# **5. CODE**

# **1. DATASET - TITANIC**

# coding: utf-8

'''

First some basic questions:

1.) Who were the passengers on the Titanic? (Ages,Gender,Class,..etc)

2.) What deck were the passengers on and how does that relate to their class?

3.) Where did the passengers come from?

4.) Who was alone and who was with family?

Then we'll dig deeper, with a broader question:

5.) What factors helped someone survive the sinking?

'''

# In[1]:

import pandas as pd

from pandas import Series , DataFrame

# In[2]:

titanic\_df = pd.read\_csv('train.csv')

# In[3]:

titanic\_df.head()

# In[4]:

titanic\_df.info()

# In[5]:

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# In[6]:

#ANSWERING QUESTION 1

sns.factorplot(x='Sex', data=titanic\_df, kind='count')

# In[7]:

#hue defines each subtype in the given column with different color

#To distinguish a certain group using differnet color

sns.factorplot('Sex', data=titanic\_df, kind='count', hue='Pclass')

# In[8]:

sns.factorplot('Pclass', data=titanic\_df, kind='count', hue='Sex')

# In[9]:

#Assuming child age will be below 16

def mfc(passenger):

age, sex = passenger

if age < 16:

return 'child'

else:

return sex

# In[10]:

#apply a function to dataset by creating a new column named=person

titanic\_df['person'] = titanic\_df[['Age','Sex']].apply(mfc, axis=1)

titanic\_df.head()

# In[11]:

titanic\_df[0:10]

# In[12]:

sns.factorplot('Pclass' , data=titanic\_df, kind='count', hue='person')

# In[13]:

titanic\_df['Age'].hist(bins=70)

# In[14]:

titanic\_df['Age'].mean()

# In[15]:

titanic\_df['person'].value\_counts()

# In[16]:

#Multiple types of graphs in same Grid

fig = sns.FacetGrid(titanic\_df, hue='Sex', aspect=4)

fig.map(sns.kdeplot, 'Age', shade= True)

oldest = titanic\_df['Age'].max()

fig.set(xlim= (0, oldest))

fig.add\_legend()

# In[17]:

#Multiple types of graphs in same Grid

fig = sns.FacetGrid(titanic\_df, hue='person', aspect=4)

fig.map(sns.kdeplot, 'Age', shade= True)

oldest = titanic\_df['Age'].max()

fig.set(xlim= (0, oldest))

fig.add\_legend()

# In[18]:

#Multiple types of graphs in same Grid

fig = sns.FacetGrid(titanic\_df, hue='Pclass', aspect=4)

fig.map(sns.kdeplot, 'Age', shade= True)

oldest = titanic\_df['Age'].max()

fig.set(xlim= (0, oldest))

fig.add\_legend()

# In[19]:

#ANSWERING QUESTION 2

# In[20]:

titanic\_df.head()

# In[21]:

#removing null vlaues, creating new column

deck = titanic\_df['Cabin'].dropna()

# In[22]:

deck.head()

# In[23]:

#Grabbing deck character (First letter) , defining different cabins.

levels = []

for level in deck:

levels.append(level[0])

cabin\_df = DataFrame(levels)

#palette defining the colour of the plot shaded in winter format

#The below link defines different color shades in 'matplotlib'

#https://matplotlib.org/examples/color/colormaps\_reference.html

cabin\_df.columns = ['Cabin']

sns.factorplot('Cabin', data=cabin\_df.sort\_values('Cabin'), palette='winter\_d',

kind='count')

# In[24]:

cabin\_df = cabin\_df[cabin\_df.Cabin != 'T']

sns.factorplot('Cabin', data=cabin\_df.sort\_values('Cabin'), palette='summer',

kind='count')

# In[25]:

titanic\_df.head()

# In[26]:

#ANSWERING QUESTION 3

# In[27]:

# Three places from where the people came form :

# C = Cherbourg, Q = Queenstown, S = Southampton

sns.factorplot('Embarked', data=titanic\_df, hue='Pclass',

kind='count',order=['C', 'Q', 'S'])

