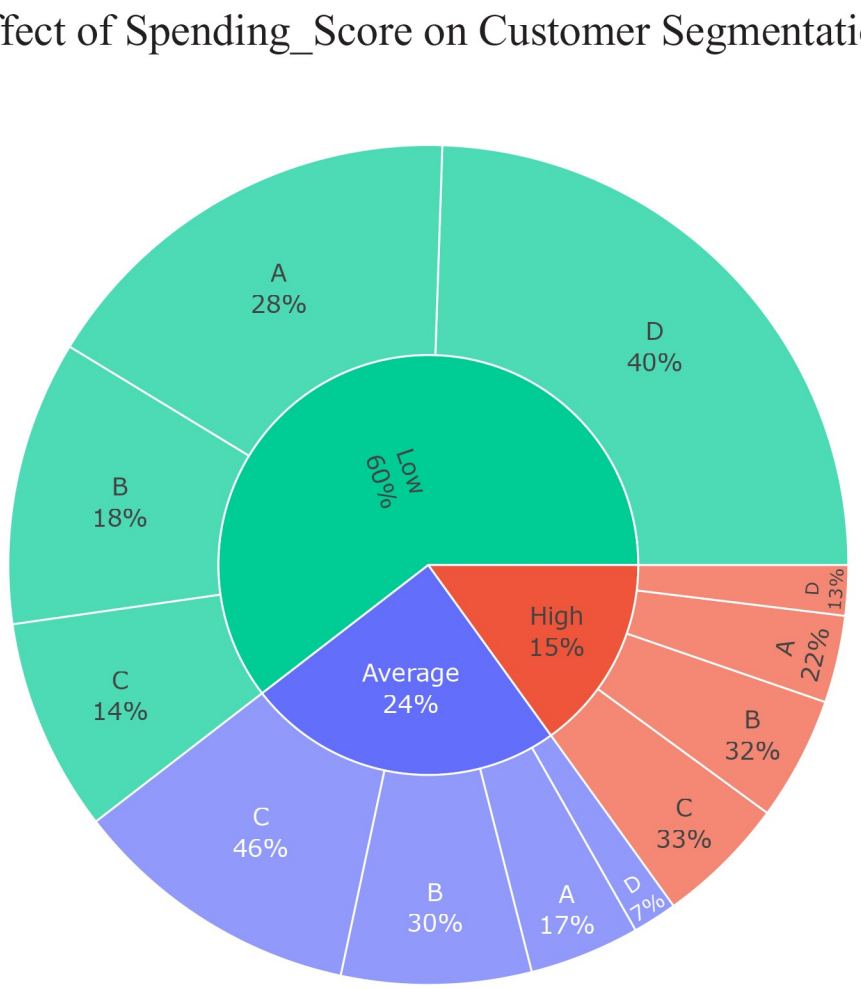
```
C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.

C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
```



```
C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.

C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
```

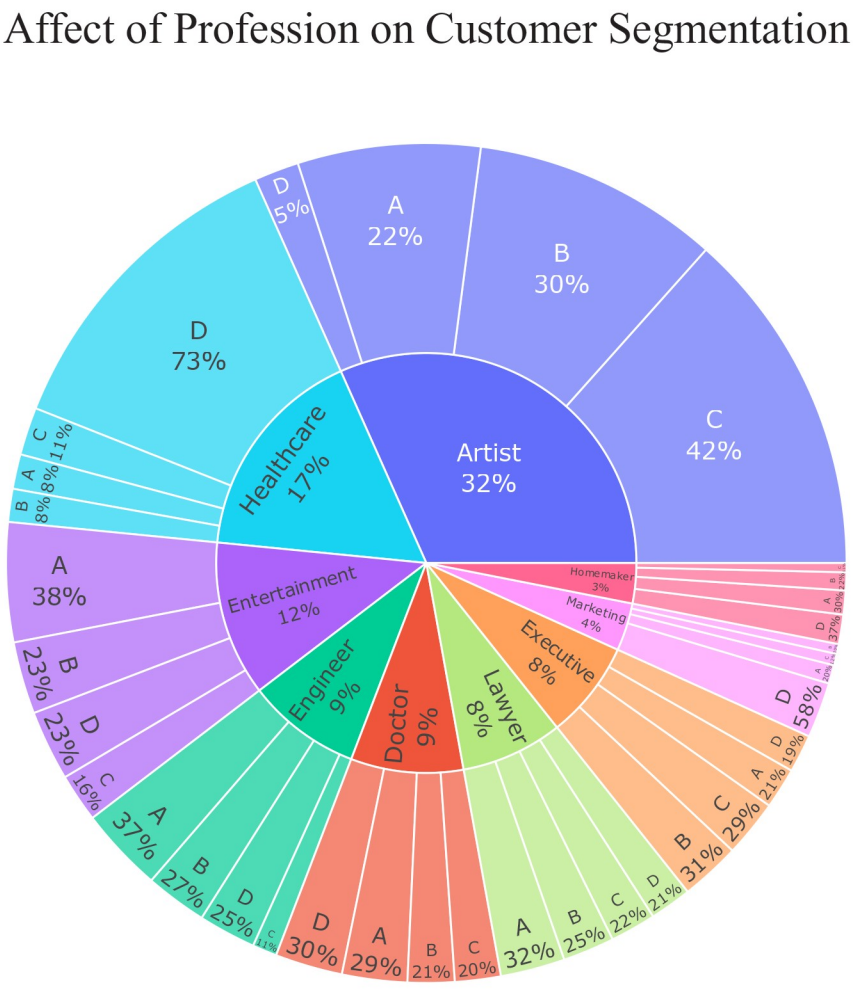


```
C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.

C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
```

