# Predicting Customer Spending from Website, Mobile App, and In-Store Activity

June 15, 2020

Annalisa Donat, Data Scientist Connect with me on LinkedIn

**Predicting Customer Spending from Website, Mobile App, and In-Store Activity**

Table of Contents



## 1 Background

**Client:** An e-commerce company based in New York City that sells high-end clothing inspired by traditional textiles of Guam. Customers come in to the store, have (in-store) sessions/meetings with a personal stylist, then they can go home and order either on a mobile app or website for the clothes they want. The company only ships to areas served by the US Postal Service.

**Task:** Predict the amount a customer will spend each year based on website, mobile app, and in-store activity.

**Data:** - Avg. Session Length: Average session of in-store style advice sessions - Time on App: Average time spent on App in minutes - Time on Website: Average time spent on Website in minutes - Length of Membership: How many years the customer has been a member - Avatar: Color of the avatar the customer selected on the website - Email: Email address - Address: Shipping address

## 2 Load and Check Data

Import libraries:

[1]:
```
import numpy as np
import pandas as pd
import re
from scipy.stats import f_oneway, pearsonr, spearmanr
from sklearn.ensemble import AdaBoostRegressor
from sklearn.linear_model import LinearRegression, RANSACRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor

import matplotlib.pyplot as plt
import seaborn as sns
```

Load the data into a pandas DataFrame:

[17]:
```
df = pd.read_csv(
    ".../Ecommerce Customers.csv")
```

View the data:

[18]:
```
print("Dataset contains %d rows and %d columns" % df.shape)
```

```
Dataset contains 500 rows and 8 columns
```

[19]:
```
df.head()
```

[19]:
```
                          Email  \
0      mstephenson@fernandez.com
1              hduke@hotmail.com
2               pallen@yahoo.com
3        riverarebecca@gmail.com
4  mstephens@davidson-herman.com

                                             Address            Avatar  \
0      835 Frank Tunnel\nWrightmouth, MI 82180-9605            Violet
1    4547 Archer Common\nDiazchester, CA 06566-8576         DarkGreen
2  24645 Valerie Unions Suite 582\nCobbborough, D…            Bisque
3   1414 David Throughway\nPort Jason, OH 22070-1220       SaddleBrown
```

```
4  14023 Rodriguez Passage\nPort Jacobville, PR 3…  MediumAquaMarine

   Avg. Session Length  Time on App  Time on Website  Length of Membership  \
0            34.497268    12.655651        39.577668              4.082621
1            31.926272    11.109461        37.268959              2.664034
2            33.000915    11.330278        37.110597              4.104543
3            34.305557    13.717514        36.721283              3.120179
4            33.330673    12.795189        37.536653              4.446308

   Yearly Amount Spent
0           587.951054
1           392.204933
2           487.547505
3           581.852344
4           599.406092
```

Check datatype of each column:

[20]:
```
df.dtypes
```

[20]:
```
Email                    object
Address                  object
Avatar                   object
Avg. Session Length     float64
Time on App             float64
Time on Website         float64
Length of Membership    float64
Yearly Amount Spent     float64
dtype: object
```

Check for null values:

[21]:
```
df.isna().sum()
```

[21]:
```
Email                   0
Address                 0
Avatar                  0
Avg. Session Length     0
Time on App             0
Time on Website         0
Length of Membership    0
Yearly Amount Spent     0
dtype: int64
```

Number of unique values feature:

[22]:
```
df.nunique()
```

[22]:
```
Email                   500
Address                 500
Avatar                  138
Avg. Session Length     500
Time on App             500
Time on Website         500
Length of Membership    500
Yearly Amount Spent     500
dtype: int64
```

As a sanity check, use the .describe() function to make sure the data does not contain any obvious errors (ex. a negative values):

[30]:
```
df.describe().round(1)
```

[30]:
```
       Avg. Session Length  Time on App  Time on Website  \
count                500.0        500.0            500.0
mean                  33.1         12.1             37.1
std                    1.0          1.0              1.0
min                   29.5          8.5             33.9
25%                   32.3         11.4             36.3
50%                   33.1         12.0             37.1
75%                   33.7         12.8             37.7
max                   36.1         15.1             40.0

       Length of Membership  Yearly Amount Spent
count                 500.0                500.0
mean                    3.5                499.3
std                     1.0                 79.3
min                     0.3                256.7
25%                     2.9                445.0
50%                     3.5                498.9
75%                     4.1                549.3
max                     6.9                765.5
```

There are no obvious issues.

# 3 Data Transformation

## 3.1 Email

Email is a categorical variable with a unique value for each customer. Thus, it is impossible for Email to have any predictive power in its current form. However, natural language processing and text mining techniques might be useful for extracting meaning from the Email feature.

Potentially meaningful information that could be extracted from Email: - email service provider - simplicity of email address (proxy for age): - a combination of the following variables: - popularity of email service provider - when the email service provider was founded - length of email address (the part before the '@' sign) - whether email address contains any English words and how common those words are. - Very simple email addresses may indicate that the user is over a certain age: ex. there is a user in the dataset with the email address "pallen@yahoo . com". This user was likely old enough to use email when yahoo mail launched in 1997, as allen is a common first and last name and thus this username would be in high demand. - gender - use a library to detect female and male names in the email address. - ex. there is a user in the dataset with the email "riverarebecca@gmail . com". This is likely a woman named Rebecca Rivera. - education and place of employment: - work email addresses or email addresses ending in .edu may contain this information.

For the purpose of simplicity, the only information I will extract from Email will be the email service provider.

[23]:
```
df["email_serv"] = df["Email"].apply(lambda x: re.split("@", x)[1])
```

[24]:
```
df["email_serv"].nunique()
```

[24]:
```
244
```

[25]:
```
pd.DataFrame(df["email_serv"].value_counts()).head()
```

[25]:
```
             email_serv
gmail.com            87
hotmail.com          87
yahoo.com            76
jones.com             2
edwards.com           2
```

[26]:
```
df["email_type"] = df["email_serv"].apply(lambda x: "." + re.split("\.", x)[1])
```

[27]:
```
df["email_type"].unique()
```

[27]:
```
array(['.com', '.biz', '.net', '.info', '.org'], dtype=object)
```

[28]:
```
df["email_serv"] = df["email_serv"].apply(lambda x: re.split("\.", x)[0])
df["email_serv"] = df["email_serv"].apply(
    lambda x: x if x in ["gmail", "hotmail", "yahoo"] else "other")
```

[29]:
```
df.head()
```

[29]:
```
                          Email  \
0      mstephenson@fernandez.com
1              hduke@hotmail.com
2               pallen@yahoo.com
3        riverarebecca@gmail.com
4  mstephens@davidson-herman.com

                                             Address            Avatar  \
```

```
0      835 Frank Tunnel\nWrightmouth, MI 82180-9605           Violet
1     4547 Archer Common\nDiazchester, CA 06566-8576        DarkGreen
2  24645 Valerie Unions Suite 582\nCobbborough, D…           Bisque
3   1414 David Throughway\nPort Jason, OH 22070-1220      SaddleBrown
4  14023 Rodriguez Passage\nPort Jacobville, PR 3…  MediumAquaMarine

   Avg. Session Length  Time on App  Time on Website  Length of Membership  \
0            34.497268    12.655651        39.577668              4.082621
1            31.926272    11.109461        37.268959              2.664034
2            33.000915    11.330278        37.110597              4.104543
3            34.305557    13.717514        36.721283              3.120179
4            33.330673    12.795189        37.536653              4.446308

   Yearly Amount Spent email_serv email_type
0           587.951054      other       .com
1           392.204933    hotmail       .com
2           487.547505      yahoo       .com
3           581.852344      gmail       .com
4           599.406092      other       .com
```

## 3.2 Address

Just like Email, Address is a categorical variable with a unique value for each customer. Thus, it is impossible for Address to have any predictive power in its current form. However, natural language processing and text mining techniques might be useful for extracting meaning from the Address feature.

