Assessment_Shivam

November 10, 2022

1 Python Assessment

Name: Shivam Panchal

1.1 Importing Data

[ ]:
```
import pandas as pd
```

[ ]:
```
df_employee = pd.read_csv("employee_data.csv")
print(f"Shape: {df_employee.shape}")
df_employee.head()
```

```
Shape: (1200, 10)
```

[ ]:

	AGENT_ID	AGENT_NAME	DATE_OF_JOINING	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	POSTAL_CODE	EMP_ROUTING_NUMBER	EMP_ACCT_NUMBER
0	AGENT00001	Ray Johns	1993-06-05	1402 Maggies Way	NaN	Waterbury Center	VT	5677	34584958	HKUN51252328472585
1	AGENT00002	Angelo Borjon	2005-12-27	414 Tanya Pass	NaN	Panama City	FL	32404	107363763	OPIS19290040088204
2	AGENT00003	Candy Spellman	2003-09-02	606 National Street	#306	Fayetteville	AR	72701	81744097	YSCJ67489688482590
3	AGENT00004	Mary Smith	2004-09-23	235 Hugh Thomas Drive	NaN	Panama City	FL	32404	67563771	ZANG21285355574581
4	AGENT00005	Mildred Diaz	2011-06-21	3426 Broadview Street	NaN	Montgomery	AL	36110	114951317	DZFS82244494451134

[ ]:
```
df_insurance = pd.read_csv("insurance_data.csv")
print(f"Shape: {df_insurance.shape}")
with pd.option_context("display.max_columns", 0):
```

```
    display(df_insurance.head())
```

```
Shape: (10000, 38)
```

	TXN_DATE_TIME	TRANSACTION_ID	…	AGENT_ID	VENDOR_ID
0	2020-06-01 00:00:00	TXN00000001	…	AGENT00413	VNDR00556
1	2020-06-01 00:00:00	TXN00000002	…	AGENT00769	VNDR00592
2	2020-06-01 00:00:00	TXN00000003	…	AGENT00883	VNDR00031
3	2020-06-01 00:00:00	TXN00000004	…	AGENT00278	VNDR00075
4	2020-06-01 00:00:00	TXN00000005	…	AGENT00636	VNDR00472

[5 rows x 38 columns]

[ ]:
```
df_insurance.columns.sort_values()
```

[ ]:
```
Index(['ACCT_NUMBER', 'ADDRESS_LINE1', 'ADDRESS_LINE2', 'AGE', 'AGENT_ID',
       'ANY_INJURY', 'AUTHORITY_CONTACTED', 'CITY', 'CLAIM_AMOUNT',
       'CLAIM_STATUS', 'CUSTOMER_EDUCATION_LEVEL', 'CUSTOMER_ID',
       'CUSTOMER_NAME', 'EMPLOYMENT_STATUS', 'HOUSE_TYPE', 'INCIDENT_CITY',
       'INCIDENT_HOUR_OF_THE_DAY', 'INCIDENT_SEVERITY', 'INCIDENT_STATE',
       'INSURANCE_TYPE', 'LOSS_DT', 'MARITAL_STATUS', 'NO_OF_FAMILY_MEMBERS',
       'POLICE_REPORT_AVAILABLE', 'POLICY_EFF_DT', 'POLICY_NUMBER',
       'POSTAL_CODE', 'PREMIUM_AMOUNT', 'REPORT_DT', 'RISK_SEGMENTATION',
       'ROUTING_NUMBER', 'SOCIAL_CLASS', 'SSN', 'STATE', 'TENURE',
       'TRANSACTION_ID', 'TXN_DATE_TIME', 'VENDOR_ID'],
      dtype='object')
```

[ ]:
```
df_vendor = pd.read_csv("vendor_data.csv")
print(f"Shape: {df_vendor.shape}")
df_vendor.head()
```

```
Shape: (600, 7)
```

[ ]:

	VENDOR_ID	VENDOR_NAME	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	POSTAL_CODE
0	VNDR00001	King, Proctor and Jones	2027 North Shannon Drive	#5	Fayetteville	AR	72703
1	VNDR00002	Garcia Ltd	5701 East Shirley Lane	NaN	Montgomery	AL	36117
2	VNDR00003	Cherry LLC	1217 Cottondale Road	NaN	Montgomery	AL	36109
3	VNDR00004	Mays-Benson	227 West Montgomery Cross Road	#736	Savannah	GA	31406
4	VNDR00005	Wilson PLC	23 North Hill Street	NaN	Nashville	TN	37210

1.2 Task 1

- Merge the 3 dataset and create 1 view of data.
- You can merge `insurance_data.csv` and `employee_data.csv` on `AGENT_ID`
- You can merge `insurance_data.csv` and `vendor_data.csv` on `VENDOR_ID`
- **Note:** Use left Outer join as not all claims require Vendor

[ ]:
```
df_merged = df_insurance.merge(df_employee, on="AGENT_ID",␣
 ↪suffixes=["_INSURANCE", "_AGENT"]).merge(
    df_vendor, on="VENDOR_ID", how="left"
)
print(f"Shape: {df_merged.shape}")
with pd.option_context("display.max_columns", 0):
    display(df_merged.head())
```

```
Shape: (10000, 53)
```

	TXN_DATE_TIME	TRANSACTION_ID	…	STATE	POSTAL_CODE
0	2020-06-01 00:00:00	TXN00000001	…	FL	32405.0
1	2020-06-10 00:00:00	TXN00000258	…	NaN	NaN
2	2020-09-07 00:00:00	TXN00002428	…	NaN	NaN
3	2020-09-22 00:00:00	TXN00002811	…	NaN	NaN
4	2020-09-24 00:00:00	TXN00002861	…	AL	36104.0

```
[5 rows x 53 columns]
```

