Name: Chicheng Zhang

1. a, b, d, h

2. $Var(X) = E(X^2) - E(X)^2,\ Var(Y) = E(Y^2) - E(Y)^2,\ Cov(X,Y) = E(XY) - E(X)E(Y)$
   $E(X) = 0, E(Y) = 0, E(XY) = 1 \times \Pr(XY = 1) + (-1) \times \Pr(XY = -1)$

$$\begin{aligned} Corr(X,Y) &= \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}} = \frac{Cov(X,Y)}{sqrt(Var(X)Var(Y))} \\ &= \frac{Cov(X,Y)}{sqrt((E(X^2) - E(X)^2)(E(Y^2) - E(Y)^2))} \\ &= \frac{E(XY) - E(X)E(Y)}{sqrt((E(X^2) - E(X)^2)(E(Y^2) - E(Y)^2))} \\ &= \frac{\Pr(XY = 1) - \Pr(XY = -1)}{sqrt((E(X^2) - E(X)^2)(E(Y^2) - E(Y)^2))} \end{aligned}$$

3. $Sample\ Pearson's\ correlation\ coefficient = r_0 = \frac{\sum_{i=1}^{5}[(X_i - \bar{X})(Y_i - \bar{Y})]}{\sqrt{\sum_{i=1}^{5}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{5}(Y_i - \bar{Y})^2}} = -0.8686$

t-test for testing $H_0: \rho = 0, H_1: \rho \neq 0$

$$Test\ statistic: t^* = \frac{r\sqrt{n-2}}{\sqrt{1 - r^2}} \sim t_{n-2}$$

$$t_0 = \frac{-0.8686\sqrt{5-2}}{\sqrt{1 - (-0.8686)^2}} = -3.0361$$

To answer the question, we need to find the probability that a data set statistic value is as extreme or more extreme than -3.0361.

$$P - value = Pr(|r^*| \geq |-0.869||H_0) = \Pr(|t^*| \geq |-3.042||H_0) = 2\Pr(t^* \leq -3.042|H_0)$$

Compare the statistic value to a t-distribution with 3 degrees of freedom,

$$\Pr(t^* \leq -3.042|H_0) = 0.028$$
$$P - value = 0.028 \times 2 = 0.056$$

Thus, under the null hypothesis that X and Y are uncorrelated, the probability of observing a data set at least as correlated (negatively or positively) as the above data set is 0.056.

Whether we should reject the null hypothesis depends on the significant level we choose. If the significant level $\alpha \geq 0.056$, reject the null hypothesis. If the significant level $\alpha < 0.056$, do not reject the null hypothesis.

4.

Approach #1: Logistic Regression

Pros:

(1) Suit for cases where the outcome is binary variable

(2) Easy to implement and calculate, efficient.

(3) Robust to small noise, can use L1 or L2 regularization to avoid overfitting issue.

Cons:

(1) May have underfitting issue, inaccurate in classification.

(2) May suffer multicollinearity.

(3) May not work for too many features or independent variables.

Approach #2: Random Forest

Pros:

(1) Efficient on large database.

(2) Can gain feature importance after training.

(3) Tolerate to missing data and unbalanced data set.

Cons:

(1) May be more difficult to interpret the results.

(2) May have overfitting issue in case of noisy data.

Approach #3: Naïve Bayes

Pros:

(1) High efficient in computation.

(2) Easy to explain and implement.

(3) Perform well for categorical input data.

(4) not likely to be affected by noise and missing data.

Cons:

(1) Relies on independence assumption. Not suit for real-life data.

(2) Very sensitive to input data type. If the input data is numerical, normal distribution is strongly assumed.

5. I picked 500 players for clustering. I divided them into 5 clusters because there are usually 5 different positions in a basketball team.

   The script may have connection error even though I've adopted the user-agent header. Please be patient if there shows a connection error.

   ```
   ConnectionError: ('Connection aborted.', OSError("(10060, 'WSAE
   TIMEDOUT')",))
   ```

   The run time is around 250 seconds. I think the script meets all the requirements as you ask. The output is in the same format as the example.

   If you have any other question, please let me know.

6.

Player
• Player ID (PK) • First Name • Last Name • Roster Status • Date of Birth • Height • Weight • Nationality • City, State, Country of birth • From Year • To Year • Player Code • Team ID (FK=Team.Team ID) • Team Name • Current Position • Game Play FLAG • College • Salary Index(First Name, Last Name, Team Name, Team ID)

Team
• Team ID (PK) • Team Name • Team Abbreviation • Team Code • Team City • Region • Number of Players • Coach • Champion Season • Team Logo Index(Team Name, Team Abbreviation, Team City)

Game
• Game ID (PK) • Game Type • Date • Season • Start Time • End Time • Host Team ID (FK=Team.Team ID) • Host Team Score • Guest Team ID (FK=Team.Team ID) • Guest Team Score Index(Date, Host Team, Guest Team)

Player Stats

- Game ID (FK=Game.Game ID)
- Player ID (FK=Player.Player ID)
- GP
- GS
- MIN
- FGM
- FGA
- FG_PCT
- FG3M
- FG3A
- FG3_PCT
- FTM
- FTA
- FT_PCT
- OREB
- DREB
- REB
- AST
- STL
- BLK
- TOV
- PF
- PTS
- TS_percent

Primary Key (Game ID, Player ID)
Index(Game ID, Player ID)

Team Stats

- Game ID (FK=Game.Game ID)
- Team ID (FK=Team.Team ID)
- GP
- GS
- MIN
- FGM
- FGA
- FG_PCT
- FG3M
- FG3A
- FG3_PCT
- FTM
- FTA
- FT_PCT
- OREB
- DREB
- REB
- AST
- STL
- BLK
- TOV
- PF
- PTS

Primary Key (Game ID, Team ID)
Index(Game ID, Team ID)

7.

I choose **c** and **d**.

Index enables speeding up the searching for a matching field within the records, which will use binary search instead of full table scan. If the join column doesn't have index, we can create one on it.

SQL engine goes through three steps for every query: Parse, Optimize, Execute. The query optimizer first performs straightforward optimizations. Then consider different query plans by estimating the cost of CPU and time. Then it picks the optimal plan and passes it to the next step. Sometimes the query is complex, so the optimizer can't choose the optimal query plan.

Strategy:

For this query, what I can think about is as follows.

Firstly, if we know one column will be queried very frequently by $where$ and $order\ by$, we can create index. The database engine will be able to use that index to efficiently find the matching rows.

Secondly, we can manually set the searching order by reordering the tables in the constrains. For example, constrain $t.A = g.A$ will let database engine search g table first, then search t table. we should choose the table with fewer return counts as driving table.

Thirdly, when using outer join, we can put constrains from $where$ into on because the database engine process on before $where$.