Extract state/territory/military location from address:

[31]:
```
military_addresses = ["%s %s" % (code, area)
                      for code in ["APO", "DPO", "FPO"]
                      for area in ["AA", "AE", "AP"]]
```

[32]:
```
def get_state(addr):
    states_found = re.findall(", [A-Z][A-Z] ", addr)
    if len(states_found) == 1:
        clean_state = states_found[0].replace(", ", "")[:-1]
        return clean_state
    else:
        for military_addr in military_addresses:
            if military_addr in addr:
                return military_addr
    return ""
```

[33]:
```
df["State"] = df["Address"].apply(lambda x: get_state(x))
```

Note that the following are abbreviations for US territories and Sovereign Nations served by the US Postal Service, not errors:

| Country/Territory | Abbreviation |
|---|---|
| American Samoa | AS |
| Guam | GU |
| Northern Mariana | MP |
| Marshall Islands | MH |
| Micronesia | FM |
| Palau | PL |
| Puerto Rico | PR |
| Virgin Islands | VI |

50 states + 1 District of Columbia + 9 military locations + 8 other countries/territories = 68 "states" in total.

Given that the client sells clothing inspired by the textiles of Guam, people from islands in Oceania and/or living on military bases may represent a disproportionately large number of customers.

[34]:
```
df["State"].nunique()
```

[34]:
```
68
```

[35]:
```
df["State"].value_counts()
```

[35]:
```
DE        13
MO        13
SC        13
OR        12
VT        12
          ..
WA         4
FPO AP     4
FPO AA     4
ID         3
APO AP     2
Name: State, Length: 68, dtype: int64
```

There are too many distinct values for State for State to be a meaningful feature. However, proximity to New York City may be significant.

[38]:
```
f_oneway(df[df["State"]=="NY"]["Yearly Amount Spent"],
         df[df["State"]!="NY"]["Yearly Amount Spent"])
```

[38]:
```
F_onewayResult(statistic=0.4962441746981331, pvalue=0.48148435280235435)
```

[44]:
```
f_oneway(df[df["State"].isin(["CT", "NJ"])]["Yearly Amount Spent"],
         df[~df["State"].isin(["CT", "NJ"])]["Yearly Amount Spent"])
```

[44]:
```
F_onewayResult(statistic=0.022549013496934 98, pvalue=0.8806965864769287)
```

Proximity to NYC is not significant, but whether a client resides in Guam may be a meaningful feature.

[46]:
```
f_oneway(df[df["State"]=="GU"]["Yearly Amount Spent"],
         df[df["State"]!="GU"]["Yearly Amount Spent"])
```

[46]:
```
F_onewayResult(statistic=6.994277104849975, pvalue=0.008435256826885338)
```

[51]:
```
print(df[df["State"]=="GU"]["Yearly Amount Spent"].mean())
print(df[df["State"]!="GU"]["Yearly Amount Spent"].mean())
```

```
414.7013601333333
500.3417226105265
```

[47]:
```
df[df["State"]=="GU"].shape
```

[47]:
```
(6, 11)
```

Individuals from Guam did spend less. I will reexamine this difference later in the analysis, when I will have a better understanding of potential third variables that could explain this trend.

Group State by region.

[74]:
```
region_states = {
    "northeast": ["CT", "ME", "MA", "NH", "NJ", "NY", "PA", "RI", "VT",
                  "DE", "MD", "DC"],
    "south": ["AL", "AR", "FL", "GA", "KY", "LA", "MD", "MS", "NC", "SC", "TN",
             "VA", "WV", "TX", "OK"],
    "west": ["WA", "MT", "OR", "ID", "WY", "CA", "NV", "UT", "CO", "AK",
              "HI", "AZ", "NM"],
    "midwest": ["WI", "IL", "ND", "SD", "NE", "KS", "MI", "IN", "MN",
                 "IA", "MO", "OH"],
    "territories": ["AS", "GU", "MP", "FM", "PR", "VI", "MH", "PW"],
    "military": military_addresses
}
```

[75]:
```
state_region = {state: region for region in
                region_states.keys() for state in region_states[region]}
```

[76]:
```
df["Region"] = df["State"].map(state_region)
```

[ ]:
```
df = df.drop(columns=["Address"])
```

### 3.3 Avatar

[63]:
```
df["Avatar"].value_counts()
```

[63]:
```
CadetBlue         7
SlateBlue         7
GreenYellow       7
Cyan              7
Teal              7
                 ..
PaleGreen         1
PaleTurquoise     1
Yellow            1
DeepSkyBlue       1
LightSlateGray    1
Name: Avatar, Length: 138, dtype: int64
```

Just like Email and Address, Avatar is a categorical variable with a unique value for each customer. Thus, it is impossible for Avatar to have any predictive power in its current form. However, given that there is no objective way to group these colors into an overarching hierarchy (ex. should pastel blue be grouped with blues or pastels), I am just going to drop the Avatar column from the dataset.

While Avatar is not useful for this analysis, I would recommend that the company investigate whether there is a relationship between Avatar color and the color of future clothing purchases. If so, avatar color could indicate which colors will be in demand next fashion season.

[64]:
```
df = df.drop(columns=["Avatar"])
```

[65]:
```
df.head()
```

[65]:
```
   Avg. Session Length  Time on App  Time on Website  Length of Membership  \
0            34.497268    12.655651        39.577668              4.082621
1            31.926272    11.109461        37.268959              2.664034
2            33.000915    11.330278        37.110597              4.104543
3            34.305557    13.717514        36.721283              3.120179
4            33.330673    12.795189        37.536653              4.446308

   Yearly Amount Spent email_serv email_type State       Region
0           587.951054      other       .com    MI      midwest
1           392.204933    hotmail       .com    CA         west
2           487.547505      yahoo       .com    DC    northeast
3           581.852344      gmail       .com    OH      midwest
4           599.406092      other       .com    PR  territories
```

# 4 Explore Dataset

## 4.1 Create Training and Test Sets

Creating a test set allows us to evaluate the variance of our model. Variance is the extent to which patterns detected in the training data can be generalized to unseen data (such as the test set).