# In[28]:

#ANSWERING QUESTION 4

# In[29]:

#As Fmaily 2 Categories are SibSp and Parch

# SibSp = Sibling Special and Parch = Parent Child

# If both Columns are empty that means that the passenger came alone

# Else with a family

# In[30]:

titanic\_df['Alone'] = titanic\_df.SibSp + titanic\_df.Parch

# In[31]:

# Anything other than 0 are with family

titanic\_df['Alone']

# In[32]:

#Warning that changing the original dataset

titanic\_df['Alone'].loc[titanic\_df['Alone'] > 0] = 'With Family'

titanic\_df['Alone'].loc[titanic\_df['Alone'] == 0] = 'Alone'

# In[33]:

titanic\_df.head()

# In[34]:

#Most people are alone than with familt

sns.factorplot('Alone', data=titanic\_df, palette='Blues',

kind='count')

# In[35]:

#ANSWERING QUESTION 5

# In[36]:

#Survived Column chaning to alphanumeric values

titanic\_df['Survivor'] = titanic\_df.Survived.map({0:'no' , 1:'yes'})

sns.factorplot('Survivor', data=titanic\_df, palette='Set2', kind='count')

# In[37]:

#What Factors affecting the Survivor Rate:

sns.factorplot('Pclass', 'Survived', data=titanic\_df)

# In[41]:

sns.factorplot('Pclass', 'Survived', data=titanic\_df, hue='person')

# In[42]:

#Linear Regression plot for the survivors

sns.lmplot('Age', 'Survived', data=titanic\_df)

# In[43]:

sns.lmplot('Age', 'Survived', data=titanic\_df,

hue='Pclass', palette='winter')

# In[44]:

generations = [10,20,40,60,80]

sns.lmplot('Age', 'Survived', hue='Pclass', data=titanic\_df

,palette='winter', x\_bins=generations )

# In[45]:

sns.lmplot('Age', 'Survived', hue='Sex', data=titanic\_df

,palette='winter', x\_bins=generations)

# 2. STOCK MARKET ANALYSIS

# coding: utf-8

'''

# Stock Market Analysis:

1.) What was the change in price of the stock over time?

2.) What was the daily return of the stock on average?

3.) What was the moving average of the various stocks?

4.) What was the correlation between different stocks' closing prices?

4.) What was the correlation between different stocks' daily returns?

5.) How much value do we put at risk by investing in a particular stock?

6.) How can we attempt to predict future stock behavior?

'''

# In[2]:

import pandas as pd

from pandas import Series, DataFrame

import numpy as np

# In[3]:

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('whitegrid')

# In[4]:

get\_ipython().run\_line\_magic('matplotlib', 'inline')

#NOTE

All the depricated data scaping methods are mentioned below

# In[16]:

'''

from pandas\_datareader import wb, data #DEPRICATED

depimport pandas\_datareader.data as pdr #DEPRICATED

from pandas.io.data import DataReader #DEPRICATED

'''

# In[17]:

from datetime import datetime

# In[22]:

end = datetime.now()

start = datetime(end.year - 1,end.month,end.day)

# In[23]:

#Getting DATA from Yahoo/Google

tech\_list = ['AAPL','GOOG','MSFT','AMZN']

# In[24]:

from yahoo\_fin import stock\_info as si

# In[25]:

# globals() used for setting all the string names like AAPL, GOOG as global variables

for stock in tech\_list:

globals()[stock] = si.get\_data(stock, start, end)

# In[26]:

# This global variable can get the data for this particular Stock

AAPL.head()

# In[27]:

MSFT.head()

# In[28]:

#Opening price, closing price, low, high and Split stock changes

GOOG.head()

# In[29]:

#describes all the statistical data for the stock data.

