Randy Garcia
rgarci10@syr.edu
SUID: 342851174
IST 782 MS ADS Portfolio

School of Information Studies
SYRACUSE UNIVERSITY

Table of Contents

Randy Garcia

Data Scientist | Financial Analytics | Business Intelligence

Bronx, NY 10451 | 917-843-5896 | garciarandy314@gmail.com

LinkedIn: www.linkedin.com/in/randy-garcia-b0b0329a | Portfolio: https://github.com/randygar314/Portfolio.git

PROFESSIONAL SUMMARY

Data Scientist with 9+ years of financial analytics experience and an MSc in Applied Data Science. Proven track record of leveraging statistical modeling and advanced analytics to drive strategic decision-making for multi-billion-dollar business units. Expert in translating complex financial data into actionable insights for executive leadership and regulatory reporting.

TECHNICAL SKILLS

Programming Languages: Python, R, SQL
Data Analysis & Visualization: Tableau, Power BI, Alteryx, Excel Advanced Analytics
Databases & Systems: Oracle, Hyperion ESSBASE, Strategic Solar (S2), Pearl, FullSuite, SAP
Statistical Methods: Predictive Modeling, Variance Analysis, Time Series Forecasting, Business Intelligence
Financial Modeling: CCAR Stress Testing, Balance Sheet Analytics, Risk Management, Financial Planning & Analysis, Financial KPIs

EDUCATION

Master of Science, Applied Data Science | *Business Analytics Track* — ***Jul 2024***
Syracuse University, Syracuse, NY

Bachelor of Business Administration, Finance | *Computer Information Systems Minor* — ***Dec 2015***
City University of New York - Lehman College, Bronx, NY | GPA: 3.7/4.0

PROFESSIONAL EXPERIENCE

Data Analytics Leader | Citigroup Inc. - US Personal Banking Deposits (USPB), New York, NY — ***Sep 2019 – Present***
Vice President

- **Strategic Analytics & Forecasting:** Lead data-driven three-year strategic planning and monthly forecasting processes, utilizing statistical models to predict deposit balances and Net Interest Income improving accuracy and data quality
- **Executive Reporting & Data Visualization:** Develop automated dashboards and analytics reports for senior management using advanced data visualization techniques, reducing report generation time by 60%
- **Variance Analysis & Predictive Modeling:** Perform variance analysis against annual and quarterly forecasts, implementing machine learning algorithms to identify key drivers and predict future performance trends
- **Regulatory Analytics:** Maintain analytical frameworks for regulatory compliance reporting, ensuring data accuracy and timeliness for external auditor reviews

Financial Data Analyst | Citigroup Inc. - Institutional Clients Group, New York, NY — ***Aug 2018 – Sep 2019***
Balance Sheet Consolidation FP&A Analyst

- **Large-Scale Data Analytics:** Processed and analyzed multi-terabyte datasets to create comprehensive Balance Sheet Analytics across business segments, identifying variance drivers through statistical analysis
- **Predictive Modeling for Risk Management:** Maintained and upgraded forecasting models for CCAR stress testing scenarios, enabling senior management to make data-driven strategic decisions under various economic conditions
- **Automated Reporting Solutions:** Developed automated variance analysis templates and reporting systems, reducing manual processing time by 40% while improving accuracy and consistency

Financial Data Systems Analyst | Citigroup Inc. - Treasury and Trade Solutions, New York, NY — ***Jul 2016 – Aug 2018***
Global Controller Analyst

- **Data Engineering & ETL:** Implemented data retrieval and structuring processes from multiple financial reporting systems, ensuring data quality and integrity for month-end reconciliations
- **Anomaly Detection:** Tracked daily deposit balances and identify out-of-trend movements, implementing early warning detection for stakeholder escalation
- **Cross-Functional Data Projects:** Led data integration initiatives including Data Bridge automation project, coordinating between Technology and Finance teams to streamline reporting processes and improve data accessibility
- **Global Data Management:** Managed data collection and analysis for 100+ business units and tens of thousands for clients worldwide, ensuring compliance with intercompany revenue allocation and reporting requirements

KEY PROJECTS & ACHIEVEMENTS

Financial Forecasting Model Enhancement • Improved forecast accuracy by 25% through implementation of advanced time series analysis while working with model convergence team while also reducing quarterly reporting cycle time by 35% through development of data pipelines and validation processes

Risk Analytics Dashboard • Created executive-level risk analytics dashboards providing real-time insights into balance sheet movements and variance drivers

PROFESSIONAL DEVELOPMENT

The Colorwave Fellowship | *Cohort 3*
10-week intensive program focused on innovation economy and Silicon Valley employment opportunities | *Sep 2021 – Nov 2021*

KIPP & MLT Prep Corporate Leadership Coaching
Selected from 2,500+ national applicants for professional leadership development program | *Dec 2019 – Aug 2020*

Project Overview

I am currently a Financial Planning and Analysis Lead Analyst at a major bank. I work with substantial amounts of financial data related to consumer deposits, varying from deposit product balances, sales numbers, transactions, number of households, and other financial statement related information. My key work duties include accessing different data source systems in order to put together historical and forward looking analysis reports, presentations with graphs and charts, and many ad hoc analysis requested by various stakeholders that require me to take data and put it in a readable and organized form to provide insight and aid in management decision making. Although most data sources provide structured data, I often times need to combine data from various databases and data warehouses. Depending on the business and management needs, I have to be able to pull data then manipulate it into usable reports that can be used to create a story of bank's financials. I often pull data directly management reporting systems and then then disseminate them to various audiences.

When I enrolled in the Applied Data Science master's program, my goal was to acquire new technical skills and become more marketable as an employee, advance my career, become a more valuable and productive business asset, and gain skills that I can apply in my professional everyday life.

Throughout my time in the Applied Data Science master's program, I have been able to enhance both my technical skills and my people skills. Below is a list of skills that I have been able to acquire and refine during the program:

- Develop stronger presentation and communication skills conveying data and analyses to various individuals; like senior executives, risk managers, technology specialists, while being able to discern what level of information they need to receive.
- Learn best practices in data science which helped me organize my ideas and programming to create effective and easy to understand models and analyses that can be shared with peers and colleagues.
- Gather and structure data effectively.
- Recognize patterns in data through visualization, statistical analysis, and data mining.
- Formulate alternative strategies based on the analyzed data.

Demonstrate data collection and structuring effectively

Conceptual works that demonstrates data collection

- ☑ R
  - Using R programming I looked at various processed and cleaned data for analytical use:
    - Reading data from a csv file into a data-frame
    - Summarize the ingested data.
    - Establish if data is missing and remove.
    - Create sub data frames.
    - Structure data into desired format.

  - Upon Extraction and preprocessing, I examine the datasets and determine patterns by:
    - Assessing the data types of variables
    - Utilizing Association rule mining, and Clustering to discover readable relations between variables.

  - Create visuals that display data stories by:
    - Choosing the appropriate type of visualization for datasets.
    - Create conceptual data plot graphs.

IST 687 Final Project

Introduction

This report presents the findings of the Top 1,000 Most Popular HULU Shows dataset consisting of 1000 rows and 181 columns consisting of various aspects of the HULU shows, such as episode count, network, and genre. The objective of this report is to look at the variables that affect a shows popularity to provide insight to HULU on what changes they can make to less popular shows to make them popular, and thus increase business. Our team developed various business questions to steer our data analysis direction. After reviewing and reading in the data, our team cleaned, munged, and prepared the data set to create data visualization plots to better illustrate our findings. Our team also developed models to analyze the data to identify attributes of various variables that have the most impact on show popularity.

Business Questions:

1. What makes a show “popular”? What factors make a show “popular”?
2. How does show popularity compare to other values within the dataset?
3. Are there correlations between variables that result in a show being “popular”?

Data Cleanse/Munge/Preparation

1. We began by reading in the data and linking it to a data frame name.
2. Due to the original dataset having 181 column, we selected a few to keep in our analysis process in order to keep this report concise. We “deleted” all the columns except for #1, #2, #3, #6, #9, #10, #15, #16, #17, #21, #39, #40, and #41
    a. #1 Show ID – the unique id for each show.
    b. #2 Show Time – the time the show airs.
    c. #3 Show Conical Name – the full name of the show.
    d. #6 Show Episode Count – the number of episodes for each show.
    e. #9 Show Genre – the genre of each show.

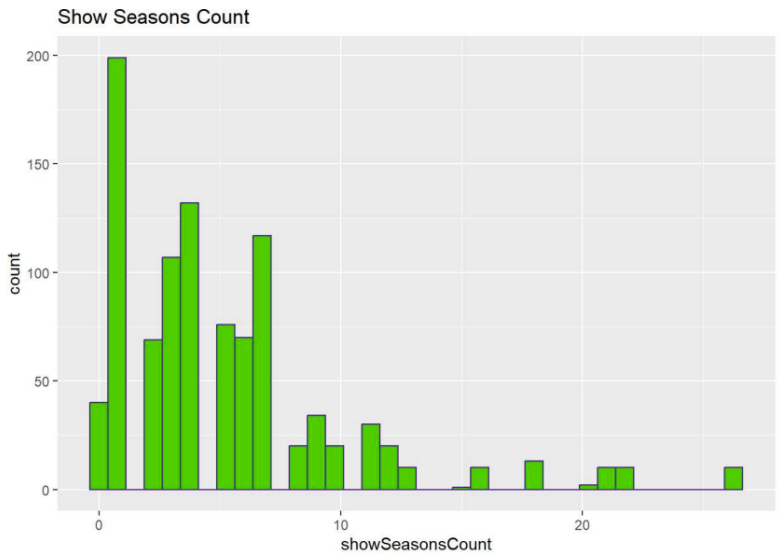
f. #10 Show Sub Genre – each possible genre of each show.
g. #15 Show Name – the name of each show as it appears on air.
h. #16 Show Rating – the rating from 1-5 of each show.
i. #17 Show Season Count- the number of seasons for each show.
j. #21 Show Type – the type of show.
k. #39 Show Company ID – the unique id of the company where the show is aired.
l. #40 Show Company Channel ID- the unique id of the channel the show is aired on.
m. #41 Show Company Name – the full name of the company that owns the show.

3. The original dataset had longer and more confusing column names, so our team renamed them to be simplified and easier to read.
4. Next, our team removed any weird symbols from the showType column and removed the "~" symbol from the showSubGenre column, as well as converted the showTime into a readable time of show.

Descriptive Statistics and Visualization

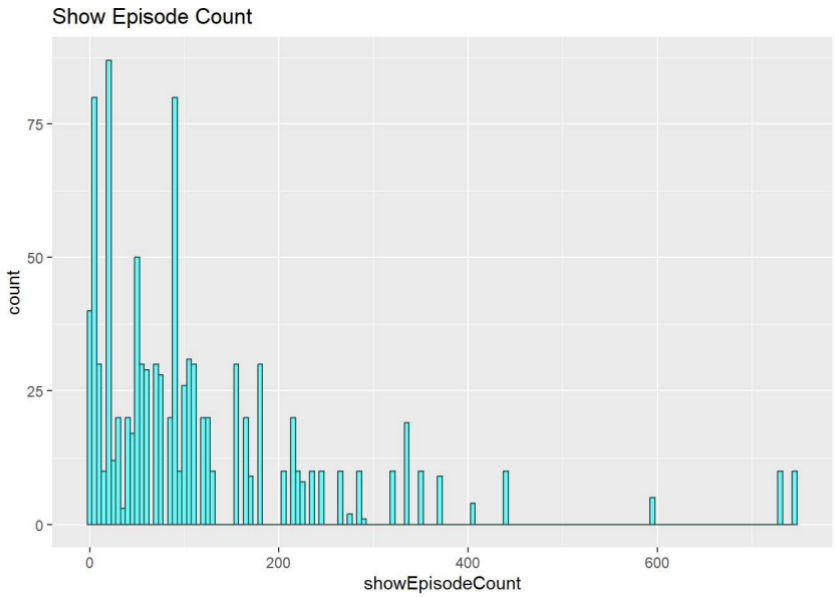
Histogram 1

Show season count: This histogram shows that as season numbers increase, the overall episodes per season tend to decrease.



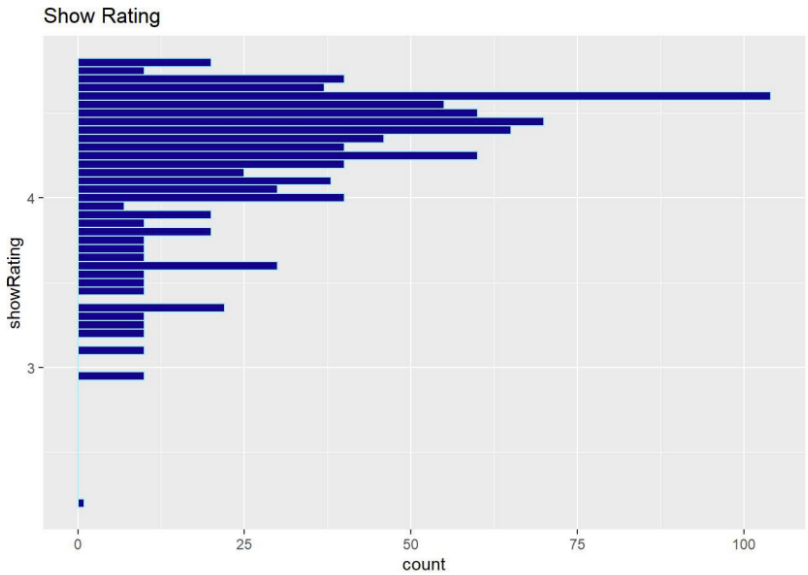
Histogram 2

Show Episode count: This histogram shows the general count of episodes. We can see that most shows remain in the lower episode count range.



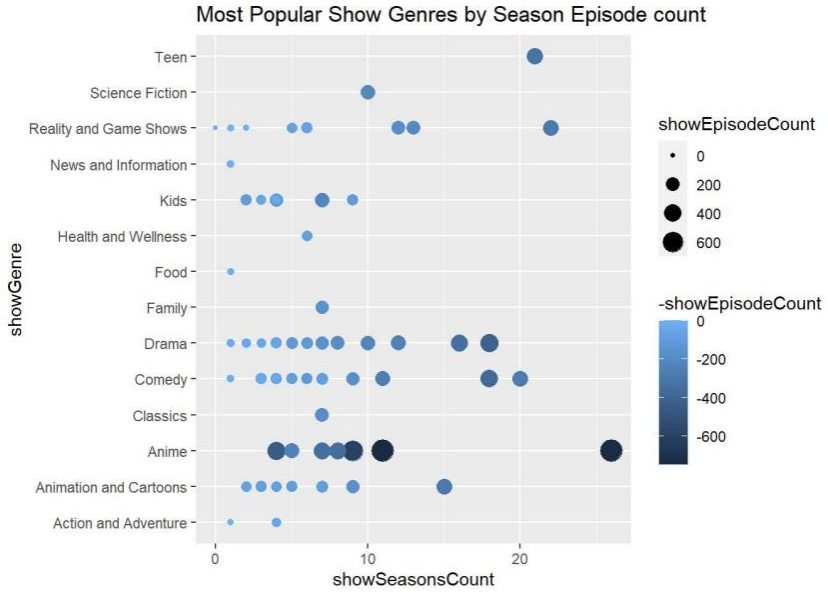
Histogram 3

Show rating: Since our overall goal is to assess show rating, we created a histogram of show rating. Since the dataset we originally pulled from is the 1000 most popular shows on Hulu, we can see that show ratings remain at about a 3 or higher.



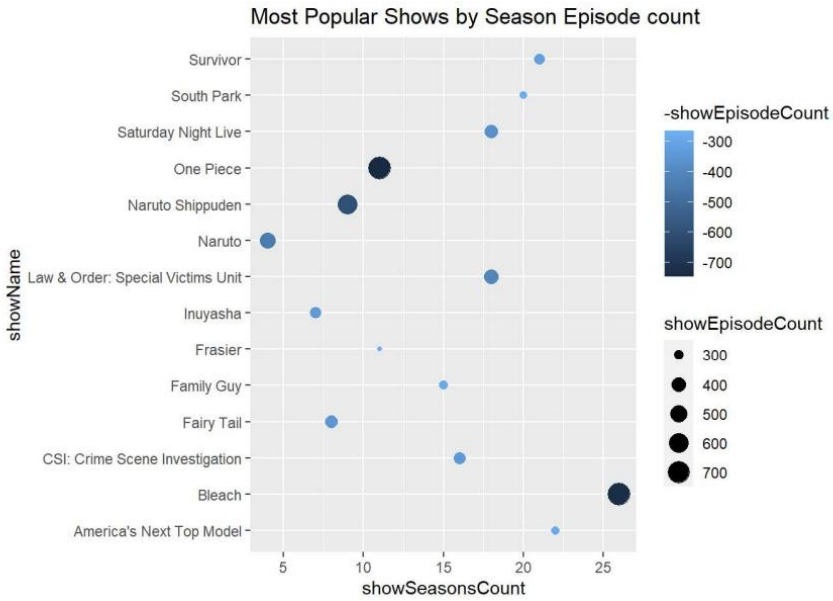
Plot 1

Season Episode Count: This histogram plots show episode count based on show season count and show genre. For each genre, we can see what shows had the most seasons and most episodes. We can see a trend where, as the number of seasons increases, the number of episodes increases as well.