[78]:
```
train_df, test_df = train_test_split(
    df, test_size=0.20, random_state=1)
```

[79]:
```
continuous_predictors = ["Avg. Session Length", "Time on App",
                         "Time on Website", "Length of Membership"]
categorical_predictors = ["email_serv", "email_type", "Region"]
```

[80]:
```
train_X = train_df[continuous_predictors].values
train_y = train_df["Yearly Amount Spent"].values
test_X = test_df[continuous_predictors].values
test_y = test_df["Yearly Amount Spent"].values
```

### 4.2 Define Visualization Functions

[69]:
```
def calculate_pvalues(dfr, method_func=pearsonr):
    """
    dfr : dataframe
    method_func : function (NOT A STRING)
        - either pearsonr or spearmanr from scipy.stats
    """
    dfcols = pd.DataFrame(columns=dfr.columns)
    pvalues = dfcols.transpose().join(dfcols, how='outer')
    for r in dfr.columns:
        for c in dfr.columns:
            pvalues[r][c] = round(method_func(dfr[r], dfr[c])[1], 2)
    return pvalues
```

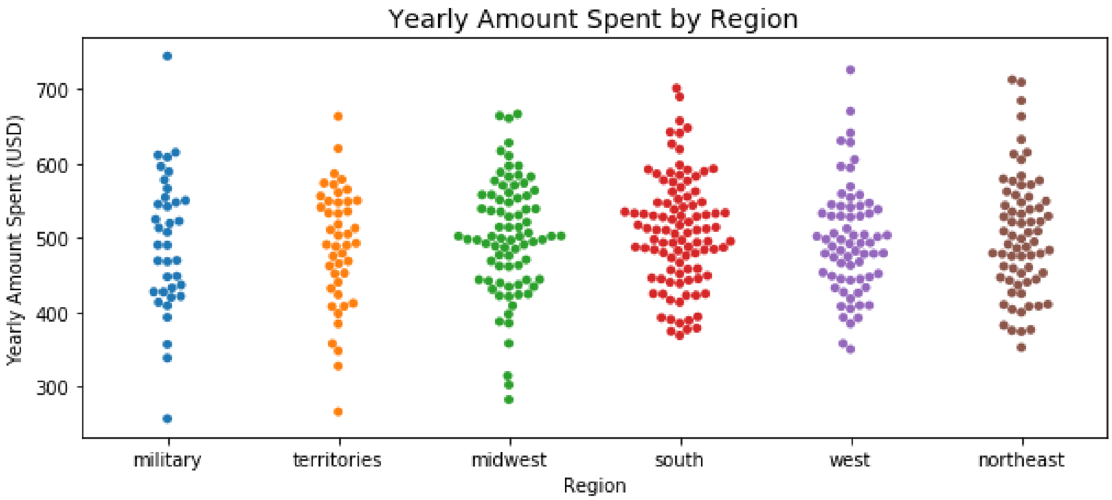
[70]:
```
def get_p_val_mask(dfr, method_func=pearsonr, p_val_cutoff=0.1):
    """
    dfr : dataframe
    method_func : function (NOT A STRING)
        - either pearsonr or spearmanr from scipy.stats
    p_val_cutoff : float
        - 0 < p_val_cutoff < 1
    """
    p_vals_mtrx = calculate_pvalues(dfr, method_func=method_func)
    for c in list(p_vals_mtrx):
        p_vals_mtrx[c] = p_vals_mtrx[c].apply(
            lambda x: 0 if x >= 0.1 else 1)
    return p_vals_mtrx
```

### 4.3 Explore Training Data

#### 4.3.1 Categorical Data

**Region**

[85]:
```
fig, my_ax = plt.subplots(figsize=(10, 4))
sns.swarmplot(x="Region", y="Yearly Amount Spent", data=train_df, ax=my_ax)
my_ax.set_title("Yearly Amount Spent by Region", fontsize=14)
my_ax.set_ylabel("Yearly Amount Spent (USD)")

plt.show()
```




Conduct an ANOVA test to see if there is a significant difference in the Yearly Amount Spent in each Region:

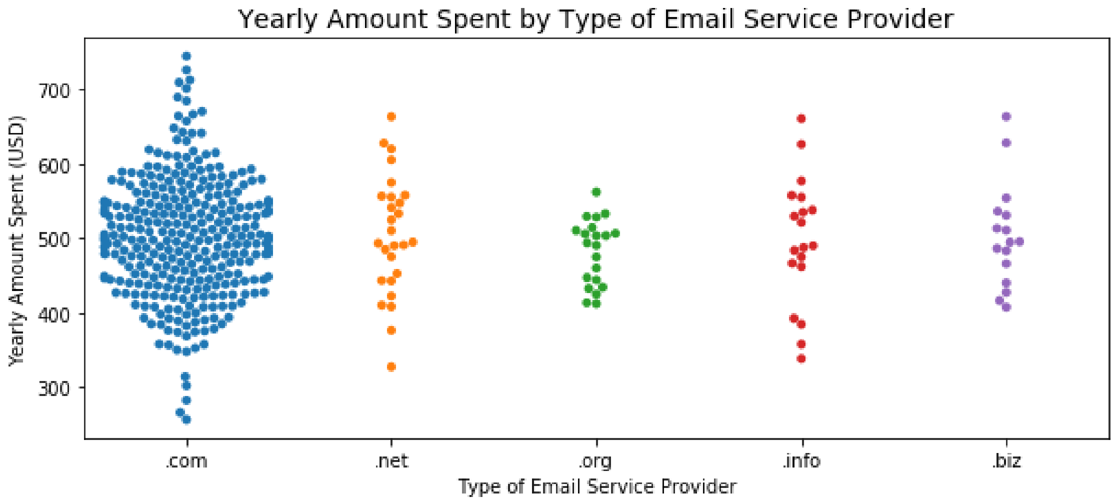
[86]:
```
yas_regions = []
for area in train_df["Region"].unique():
    yas_regions.append(train_df[train_df["Region"]==area].copy()["Yearly Amount␣
↪Spent"])
f_oneway(*yas_regions)
```

[86]:
```
F_onewayResult(statistic=0.5507746302861277, pvalue=0.737737378414197)
```

p-value = 0.74 > 0.1, meaning that Region is not a significant predictor of Yearly Amount Spent.

**Type of Email Service Provider**

[88]:
```
fig, my_ax = plt.subplots(figsize=(10, 4))
sns.swarmplot(x="email_type", y="Yearly Amount Spent", data=train_df, ax=my_ax)
my_ax.set_title("Yearly Amount Spent by Type of Email Service Provider",
                fontsize=14)
my_ax.set_ylabel("Yearly Amount Spent (USD)")
my_ax.set_xlabel("Type of Email Service Provider")
plt.show()
```




Conduct an ANOVA test to see if there is a significant difference in the Yearly Amount Spent for each email_type:

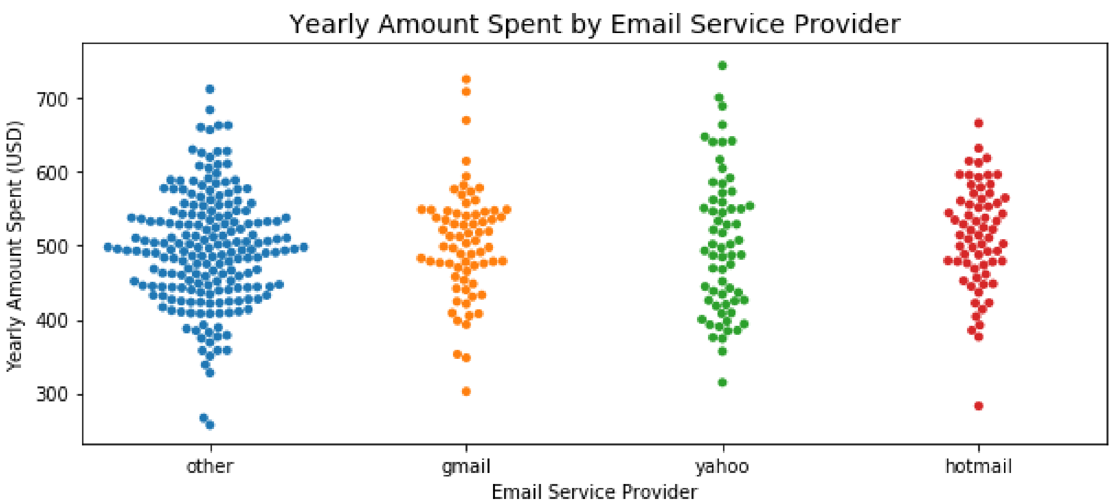
[89]:
```
yas_email_types = []
for email_sp_type in train_df["email_type"].unique():
    yas_email_types.append(train_df[train_df["email_type"]==email_sp_type
                                   ].copy()["Yearly Amount Spent"])
f_oneway(*yas_email_types)
```

[89]:
```
F_onewayResult(statistic=0.34281407462547203, pvalue=0.8489876216512878)
```

p-value = 0.74 > 0.1, meaning that email_type is not a significant predictor of Yearly Amount Spent.