[ ]:
```
df_merged.columns.sort_values()
```

[ ]:
```
Index(['ACCT_NUMBER', 'ADDRESS_LINE1', 'ADDRESS_LINE1_AGENT',
       'ADDRESS_LINE1_INSURANCE', 'ADDRESS_LINE2', 'ADDRESS_LINE2_AGENT',
       'ADDRESS_LINE2_INSURANCE', 'AGE', 'AGENT_ID', 'AGENT_NAME',
       'ANY_INJURY', 'AUTHORITY_CONTACTED', 'CITY', 'CITY_AGENT',
       'CITY_INSURANCE', 'CLAIM_AMOUNT', 'CLAIM_STATUS',
       'CUSTOMER_EDUCATION_LEVEL', 'CUSTOMER_ID', 'CUSTOMER_NAME',
       'DATE_OF_JOINING', 'EMPLOYMENT_STATUS', 'EMP_ACCT_NUMBER',
       'EMP_ROUTING_NUMBER', 'HOUSE_TYPE', 'INCIDENT_CITY',
       'INCIDENT_HOUR_OF_THE_DAY', 'INCIDENT_SEVERITY', 'INCIDENT_STATE',
       'INSURANCE_TYPE', 'LOSS_DT', 'MARITAL_STATUS', 'NO_OF_FAMILY_MEMBERS',
       'POLICE_REPORT_AVAILABLE', 'POLICY_EFF_DT', 'POLICY_NUMBER',
       'POSTAL_CODE', 'POSTAL_CODE_AGENT', 'POSTAL_CODE_INSURANCE',
       'PREMIUM_AMOUNT', 'REPORT_DT', 'RISK_SEGMENTATION', 'ROUTING_NUMBER',
       'SOCIAL_CLASS', 'SSN', 'STATE', 'STATE_AGENT', 'STATE_INSURANCE',
       'TENURE', 'TRANSACTION_ID', 'TXN_DATE_TIME', 'VENDOR_ID',
       'VENDOR_NAME'],
      dtype='object')
```

1.3 Task 2

- Business Leader wants to find **Top 3 insurance Type** where we are getting `most insurance claims`?

[ ]:
```
# df_insurance["INSURANCE_TYPE"].value_counts()[:3]
df_insurance["INSURANCE_TYPE"].value_counts().nlargest(3)
```

[ ]:
```
Property    1692
Mobile      1692
Health      1690
Name: INSURANCE_TYPE, dtype: int64
```

[ ]:
```
df_insurance["CLAIM_STATUS"].unique()
```

[ ]:
```
array(['A', 'D'], dtype=object)
```

[ ]:
```
# Claim Status Accepted i.e. getting insurance claim
claim_condition = df_insurance["CLAIM_STATUS"] == "A"
# df_insurance[claim_condition]["INSURANCE_TYPE"].value_counts()[:3]
df_insurance[claim_condition]["INSURANCE_TYPE"].value_counts().nlargest(3)
```

[ ]:
```
Property    1608
Mobile      1608
Health      1605
Name: INSURANCE_TYPE, dtype: int64
```

Top 3 Insurance Type with `Accepted/Approved` status and `Over All` Insurance Claims

1. Property
2. Mobile
3. Health

[ ]:
```
df_insurance[["INSURANCE_TYPE", "CLAIM_STATUS"]].reset_index().groupby(
    ["INSURANCE_TYPE", "CLAIM_STATUS"]
).count().sort_values(by="index", ascending=False)
```

[ ]:

INSURANCE_TYPE	CLAIM_STATUS	index
Mobile	A	1608
Property	A	1608
Health	A	1605
Life	A	1605
Travel	A	1582
Motor	A	1489
Travel	D	88
Health	D	85
Motor	D	85
Mobile	D	84

```
Property        D                84
Life            D                77
```

1.4 Task 3

- Business Leader wants to find **Top 5 States** where we are getting most insurance claims for customer belonging to `HIGH(H)` risk segment?

[ ]:
```
df_insurance["RISK_SEGMENTATION"].unique()
```

[ ]:
```
array(['L', 'M', 'H'], dtype=object)
```

[ ]:
```
# High Risk Segmentation
high_risk_condition = df_insurance["RISK_SEGMENTATION"] == "H"  # &␣
↪(df_insurance["CLAIM_STATUS"] == "A")
df_insurance[high_risk_condition]["STATE"].value_counts().nlargest(5)
```

[ ]:
```
CA    148
AZ    105
FL    104
TN    100
AR     97
Name: STATE, dtype: int64
```

Top 5 States with `High` risk segmentation insurance

1. CA
2. AZ
3. FL
4. TN
5. AR

1.5 Task 4

- Business wants to create a new variable `"COLOCATION"` which will have following values > IF Customer State == Incident State == Agent Address State
  > THEN 1
  > ELSE 0
- Find the mean of this new column

[ ]:
```
df_ins_emp = df_merged.copy()
df_ins_emp.columns.sort_values()
```

[ ]:
```
Index(['ACCT_NUMBER', 'ADDRESS_LINE1', 'ADDRESS_LINE1_AGENT',
       'ADDRESS_LINE1_INSURANCE', 'ADDRESS_LINE2', 'ADDRESS_LINE2_AGENT',
       'ADDRESS_LINE2_INSURANCE', 'AGE', 'AGENT_ID', 'AGENT_NAME',
       'ANY_INJURY', 'AUTHORITY_CONTACTED', 'CITY', 'CITY_AGENT',
       'CITY_INSURANCE', 'CLAIM_AMOUNT', 'CLAIM_STATUS',
       'CUSTOMER_EDUCATION_LEVEL', 'CUSTOMER_ID', 'CUSTOMER_NAME',
```

```
       'DATE_OF_JOINING', 'EMPLOYMENT_STATUS', 'EMP_ACCT_NUMBER',
       'EMP_ROUTING_NUMBER', 'HOUSE_TYPE', 'INCIDENT_CITY',
       'INCIDENT_HOUR_OF_THE_DAY', 'INCIDENT_SEVERITY', 'INCIDENT_STATE',
       'INSURANCE_TYPE', 'LOSS_DT', 'MARITAL_STATUS', 'NO_OF_FAMILY_MEMBERS',
       'POLICE_REPORT_AVAILABLE', 'POLICY_EFF_DT', 'POLICY_NUMBER',
       'POSTAL_CODE', 'POSTAL_CODE_AGENT', 'POSTAL_CODE_INSURANCE',
       'PREMIUM_AMOUNT', 'REPORT_DT', 'RISK_SEGMENTATION', 'ROUTING_NUMBER',
       'SOCIAL_CLASS', 'SSN', 'STATE', 'STATE_AGENT', 'STATE_INSURANCE',
       'TENURE', 'TRANSACTION_ID', 'TXN_DATE_TIME', 'VENDOR_ID',
       'VENDOR_NAME'],
      dtype='object')
```