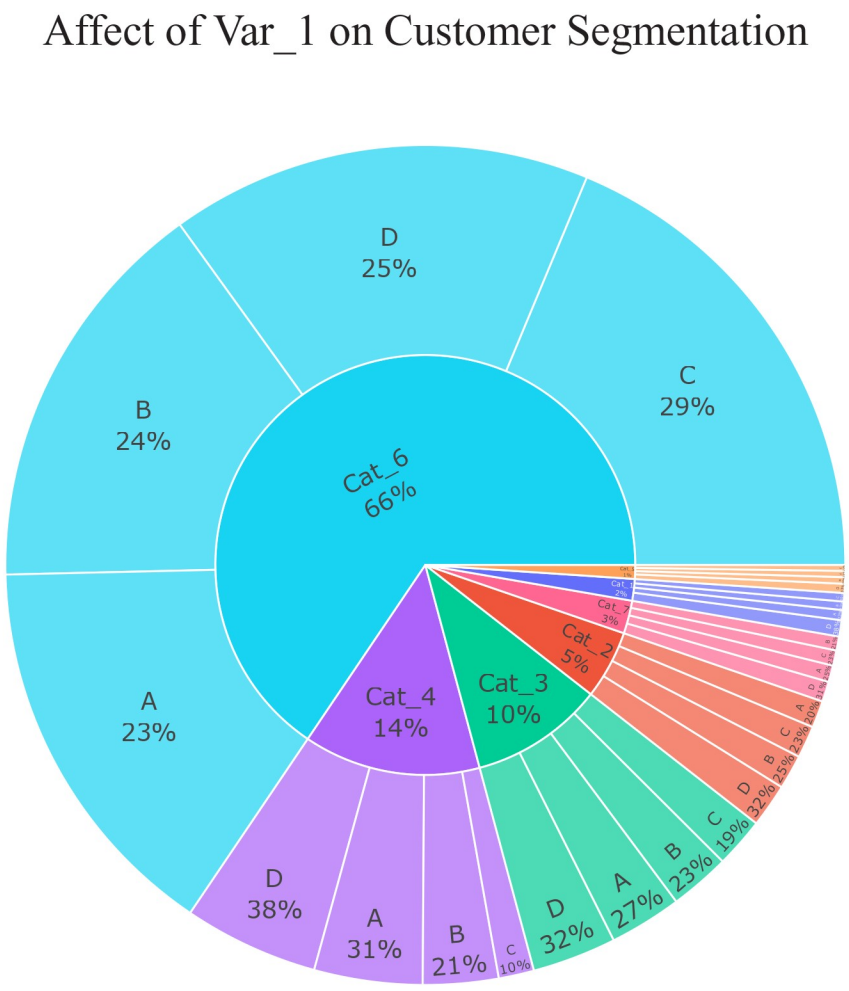


In [14]:
```
for feature in ['Profession', 'Var_1']:
    plot_pie(feature)
```

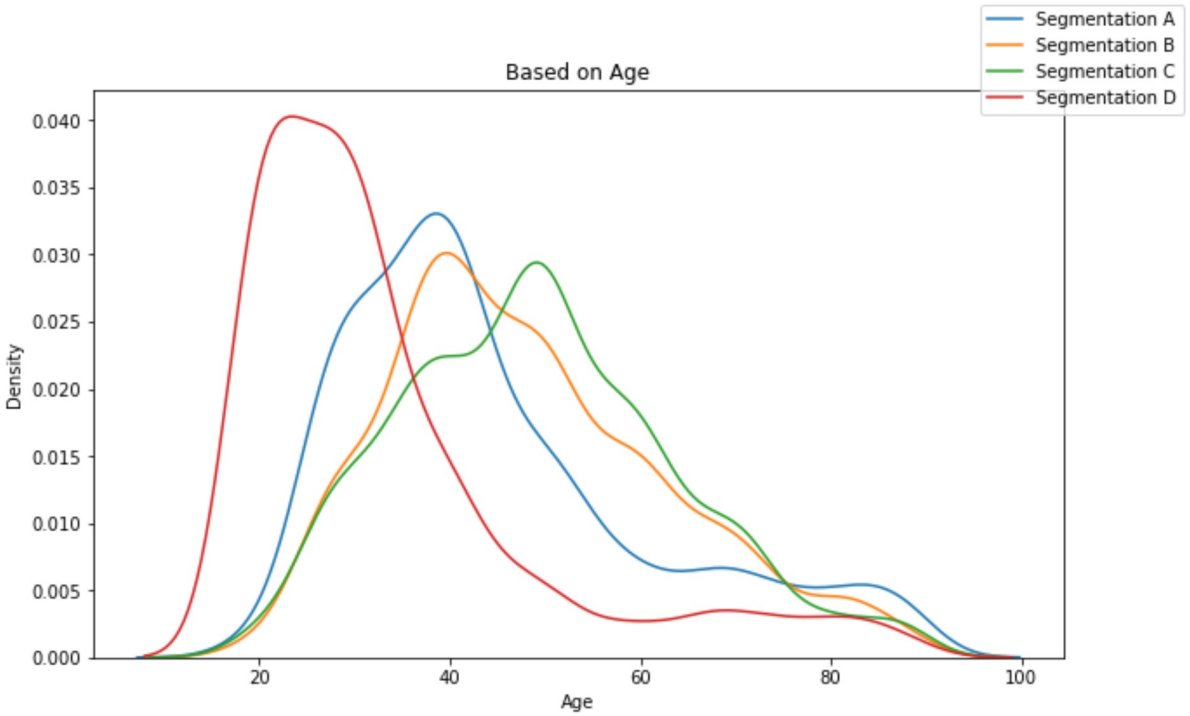
```
C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.

C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
```

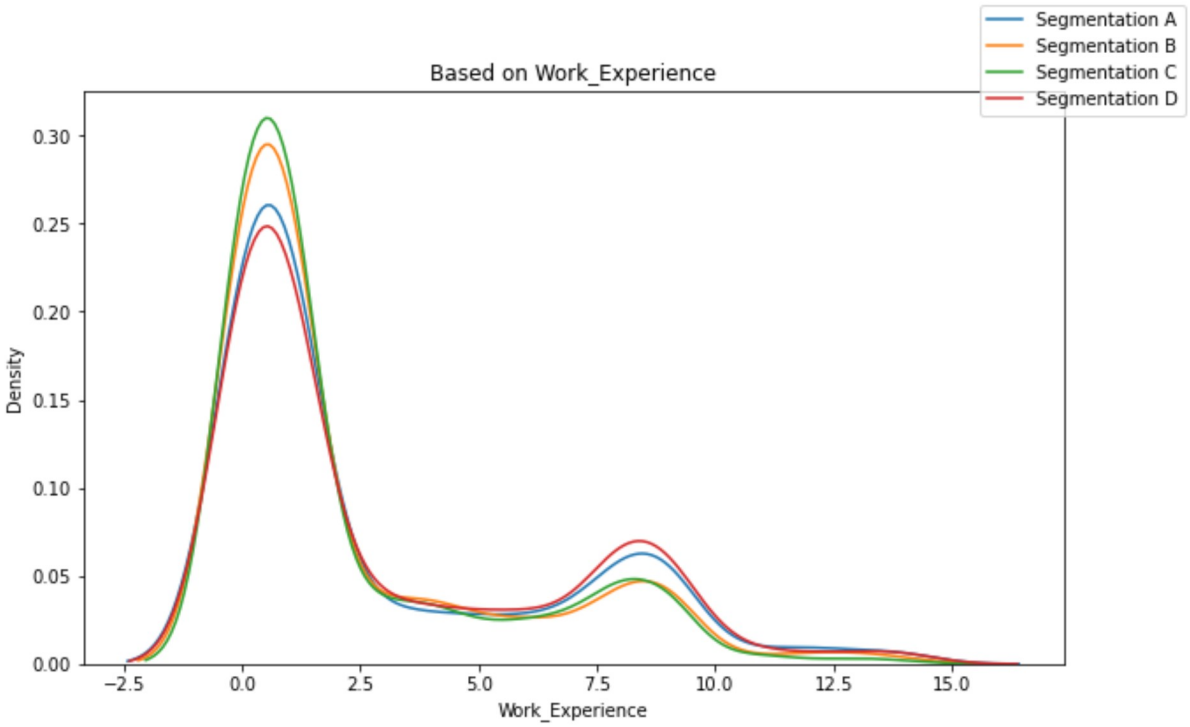


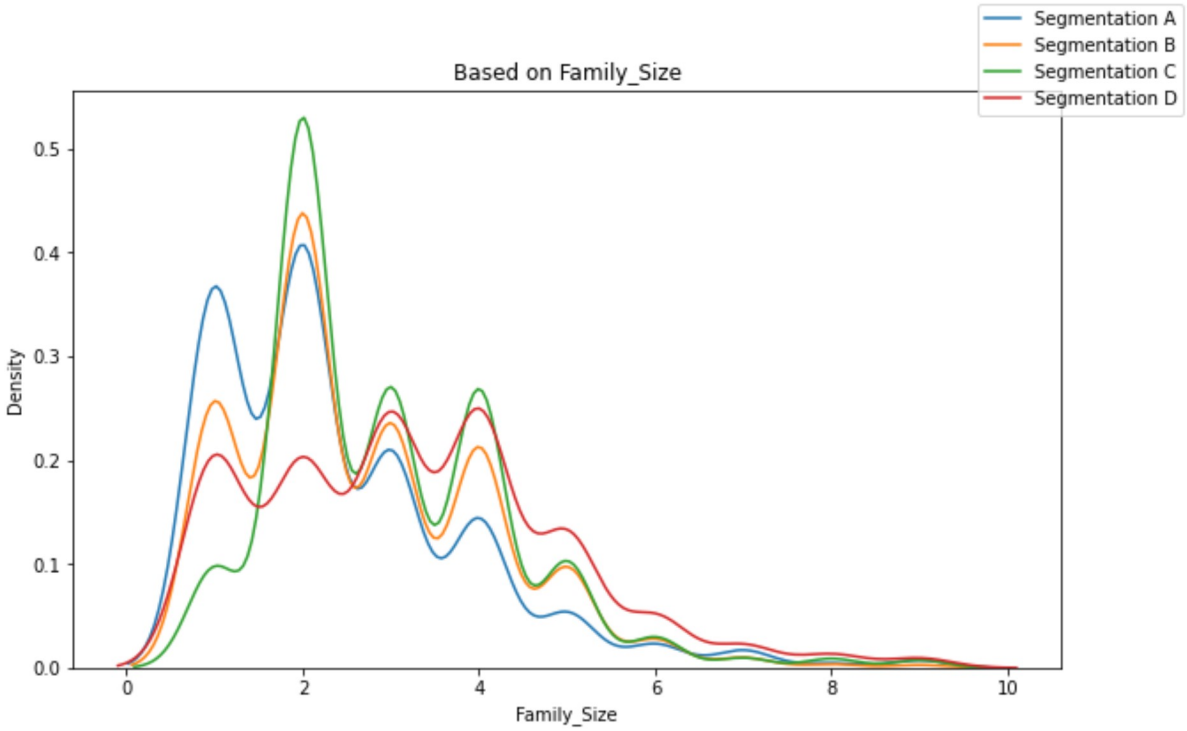
```
C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.

C:\ProgramData\Anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarn
ing:

The frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
```