**Email Service Provider**

[90]:
```
fig, my_ax = plt.subplots(figsize=(10, 4))
sns.swarmplot(x="email_serv", y="Yearly Amount Spent", data=train_df, ax=my_ax)
my_ax.set_title("Yearly Amount Spent by Email Service Provider",
                fontsize=14)
my_ax.set_ylabel("Yearly Amount Spent (USD)")
my_ax.set_xlabel("Email Service Provider")
plt.show()
```




Conduct an ANOVA test to see if there is a significant difference in the Yearly Amount Spent for each email_serv:

[91]:
```
yas_email_servs = []
for email_service_provider in train_df["email_serv"].unique():
    yas_email_servs.
 ↪append(train_df[train_df["email_serv"]==email_service_provider
                                        ].copy()["Yearly Amount Spent"])
f_oneway(*yas_email_servs)
```

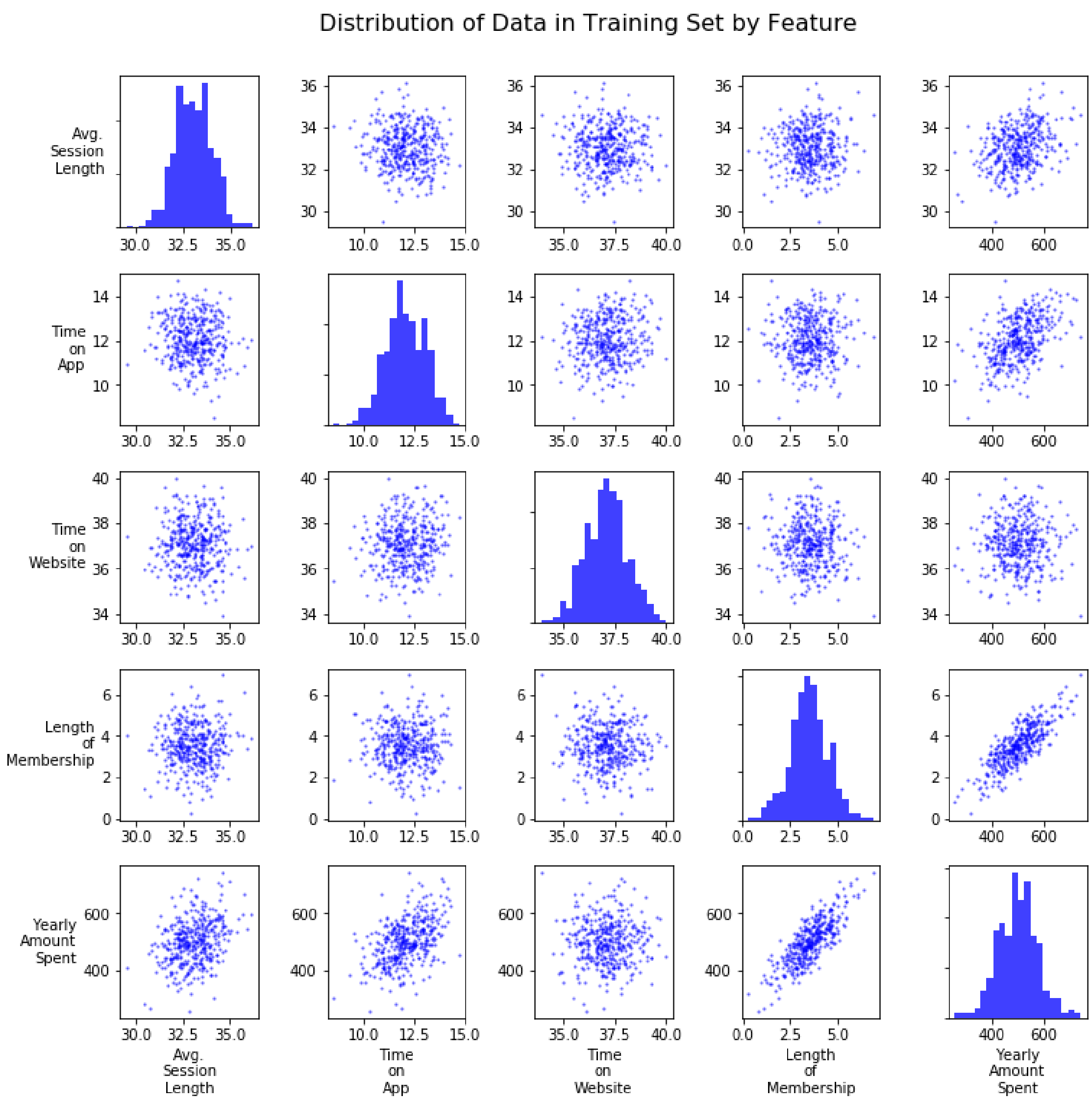
[91]:
```
F_onewayResult(statistic=1.0082937956670057, pvalue=0.38898311660136486)
```

p-value = 0.39 > 0.1, meaning that email_serv is not a significant predictor of Yearly Amount Spent.