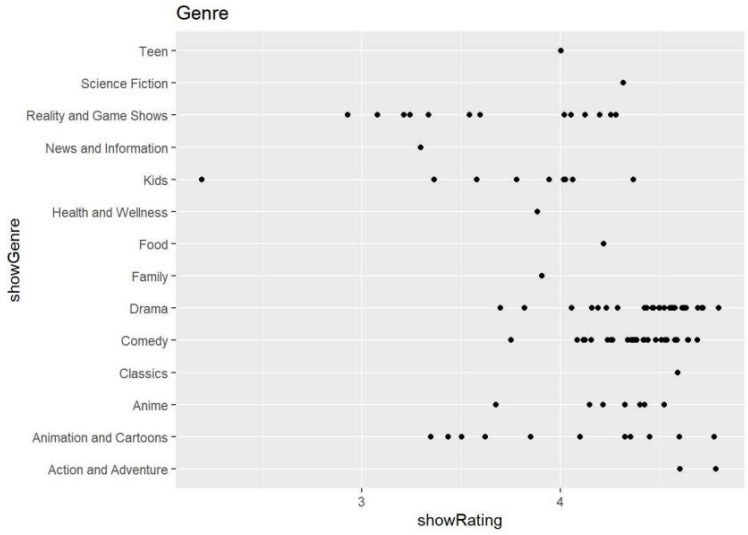
Plot 2

Season episode count 2: Since most shows that are popular have a lot of episodes, we also decided to condense our view and plot shows that had 250 episodes or more. With this condensed view, we can see that there is less of a pattern once we look at shows with more than 250 episodes.



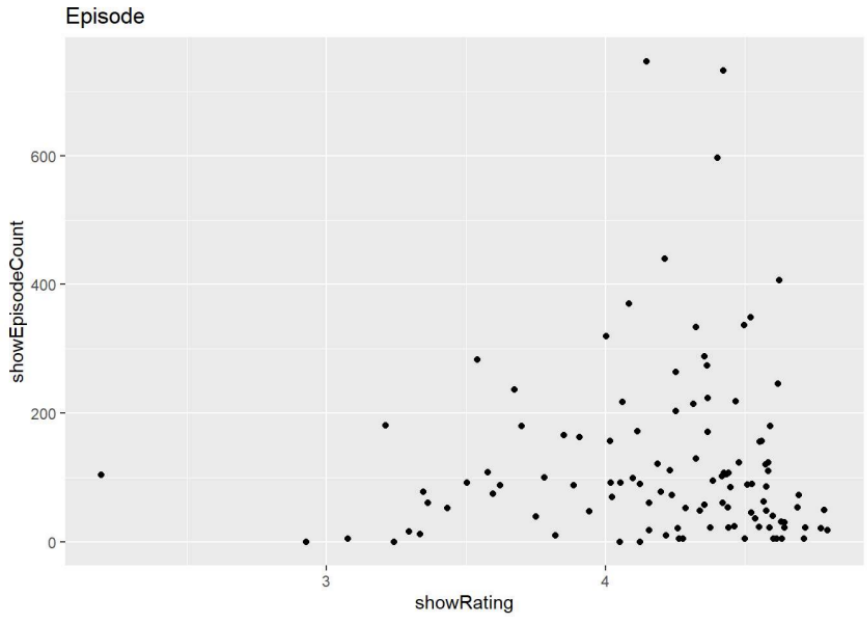
Plot 3

Show Genre vs Show Rating: This plot shows us what shows within their respective genres received the highest show ratings. From this plot we can see that “Drama” and “Comedy” genres contained the most shows with the highest ratings.



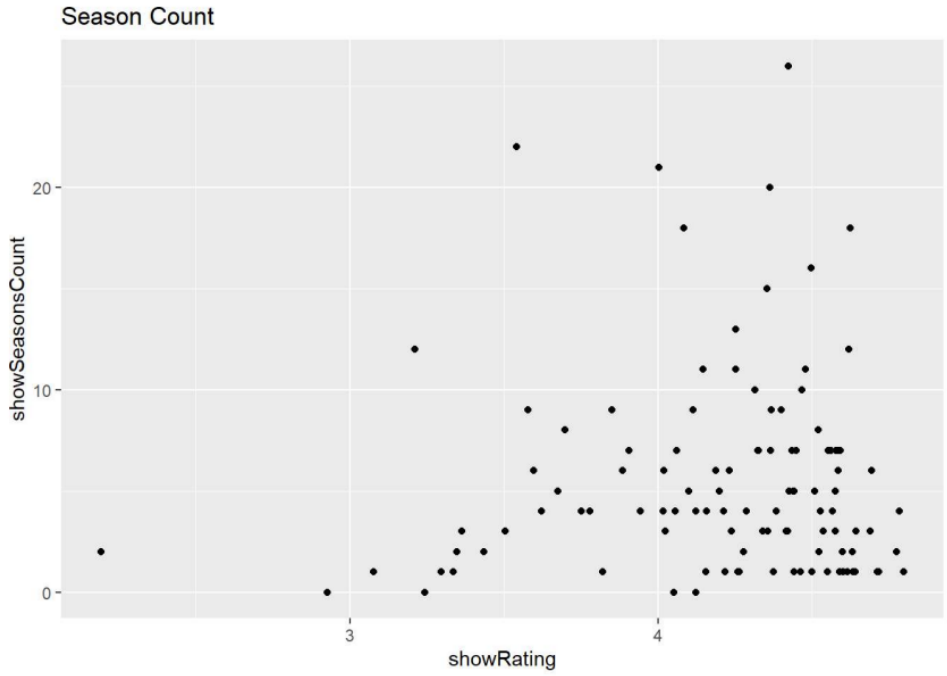
Plot 4

Show Episode Count vs Show Rating: This plot shows us what episode counts have the highest show ratings. From this plot we can see that as episode count increases, show ratings tend to also be higher.



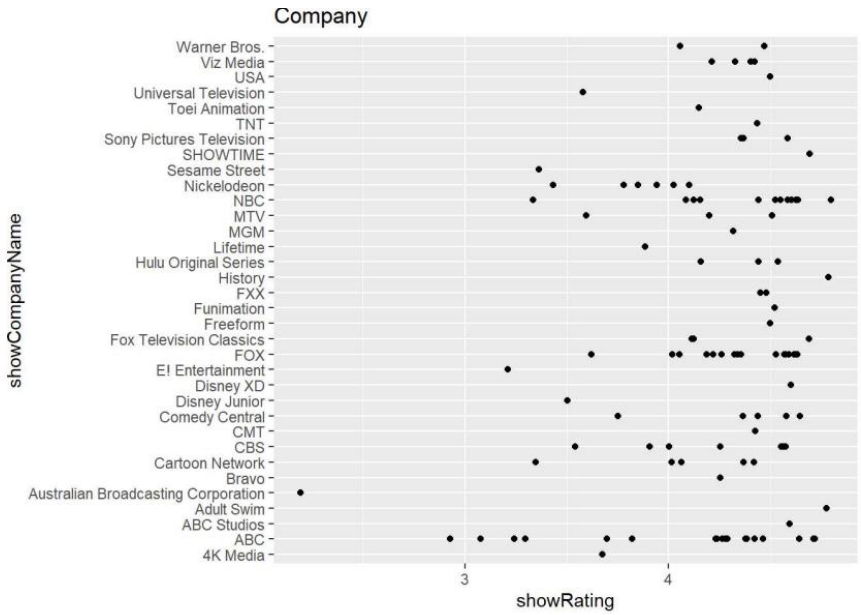
Plot 5

Show Season Count vs Show Rating: This plot shows us what shows have the highest show ratings based on season count. From this plot we can see that as season count increases, show ratings tend to also be higher.



Plot 6

Show Company vs Show Rating: This plot shows us what shows have the highest show ratings based on the company they belong to. From this plot we can see a that companies such as Nickelodeon, FOX and ABC, have more shows with higher ratings.



Data Modeling

Linear Regression Model

Introduction:

In total we ran seven linear regression models to rest the relationships between independent and dependent variables. Based on these models, we discovered what independent variables affected our dependent variable of show rating.

Model 1

In our first model, we put show rating as y, and show episode count as our x variable. We got an R^2 of 0.0004533 and adjusted R^2 of -0.0005482, showing us that there is not a strong, or significant relationship between show rating and show episode count.

```
##
## Call:
## lm(formula = showRating ~ showEpisodeCount, data = Hulu)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0202 -0.1883  0.1089  0.3349  0.5881
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4.206e+00  1.792e-02 234.687   <2e-16 ***
## showEpisodeCount 6.748e-05  1.003e-04   0.673    0.501
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.428 on 998 degrees of freedom
## Multiple R-squared:  0.0004533,  Adjusted R-squared:  -0.0005482
## F-statistic: 0.4526 on 1 and 998 DF,  p-value: 0.5012
```

Model 2

Our second model looks at show ratings as our y variable and show genre as our x variable. We got an R^2 of 0.4843 and adjusted R^2 of 0.4665, which, compared to our model 1, is statistically significant. We can take a closer look at our intercepts and see that certain columns are also more statistically significant than other columns. From that perspective, showGenreClassics and showGenreScience Fiction are less statistically significant than other aspects within show genre.

```
##
## Call:
## lm(formula = showRating ~ showGenre, data = Hulu)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.70354 -0.13561  0.03291  0.17800  0.75959
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     4.69189    0.06915  67.849  < 2e-16 ***
## showGenreAnimation and Cartoons -0.67860    0.07608  -8.919  < 2e-16 ***
## showGenreAnime                  -0.46215    0.07922  -5.833 7.36e-09 ***
## showGenreClassics               -0.10063    0.11978  -0.840 0.401004
## showGenreComedy                 -0.30477    0.07158  -4.258 2.26e-05 ***
## showGenreDrama                  -0.23947    0.07166  -3.342 0.000864 ***
## showGenreFamily                 -0.78606    0.11978  -6.563 8.52e-11 ***
## showGenreFood                   -0.47481    0.11978  -3.964 7.90e-05 ***
## showGenreHealth and Wellness    -0.80707    0.11978  -6.738 2.72e-11 ***
## showGenreKids                   -0.79519    0.07841 -10.141  < 2e-16 ***
## showGenreNews and Information   -1.39520    0.11978 -11.649  < 2e-16 ***
## showGenreReality and Game Shows -1.01065    0.07428 -13.606  < 2e-16 ***
## showGenreScience Fiction        -0.37659    0.11978  -3.144 0.001716 **
## showGenreTeen                   -0.68965    0.11978  -5.758 1.14e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3093 on 986 degrees of freedom
## Multiple R-squared:  0.4843, Adjusted R-squared:  0.4775
## F-statistic: 71.24 on 13 and 986 DF,  p-value: < 2.2e-16
```

Model 3

Model 3 looks at show rating versus show season count, being our y and x variables respectively. We get an R^2 of 4.283e-07 and adjusted R^2 of -0.001002, which, similar to model 1, is not statistically significant when we compare our x variable to our y variable.

```
##
## Call:
## lm(formula = showRating ~ showSeasonsCount, data = Hulu)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0213 -0.1912  0.1112  0.3376  0.5811
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.215e+00  2.007e-02 210.032   <2e-16 ***
## showSeasonsCount -5.768e-05  2.790e-03  -0.021    0.984
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4281 on 998 degrees of freedom
## Multiple R-squared:  4.283e-07,  Adjusted R-squared:  -0.001002
## F-statistic: 0.0004275 on 1 and 998 DF,  p-value: 0.9835
```

Model 4

Our fourth model looks at the show rating as our y variable and show company id as our x variable. This model produces an R^2 of 0.00948 and adjusted R^2 of 0.008488, which is not statistically significant, showing us that show company id does not affect the outcome of show ratings as much as other models.

```
##
## Call:
## lm(formula = showRating ~ showCompanyID, data = Hulu)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.00330 -0.19242  0.08684  0.32011  0.63270
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.256e+00  1.907e-02 223.165  < 2e-16 ***
## showCompanyID -1.353e-04  4.377e-05  -3.091  0.00205 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.426 on 998 degrees of freedom
## Multiple R-squared:  0.00948,    Adjusted R-squared:  0.008488
## F-statistic: 9.552 on 1 and 998 DF,  p-value: 0.002053
```

Model 5

This model looks at show rating versus show company channel id, as our y and x variables. The R^2 and adjusted R^2 values of 0.01081 and 0.009818 are more statistically significant than other models. This shows, if we company model 5 to model 4, that show channel id has more of an effect on the outcome of show ratings than show company id does.

```
##
## Call:
## lm(formula = showRating ~ showCompanyChannelID, data = Hulu)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9653 -0.1731  0.0945  0.3275  0.6145
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           4.250e+00  1.721e-02 246.931  < 2e-16 ***
## showCompanyChannelID -1.825e-09  5.526e-10  -3.302 0.000993 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4257 on 998 degrees of freedom
## Multiple R-squared:  0.01081,    Adjusted R-squared:  0.009818
## F-statistic:  10.9 on 1 and 998 DF,  p-value: 0.0009931
```

Model 6

Model six looks at show ratings as the y variable and show company name as our x variable. We got an R^2 of 0.3755 and adjusted R^2 of 0.3541, which is statistically significant. We can also look at the intercepts of the columns belonging to show company id and see that showCompanyNameMTV and showCompanyNameABC are among some of our statistically significant columns compared to showCompanyNameDisney Junior which is not statistically significant.

```
##
## Call:
## lm(formula = showRating ~ showCompanyName, data = Hulu)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1219 -0.1189  0.0000  0.2324  0.6662
##
## Coefficients:
##                                                 Estimate Std. Error t value
## (Intercept)                                      3.67461    0.10874  33.794
## showCompanyNameABC                               0.37494    0.11177   3.355
## showCompanyNameABC Studios                       0.91665    0.15378   5.961
## showCompanyNameAdult Swim                        1.09827    0.15378   7.142
## showCompanyNameAustralian Broadcasting Corporation -1.48145    0.36064  -4.108
## showCompanyNameBravo                             0.57814    0.15378   3.760
## showCompanyNameCartoon Network                   0.38287    0.11997   3.191
## showCompanyNameCBS                               0.52389    0.11624   4.507
## showCompanyNameCMT                               0.74989    0.15378   4.877
## showCompanyNameComedy Central                    0.67791    0.12099   5.603
## showCompanyNameDisney Junior                    -0.17099    0.15378  -1.112
## showCompanyNameDisney XD                         0.92418    0.15378   6.010
## showCompanyNameE! Entertainment                 -0.46300    0.15378  -3.011
## showCompanyNameFOX                               0.66802    0.11228   5.950
## showCompanyNameFox Television Classics           0.69304    0.13025   5.321
## showCompanyNameFreeform                          0.82431    0.15378   5.360
## showCompanyNameFunimation                        0.84635    0.15378   5.504
## showCompanyNameFXX                               0.78905    0.13317   5.925
## showCompanyNameHistory                           1.10851    0.15378   7.209
## showCompanyNameHulu Original Series              0.70337    0.13562   5.186
## showCompanyNameLifetime                          0.21021    0.15378   1.367
## showCompanyNameMGM                               0.64069    0.15378   4.166
## showCompanyNameMTV                               0.42617    0.12556   3.394
## showCompanyNameNBC                               0.67055    0.11380   5.893
## showCompanyNameNickelodeon                       0.15684    0.11855   1.323
## showCompanyNameSesame Street                    -0.31085    0.20343  -1.528
## showCompanyNameSHOWTIME                          1.01747    0.15378   6.617
## showCompanyNameSony Pictures Television          0.77099    0.12795   6.026
## showCompanyNameTNT                               0.75914    0.15378   4.937
## showCompanyNameToei Animation                    0.47208    0.15378   3.070
## showCompanyNameUniversal Television             -0.09602    0.15378  -0.624
## showCompanyNameUSA                               0.82230    0.15378   5.347
## showCompanyNameViz Media                         0.65719    0.12370   5.313
## showCompanyNameWarner Bros.                      0.58654    0.13317   4.404
##                                                 Pr(>|t|)
## (Intercept)                                      < 2e-16 ***
## showCompanyNameABC                              0.000825 ***
## showCompanyNameABC Studios                      3.51e-09 ***
## showCompanyNameAdult Swim                       1.81e-12 ***
## showCompanyNameAustralian Broadcasting Corporation 4.33e-05 ***
## showCompanyNameBravo                            0.000180 ***
## showCompanyNameCartoon Network                  0.001462 **
```

```
## showCompanyNameCBS                                7.39e-06 ***
## showCompanyNameCMT                                1.26e-06 ***
## showCompanyNameComedy Central                     2.75e-08 ***
## showCompanyNameDisney Junior                      0.266437
## showCompanyNameDisney XD                          2.63e-09 ***
## showCompanyNameE! Entertainment                   0.002673 **
## showCompanyNameFOX                                3.75e-09 ***
## showCompanyNameFox Television Classics            1.28e-07 ***
## showCompanyNameFreeform                           1.04e-07 ***
## showCompanyNameFunimation                         4.76e-08 ***
## showCompanyNameFXX                                4.34e-09 ***
## showCompanyNameHistory                            1.14e-12 ***
## showCompanyNameHulu Original Series               2.61e-07 ***
## showCompanyNameLifetime                           0.171941
## showCompanyNameMGM                                3.37e-05 ***
## showCompanyNameMTV                                0.000716 ***
## showCompanyNameNBC                                5.25e-09 ***
## showCompanyNameNickelodeon                        0.186163
## showCompanyNameSesame Street                      0.126825
## showCompanyNameSHOWTIME                           6.08e-11 ***
## showCompanyNameSony Pictures Television           2.39e-09 ***
## showCompanyNameTNT                                9.36e-07 ***
## showCompanyNameToei Animation                     0.002201 **
## showCompanyNameUniversal Television               0.532505
## showCompanyNameUSA                                1.11e-07 ***
## showCompanyNameViz Media                          1.34e-07 ***
## showCompanyNameWarner Bros.                       1.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3439 on 966 degrees of freedom
## Multiple R-squared:  0.3755, Adjusted R-squared:  0.3541
## F-statistic:  17.6 on 33 and 966 DF,  p-value: < 2.2e-16
```