In [15]:
```
for feature in ['Age', 'Work_Experience', 'Family_Size']:
    plot_numerical(feature)
```


Based on Age
Segmentation A
Segmentation B
Segmentation C
Segmentation D
Density
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
20
40
60
80
100
Age


Based on Work_Experience
Segmentation A
Segmentation B
Segmentation C
Segmentation D
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
Work_Experience



Observations-

- Ever_Married - UnMarried customers are usually in segment D while married are in segment A, B or C
- Graduated - Graduated customers are usually in segment A, B or C while Ungraduated are in segment D
- Profession - Customers in healthcare & marketing are mostly in segment D, Artist & engineers are usually in A, B or C
- Spending_Score - Usually 'Low' spenders are in segment A or D while 'high' and 'average' spenders are in segment B or C
- Age - <30 are in segment D, 30-40 or >70 are in segment A while 45-70 are in segment C
- Work_Experience - <2 are in segment C while 6-11 are in segment A & D
- Family_Size - <1 are in segment A, 1-3 are in Segment C and 4+ in segment D

In [16]:
```
categorical_features = ['Gender', 'Ever_Married', 'Graduated', 'Profession', 'Spend
numerical_features = ['Age', 'Work_Experience', 'Family_Size']

to_drop = ['ID'] # contain unique values
```

CORRELATION

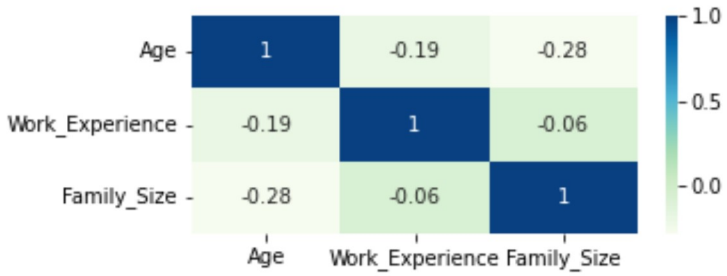
Label encoding category features for correlation

In [17]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import os
import joblib
```

In [18]:
```
df = data.copy()
path = '/working'
for i, feature in enumerate(categorical_features):
    le = LabelEncoder()

    # create directory to save label encoding models
    if not os.path.exists(os.path.join(path, "TextEncoding")):
        os.makedirs(os.path.join(path, "TextEncoding"))

    # perform label encoding
    le.fit(df[feature])

    # save the encoder
    joblib.dump(le, open(os.path.join(path, "TextEncoding/le_{}.sav".format(feature

    # transfrom training data
    df[feature] = le.transform(df[feature])

    # get classes & remove first column to elude from dummy variable trap
    columns = list(map(lambda x: feature+' '+str(x), list(le.classes_)))[1:]

    # save classes
    joblib.dump(columns,
                open(os.path.join(path, "TextEncoding/le_{}_classes.sav".format(fea
```

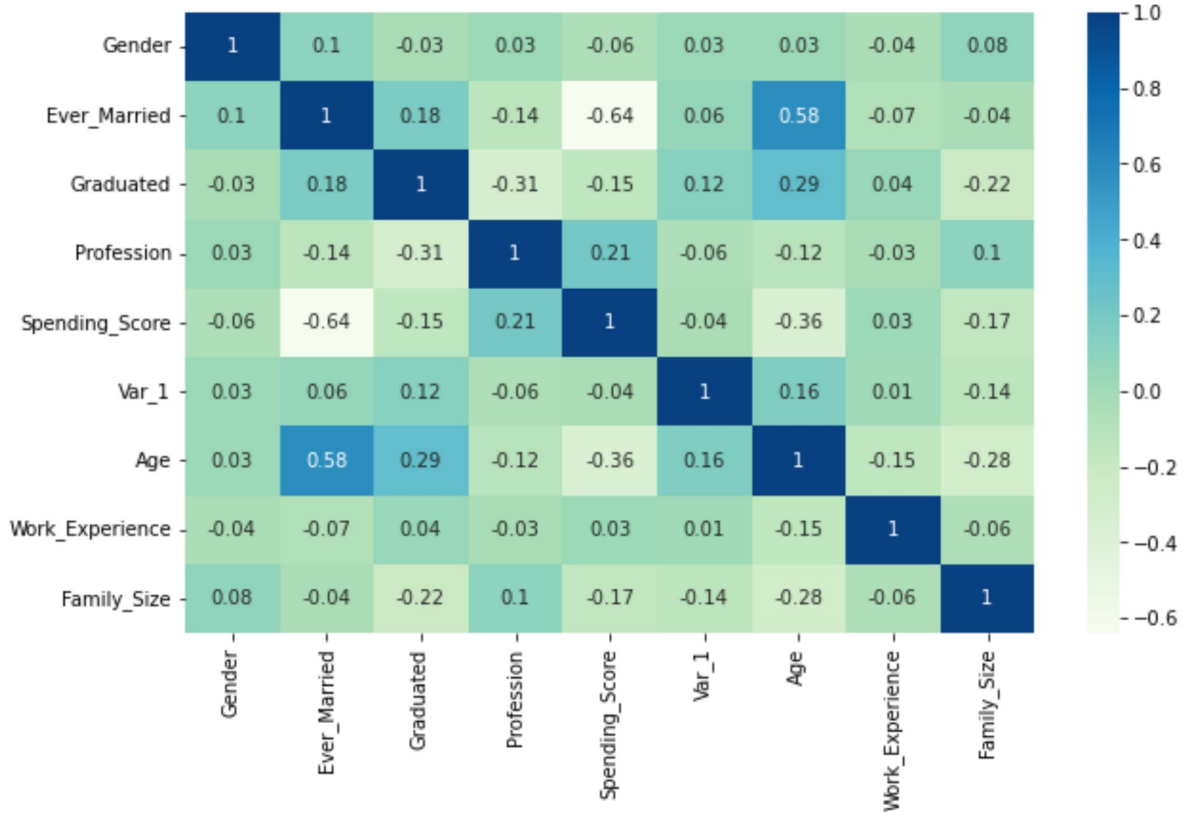
Bivariate Analysis Correlation plot with the Numeric variables

In [19]:
```
plt.figure(figsize=(5, 2))
sns.heatmap(round(data[numerical_features].corr(), 2), annot=True,
            mask=None, cmap='GnBu')
corr_mat = data[numerical_features].corr()
plt.show()
```



Bivariate Analysis Correlation plot with the Categorical variables

In [20]:
```
plt.figure(figsize=(10, 6))
sns.heatmap(round(df[categorical_features+numerical_features].corr(method='spearman
            mask=None, cmap='GnBu')
plt.show()
```



Observations-

- Ever_Married - Spending_Score
- Ever_Married - Age

Analyzing features using VIF

In [21]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [22]:
```
# Calculating VIF
vif = pd.DataFrame()
temp = df.dropna()
vif["variables"] = [feature for feature in categorical_features+numerical_features
vif["VIF"] = [variance_inflation_factor(temp[vif['variables']].values, i) for i in
print(vif)
```