### 4.3.2 Numerical Data

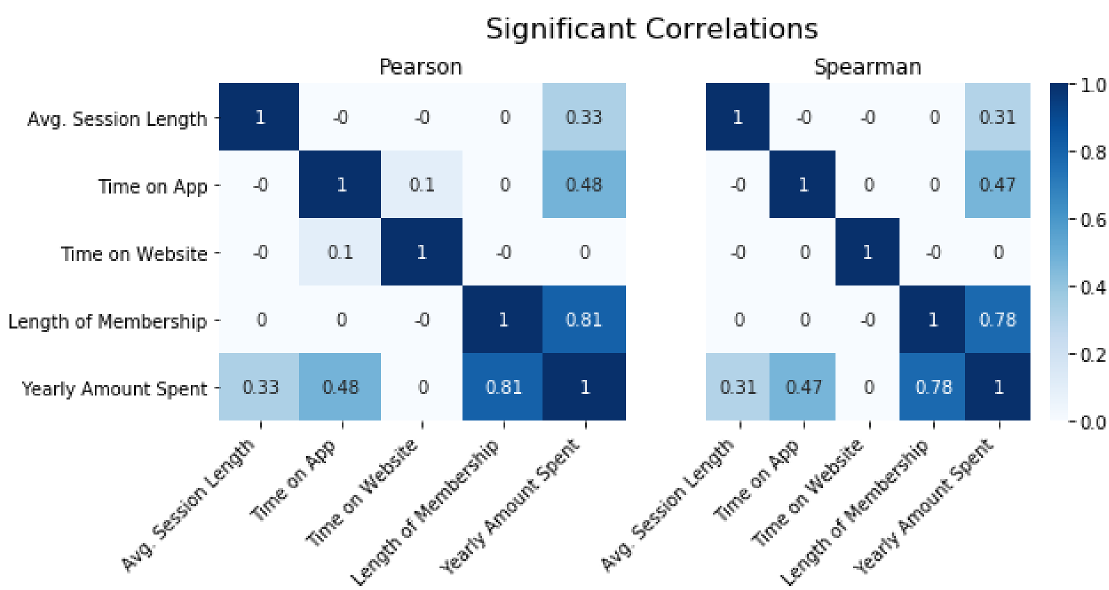
[92]:
```
continuous_vars = continuous_predictors + ["Yearly Amount Spent"]

fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(12, 12))

# plot data:
for j in range(5):
    for i in range(5):
        if i==j:
            axes[j][i].hist(x=train_df[continuous_vars[i]], bins=20,
                            color="blue", alpha=0.75)
            axes[j][i].set_yticklabels([])
        else:
            axes[j][i].scatter(x=train_df[continuous_vars[i]],
```

```
                              y=train_df[continuous_vars[j]],
                              color="blue", s=2, alpha=0.4)

# formatting: label x-axis of bottom row and y-axis of left column
for k in range(5):
    label_text = continuous_vars[k].replace(" ", "\n")
    axes[4][k].set_xlabel(label_text)
    axes[k][0].set_ylabel(label_text, rotation=0, ha="right", va="center")

fig.suptitle("Distribution of Data in Training Set by Feature",
             fontsize=16, y=.93)

plt.subplots_adjust(wspace=.5, hspace=.3)
plt.show()
```



All variables have a normal distribution.

[93]:
```
fig, axes = plt.subplots(figsize=(9, 3.5), ncols=2)
sns.heatmap(train_df[continuous_vars].corr(
    method="pearson").round(2)*get_p_val_mask(train_df[continuous_vars],␣
 ↪pearsonr),
            cmap="Blues", ax=axes[0], center=.5, annot=True, cbar=False)
axes[0].set_title("Pearson")

sns.heatmap(train_df[continuous_vars].corr(
    method="spearman").round(2)*get_p_val_mask(train_df[continuous_vars],␣
 ↪spearmanr),
            cmap="Blues", ax=axes[1], center=.5, annot=True)
axes[1].set_title("Spearman")

axes[1].set_yticklabels([])
axes[1].set_yticks([])


for my_ax in axes:
    bottom, top = my_ax.get_ylim()
    my_ax.set_ylim(bottom + 0.5, top - 0.5)
    my_ax.set_xticklabels(my_ax.get_xticklabels(), rotation=45, ha="right")

fig.suptitle("Significant Correlations", y=1.03, fontsize=16)

plt.show()
```



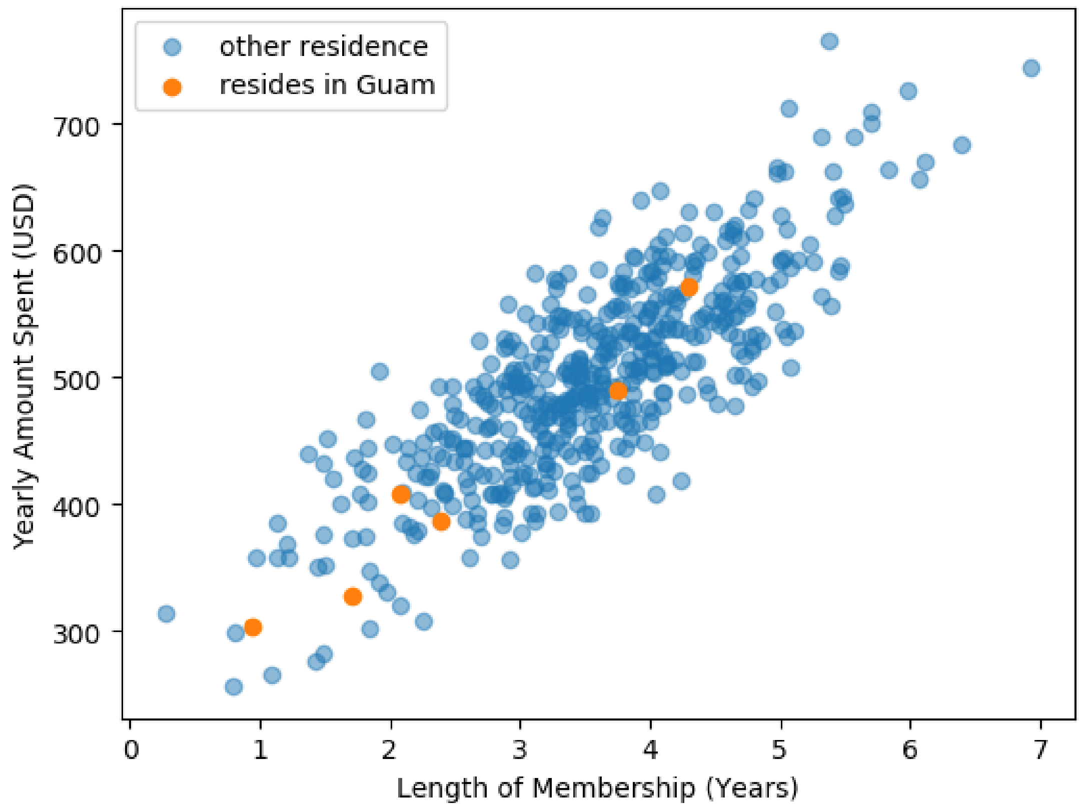
**Observations:** Avg. Session Length, Time on App, and Length of Membership all have a positive linear relationship with Yearly Amount Spent and are not correlated with each other. Length of Membership is the greatest predictor of Yearly Amount Spent, followed by Time on App, and Avg. Session Length. There is no relationship between Time on Website and Yearly Amount Spent.

**Implications for Model:** As the three predictor variables have no relationship with each other, a Principal Components Analysis is not needed. Furthermore, a Multivariate Linear Regression (perhaps with RANSAC) will likely fit the data well. Given the lack of complex relationships in the dataset, more complex methods, such as a Decision Tree Regression or AdaBoost Regression, will not suit the data as well.

### Effect of Residence in Guam, Revisited

As previously discussed, customers living in Guam have lower values for Yearly Amount Spent. A quick scatter plot of Length of Membership (the strongest predictor of Yearly Amount Spent) and Yearly Amount Spent will show whether a l

[132]:
```
fig, ax = plt.subplots()
ax.scatter("Length of Membership", "Yearly Amount Spent",
            data=df[df["State"]!="GU"], alpha=.5, label="other residence")
ax.scatter("Length of Membership", "Yearly Amount Spent",
            data=df[df["State"]=="GU"], label="resides in Guam")
ax.set_ylabel("Yearly Amount Spent (USD)")
ax.set_xlabel("Length of Membership (Years)")

ax.legend()
plt.show()
```




Given that only 6 customers reside in Guam and that their lower Yearly Amount Spent values are proportional to their lower Length of Membership values, I will not include whether a customer resides in Guam from the model.