AAPL.describe()

# In[30]:

#taotal count and other info

AAPL.info()

# In[31]:

#Set Figure size not necessary

AAPL['adjclose'].plot(legend=True, figsize=(15,6))

# In[32]:

AAPL['volume'].plot(legend=True, figsize=(15,6))

# In[39]:

# For financial data Moving averages is one of the important factors for analyzing

# and predicting stock prices and other entities.

ma\_day = [10, 20, 50]

'''

for ma in ma\_day:

column\_name = "MA for %s days" %(str(ma)) # DEPRICATED

AAPL[column\_name] = pd.rolling\_mean(AAPL['adjclose'], ma) # DEPRICATED

'''

# In[40]:

#moving average and ints convergence of foloow

#This can give different treands in data

for ma in ma\_day:

column\_name = "MA for %s days" %(str(ma))

AAPL[column\_name] = AAPL['adjclose'].rolling(ma).mean()

# In[42]:

#Plotting the data for All categories

AAPL[['adjclose', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(subplots=False,

figsize=(15,6))

# In[68]:

#Creating Daily Return using Percentage change in values

AAPL['daily return'] = AAPL['adjclose'].pct\_change()

# In[44]:

#Risk Analysis using plotting daily returns (ups and downs)

AAPL['daily return'].plot(figsize=(15,6), legend=True, linestyle='--', marker='o')

# In[45]:

#Dive deep in seaborn #DEPRICATED normed

sns.distplot(AAPL['daily return'].dropna(), bins=100, color='purple')

# In[46]:

#Pandas histogram for smae implementation

AAPL['daily return'].hist(bins=100)

# In[47]:

all\_data = pd.DataFrame()

all\_data = [AAPL.adjclose , GOOG.adjclose, MSFT.adjclose, AMZN.adjclose]

total = pd.DataFrame(all\_data, index=['AAPL' , 'GOOG', 'MSFT', 'AMZN']).transpose()

all\_close = [AAPL.close , GOOG.close, MSFT.close, AMZN.close]

total\_close = pd.DataFrame(all\_close, index=['AAPL' , 'GOOG', 'MSFT', 'AMZN']).transpose()

# In[72]:

tech\_rets = total.pct\_change()

tech\_rets.head()

# In[48]:

total.head()

# In[73]:

#pearson value = 1 (Pearson product-moment correlation coefficient)

sns.jointplot('GOOG','GOOG', tech\_rets, kind='scatter', color='seagreen' )

# In[74]:

#pearson value = 0.66 (Pearson product-moment correlation coefficient)

sns.jointplot('GOOG', 'MSFT', tech\_rets, kind='scatter')

# In[52]:

total.head()

# In[75]:

sns.pairplot(tech\_rets.dropna(), kind='scatter')

# In[76]:

return\_fig = sns.PairGrid(tech\_rets.dropna())

return\_fig.map\_upper(plt.scatter, color='purple')

return\_fig.map\_lower(sns.kdeplot, cmap='cool\_d')

return\_fig.map\_diag(plt.hist)

# In[78]:

return\_fig = sns.PairGrid(total.dropna())

return\_fig.map\_upper(plt.scatter, color='purple')

return\_fig.map\_lower(sns.kdeplot, cmap='cool\_d')

return\_fig.map\_diag(plt.hist, bins=30)

# In[ ]:

#sns.corrplot(total.dropna(),annot=True) #DEPRICATED

#Seaborn Correlation plot is depricated, instead heatmaps are more robust to outliers

# and gives better picture than correlation matrices

# In[79]:

corr = tech\_rets.corr()

# @This masking function is taken from internet source

mask = np.zeros\_like(corr, dtype=np.bool)

# @This masking function is taken from internet source

mask[np.triu\_indices\_from(mask)] = True

sns.heatmap(corr, annot=True, mask=mask)

# # RISK ANALYSYS

#

# In[90]:

rets = tech\_rets.dropna()

area = np.pi\*20

plt.scatter(rets.mean(), rets.std(),alpha = 0.5,s =area)

# Set the x and y limits of the plot (optional, remove this if you don't see anything in your plot)

plt.ylim([0.01,0.030])

plt.xlim([-0.001,0.003])