[ ]:
```
# other option: np.where
df_ins_emp["COLOCATION"] = 0  # setting all as 0
colocation_condition = (df_ins_emp["STATE_INSURANCE"] ==␣
 ↪df_ins_emp["INCIDENT_STATE"]) & (
    df_ins_emp["INCIDENT_STATE"] == df_ins_emp["STATE_AGENT"]
)
df_ins_emp.loc[colocation_condition, "COLOCATION"] = 1  # setting 1 if␣
 ↪condition is met
# df_ins_emp["COLOCATION"] = df_ins_emp["COLOCATION"].
 ↪mask(colocation_condition, 1).sample(10, random_state=6)
print(f'Mean:{df_ins_emp["COLOCATION"].mean()}')
with pd.option_context("display.max_columns", 0):
    display(df_ins_emp[["STATE_INSURANCE", "INCIDENT_STATE", "STATE_AGENT",␣
 ↪"COLOCATION"]].sample(10, random_state=6))
```

```
Mean:0.0044
```

	STATE_INSURANCE	INCIDENT_STATE	STATE_AGENT	COLOCATION
739	KY	CA	MA	0
9426	VT	DC	MA	0
8200	GA	CO	CA	0
5880	VT	KY	OK	0
9114	TN	MD	CA	0
1060	CT	CT	CT	1
1841	TN	AZ	VT	0
5105	AR	GA	OK	0
2109	DC	MA	GA	0
4597	TN	KY	GA	0

[ ]:
```
df_ins_emp["COLOCATION_2"] = colocation_condition.astype(int)  # Boolean are␣
 ↪Sub-group of numeric data-type
print(f'Mean:{df_ins_emp["COLOCATION_2"].mean()}')
with pd.option_context("display.max_columns", 0):
    display(
```

```
        df_ins_emp[["STATE_INSURANCE", "INCIDENT_STATE", "STATE_AGENT",␣
↪"COLOCATION", "COLOCATION_2"]].sample(
            10, random_state=6
        )
    )
```

Mean:0.0044

	STATE_INSURANCE	INCIDENT_STATE	STATE_AGENT	COLOCATION	COLOCATION_2
739	KY	CA	MA	0	0
9426	VT	DC	MA	0	0
8200	GA	CO	CA	0	0
5880	VT	KY	OK	0	0
9114	TN	MD	CA	0	0
1060	CT	CT	CT	1	1
1841	TN	AZ	VT	0	0
5105	AR	GA	OK	0	0
2109	DC	MA	GA	0	0
4597	TN	KY	GA	0	0

Mean of Colocation: $0.0044 = 0.44\%$

1.6 Task 5

- Data entry error was detected in the data and you are required to correct it. If for any claim transaction `"AUTHORITY_CONTACTED"` is NOT `"Police"` and `POLICE_AVAILABLE == 1` Then Update `"AUTHORITY_CONTACTED"` to `"Police"`.

[ ]:
```
df_ins_corrected = df_insurance.copy()  # copy of data to avoid mutation
df_ins_corrected[["AUTHORITY_CONTACTED", "POLICE_REPORT_AVAILABLE"]].
↪value_counts()
```

[ ]:
```
AUTHORITY_CONTACTED  POLICE_REPORT_AVAILABLE
Ambulance            1                        2821
Police               1                        2058
None                 1                        1367
Ambulance            0                        1261
Police               0                         924
Other                1                         682
None                 0                         578
Other                0                         309
dtype: int64
```

[ ]:
```
contacted_condition = (df_ins_corrected["AUTHORITY_CONTACTED"] != "Police") & (
    df_ins_corrected["POLICE_REPORT_AVAILABLE"]
)
df_ins_corrected.loc[contacted_condition, "AUTHORITY_CONTACTED"] = "Police"
df_ins_corrected[["AUTHORITY_CONTACTED", "POLICE_REPORT_AVAILABLE"]].
↪value_counts()
```

[ ]:
```
AUTHORITY_CONTACTED  POLICE_REPORT_AVAILABLE
Police               1                        6928
Ambulance            0                        1261
Police               0                         924
None                 0                         578
Other                0                         309
dtype: int64
```

1.7 Task 6

- Business wants to check the Claim Amount for deviation for each transaction, they would like you to calculate as follow > CLAIM_DEVIATION = AVG_CLAIM_AMOUNT_FOR_LAST_30DAYS (same insurance type) / CURRENT_CLAIM_AMOUNT
- If the `value < 0.5` THEN `CLAIM_DEVIATION = 1` ELSE `0`
- If there is less than 30 days of transaction history THEN -1
- **Note:** LAST_30DAYS does not include current day

Static date approach in Fiddle > Task 6

[ ]:
```
df_claim_deviation = df_insurance[["TXN_DATE_TIME", "INSURANCE_TYPE",␣
↪"CLAIM_AMOUNT"]].copy()
df_claim_deviation["TXN_DATE_TIME"] = pd.
↪to_datetime(df_claim_deviation["TXN_DATE_TIME"])
df_claim_deviation.describe(datetime_is_numeric=True)
```