# 5 Predict Yearly Amount Spent

[94]:
```
significant_predictors = ["Avg. Session Length",
                          "Time on App",
                          "Length of Membership"]
```

[95]:
```
train_X = train_df[significant_predictors].values
test_X = test_df[significant_predictors].values
```

## 5.1 Define Visualization Functions

[97]:
```
color_explanation = "Darker purple dots correspond to higher values for "\
"the sum of the standard deviations of the other two predictor variables."
```

[98]:
```
sc = StandardScaler()
```

[99]:
```
def plot_model_and_residuals(model, thin_trendline=False, model_name=None):
    plt.rcdefaults()
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(12, 8))

    for i in range(3):
        model = model.fit(train_X[:, i].reshape(-1, 1), train_y)
        x_span = train_X[:, i].max()-train_X[:, i].min()
        x_all = np.linspace(train_X[:, i].min()-int(x_span/6),
                            train_X[:, i].max()+int(x_span/6), 100)
        y_all_pred = model.predict(x_all.reshape(-1, 1))
        other_X_cols = [j for j in range(3) if j!=i]
        ja_std = sc.fit_transform(train_X[:, other_X_cols[0]].reshape(-1, 1))
        jb_std = sc.fit_transform(train_X[:, other_X_cols[1]].reshape(-1, 1))
        hue_col = ja_std + jb_std
        hue_col = hue_col.reshape(train_X[:, i].shape)
        sns.scatterplot(train_X[:, i], train_y,
                        hue=hue_col,
                        alpha=.7, ax=axes[0][i])
        if thin_trendline:
            axes[0][i].plot(x_all, y_all_pred, color="black",
                            linewidth=1, alpha=0.6)
        else:
            axes[0][i].plot(x_all, y_all_pred, color="black")
        axes[0][i].set_title(significant_predictors[i])
        axes[0][i].set_xlabel(significant_predictors[i])
        axes[0][i].set_ylabel("Yearly Amount Spent")
        axes[0][i].legend_.remove()

        pred_y = model.predict(train_X[:, i].reshape(-1, 1))
        #print(train_y.mean())
        #print(pred_y.mean())
        sns.scatterplot(train_y, train_y-pred_y,
                        hue=hue_col,
                        ax=axes[1][i], alpha=0.6)
        axes[1][i].axhline(color="black", linewidth=1)
        axes[1][i].set_xlabel("Yearly Amount Spent")
        axes[1][i].set_ylabel("Residuals")
        axes[1][i].set_ylim(-275, 275)
        axes[1][i].set_title(significant_predictors[i])
        axes[1][i].legend_.remove()

    if model_name:
        fig.suptitle(model_name + ": Univariate Fit and Residuals",
                     fontsize=14)
    else:
        fig.suptitle("Univariate Fit and Residuals", fontsize=14)
```

```python
    plt.figtext(0.5, 0.01, color_explanation, horizontalalignment='center')
    plt.subplots_adjust(hspace=.4, wspace=.4)
    plt.show()
```

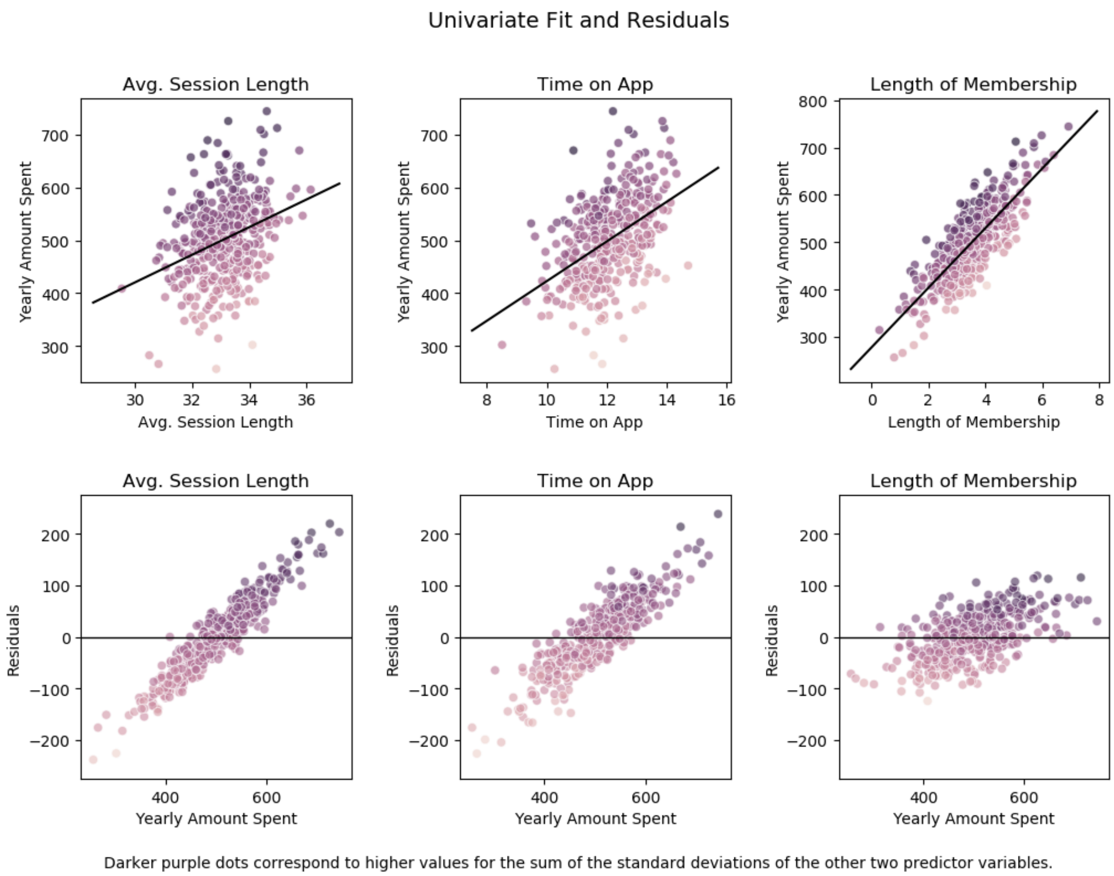
[100]:
```python
def evaluate_model(model):
    model = model.fit(train_X, train_y)
    print("Training Set Score: ",
          model.score(train_X, train_y).round(3))
    print("Test Set Score:     ",
          model.score(test_X, test_y).round(3))
```

[101]:
```python
def summarize_gs(gs):
    """Prints best test (validation) score of a fitted GridSearchCV object
    Parameters
    ------------
    gs : fitted GridSearchCV object

    Returns
    -------
    None
    """
    if len(list(gs.best_params_.keys())) > 0:
        print("Best Parameters:", gs.best_params_)
    gs_std_test_score = gs.cv_results_["std_test_score"][gs.best_index_]
    print("Best Score: %.3f +/- %.3f" % (gs.best_score_, gs_std_test_score))
```

## 5.2 Linear Regression

[102]:
```python
lnr = LinearRegression(normalize=True)
```

[103]:
```python
pipe_lnr = Pipeline([("sc", StandardScaler()), ("lnr", lnr)])
plot_model_and_residuals(pipe_lnr)
```




```
[104]: gs_lnr = GridSearchCV(
           estimator=pipe_lnr, param_grid={},
           scoring="r2", return_train_score=True, cv=5, n_jobs=-1, iid=True)
       gs_lnr = gs_lnr.fit(train_X, train_y)
       summarize_gs(gs_lnr)
```

```
Best Score: 0.982 +/- 0.005
```

```
[105]: evaluate_model(pipe_lnr)
```

```
Training Set Score:  0.983
Test Set Score:      0.989
```

Wow! Just as I predicted, the multivariate linear regression model fits the data very well.