#Set the plot axis titles

plt.xlabel('Expected returns')

plt.ylabel('Risk')

# @Content for annotation extracted from matplotlib online resourse

# http://matplotlib.org/users/annotations\_guide.html

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):

plt.annotate(

label,

xy = (x, y), xytext = (50, 50),

textcoords = 'offset points', ha = 'right', va = 'bottom',

arrowprops = dict(arrowstyle = '-', connectionstyle = 'arc3,rad=-0.2'))

# https://matplotlib.org/users/annotations\_guide.html

# Setting annotations

# In[66]:

#Value at risk

#BOOTSTRAP METHOD: quantile and percentile

sns.distplot(AAPL['daily return'].dropna(), bins=100, color='purple')

# In[92]:

# Worst case loss with 95% confidence you can loose 0.02795, it cannot exceed this amount

rets['AAPL'].quantile(0.05)

# # MONTE CARLO METHOD STOCK SIMULATION

Formula for calcullations : ΔS=S(μΔt+σϵ√Δt)

Here the S (stock price is multiplied by 2 terms)

1st term: Drift :

Drift is forward movement of values based on average daily return multiplied by the change of time.

2nd term: Shock :

This is a vertical movement of price (up or down) randomly.

For predicting stock everytime the stock price will Drift and experience a SHock either up or down. Multiple simulations of these will generate a histogram of lines that will predict stock price at a certain point in time.

# In[97]:

days= 365

dt = 1/days

#Average return

mu = rets.mean()['GOOG']

#Standard deviation on daily return

sigma = rets.std()['GOOG']

# In[98]:

def stock\_monte\_carlo(start\_price, days, mu, sigma):

price = np.zeros(days)

price[0] = start\_price

shock = np.zeros(days)

drift = np.zeros(days)

for x in range(1, days):

shock[x] = np.random.normal(loc=mu\*dt, scale=sigma\*np.sqrt(dt))

drift[x] = mu\*dt

price[x] = price[x-1] + (price[x-1] \* (drift[x] + shock[x]))

return price

# In[95]:

GOOG.head()

# In[99]:

start\_price = 1035.50

for run in range(100):

plt.plot(stock\_monte\_carlo(start\_price, days, mu, sigma))

plt.xlabel('Days')

plt.ylabel('Price')

plt.title('Monte Carlo Analysis for GOOGLE')

# In[100]:

runs = 10000

sims = np.zeros(runs)

for run in range(runs):

sims[run] = stock\_monte\_carlo(start\_price, days, mu, sigma)[days-1]

# In[102]:

#PLot a histoogram

q = np.percentile(sims, 1)

plt.hist(sims, bins=200)

plt.figtext(0.6, 0.8, s="Start price: $%.2f" %start\_price)

# Mean ending price

plt.figtext(0.6, 0.7, "Mean final price: $%.2f" % sims.mean())

# Variance of the price (within 99% confidence interval)

plt.figtext(0.6, 0.6, "VaR(0.99): $%.2f" % (start\_price - q,))

# Display 1% quantile

plt.figtext(0.15, 0.6, "q(0.99): $%.2f" % q)

# Plot a line at the 1% quantile result

plt.axvline(x=q, linewidth=4, color='r')

# Title

plt.title(u"Final price distribution for Google Stock after %s days" % days, weight='bold');

3. COMBINATION OF 2 PARTS:

A. ELECTION DATASET ANALYSIS

B. ELECTION DONOR DATASET

# coding: utf-8

'''

Analysis of Election Data using matlotlib, Seaborn

1.) Who was being polled and what was their party affiliation?

2.) Did the poll results favor Romney or Obama?

3.) How do undecided voters effect the poll?

4.) Can we account for the undecided voters?

5.) How did voter sentiment change over time?

6.) Can we see an effect in the polls from the debates?