[ ]:
```
                     TXN_DATE_TIME   CLAIM_AMOUNT
count                        10000   10000.000000
mean   2020-12-16 10:45:41.760000   16563.830000
min            2020-06-01 00:00:00     100.000000
25%            2020-09-10 00:00:00    2000.000000
50%            2020-12-18 00:00:00    7000.000000
75%            2021-03-24 00:00:00   21000.000000
max            2021-06-30 00:00:00  100000.000000
std                            NaN   22037.489735
```

[ ]:
```
def claim_deviations(row):
    current_date = row["TXN_DATE_TIME"]
    df_window = current_date - df_claim_deviation["TXN_DATE_TIME"]  # get␣
↪series with time_delta
    window_condition = (df_window <= pd.to_timedelta("30d")) & (
        df_window > pd.to_timedelta("0d")
    )  # last 30 days condition where current date is excluded
    df_window_30 = df_claim_deviation[window_condition]
    avg_claim = df_window_30.groupby("INSURANCE_TYPE")["CLAIM_AMOUNT"].mean() ␣
↪# calculating mean wrt insurance type
    # if less than 30 days transaction for particular type then return -1
```

```
    # if less than 30 days transaction criteria is not for particular type then
    # .get(row["INSURANCE_TYPE"], 0) is not required
    if df_window_30.groupby("INSURANCE_TYPE")["TXN_DATE_TIME"].nunique().
 ↪get(row["INSURANCE_TYPE"], 0) < 30:
        return -1
    return 1 if avg_claim[row["INSURANCE_TYPE"]] / row["CLAIM_AMOUNT"] < 0.5␣
 ↪else 0


df_claim_deviation["CLAIM_DEVIATION"] = df_claim_deviation.
 ↪apply(claim_deviations, axis=1)
```

[ ]:
```
df_claim_deviation.sample(10, random_state=26)
```

[ ]:

	TXN_DATE_TIME	INSURANCE_TYPE	CLAIM_AMOUNT	CLAIM_DEVIATION
8884	2021-05-19	Property	39000	-1
4629	2020-12-04	Property	34000	-1
726	2020-06-29	Life	54000	-1
3890	2020-11-05	Motor	5000	0
711	2020-06-28	Life	87000	-1
9144	2021-05-29	Travel	3000	0
2875	2020-09-25	Travel	3000	0
6987	2021-03-04	Life	57000	-1
2348	2020-09-04	Motor	5000	-1
8801	2021-05-16	Travel	2000	0

[ ]:
```
df_claim_deviation["CLAIM_DEVIATION"].value_counts()
```

[ ]:
```
 0    5610
-1    4384
 1       6
Name: CLAIM_DEVIATION, dtype: int64
```

1.8 Task 7

- Find All Agents who have worked on `more than 2 types of Insurance Claims`. Sort them by `Total Claim Amount Approved` under them in `descending order`.

[ ]:
```
df_agents = df_merged[["AGENT_ID", "AGENT_NAME", "INSURANCE_TYPE",␣
 ↪"CLAIM_AMOUNT", "CLAIM_STATUS"]].copy()
df_agents.head()
```

[ ]:

	AGENT_ID	AGENT_NAME	INSURANCE_TYPE	CLAIM_AMOUNT	CLAIM_STATUS
0	AGENT00413	Amy Wangler	Health	9000	A
1	AGENT00413	Amy Wangler	Life	54000	A
2	AGENT00413	Amy Wangler	Life	13000	A
3	AGENT00413	Amy Wangler	Life	42000	D

```
4  AGENT00413  Amy Wangler         Mobile           500            A
```

[ ]:
```
multi_type_condition = (
    df_agents.groupby("AGENT_ID")["INSURANCE_TYPE"].nunique() > 2
)  # More than 2 Insurance Type of Agent
multi_type_agent = multi_type_condition[multi_type_condition].index
multi_type_agent
```

[ ]:
```
Index(['AGENT00001', 'AGENT00003', 'AGENT00004', 'AGENT00005', 'AGENT00006',
       'AGENT00007', 'AGENT00008', 'AGENT00009', 'AGENT00010', 'AGENT00011',
       …
       'AGENT01191', 'AGENT01192', 'AGENT01193', 'AGENT01194', 'AGENT01195',
       'AGENT01196', 'AGENT01197', 'AGENT01198', 'AGENT01199', 'AGENT01200'],
      dtype='object', name='AGENT_ID', length=1164)
```

[ ]:
```
# condition with Approved Claim Status and Multi Insurance Type Agent
multi_approved_condition = (df_agents["AGENT_ID"].isin(multi_type_agent)) &␣
 ↪(df_agents["CLAIM_STATUS"] == "A")
df_agents[multi_approved_condition].groupby(["AGENT_ID", "AGENT_NAME"]).sum().
 ↪sort_values(
    "CLAIM_AMOUNT", ascending=False
)
```

[ ]:
```
                           CLAIM_AMOUNT
AGENT_ID   AGENT_NAME
AGENT00679 Clara Barnett         489000
AGENT00771 Roger Burns           422100
AGENT00807 Don Filkins           396800
AGENT00789 Alison Baron          392900
AGENT00525 Don Ritchie           385900
…                                   …
AGENT00732 Sylvia Tran            11300
AGENT00885 David Montes           11000
AGENT00706 Laura Staggs           10800
AGENT00571 Michele Downs           7500
AGENT00604 Thelma Salinas          5500

[1164 rows x 1 columns]
```

1.9 Task 8

- `Mobile & Travel Insurance` premium are `discounted by 10%`
- `Health and Property Insurance` premium are `increased by 7%`
- `Life and Motor Insurance` premium are marginally `increased by 2%`
- What will be `overall change in % of the Premium Amount Collected` for all these Customer?