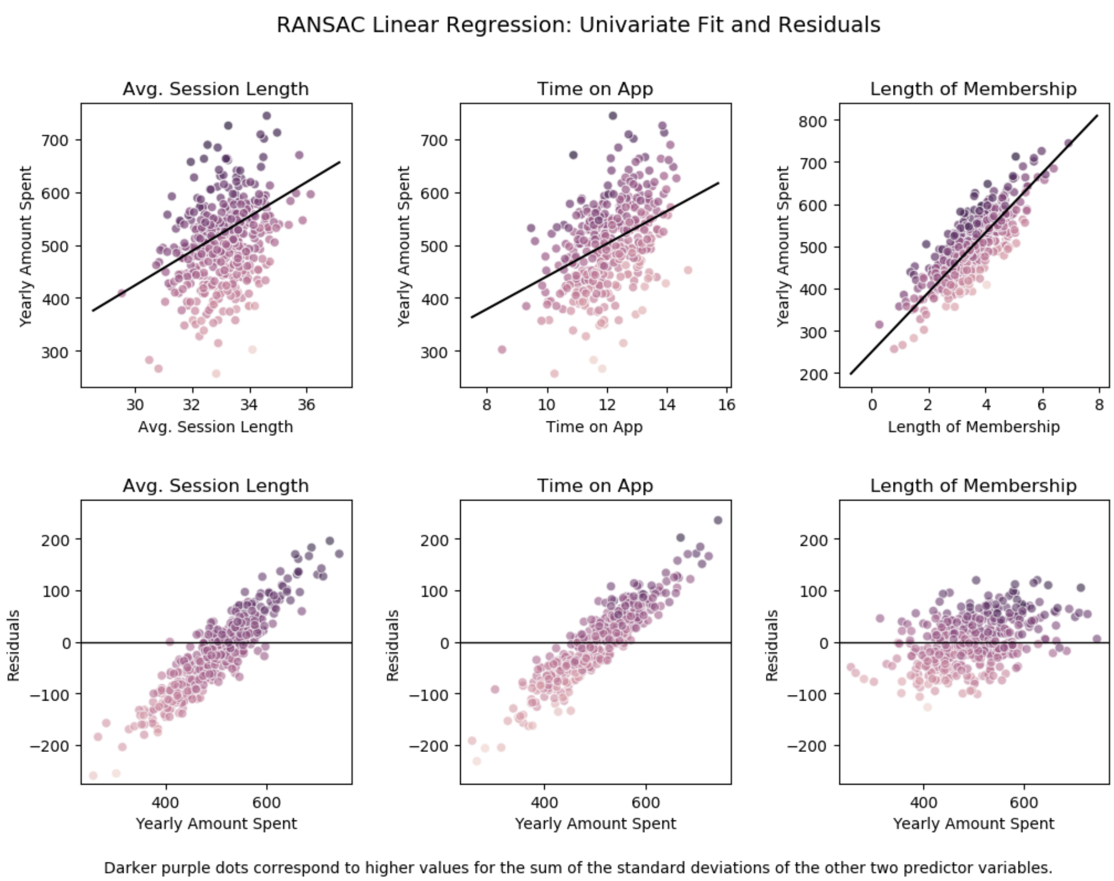
### 5.3 RANSAC Linear Regression

RANSAC (Random Sample Consensus) reduces the influence of outliers in a linear regression. While there is little room for improvement, I will try applying RANSAC to the multivariate linear regression model.

[107]:
```
ransac = RANSACRegressor(
    LinearRegression(), max_trials=100, min_samples=50,
    loss='absolute_loss', residual_threshold=3.0, random_state=1)
```

[111]:
```
ransac = RANSACRegressor(
    LinearRegression(), max_trials=100,
    loss='absolute_loss', random_state=1)
```

[112]:
```
pipe_lnr_ransac = Pipeline([("sc", StandardScaler()), ("ransac", ransac)])
plot_model_and_residuals(pipe_lnr_ransac,
                         model_name="RANSAC Linear Regression")
```




Fit multivariate RANSAC linear regression:

[113]:
```
gs_lnr_ransac = GridSearchCV(
    estimator=pipe_lnr_ransac,
    param_grid={"ransac__min_samples": [50, 200, 300, 310],
                "ransac__residual_threshold": [1, 2, 3, 4, 5, 6]
               },
    scoring="r2", return_train_score=True, cv=5,
```

```
    n_jobs=-1, iid=True)

gs_lnr_ransac = gs_lnr_ransac.fit(train_X, train_y)
summarize_gs(gs_lnr_ransac)
```

```
Best Parameters: {'ransac__min_samples': 300, 'ransac__residual_threshold': 3}
Best Score: 0.982 +/- 0.005
```

Further fine tune the hyperparameters:

[116]:
```
gs_lnr_ransac = GridSearchCV(
    estimator=pipe_lnr_ransac,
    param_grid={"ransac__min_samples": [250, 300, 310],
                "ransac__residual_threshold": [2.5, 3, 3.5]
               },
    scoring="r2", return_train_score=True, cv=5,
    n_jobs=-1, iid=True)

gs_lnr_ransac = gs_lnr_ransac.fit(train_X, train_y)
summarize_gs(gs_lnr_ransac)
```

```
Best Parameters: {'ransac__min_samples': 300, 'ransac__residual_threshold': 2.5}
Best Score: 0.982 +/- 0.005
```

[117]:
```
pipe_lnr_ransac = pipe_lnr_ransac.set_params(**gs_lnr_ransac.best_params_)
```