```
         variables       VIF
0           Gender  2.172928
1     Ever_Married  2.547924
2        Graduated  2.526486
3       Profession  2.365052
4   Spending_Score  3.231881
5  Work_Experience  1.546193
6      Family_Size  3.222219
```

Age and Var_1 have high VIF score

Handling Missing Values

In [23]:
```
missingValueFeatures = pd.DataFrame({'missing %': data.isnull().sum()*100/len(data)
missingValueFeatures[missingValueFeatures['missing %']>0]
```

Out[23]:

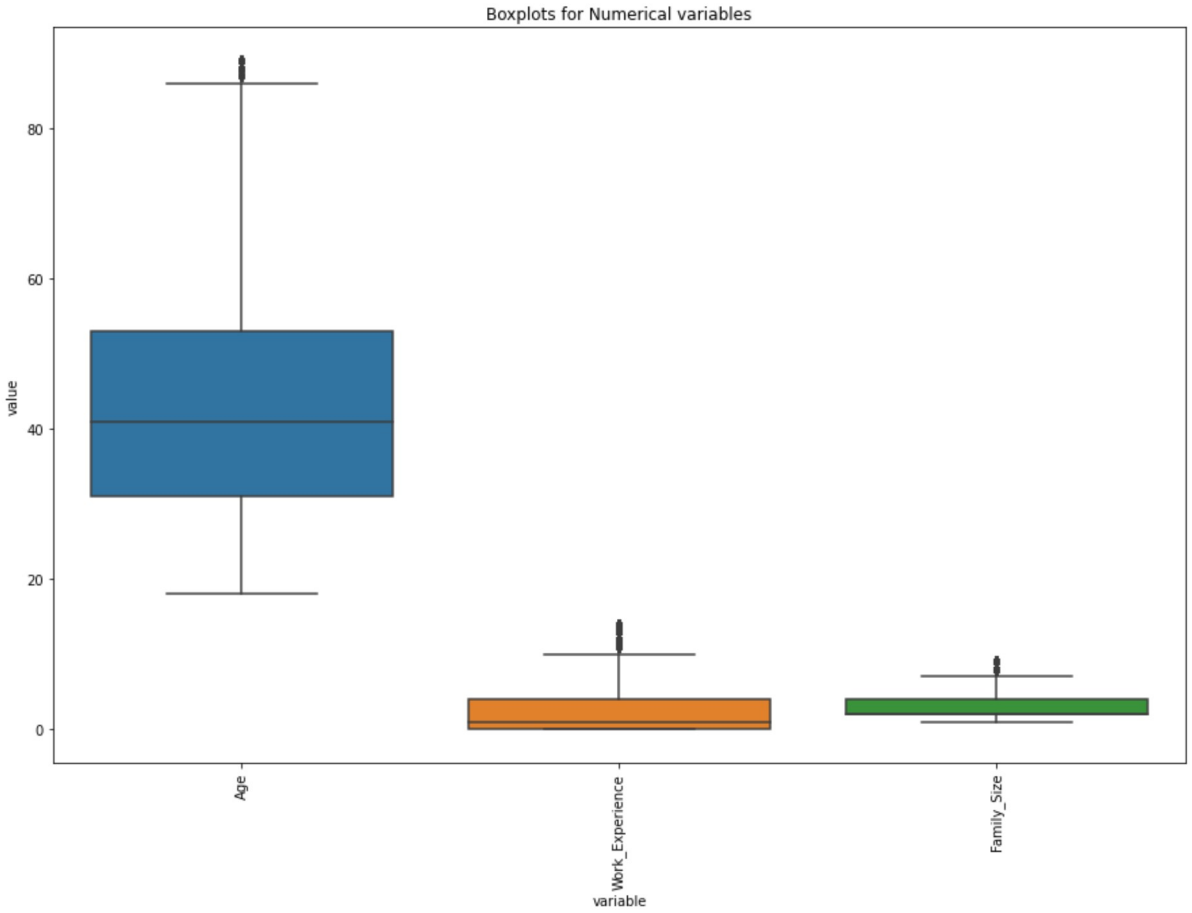
	missing %
Ever_Married	1.735250
Graduated	0.966782
Profession	1.536936
Work_Experience	10.275161
Family_Size	4.152206
Var_1	0.941993

As most of the features are uncorrelated, it is difficult to fill the NA values. Hence for the time being let's drop all NA.

In [24]:
```
data_missing = data.dropna().reset_index(drop=True)
```

Looking at Outliers

In [25]:
```
NumericData = data_missing[[feature for feature in numerical_features if feature no
NumericMelt = NumericData.melt()
plt.figure(figsize=(15,10))
plt.title("Boxplots for Numerical variables")
bp = sns.boxplot(x='variable', y='value', data=NumericMelt)
bp.set_xticklabels(bp.get_xticklabels(), rotation=90)
plt.show()
```



In [26]:
```
# Percentage of outliers present in each variable
outlier_percentage = {}
for feature in numerical_features:
    tempData = data_missing.sort_values(by=feature)[feature]
    Q1, Q3 = tempData.quantile([0.25, 0.75])
    IQR = Q3 - Q1
    Lower_range = Q1 - (1.5 * IQR)
    Upper_range = Q3 + (1.5 * IQR)
    outlier_percentage[feature] = round((((tempData<(Q1 - 1.5 * IQR)) | (tempData>(
outlier_percentage
```

Out[26]:
```
{'Age': 1.14, 'Work_Experience': 2.64, 'Family_Size': 1.2}
```

Handling Categorical Features (Label and One Hot Encoding)

In [27]:
```
df = data_missing.copy()
path = '/working'
for i, feature in enumerate(categorical_features):

    le = LabelEncoder()
    ohe = OneHotEncoder(sparse=False)

    # create directory to save label encoding models
    if not os.path.exists(os.path.join(path, "TextEncoding")):
        os.makedirs(os.path.join(path, "TextEncoding"))

    # perform label encoding
    le.fit(df[feature])
    # save the encoder
    joblib.dump(le, open(os.path.join(path, "TextEncoding/le_{}.sav".format(feature

    # transfrom training data
    df[feature] = le.transform(df[feature])

    # get classes & remove first column to elude from dummy variable trap
    columns = list(map(lambda x: feature+' '+str(x), list(le.classes_)))[1:]

    # save classes
    joblib.dump(columns,
                open(os.path.join(path, "TextEncoding/le_{}_classes.sav".format(fea
    # load classes
    columns = joblib.load(
        open(os.path.join(path, "TextEncoding/le_{}_classes.sav".format(feature)),

    if len(le.classes_)>2:
        # perform one hot encoding
        ohe.fit(df[[feature]])
        # save the encoder
        joblib.dump(ohe, open(os.path.join(path, "TextEncoding/ohe_{}.sav".format(f

        # transfrom training data
        # removing first column of encoded data to elude from dummy variable trap
        tempData = ohe.transform(df[[feature]])[:, 1:]

        # create Dataframe with columns as classes
        tempData = pd.DataFrame(tempData, columns=columns)
    else:
        tempData = df[[feature]]

    # create dataframe with all the label encoded categorical features along with h
    if i==0:
        encodedData = pd.DataFrame(data=tempData, columns=tempData.columns.values.t
    else:
        encodedData = pd.concat([encodedData, tempData], axis=1)
```

In [28]:
```
# merge numerical features and categorical encoded features
df = df[numerical_features+['Segmentation']]
df = pd.concat([df, encodedData], axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6665 entries, 0 to 6664
Data columns (total 23 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Age                        6665 non-null   int64
 1   Work_Experience            6665 non-null   float64
 2   Family_Size                6665 non-null   float64
 3   Segmentation               6665 non-null   object
 4   Gender                     6665 non-null   int32
 5   Ever_Married               6665 non-null   int32
 6   Graduated                  6665 non-null   int32
 7   Profession Doctor          6665 non-null   float64
 8   Profession Engineer        6665 non-null   float64
 9   Profession Entertainment   6665 non-null   float64
 10  Profession Executive       6665 non-null   float64
 11  Profession Healthcare      6665 non-null   float64
 12  Profession Homemaker       6665 non-null   float64
 13  Profession Lawyer          6665 non-null   float64
 14  Profession Marketing       6665 non-null   float64
 15  Spending_Score High        6665 non-null   float64
 16  Spending_Score Low         6665 non-null   float64
 17  Var_1 Cat_2                6665 non-null   float64
 18  Var_1 Cat_3                6665 non-null   float64
 19  Var_1 Cat_4                6665 non-null   float64
 20  Var_1 Cat_5                6665 non-null   float64
 21  Var_1 Cat_6                6665 non-null   float64
 22  Var_1 Cat_7                6665 non-null   float64
dtypes: float64(18), int32(3), int64(1), object(1)
memory usage: 1.1+ MB
```

Training Model

In [29]:
```
# !pip install xgboost
```

In [30]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import metrics, preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from xgboost import XGBClassifier
```