Model 7

Our final model is our multiple regression model looking at show rating, as the y variable, versus show company id, show company channel id and show company name as our x variables. This multiple regression model produces an R^2 of 0.3755 and adjusted R^2 of 0.3541. This happens to be the same values from our model 6, which gives an idea that show company name has more of an effect on show rating compared to the other variables. Looking at model 7 as a whole, it is still statistically significant, meaning our x variables together result in a large effect on the outcome of show rating. We can also look at the combined aspects of the variables to see that showCompanyNameHulu Original Series and showCompanyNameAdult Swim are among some of the more statistically significant columns within our variables.

```
##
## Call:
## lm(formula = showRating ~ showCompanyID + showCompanyChannelID +
##     showCompanyName, data = Hulu)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1219 -0.1189  0.0000  0.2324  0.6662
##
## Coefficients: (2 not defined because of singularities)
##                                                  Estimate Std. Error
## (Intercept)                                     4.109e+00  8.206e-02
## showCompanyID                                   2.086e-03  3.927e-04
## showCompanyChannelID                           -3.560e-08  6.667e-09
## showCompanyNameABC                             -5.412e-01  1.021e-01
## showCompanyNameABC Studios                      1.277e-01  1.385e-01
## showCompanyNameAdult Swim                       6.581e-01  1.211e-01
## showCompanyNameAustralian Broadcasting Corporation -1.054e+00  3.898e-01
## showCompanyNameBravo                            1.147e-01  1.352e-01
## showCompanyNameCartoon Network                 -5.525e-02  7.369e-02
## showCompanyNameCBS                              9.724e-01  1.918e-01
## showCompanyNameCMT                              1.282e+00  2.298e-01
## showCompanyNameComedy Central                   3.712e-02  9.397e-02
## showCompanyNameDisney Junior                   -6.467e-01  1.206e-01
## showCompanyNameDisney XD                        1.627e-01  1.296e-01
## showCompanyNameE! Entertainment                -9.244e-01  1.352e-01
## showCompanyNameFOX                              2.171e-01  8.571e-02
## showCompanyNameFox Television Classics          2.379e-01  1.080e-01
## showCompanyNameFreeform                         1.317e-01  1.254e-01
## showCompanyNameFunimation                       1.305e-01  1.354e-01
## showCompanyNameFXX                              6.806e-01  1.194e-01
## showCompanyNameHistory                          2.946e-01  1.399e-01
## showCompanyNameHulu Original Series             1.023e+00  1.830e-01
## showCompanyNameLifetime                         9.300e-01  2.606e-01
## showCompanyNameMGM                              1.480e-01  1.343e-01
## showCompanyNameMTV                              9.957e-01  2.185e-01
## showCompanyNameNBC                              2.321e-01  8.840e-02
## showCompanyNameNickelodeon                      2.841e-01  1.386e-01
## showCompanyNameSesame Street                   -1.066e+00  1.910e-01
## showCompanyNameSHOWTIME                         1.138e-01  1.459e-01
## showCompanyNameSony Pictures Television         8.586e-01  1.407e-01
## showCompanyNameTNT                              2.960e-01  1.208e-01
## showCompanyNameToei Animation                  -4.253e-01  1.455e-01
## showCompanyNameUniversal Television            -2.671e-01  1.361e-01
## showCompanyNameUSA                              3.547e-01  1.350e-01
## showCompanyNameViz Media                               NA         NA
## showCompanyNameWarner Bros.                            NA         NA
##                                                 t value Pr(>|t|)
## (Intercept)                                      50.073  < 2e-16 ***
## showCompanyID                                     5.313 1.34e-07 ***
## showCompanyChannelID                             -5.339 1.16e-07 ***
## showCompanyNameABC                               -5.300 1.44e-07 ***
```

```
## showCompanyNameABC Studios                              0.922 0.356662
## showCompanyNameAdult Swim                                5.433 7.03e-08 ***
## showCompanyNameAustralian Broadcasting Corporation      -2.703 0.006983 **
## showCompanyNameBravo                                     0.848 0.396370
## showCompanyNameCartoon Network                          -0.750 0.453546
## showCompanyNameCBS                                       5.071 4.75e-07 ***
## showCompanyNameCMT                                       5.578 3.16e-08 ***
## showCompanyNameComedy Central                            0.395 0.692911
## showCompanyNameDisney Junior                            -5.362 1.03e-07 ***
## showCompanyNameDisney XD                                 1.255 0.209639
## showCompanyNameE! Entertainment                         -6.835 1.45e-11 ***
## showCompanyNameFOX                                       2.533 0.011471 *
## showCompanyNameFox Television Classics                   2.203 0.027822 *
## showCompanyNameFreeform                                  1.050 0.293982
## showCompanyNameFunimation                                0.964 0.335479
## showCompanyNameFXX                                       5.701 1.58e-08 ***
## showCompanyNameHistory                                   2.106 0.035480 *
## showCompanyNameHulu Original Series                      5.588 2.99e-08 ***
## showCompanyNameLifetime                                  3.569 0.000376 ***
## showCompanyNameMGM                                       1.102 0.270757
## showCompanyNameMTV                                       4.557 5.86e-06 ***
## showCompanyNameNBC                                       2.626 0.008779 **
## showCompanyNameNickelodeon                               2.050 0.040640 *
## showCompanyNameSesame Street                            -5.583 3.07e-08 ***
## showCompanyNameSHOWTIME                                  0.780 0.435680
## showCompanyNameSony Pictures Television                  6.103 1.51e-09 ***
## showCompanyNameTNT                                       2.451 0.014418 *
## showCompanyNameToei Animation                           -2.924 0.003537 **
## showCompanyNameUniversal Television                     -1.962 0.050021 .
## showCompanyNameUSA                                       2.626 0.008765 **
## showCompanyNameViz Media                                    NA       NA
## showCompanyNameWarner Bros.                                 NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3439 on 966 degrees of freedom
## Multiple R-squared:  0.3755, Adjusted R-squared:  0.3541
## F-statistic:  17.6 on 33 and 966 DF,  p-value: < 2.2e-16
```

IST 707 HW #3

Introduction

In 1989, UK banks introduced the Personal Equity Plan (PEP), targeting individuals over 18 to invest primarily in British companies. PEPs provided tax-free accumulation of income and capital gains from shares, unit trusts, or investment trusts, making them highly appealing. This sparked a surge in potential clients seeking tax-free investment income.

However, alongside PEPs' introduction came the challenge of identifying eligible clients for this investment opportunity. This report analyzes bank data containing 11 initial client attributes, including demographic and banking information. These attributes will be used to assess clients' eligibility for the new Personal Equity Plan. Through Association Rule discovery on the bank data, the analysis seeks to identify strong rules indicating a customer's suitability for a PEP.

Analysis and Models

The first stage involves integrating and configuring the necessary libraries and packages into the script file. Following this, the data will be imported from a CSV file and organized into a data frame named "bankdata." Utilizing the "str()" function, an examination of the data was conducted, leading to the following observations:

```
'data.frame':   600 obs. of  12 variables:
 $ id         : chr  "ID12101" "ID12102" "ID12103" "ID12104" ...
 $ age        : int  48 40 51 23 57 57 22 58 37 54 ...
 $ sex        : chr  "FEMALE" "MALE" "FEMALE" "FEMALE" ...
 $ region     : chr  "INNER_CITY" "TOWN" "INNER_CITY" "TOWN" ...
 $ income     : num  17546 30085 16575 20375 50576 ...
 $ married    : chr  "NO" "YES" "YES" "YES" ...
 $ children   : int  1 3 0 3 0 2 0 0 2 2 ...
 $ car        : chr  "NO" "YES" "YES" "NO" ...
 $ save_act   : chr  "NO" "NO" "YES" "NO" ...
 $ current_act: chr  "NO" "YES" "YES" "YES" ...
 $ mortgage   : chr  "NO" "YES" "NO" "NO" ...
 $ pep        : chr  "YES" "NO" "NO" "NO" ...
```

Data Cleaning

Before proceeding to the subsequent data processing step, it is imperative to eliminate, discretize, or convert certain unnecessary data. Initially, the "id" attribute and all instances of missing values (NAs) within the dataset were removed.

```r
# Removing unnecessary data and descretizing
bankdata <- bankdata[,-1]
```

```r
# Checking for and removing NAs
(sum(is.na(bankdata)))
bankdata <- bankdata[complete.cases(bankdata),]
(sum(is.na(bankdata)))
```

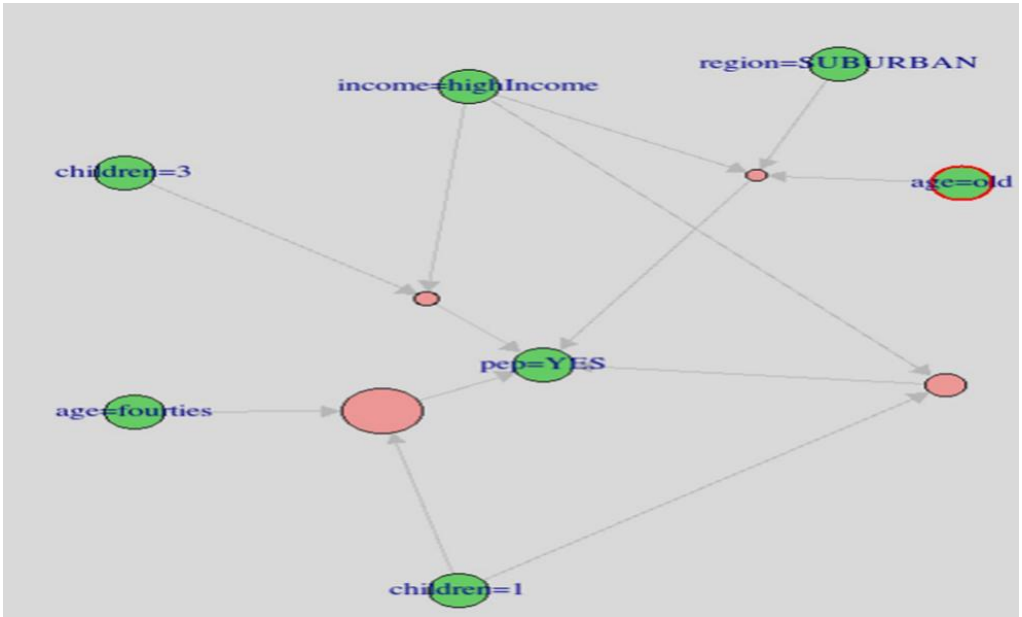
Two variables, income and age, underwent discretization. Income was categorized into three groups: "LowIncome," "MidIncome," and "HighIncome." Age was divided into seven groups: "Child," "Teen," "Twenties," "Thirties," "Forties," "Fifties," and "Senior." Additionally, the "children" attribute was transformed from numeric to factor format.

```r
# Using cut function to Descretize age variable into 7 age groups.
bankdata$age <- cut(bankdata$age, breaks = c(0,10,20,30,40,50,60,Inf),
                    labels=c("Child","Teens","Twenties","Thirties","Fourties"
,"Fifties","Senior"))
```

```r
# Converting children variable from numeric
bankdata$children <- factor(bankdata$children)
```

Several Association Rule Discovery tests were conducted (totaling 7 tests), yielding the following outcomes:

```
     lhs                                  rhs                 support confidence coverage lift count
[1]  {income=HighIncome}                  => {save_act=YES}      0.134       1.00    0.134  1.5    80
[2]  {region=SUBURBAN,
      income=HighIncome}                  => {pep=YES}           0.017       0.91    0.018  2.0    10
[3]  {region=SUBURBAN,
      income=HighIncome}                  => {save_act=YES}      0.018       1.00    0.018  1.5    11
[4]  {region=SUBURBAN,
      current_act=NO}                     => {married=YES}       0.018       0.92    0.020  1.4    11
[5]  {region=SUBURBAN,
      married=NO}                         => {current_act=YES}   0.032       0.95    0.033  1.3    19
[6]  {region=SUBURBAN,
      mortgage=NO}                        => {current_act=YES}   0.060       0.90    0.067  1.2    36
[7]  {income=HighIncome,
      children=3}                         => {region=RURAL}      0.013       1.00    0.013  6.2     8
[8]  {income=HighIncome,
      children=3}                         => {pep=YES}           0.013       1.00    0.013  2.2     8
[9]  {income=HighIncome,
      children=3}                         => {mortgage=NO}       0.013       1.00    0.013  1.5     8
[10] {income=HighIncome,
      children=3}                         => {save_act=YES}      0.013       1.00    0.013  1.5     8
[11] {region=TOWN,
      children=3}                         => {pep=NO}            0.032       0.95    0.033  1.8    19
[12] {children=3,
      save_act=NO}                        => {pep=NO}            0.037       1.00    0.037  1.8    22
[13] {children=3,
```



Insights

1. Clients who have either a savings account or a checking account, with one child, and are in their "Forties" have a very high chance of qualifying for a PEP.
2. Teenagers are 70% NOT likely to be considered for a PEP (Must also be at least 18 years of age).
3. People with a single child have an 81% chance of being approved for a PEP.
4. Individuals who earn a high income and live in a suburban area have a high chance of being approved for a PEP.
5. Clients with a high income and more than one child have over 96% probability of favorable approval for a PEP.

☑ Python

- Display use of Python to transform structured data by:
  - Extracting data from a structured data source
  - Process the data for analysis
  - Creating graphs
  - Recognize patterns and attributes that contribute to car value depreciation and how to predict future resale value

- Display my use of Python to optimize a stock portfolio data by:
  - Web scraping stock data
  - Create reports of the stock data and create the optimal weighted average percent of stocks that should be in a stock portfolio to generate highest returns

IST 652 Final Project

Introduction

This analysis is aimed at discovering which top 25 vehicles have the best resale value after a minimum of 2.5 years of ownership.