[118]:
```
evaluate_model(pipe_lnr_ransac)
```

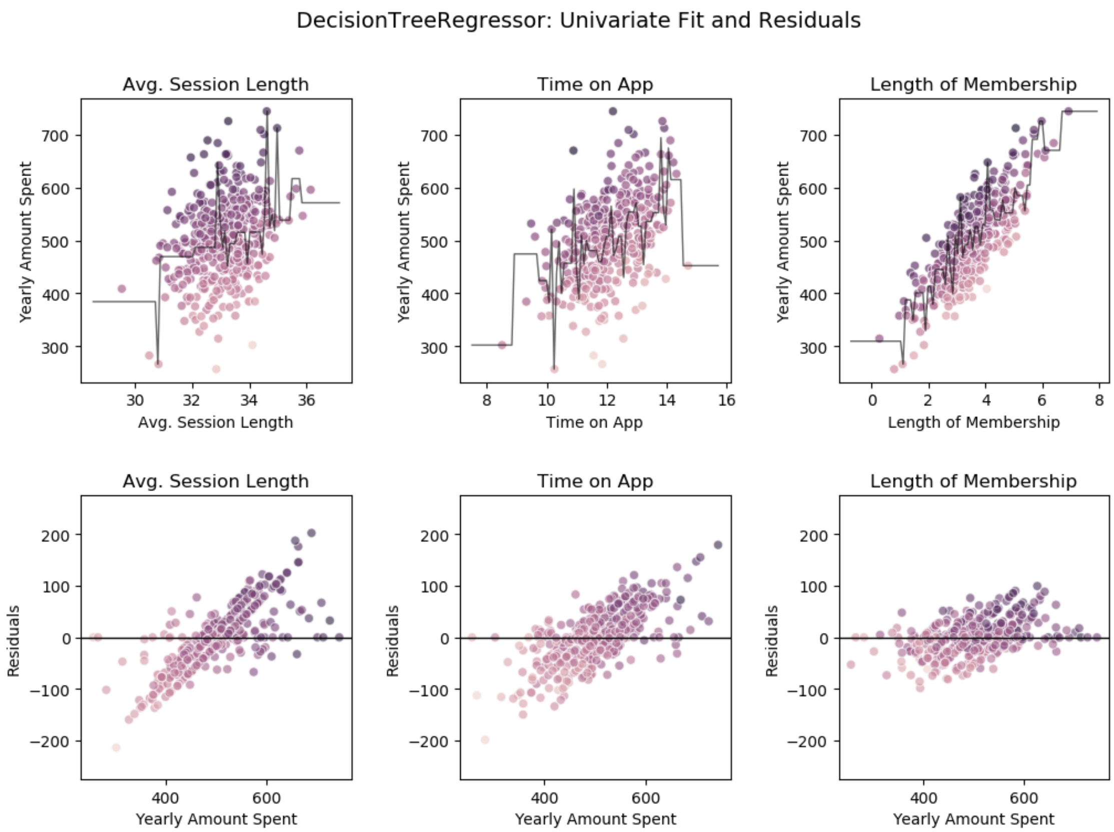
```
Training Set Score:  0.983
Test Set Score:      0.99
```

The r2 score for the test set improved by 0.001, but there was very little room for improvement.

### 5.4 Decision Tree Regressor

As discussed earlier, I do not believe that a Decision Tree Regression will fit the dataset well.

[119]:
```
dt_reg = DecisionTreeRegressor(max_depth=8, min_samples_split=4)
plot_model_and_residuals(dt_reg, thin_trendline=True,
                         model_name="DecisionTreeRegressor")
```



Darker purple dots correspond to higher values for the sum of the standard deviations of the other two predictor variables.

[120]:
```
dtr_params = {"max_depth": [3, 6, 8, 10, 15, 50],
              "min_samples_split": [2, 5, 20, 30, 50]}
```

[121]:
```
gs_dtr = GridSearchCV(
    estimator=DecisionTreeRegressor(random_state=1),
    param_grid=dtr_params,
    scoring="r2", return_train_score=True, cv=5,
    n_jobs=-1, iid=True)
gs_dtr = gs_dtr.fit(train_X, train_y)
summarize_gs(gs_dtr)
```

```
Best Parameters: {'max_depth': 10, 'min_samples_split': 2}
Best Score: 0.833 +/- 0.016
```

[122]:
```
dtr = DecisionTreeRegressor(
    random_state=1).set_params(**gs_dtr.best_params_)
```

[123]:
```
evaluate_model(dtr)
```

```
Training Set Score:  0.998
```

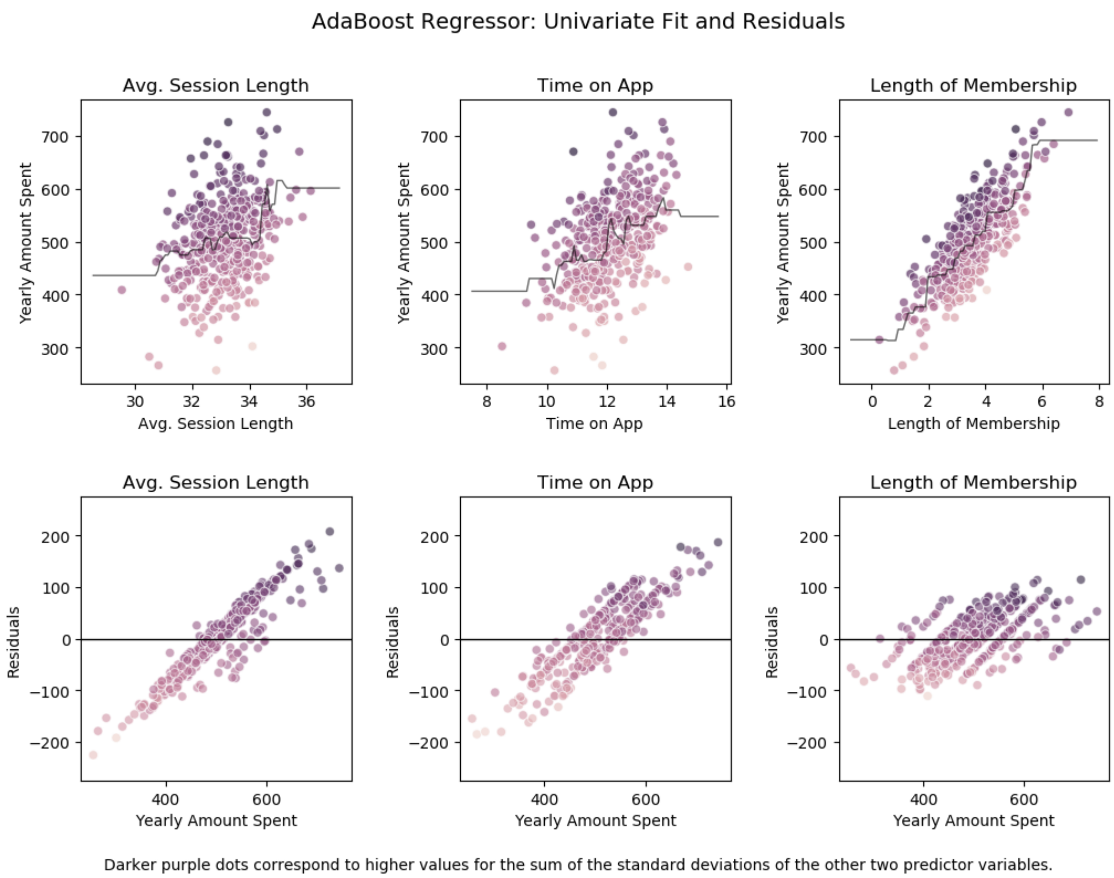
```
Test Set Score:      0.914
```

Indeed, the Decision Tree Regression is not as good a model as the Multivariate Linear Regression Model.

### 5.5 ADABoost Regression

As discussed earlier, I do not believe that an AdaBoost Regression will fit the dataset well. As the Decision Tree Regression didn't fit the dataset nearly as well as did the Multivariate Linear Regression, I am quite confident that the AdaBoost Regression will not fit the data as well as the Multivariate Linear Regression.

[124]:
```python
adabr = AdaBoostRegressor(random_state=1)
plot_model_and_residuals(adabr, thin_trendline=True,
                         model_name="AdaBoost Regressor")
```




[125]:
```python
evaluate_model(adabr)
```

```
Training Set Score:  0.918
Test Set Score:      0.861
```

Indeed, the Decision Tree Regression is not as good a model as the Multivariate Linear Regression Model.

## 6 Conclusion

Avg. Session Length, Time on App, and Length of Membership all have a positive linear relationship with Yearly Amount Spent and are not correlated with each other. Length of Membership is the greatest predictor of Yearly Amount Spent, followed by Time on App, and Avg. Session Length. As the three predictor variables have no relationship with each other, the Multivariate Linear Regression with RANSAC fit the data nearly perfectly. The r2 score for the test data was 0.990.