'''

# In[7]:

import pandas as pd

from pandas import Series, DataFrame

import numpy as np

# In[8]:

import seaborn as sns

sns.set\_style('whitegrid')

import matplotlib.pyplot as plt

# In[9]:

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# In[10]:

import requests

# In[11]:

#from StringIO import StringIO #DEPRICATED

from io import StringIO

# In[12]:

# This is the url link for the poll data in csv form

url = "http://elections.huffingtonpost.com/pollster/2012-general-election-romney-vs-obama.csv"

# Getting information in text form and aviod error

source = requests.get(url).text

poll\_data = StringIO(source)

# In[13]:

poll\_df = pd.read\_csv(poll\_data)

# In[14]:

poll\_df.head()

# In[15]:

poll\_df.info()

# In[16]:

sns.factorplot('Affiliation', data=poll\_df, kind='count')

# In[17]:

sns.factorplot('Affiliation', data=poll\_df, kind='count', hue='Population')

# In[18]:

avg = pd.DataFrame(poll\_df.mean())

avg.drop('Number of Observations', axis=0, inplace=True)

avg.drop('Question Text', axis=0, inplace=True)

avg.drop('Question Iteration', axis=0, inplace=True)

# In[19]:

avg.head()

# In[20]:

std = pd.DataFrame(poll\_df.std())

std.drop('Number of Observations', axis=0, inplace=True)

#avg.drop('Question Text', axis=0, inplace=True)

#avg.drop('Question Iteration', axis=0, inplace=True)

# In[21]:

std.head()

# In[22]:

avg.plot(yerr=std, kind='bar', legend=False)

# In[28]:

poll\_avg = pd.concat([avg, std], axis=1, sort=True)

poll\_avg.drop('Question Text', axis=0, inplace=True)

poll\_avg.drop('Question Iteration', axis=0, inplace=True)

poll\_avg

# In[29]:

poll\_avg.columns = ['Average', 'STD']

# In[30]:

poll\_avg

# In[31]:

#Time series analysis

poll\_df.head()

# In[34]:

#Scatter plot over the period of time, converging

poll\_df.plot(x='End Date', y=['Obama','Romney','Other','Undecided'], linestyle='', marker='o')

# In[35]:

from datetime import datetime

# In[37]:

poll\_df['difference'] = (poll\_df.Obama - poll\_df.Romney) / 100

poll\_df.head()

# In[38]:

poll\_df = poll\_df.groupby(['Start Date'], as\_index=False).mean()

poll\_df.head()

# In[48]:

poll\_df.plot('Start Date', 'difference', figsize=(15,6), marker='o', linestyle='-', legend=True)

# In[42]:

#Circling thruogh till it finds october 2012

row\_in = 0

xlim = []

for date in poll\_df['Start Date']:

if date[0:7] == '2012-10':

xlim.append(row\_in)

row\_in += 1

else:

row\_in += 1

print (min(xlim))

print (max(xlim))

# In[50]:

#Just analyzing October data

poll\_df.plot('Start Date', 'difference', figsize=(15,6), marker='o',

linestyle='-', xlim=(325,352))

#vertical line through the axis

#Certain Debates Occured on these dates and their effects on polling

#Polling results are not ideal

# Oct 3rd

plt.axvline(x=325+2, linewidth=4, color='grey')

# Oct 11th

plt.axvline(x=325+10, linewidth=4, color='grey')

# Oct 22nd

plt.axvline(x=325+21, linewidth=4, color='grey')

# # Donor Data Set

1.) How much was donated and what was the average donation?

2.) How did the donations differ between candidates?

3.) How did the donations differ between Democrats and Republicans?

4.) What were the demographics of the donors?

5.) Is there a pattern to donation amounts?

# In[51]:

donor\_df = pd.read\_csv('Election\_Donor\_Data.csv')

# In[52]:

donor\_df.head()

# In[53]:

#Million rows data set (Big Data)

donor\_df.info()

# In[54]:

type(donor\_df.contbr\_zip)

# In[60]:

donor\_df['contb\_receipt\_amt'].value\_counts()

# In[62]:

don\_mean = donor\_df['contb\_receipt\_amt'].mean()

don\_std = donor\_df['contb\_receipt\_amt'].std()

# In[63]:

print ('Donation was %.2f with std %.2f' %(don\_mean, don\_std))

# In[78]:

#Huge standard Deviation with respect to average

top\_donor = donor\_df['contb\_receipt\_amt'].copy()

top\_donor.sort\_values()

top\_donor.head()

# In[79]:

#Getting rid of Negatives (Refunds)

top\_donor = top\_donor[top\_donor > 0]

top\_donor.sort\_values()

top\_donor.value\_counts().head(10)

# In[82]:

common\_don = top\_donor[top\_donor < 2500]

common\_don.hist(bins=100, figsize=(15,6))

# In[84]:

#Seperating Donations by party, creating new party column

candidate = donor\_df.cand\_nm.unique()

candidate

# In[91]:

# Seperating Obama because others are republic candidates

# Fast and ditty way to do this

party\_map = {'Bachmann, Michelle': 'Republican',

'Cain, Herman': 'Republican',

'Gingrich, Newt': 'Republican',

'Huntsman, Jon': 'Republican',

'Johnson, Gary Earl': 'Republican',

'McCotter, Thaddeus G': 'Republican',

'Obama, Barack': 'Democrat',

'Paul, Ron': 'Republican',

'Pawlenty, Timothy': 'Republican',

'Perry, Rick': 'Republican',

"Roemer, Charles E. 'Buddy' III": 'Republican',

'Romney, Mitt': 'Republican',

'Santorum, Rick': 'Republican'}

# Now map the party with candidate

donor\_df['Party'] = donor\_df.cand\_nm.map(party\_map)

# In[92]:

# Slow

'''

for i in range(0,len(donor\_df)):

if donor\_df['cand\_nm'].iloc == 'Obama,Barack':

donor\_df['party'].iloc = 'Democrat'

else:

donor\_df['party'].iloc = 'Republican'

'''

# In[95]:

donor\_df = donor\_df[donor\_df.contb\_receipt\_amt > 0]

donor\_df.head(10)

# In[97]:

# Number of Daoaitons for each party

donor\_df.groupby('cand\_nm')['contb\_receipt\_amt'].count()

# In[98]:

# Total dollar amounts for each party

donor\_df.groupby('cand\_nm')['contb\_receipt\_amt'].sum()

# In[104]:

cnad\_amount = donor\_df.groupby('cand\_nm')['contb\_receipt\_amt'].sum()

i=0

for don in cnad\_amount:

print ('The Dandidate %s raised %.0f dollars' %(cnad\_amount.index[i], don) )

print ('\n')

i += 1

# In[108]:

# Party members donations

cnad\_amount.plot(kind='bar', figsize=(15,6))

# In[107]:

# SIngle candidate entry is less for Obama with respect otall republican party members

donor\_df.groupby('Party')['contb\_receipt\_amt'].sum().plot(kind='bar')

# In[110]:

# Donations and who they came from , A in occupations of the donors and create pivot

# and column to find for which party and find the asum of the contribution

# from the same occupation people

occupation = donor\_df.pivot\_table('contb\_receipt\_amt',

index='contbr\_occupation',

columns='Party',

aggfunc='sum')

# In[112]:

occupation

# In[113]:

# Number of Reported Occupation

occupation.shape

# In[114]:

# Cannot display all the contribution plot

# Contribution bigger than a million dollars

occupation = occupation[occupation.sum(1) > 1000000]

occupation.shape

# In[122]:

occupation.plot(kind='barh', figsize=(15,12), cmap='winter')

# In[124]:

#Combining same prfessions and drop wrong occupations

occupation.drop(['INFORMATION REQUESTED PER BEST EFFORTS',

'INFORMATION REQUESTED'], axis=0, inplace=True)

# In[125]:

occupation.shape

# In[127]:

occupation.loc['CEO'] = occupation.loc['CEO'] + occupation.loc['C.E.O.']

# In[129]:

occupation.drop('C.E.O.', inplace=True)

# In[130]:

occupation.plot(kind='barh',figsize=(15,12), cmap='winter' )