Install seaborn for statistical visualization and import necessary modules for data analysis and more visualizations.

```
pip install seaborn
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
cars = pd.read_csv('/Users/randy/OneDrive/Documents/Syracuse Stuff/IST 652/data/data/vehicles.csv')
cars = cars[cars.price<1e5]#drop cars with prices above $100,000 and below $100
cars = cars[cars.price>100]
cars = cars[['id', 'price', 'year', 'manufacturer', 'model', 'odometer', 'type', 'lat', 'long']]#we will only look at these columns in the analysis
cars['age'] = 2020 - cars['year'] #add an age column
cars.columns
```

```
Index(['id', 'price', 'year', 'manufacturer', 'model', 'odometer', 'type',
       'lat', 'long', 'age'],
      dtype='object')
```

The previous follows a sequential structure, where each line of code performs a specific operation on the dataset. Let's break down the code step by step:

Reading the Dataset: The first line of code reads the car data from a CSV file using the read_csv() function from the Pandas library. The file path is specified as '/Users/randy/OneDrive/Documents/Syracuse Stuff/IST 652/data/data/vehicles.csv'.

Filtering by Price: The next two lines of code filter out cars with prices above $100,000 and below $100. This is achieved by using conditional filtering with the price column of the dataset.

Column Selection: The following line of code selects specific columns for analysis. The columns included are 'id', 'price', 'year', 'manufacturer', 'model', 'odometer', 'type', 'lat', and 'long'. By selecting only these columns, we can focus on the relevant information for our analysis.

Adding an Age Column: The last line of code adds an 'age' column to the dataset. This is done by subtracting the 'year' column from the current year (2020). The resulting 'age' column represents the age of each car in years.

Data Cleaning

```
def get_missing_info(df):
    num_entries = df.shape[0]*df.shape[1]
    null_entries = df.isnull().sum().sum()
    percent_empty = null_entries/num_entries*100
    num_missing = df.isna().sum()
    percent_missing = num_missing/len(df)*100
    col_modes = df.mode().loc[0]
    percent_mode = [df[x].isin([df[x].mode()[0]]).sum()/len(df)*100 for x in df]
    missing_value_df = pd.DataFrame({'num_missing': num_missing,
                                     'percent_missing': percent_missing,
                                     'mode': col_modes,
                                     'percent_mode':percent_mode})
    print('total empty percent:', percent_empty, '%')
    print('columns that are more than 97% mode:', missing_value_df.loc[missing_value_df['percent_mode']>97].index.values)
    return(missing_value_df)
get_missing_info(cars)
```

```
total empty percent: 4.275598390825901 %
columns that are more than 97% mode: []
```

	num_missing	percent_missing	mode	percent_mode
id	0	0.000000	7208549803	0.000238
price	0	0.000000	6995.0	0.951200
year	1024	0.243325	2017.0	8.842378
manufacturer	15977	3.796482	ford	17.229474
model	4225	1.003952	f-150	1.711589
odometer	48124	11.435306	0.0	0.322690
type	103813	24.668221	sedan	20.126795
lat	2873	0.682687	33.779214	1.094723
long	2873	0.682687	-84.411811	1.094723
age	1024	0.243325	3.0	8.842378

The provided code defines a function, get_missing_info, designed to analyze and report missing information within a DataFrame. The function conducts a comprehensive assessment by calculating various metrics, including the total number of entries, the number of null entries, the percentage of empty cells, the number of missing values in each column, the percentage of missing values in each column, the mode of each column, and the percentage of values matching the mode.

Upon execution, the function prints the total empty percentage and identifies columns where more than 97% of the values match the mode. The calculated information is then compiled into a new DataFrame, missing_value_df, which is subsequently returned by the function.

This function serves as a valuable tool for data analysts and researchers, facilitating a nuanced understanding of missing data patterns and aiding in decision-making processes related to data cleaning and imputation strategies.

```
import math
```

```
#fill missing type column with mode from model column
def modef(x):#get mode of groupby row
    m = pd.Series.mode(x)
    if len(m)==1:
        return m
    if len(m)==0:
        return 'unknown'
    else: return m[0]

def isna(x):#check if entry is nan
    try:
        out = math.isna(float(x))
    except:
        out = False
    return(out)

def fill_type(x):#fill type column with mode of model columns
    if isna(x['type']):
        try:
            out = model_types[x['model']]
        except:
            out = 'unknown'
    else:
        out = x['type']
    return(out)

model_types = cars.groupby(['model'])['type'].agg(modef)
cars['type'] = cars.apply(fill_type, axis=1)
```

The code is made up of three functions and a few lines of code to apply these functions to the dataset.

The modef(x): function takes a group of values as input (x) and returns the mode of that group. It uses the pd.Series.mode() function to calculate the mode. If there is only one mode, it returns that mode. If there are no modes (i.e., all values are unique), it returns 'unknown'. If there are multiple modes, it returns the first mode.

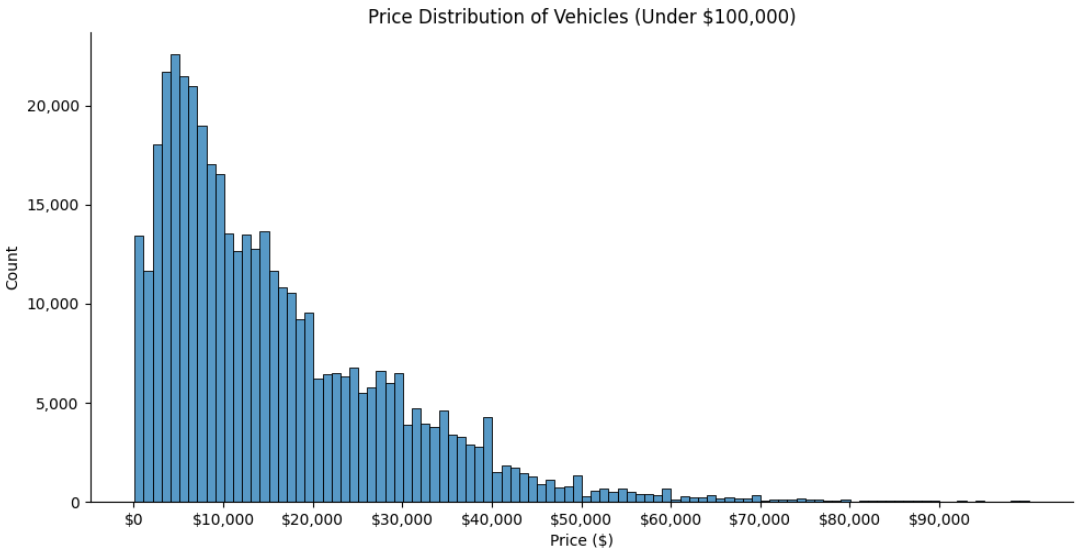
The isna(x): function checks if a value (x) is NaN (not a number) or not. It uses the math.isna() function to check if the value is NaN. If the value is NaN, it returns True. Otherwise, it returns False.

The fill_type(x): function fills the missing values in the "type" column based on the corresponding "model" group. If the "type" value is NaN, it tries to find the mode of the "model" group using the model_types dictionary. If the mode is found, it returns the mode. If the mode is not found or the "type" value is not NaN, it returns the original "type" value.

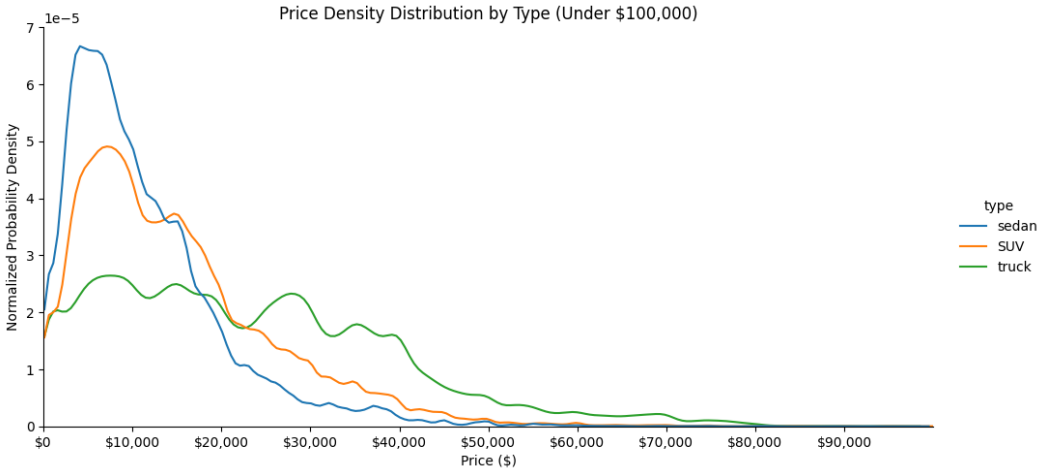
The code aims to fill missing values in the "type" column of a dataset using the mode of the corresponding "model" group.

Data Visualization

```
#plot price histogram
sns.displot(cars, x='price', binwidth=1000, height=5, aspect=2)#, bw_adjust=0.4)
plt.xticks(range(0,int(1e5), int(1e4)), labels=['${:,.0f}'.format(x) for x in range(0, int(1e5), int(1e4))],)
plt.gca().get_yaxis().set_major_formatter(plot.FuncFormatter(lambda x, loc: "{:,}".format(int(x))))
plt.xlabel('Price ($)')
plt.title('Price Distribution of Vehicles (Under $100,000)')
plt.show()
```




```
#plot pricing probability density for different types of vehicle
cars_plt = cars[cars.type.isin(['sedan', 'SUV', 'truck'])]
sns.displot(cars_plt, x='price', hue='type', kind='kde', bw_adjust=0.6, cut=0,
common_norm=False, height=5, aspect=2)
plt.xticks(range(0,int(1e5), int(1e4)), labels=['${:,}'.format(x) for x in range(0, int(1e5),
int(1e4))])
plt.xlabel('Price ($)')
plt.xlim(0,int(1e5))
plt.ylabel('Normalized Probability Density')
plt.title('Price Density Distribution by Type (Under $100,000)')
plt.show()
```



Analyzing the price distribution of vehicles is essential for understanding the affordability and popularity of different models.

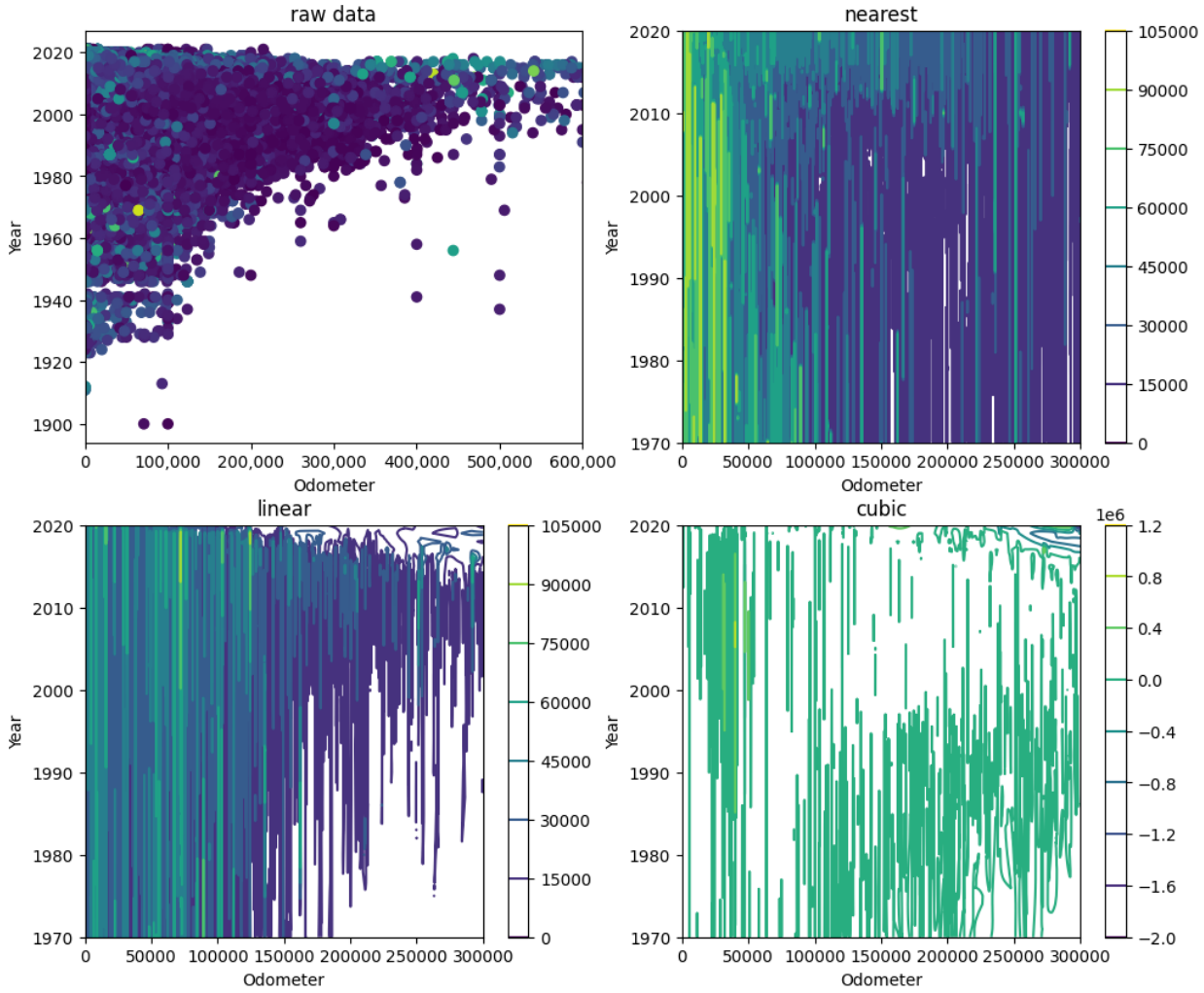
The code explored the price density distribution for different types of vehicles using a density plot. By visualizing the probability density of vehicle prices, we can gain insights into the distribution patterns and compare them across different vehicle types. This analysis can be useful for understanding market trends, pricing strategies, and customer preferences in the automotive industry.

The Price Density diagram illustrates the smoothed distribution of prices for sedans, SUVs, and trucks, with each curve normalized to have an area under 1. Notably, sedans exhibit a pronounced skewness toward the lower price spectrum. In comparison, the SUV curve bears closer semblance to the sedan curve than the truck curve, suggesting a discernible trend wherein SUVs are increasingly adopted as the primary mode of transportation, colloquially referred to as the 'daily driver'.

Trucks, in contrast, demonstrate a bifurcated pricing structure, featuring lower-end options priced below $10,000 alongside a substantial presence in the $30,000-$40,000 range. Across all vehicle categories, a discernible pricing strategy is observed, wherein prices often strategically hover just below multiples of $10,000. This pricing tactic is evidently employed to create a psychological perception of reduced cost, thereby potentially influencing consumer perceptions regarding the affordability of the respective vehicles.

```
import numpy as np
from scipy.interpolate import griddata
from matplotlib.ticker import FuncFormatter
```

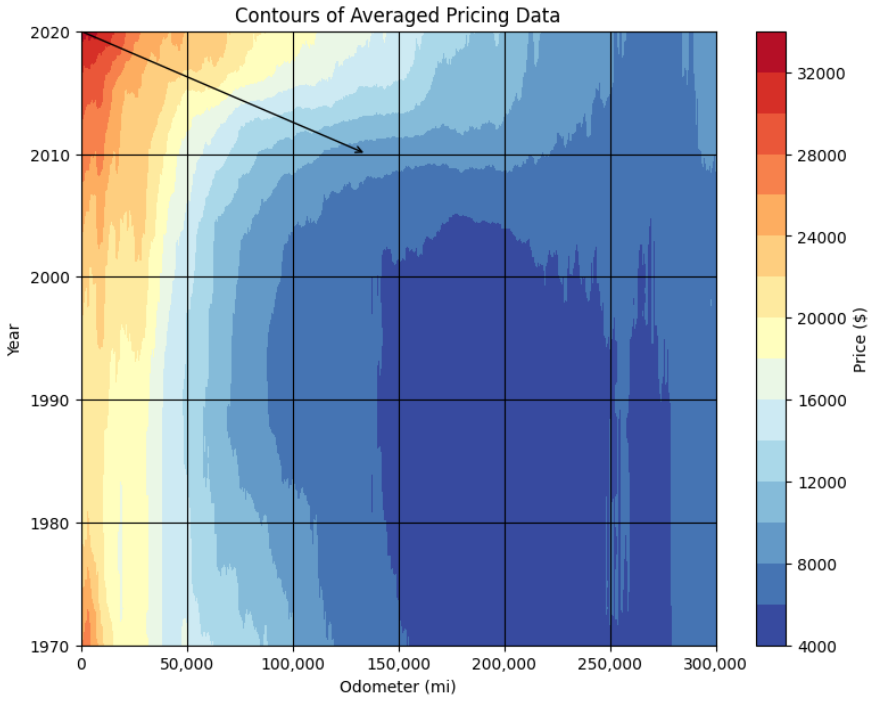
```
carsd = cars.dropna(axis=0, subset=['odometer', 'year'])  # drop rows if odometer or year are missing

# available sample data for the contour plot
xs = carsd['odometer']
ys = carsd['year']
zs = carsd['price']
points = np.array([xs, ys]).T

# we wish to interpolate the data above onto the grid below
grid_x, grid_y = np.meshgrid(
    np.linspace(0, 3e5, 10000),  # odometer goes from 0 to 300,000km with steps of 300,000/10000 = 30mi
    np.arange(1970, 2021, 1)  # year goes from 1970 to 2021 in steps of 1 year
)

# try out three different methods of interpolation
grid_z0 = griddata(points, zs, (grid_x, grid_y), method='nearest')
grid_z1 = griddata(points, zs, (grid_x, grid_y), method='linear')
grid_z2 = griddata(points, zs, (grid_x, grid_y), method='cubic')

# plot the raw data
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs[0, 0].scatter(xs, ys, c=zs)
axs[0, 0].set_xlim(0, 6e5)
axs[0, 0].set_title('raw data')
axs[0, 0].set_xlabel('Odometer')
axs[0, 0].set_ylabel('Year')

# Define a formatter function to add commas to the x-axis ticks
def comma_formatter(x, pos):
    return "{:,}".format(int(x))

# Apply the formatter to the x-axis ticks
axs[0, 0].xaxis.set_major_formatter(FuncFormatter(comma_formatter))

# plot the three different interpolation methods
im = axs[0, 1].contour(grid_x, grid_y, grid_z0)
fig.colorbar(im, ax=axs[0, 1])
axs[0, 1].set_title('nearest')
axs[0, 1].set_xlabel('Odometer')
axs[0, 1].set_ylabel('Year')

im = axs[1, 1].contour(grid_x, grid_y, grid_z2)
fig.colorbar(im, ax=axs[1, 1])
axs[1, 1].set_title('cubic')
axs[1, 1].set_xlabel('Odometer')
axs[1, 1].set_ylabel('Year')

plt.show()
```



The production of a contour plot juxtaposing price against odometer reading and year provides a comprehensive overview of vehicular depreciation trends throughout their lifespan. However, the complexity of generating such a plot is compounded by the inherent noise and sparsity characterizing the pricing dataset. Specifically, for a given combination of year and odometer reading, the dataset exhibits multiple discrete price points, and the data distribution lacks a structured grid pattern. To address this challenge, an initial step involved the interpolation of pricing data, subsequently augmented by a smoothing process employing a moving average filter.

```
from scipy.signal import convolve2d
```

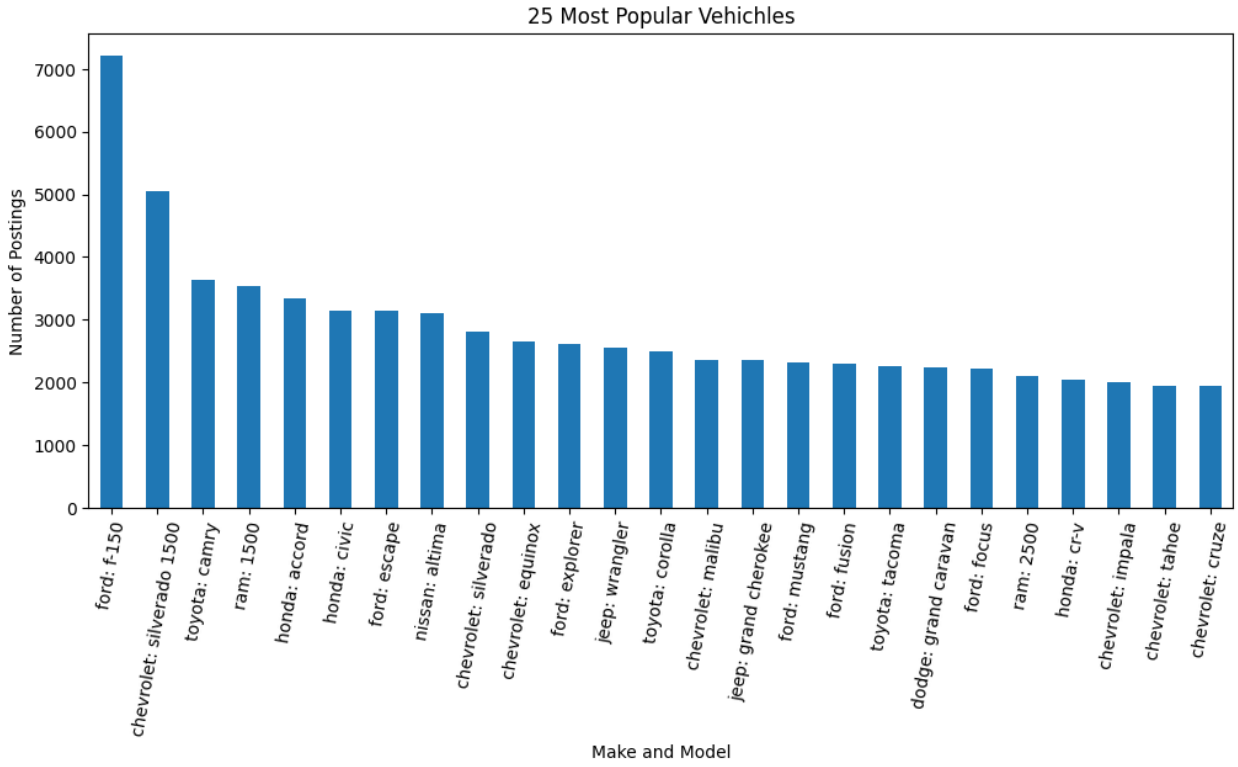
```python
# Filter out noise in the interpolated dataset and plot the final contour.
sz_o = 500  # size of averaging window for odometer (500 steps * 60 mi/step = 30,000 mi)
sz_y = 3    # size of averaging window for year (3 years or +-1 year)
kernel = np.ones((sz_y, sz_o)) / (sz_y * sz_o)  # averaging kernel, corresponds to averaging over +-15000 mi and +-1 year
grid_z0f = convolve2d(grid_z0, kernel, boundary='symm', mode='same')  # run a moving average over the 'nearest' interpolated dataset

fig, ax = plt.subplots(1, figsize=(9, 7))
im = ax.contourf(grid_x, grid_y, grid_z0f, levels=15, cmap='RdYlBu_r', zorder=0)
cbar = fig.colorbar(im, ax=ax)
cbar.set_label('Price ($)')

# Define a formatter function to add commas to the x-axis ticks (odometer)
def comma_formatter(x, pos):
    return "{:,}".format(int(x))

# Apply the formatter to the x-axis ticks (odometer)
ax.set_xlim(0, 3e5)
ax.set_xlabel('Odometer (mi)')
ax.xaxis.set_major_formatter(FuncFormatter(comma_formatter))


# Apply the formatter to the y-axis ticks (price)
ax.set_ylabel('Year')
ax.set_title('Contours of Averaged Pricing Data')
ax.grid(True, color='k')
ax.annotate("", xy=(1.35e5, 2010), xytext=(0, 2020), arrowprops=dict(arrowstyle="->", color='k'))
plt.show()


xloc_e = np.where((1.349e5<grid_x[0]) & (grid_x[0]<1.36e5)) #find x grid location where km's driven is what we want
yloc_e = 40 #row 30 of the y grid is 2010
price_end = grid_z0f[yloc_e,xloc_e[0]]
yloc_s = 50 #row 40 of the y grid is 2020
xloc_s = np.where(grid_x[0]==0)
price_start = grid_z0f[yloc_s,xloc_s[0]]
depr_rate = ((price_start-price_end)/1.35e5)[0]
print('Benchmark Depreciation rate: ${:.2f}/mi'.format(depr_rate))
```



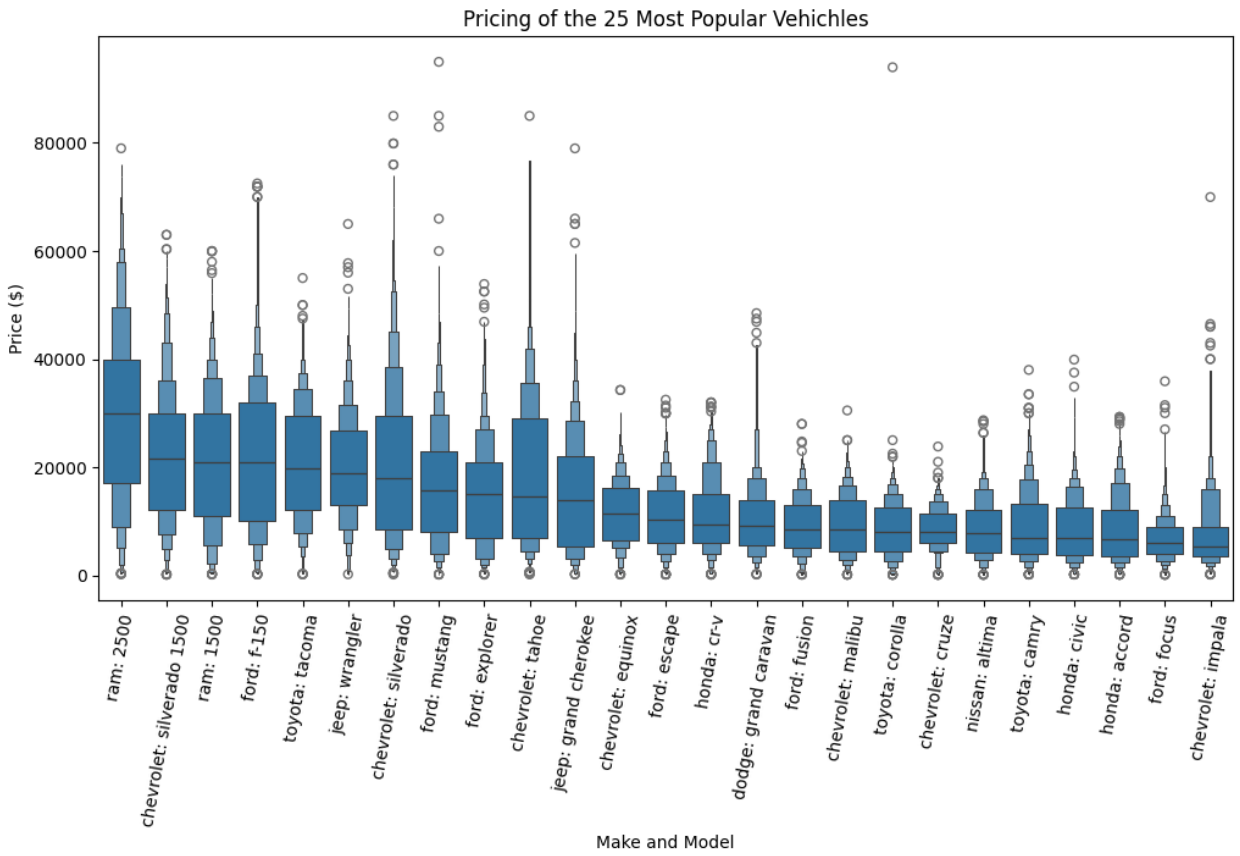
Benchmark Depreciation rate: $0.18/mi

According to the data analysis, a newly acquired vehicle undergoes an approximate depreciation of 45% (equivalent to $16,000) over a 20-year period in the absence of any driving activity. In contrast, a comparable vehicle, subjected to an annual mileage of 50,000 miles, experiences a depreciation of 30% ($10,000) within a single year. To provide a practical context, the plotted arrow delineates the depreciation trajectory for a vehicle driven at an annual rate of 13,500 miles, reflective of the average mileage in the United States, over a span of 10 years. This scenario results in a total depreciation of 72% ($34,000 to $11,000), corresponding to a depreciation rate of $0.18 per mile.

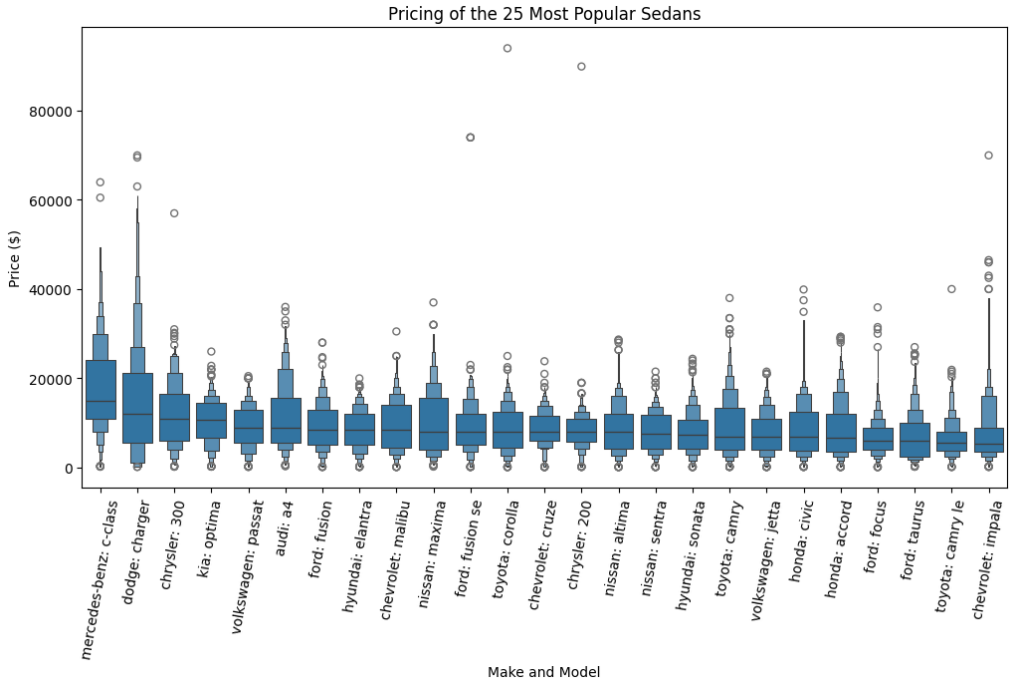
```
#Take a look at the most popular vehicles for sale
cars['make_model'] = cars['manufacturer'] + ': ' + cars['model'] #add a column with the make and model in one string (for plotting)
com_cars = cars.make_model.value_counts()[:25]#the 25 most popular cars

#plot the results
fig = com_cars.plot.bar(figsize=(12,5))
plt.xlabel('Make and Model')
plt.ylabel('Number of Postings')
plt.title('25 Most Popular Vehichles')
plt.xticks(rotation=80)
plt.show()
```

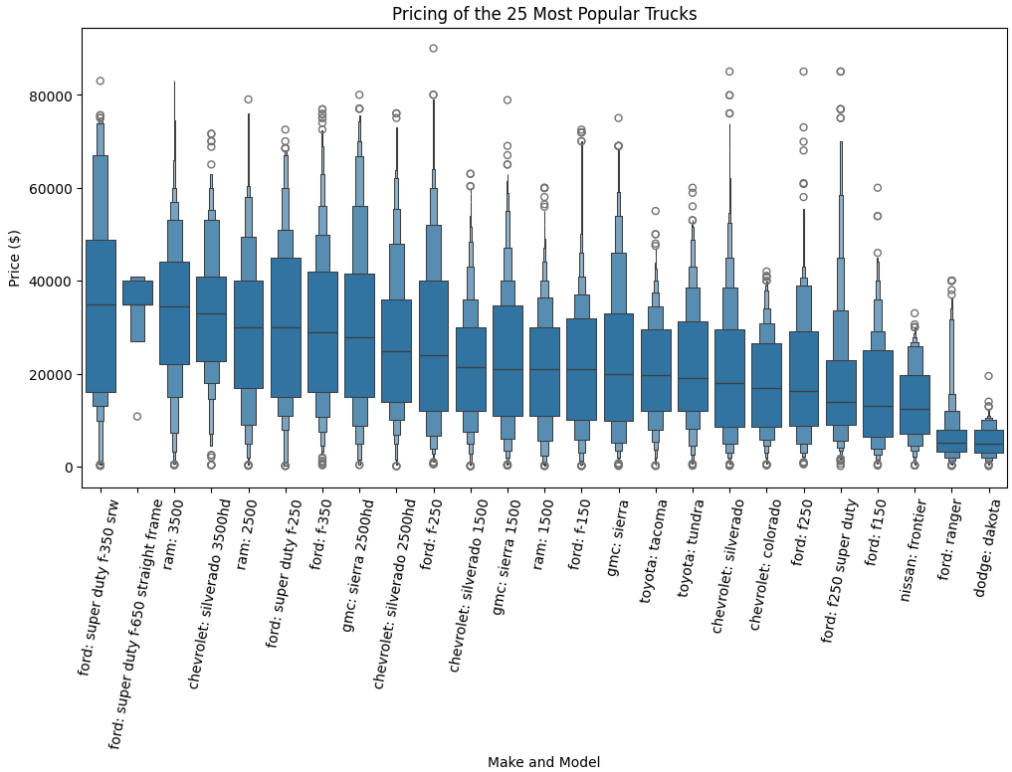


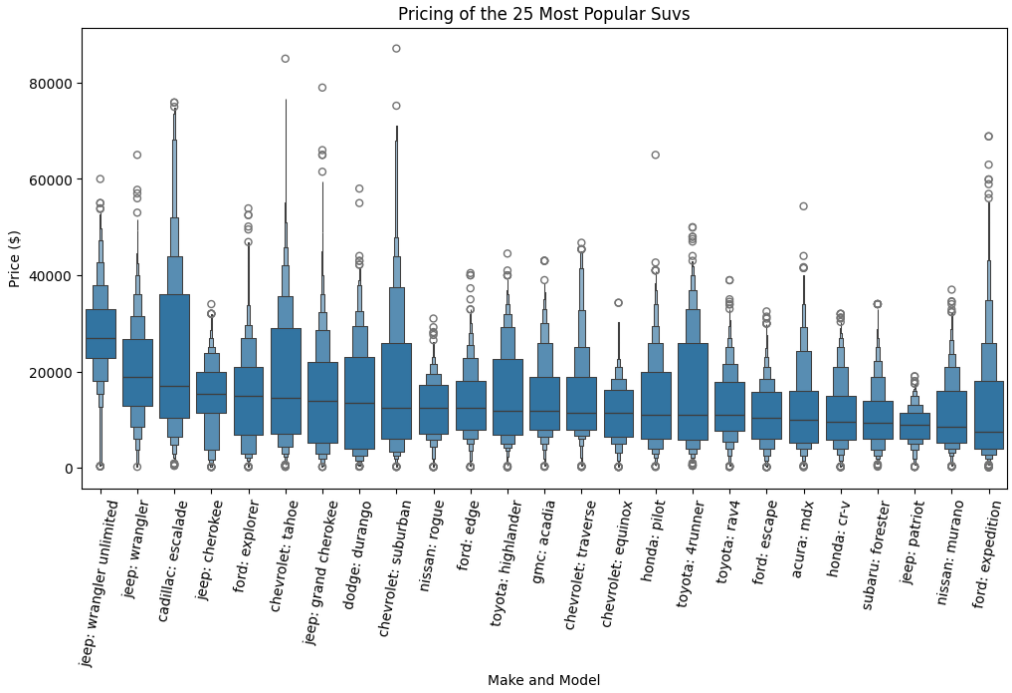

```
#plot the average prices of the 25 most popular cars
com_price = cars.loc[cars.make_model.isin(com_cars.index)]
ordered_labels =
com_price.groupby('make_model').price.median().sort_values(ascending=False).index.values

fig, ax = plot.subplots(figsize=(12,6))
sns.boxenplot(data=com_price, x="make_model", y="price", order=ordered_labels, ax=ax)
plt.xticks(rotation = 80)
plt.xlabel('Make and Model')
plt.ylabel('Price ($)')
plt.title('Pricing of the 25 Most Popular Vehichles')
plt.show()
```




```
#plot the average prices of the 25 most popular trucks, trucks, and SUVS
for thing in ['sedan', 'truck', 'SUV']:
    com = cars[cars['type']==thing].make_model.value_counts()[0:25].index
    com_price = cars.loc[cars.make_model.isin(com)]
    ordered_labels =
com_price.groupby('make_model').price.median().sort_values(ascending=False).index.values

    fig, ax = plot.subplots(figsize=(12,6))
    seab.boxenplot(data=com_price, x="make_model", y="price", order=ordered_labels, ax=ax)
    plot.xticks(rotation = 80)

    plot.xlabel('Make and Model')
    plot.ylabel('Price ($)')
    plot.title('Pricing of the 25 Most Popular {}s'.format(thing.capitalize()))
    plot.show()
```

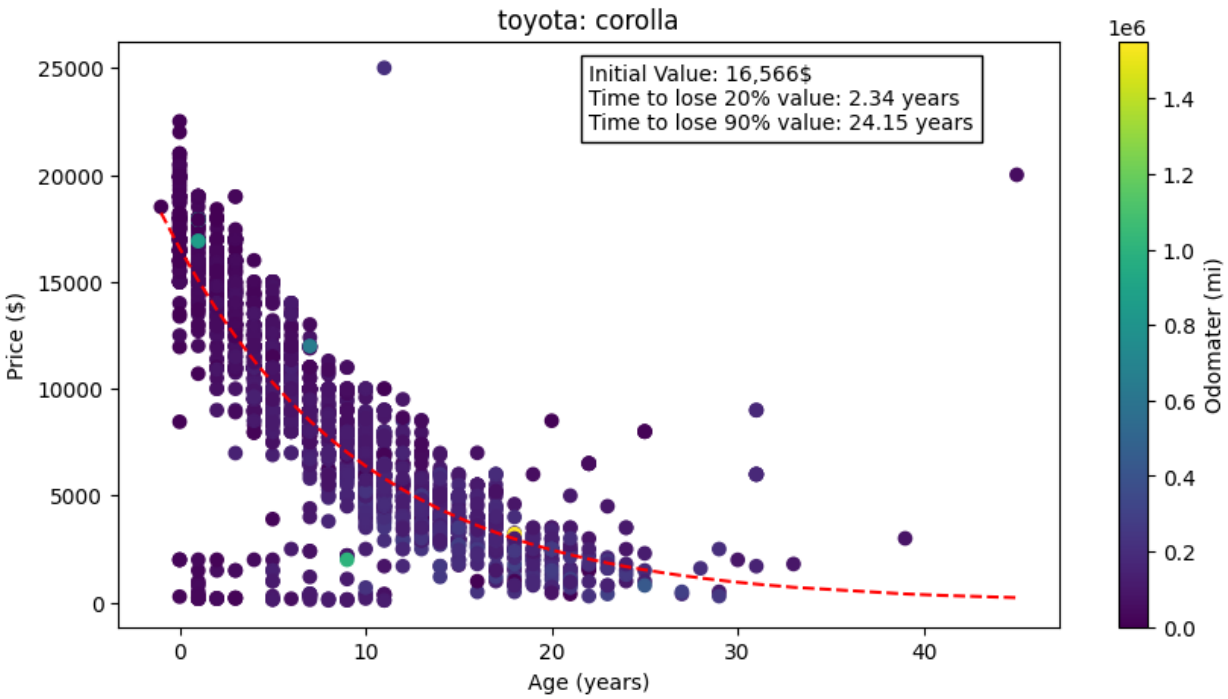

Pricing of the 25 Most Popular Sedans
Price ($)
0
20000
40000
60000
80000
mercedes-benz: c-class
dodge: charger
chrysler: 300
kia: optima
volkswagen: passat
audi: a4
ford: fusion
hyundai: elantra
chevrolet: malibu
nissan: maxima
ford: fusion se
toyota: corolla
chevrolet: cruze
chrysler: 200
nissan: altima
nissan: sentra
hyundai: sonata
toyota: camry
volkswagen: jetta
honda: civic
honda: accord
ford: focus
ford: taurus
toyota: camry le
chevrolet: impala
Make and Model


Pricing of the 25 Most Popular Trucks
Price ($)
0
20000
40000
60000
80000
ford: super duty f-350 srw
ford: super duty f-650 straight frame
ram: 3500
chevrolet: silverado 3500hd
ram: 2500
ford: super duty f-250
ford: f-350
gmc: sierra 2500hd
chevrolet: silverado 2500hd
ford: f-250
chevrolet: silverado 1500
gmc: sierra 1500
ram: 1500
ford: f-150
gmc: sierra
toyota: tacoma
toyota: tundra
chevrolet: silverado
chevrolet: colorado
ford: f250
ford: f250 super duty
ford: f150
nissan: frontier
ford: ranger
dodge: dakota
Make and Model




```
from scipy.optimize import curve_fit
#fit an exponential to the toyota corolla data to determine how well it holds its value

def func(x, a, b):
    return a * np.exp(-b*x)#exponential function we will use to fit

def plot_depr(data, func, model):
    #get model data and filter out cars older than 50 years
    df = data[(data['make_model']==model) & (data['age']<=50)].sort_values(by='age')
    xdata = df['age']
    ydata = df['price']

    #fit to the data
    popt, _ = curve_fit(func, xdata, ydata, p0=[4e4, 0.1])#fit the exponential to the data
    init = popt[0]#intiial value (age=0) according to the curve fit
    depr20 = -np.log(0.80)/popt[1]#time to depreciate 20% according to the curve fit
    depr90 = -np.log(0.10)/popt[1]#time to depreciate 90% according to the curve fit

    fig, ax = plot.subplots(figsize=(10,5))
    carplt = ax.scatter(xdata, ydata, c=df['odometer'], cmap='viridis')#scatter plot of age
vs price, colored by odometer
    plot.plot(xdata, func(xdata, *popt), 'r--')#plot the fitted curve

    plot.text(0.5, 0.85,
              'Initial Value: {:,.0f}$\n'
              'Time to lose 20% value: {:.2f} years\n'
              'Time to lose 90% value: {:.2f} years'.format(init, depr20, depr90),
              transform = ax.transAxes,
             bbox=dict(facecolor='white', edgecolor='black'))
```

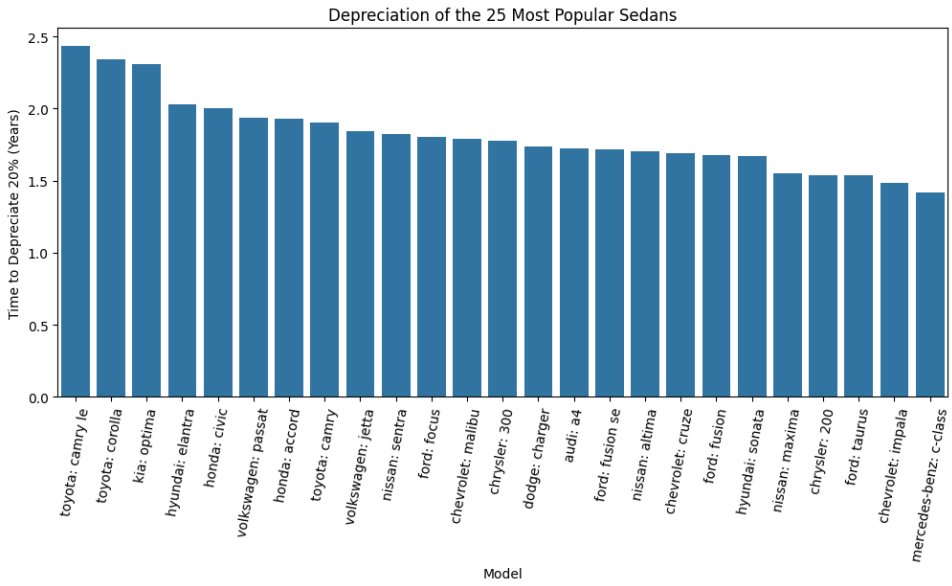
```
    cbar=plot.colorbar(carplt)
    cbar.set_label('Odomater (mi)')
    plot.xlabel('Age (years)')
    plot.ylabel('Price ($)')
    plot.title(model)
    plot.show()
```

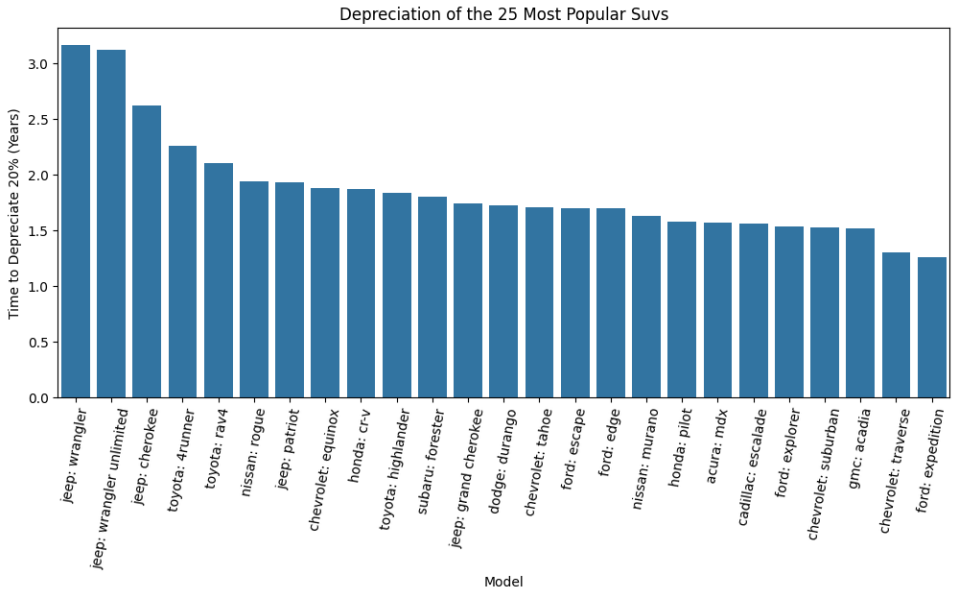


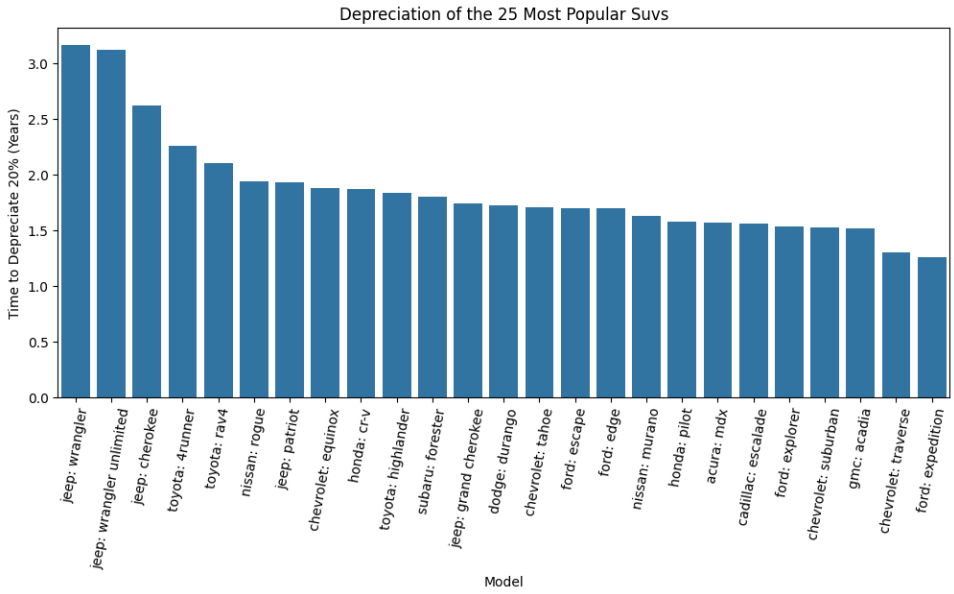
When purchasing a vehicle, considerations extend beyond the immediate cost as well as future resale values. In this context, a methodological approach involves fitting a decaying exponential function to each specific vehicle model, facilitating a quantitative assessment of its depreciation dynamics over time. This analytical framework enables the determination of whether a given model maintains its value over the course of ownership.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import curve_fit
# Define the function for curve fitting
def func(x, a, b):
    return a * np.exp(-b * x)

# Run curve fits for the 25 most popular sedans, SUVs, and trucks and plot
for kind in ['sedan', 'SUV', 'truck']:
    com = cars[cars['type'] == kind].make_model.value_counts().index[:25]

    depr_df = pd.DataFrame(columns=['Model', 'val0', 'depr20', 'depr90'])
```

```
    for name in com:
        df = cars[(cars['make_model'] == name) & (cars['age'] <= 50)].sort_values(by='age')
        xdata = df['age']
        ydata = df['price']

        popt, pcov = curve_fit(func, xdata, ydata, p0=[4e4, 0.1])

        init = popt[0]
        depr20 = -np.log(0.80) / popt[1]
        depr90 = -np.log(0.10) / popt[1]

        # Use pd.DataFrame instead of DataFrame if there's a conflict with another variable
named pd
        depr_df = pd.concat([depr_df, pd.DataFrame({'Model': [name], 'val0': [init],
'depr20': [depr20], 'depr90': [depr90]})], ignore_index=True)

    depr_df = depr_df.sort_values(by='depr20', ascending=False)

    fig, ax = plt.subplots(figsize=(12, 5))
    sns.barplot(data=depr_df, x='Model', y='depr20', ax=ax)
    plt.title('Depreciation of the 25 Most Popular {}s'.format(kind.capitalize()))
    plt.ylabel('Time to Depreciate 20% (Years)')
    plt.xticks(rotation=80)
    plt.show()
```

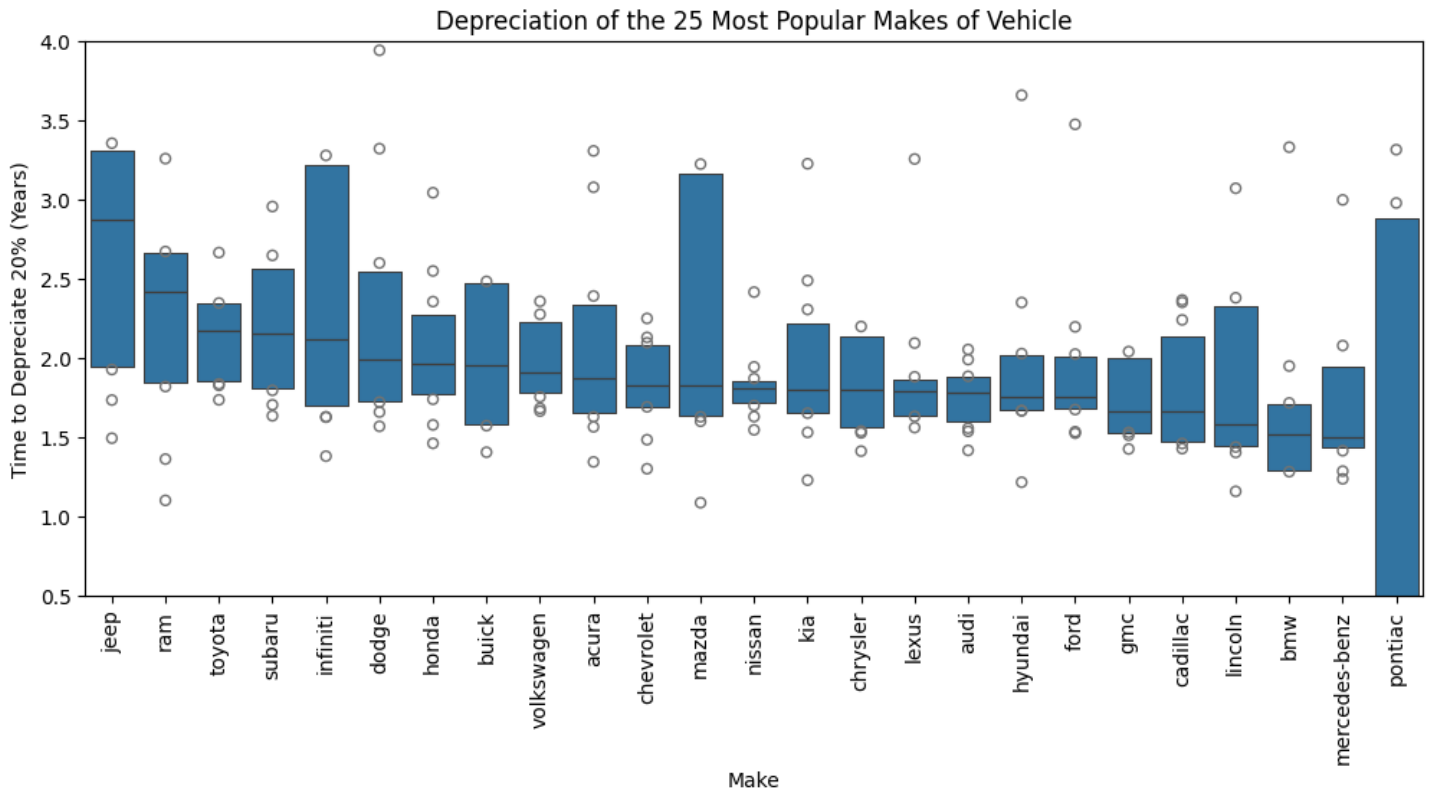







```
# Define the function for curve fitting
def func(x, a, b):
    return a * np.exp(-b * x)

# Boxplot of depreciation for different manufacturers
makes = cars['manufacturer'].value_counts()[:25].index  # 15 most popular manufacturers
depr_df = pd.DataFrame(columns=['Make', 'Model', 'val0', 'depr20', 'depr90'])  # this will
hold the depreciation data

for make in makes:
    com = cars[cars['manufacturer'] == make].model.value_counts()[0:10].index  # get the 10
most popular models by the manufacturer
    for name in com:
        df = cars[(cars['model'] == name) & (cars['age'] < 50)].sort_values(by='age')  # get
data for the model for ages under 50
        xdata = df['age']
        ydata = df['price']
        popt, pcov = curve_fit(func, xdata, ydata, p0=[4e4, 0.1])  # fit to the data
```

```
        init = popt[0]  # initial value
        depr20 = -np.log(0.80) / popt[1]  # time to depreciate 20%
        depr90 = -np.log(0.10) / popt[1]  # time to depreciate 90%

        # Use pd.DataFrame instead of DataFrame if there's a conflict with another variable
named pd
        depr_df = pd.concat([depr_df, pd.DataFrame({'Make': [make], 'Model': [name], 'val0':
[init], 'depr20': [depr20], 'depr90': [depr90]})], ignore_index=True)

# order the data in terms of decreasing median depreciation time
order = depr_df.groupby('Make')['depr20'].median().sort_values(ascending=False).index

fig, ax = plt.subplots(figsize=(12, 5))
sns.boxenplot(data=depr_df, x='Make', y='depr20', order=order)
plt.title('Depreciation of the 25 Most Popular Makes of Vehicle')
plt.ylabel('Time to Depreciate 20% (Years)')
plt.ylim(0.5, 4)
plt.xticks(rotation=90)
plt.show()
```

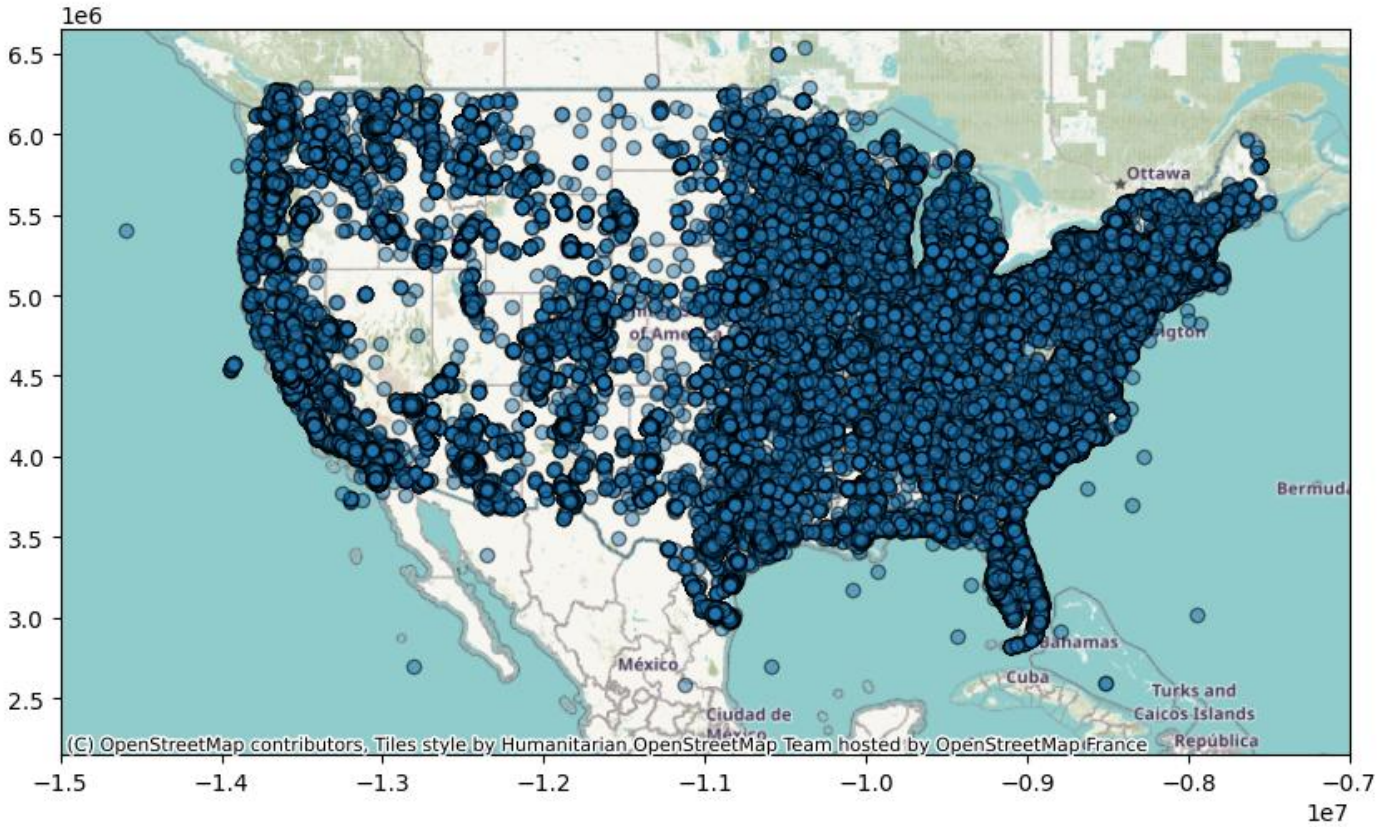


In the United States, empirical data indicates that Jeeps and Rams exhibit a comparatively robust retention of value, suggesting a sustained desirability for larger vehicles among consumers. Notably, certain older Jeep models have garnered heightened demand, leading to instances where their market value may appreciate over time. Concurrently, Toyota vehicles also demonstrate notable resilience in retaining value, with a median depreciation period of approximately 2.2 years required for a vehicle to undergo a 20% depreciation.

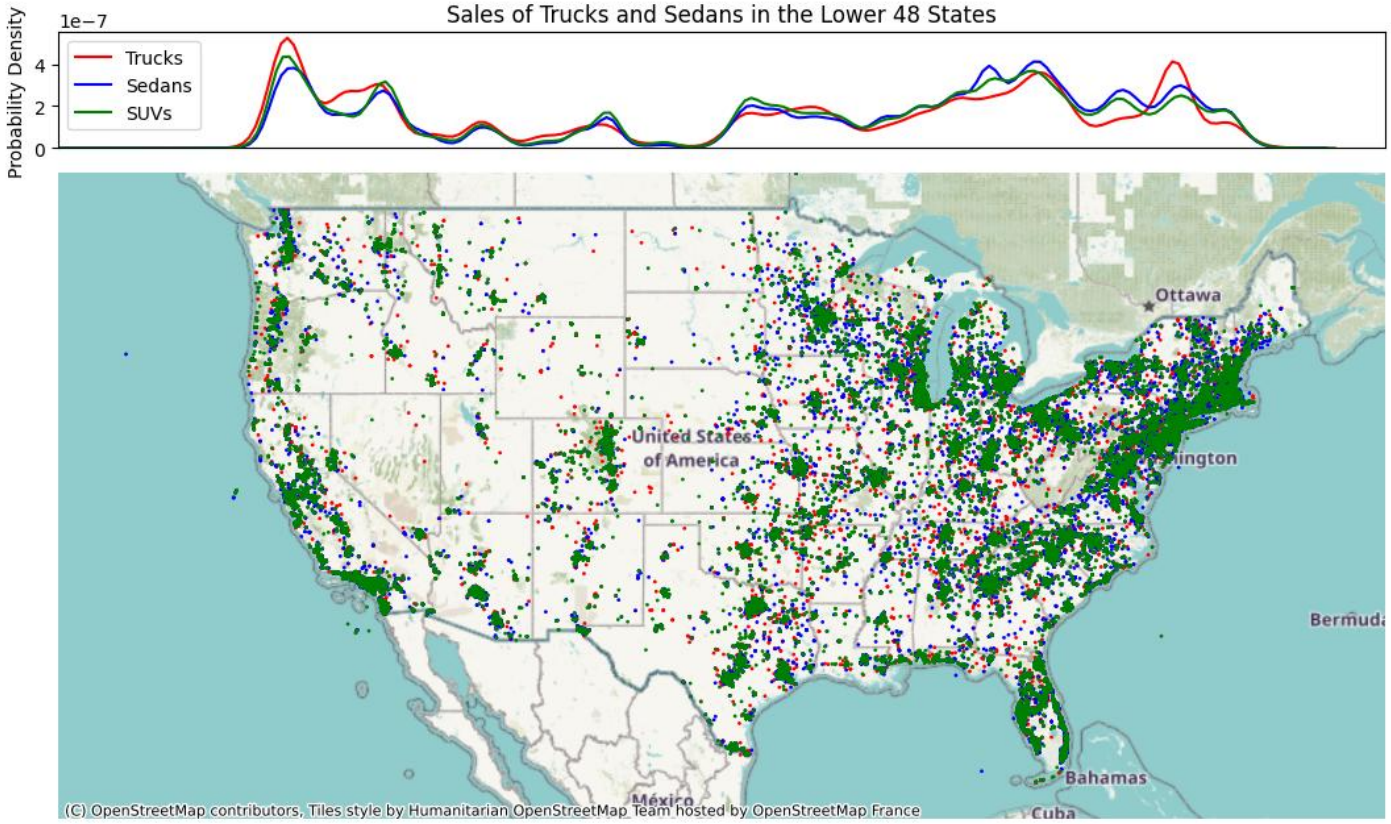
```
pip install geopandas
```

```
pip install contextily
```

```
#map distribution of vehichles
import geopandas as gpd
import contextily as ctx
gdf = gpd.GeoDataFrame(#convert the data to a geodataframe so it can be plotted on a map
    cars, geometry=gpd.points_from_xy(cars.long, cars.lat))
#remove data outside the geographic area of interest
gdf = gdf[(22<gdf.lat) & (gdf.lat<65) & (-144<gdf.long) & (gdf.long<-56)]
#tell geopandas what the coordinate system of our data is
gdf = gdf.set_crs(epsg=4326)#this is the latitude/longitude system the scraped data was in
gdf = gdf.to_crs(epsg=3857)#this is the coordinate system that the imported map is in

#plot the geographic area where data was collected and the points of the vehicles
ax = gdf.plot(figsize=(10, 10), alpha=0.5, edgecolor='k')
ctx.add_basemap(ax, zoom=4)
plt.xlim(-1.5e7,-0.7e7)
plt.ylim(2.15e6,6.65e6)
plt.show()
```



```
#plot the distribution of sedans, trucks, and suvs in the area around Vancouver
import geoplot
extent = (-1.5e7,-0.7e7, 2.6e6,6.5e6) #x and y limits of where we will plot

#look at sales of trucks, sedans, and SUVS
gtrucks = gdf[(gdf['type']=='truck')]
gsedans = gdf[(gdf['type']=='sedan')]
gsuvs = gdf[(gdf['type']=='SUV')]
#get the longitude data for each type of vehicle
trucks= pd.Series(gtrucks.geometry.x)
sedans= pd.Series(gsedans.geometry.x)
suvs= pd.Series(gsuvs.geometry.x)

fig, ax = plot.subplots(2, figsize=(13, 8), gridspec_kw={'height_ratios': [1, 6],
'hspace':0})
#plot the density distribution of each type by longitde
sns.kdeplot(data=trucks, ax=ax[0], clip=(extent[0], extent[1]), bw_adjust=0.35,
label='Trucks', color='red')
sns.kdeplot(data=sedans, ax=ax[0], clip=(extent[0], extent[1]), bw_adjust=0.35,
label='Sedans', color='blue')
sns.kdeplot(data=suvs, ax=ax[0], clip=(extent[0], extent[1]), bw_adjust=0.35, label='SUVs',
color='green')
ax[0].set_xlim(extent[0], extent[1])#set the x limits of the plot to be the same as our map
ax[0].set_ylabel('Probability Density')
ax[0].set_title('Sales of Trucks and Sedans in the Lower 48 States')
ax[0].set_xticks([])
ax[0].legend()

#Plot the location of each posting on a map
geoplot.pointplot(gtrucks, ax=ax[1], s=1, color='red')
geoplot.pointplot(gsedans, ax=ax[1], s=1, color='blue')
geoplot.pointplot(gsuvs, ax=ax[1], s=1, color='green')
ax[1].axis(extent)
ctx.add_basemap(ax[1], zoom=4)

plt.show()
```


Sales of Trucks and Sedans in the Lower 48 States
Probability Density
1e−7
4
2
0
Trucks
Sedans
SUVs
Ottawa
United States
of America
Bermuda
Bahamas
México
Cuba
(C) OpenStreetMap contributors, Tiles style by Humanitarian OpenStreetMap Team hosted by OpenStreetMap France

FIN 554 HW #3 Portfolio Optimization

We created a function to leverage QuantStats Library to analyze and visualize investment performance metrics. We are also using yfinance to download historical stock data for specified symbols and time periods. The function calculates daily returns from adjusted closing prices and then aggregates them into monthly returns. We also create a variable called PAST that looks at past monthly returns for a group of stocks at a specific timeframe.