[ ]:
```
df_premium = df_insurance[["INSURANCE_TYPE", "PREMIUM_AMOUNT"]].copy()
df_premium["INSURANCE_TYPE"].unique()
```

[ ]:
```
array(['Health', 'Property', 'Travel', 'Life', 'Motor', 'Mobile'],
      dtype=object)
```

[ ]:
```
df_premium_change = df_premium.groupby("INSURANCE_TYPE").sum()
df_premium_change["NEW_PREMIUM_AMT"] = df_premium_change["PREMIUM_AMOUNT"]
df_premium_change
```

[ ]:

INSURANCE_TYPE	PREMIUM_AMOUNT	NEW_PREMIUM_AMT
Health	252455.73	252455.73
Life	125621.22	125621.22
Mobile	15191.42	15191.42
Motor	165391.98	165391.98
Property	202285.83	202285.83
Travel	124139.77	124139.77

[ ]:
```
df_premium_change.loc[["Mobile", "Travel"], "NEW_PREMIUM_AMT"] = (
    df_premium_change.loc[["Mobile", "Travel"], "PREMIUM_AMOUNT"] * 0.9
)  # discounted by 10%
df_premium_change.loc[["Health", "Property"], "NEW_PREMIUM_AMT"] = (
    df_premium_change.loc[["Health", "Property"], "PREMIUM_AMOUNT"] * 1.07
)  # increased by 7%
df_premium_change.loc[["Life", "Motor"], "NEW_PREMIUM_AMT"] = (
    df_premium_change.loc[["Life", "Motor"], "PREMIUM_AMOUNT"] * 1.02
)  # increased by 2%
df_premium_change
```

[ ]:

INSURANCE_TYPE	PREMIUM_AMOUNT	NEW_PREMIUM_AMT
Health	252455.73	270127.6311
Life	125621.22	128133.6444
Mobile	15191.42	13672.2780
Motor	165391.98	168699.8196
Property	202285.83	216445.8381
Travel	124139.77	111725.7930

[ ]:
```
df_premium_change.sum().pct_change()
```

[ ]:
```
PREMIUM_AMOUNT          NaN
NEW_PREMIUM_AMT    0.026799
dtype: float64
```

[ ]:
```
prem_amt, new_prem_amt = df_premium_change.sum()
pct_change = ((new_prem_amt - prem_amt) / prem_amt) * 100
```

```
print(f"Total Premium Amt: {prem_amt:.2f}\nTotal New Premium Amt: {new_prem_amt:
↪.2f}\n% Change: {pct_change:.5f}%")
```

```
Total Premium Amt: 885085.95
Total New Premium Amt: 908805.00
% Change: 2.67986%
```

Overall change in % of the Premium Amount Collected: 2.679%

1.10 Task 9

- Business wants to give discount to customer who are loyal and under stress due to Covid 19. They have laid down an eligibility Criteria as follow
- IF `CUSTOMER_TENURE > 60 AND EMPLOYMENT_STATUS = "N" AND NO_OF_FAMILY_MEMBERS >=4` THEN `1` ELSE `0`
- Create a new column `"ELIGIBLE_FOR_DISCOUNT"` and find it `mean`.

[ ]:
```
df_eligible = df_insurance[["TENURE", "EMPLOYMENT_STATUS",␣
↪"NO_OF_FAMILY_MEMBERS"]].copy()
eligible_condition = (
    (df_eligible["TENURE"] > 60)
    & (df_eligible["EMPLOYMENT_STATUS"] == "N")
    & (df_eligible["NO_OF_FAMILY_MEMBERS"] >= 4)
)
df_eligible["ELIGIBLE_FOR_DISCOUNT"] = eligible_condition.astype(int)
df_eligible["ELIGIBLE_FOR_DISCOUNT"].mean()
```

[ ]:
```
0.0299
```

Mean for discount eligibility customer: 0.0299

1.11 Task 10

- Business wants to check `Claim Velocity` which is defined as follow > CLAIM_VELOCITY = NO_OF_CLAIMS_IN_LAST30DAYS (for the current insurance type) / NO_OF_CLAIMS_IN_LAST3DAYS (for the current insurance type)
- **Note**: LAST30DAYS & LAST3DAYS does not include current day

[ ]:
```
df_insurance.columns.sort_values()
```

[ ]:
```
Index(['ACCT_NUMBER', 'ADDRESS_LINE1', 'ADDRESS_LINE2', 'AGE', 'AGENT_ID',
       'ANY_INJURY', 'AUTHORITY_CONTACTED', 'CITY', 'CLAIM_AMOUNT',
       'CLAIM_STATUS', 'CUSTOMER_EDUCATION_LEVEL', 'CUSTOMER_ID',
       'CUSTOMER_NAME', 'EMPLOYMENT_STATUS', 'HOUSE_TYPE', 'INCIDENT_CITY',
       'INCIDENT_HOUR_OF_THE_DAY', 'INCIDENT_SEVERITY', 'INCIDENT_STATE',
       'INSURANCE_TYPE', 'LOSS_DT', 'MARITAL_STATUS', 'NO_OF_FAMILY_MEMBERS',
       'POLICE_REPORT_AVAILABLE', 'POLICY_EFF_DT', 'POLICY_NUMBER',
       'POSTAL_CODE', 'PREMIUM_AMOUNT', 'REPORT_DT', 'RISK_SEGMENTATION',
       'ROUTING_NUMBER', 'SOCIAL_CLASS', 'SSN', 'STATE', 'TENURE',
```

```
       'TRANSACTION_ID', 'TXN_DATE_TIME', 'VENDOR_ID'],
      dtype='object')
```

[ ]:
```
df_claim_velocity = df_insurance[["TXN_DATE_TIME", "INSURANCE_TYPE",␣
↪"CLAIM_STATUS"]].copy()
df_claim_velocity["TXN_DATE_TIME"] = pd.
↪to_datetime(df_claim_velocity["TXN_DATE_TIME"])
```

[ ]:
```
import numpy as np


def claim_velocities(row):
    current_date = row["TXN_DATE_TIME"]
    df_window = current_date - df_claim_velocity["TXN_DATE_TIME"]  # get series␣
↪with time_delta
    window_30_condition = (df_window <= pd.to_timedelta("30d")) & (
        df_window > pd.to_timedelta("0d")
    )  # last 30 days condition where current date is excluded
    df_window_30 = df_claim_velocity[window_30_condition]
    window_3_condition = (df_window <= pd.to_timedelta("3d")) & (
        df_window > pd.to_timedelta("0d")
    )  # last 3 days condition where current date is excluded
    df_window_3 = df_claim_velocity[window_3_condition]
    # If claim_status = A is required we can pass filter as
    # df_window_3X[df_window_3X['CLAIM_STATUS']=='A'].
↪groupby("INSURANCE_TYPE")["CLAIM_STATUS"].count().
↪get(row["INSURANCE_TYPE"],np.nan)
    claim_count_30 = (
        df_window_30.groupby("INSURANCE_TYPE")["CLAIM_STATUS"].count().
↪get(row["INSURANCE_TYPE"], np.nan)
    )  # calculating no of insurance claim wrt to insurance type
    claim_count_3 = (
        df_window_3.groupby("INSURANCE_TYPE")["CLAIM_STATUS"].count().
↪get(row["INSURANCE_TYPE"], np.nan)
    )  # calculating no of insurance claim wrt to insurance type
    return claim_count_30 / claim_count_3


df_claim_velocity["CLAIM_VELOCITY"] = df_claim_velocity.apply(claim_velocities,␣
↪axis=1)
```

[ ]:
```
df_claim_velocity.sample(10, random_state=14)
```

[ ]:

	TXN_DATE_TIME	INSURANCE_TYPE	CLAIM_STATUS	CLAIM_VELOCITY
5431	2021-01-04	Life	D	11.555556
4771	2020-12-10	Travel	A	7.812500
1385	2020-07-25	Health	A	19.166667

```
8091    2021-04-17          Life            A       14.750000
9202    2021-05-31          Life            A        9.933333
902     2020-07-07      Property            A       14.777778
2152    2020-08-26        Travel            A        9.818182
9459    2021-06-09      Property            A        9.769231
1328    2020-07-23        Travel            A       11.181818
8977    2021-05-22      Property            A        7.333333
```

1.12 Task 11

- Find all low performing agents i.e. employees who are in the bottom 5 percentile based on Claims worked by them.

[ ]:
```
df_performance = df_insurance[["AGENT_ID", "CLAIM_AMOUNT"]].copy()
df_performance.head()
```

[ ]:
```
     AGENT_ID  CLAIM_AMOUNT
0  AGENT00413          9000
1  AGENT00769         26000
2  AGENT00883         13000
3  AGENT00278         16000
4  AGENT00636          3000
```

Based on No. of Claims

[ ]:
```
agent_performance = df_performance.groupby("AGENT_ID").count()
quantile_5 = agent_performance.quantile(0.05).values[0]
res_agent_performance = agent_performance[agent_performance["CLAIM_AMOUNT"] <␣
 ↪quantile_5]
print(f"Total Entries: {res_agent_performance.shape[0]}")
res_agent_performance.rename({"CLAIM_AMOUNT": "NO_OF_CLAIMS"}, axis=1)
```

```
Total Entries: 33
```

[ ]:
```
            NO_OF_CLAIMS
AGENT_ID
AGENT00002             3
AGENT00014             3
AGENT00106             3
AGENT00120             3
AGENT00156             2
AGENT00252             1
AGENT00253             3
AGENT00300             3
AGENT00343             3
AGENT00376             3
AGENT00454             3
AGENT00462             3
```

```
AGENT00469              3
AGENT00536              2
AGENT00603              3
AGENT00621              2
AGENT00628              2
AGENT00645              3
AGENT00656              3
AGENT00773              3
AGENT00790              2
AGENT00804              3
AGENT00863              3
AGENT00887              3
AGENT00926              3
AGENT00934              3
AGENT00958              3
AGENT01037              3
AGENT01077              2
AGENT01129              3
AGENT01154              2
AGENT01161              1
AGENT01191              3
```

Based on Claim Amount

[ ]:
```
agent_performance = df_performance.groupby("AGENT_ID").sum()
quantile_5 = agent_performance.quantile(0.05).values[0]
res_agent_performance = agent_performance[agent_performance["CLAIM_AMOUNT"] <␣
↪quantile_5]
print(f"Total Entries: {res_agent_performance.shape[0]}")
res_agent_performance
```

```
Total Entries: 60
```

[ ]:
```
            CLAIM_AMOUNT
AGENT_ID
AGENT00006         22400
AGENT00014          7400
AGENT00016         27000
AGENT00085         25700
AGENT00088         27700
AGENT00091         30000
AGENT00098         28200
AGENT00134         22000
AGENT00149         28900
AGENT00156         13200
AGENT00172         22200
AGENT00185         16100
AGENT00186         27700
```

AGENT00188	28400
AGENT00197	18500
AGENT00252	20000
AGENT00263	11500
AGENT00275	16200
AGENT00279	28000
AGENT00300	12600
AGENT00310	28800
AGENT00316	22200
AGENT00351	21700
AGENT00425	17800
AGENT00442	30300
AGENT00536	20000
AGENT00562	21100
AGENT00571	24200
AGENT00585	16600
AGENT00590	22200
AGENT00604	5500
AGENT00621	5300
AGENT00628	6100
AGENT00672	21300
AGENT00689	13700
AGENT00706	10800
AGENT00721	9000
AGENT00732	11300
AGENT00773	15000
AGENT00780	11300
AGENT00790	6000
AGENT00804	20000
AGENT00805	22600
AGENT00830	23500
AGENT00841	17800
AGENT00856	24200
AGENT00894	27600
AGENT00910	15500
AGENT00969	28300
AGENT00985	30000
AGENT01021	24300
AGENT01037	5600
AGENT01052	28000
AGENT01077	4200
AGENT01107	24700
AGENT01128	28000
AGENT01131	27500
AGENT01151	28000
AGENT01154	800
AGENT01179	27300

1.13 Task 12

- Business wants to find all Suspicious Employees (Agents).
- IF `TOTAL CLAIM AMOUNT` which meet below criteria is `>= 15000` THEN AGENT is classified as `Suspicious` ELSE `Not`
- `CLAIM_STATUS = Approved` AND `CUSTOMER_RISK_SEGMENTATION = High` AND `INCIDENT_SEVERITY = "Major Loss"`
- If `Suspicious`, THEN `1` ELSE `0`.
- Find `mean` of this column.