In [31]:
```
train_data = df.copy()
feature_cols = [feature for feature in train_data.columns if feature not in(['Segme
print('features used- ', feature_cols)

''' Rescaling to [0,1] '''
scaler = StandardScaler()
scaler.fit(train_data[feature_cols])
train_data[feature_cols] = scaler.transform(train_data[feature_cols])
```

```
features used-  ['Age', 'Work_Experience', 'Family_Size', 'Gender', 'Ever_Married',
'Graduated', 'Profession Doctor', 'Profession Engineer', 'Profession Entertainment
', 'Profession Executive', 'Profession Healthcare', 'Profession Homemaker', 'Profes
sion Lawyer', 'Profession Marketing', 'Spending_Score High', 'Spending_Score Low',
'Var_1 Cat_2', 'Var_1 Cat_3', 'Var_1 Cat_4', 'Var_1 Cat_5', 'Var_1 Cat_6', 'Var_1 C
at_7']
```

In [32]:
```
train_data['Segmentation'] = train_data['Segmentation'].map({'A':0, 'B':1, 'C':2, '
X = train_data[feature_cols]
y = train_data['Segmentation']

validation_size = 0.25
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=validation_size
                                                    random_state=0, stratify=y)
```

Model 1: Logistic Regression

In [33]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[33]:
```
LogisticRegression()
```

In [34]:
```
y_pred = model.predict(X_train)

print('Train metrics...')
print(confusion_matrix(y_train, y_pred))
print(classification_report(y_train, y_pred))

y_pred = model.predict(X_test)

print('Validation metrics...')
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Train metrics...
[[609 150 235 218]
 [339 267 453 120]
 [168 157 810 155]
 [270  69  64 914]]
              precision    recall  f1-score   support

           0       0.44      0.50      0.47      1212
           1       0.42      0.23      0.29      1179
           2       0.52      0.63      0.57      1290
           3       0.65      0.69      0.67      1317

    accuracy                           0.52      4998
   macro avg       0.51      0.51      0.50      4998
weighted avg       0.51      0.52      0.51      4998

Validation metrics...
[[211  54  70  69]
 [120  90 135  48]
 [ 54  40 285  51]
 [ 96  22  19 303]]
              precision    recall  f1-score   support

           0       0.44      0.52      0.48       404
           1       0.44      0.23      0.30       393
           2       0.56      0.66      0.61       430
           3       0.64      0.69      0.67       440

    accuracy                           0.53      1667
   macro avg       0.52      0.53      0.51      1667
weighted avg       0.52      0.53      0.52      1667
```

In [35]:

```
''' metrics on original data '''
y_pred = model.predict(train_data[feature_cols])

def make_cm(matrix, columns):
    n = len(columns)
    act = ['actual Segmentation'] * n
    pred = ['predicted Segmentation'] * n

    cm = pd.DataFrame(matrix,
        columns=[pred, columns], index=[act, columns])
    return cm

df_matrix=make_cm(
    confusion_matrix(train_data['Segmentation'], y_pred),['A', 'B', 'C', 'D'])

display(df_matrix)
print(classification_report(train_data['Segmentation'], y_pred))
```

		predicted Segmentation			
		A	**B**	**C**	**D**
actual Segmentation	**A**	820	204	305	287
	B	459	357	588	168
	C	222	197	1095	206
	D	366	91	83	1217

	precision	recall	f1-score	support
0	0.44	0.51	0.47	1616
1	0.42	0.23	0.29	1572
2	0.53	0.64	0.58	1720
3	0.65	0.69	0.67	1757
accuracy			0.52	6665
macro avg	0.51	0.52	0.50	6665
weighted avg	0.51	0.52	0.51	6665

Model 2: XGB

In [36]:
```
model = XGBClassifier(
    learning_rate=0.05,
    max_depth=3,
    min_child_weight=5,
    n_estimators=1000,
    random_state=7,
    reg_lambda=1.5,
    reg_alpha=0.5,
    use_label_encoder=False
)

model.fit(X_train, y_train,
          eval_metric='mlogloss',
          verbose=False)
```

```
C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\xgboost\sklearn.py:793:
UserWarning:

`eval_metric` in `fit` method is deprecated for better compatibility with scikit-le
arn, use `eval_metric` in constructor or`set_params` instead.
```

Out[36]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=3, max_leaves=0, min_child_weight=5,
              missing=nan, monotone_constraints='()', n_estimators=1000,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=7, reg_alpha=0.5, ...)
```

In [37]:
```
y_pred = model.predict(X_train)

print('Train metrics...')
print(confusion_matrix(y_train, y_pred))
print(classification_report(y_train, y_pred))

y_pred = model.predict(X_test)

print('Test metrics...')
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Train metrics...
[[ 736  173  135  168]
 [ 234  513  313  119]
 [ 112  171  859  148]
 [ 204   64   27 1022]]
              precision    recall  f1-score   support

           0       0.57      0.61      0.59      1212
           1       0.56      0.44      0.49      1179
           2       0.64      0.67      0.65      1290
           3       0.70      0.78      0.74      1317

    accuracy                           0.63      4998
   macro avg       0.62      0.62      0.62      4998
weighted avg       0.62      0.63      0.62      4998

Test metrics...
[[191  89  45  79]
 [106 126 114  47]
 [ 42  73 265  50]
 [ 89  27  24 300]]
              precision    recall  f1-score   support

           0       0.45      0.47      0.46       404
           1       0.40      0.32      0.36       393
           2       0.59      0.62      0.60       430
           3       0.63      0.68      0.66       440

    accuracy                           0.53      1667
   macro avg       0.52      0.52      0.52      1667
weighted avg       0.52      0.53      0.52      1667
```

In [38]:
```
''' metrics on original data '''
y_pred = model.predict(train_data[feature_cols])

def make_cm(matrix, columns):
    n = len(columns)
    act = ['actual Segmentation'] * n
    pred = ['predicted Segmentation'] * n

    cm = pd.DataFrame(matrix,
        columns=[pred, columns], index=[act, columns])
    return cm

df_matrix=make_cm(
    confusion_matrix(train_data['Segmentation'], y_pred),['A', 'B', 'C', 'D'])

display(df_matrix)
print(classification_report(train_data['Segmentation'], y_pred))
```

		predicted Segmentation			
		A	**B**	**C**	**D**
actual Segmentation	**A**	927	262	180	247
	B	340	639	427	166
	C	154	244	1124	198
	D	293	91	51	1322

```
              precision    recall  f1-score   support

           0       0.54      0.57      0.56      1616
           1       0.52      0.41      0.46      1572
           2       0.63      0.65      0.64      1720
           3       0.68      0.75      0.72      1757

    accuracy                           0.60      6665
   macro avg       0.59      0.60      0.59      6665
weighted avg       0.60      0.60      0.60      6665
```

Model 3: ANN

In [39]:
```
# !pip uninstall keras
```

In [40]:
```
# !pip install tensorflow
```

In [41]:
```
np.random.seed(123)  # for reproducibility

import keras
from tensorflow.python.keras.models import Input
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout
```

In [42]:
```
# Initialize the constructor
model = Sequential()
# Add an input layer
# arguemtns of dense: output shape, activation, input shape
# activation(define the output function) = [relu', 'tanh']
model.add(Dense(64, activation='relu', input_shape=(len(feature_cols),)))
model.add(Dropout(0.3))
# Add one hidden layer
# after first layer no need to give input shape
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(8, activation='relu'))
# Add an output layer
# activation = [regression: 'linear', binary classification: 'sigmoid', multiclass:
# threshold: >0 or <0
# sigmoid: 1/(1+e^-x), usually applied in output layer
# rectifier: max(x,0), usually applied in hidden layer: relu
# hyperbolic tangent tanh: (1-e^-2x)/(1+e^-2x)
model.add(Dense(4, activation='softmax'))

# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
# loss = [regression: 'mse', binary: 'binary_crossentropy', multi: 'categorical_cro
model.compile(loss='categorical_crossentropy',
              optimizer='RMSprop',
              metrics=['accuracy']
         )

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1472

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 16)                528

 dropout_2 (Dropout)         (None, 16)                0

 dense_3 (Dense)             (None, 8)                 136

 dense_4 (Dense)             (None, 4)                 36

=================================================================
Total params: 4,252
Trainable params: 4,252
Non-trainable params: 0
_________________________________________________________________
None
```

In [43]:
```
fit = model.fit(X_train, pd.get_dummies(y_train), epochs=150, batch_size=500, verbo
```

```
Epoch 1/150
8/8 [==============================] - 1s 41ms/step - loss: 1.4105 - accuracy: 0.27
61 - val_loss: 1.3701 - val_accuracy: 0.3820
Epoch 2/150
8/8 [==============================] - 0s 6ms/step - loss: 1.3843 - accuracy: 0.311
4 - val_loss: 1.3529 - val_accuracy: 0.4060
Epoch 3/150
8/8 [==============================] - 0s 6ms/step - loss: 1.3643 - accuracy: 0.344
2 - val_loss: 1.3326 - val_accuracy: 0.4260
Epoch 4/150
8/8 [==============================] - 0s 6ms/step - loss: 1.3423 - accuracy: 0.374
9 - val_loss: 1.3100 - val_accuracy: 0.4380
Epoch 5/150
8/8 [==============================] - 0s 5ms/step - loss: 1.3193 - accuracy: 0.380
9 - val_loss: 1.2818 - val_accuracy: 0.4520
Epoch 6/150
8/8 [==============================] - 0s 5ms/step - loss: 1.3011 - accuracy: 0.401
7 - val_loss: 1.2538 - val_accuracy: 0.4630
Epoch 7/150
8/8 [==============================] - 0s 6ms/step - loss: 1.2868 - accuracy: 0.409
5 - val_loss: 1.2348 - val_accuracy: 0.4670
Epoch 8/150
8/8 [==============================] - 0s 6ms/step - loss: 1.2689 - accuracy: 0.404
2 - val_loss: 1.2156 - val_accuracy: 0.4740
Epoch 9/150
8/8 [==============================] - 0s 6ms/step - loss: 1.2446 - accuracy: 0.431
7 - val_loss: 1.1999 - val_accuracy: 0.4770
Epoch 10/150
8/8 [==============================] - 0s 6ms/step - loss: 1.2431 - accuracy: 0.435
5 - val_loss: 1.1882 - val_accuracy: 0.4840
Epoch 11/150
8/8 [==============================] - 0s 5ms/step - loss: 1.2300 - accuracy: 0.433
5 - val_loss: 1.1820 - val_accuracy: 0.4760
Epoch 12/150
8/8 [==============================] - 0s 6ms/step - loss: 1.2147 - accuracy: 0.453
0 - val_loss: 1.1677 - val_accuracy: 0.4920
Epoch 13/150
8/8 [==============================] - 0s 5ms/step - loss: 1.2178 - accuracy: 0.446
7 - val_loss: 1.1618 - val_accuracy: 0.4930
Epoch 14/150
8/8 [==============================] - 0s 5ms/step - loss: 1.2024 - accuracy: 0.454
7 - val_loss: 1.1566 - val_accuracy: 0.4940
Epoch 15/150
8/8 [==============================] - 0s 5ms/step - loss: 1.2125 - accuracy: 0.454
0 - val_loss: 1.1537 - val_accuracy: 0.4960
Epoch 16/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1912 - accuracy: 0.467
7 - val_loss: 1.1495 - val_accuracy: 0.4960
Epoch 17/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1899 - accuracy: 0.466
2 - val_loss: 1.1453 - val_accuracy: 0.5060
Epoch 18/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1790 - accuracy: 0.476
7 - val_loss: 1.1411 - val_accuracy: 0.5090
Epoch 19/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1796 - accuracy: 0.479
0 - val_loss: 1.1381 - val_accuracy: 0.5090
Epoch 20/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1837 - accuracy: 0.477
```

```
0 - val_loss: 1.1377 - val_accuracy: 0.5090
Epoch 21/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1693 - accuracy: 0.477
2 - val_loss: 1.1348 - val_accuracy: 0.5120
Epoch 22/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1674 - accuracy: 0.481
5 - val_loss: 1.1349 - val_accuracy: 0.5090
Epoch 23/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1678 - accuracy: 0.475
5 - val_loss: 1.1333 - val_accuracy: 0.5070
Epoch 24/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1631 - accuracy: 0.486
5 - val_loss: 1.1316 - val_accuracy: 0.5040
Epoch 25/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1505 - accuracy: 0.498
5 - val_loss: 1.1276 - val_accuracy: 0.5050
Epoch 26/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1413 - accuracy: 0.490
5 - val_loss: 1.1257 - val_accuracy: 0.5060
Epoch 27/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1514 - accuracy: 0.489
2 - val_loss: 1.1255 - val_accuracy: 0.5040
Epoch 28/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1417 - accuracy: 0.485
7 - val_loss: 1.1235 - val_accuracy: 0.5080
Epoch 29/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1441 - accuracy: 0.487
2 - val_loss: 1.1221 - val_accuracy: 0.5010
Epoch 30/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1257 - accuracy: 0.499
7 - val_loss: 1.1209 - val_accuracy: 0.5030
Epoch 31/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1351 - accuracy: 0.483
5 - val_loss: 1.1207 - val_accuracy: 0.5030
Epoch 32/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1319 - accuracy: 0.498
0 - val_loss: 1.1202 - val_accuracy: 0.5080
Epoch 33/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1299 - accuracy: 0.507
0 - val_loss: 1.1198 - val_accuracy: 0.5050
Epoch 34/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1433 - accuracy: 0.494
7 - val_loss: 1.1200 - val_accuracy: 0.5090
Epoch 35/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1310 - accuracy: 0.496
2 - val_loss: 1.1182 - val_accuracy: 0.5070
Epoch 36/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1264 - accuracy: 0.501
0 - val_loss: 1.1184 - val_accuracy: 0.5110
Epoch 37/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1137 - accuracy: 0.518
3 - val_loss: 1.1165 - val_accuracy: 0.5080
Epoch 38/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1168 - accuracy: 0.512
0 - val_loss: 1.1154 - val_accuracy: 0.5060
Epoch 39/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1298 - accuracy: 0.506
5 - val_loss: 1.1151 - val_accuracy: 0.5050
Epoch 40/150
```