```
!pip install quantstats
def MONTHLY_RETURNS(SYMBOLS , FROM , TO):
  from yfinance import download
  DATA = download(tickers=SYMBOLS , start=FROM, end=TO)
  ADJUSTED = DATA["Adj Close"]
  DAILY = ADJUSTED.pct_change()
  import quantstats as qs
  MONTHLY = qs.utils.aggregate_returns(returns=DAILY , period="M")
  from pandas import date_range
  MONTHLY.index = date_range(start=FROM, end=TO , freq="M")
  return MONTHLY
```

```
PAST = MONTHLY_RETURNS( SYMBOLS=["AAPL","BRK-A","DOGE-USD","GLD","SBUX","TSLA"] ,
                        FROM="2018-07-1" ,
                        TO="2023-07-1" )
```

Riskfolio-Lib is a Python library that provides tools for portfolio optimization and risk management.

```
!pip install Riskfolio-Lib
import riskfolio as OWL
```

```
import numpy as ELK

LYNX = ELK.identity( n=6 ) * (-1)

COUGAR = ELK.full( shape=[6,1] ,
                   fill_value=-0.3 )
```

```
DEER = OWL.Portfolio( returns=PAST ,
                      ainequality=LYNX ,
                      binequality=COUGAR ,
                      sht=True )
DEER.upperlng = 1.5

DEER.assets_stats()

MOOSE = DEER.optimization( rm="MV" ,

obj="Sharpe" ,
                   rf=0.0015 )
```

Python code demonstrates the process of portfolio optimization using the OWL library. By initializing a portfolio object, computing asset statistics, and optimizing the portfolio based on the Sharpe ratio, investors can make informed decisions to achieve optimal risk-adjusted returns. Understanding such code snippets is essential for financial analysts and investors looking to enhance their portfolio management strategies.

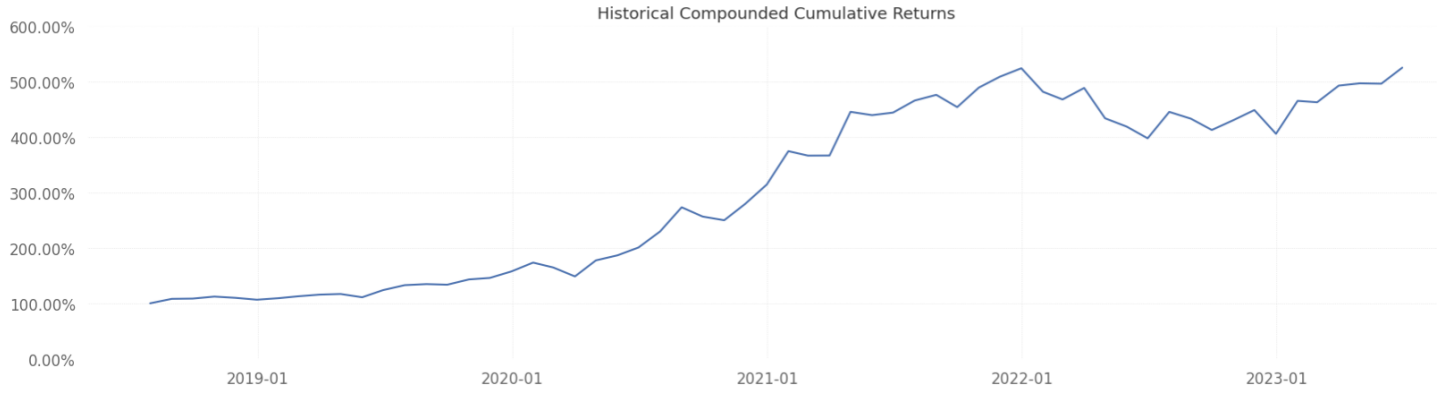
Generated Visuals for Portfolio optimization:

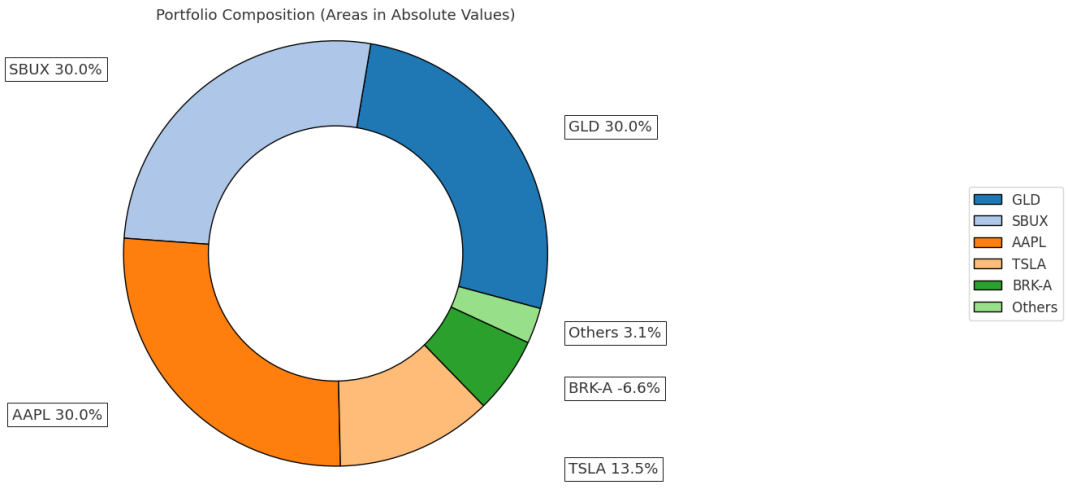
Python Code to Generate Reports and graphs.

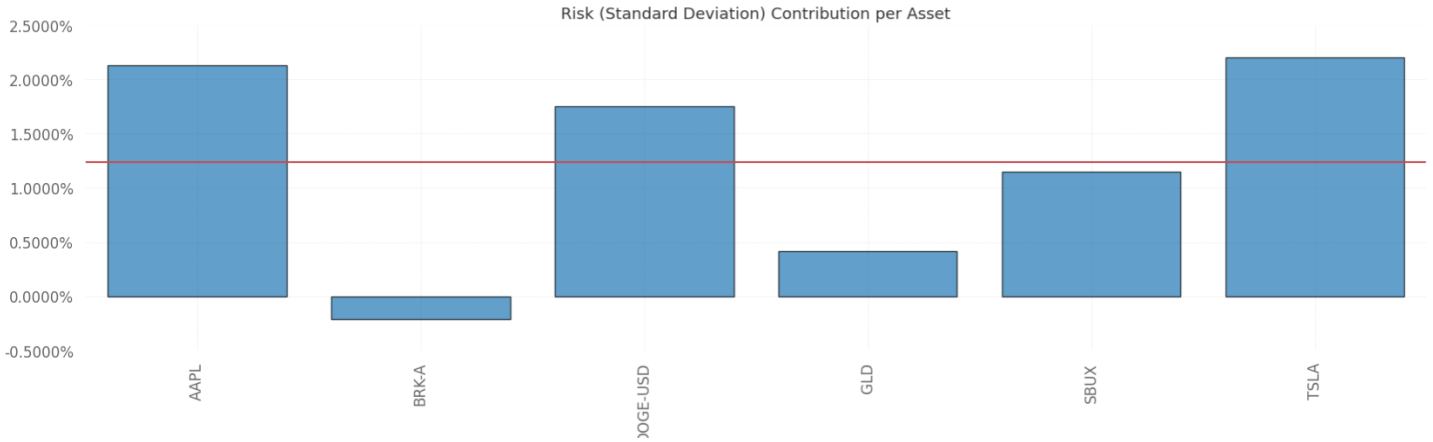
```
OWL.jupyter_report( w=MOOSE , returns=PAST ,t_factor=1 , days_per_year=365 , rf=0.0015 , width=18 )
```

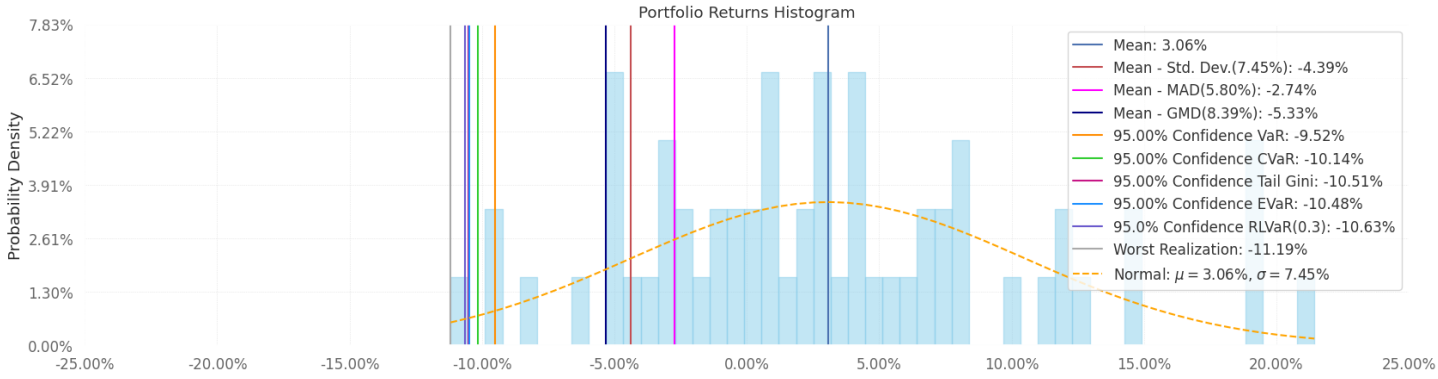
Riskfolio-Lib Report

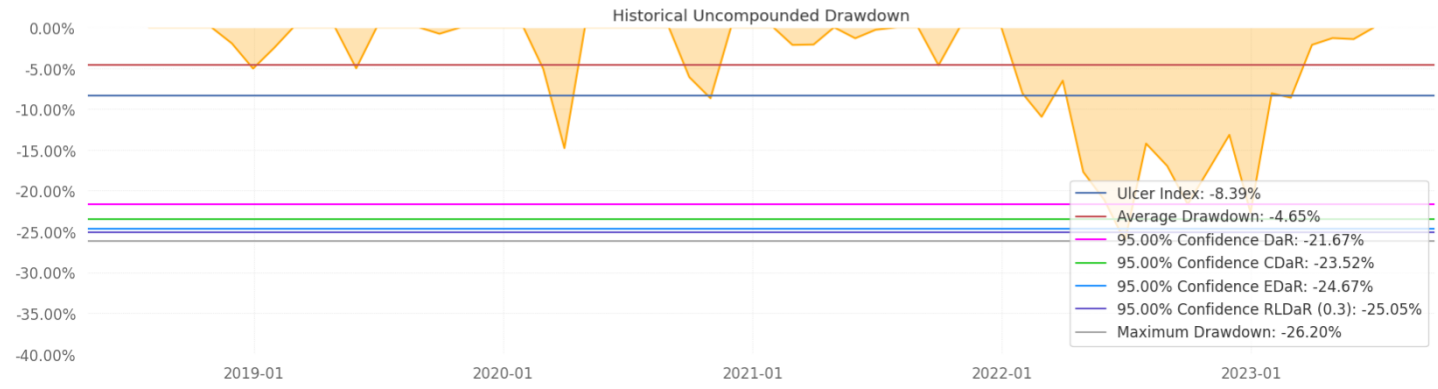
	Values	(Return - MAR)/Risk
Profitability and Other Inputs		
Mean Return (1)	3.0615%	
Compound Annual Growth Rate (CAGR)	40.0648%	
Minimum Acceptable Return (MAR) (1)	0.1500%	
Significance Level	5.0000%	
Risk Measures based on Returns		
Standard Deviation (2)	7.4465%	0.390986
Mean Absolute Deviation (MAD) (2)	5.7966%	0.502278
Semi Standard Deviation (2)	4.8366%	0.601969
First Lower Partial Moment (FLPM) (2)	1.5665%	1.858581
Second Lower Partial Moment (SLPM) (2)	3.2305%	0.901250
Value at Risk (VaR) (2)	9.5248%	0.305675
Conditional Value at Risk (CVaR) (2)	10.1435%	0.287028
Entropic Value at Risk (EVaR) (2)	10.5130%	0.276941
Tail Gini of Losses (TG) (2)	10.4777%	0.277873
Relativistic Value at Risk (RLVaR) (2)	10.6334%	0.273806
Worst Realization (2)	11.1874%	0.260248
Skewness	0.51307	
Kurtosis	0.03202	
Risk Measures based on Drawdowns (3)		
Ulcer Index (UCI)	8.3931%	0.346889
Average Drawdown (ADD)	4.6486%	0.626313
Drawdown at Risk (DaR)	21.6711%	0.134349
Conditional Drawdown at Risk (CDaR)	23.5201%	0.123787
Entropic Drawdown at Risk (EDaR)	24.6739%	0.117998
Relativistic Drawdown at Risk (RLDaR)	25.0489%	0.116232
Max Drawdown (MDD)	26.1981%	0.111133






Risk (Standard Deviation) Contribution per Asset
2.5000%
2.0000%
1.5000%
1.0000%
0.5000%
0.0000%
-0.5000%
AAPL
BRK-A
)OGE-USD
GLD
SBUX
TSLA


Portfolio Returns Histogram
Probability Density
7.83%
6.52%
5.22%
3.91%
2.61%
1.30%
0.00%
-25.00%
-20.00%
-15.00%
-10.00%
-5.00%
0.00%
5.00%
10.00%
15.00%
20.00%
25.00%
Mean: 3.06%
Mean - Std. Dev.(7.45%): -4.39%
Mean - MAD(5.80%): -2.74%
Mean - GMD(8.39%): -5.33%
95.00% Confidence VaR: -9.52%
95.00% Confidence CVaR: -10.14%
95.00% Confidence Tail Gini: -10.51%
95.00% Confidence EVaR: -10.48%
95.0% Confidence RLVaR(0.3): -10.63%
Worst Realization: -11.19%
Normal: $\mu = 3.06\%$, $\sigma = 7.45\%$


Historical Uncompounded Drawdown
0.00%
-5.00%
-10.00%
-15.00%
-20.00%
-25.00%
-30.00%
-35.00%
-40.00%
2019-01
2020-01
2021-01
2022-01
2023-01
Ulcer Index: -8.39%
Average Drawdown: -4.65%
95.00% Confidence DaR: -21.67%
95.00% Confidence CDaR: -23.52%
95.00% Confidence EDaR: -24.67%
95.00% Confidence RLDaR (0.3): -25.05%
Maximum Drawdown: -26.20%

Optimized Portfolio recommendation

```
DEER = OWL.Portfolio( returns=PAST ,
                      ainequality=LYNX ,
                      binequality=COUGAR ,
                      sht=True )
DEER.upperlng = 1.5

DEER.assets_stats()

MOOSE = DEER.optimization( rm="MV" ,

obj="Sharpe" ,
                   rf=0.0015 )

round(MOOSE, 4)
```

Python code showcases the process of portfolio optimization using the OWL library. By setting parameters, calculating statistics, and optimizing the portfolio, the code aims to find the optimal asset mix based on the mean-variance approach. Understanding and implementing such optimization techniques are essential for effective portfolio management in the financial domain. Below is a table showing the recommended stock weights to optimize a portfolio and maximize returns based on the given stocks. Our recommendation is as follows:

1. The following stocks should each make up 30% of the portfolio: Apple (AAAPL), Gilead (GLD), and Starbucks (SBUX).
2. Remove Berkshire Hathaway Class A Stock, 6.93% of the portfolio should Tesla (TSLA) stock.
3. The remaining 3.07% of the portfolio should consist of Dogecoin (DOGE-USD).

	weights
AAPL	0.3000
BRK-A	-0.0658
DOGE-USD	0.0307
GLD	0.3000
SBUX	0.3000
TSLA	0.1351