[ ]:
```
df_suspicious = df_insurance[
    ["AGENT_ID", "CLAIM_AMOUNT", "CLAIM_STATUS", "RISK_SEGMENTATION",␣
 ↪"INCIDENT_SEVERITY"]
].copy()
df_suspicious.head()
```

[ ]:

	AGENT_ID	CLAIM_AMOUNT	CLAIM_STATUS	RISK_SEGMENTATION	INCIDENT_SEVERITY
0	AGENT00413	9000	A	L	Major Loss
1	AGENT00769	26000	A	L	Total Loss
2	AGENT00883	13000	A	L	Total Loss
3	AGENT00278	16000	A	L	Minor Loss
4	AGENT00636	3000	A	M	Major Loss

[ ]:
```
df_suspicious["CLAIM_STATUS"].unique()
```

[ ]:
```
array(['A', 'D'], dtype=object)
```

[ ]:
```
df_suspicious["RISK_SEGMENTATION"].unique()
```

[ ]:
```
array(['L', 'M', 'H'], dtype=object)
```

[ ]:
```
df_suspicious["INCIDENT_SEVERITY"].unique()
```

[ ]:
```
array(['Major Loss', 'Total Loss', 'Minor Loss'], dtype=object)
```

[ ]:
```
suspicious_condition = (
    (df_suspicious["CLAIM_STATUS"] == "A")
    & (df_suspicious["RISK_SEGMENTATION"] == "H")
    & (df_suspicious["INCIDENT_SEVERITY"] == "Major Loss")
)
df_suspicious_group = df_suspicious[suspicious_condition].groupby("AGENT_ID").
 ↪sum()
suspicious_agent = df_suspicious_group[df_suspicious_group["CLAIM_AMOUNT"] >=␣
 ↪15000].index
suspicious_agent
```

[ ]:
```
Index(['AGENT00001', 'AGENT00003', 'AGENT00011', 'AGENT00021', 'AGENT00043',
       'AGENT00052', 'AGENT00076', 'AGENT00080', 'AGENT00097', 'AGENT00112',
```

```
       …
       'AGENT01130', 'AGENT01142', 'AGENT01168', 'AGENT01175', 'AGENT01177',
       'AGENT01181', 'AGENT01182', 'AGENT01184', 'AGENT01188', 'AGENT01197'],
      dtype='object', name='AGENT_ID', length=149)
```

[ ]:
```
df_suspicious_employee = df_employee.copy()
df_suspicious_employee["SUSPICIOUS"] = 0
df_suspicious_employee.loc[df_suspicious_employee["AGENT_ID"].
 ↪isin(suspicious_agent), "SUSPICIOUS"] = 1
df_suspicious_employee.head()
```

[ ]:

	AGENT_ID	AGENT_NAME	DATE_OF_JOINING	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	POSTAL_CODE	EMP_ROUTING_NUMBER	EMP_ACCT_NUMBER	SUSPICIOUS
0	AGENT00001	Ray Johns	1993-06-05	1402 Maggies Way	NaN	Waterbury Center	VT	5677	34584958	HKUN51252328472585	1
1	AGENT00002	Angelo Borjon	2005-12-27	414 Tanya Pass	NaN	Panama City	FL	32404	107363763	OPIS19290040088204	0
2	AGENT00003	Candy Spellman	2003-09-02	606 National Street	#306	Fayetteville	AR	72701	81744097	YSCJ67489688482590	1
3	AGENT00004	Mary Smith	2004-09-23	235 Hugh Thomas Drive	NaN	Panama City	FL	32404	67563771	ZANG21285355574581	0
4	AGENT00005	Mildred Diaz	2011-06-21	3426 Broadview Street	NaN	Montgomery	AL	36110	114951317	DZFS82244494451134	0

[ ]:
```
df_suspicious_employee["SUSPICIOUS"].value_counts()
```

[ ]:
```
0    1051
1     149
Name: SUSPICIOUS, dtype: int64
```

[ ]:
```
mean_suspicious = df_suspicious_employee["SUSPICIOUS"].mean()
print(f"Mean Suspicious: {mean_suspicious:.05f}")
```

```
Mean Suspicious: 0.12417
```

Mean of Suspicious Employee: 0.12417

2 Fiddle

2.1 Task 6

Static Date
Assuming Current Date as 2021-03-24

[ ]:
```
dfid_claim_deviation = df_insurance[["TXN_DATE_TIME", "INSURANCE_TYPE",␣
 ↪"CLAIM_AMOUNT"]].copy()
dfid_claim_deviation["TXN_DATE_TIME"] = pd.
 ↪to_datetime(dfid_claim_deviation["TXN_DATE_TIME"])
current_date = pd.to_datetime("2021-03-24")
df_window = current_date - dfid_claim_deviation["TXN_DATE_TIME"]
window_condition = (df_window <= pd.to_timedelta("30d")) & (df_window > pd.
 ↪to_timedelta("0d"))
df_window_30 = dfid_claim_deviation[window_condition]
avg_claim = df_window_30.groupby("INSURANCE_TYPE")["CLAIM_AMOUNT"].mean()
avg_claim
```

[ ]:
```
INSURANCE_TYPE
Health      11121.951220
Life        54336.283186
Mobile        427.102804
Motor        6090.909091
Property    24534.351145
Travel       3115.384615
Name: CLAIM_AMOUNT, dtype: float64
```

[ ]:
```
def claim_deviations(row):
    if current_date - row["TXN_DATE_TIME"] < pd.to_timedelta("30d"):
        return -1
    return 1 if avg_claim[row["INSURANCE_TYPE"]] / row["CLAIM_AMOUNT"] < 0.5␣
 ↪else 0


dfid_claim_deviation["CLAIM_DEVIATION"] = dfid_claim_deviation.
 ↪apply(claim_deviations, axis=1)
dfid_claim_deviation["CLAIM_DEVIATION"].value_counts()
```

[ ]:
```
 0    6778
-1    3222
Name: CLAIM_DEVIATION, dtype: int64
```

[ ]:
```

```

[ ]:
```
import numpy as np

df = pd.DataFrame(
```

```
    {
        "dates": pd.date_range(start="01-10-2022", periods=31, freq="d"),
        "count": range(31),
        "group": np.random.randint(0, 3, 31),
    }
)
# df.set_index('dates',inplace=True)
df.head()
```

[ ]:

	dates	count	group
0	2022-01-10	0	2
1	2022-01-11	1	2
2	2022-01-12	2	0
3	2022-01-13	3	2
4	2022-01-14	4	0

[ ]:
```
df.rolling(window="3d", min_periods=3, closed="left", on="dates").sum().head()
```

[ ]:

	dates	count	group
0	2022-01-10	NaN	NaN
1	2022-01-11	NaN	NaN
2	2022-01-12	NaN	NaN
3	2022-01-13	3.0	4.0
4	2022-01-14	6.0	4.0

[ ]:
```
df[df["group"] == 0].rolling("5d", 1, closed="left", on="dates").sum()
```

[ ]:

	dates	count	group
2	2022-01-12	NaN	NaN
4	2022-01-14	2.0	0.0
7	2022-01-17	6.0	0.0
8	2022-01-18	11.0	0.0
9	2022-01-19	19.0	0.0
14	2022-01-24	9.0	0.0
15	2022-01-25	14.0	0.0
21	2022-01-31	NaN	NaN
22	2022-02-01	21.0	0.0
25	2022-02-04	43.0	0.0
27	2022-02-06	47.0	0.0
28	2022-02-07	52.0	0.0

[ ]:
```
df.groupby("group").rolling(window="3d", closed="left", on="dates").mean()
```

[ ]:

		dates	count
group			
0	2	2022-01-12	NaN
	4	2022-01-14	2.0

```
        7  2022-01-17    4.0
        8  2022-01-18    7.0
        9  2022-01-19    7.5
        14 2022-01-24    NaN
        15 2022-01-25   14.0
        21 2022-01-31    NaN
        22 2022-02-01   21.0
        25 2022-02-04   22.0
        27 2022-02-06   25.0
        28 2022-02-07   26.0
1       11 2022-01-21    NaN
        12 2022-01-22   11.0
        13 2022-01-23   11.5
        16 2022-01-26   13.0
        19 2022-01-29   16.0
        20 2022-01-30   19.0
        23 2022-02-02   20.0
        24 2022-02-03   23.0
2       0  2022-01-10    NaN
        1  2022-01-11    0.0
        3  2022-01-13    0.5
        5  2022-01-15    3.0
        6  2022-01-16    4.0
        10 2022-01-20    NaN
        17 2022-01-27    NaN
        18 2022-01-28   17.0
        26 2022-02-05    NaN
        29 2022-02-08   26.0
        30 2022-02-09   29.0
```

[ ]:
```
df_claim_deviation_2 = df_insurance[["TXN_DATE_TIME", "INSURANCE_TYPE",␣
↪"CLAIM_AMOUNT"]].copy()
df_claim_deviation_2["TXN_DATE_TIME"] = pd.
↪to_datetime(df_claim_deviation_2["TXN_DATE_TIME"])
df_claim_deviation_2.describe(datetime_is_numeric=True)
```

[ ]:
```
                      TXN_DATE_TIME   CLAIM_AMOUNT
count                         10000   10000.000000
mean   2020-12-16 10:45:41.760000   16563.830000
min           2020-06-01 00:00:00     100.000000
25%           2020-09-10 00:00:00    2000.000000
50%           2020-12-18 00:00:00    7000.000000
75%           2021-03-24 00:00:00   21000.000000
max           2021-06-30 00:00:00  100000.000000
std                           NaN   22037.489735
```

[ ]:
```
df_claim_deviation_2
```

[ ]:

	TXN_DATE_TIME	INSURANCE_TYPE	CLAIM_AMOUNT
0	2020-06-01	Health	9000
1	2020-06-01	Property	26000
2	2020-06-01	Property	13000
3	2020-06-01	Health	16000
4	2020-06-01	Travel	3000
…	…	…	…
9995	2021-06-30	Motor	1000
9996	2021-06-30	Life	67000
9997	2021-06-30	Health	8000
9998	2021-06-30	Motor	2000
9999	2021-06-30	Mobile	300

[10000 rows x 3 columns]

[ ]:
```
df_claim_deviation_2.set_index("TXN_DATE_TIME").
 ↪groupby("INSURANCE_TYPE")["CLAIM_AMOUNT"].rolling(
    "30D", closed="left"
).mean().groupby(["INSURANCE_TYPE", "TXN_DATE_TIME"]).last()
```

[ ]:
```
INSURANCE_TYPE  TXN_DATE_TIME
Health          2020-06-01       13142.857143
                2020-06-03       11700.000000
                2020-06-04       12230.769231
                2020-06-05       11466.666667
                2020-06-06       11523.809524
                                        …
Travel          2021-06-26        2742.187500
                2021-06-27        2752.000000
                2021-06-28        2757.812500
                2021-06-29        2782.258065
                2021-06-30        2771.653543
Name: CLAIM_AMOUNT, Length: 2329, dtype: float64
```