```
8/8 [==============================] - 0s 5ms/step - loss: 1.1213 - accuracy: 0.499
7 - val_loss: 1.1147 - val_accuracy: 0.5130
Epoch 41/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1223 - accuracy: 0.504
3 - val_loss: 1.1144 - val_accuracy: 0.5060
Epoch 42/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1129 - accuracy: 0.512
8 - val_loss: 1.1114 - val_accuracy: 0.5060
Epoch 43/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1041 - accuracy: 0.512
8 - val_loss: 1.1099 - val_accuracy: 0.5070
Epoch 44/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1166 - accuracy: 0.504
5 - val_loss: 1.1108 - val_accuracy: 0.5060
Epoch 45/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1045 - accuracy: 0.516
0 - val_loss: 1.1090 - val_accuracy: 0.5100
Epoch 46/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1182 - accuracy: 0.503
5 - val_loss: 1.1092 - val_accuracy: 0.5110
Epoch 47/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1146 - accuracy: 0.512
8 - val_loss: 1.1085 - val_accuracy: 0.5140
Epoch 48/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1144 - accuracy: 0.508
0 - val_loss: 1.1101 - val_accuracy: 0.5140
Epoch 49/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1187 - accuracy: 0.515
0 - val_loss: 1.1110 - val_accuracy: 0.5170
Epoch 50/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1054 - accuracy: 0.521
5 - val_loss: 1.1082 - val_accuracy: 0.5150
Epoch 51/150
8/8 [==============================] - 0s 6ms/step - loss: 1.1045 - accuracy: 0.515
8 - val_loss: 1.1069 - val_accuracy: 0.5150
Epoch 52/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0945 - accuracy: 0.525
8 - val_loss: 1.1071 - val_accuracy: 0.5140
Epoch 53/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0962 - accuracy: 0.518
0 - val_loss: 1.1060 - val_accuracy: 0.5150
Epoch 54/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1010 - accuracy: 0.519
8 - val_loss: 1.1059 - val_accuracy: 0.5200
Epoch 55/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0938 - accuracy: 0.522
0 - val_loss: 1.1063 - val_accuracy: 0.5250
Epoch 56/150
8/8 [==============================] - 0s 5ms/step - loss: 1.1048 - accuracy: 0.519
0 - val_loss: 1.1056 - val_accuracy: 0.5200
Epoch 57/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0982 - accuracy: 0.515
8 - val_loss: 1.1045 - val_accuracy: 0.5190
Epoch 58/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0865 - accuracy: 0.524
5 - val_loss: 1.1036 - val_accuracy: 0.5210
Epoch 59/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0851 - accuracy: 0.527
3 - val_loss: 1.1040 - val_accuracy: 0.5190
```

```
Epoch 60/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0930 - accuracy: 0.518
0 - val_loss: 1.1060 - val_accuracy: 0.5170
Epoch 61/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0990 - accuracy: 0.522
3 - val_loss: 1.1056 - val_accuracy: 0.5160
Epoch 62/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0875 - accuracy: 0.521
0 - val_loss: 1.1061 - val_accuracy: 0.5190
Epoch 63/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0853 - accuracy: 0.526
0 - val_loss: 1.1055 - val_accuracy: 0.5210
Epoch 64/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0987 - accuracy: 0.518
3 - val_loss: 1.1059 - val_accuracy: 0.5200
Epoch 65/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0844 - accuracy: 0.516
3 - val_loss: 1.1055 - val_accuracy: 0.5210
Epoch 66/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0924 - accuracy: 0.519
8 - val_loss: 1.1058 - val_accuracy: 0.5150
Epoch 67/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0928 - accuracy: 0.521
5 - val_loss: 1.1051 - val_accuracy: 0.5160
Epoch 68/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0818 - accuracy: 0.524
0 - val_loss: 1.1042 - val_accuracy: 0.5250
Epoch 69/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0756 - accuracy: 0.532
5 - val_loss: 1.1031 - val_accuracy: 0.5180
Epoch 70/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0919 - accuracy: 0.524
5 - val_loss: 1.1030 - val_accuracy: 0.5240
Epoch 71/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0807 - accuracy: 0.528
5 - val_loss: 1.1027 - val_accuracy: 0.5170
Epoch 72/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0834 - accuracy: 0.526
8 - val_loss: 1.1026 - val_accuracy: 0.5190
Epoch 73/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0872 - accuracy: 0.530
5 - val_loss: 1.1022 - val_accuracy: 0.5210
Epoch 74/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0893 - accuracy: 0.524
5 - val_loss: 1.1028 - val_accuracy: 0.5220
Epoch 75/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0783 - accuracy: 0.533
0 - val_loss: 1.1017 - val_accuracy: 0.5200
Epoch 76/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0787 - accuracy: 0.528
5 - val_loss: 1.1006 - val_accuracy: 0.5200
Epoch 77/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0767 - accuracy: 0.530
0 - val_loss: 1.1005 - val_accuracy: 0.5180
Epoch 78/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0839 - accuracy: 0.522
5 - val_loss: 1.1003 - val_accuracy: 0.5200
Epoch 79/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0793 - accuracy: 0.540
```

```
8 - val_loss: 1.1004 - val_accuracy: 0.5180
Epoch 80/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0714 - accuracy: 0.532
3 - val_loss: 1.0996 - val_accuracy: 0.5150
Epoch 81/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0765 - accuracy: 0.540
0 - val_loss: 1.0993 - val_accuracy: 0.5160
Epoch 82/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0720 - accuracy: 0.534
8 - val_loss: 1.0995 - val_accuracy: 0.5160
Epoch 83/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0703 - accuracy: 0.534
0 - val_loss: 1.1012 - val_accuracy: 0.5220
Epoch 84/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0664 - accuracy: 0.532
8 - val_loss: 1.0983 - val_accuracy: 0.5210
Epoch 85/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0774 - accuracy: 0.530
0 - val_loss: 1.0990 - val_accuracy: 0.5170
Epoch 86/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0738 - accuracy: 0.542
8 - val_loss: 1.0997 - val_accuracy: 0.5220
Epoch 87/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0612 - accuracy: 0.529
8 - val_loss: 1.0987 - val_accuracy: 0.5210
Epoch 88/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0653 - accuracy: 0.533
5 - val_loss: 1.0992 - val_accuracy: 0.5240
Epoch 89/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0761 - accuracy: 0.536
8 - val_loss: 1.1021 - val_accuracy: 0.5240
Epoch 90/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0767 - accuracy: 0.520
0 - val_loss: 1.0994 - val_accuracy: 0.5290
Epoch 91/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0714 - accuracy: 0.531
0 - val_loss: 1.0984 - val_accuracy: 0.5230
Epoch 92/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0751 - accuracy: 0.529
0 - val_loss: 1.0987 - val_accuracy: 0.5200
Epoch 93/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0642 - accuracy: 0.536
3 - val_loss: 1.0968 - val_accuracy: 0.5250
Epoch 94/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0649 - accuracy: 0.537
3 - val_loss: 1.0963 - val_accuracy: 0.5230
Epoch 95/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0712 - accuracy: 0.536
3 - val_loss: 1.0967 - val_accuracy: 0.5200
Epoch 96/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0616 - accuracy: 0.533
8 - val_loss: 1.0970 - val_accuracy: 0.5170
Epoch 97/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0699 - accuracy: 0.540
8 - val_loss: 1.0972 - val_accuracy: 0.5180
Epoch 98/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0587 - accuracy: 0.540
8 - val_loss: 1.0994 - val_accuracy: 0.5220
Epoch 99/150
```

```
8/8 [==============================] - 0s 6ms/step - loss: 1.0646 - accuracy: 0.528
5 - val_loss: 1.0972 - val_accuracy: 0.5150
Epoch 100/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0656 - accuracy: 0.529
3 - val_loss: 1.0971 - val_accuracy: 0.5190
Epoch 101/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0613 - accuracy: 0.538
8 - val_loss: 1.0962 - val_accuracy: 0.5170
Epoch 102/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0655 - accuracy: 0.539
8 - val_loss: 1.0964 - val_accuracy: 0.5150
Epoch 103/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0595 - accuracy: 0.539
0 - val_loss: 1.0953 - val_accuracy: 0.5180
Epoch 104/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0582 - accuracy: 0.533
3 - val_loss: 1.0974 - val_accuracy: 0.5220
Epoch 105/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0669 - accuracy: 0.535
0 - val_loss: 1.0984 - val_accuracy: 0.5210
Epoch 106/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0616 - accuracy: 0.540
3 - val_loss: 1.0997 - val_accuracy: 0.5200
Epoch 107/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0481 - accuracy: 0.543
5 - val_loss: 1.0995 - val_accuracy: 0.5200
Epoch 108/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0553 - accuracy: 0.545
3 - val_loss: 1.0996 - val_accuracy: 0.5240
Epoch 109/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0636 - accuracy: 0.536
8 - val_loss: 1.0979 - val_accuracy: 0.5250
Epoch 110/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0546 - accuracy: 0.542
0 - val_loss: 1.0958 - val_accuracy: 0.5240
Epoch 111/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0545 - accuracy: 0.537
3 - val_loss: 1.0963 - val_accuracy: 0.5230
Epoch 112/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0624 - accuracy: 0.533
3 - val_loss: 1.0963 - val_accuracy: 0.5220
Epoch 113/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0577 - accuracy: 0.545
0 - val_loss: 1.0970 - val_accuracy: 0.5210
Epoch 114/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0578 - accuracy: 0.541
8 - val_loss: 1.0955 - val_accuracy: 0.5230
Epoch 115/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0563 - accuracy: 0.536
0 - val_loss: 1.0965 - val_accuracy: 0.5240
Epoch 116/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0483 - accuracy: 0.549
0 - val_loss: 1.0968 - val_accuracy: 0.5200
Epoch 117/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0597 - accuracy: 0.540
5 - val_loss: 1.0969 - val_accuracy: 0.5210
Epoch 118/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0634 - accuracy: 0.544
8 - val_loss: 1.0973 - val_accuracy: 0.5270
```

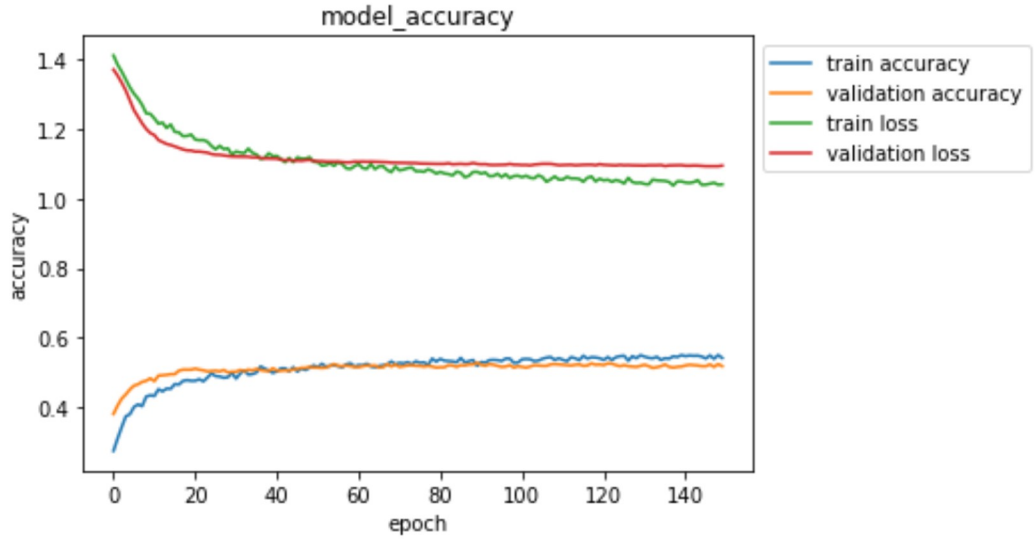
```
Epoch 119/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0493 - accuracy: 0.544
0 - val_loss: 1.0957 - val_accuracy: 0.5230
Epoch 120/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0569 - accuracy: 0.541
8 - val_loss: 1.0986 - val_accuracy: 0.5240
Epoch 121/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0521 - accuracy: 0.536
8 - val_loss: 1.0968 - val_accuracy: 0.5220
Epoch 122/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0542 - accuracy: 0.545
3 - val_loss: 1.0965 - val_accuracy: 0.5270
Epoch 123/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0479 - accuracy: 0.536
5 - val_loss: 1.0957 - val_accuracy: 0.5260
Epoch 124/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0533 - accuracy: 0.545
0 - val_loss: 1.0951 - val_accuracy: 0.5220
Epoch 125/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0497 - accuracy: 0.547
8 - val_loss: 1.0966 - val_accuracy: 0.5190
Epoch 126/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0431 - accuracy: 0.549
0 - val_loss: 1.0955 - val_accuracy: 0.5220
Epoch 127/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0594 - accuracy: 0.534
5 - val_loss: 1.0960 - val_accuracy: 0.5190
Epoch 128/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0473 - accuracy: 0.547
8 - val_loss: 1.0954 - val_accuracy: 0.5180
Epoch 129/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0444 - accuracy: 0.541
8 - val_loss: 1.0954 - val_accuracy: 0.5170
Epoch 130/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0456 - accuracy: 0.550
5 - val_loss: 1.0950 - val_accuracy: 0.5200
Epoch 131/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0418 - accuracy: 0.546
3 - val_loss: 1.0946 - val_accuracy: 0.5230
Epoch 132/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0406 - accuracy: 0.543
0 - val_loss: 1.0955 - val_accuracy: 0.5190
Epoch 133/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0545 - accuracy: 0.541
5 - val_loss: 1.0954 - val_accuracy: 0.5140
Epoch 134/150
8/8 [==============================] - 0s 6ms/step - loss: 1.0520 - accuracy: 0.546
0 - val_loss: 1.0955 - val_accuracy: 0.5160
Epoch 135/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0516 - accuracy: 0.536
8 - val_loss: 1.0962 - val_accuracy: 0.5190
Epoch 136/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0507 - accuracy: 0.541
8 - val_loss: 1.0932 - val_accuracy: 0.5240
Epoch 137/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0476 - accuracy: 0.540
5 - val_loss: 1.0951 - val_accuracy: 0.5170
Epoch 138/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0373 - accuracy: 0.547
```

```
3 - val_loss: 1.0952 - val_accuracy: 0.5150
Epoch 139/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0493 - accuracy: 0.545
3 - val_loss: 1.0953 - val_accuracy: 0.5170
Epoch 140/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0493 - accuracy: 0.550
8 - val_loss: 1.0958 - val_accuracy: 0.5180
Epoch 141/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0458 - accuracy: 0.547
8 - val_loss: 1.0943 - val_accuracy: 0.5210
Epoch 142/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0448 - accuracy: 0.549
8 - val_loss: 1.0942 - val_accuracy: 0.5230
Epoch 143/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0471 - accuracy: 0.548
8 - val_loss: 1.0948 - val_accuracy: 0.5220
Epoch 144/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0527 - accuracy: 0.549
3 - val_loss: 1.0945 - val_accuracy: 0.5210
Epoch 145/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0393 - accuracy: 0.546
5 - val_loss: 1.0935 - val_accuracy: 0.5190
Epoch 146/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0400 - accuracy: 0.541
0 - val_loss: 1.0928 - val_accuracy: 0.5190
Epoch 147/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0425 - accuracy: 0.550
0 - val_loss: 1.0927 - val_accuracy: 0.5240
Epoch 148/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0453 - accuracy: 0.540
0 - val_loss: 1.0930 - val_accuracy: 0.5170
Epoch 149/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0397 - accuracy: 0.551
0 - val_loss: 1.0928 - val_accuracy: 0.5250
Epoch 150/150
8/8 [==============================] - 0s 5ms/step - loss: 1.0407 - accuracy: 0.542
5 - val_loss: 1.0945 - val_accuracy: 0.5190
```

In [44]:

```
# Final evaluation of the model
loss, accuracy = model.evaluate(X_test, pd.get_dummies(y_test), verbose=1)
#print("Accuracy: ", accuracy*100, "%")

# plot training vs validation for overfiiting
plt.plot(fit.history['accuracy'], label='train accuracy')
plt.plot(fit.history['val_accuracy'], label='validation accuracy')
plt.plot(fit.history['loss'], label='train loss')
plt.plot(fit.history['val_loss'], label='validation loss')
plt.title('model_accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc=0, bbox_to_anchor=[1,1])
plt.show()
```

```
53/53 [==============================] - 0s 819us/step - loss: 1.0538 - accuracy:
0.5375
```


model_accuracy
accuracy
epoch
train accuracy
validation accuracy
train loss
validation loss
1.4
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
100
120
140

In [ ]: